Thomas R. Blakeslee

Proiectarea cu circuite logice MSI și LSI standard

Seria
Electronică
Aplicată

BIBLIOTECA DE AUTOMATICĂ, INFORMATICĂ, ELECTRONICĂ. MANAGEMENT

Au apărut în cadrul colecției „Electronica aplicată" :

B. Boconcios, I. Diaconescu — *Voltmetre electronice.*
B. Popescu — *Frecvențmetre electronice.*
B. Andreescu — *Generatoare de semnal.*
C. Negoiță, M. Ivan — *Aparate electronice pentru măsurarea mărimilor geometrice.*
V. Malcoci — *Aparate electronice pentru măsurarea parametrilor rezistențelor, bobinelor și condensatoarelor.*
St. Boboc — *Aparate electronice pentru măsurarea maselor, forțelor și cuplurilor.*
I. Mateescu, I. Mateescu — *Osciloscopul.*
A. Goncearov — *Înregistrarea magnetică a imaginii.*
Th. Nicolau, I. Apostol — *Umidimetre electronice.*
H. Sinnreich, A. Vasilescu — *Transmisiuni cu modulația impulsurilor în cod.*
Gh. Boldea — *Localizarea deranjamentelor din cablurile de telecomunicații.*
A. Barna — *Amplificatoare operaționale.*
I. Feier, I. Dragu, V. Vulpe — *Dioda Zenner. Aplicații.*
T. Wilmore — *Electronica fizică. Întrebări și răspunsuri.*
B. Ibbotson — *Telecomunicații. Întrebări și răspunsuri.*
B. Barker — *Electronica aplicată. Întrebări și răspunsuri.*
P. Constantin, R. Ovidiu — *Tranzistoare unijoncțiune.*
A. Manea, M. Scărlătescu — *Aparate electronice pentru protecția muncii.*
E. Sofron, Șt. Târâcă — *Dispozitive optoelectronice cu cristale lichide.*
Al. Popescu, A. Nica — *Aparate electronice pentru măsurări industriale.*
G. Antonescu — *Dispozitive semiconductoare pentru microunde.*
R. M.M. Obermann — *Numărătoare electronice.*
I. Marghescu, Gg. Bădescu — *Transmiterea discretă a semnalelor.*
G. Băjeu, Gh. Stancu — *Generatoare de semnale sinusoidale.*
I. Mitican — *Radiotelefoane.*
I. Dragu, I. Iosif — *Dispozitive videoreproductoare și videocaptoare.*
O. Radu, Gh. Săndulescu — *Filtre numerice.*
I. Bîrcă Gălățeanu — *Amplificatoare de bandă largă.*
E. Damachi — *Dispozitive semiconductoare multijoncțiune.*
M. Sâmpăleanu — *Circuite pentru conversia datelor.*
G. Mitrofan — *Generatoare de impulsuri și de tensiune liniar variabilă.*
E. Spindler — *Antene.*
Gh. Săndulescu — *Protecția la perturbații în electronica industrială și automatizări.*
I. Sztojanov ș.a. — *De la poarta TTL la microprocesor vol. 1 și 2.*

THOMAS R. BLAKESLEE

Proiectarea cu circuite logice MSI şi LSI standard

Traducere din limba engleză după ediția a II-a

Postfață : **M. Bodea**
Gh. Ştefan

EDITURA TEHNICA
Bucureşti, 1988

Thomas R. Blakeslee
Vice President — Engineering
Logisticon Inc., Sunnyvale California

Digital Design with Standard MSI and LSI
Design tehniques for the microcomputer age.
(Second edition)

Traducere : **M. Bodea** *cap. 1, 2, 14, 15*
Gh. Ştefan *cap. 3 . . . 13*

Redactor : ing. **Smaranda Dimitriu**
Tehnoredactor : **Olimpiada Nistor**
Coperta : **Simona Dumitrescu**
Execuţia desenelor : **Viorica Iftimie**

Bun de tipar : 17.10.1988.
Coli de tipar : 23,25
C. Z. : 621.381

I. P. „Oltenia" Craiova
Str. Mihai Viteazul, nr. 4
Republica Socialistă România

PREFAȚĂ

Revoluția determinată de circuitele integrate a făcut ca multe din obiectivele tradiționale ale proiectării digitale să devină învechite. Obiectivul principal al proiectării nu îl mai constituie reducerea la minimum a numărului de porți și circuite basculante, deoarece un singur circuit integrat, ieftin, poate să conțină acum sute de componente.

În prima ediție a acestei cărți am încercat să prezint o modalitate unitară de proiectare, care să reflecte aceste noi realități. Dezvoltările din ultimii ani au confirmat valabilitatea strategiilor pe care le-am prezentat. În ediția de față am adus la zi și am extins cuprinsul cărții astfel încît să includă cîteva din dezvoltările remarcabile din domeniul microcalculatoarelor.

Oricît de important ar fi microcalculatorul, el constituie numai unul din multele mijloace pe care trebuie să le înțeleagă și să le stăpînească cel ce lucrează în proiectarea digitală. În această carte am încercat să adun o colecție echilibrată de mijloace de proiectare care să-și păstreze utilitatea pe măsură ce revoluția determinată de circuitele integrate continuă și se dezvoltă.

Deoarece detaliile specifice ale microcalculatoarelor vor continua să se schimbe cu rapiditate încă mulți ani, m-am concentrat, pe cît de mult a fost posibil, asupra conceptelor de bază și a tehnicilor de proiectare. *Multe din cărțile de astăzi consacrate microcalculatoarelor încearcă să treacă în revistă* detaliile *funcționării unui mare număr de microprocesoare particulare. Deoarece aceste detalii tind să devină depășite odată la doi ani, ele sînt disponibile — mult mai economic — de la fabricantul de circuite integrate. Ca urmare aș recomanda ca acestei cărți să i se adauge în atenția cititorului cel puțin un manual al utilizatorului, adus la zi, al unui microcalculator sau un catalog de circuite integrate.*

Această carte pune accentul pe proiectarea de sistem, deoarece blocurile funcționale LSI și MSI sînt de fapt componente de sistem, care trebuie combinate convenabil cu alte componente mecanice și electrice pentru a se obține un rezultat global optim. Deoarece costul logicii devine o parte neglijabilă din costul total al sistemului, este din ce în ce mai important să se atace problema sistemului ca o entitate. Cu toate că această carte se concentrează asupra utilizării circuitelor LSI și MSI standard, *cele mai multe dintre principii rămîn valabile și atunci cînd volumul producției justifică utilizarea unor circuite LSI realizate la cererea clientului*.*

*În esență, tehnica de proiectare se axează pe utilizarea „componentelor ieftine"**, standardizate, pentru a satisface majoritatea cerințelor sistemului și apoi, pentru a compleia ce mai este nevoie, pe utilizarea circuitelor integrate SSI. Principalul rezultat urmărit constă în* aducerea la minimum a numărului de capsulede circuite integrate *și nu a numărului de circuite basculante și porți în conformitate cu abordarea tradițională. În carte sînt incluse și multe considerații practice, care în mod obișnuit nu apar în literatură, deoarece necunoașterea acestor probleme determină efecte mult mai dezastruoase decît deteriorare cîtorva porți.*

---

* *custom circuits*, în literatura de limbă engleză (n.t.).

** *bargain components* (n.t.).

Am încercat să fac această carte utilă, atît profesioniștilor din domeniu cît și studenților, prin strîngerea într-un glosar — și nu prin explicarea în text — a conceptelor de bază, a terminologiei și a abrevierilor curent utilizate. În acest fel nici profesioniștii și nici studenții avansați nu sînt plictisiți cu lucruri pe care deja le știu, în timp ce începătorii pot înțelege termenii, pe măsură ce apar, căutîndu-i în glosar.

Primele două capitole stabilesc și ilustrează prin exemple ideile de proiectare utilizate ulterior în tot restul cărții. Cu toate că în capitolele 3...5 accentul este pus pe utilizarea — acolo unde este posibil — a circuitelor MSI și LSI, sînt prezentate și tehnicile de bază ale proiectării logice tradiționale deoarece ele sînt necesare atît pentru „a umple interstițiile" dintre circuitele MSI și LSI cît și pentru subsistemele foarte mici. Se prezintă o metodă intuitivă de atribuire a stărilor care permite aducerea la minimum a întregii scheme logice, fără a ascunde funcționarea sistemului prin abstracții matematice.

Capitolele 7... 10 acoperă conceptul de programare, limbajele de programare, sistemele de dezvoltare pentru microcalculatoare și tehnicile de proiectare pentru hardware-ul de microcalculator.

În capitolul 11 se discută metodele de utilizare multiplă a circuitelor pentru prelucrarea serie a biților sau pentru funcționarea cu divizarea timpului, utilizînd distribuirea în timp a procesării. Deoarece viteza circuitelor crește și circuitele LSI sînt limitate de interconexiuni, această metodă devine o tehnică din ce în ce mai importantă.

Capitolele 6 și 12 iau în considerare realitățile neplăcute ale lumii reale cum ar fi, de exemplu, efectele de propagare, blocarea în stări necontrolabile, zgomotul, reflexiile și diafonia. Capitolul 13 prezintă cîteva tehnici, ca reacția negativă și funcționarea incrementală, care sînt utilizate pentru a face mai simple dispozitivele moderne de intrare/ieșire. Se descriu, în calitate de exemplu, cîteva periferice de calculator, de real succes. Capitolul 14 discută, dintr-un punct de vedere strict practic, utilizarea statisticii în fiabilitate și în calculele de supraîncărcare a memoriilor tampon. În final, capitolul 15, discută consecințele sociale ale ingineriei și posibilitățile de utilizare a fantasticelor resurse oferite de circuitele integrate pentru a îmbunătăți calitatea vieții.

Thomas R. Blakeslee

Woodside, California
Februarie 1979

CUPRINS

1

IDEEA
Să adaptăm ceea ce avem de făcut la componentele ieftine

Atunci cînd luăm în considerare gama incredibilă a problemelor, aparent nelegate între ele, care pot fi soluționate de **circuitele integrate digitale*** (CI), se pare că într-adevăr logica digitală este pe cale să controleze întreaga lume. Cum cele mai multe dintre aceste probleme nu sînt în esența lor digitale sau cel puțin electrice, este natural să ne punem întrebarea de ce rezolvarea lor se face atît de eficient cu logica digitală ? Explorînd răspunsurile la această întrebare, putem obține o înțelegere a motivelor reale care determină economia extraordinară pe care o realizează abordarea digitală învățînd astfel și modalitatea în care putem să-i exploatăm integral potențialul.

1.1. STANDARDIZAREA

Standardizarea constituie cheia fantasticei bogății materiale pe care o avem în prezent. Prin realizarea pe o linie de fabricație a unui număr mare de produse identice, sau aproape identice, este posibilă obținerea unei economii însemnate. Un volum mare al producției justifică investiții importante în echipamente tehnologice speciale și punerea la punct a unei linii de fabricație eficiente. Prăpastia dintre lucrurile ieftine, disponibile în cantități mari de produse standardizate, și costul acelor lucruri care nu pot fi produse în serii mari s-a adîncit și lărgit în mod constant din momentul în care Henry Ford a pornit fabricația de serie. Astăzi se poate cumpăra o mașină de scris completă, cu toate părțile sale atît de încîlcite, pentru mai puțin decît ar costa făcutul unei pîrghii de la o clapă într-un atelier mecanic. Să ne gîndim care ar fi costul a sute de componente complicate ale unui ceas de $ 10, făcut manual, de comandă, de către un ceasornicar !

Procesul folosit pentru fabricația de CI este probabil exemplul fundamental pentru acest tip de fabricație eficientă. Mii de replici ale aceluiași circuit sînt produse pe o singură **plachetă** din siliciu cu diametrul de numai 3 țoli (vezi figura 1-1). În acest fel fiecare etapă a procesului de fabricație se realizează simultan pentru mii de circuite. Aceasta înseamnă că CI pot fi pro-

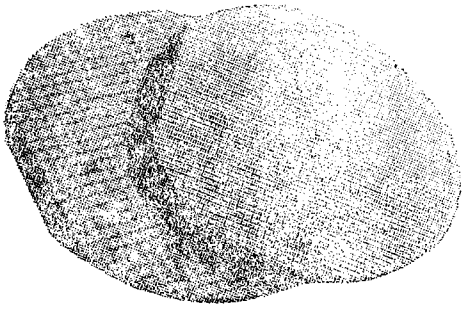


Fig. 1-1. Plachetă cu CI.

* **Cuvintele tipărite cu litere aldine arată că noțiunea respectivă din glosar apare prima dată în text.**

duse *foarte* economic, *însă numai cu condiția de a putea utiliza o mare cantitate de circuite identice.* Cu excepția încapsulării finale, efortul cerut pentru realizarea unui singur circuit este același ca și efortul cerut de realizarea unei mii ! Desigur că prelucrarea plachetelor se face în modul cel mai eficient într-o linie de fabricație destinată pentru o producție în cantități mari. Dacă fiecare plachetă produsă conține 1 000 de circuite, atunci producerea a numai 100 de plachete pe zi ne furnizează 100 000 de circuite integrate ! Devine evident că, pentru a recolta integral beneficiul oferit de această tehnică de producție, va fi necesar să utilizăm un anumit tip de CI în cantități imense. Dacă fabricăm un radio portabil, un televizor sau un calculator de buzunar această problemă se rezolvă de la sine : vom proiecta un circuit specializat care va fi produs cu necesitate în cantități mari, deoarece și numărul produselor pe care le fabricăm este mare.

1.2. CIRCUITE DIGITALE STANDARD

Majoritatea sistemelor digitale sînt produse, din păcate, numai în cantități moderate. Singura modalitate prin care putem într-adevăr culege roadele oferite de producția în cantități mari a CI este aceea în care în toată industria, într-un mare număr de *diferite* sisteme digitale, se utilizează aceleași circuite integrate de bază. Această cerință constituie în prezent o realitate, ca urmare a însușirilor cu totul particulare pe care le are logica **binară.**

Sistemul de numerație binar este cel mai simplu posibil deoarece numerele pot avea doar una din cele două valori 0 și 1. Din această cauză numărul de combinații posibile este foarte restrîns. De exemplu, tabla înmulțirii — pentru care am pierdut ani ca să o învățăm pe de rost — în sistemul zecimal (10 valori) de numerație, este banală în sistemul binar*. Întregul sistem al algebrei binare (Boole) se poate descrie, cu demonstrații, în cîteva pagini (vezi Ref. 1 și 2). Factorul care a făcut posibilă standardizarea CI digitale constă în simplitatea acestei algebre. Este efectiv posibil ca *orice* sistem logic, inclusiv un mare calculator, să se realizeze în totalitate utilizînd numai un singur tip de **poartă** și cu un singur tip de **circuit basculant bistabil**** (așa cum, de altfel, se întîmpla în perioada de început a CI). Prin construirea unui întreg calculator numai din porți NAND și bistabili *J-K* se poate consuma o cantitate imensă din aceste circuite chiar de către o singură companie care produce, de exemplu, numai 100 de calculatoare.

Această împingere la limită a standardizării este posibilă numai pentru logica digitală. Deși pot fi produse foarte economic, și cantități mari de elemente mecanice, așa cum o dovedește de exemplu un ceas deșteptător de $5, gradul de standardizare realizat de CI digitale nu poate fi atins deoarece pentru componentele mecanice sînt posibile prea multe versiuni. Roțile dințate de la ceas, de exemplu, pot avea orice număr de dinți și pot fi de o infinitate de dimensiuni. Desigur că există cataloage de roți dințate standard, dar au pagini și pagini de tabele în funcție de mărimea și numărul de dinți ai roților. Cu o astfel de selecție largă este imposibil să se producă roți dințate standard pentru uz general în cantități foarte mari.

---

* Întreaga tablă a înmulțirii este următoarea : $1 \times 1 = 1$, $1 \times 0 = 0$ și $0 \times 0 = 0$. Adunarea este la fel de simplă $1 + 1 = 0$ (ținem 1), $1 + 0 = 0$ și $0 + 0 = 0$.

** În realitate un calculator *se poate* realiza și cu un singur tip de circuit basculant bistabil completat cu porți pe intrări, această soluție fiind însă neeconomică.

În cazul CI digitale standardizarea nu pune probleme. Echivalentul logic al unui angrenaj utilizat pentru reducerea turației îl constituie un lanț de bistabili identici, standard, fiecare micșorînd viteza cu un factor de 2. Acești bistabili sînt identici cu aceia utilizați într-un calculator, într-o unitate de bandă sau în orice alt dispozitiv logic. Din acest motiv este efectiv convenabilă realizarea cu componente digitale standard — a unui singur ceas electronic (așa cum, de altfel, mulți amatori o și fac), în timp ce pentru realizarea unui singur ceas mecanic ne trebuie pentru părțile mecanice o sumă de sute de dolari.

Această discuție ne oferă și explicația controlului pe care îl exercită logica digitală asupra întregii lumi : *componente ieftine standardizate.* Avantajele oferite de standardizare sînt mari chiar dacă lucrul care trebuie realizat nu se potrivește exact cu componentele. Să luăm ca exemplu ceasul. Deoarece în final vrem să obținem o reprezentare mecanică a timpului pare logic ca în primul rînd să facem ceasul mecanic. Cu toate că bistabilii standard ne convin ei nu produc la ieșiri altceva decît semnale electrice 1 și 0, pe care apoi trebuie să le decodăm și să le convertim în ceva pe care să îl putem vedea. Deși aceste operații înseamnă încă multă bătaie de cap în plus ceea ce se obține este mult mai puțin scump.

Economia imensă pe care o facem utilizînd componentele standardizate realizează mai mult decît o compensare a ineficienței care rezultă din neadaptarea componentelor la aplicație. Această situație nu constituie un caz excepțional. În realitate, prăpastia dintre eficiența per dolar a CI și a oricăror alte componente face ca această afirmație să fie practic întotdeauna adevărată. Desigur că dacă putem adapta într-un fel oarecare lucrul pe care îl avem de făcut la aceste componente rezultatul va fi mult mai economic.

Acum, cunoscînd motivele pentru care logica digitală a pătruns în atît de multe probleme nedigitale, putem să generalizăm și să formulăm un principiu general de proiectare ; *„Să adaptăm ceea ce avem de făcut la componentele ieftine"*. Tot ceea ce se cere pentru ca această tehnică să meargă este ca ineficiența adaptării la componentele ieftine să fie mai mică decît avantajele oferite de utilizarea lor. În competiția circuite integrate-componente mecanice primele sînt mult înainte ; un raport de cost de 1 : 100 nu este neuzual, chiar dacă este vorba de cantități mici. Aceasta înseamnă că folosind CI vom fi în avantaj cu condiția ca eficiența noastră să fie mai mare de 1% ! Deoarece decalajul tehnologic între componentele mecanice și CI crește în permanență, această tehnică de proiectare, eficientă în prezent, se va dovedi în viitor și mai eficientă.

Această tehnică nu este limitată de fel la nivelul **proiectării de sistem.** Există variații de cost de 100 : 1 între diferitele CI cu care se poate rezolva aceiași problemă (de exemplu între bistabilele discrete și memoriile cu acces aleator sau între **porți** și **memoriile** de tip **ROM**). Același principiu, de a încerca să folosești componente ieftine se aplică și la **proiectarea logică.**

Raportul de 100 : 1 între costul componentelor mecanice și costul CI care realizează funcții similare poate părea cam greu de acceptat. Revoluția CI s-a petrecut atît de rapid încît atît aprecierea impactului său real cît și arta de a fructifica integral posibilitățile CI au rămas în urmă.

Dacă luăm în considerare rata de dezvoltare, între 1959 și 1977, a tehnologiei de circuite integrate, devine ușor de înțeles de ce CI au depășit atît de rapid toate celelalte tehnologii. În timp ce tehnologiile de fabricație a com-

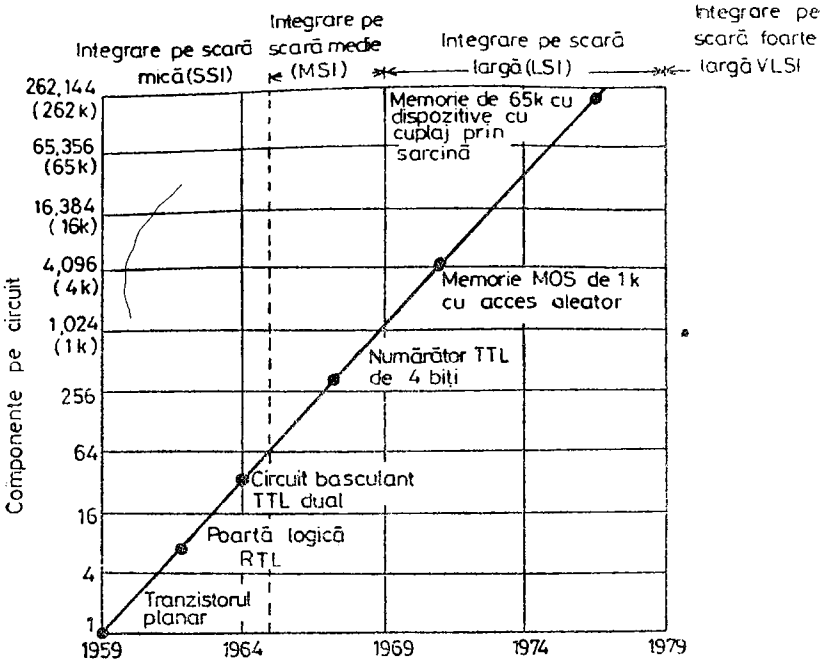


Fig. 1-2. Numărul de componente pe cip în CI modern (după Noyce, 1977).

ponentelor mecanice au evoluat numai gradual tehnologia de CI a avut o creștere explozivă, exponențială. După cum se poate vedea în figura 1-2, numărul de componente care pot fi înghesuite pe un cip de CI s-a dublat în fiecare an, pe o perioadă de 18 ani, din 1959 pînă în 1977. Mai important este faptul că această tendință de evoluție pare să se mențină pentru încă mulți ani de acum înainte. Făcînd o corecție care ține seama de inflație vom găsi că pentru un cip de CI de nivelul actual, costul a rămas constant din 1959. Rezultatul net este acela că *în 20 de ani costul per componentă în CI a scăzut cu un factor de un milion!*

1.3. CÎTEVA EXEMPLE

Deoarece relația între componenta ieftină și elementul pe care îl înlocuiește este adeseori foarte abstractă, să ilustrăm în continuare principiul cu cîteva exemple.

CALCULATORUL DE BIROU. Calculatorul de birou are o intrare mecanică (claviatura) și o ieșire mecanică (numere vizibile). Mulți ani calculatorul de birou a fost o colecție incredibilă de roți dințate, pîrghii și arcuri. Prin 1960 a început realizarea calculatoarelor de birou cu CI digitale. Deși mai era necesară o conversie de la semnalele codificate mecanic la cele codificate electric și înapoi, pentru afișarea cu **tuburi nixie,** ceea ce s-a obținut era mai bun ca înainte din orice punct de vedere.

Cu toate că versiunea cu **circuite integrate pe scară mică (SSI)*** a reprezentat o adaptare la componenta ieftină la nivelul proiectării de sistem, evoluția a continuat la nivelul proiectării logice prin introducerea unor circuite integrate pe scară largă. În 1971 a apărut versiunea finală : un singur CI care face totul.

TELEFONIA CU MODULAȚIA IMPULSURILOR ÎN COD. În acest sistem sînt transmise printr-o singură linie telefonică 24 de semnale vocale. Semnalele sînt eșantionate și convertite în coduri digitale care sînt apoi transmise serie, pe linie. La celălalt capăt are loc o conversie digital-analog obținîndu-se cele 24 de semnale **analogice.** În ciuda tuturor acestor conversii analog-digital și digital-analog sistemul digital este de fapt mult mai economic decît sistemul analogic echivalent. Într-un sens se poate spune că înlocuim prin logica digitală componentele mecanice (cele 23 de linii eliminate). Acest sistem este descris mult mai în detaliu în Secțiunea 13.11.

PANOU INDICATOR. Un panou indicator de 1 200 de becuri aranjate într-o matrice de 40 × 30 indică starea a 1 200 de puncte dintr-un sistem de rachete. Deși se pot utiliza 1 200 de becuri, aprinse de 1 200 de circuite speciale comandate de 1 200 de bistabili, aceiași funcție este perfect realizată și de o combinație de două componente ieftine : un monitor TV și un **registru de deplasare** MOS de 1 200 de **biți.** Pe măsură ce fascicolul tubului catodic baleiază ecranul monitorului TV, registrul de deplasare MOS produce un semnal care aprinde spotul dacă s-a stocat „1" pentru punctul corespunzător al poziției spotului. Pe ecranul *TV* se obține în acest fel o matrice de 40 × 30 zone luminoase sau întunecate.

După cum au arătat exemplele precedente, relația dintre componenta ieftină și aceea pe care o înlocuiește nu este întotdeauna evidentă. De pildă, în cel de al doilea exemplu, 23 de linii telefonice sînt înlocuite de cîteva circuite logice. Cu toate că este cu siguranță imposibil să se tragă o concluzie generală relativ la numărul de CI echivalente unei linii telefonice, se poate spune că logica realizată cu CI este atît de economică încît dacă ea va fi utilizată pentru înlocuirea unei „componente" scumpe, ca linia telefonică, ea va duce probabil la economie de bani. Cu alte cuvinte în general, CI sînt „componente ieftine" iar liniile telefonice *nu* sînt.

1.4. ALTE AVANTAJE

Pînă în acest punct discuția s-a purtat în special în ceea ce privește costul ; standardizarea are însă și alte avantaje care sînt adeseori mult mai importante. O componentă standard se fabrică în cantități mari, se utilizează de către mulți oameni și este produsă de mai multe firme. Ca urmare *erorile de proiectare, problemele de siguranță în funcționare și dificultățile de fabricație sînt rezolvate* de către utilizatorii inițiali ai componentei. Cu toate că pentru orice proiect propriu, aceste probleme trebuie să le rezolvăm noi înșine ele sînt, practic, gata eliminate, dacă lucrăm cu componente standardizate cunoscute. Prin utilizarea componentelor standard ciclul de proiectare se scurtează mult deoarece componentele se pot introduce în proiectare imediat, pe baza informațiilor din foaia de catalog. Întrucît componentele se află de obicei în stoc la distribuitori, de multe ori le putem avea în mînă înainte

* Inițialele SSI vin de la cuvintele din limba engleză *small scale integration* — integrare pe scară mică (n.t.).

ca proiectul să fie terminat, dacă am avut grijă să le comandăm în fazele sale inițiale. Componentele fiind stocate de mulți distribuitori este puțin probabil ca producția să se oprească din cauza lipsei lor. În sfîrșit activitatea de service este mult simplificată deoarece elementele standard sînt stocate de distribuitori din toate orașele din lume. Elementele care trebuie înlocuite pot fi cumpărate pur și simplu la fața locului de către tehnicianul care face service-ul.

În timp ce toate aceste avantaje se regăsesc pentru sute de CI „componente ieftine", foarte puține componente mecanice au acest grad de standardizare. Ca urmare aproape toate componentele mecanice pot fi caracterizate ca nefiind — în mod sigur — „componente ieftine". În stadiul inițial de proiectare al unui sistem, vom încerca deci să reducem la minimum componentele mecanice — chiar dacă acest lucru se soldează cu o mare creștere a complexității electronice.

Deși aproape toate CI sînt ieftine (în comparație cu celelalte tipuri de componente) unele din ele se detașează fiind ieftine în mod efectiv. *Ca urmare vom putea aplica regula „de a adapta ceea ce avem de făcut la componentele ieftine"* și la nivelul proiectării logice. Ideea constă în a încerca să utilizăm **circuite** ieftine **integrate pe scară largă (LSI)** sau **pe scară medie (MSI)*** pentru cea mai mare parte a *proiectului logic care urmează a fi realizat,* iar apoi să utilizăm circuitele SSI ca un liant necesar care „ține totul la un loc". Deoarece componentele ieftine LSI oferă posibilități atît de mari raportate la cost, ele pot fi utilizate chiar ineficient, realizînd totuși încă o economie netă.

Ca și în cazurile anterioare pentru a ne potrivi la componentele ieftine, trebuie să facem adeseori multe adaptări. O mare parte a acestei cărți se ocupă cu prezentarea modului în care se realizează acest lucru.

Ca regulă generală cu cît un CI este mai complex, cu atît este mai apropiat de „componenta ieftină". Această afirmație devine evidentă dacă ne reamintim procesul de fabricație de mare serie a CI (vezi fig. 1-1). Cea mai mare parte a costului unui CI este dată de încapsularea finală și de diferitele etape de testare care trebuie realizate individual, cip cu cip. Deoarece pe o plachetă de 4 țoli se fabrică simultan un număr de CI de ordinul unei mii, costul unei structuri de CI pe plachetă este aproape neglijabil. Deși un circuit LSI complex poate avea aria cipului mai mare, *chiar și în cazul celor mai mari cipuri dintr-o plachetă tot se mai obțin cîteva sute de CI.* Costul pe care îl implică producerea unor circuite LSI și MSI nu este deci mult mai mare decît acela pe care îl implică producerea unui circuit SSI. La fel de interesant este și faptul că rata de defectare a circuitelor LSI este aproape aceeași cu a circuitelor SSI, deoarece cele mai multe defectări sînt date de încapsulare sau de interconexiuni.

Prețul de vînzare al circuitelor LSI este în mod curent mult mai mare decît acela al circuitelor SSI. Unele din motivele care duc la această situație sînt temporare. În circuitele LSI se utilizează deseori procese noi, a căror cost de dezvoltare trebuie recuperat ; în plus, procesul fiind nou randamentul de fabricație este mai mic deoarece mult mai multe circuite sînt defecte. Există însă și un alt motiv, fundamental, care explică prețul mai ridicat : cu cît o componentă este mai complexă cu atît ea este mai particulară, astfel că *standardizarea este dificilă* și este imposibil să se atingă producții cu adevărat mari. De exemplu sînt ușor de standardizat piulițele și șuruburile, dar nu tot atît de ușor pistoanele.

---

* Inițialele LSI și MSI vin de la cuvintele din limba engleză *large scale integration* și *medium scale integration* — integrare pe scară largă, respectiv medie (n.t.).

Această problemă a standardizării — pentru realizarea unei producții în cantități mari — este aceea care face ca ideea utilizării produselor ieftine LSI și MSI să fie atît de importantă. Unele produse prind și devin „standarde industriale". Circuitele sînt fabricate întotdeauna de mai mulți fabricanți astfel încît concurența menține un preț foarte scăzut. Prețul mic face ca din ce în ce mai mulți proiectanți să utilizeze în sistemele lor aceste componente, făcînd ca volumul producției să crească în continuare. Pe măsură ce aceste componente sînt din ce în ce mai apropiate de „componenta ieftină", adaptarea lor în sisteme este din ce în ce mai dorită, chiar dacă ele nu se potrivesc pe deplin, astfel că volumul producției lor continuă să crească, iar prețul să scadă.

Putem vedea deci că principiul de „a adapta ceea ce avem de făcut la componentele ieftine" acționează ca o forță naturală care tinde să facă o „standardizare de facto" a unor componente, fără nici un fel de coordonare centralizată. Această situație este foarte sănătoasă din cauză că, pentru nenumăratele CI noi care se anunță, procesul favorabil descris anterior are loc în mod obișnuit numai pentru cel mai bun dintre tipuri — restul tinzînd să rămînă scumpe și să dispară încet, încet. Componenta standard, odată acceptată stă în frunte pînă cînd apare ceva semnificativ mai bun. Două exemple excelente care ilustrează această standardizare de facto sînt date de seria 7400 de circuite **TTL** Texas Instruments și **memoriile cu acces aleator (RAM)** de 16 k Mostek 4116.

1.5. SĂ ALEGEM COMPONENTELE DIN PUNCTUL DE VEDERE AL COSTULUI PE CARE ÎL VOR AVEA ÎN VIITOR

În domeniul CI lucrurile se schimbă atît de rapid încît de multe ori diverse proiecte devin învechite după 2 ani. Deoarece punerea efectivă pe picioare a fabricației unui produs care folosește CI ia adeseori unul sau doi ani, un proiectant de sisteme logice trebuie să învețe să anticipeze viitorul. În acest fel costul *total* al elementelor componente se poate aduce la minimum pentru întreaga viață a produsului și nu numai pentru perioada în care s-a realizat proiectarea.

În figura 1-3 este prezentată evoluția prețului pentru două familii de circuite logice. În 1969 puteam fi tentați să alegem, din economie, circuitele **RTL,** deși circuitele TTL ar fi funcționat mult mai bine. Este adevărat că nu trebuie să plătești pentru performanțe de care nu ai nevoie, dar măcar o cunoaștere restrînsă a ciclului normal de viață al componentelor și a ceea ce se află în interiorul unui CI l-ar fi făcut pe un proiectant cu scaun la cap să aleagă TTL. După cum rezultă din figura 1-3 alegerea familiei RTL ar fi fost o greșeală din toate punctele de vedere. În plus, în figură nu apare faptul că niciunul din circuitele MSI și LSI disponibile astăzi nu lucrează direct cu RTL; rezultă că ar fi fost necesară și o comutare spre TTL pentru a putea beneficia la produsele noi de avantajele oferite de MSI. Aceasta ar duce în companie la un dublu standard, generator de necazuri, care face necesară stocarea pentru fabricație sau pentru piese de schimb în service atît a componentelor RTL, cît și a celor TTL, atît timp cît mai există produsele realizate cu RTL

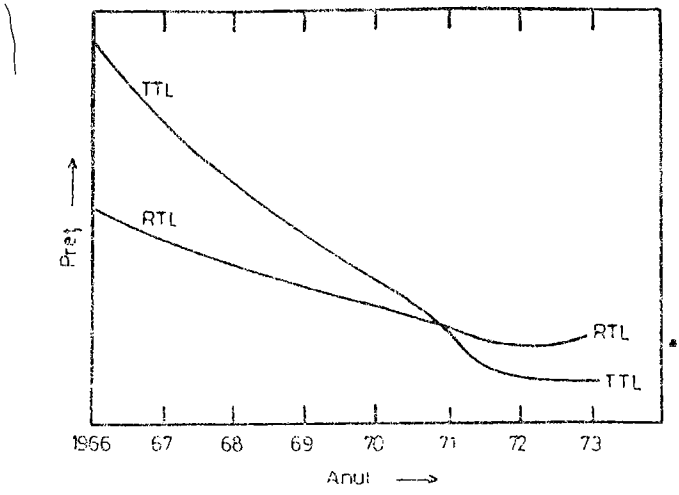


Fig. 1-3. Prețurile porților RTL și TTL, 1966—1973.

Dacă analizăm ciclul normal de viață al unui CI (vezi fig. 1-4) vom vedea că în momentul lansării produsului prețul este mare deoarece, evident, cantitatea fabricată este zero. Prețul din acest punct are o slabă legătură cu costul necesar pentru producerea componentei fiind mai mult o chestiune de strategie. Dacă performanțele componentei sînt mai bune decît acelea ale oricăror altor componente disponibile, nu este neobișnuit să se ceară un preț *foarte* mare pînă în momentul în care apare concurența ; întotdeauna există proiectanți care au nevoie de performanțe mai bune și ca urmare vor plăti pentru ele aproape orice preț. Cumpărarea, în acest moment, a unor cantități mari poate să coboare prețul pînă la o cincime din prețul de catalog.

Dacă circuitul nu are succes, nu se dezvoltă o fabricație în cantități mari, nu apare concurența și în cele din urmă moare. Un circuit de succes este produs repede și de alți fabricanți* și intră într-o fază în care concurența

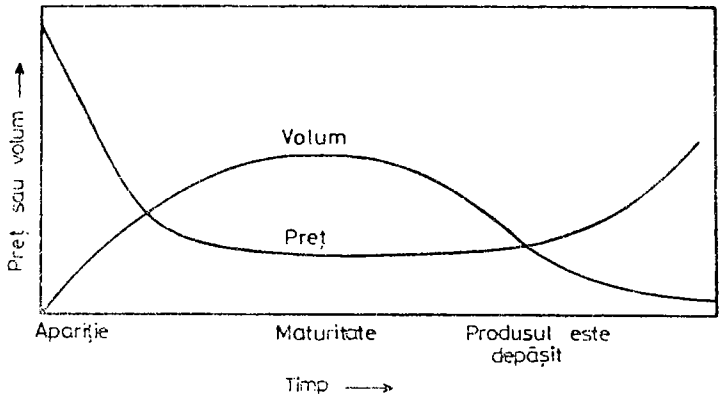


Fig. 1-4. Ciclul vieții unui CI.

* „is *second sourced*", în limba engleză (n.t.).

forțează scăderea prețului pînă la un nivel care corespunde costului real de fabricație. Pe măsură ce cantitatea fabricată crește, prețul scade. Scăderea prețului se datorează numai parțial creșterii cantității fabricate; ea se datorează și „curbei de învățare".

Curba de învățare (vezi fig. 1-5) este dată pur și simplu de scăderea costului de fabricație pe măsură ce se acumulează mai multă experiență (reflectată de numărul de componente fabricate și vîndute). Există cîte o curbă de învățare pentru fiecare circuit și una pentru procesul de fabricație în general. Un proces de fabricație (de exemplu bipolar, MOS canal *p* sau MOS canal *n*) are și el un ciclu de viață asemănător cu acela prezentat în figura 1-4. Viteza de scădere a prețului unui circuit oarecare este legată de gradul de dezvoltare a procesului utilizat pentru fabricarea lui. Viteza de scădere a prețului este întotdeauna mai mare pentru o componentă nouă, în comparație cu una veche, mult mai matură.

După aproximativ doi ani de creștere constantă a cantității fabricate (vezi fig. 1-4) și de scădere constantă a prețului se atinge un punct (maturitatea) în care apare un alt produs, în mod clar superior. Începînd din acest punct, prețul încetează să mai scadă, dar volumul producției continuă să crească încă o perioadă de cîțiva ani. Creșterea volumului are loc în principal din cauza livrărilor, care continuă, ale produselor în care se utilizează circuitul. Necesitatea de a scădea în continuare prețul nu mai există deoarece nimeni nu va mai utiliza în nici un fel componenta în proiecte noi, iar cei care au folosit-o în proiecte îi rămîn credincioși. În acest punct componenta a ajuns la apogeu. În cele din urmă, componenta devine învechită; volumul producției începe să scadă, prețurile încep să crească încet, cîte puțin, odată cu scăderea volumului și în final fabricantul o scoate de pe piață.

Cazul deciziei TTL sau RTL, în 1969, devine acum complet clar. În acea perioadă RTL aproape că ajunsese la maturitate, în timp ce TTL tocmai demara. Deoarece mărimea cipului pentru TTL era mai mică decît pentru RTL, iar procesul era virtual același, costul de fabricație al TTL era potențial mai mic. Mai mult, în 1968 se știa bine că circuitele MSI erau în pregătire și că vor fi sigur compatibile cu noua familie logică, atunci cînd va apare.

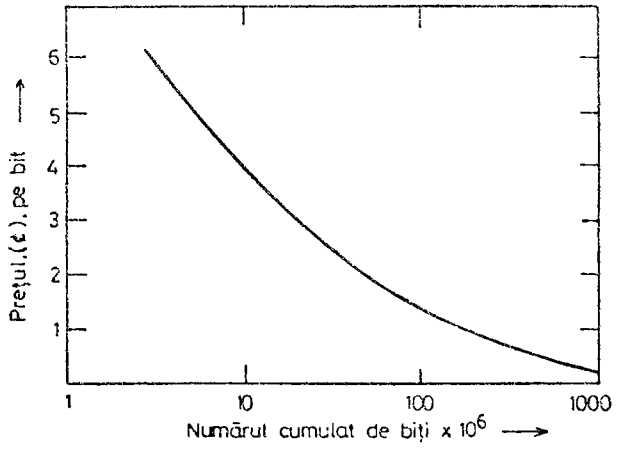


Fig. 1-5. Curbă de învățare tipică pentru prețul unui registru de deplasare lung.

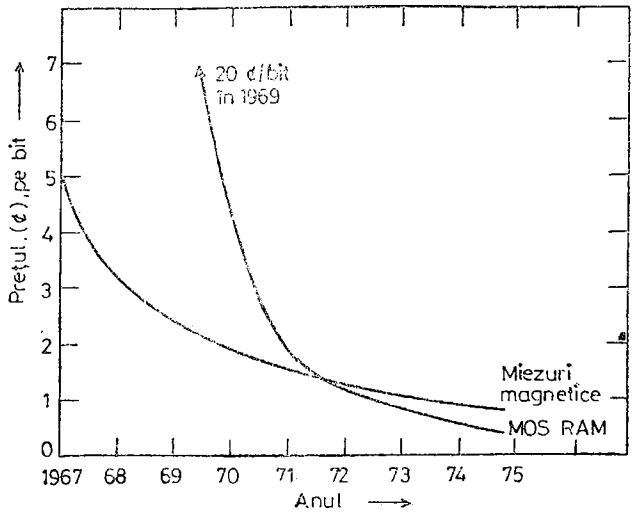


Fig. 1-6. Costul memoriilor MOS și a celor cu miezuri magnetice (în cantități mari).

Prin urmare sistemul în chestiune ar fi fost mult mai ieftin, *pentru întreaga sa viață*, dacă ar fi fost proiectat cu TTL — chiar dacă în perioada proiectării TTL era mai scump!

Figura 1-6 prezintă un exemplu similar. Aici întrecerea este între a introduce în proiecte memorii cu miezuri din ferită sau memorii MOS. Dacă decizia era de luat în 1971, ar fi putut să pară că alegerea memoriilor cu miezuri din ferită constituie soluția cea mai economică. Cu toate acestea, deoarece memoriile au miezuri din ferită erau un produs „matur" iar memoriile MOS erau chiar la început, era clar că cea mai bună alegere ar constitui-o memoriile MOS. În acest moment este util să privim din nou în „interiorul" componentei. O memorie cu miezuri este efectiv complicată și implică multe elemente și tehnologii. Pe de altă parte, memoriile semiconductoare sînt fabricate printr-un proces care este de fapt mai simplu decît unul utilizat pentru a fabrica circuite integrate TTL (care se vindeau în acel timp cu 15...20 cenți bucata) singura diferență consta în aceea că procesul MOS era într-un stadiu de început al curbei de învățare iar memoriile cu miezuri de ferită erau prin 1971 în mod clar un produs matur. De fapt cea mai mare parte a scăderii costului per bit, între 1967 și 1970 era datorată în realitate circuitelor integrate; circuitele de scriere și **amplificatoarele de citire** au redus costul memoriilor cu miezuri din ferită la un nivel cu puțin mai mare decît chiar costul planelor de memorie.

1.6. CUM GĂSIM COMPONENTELE IEFTINE

Proiectanții de produse care se fabrică în cantități mari ca, de exemplu, receptoarele *TV*, calculatoarele de buzunar și memoriile pentru calculatoare pot să specifice componentele care se potrivesc exact cu cerințele produsului. Totuși cea mai mare parte a muncii de proiectare actuale este destinată pentru sisteme care se fabrică doar în cantități moderate. Proiectantul acestor

tipuri de sisteme trebuie să învețe să găsească componentele ieftine existente și să le adapteze la nevoile sale. *Una din cele mai bune surse de componente ieftine autentice o constituie componentele proiectate de fapt pentru produsele care se fabrică în cantități mari.*

De exemplu, un calculator de buzunar conține un circuit LSI foarte puternic și un afișaj numeric care în multe cazuri pot fi adaptate la alte aplicații. Simulînd, de exemplu, comenzile de la claviatură cu o altă logică, putem să realizăm o părticică de logică și memorie și să afișăm rezultatul, plătind pentru tot ce am realizat un preț foarte mic. Claviatura unui calculator este și ea o componentă ieftină, deoarece este fabricată în cantități foarte mari. Dacă putem adapta proiectul nostru, astfel încît să putem utiliza exact aceeași dispoziție a clapelor, ca aceea a calculatorului, putem adeseori să cumpărăm o claviatură la jumătate din prețul pe care l-ar avea o claviatură proiectată la comandă. În acest fel utilizăm de fapt, „gratuit", produsele fabricate, în cantități mari, pentru aplicația pentru care au fost proiectate. În afara economiei de bani, cîștigăm siguranța în funcționare și disponibilitatea specifice unei componente standard produsă în cantități mari.

REFERINȚE

1. M. Phister, *Logical Design of Digital Computers*, Wiley, New-York, 1958, cap. 3.
2. George Boole, *An Investigation of the Law of Thought*, Dover, New-York, 1954.

BIBLIOGRAFIE

Fabricarea circuitelor integrate

Carr, W. N. și Mize, J. P., *MOS/LSI Design and Application*, McGraw-Hill, New York, 1972, Capitolele 1, 2 și 10.

Hibbard, Robert, *Integrated Circuits — A Basic Course for Engineers and Technicians*, McGraw-Hill, New York, 1969.

Dezvoltarea tehnologiei circuitelor integrate

Graham, Robert,, F., „Semiconductor Memories: Evolution or Revolution", *Datamation*, June 1969.

Martino, J., „Tools for Looking Ahead", *IEEEE Spectrum*, October 1972, pp. 32—40.

Moore, Gordon E., „What Level of LSI is Best for You? " *Electronics*, February 16, 1970, pp. 126—129.

Mudge, J., și K. Taft, „V-ATE Memory Scores a New High in Combining Speed and Bit Density", *Electronics*, July 17, 1972, pp. 65—69.

Noyce, Robert, „Microelectronics", *Scientific American*, September 1977, pp. 62—69.

Special Issue: „The Great Takeover" (of IC Technology), *Electronics*, October, 25, 1973, pp. 68—191.

Telefonia cu modulația impulsurilor în cod

Davis, C. G., „An Experimental Time Multiplex Terminal", *Bell System Technical Journal*, January 1962.

2

OBIECTIVELE
proiectării unui sistem digital

Una din cele mai surprinzătoare probleme pe care ne-o putem pune în legătură cu analiza costurilor unui sistem digital este aceea a ponderii pe care o are costul total al circuitelor integrate în costul total al unui sistem real. Prețul CI a scăzut atît de mult încît *ponderea lor în costul total al unui sistem este mai mică de 10% !* Desigur că acest procent variază în anumite limite în funcție de volumul producției sistemului respectiv, fiind pentru sistemele unicat chiar mult mai mic. Totuși, în mod cert tendința de evoluție a costului CI este aceea de a deveni *o parte neglijabilă* a costului întregului sistem. Aceeași tendință de evoluție se constată și în rapoartele privind siguranța în funcționare : CI constituie o sursă din ce în ce mai neglijabilă în defectarea sistemului.

Motivul care determină această tendință de evoluție este simplu. Deși CI și-au îmbunătățit costul și fiabilitatea cu o rată fantastică, aspectele legate de problemele mecanice și de asamblare ale sistemului s-au perfecționat în mod neînsemnat. În domeniul surselor de alimentare, al răcirii și al asamblării n-a existat o revoluție ci numai o evoluție relativ înceată. Situația este efectiv asemănătoare cu aceea a unui avion supersonic. În momentul de față putem zbura de la Londra la New-York în aproximativ același timp în care ajungi la aeroport, parchezi, te înregistrezi și aștepți pe pămînt bagajele. Motivul este același : dezvoltarea spectaculoasă a avionului fără îmbunătățiri comparabile la nivelul solului.

Asemănarea între dezvoltarea CI și aceea a avionului se referă însă numai la aspectul calitativ. Rata cu care s-au dezvoltat CI este fără seamăn : în perioada de 18 ani dintre 1959 și 1977 costul raportat la funcție s-a îmbunătățit cu un factor de 200 000. Dacă și aviația s-ar fi dezvoltat tot atît de repede costul unei călătorii New-York—Londra ar fi fost o fracțiune de cent, iar dacă viteza unui avion ar fi ținut pasul cu ritmul de îmbunătățire al vitezei CI un zbor New-York—Londra ar fi ajuns să dureze mai puțin de un minut !

Dacă ținem cont de această rată de dezvoltare fără precedent devine ușor de înțeles de ce tehnologia CI a depășit deocamdată capacitatea noastră de a le utiliza. De altfel, deja ne găsim în situația ridicolă în care costul CI dintr-un sistem a devenit o parte neglijabilă a costului total. Mai mult chiar, tendința de evoluție caracterizată de *înjumătățirea în fiecare an a costului per funcție* pare să continue foarte ferm.

Pentru proiectantul de sisteme logice moderne, această idee a costului neglijabil a CI este extrem de importantă deoarece *face ca obiectivele tradiționale ale proiectării logice să nu mai fie valabile.* În abordarea tradițională, de pînă acum, proiectarea logică constituia de fapt o problemă de simplificare a logicii prin matematică, pentru a reduce la minimum numărul de circuite basculante și de porți. În condițiile în care contribuția dată de costul și fiabi-

fiabilitatea circuitelor integrate este neglijabilă sîntem însă siliţi să fixăm obiective de proiectare noi, care să aibă la bază aducerea la minimum a surselor reale de cost şi de defectare din sistemele logice moderne.

Înainte de a trece la analiza costului unui sistem tipic vom descrie sistemul de asamblare modular, sistem pe care se va baza această analiză.

2.1. ASAMBLAREA MODULARĂ

Sistemul dominant, încă din timpurile logicii cu tranzistoare discrete, pentru asamblarea sistemelor logice mari are o structură ca aceea prezentată în figura 2-1. Acest sistem este încă şi astăzi cel mai economic cu toate că

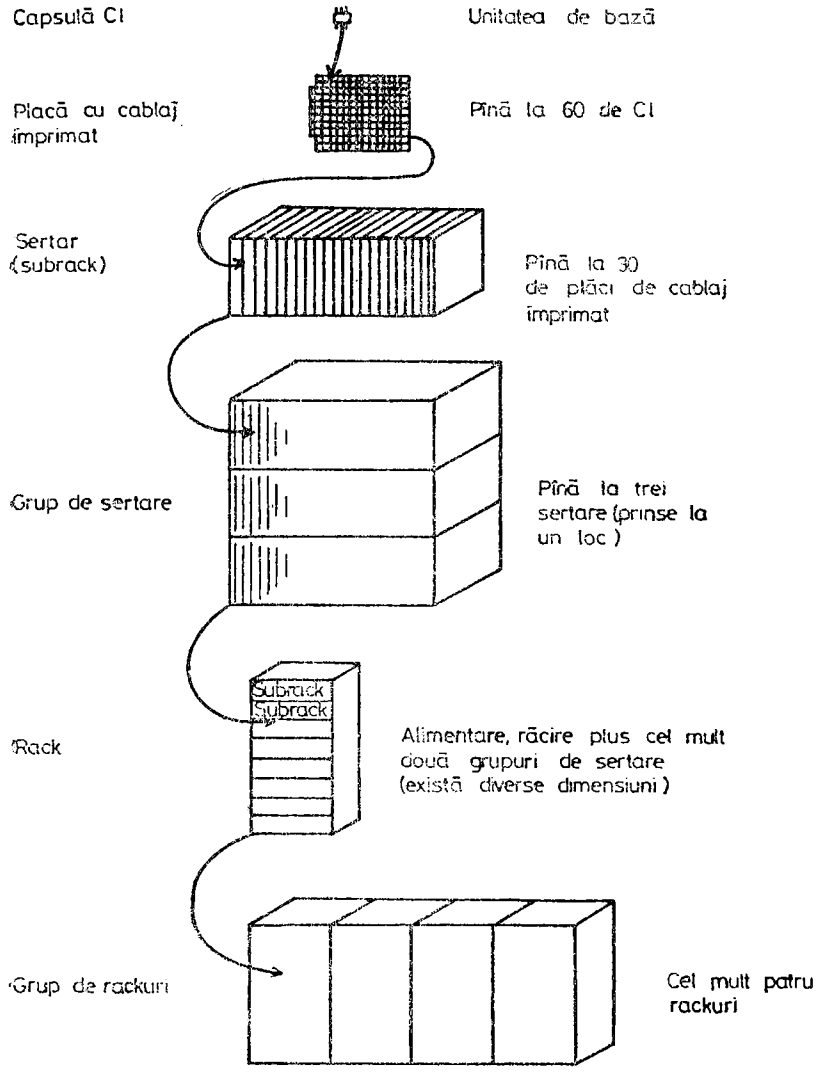


Fig. 2.1. Asamblarea modulară.

circuitele LSI fac ca pe o **placă de cablaj imprimat** să încapă din ce în ce mai multe sisteme. Motivul superiorității acestui sistem este **standardizarea**, orice sistem, de orice mărime, putînd fi realizat din aceleași părți mecanice de bază.

Unitatea de bază o constituie placa de cablaj imprimat care are pe o margine un conector. Pe aceste plăci de cablaj imprimat se află CI și celelalte componente, interconectate cu trasee din cupru definite prin procedee fotografice. Cu toate că traseele de cupru sînt specifice pentru fiecare placă mărimea și forma plăcii, modul de montare al conectorului etc. sînt standardizate. Această standardizare face posibilă realizarea unei economii însemnate la fabricarea și testarea plăcii. Placa este de fapt un subsistem de mărime optimă din punctul de vedere al manipulării și al testării. De cele mai multe ori repararea unui sistem defect se poate face prin înlocuirea uneia din plăci. În prezent tot mai multe sisteme se realizează pe o singură placă de cablaj imprimat.

Realizarea unui sistem mai mare implică parcurgerea ierarhiei din figura 2-1, pînă la nivelul cerut de mărimea sistemului.

Nivelul următor plăcii de cablaj imprimat este dat de **sertar***. Sertarul este în esență o cutie din tablă prevăzută cu conectoare și ghidaje mecanice pentru mai multe plăci de cablaj imprimat. În fiecare sertar se pot plasa pînă la 32 de plăci de cablaj imprimat. Interconectarea terminalelor de la conectoarele sertarului se face în mod obișnuit prin **wire wrap**, cu excepția masei și a alimentării de curent continuu, care se realizează prin intermediul unui plan de masă și al unui plan de alimentare ale „plăcii mamă" de cablaj imprimat. Uneori interconexiunile pentru semnale sînt realizate în totalitate sau parțial printr-o placă mamă de cablaj imprimat multistrat.

Pentru sistemele și mai mari, două pînă la patru sertare se pot prinde între ele cu șuruburi, avînd în comun un plan de masă al unei plăci mamă mai mari, fapt care permite manipularea și interconectarea lor prin wire wrap ca unitate independentă.

Nivelul de asamblare care urmează este **rack-ul**, care în mod obișnuit este rack-ul standard de 19 țoli existent încă dinaintea construirii primului calculator electronic. În mod obișnuit în rack se includ ventilatoarele de răcire, sursele de alimentare și panourile de protecție. Un rack mare se compune din trei sau patru sertare.

Pentru sisteme și mai mari rack-urile se prind între ele cu șuruburi, în grupuri de cîte patru sau mai mult.

Deși în practică acest mod de asamblare cunoaște multe versiuni, ele diferă numai în ceea ce privește detaliile ; în continuare el ne va servi ca model în analiza modului în care se cheltuie banii într-un sistem digital.

2.2. PE CE SE DUC TOȚI BANII ?

Socotirea costurilor într-un sistem digital este foarte asemănătoare cu aceea pe care o faci pentru modul în care se cheltuie salariul. Dacă, pur și simplu, ții seama numai de cheltuielile majore, vizibile, ajungi la concluzia că lunar poți economisi jumătate din salariu ! Singura cale prin care se obține un rezultat corect este aceea în care se ține seama de fiecare bănuț ; un exemplu

* Denumirile din literatura de limbă engleză sînt *subrack*, *card file*, *card gage* (n.t.)

relativ la modul în care se repartizează costurile — valabil pentru perioada de la sfîrşitul anului 1977 — este dat în Tabelul 2-1. În acest tabel s-au indicat numai costurile proporţionale cu mărimea sistemului. Costurile prezentate sînt medii, corespunzînd unui volum de producţie moderat — al unui calculator de mărime medie — şi se referă la perioada indicată. Valorile reale se schimbă în timp, dar principiul general de a te socoti pînă la ultimul bănuţ rămîne universal valabil.

2.3. CONCLUZIA: SĂ REDUCEM LA MINIMUM NUMĂRUL DE CAPSULE DE CIRCUITE INTEGRATE

Tabelul 2-1 pune în evidenţă faptul foarte important că, din punctul de vedere al reducerii la minimum al costului sistemului, nici unul din factorii listaţi nu este dominant, ci dimpotrivă *toate costurile sînt aproape proporţionale*

Tabelul 2-1. **Costurile dintr-un sistem digital care sînt proporţionale cu numărul de capsule de circuite integrate**

	Costul pe bucată	Numărul de circuite integrate pe bucată[a]	Costul pe circuit integrat[b]
Placa de cablaj imprimat	$ 20/placă	50/placă	$ 0,40
Conectorul plăcii de cablaj imprimat	$ 5/placă	50/placă	0,10
Sertarul (cutia metalică, ghidajele, manopera)	$ 200/sertar	1000/sertar	0,20
Placa de fund (planele de masă şi de alimentare ale sertarului)	$ 100/sertar	1000/sertar	0,10
Wire wrap (automat)	6 cent/conexiune	1/conexiune	0,06
Sursele de alimentare ($ 1/W, utilizate 70%)	$ 1,50/W	10/W	0,15
Rack-ul (inclusiv uşile, ventilatoarele, reţeaua de alimentare)	$ 500/rack	4000/rack	0,13
Comanda, primirea şi inventarierea circuitelor integrate	2 cent/circuit integrat	1	0,02
Testarea circuitelor integrate	8 cent/circuit integrat	1	0,08
Testarea plăcii de cablaj imprimat	$ 5/placă	50/placă	0,10
Verificarea sistemului	$ 800/rack	4000/rack	0,20
Capacitoarele de decuplare (inclusiv manopera de inserţie)	$ 1/placă	50/placă	0,02
Inserţia şi lipirea circuitelor integrate	6 cent/circuit integrat	1	0,06
Cablurile de interconexiune	$ 80/rack	4000/rack	0,02
Panoul frontal	$ 300/rack	4000/rack	0,07
Asamblarea sistemului	$ 200/rack	4000/rack	0,05
Ambalarea şi expedierea sistemului	$ 200/rack	4000/rack	0,05
Proiectarea şi elaborarea sistemului, realizarea prototipului (pentru 100 de sisteme)	$ 2000/rack	4000/rack	0,50
Costul de service pe 5 ani	$ 4000/rack	4000/rack	1,00
Costul suplimentar total raportat la numărul de circuite integrate			=$ 3,31

a S-a presupus un număr mediu (şi nu maxim) de circuite integrate pe placă, sertar etc. deoarece nu este posibil ca o placă de cablaj imprimat să fie complet ocupată.

b Costul pe circuit integrat = (costul pe bucată)/(numărul de circuite integrate pe bucată).

cu numărul de capsule de CI din sistem. Cu alte cuvinte reducerea la jumătate a numărului de capsule de CI dintr-un sistem înseamnă, aproximativ, reducerea la jumătate a numărului de plăci cu cablaj imprimat, a numărului de sertare, a necesarului de putere, a efortului de proiectare, a costului de service etc. Ca urmare rezultă că *adevărata cale de reducere la minimum a costului sistemului este de a reduce la minimum numărul de capsule de circuite integrate.* Din fericire, această reducere rămîne un obiectiv realist și în cazul utilizării circuitelor LSI și MSI. În Capitolele 3 și 4 vom vedea că de multe ori este posibil să înlocuim mai multe circuite SSI printr-un singur circuit MSI sau LSI, deși acesta este mult mai scump.

Valoarea totală a tuturor acestor costuri raportată la numărul de circuite integrate este dată la sfîrșitul Tabelului 2-1. Surprinzător poate apare faptul că această valoare este egală cu de cîteva ori costul unui circuit integrat. Valabilitatea acestui rezultat se poate ușor verifica plecînd, în sens invers, de la cifrele de costuri totale:

$$\text{costul suplimentar raportat la numărul de CI} = \frac{\text{costul total al sistemului logic} - \text{costul total al CI}}{\text{numărul de CI din sistem}}$$

Costul toal al sistemului logic va include costul de proiectare amortizat, costul de service și orice alte costuri care sînt proporționale cu mărimea sistemului. El nu va conține costul echipamentelor periferice, care nu este afectat de mărimea sistemului.

Odată ce cunoaștem valoarea costurilor suplimentare raportată la numărul de circuite integrate vom putea realiza o proiectare mult mai inteligentă pentru a aduce la minimum costul sistemului. De exemplu, vom putea determina pragul de la care este convenabilă înlocuirea a două circuite SSI printr-un singur circuit MSI care realizează aceiași funcție. Dacă notăm:

O = valoarea costurilor suplimentare raportată la numărul de CI,

S = costul unui circuit SSI,

M = costul unui circuit MSI

pragul corespunde situației în care costul unui circuit MSI (din punctul de vedere al costului sistemului) devine egal cu costul (tot din punctul de vedere al costului sistemului) al celor două circuite SSI care se înlocuiesc:

$$M + O = 2(S + O).$$

Rezolvînd această ecuație se găsește pentru M expresia:

$$M = 2S + O.$$

De exemplu dacă O = \$ 3,31 și S = \$ 0,17 atunci pragul va fi dat de $M = 2 \times 0{,}17 + 3{,}31$ = \$ 3,65. Cu alte cuvinte este posibil să înlocuim două circuite SSI de 17 cent cu un circuit MSI — numai dacă prețul circuitului MSI este mai mic de \$ 3,65. Deoarece într-un circuit MSI costul raportat la poartă sau la bistabil este întotdeauna mai mic decît costul acelorași porți sau bistabile SSI, un circuit MSI care are un preț de \$ 3,65 trebuie să aibă de cel puțin 3,65/0,34 = 11 ori mai multe porți sau bistabile decît cele două circuite SSI înlocuite.

Vom avea deci o nouă implementare mult mai economică, mult mai sigură în funcționare a aceleiași funcțiuni, chiar dacă ea utilizează de 12 ori mai multe porți sau bistabile! Este evident, în aceste condiții, că obiectivul nostru în

proiectarea logică *nu va fi* reducerea la minimum a numărului de porți sau de bistabile. Un obiectiv mult mai realist îl constituie astăzi aducerea la minimum a numărului de capsule de circuite integrate. Într-un anumit sens, această regulă importantă nu constituie în fond decît o altă formulare a regulii din Capitolul 1, deoarece calea pe care aducem la minimum costul este dată de utilizarea circuitelor MSI, a „componentelor ieftine", chiar dacă ele nu se potrivesc perfect la aplicația respectivă. În cazul exemplului anterior sîntem încă în cîștig, chiar dacă circuitul MSI se folosește cu o eficiență de numai 9%.

2.4. CREȘTERI CUANTIZATE ALE PREȚULUI

Conceptul de asamblare modulară descris anterior (vezi fig. 2-1) asigură în fabricarea sistemelor logice un anumit grad de standardizare. La fel ca în cazul oricărei standardizări cîștigul care se obține poate fi diminuat, într-o oarecare măsură, de ineficiența introdusă de faptul că standardul nu se potrivește întru-totul cu aplicația. De exemplu ce se întîmplă dacă în rack se pot plasa 250 de plăci cu circuite imprimate, iar sistemul este realizat numai pe 125 de plăci ? Desigur că în cazul în care volumul producției este corespunzător, putem proiecta și construi un rack cît jumătate din acela standard, dar volumul producției se va împărți între rack-ul mare și cel mic, făcîndu-le pe amîndouă mai scumpe.

Cu toate celelalte părți din sistem vom avea aceeași problemă : sertare pe jumătate pline, plăci cu cablaje imprimate pe jumătate pline și surse de alimentare încărcate pe jumătate. Acest aspect constituie efectiv unul din motivele pentru care un produs, cu o producție într-adevăr de volum mare, cum ar fi de exemplu un receptor TV, poate fi fabricat atît de economic : fiecare parte poate fi special proiectată pentru a se potrivi perfect aplicației. Atunci cînd volumul producției scade trebuie să alegem între niște componente standard care nu se prea potrivesc și niște componente speciale, scumpe.

În domeniul calculatoarelor, al echipamentelor periferice și al instrumentației un tip oarecare de sistem se produce în mod obișnuit în cantități de ordinul sutelor. La acest nivel de producție sîntem forțați să folosim standardizarea bazată pe sistemul de asamblare modulară, de tipul aceluia descris în figura 2-1. Ca urmare costul sistemului tinde să varieze în salturi cuantificate (vezi fig. 2-2). Se pornește cu costul de bază dat de rack, ventila-

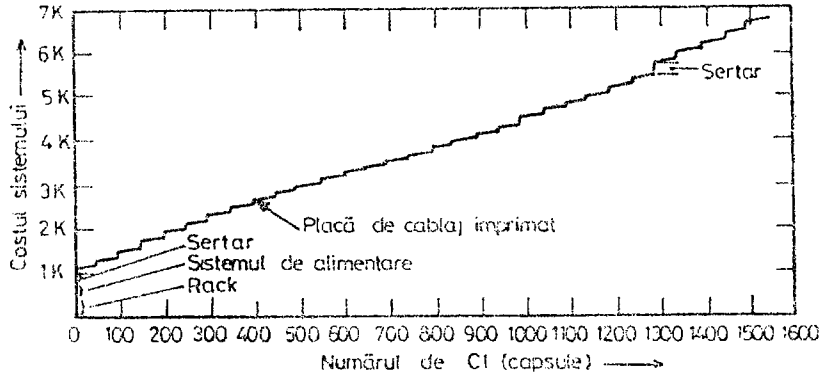


Fig. 2-2. Costul unui sistem în funcție de numărul de capsule de CI.

toare, surse de alimentare și de un sertar. Pe măsură ce numărul de CI din sistem crește, vom avea mici salturi ale costului, de fiecare dată cînd se umple o placă cu cablaj imprimat. Aceste salturi sînt echivalente cu costul multor CI, deoarece atunci cînd o placă cu cablaje imprimate este plină, un singur circuit integrat în plus determină acest salt al costului. Atunci cînd un sertar este plin, vom avea din nou un salt, mai mare, al costului; un singur circuit integrat în plus înseamnă adăugarea unui sertar suplimentar.

Aceste puncte cruciale trebuie luate foarte serios în considerație încă din stadiile inițiale ale proiectării. Pentru a simplifica activitatea de service și testare este de dorit ca logica să se împartă în module funcționale bine definite. Realizarea simultană a acestei împărțiri și a umplerii, complete a fiecărei plăci cu cablaj imprimat este aproape imposibilă. Această problemă — care va fi discutată ulterior mai amplu — constituie motivul pentru care relativ la costurile din Tabelul 2-1 am presupus utilizarea incompletă a modulelor de asamblare.

2.5. OBIECTIVELE *REALE* ALE PROIECTĂRII

Pînă în acest moment aproape toată discuția noastră a fost consacrată problemei reducerii la minimum a costului. Totuși există și alte obiective ale proiectării unui sistem digital care, în ordinea importanței, sînt următoarele:

1. Sistemul să *funcționeze* (de multe ori se întîmplă contrariul);
2. Siguranța în funcționare;
3. Simplitatea concepției (fabricație și service ușoare);
4. Economicitate.

Deși în această înșiruire economicitatea este pe ultima poziție, în continuare discuția se va purta în principal tot relativ la cost, deoarece costul este dat de un număr concret, care poate fi analizat cu oarecare certitudine. În plus, *toate* aceste obiective se optimizează urmînd aceeași cale. Sistemele care „nu reușesc să-și ia zborul" niciodată sînt în mod obișnuit excesiv de complexe. Siguranța în funcționare și economicitatea sînt optime atunci cînd numărul de componente a fost redus la minimum. Rezultă că, urmînd cele două reguli — „să adaptăm ceea ce avem de făcut la componentele ieftine" și „să reducem la minimum numărul de circuite integrate" — putem optimiza toate cele patru obiective.

Singura situație în care aceste obiective pot ajunge în conflict se referă la alegerea calității componentelor. Este sigur că a cumpăra o componentă ieftină reprezintă o falsă economie dacă în acest fel se sacrifică fiabilitatea. Din fericire, în ceea ce privește CI, în general nu acesta este cazul, deoarece cea mai ieftină modalitate de fabricație a CI se realizează pe baza ridicării la maximum a randamentului de fabricație, adică printr-o prelucrare perfectă. Deși există unele variații în calitatea CI furnizate de diferiți producători, aceste diferențe nu se corelează în mod uzual cu prețul lor.

2.6. STRUCTURA GENERALĂ A UNUI SISTEM DIGITAL

În acest moment, după ce am stabilit cîteva principii de proiectare foarte generale, putem trece la descrierea structurii generale a unui sistem digital și a modului în care se proiectează sistemul.

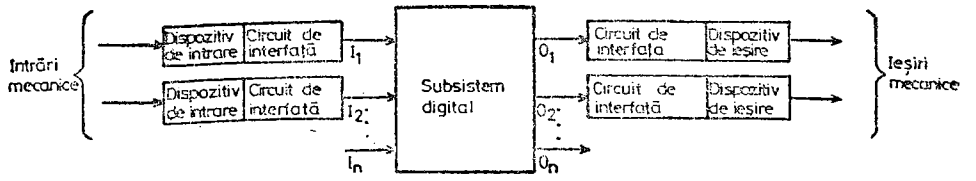


Fig. 2-3. Structura generală a unui sistem digital.

Structura generală a unui sistem este indicată în figura 2-3. Sistemul se compune din (*a*) unul sau mai multe „dispozitive de intrare" care convertesc mișcarea mecanică, poziția, lumina etc... în semnale logice binare, (*b*) „subsistemul logic digital" care furnizează la ieșire semnale binare care sînt funcții de ceea ce se primește la intrare și (*c*) unul sau mai multe „dispozitive de ieșire" care convertesc semnalele binare de la ieșirea subsistemului logic digital în mărimile mecanice* de ieșire pe care le dorim.

Dispozitivele de intrare sînt dispozitive electromecanice* și circuite de interfață pentru conversia semnalelor electrice (care sînt de obicei semnale analogice slabe) în semnale logice binare. Dispozitivul de ieșire realizează funcția inversă: circuitele de interfață convertesc semnalele logice binare într-o formă convenabilă (de obicei semnale analogice sau digitale de putere) pentru a acționa un dispozitiv electromecanic.

În Capitolul 13 sînt descrise în detaliu diverse dispozitive moderne de intrare și ieșire. Lista care urmează conține cîteva exemple de uz curent.

Dispozitive de intrare	Dispozitive de ieșire
Claviatură	Afișaj pe tub catodic
Cititor de bandă	Imprimantă
Cititor de cartele	Difuzor
Cititor de coduri grafice	Perforator de cartele
Cititor optic de caractere	Sonerie de alarmă
Microfon	Perforator de bandă
Termocuplu	Supapă
Traductor de presiune	Afișaj digital
Codor de poziție unghiulară	Bandă magnetică
Fotocelulă	Motor
Tachometru	Solenoid
Comutator	Bec

Un sistem cu adevărat complet, utilizabil, are întotdeauna intrări și ieșiri mecanice. Cu toate că multe produse electronice au intrări și ieșiri pur electrice — fiind de fapt subsisteme — ele sînt lipsite de utilitate prin ele însele. De exemplu un calculator este un subsistem care prin el însuși este inutil. El devine util numai atunci cînd i se conectează dispozitive de intrare și ieșire obținîndu-se astfel un *sistem* de calcul.

Toate proiectele trebuie să înceapă la acest nivel al sistemului total deoarece acesta este locul în care există cea mai mare șansă pentru adevărata creativitate. Odată ce este definită ideea generală a sistemului global, opțiunile pe care le putem face devin din ce în ce mai limitate pe măsură ce avansăm de la conceptele generale la detaliile efective ale proiectului. La nivelul

* Mecanice în sensul cel mai larg — inclusiv lumină, temperatură, sunet etc.

concepției sistemului de bază o schimbare a modului de abordare a problemei poate reduce adeseori amploarea lucrului care trebuie realizat cu un factor de 100 sau mai mult. Este evident, deci, că în această fază a proiectării nu trebuie să ne pripim deoarece simplificările din această etapă pot reduce în mod major efortul de proiectare care mai rămîne de făcut. Cele mai multe erori de proiectare își au sursa în abordarea pripită a acestor faze inițiale. Cu o concepție incorectă sau supercomplicată a sistemului mecanismul proiectării, verificării și producției este pornit pe o traiectorie greșită. Cu cît se investește mai mult timp într-o concepție incorectă cu atît ea este mai greu de schimbat — chiar dacă eroarea concepției inițiale a devenit evidentă. Mecanismul rîșnește în continuare pînă cînd proiectul este anulat sau compania eșuează.

2.7. SPECIFICAȚIA DE PROIECTARE

Modalitatea cea mai bună de a se evita un fiasco este de a se încerca să se rămînă în faza inițială de concepție a sistemului cît de mult este posibil. În momentul în care soluția corectă pentru sistem devine clară ea se va expune în scris într-o specificație de detaliu ; în acest fel se soluționează multe din detaliile de proiectare. Prin soluționarea acestor chestiuni de detaliu, în această etapă, concepția de proiectare devine mult mai clară punîndu-se totodată în evidență și punctele sale slabe. Foarte important este și faptul că se poate face o primă trecere la proiectare, fără a exista în realitate nici o legătură cu realizarea efectivă a sistemului. Sistemul poate fi ușor modificat deoarece el există numai sub forma unei specificații și nu ca „hardware" sau ca proiect de detaliu.

Alt avantaj oferit de o specificație detaliată de proiectare este dat de faptul că se evită interpretările eronate și surprizele neplăcute, odată ce specificația a fost aprobată de conducere și de compartimentul de marketing. Conducerea poate, efectiv, să revizuiască produsul înainte de a fi prea tîrziu pentru a-l schimba.

2.8. SĂ EVITAM COMPONENTELE „RELE"

Prima etapă în procesul de proiectare a sistemului de bază constă în a decide ce componente electromecanice vor fi utilizate pentru intrare și ieșire. Unele tipuri de componente mecanice sînt, din punctul de vedere al fiabilității, în mod inerent „rele" ; utilizarea lor trebuie să fie evitată. Deoarece logica digitală este relativ ieftină vom putea proiecta porțiunea logică a sistemului astfel încît în primul rînd ea să se potrivească oricărei interfețe de componentă electromecanică ; acest fapt ne oferă o mare flexibilitate în alegerea componentelor mecanice și permite, de cele mai multe ori, realizarea compromisului „mai multă logică, mai puține părți mecanice".

Tabelul care urmează listează unele componente care au fost surse de necazuri ; utilizarea lor trebuie evitată. Criteriile după care s-a întocmit acest tabel au fost mai întîi siguranța în funcționare și apoi costul. Din fericire, adeseori, ambele merg împreună deoarece rezultă din simplitate și standardizare.

Preferate	Rele (de evitat utilizarea)
Diode luminescente (LED)*	
Afişaje numerice cu LED	Cabluri de interconectare
Monitoare de televiziune	Motoare cu perii
Motoare pas cu pas	Comutatoare
Solenoizi	Relee
Fototranzistoare	Ajustări cu potenţiometre
Filme fotografice	Becuri
Acţionări magnetice	
Claviaturi	

Cu toate că în ordinea preferinţelor componentele ieftine stau pe primul loc, iar componentele mecanice sînt componente ieftine vom încerca, în general, ca într-un sistem să utilizăm un minimum de componente mecanice.

Pentru a lămuri mai bine modul în care se pot înlocui prin circuite logice componentele rele să dăm cîteva exemple pentru fiecare caz.

Cabluri de interconectare. Deşi cablurile nu pot fi eliminate în totalitate, numărul de cabluri se poate reduce semnificativ transmiţînd datele serie (cîte una o dată, într-o anumită ordine) pe o singură linie. În acest fel, de multe ori, se obţine şi *o reducere* a numărului necesar de CI. Această tehnică este discutată în detaliu în Secţiunea 11.8.

Motoare cu perii. Aceste componente se pot înlocui prin motoare cu inducţie sau motoare pas cu pas. Semnalele de comandă necesare (impulsuri dreptunghiulare) pot fi generate direct de logica digitală.

Comutatoare. Cu toate că eliminarea completă a comutatoarelor nu este posibilă, fiecare comutator trebuie să aibă doar un singur contact. Acest contact va genera un semnal digital, logica digitală realizînd apoi restul de comutări. Dacă sistemul are o claviatură şi un afişaj poziţiile comutatorului se pot modifica prin apăsarea într-o secvenţă dată a unui număr de taste. Starea la un moment dat a „comutatorului" se poate prezenta pe afişaj.

Pentru vechile calculatoare mecanice era specifică existenţa unui mare bloc de taste, cîte 10 pentru fiecare număr. La calculatoarele electronice este mult mai economic şi mai sigur în funcţionare să avem numai 10 taste şi să lăsăm părţii de logică sarcina de a deplasa într-un registru numerele în ordinea în care au fost introduse.

Relee. Pentru aproape fiecare funcţie realizată de un releu sînt disponibile dispozitive electronice echivalente. Semnalele de control sînt date de porţi logice obişnuite.

Ajustări cu potenţiometre. Abordarea digitală este în mod inerent lipsită de necesitatea de a realiza ajustări şi ca urmare nu va fi nevoie de potenţiometre de reglaj. De exemplu, în loc ca să folosim un potenţiometru pentru a ajusta analogic durata impulsului dat de un circuit monostabil vom realiza aceeaşi funcţie cu ajutorul unui numărător. Alte exemple de realizare digitală a unor funcţii analogice sînt date în Capitolul 13.

Becuri. Diodele luminiscente (LED) pot înlocui becurile în mod direct. Deoarece, de fapt, diodele luminiscente lucrează mai bine pentru factori de umplere mici, cablajul, logica şi electronica de comandă sînt utilizate de multe ori prin diviziune în timp. Comandînd un rînd de diode în mod periodic suficient de repede faţă de inerţia ochiului numărul de circuite de comandă se poate reduce la unul pentru un rînd şi la unul pentru o coloană.

---

* iniţialele LED sînt ale cuvintelor englezeşti *„light-emitting diodes"* (n.t.).

2.9. CONCEPTUL DE CUTIE NEAGRĂ

Unul dintre cele mai utile concepte în proiectarea de sisteme și în proiectarea logică este acela de „cutie neagră". Din cauza complexității extraordinare a majorității sistemelor digitale mintea noastră este pur și simplu incapabilă să înțeleagă dintr-o dată, în toate detaliile sale, un întreg sistem. Conceptul de cutie neagră ne permite să împărțim sistemul în felii mai mici care pot fi digerate mult mai ușor. Această împărțire o realizăm ignorînd unele detalii care sînt neimportante la nivelul de sistem la care se poartă discuția.

Atunci cînd gîndim la nivelul superior, al sistemului complet, nu este necesar să ne preocupăm de detaliile legate de porțile logice și circuitele basculante care există efectiv în sistem. Ca urmare, o întreagă parte a sistemului va fi reprezentată în schema sistemului printr-o cutie neagră (vezi fig . 2-3). În acest fel vom putea ignora detaliile legate de modul în care va funcționa logica și ne vom putea concentra asupra problemei în discuție — alegerea componentelor electromecanice.

Dacă discutăm funcționarea sistemului la un nivel și mai ridicat putem ignora chiar și detaliile modului în care se conectează la sistem dispozitivele electromecanice. La acest nivel întregul sistem este o mare cutie neagră cu intrări și ieșiri *mecanice*. Figura 2-4 oferă un exemplu specific : un calculator de buzunar cu o claviatură de intrare și un afișaj la ieșire. Putem redacta o specificație amănunțită pentru exact ceea ce trebuie să facă această cutie neagră fără a mai lua în considerație detaliile legate de ceea ce se află în interiorul ei. De exemplu, pentru un calculator de birou putem descrie ceea ce se va vedea la ieșire ca urmare a apăsării unor taste într-o secvență dată. De fiecare dată cînd apăsăm o tastă la ieșire va avea loc o modificare distinctă, bine definită. Această modificare va depinde nu numai de tasta apăsată ci și de toate celelalte taste apăsate după ultima apăsare pe tasta CLEAR*. Avem deci o definire completă a ceea ce trebuie să facă cutia neagră din figura 2-4 ; am definit deci o problemă de proiectare.

Pentru a merge mai departe cu proiectarea trebuie ca, în continuare, să împărțim problema de proiectare într-o serie de subprobleme mai mici, mult mai ușor de manevrat. Figura 2-5 ilustrează acest proces de împărțire pentru cazul unui calculator de buzunar. Pentru fiecare nivel succesiv al procesului de proiectare sîntem interesați de detalii tot mai fine astfel că numărul de subîmpărțiri necesar pentru obținerea unui bloc care să aibă o semnificație clară crește. Din fericire pe măsură ce trecem de la nivelul corespunzător sistemului general spre proiectarea detaliată a subsistemelor putem să ignorăm într-o măsură tot mai mare aspectele globale. Odată ce ne-am hotărît care sînt subsistemele și am specificat funcțiile lor, vom putea ignora în mod efectiv restul sistemului atacînd fiecare cutie neagră ca o problemă independentă de proiectare ; pentru găsirea soluției poate să fie, sau poate să nu fie, necesară o nouă subîmpărțire.

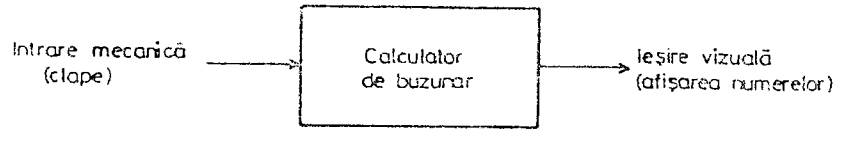


Fig.2-4. Sistemul logic : calculator de buzunar.

* *ȘTERGE*, (n.t.).

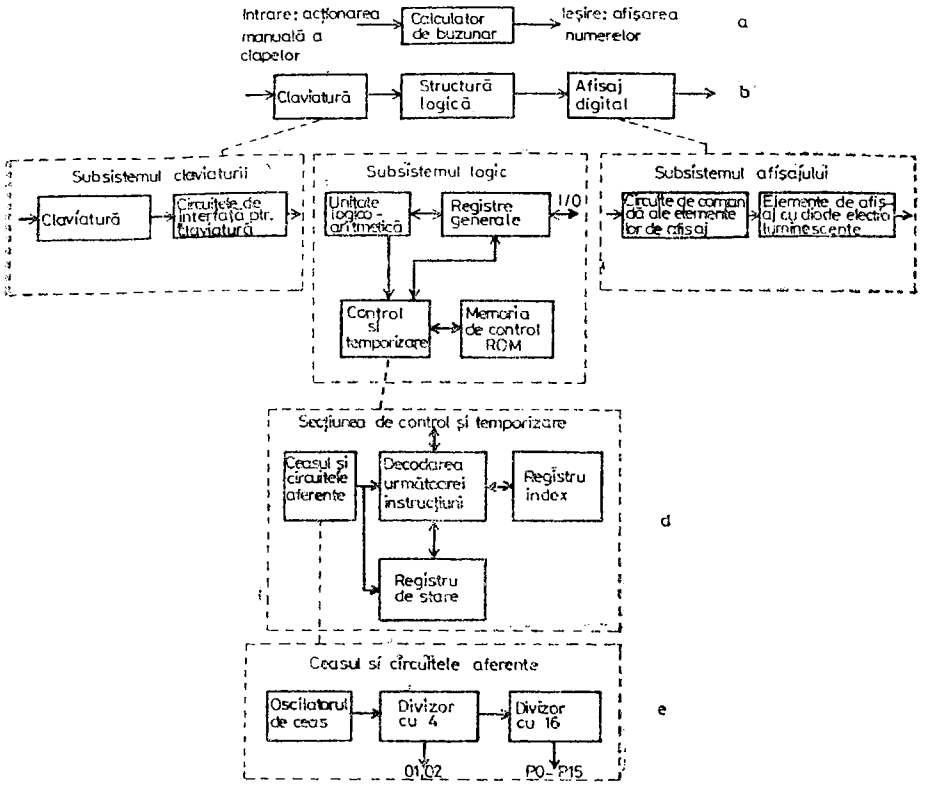


Fig. 2-5. Împărțirea sistemului logic-calculator în „cutii negre" (blocuri) *(d)*. Unul din cele opt blocuri. *(e)* Unul din subblocuri.

În concluzie proiectarea o realizăm mergînd de la ignorarea detaliilor de circuit la ignorarea detaliilor de sistem, focalizîndu-ne efortul numai pe aria corespunzătoare etapei curente de proiectare. În acest mod menținem în permanență întinderea problemei la o dimensiune care este optimă pentru posibilitățile noastre mentale. Dacă vom împărți problema în părți prea mici vom pierde timp ; dacă vom încerca să realizăm etape mari care se suprapun sîntem în situația de „a mușca mai mult decît putem mesteca". Un exemplu ar putea fi încercarea de a realiza proectarea logică de detaliu a întregului calculator din figura 2-5 *a* fără ca să îl mai subîmpărțim cel puțin pînă la nivelul din figura 2-5 *c*. Numărul necesar de nivele de proiectare este dependent de complexitatea sistemului și de capacitatea mintală a proiectantului.

Acest proces de subîmpărțire este util atît în înțelegerea cît și în proiectarea sistemelor. Prin utilizarea acestei tehnici vom putea manipula sisteme de orice grad de complexitate. Atunci cînd vom explica modul în care lucrează un sistem, vom începe prin a descrie sistemul în termeni generali, apoi vom pune în evidență subsistemele majore și blocurile lor constitutive pînă vom ajunge astfel la cele mai mici detalii. În proiectarea sistemelor foarte mari este de multe ori necesar ca efortul de proiectare să fie distribuit pe mai mulți ingineri. Dîndu-i de proiectat fiecăruia din ei un subsistem (o cutie neagră), bine specificat, eventualele erori și confuzii vor fi menținute la un nivel minim.

2.10. ÎMPĂRȚIREA ÎN SUBSISTEME

Între subîmpărțirea conceptuală a unui sistem în subsisteme de diverse mărimi și conceptul de asamblare modulară descris anterior există o asemănare evidentă. În ambele cazuri se încearcă împărțirea problemei care trebuie soluționată în părți care sînt mai ușor de manipulat, fiind comună o structură „arborescentă" în care trunchiul arborelui reprezintă sistemul integral. Ca urmare în mod evident va fi de dorit să se încerce unificarea acestor două sisteme de subîmpărțire. Odată realizată această unificare, fiecare sertar, fiecare placă de cablaj imprimat etc. va reprezenta de fapt un subsistem specific.

Cu toate că există și unele conflicte în încercarea de a face să coincidă aceste două tipuri de subîmpărțire, există și mult mai multe puncte comune. De exemplu este de dorit să reducem la minimum numărul de conexiuni de intrare și ieșire de semnal atît în abordarea conceptuală de subsistem cît și în modulul de asamblare. De asemenea este convenabil ca mărimea optimă a modulelor de plăci de cablaj imprimat să corespundă mărimii „cutiei negre" astfel încît în etapa finală a proiectării să apară interconexiunile dintre capsulele de CI reale.

Această împărțire în subsisteme este realizată în mod obișnuit în acea etapă a procesului de proiectare în care se dă specificația scrisă, constituind o decizie de proiectare foarte importantă care trebuie luată în condițiile unei cunoașteri efectiv complete a părții de „hardware" care va fi implicată în realizarea fiecărui subsistem. Deoarece subsistemele se proiectează din punctul de vedere îngust al optimizării proiectării subsistemului, șansele de a realiza o reducere reală a complexității sistemului — odată ce a fost atins acest nivel — sînt efectiv mici. Pentru sistemele mari este de multe ori util să se elaboreze în scris o specificație detaliată de proiectare pentru fiecare subsistem, similară ca obiectiv cu specificația de proiectare a sistemului global. Atît partea de „hardware" cît și documentația sistemului vor fi divizate apoi în aceiași structură de tip „arbore".

EXERCIȚII

1. Faceți un tabel similar cu Tabelul 2-1, utilizînd costurile obținute din documentația unui sistem concret.
   (*a*) Să se calculeze, utilizînd acest tabel, costul suplimentar raportat la numărul de CI.
   (*b*) Verificați rezultatul comparînd costul de fabricație real al sistemului cu costul calculat prin adăugarea costului real al CI la costul suplimentar raportat la numărul de CI multiplicat cu numărul de CI.
2. Să se calculeze costul suplimentar raportat la numărul de CI pentru un sistem al cărui cost de fabricație este de $ 16 000. Sistemul conține 5 000 de CI a căror cost total este de $ 1 800. Costul părților legate de ingineria sistemului este de $ 60 000. Se fabrică în total 40 de sisteme cu o viață medie de 5 ani. Costul activității de service este de $ 2 000/an.
3. Pentru un sistem care este asamblat ca în figura 2-1 și are costurile date în Tabelul 2-1, să se determine:
   (*a*) Cît costă în plus sistemul dacă i se adaugă un CI în plus?
   (*b*) Cît costă în plus sistemul în cazul în care introducerea acestui CI impune adăugarea unei noi plăci de cablaj imprimat?
   (*c*) Dacă toate sertarele sînt pline și adăugarea unui CI înseamnă de fapt adăugarea unui nou sertar cît costă în plus sistemul?
4. (*a*) Să se deseneze ca sistem (la fel ca în fig. 2-4) o rețea de televiziune indicînd toate intrările și ieșirile.
   (*b*) Se va împărți sistemul în subsisteme pînă la nivelul la care receptorul *TV* constituie un bloc.
   (*c*) Se va împărți receptorul *TV* în patru subsisteme, indicîndu-se toate intrările și ieșirile (inclusiv comenzile).

BIBLIOGRAFIE

Asamblare

Fordiani, Wm. O., „New Vistas for CAM", *1971 Wescon Technical Papers,* IEEE 1971.
Harper, C., *Hanbdook of Electronic Packaging,* McGraw-Hill, New York, 1969.
„Packaging With Integrated Circuits Course", *The Electronic Engineer,* March-August 1972 (constituie o trecere în revistă completă datorată mai multor autori).

Costuri

Samaras, Thomas T., „Estimating Project Costs — What the Textbooks Don't Tell," *Electronics,* February 28, 1972, pp. 90—92.

Cutii negre

Phister, M., *Logical Design of Digital Computers,* Wiley, New York 1958, pp. 144—148.

3

LOGICA COMBINAȚIONALĂ I

Proiectarea logică tradițională

3.1. INTRODUCERE

În următoarele trei capitole dorim să prezentăm modul în care se proiectează subsistemele logice de tipul celor din figura 2.3. Această activitate de proiectare este numită în mod uzual „proiectare logică" și pe această temă s-au scris multe cărți. Urmărim, în această lucrare, proiectarea într-o manieră elementară a structurilor de bază ce compun sistemele realizate cu circuite integrate. Scopul nostru este de a minimiza numărul capsulelor cu circuite integrate utilizate. Deoarece în această fază a proiectării folosim ecuații și diagrame abstracte, șansa noastră de a găsi o modalitate optimă de utilizare a circuitelor integrate standard este serios diminuată. Din acest motiv rezolvarea acestei probleme o vom amîna cît mai mult posibil.

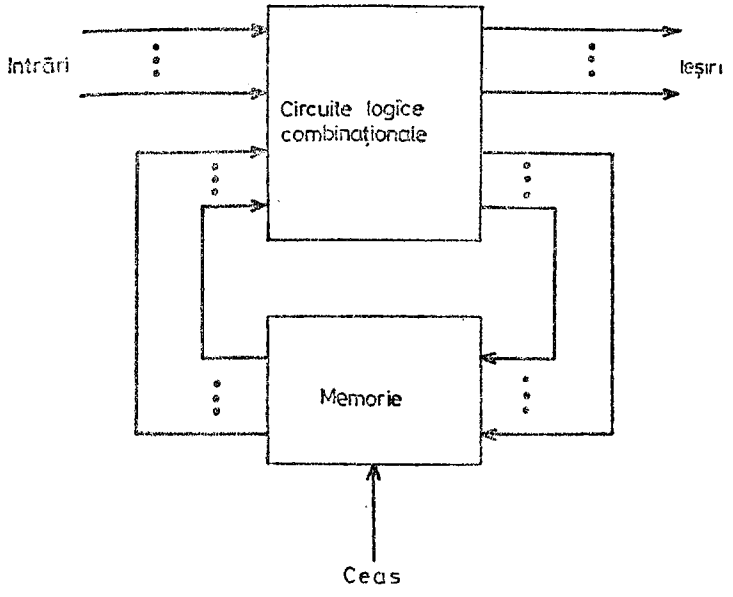
Odată porniți pe o cale incorectă, vom depune un efort inutil aplicînd tehnici de minimizare unor rețele de sute de porți logice pentru a obține soluția unei probleme ce putea fi de fapt rezolvată utilizînd un singur circuit LSI.

Un subsistem logic este pur digital dacă tuturor intrărilor li se aplică semnale electrice binare formate din două nivele de tensiune reprezentînd 1 și 0. De asemenea și ieșirile sînt caracterizate prin nivele de tensiune cu semnificație binară. Circuitele de interfațare, prezentate în figura 2.3, convertesc nivelele logice de la sau către intrările sau ieșirile mecanice.

3.2. CIRCUITE COMBINAȚIONALE – CIRCUITE SECVENȚIALE

Pentru orice set de semnale la intrarea unui sistem corespunde un set specific de semnale de ieșire în funcție de aplicație. De multe ori relația între semnalele de intrare și cele de ieșire este determinată și de evoluția anterioară a acestora. Spre exemplu, cînd utilizăm un calculator de buzunar prin apăsarea tastei „ = " este afișat un rezultat ce corespunde numerelor și operațiilor anterior specificate. Un astfel de subsistem logic este denumit „secvențial" și trebuie să conțină, sub o formă oarecare, posibilități de memorare. Cu toate că majoritatea sistemelor logice presupun și funcția de memorare, este util să le clasificăm în două categorii : secvențiale și combinaționale. *Un sistem logic pur combinațional nu posedă memorie și, în consecință, intrările în fiecare moment definesc complet ieșirile.* Din aceste motive analiza și proiectarea circuitelor logice combinaționale este mai ușoară.

Un subsistem logic secvențial poate fi divizat, conform reprezentării din figura 3.1 a, într-o zonă de memorie și un circuit pur combinațional. Această divizare



a

Nivel de integrare	Logică combinaţională	Memorii
Mic (SSI)	Porţi	Bistabili
Mediu (MSI)	Decodificatoare, multiplexoare, sumatoare	Registre, numărătoare
Mare (LSI)	Memorii fixe (ROM) Matrici logice programabile(PLA)	Memorii cu acces aleator(RAM) Registre de deplasare mari

b

Fig. 3-1. *(a)* **Structura unui circuit logic secvenţial.**
(b) **Tipuri de circuite pentru diverse niveluri de integrare.**

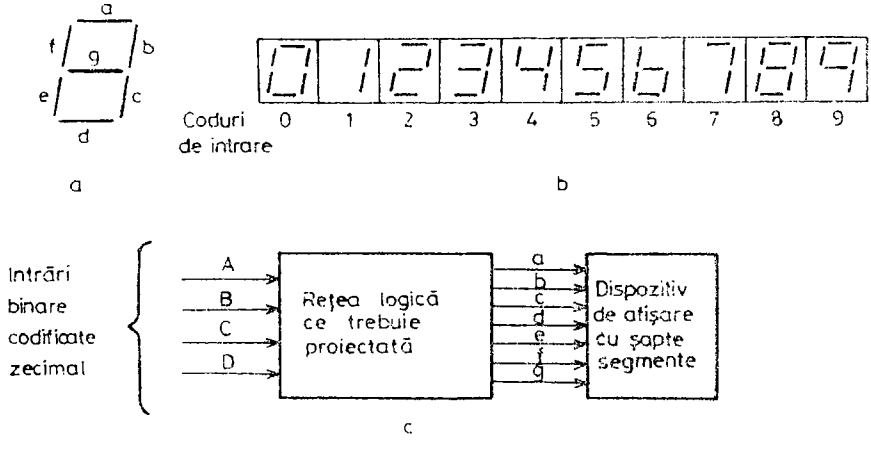
permite o abordare mai uşoară a acestor tipuri de subsisteme, deoarece odată separată zona de circuit combinaţional problema se reduce la o proiectare pur combinaţională. În figura 3.1 *b* sînt prezentate circuite tipice pentru fiecare nivel de integrare. Vom tinde, de cîte ori va fi posibil, către utilizarea unor circuite caracterizate printr-un nivel cît mai mare de integrare, deoarece o astfel de alegere poate garanta succesul produsului nostru. În Capitolul 5 este prezentat modul în care trebuie să alegem tipul de logică combinaţională pentru un sistem logic secvenţial. Pentru început vom presupune că am definit subsistemul logic combinaţional şi că în continuare trebuie să-l implementăm în modul cel mai eficient.

În următoarele două capitole sînt expuse o serie de tehnici de implementare utilizînd circuite integrate pe scară mică, medie şi largă (SSI, MSI şi LSI). Pentru început să vedem în ce mod pot fi reprezentate funcţiile logice.

3.3. DESCRIEREA FUNCȚIILOR LOGICE I: TABELUL DE ADEVĂR

Definirea concisă a problemei constituie prima etapă a oricărei activități de proiectare. De obicei problemele sînt definite inițial prin formulări generale, fiind necesară traducerea acestora sub forma *ecuațiilor logice* sau a *tabelelor de adevăr*. Spre exemplu, în figura 3-2, *a* este reprezentat un afișaj realizat cu șapte segmente luminoase. Să examinăm problema proiectării unei rețele logice care să genereze corect selectarea segmentelor corespunzătoare intrărilor **zecimale codate binar (binary coded decimal — BCD),** conform reprezentării din figura 3-2 *b*. Schema bloc a rețelei pe care o proiectăm este dată în figura 3-2 *c*. Pe cele patru intrări se aplică codul numărului de afișat iar cele șapte ieșiri generează semnalele de activare ale segmentelor ce trebuie aprinse.

Conform descrierii date de figurile 3-2 *a* și 3-2 *b* putem genera tabelul de adevăr al rețelei combinaționale (vezi fig. 3-2 *d*). Coloana din stînga nu face parte din tabelul de adevăr propriu-zis, avînd numai rolul de a facilita citirea



Cifra zecimală afișată	Intrări				Ieșiri						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

d

Fig. 3-2. *(a)* Notarea segmentelor. *(b)* Forma dorită a afișajului. *(c)* Structura logică asociată. *(d)* Tabelul de adevăr.

și înțelegerea acestuia. În rest, fiecare rînd este asociat unui număr de afișat iar fiecare coloană reprezintă un semnal de intrare sau de ieșire. Întrucît intrarea este dată de reprezentarea *BCD* a numerelor afișate sînt necesare patru coloane pentru definirea tuturor numerelor zecimale. Coloanele corespunzătoare ieșirii sînt completate ținînd cont de corespondența reprezentată de figura 3-2 *b*, introducînd 1 în tabel pentru aprinderea segmentului și 0 în cazul în care segmentul nu este selectat. De exemplu, numărul zecimal 1 presupune numai activarea segmentelor *b* și *c*, deci pe rîndul corespunzător vom completa 0110000. În această manieră putem completa tabelul de adevăr utilizînd corespondența între cod și imagine, presupusă de aplicație.

Obținem în acest mod translatarea într-o formă concisă, dată de tabelul de adevăr, a problemei general formulată, a afișării utilizînd șapte segmente. Sîntem, în acest moment, cu un pas mai aproape de soluția finală. În această etapă, majoritatea opțiunilor importante legate de proiectare (cum ar fi decizia de a utiliza reprezentarea *BCD* pentru intrări și pe cea cu șapte segmente pentru afișaj) au fost deja făcute. Dacă am fi ales un afișaj ce presupune un semnal de un bit pentru fiecare digit, tabelul de adevăr ar fi conținut 10 coloane pentru ieșiri, cu un singur 1 pe fiecare linie corespunînd numărului afișat.

Să observăm că, tabelul de adevăr din figura 3-2 *d* este de fapt incomplet. Deoarece pentru patru intrări există 16 combinații posibile, vom obține un tabel de adevăr complet dacă vom construi 16 rînduri, completînd cu *X*-uri în coloanele de ieșire pentru liniile adăugate, indicînd faptul că pentru aceste coduri de intrare suplimentare ieșirea nu este semnificativă.* Omiterea lor din tabelul de adevăr indică, de fapt același lucru.

3.4. DESCRIEREA FUNCȚIILOR LOGICE II : ECUAȚIA BOOLEANĂ

Tabelul de adevăr este ușor de generat și extrem de eficient pentru descrierea directă a numeroase funcții logice. De asemenea tabelul de adevăr pune în evidență corespondența acestor funcții logice cu realitatea deoarece fiecare coloană și linie au o semnificație precisă.

O altă modalitate foarte utilă de exprimare a funcțiilor logice este aceea a ecuațiilor logice. O ecuație logică este asociată unei singure funcții ; de exemplu pot fi scrise ecuații logice separat pentru fiecare funcție de ieșire, de la *a* la *g* din tabelul din figura 3.2 *d*. Folosirea ecuațiilor logice este foarte utilă în implementarea structurilor cu porți deoarece formal între ele există o corespondență directă. O mare parte din probleme sînt astfel formulate încît ecuațiile logice se pot scrie direct utilizînd descrierea neformală (cu cuvinte) a problemei. Cu toate că ecuațiile logice pot fi scrise direct pornind de la desenele din figura 3-2 *b*, legătura lor cu problema concretă poate să fie foarte abstractă.

Algebra Boole este deosebit de simplă deoarece operează numai cu două valori. Putem denumi aceste valori 0 și 1, adevărat și fals, nivel înalt sau coborît de tensiune, sau în alt mod. Algebra Boole poate fi descrisă utilizînd numai funcțiile logice „negare", „sau" și „și". Acestor funcții logice le corespund

* *„don't care"* în limba engleză (n.t.).

operațiile matematice de complementare, sumă și multiplicare dar semnificația lor este exact aceea pe care o au în limbajul curent. Din acest motiv o asertiune în limbajul curent poate fi scrisă sub forma unei ecuații logice. Pentru reprezentarea funcțiilor „negare", „sau" și „și" se utilizează simbolurile matematice $'$, $+$ și $\cdot$. Conform reprezentării din figura 3-2 *b* segmentul *e* este activ numai pentru afișarea numerelor 0 sau 2 sau 6 sau 8. Dacă vom utiliza simbolul $+$ pentru desemnarea funcției logice „sau", pentru variabile *e* se poate scrie următoarea ecuație logică :

$$e = 0 + 2 + 6 + 8 \tag{3-1}$$

Această ecuație trebuie transcrisă în funcție de variabilele de intrare A, B, C și D, care trebuie să înlocuiască simbolurile numerice zecimale. Deoarece codul pentru 0 este „A-negat și B-negat și C-negat și D-negat" (vezi fig. 3.2 *d*), putem scrie :

$$\text{cifra } 0 = A' \cdot B' \cdot C' \cdot D' \tag{3-2}*$$

De asemenea :

$$\text{cifra } 2 = A' \cdot B' \cdot C \cdot D' \tag{3-3}$$

$$\text{cifra } 6 = A' \cdot B \cdot C \cdot D' \tag{3-4}$$

și

$$\text{cifra } 8 = A \cdot B' \cdot C' \cdot D' \tag{3-5}$$

Înlocuind în (3-1) obținem :

$$e = A' \cdot B' \cdot C' \cdot D' + A' \cdot B' \cdot C \cdot D' + A' \cdot B \cdot C \cdot D' + A \cdot B' \cdot C' \cdot D' \tag{3-6}$$

Am obținut astfel ecuația logică asociată ieșirii *e* în funcție de variabilele de intrare A, B, C și D. Similar se pot scrie ecuațiile pentru celelalte ieșiri, după cum urmează :

$$a = A' \cdot B' \cdot C' \cdot D' + A' \cdot B' \cdot C \cdot D' + A' \cdot B' \cdot C \cdot D + A' \cdot B \cdot C' \cdot D + A' \cdot B \cdot C \cdot D + A \cdot B' \cdot C' \cdot D' + A \cdot B' \cdot C' \cdot D \tag{3-7}$$

$$b = A' \cdot B' \cdot C' \cdot D' + A' \cdot B' \cdot C' \cdot D + A' \cdot B' \cdot C \cdot D' + A' \cdot B' \cdot C \cdot D + A' \cdot B \cdot C' \cdot D' + A' \cdot B \cdot C \cdot D + A \cdot B' \cdot C' \cdot D' + A \cdot B' \cdot C \cdot D \tag{3-8}$$

$$c = A' \cdot B' \cdot C' \cdot D' + A' \cdot B' \cdot C' \cdot D + A' \cdot B' \cdot C \cdot D + A' \cdot B \cdot C' \cdot D' + A' \cdot B \cdot C' \cdot D + A' \cdot B \cdot C \cdot D' + A' \cdot B \cdot C \cdot D + A \cdot B' \cdot C' \cdot D' + A \cdot B' \cdot C' \cdot D \tag{3-9}$$

$$d = A' \cdot B' \cdot C' \cdot D' + A' \cdot B' \cdot C \cdot D' + A' \cdot B' \cdot C \cdot D + A' \cdot B \cdot C' \cdot D + A' \cdot B \cdot C \cdot D' + A \cdot B' \cdot C' \cdot D' \tag{3-10}$$

$$f = A' \cdot B' \cdot C' \cdot D' + A' \cdot B \cdot C' \cdot D' + A' \cdot B \cdot C' \cdot D + A' \cdot B \cdot C \cdot D' + A \cdot B' \cdot C' \cdot D' + A \cdot B' \cdot C' \cdot D \tag{3-11}$$

$$g \quad A' \cdot B' \cdot C' \cdot D' + A' \cdot B' \cdot C \cdot D + A' \cdot B \cdot C' \cdot D' + A' \cdot B \cdot C' \cdot D + A' \cdot B \cdot C \cdot D' + A \cdot B' \cdot C' \cdot D \tag{3-12}$$

Este evident că reprezentarea prin tabela de adevăr (figura 3-2 *d*) este mult mai avantajoasă pentru a exprima într-o formă compactă acest tip de funcție logică.

---

* Această expresie se citește astfel : „cifra zero este egală cu A prim și B prim și C prim și D prim". Negarea unei variabile se indică și printr-o *bară* superioară, de exemplu $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} = A'B'C'D'$ sau $\overline{ABC} = (ABC)'$. Notația cu prim are avantajul că este ușor de bătut la mașină și că poate fi scrisă de calculator fapt important pentru proiectarea automată.

Alte categorii de probleme pot fi exprimate mai bine direct, prin ecuații logice fără a se face apel la tabelul de adevăr. Putem exemplifica printr-un sistem ce nu permite pornirea automobilului dacă nu au fost asigurați toți pasagerii cu centuri de siguranță. Motorul va primi comanda de pornire dacă este conectat întrerupătorul de pornire *și* conducătorul automobilului s-a asigurat cu centura de siguranță *si* fiecare pasager este de asemenea asigurat *sau* prezența lor nu este detectată de senzorii de greutate. Această situație poate fi direct descrisă printr-o ecuație logică. Pentru început, vom simboliza fiecare senzor după cum urmează :

CP = comutator de pornire acționat
$CS1$ = centura de siguranță a conducătorului automobilului conectată.

$$\left.\begin{aligned} CS2 &= \\ CS3 &= \\ CS4 &= \end{aligned}\right\} \text{celelalte centuri de siguranță conectate}$$

$$\left.\begin{aligned} SG2 &= \\ SG3 &= \\ SG4 &= \end{aligned}\right\} \text{senzor de greutate activat}$$

Putem scrie deci :

$$START = CP \cdot CS1 \cdot (CS2 + SG2') \cdot (CS3 + SG3') \cdot (CS4 \cdot SG4') \qquad (3\text{-}13)$$

Deoarece avem opt variabile de intrare, numărul liniilor din tabelul de adevăr ce descrie această funcție este de $2^8 = 256$, fiind dat de toate combinațiile ce pot fi obținute cu variabilele de intrare. Este evident că în acest caz reprezentarea prin ecuația logică a problemei este net avantajoasă față de scrierea tabelului de adevăr.

3.5. FORMA CANONICĂ CE FOLOSEȘTE MINTERMENI

La fel ca în algebra obișnuită și expresiile booleene pot fi scrise sub diverse forme. Ecuațiile scrise anterior pentru afișajul cu șapte segmente utilizează reprezentarea destul de complicată a *formei canonice*. Există două tipuri de forme canonice : utilizînd *mintermeni* și utilizînd *maxtermeni*. Ecuațiile prezentate sînt scrise utilizînd mintermeni. Un *mintermen* este un produs logic conținînd fiecare variabilă o dată și numai o dată (fie variabila, fie negarea acesteia). Pentru patru variabile există $2^4 = 16$ mintermeni, după cum urmează :

$$\begin{aligned} m_0 &= A' \cdot B' \cdot C' \cdot D' \\ m_1 &= A' \cdot B' \cdot C' \cdot D \\ m_2 &= A' \cdot B' \cdot C \cdot D' \\ m_3 &= A' \cdot B' \cdot C \cdot D \\ m_4 &= A' \cdot B \cdot C' \cdot D' \\ m_5 &= A' \cdot B \cdot C' \cdot D \\ &\vdots \\ m_{14} &= A \cdot B \cdot C \cdot D' \\ m_{15} &= A \cdot B \cdot C \cdot D \end{aligned} \qquad (3\text{-}14)$$

Mintermenii astfel numerotați (se observă că numerele se obțin printr-o ponderare binară) constituie de fapt o modalitate „stenografică" de scriere a funcțiilor în forma canonică. Pentru exemplificare utilizăm ecuația 3-7 ce poate fi transcrisă direct astfel:

$$a = m_0 + m_2 + m_3 + m_5 + m_7 + m_8 + m_9 \qquad (3\text{-}15)$$

Deoarece numărul mintermenului corespunde liniei din tabelul de adevăr și numărului zecimal care apare afișat, această formă este nu numai mai scurtă dar și mai ușor de scris direct pornind de la figura 3-2 *b*.

O altă formă canonică, cea care folosește *maxtermeni*, este mai puțin utilizată. Maxtermenii (notați cu M) sînt sume logice scrise folosind toate variabilele de intrare, precum urmează:

$$\begin{aligned} M_0 &= A' + B' + C' + D' \\ M_1 &= A' + B' + C' + D \\ M_2 &= A' + B' + C + D' \\ &\vdots \\ M_{15} &= A + B + C + D \end{aligned} \qquad (3\text{-}16)$$

Trebuie observat faptul că maxtermenii iau valoarea „adevărat" pentru toate combinațiile de intrare cu excepția uneia singure. Spre exemplu M_0 este adevărat pentru toate combinațiile exceptînd pe 1111. Orice funcție poate fi scrisă ca un produs de maxtermeni. Ecuațiile 3-7 și 3-15, spre exemplu, pot fi scrise utilizînd maxtermeni astfel:

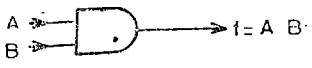
$$a = M_{14} \cdot M_{11} \cdot M_9$$

Formele care utilizează maxtermeni au drept corespondent gramatical „dubla negație". Constituind o manieră neuzuală de gîndire, folosirea lor nu este recomandată pentru definirea directă a ecuațiilor logice pornind de la descrierea neformală a problemelor.

3.6. IMPLEMENTAREA CU PORȚI A FUNCȚIILOR LOGICE

Cu toate că utilizarea circuitelor MSI sau LSI în implementarea funcțiilor logice este mult mai economică decît aplicarea metodelor tradiționale de interconectare complexă a porților logice elementare multe probleme, chiar în sisteme sofisticate, rămîn a fi rezolvate folosind circuite logice elementare. Sînt curente situațiile în care în sisteme complexe realizate cu circuite MSI și LSI, mici detalii se rezolvă utilizînd porți logice. Raportul dintre numărul capsulelor de circuite integrate conținînd porți și numărul capsulelor de circuite integrate MSI și LSI este de 2 : 1 pentru sistemele mici și de 1 : 1 pentru sistemele complexe. Cu toate că într-un sistem porțile elementare realizează de obicei o parte minoră din „munca" logică a sistemului, ele solicită o mare parte a muncii proiectantului la nivelul proiectării de detaliu și al verificărilor. Din acest motiv este foarte important ca proiectantul să stăpînească integral tehnicile simple de minimizare a rețelelor cu porți logice.

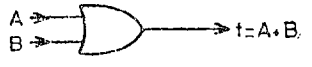
Porțile logice realizează funcții care pot fi definite atît prin tabele de adevăr cît și prin ecuații logice. În figurile 3-3 *a*, 3-3 *b*, 3-3 *c* sînt reprezentate porțile cu două intrări ce realizează funcțiile logice de bază. Poarta NAND



Intrare		Ieşire
A	B	f
0	0	0
0	1	0
1	0	0
1	1	1

(Ieşire adevărată dacă toate intrările sînt adevărate)

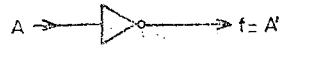
a



Intrare		Ieşire
A	B	f
0	0	0
0	1	1
1	0	1
1	1	1

(Ieşirea adevărată dacă una din intrări este adevărată)

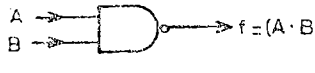
b



Intrare	Ieşire
A	f
0	1
1	0

(Ieşirea adevărată dacă intrarea este falsă)

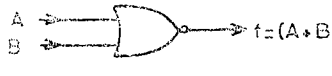
c



Intrare		Ieşire
A	B	f
0	0	1
0	1	1
1	0	1
1	1	0

(Ieşirea falsă dacă toate intrările sînt adevărate)

d



Intrare		Ieşire
A	B	f
0	0	1
0	1	0
1	0	0
1	1	0

(Ieşirea falsă dacă una din intrări este adevărată)

e

Fig. 3-3. *(a)* Poartă AND ; *(b)* Poarta OR ; *(c)* Inversor (NOT) ; *(d)* Poartă NAND ; *(e)* Poartă NOR.

(și-nu) este reprezentată în figură 3-3 *d*, fiind o combinație uzuală a funcțiilor AND (și) și NOT (negare). Denumirea NAND rezultă prin contracția expresiei NOT AND. Figura 3-3 *e* caracterizează funcția NOR (sau-nu), denumire rezultată prin contracția expresiei NOT OR. De observat faptul că funcțiile NAND și NOR sînt simbolizate utilizînd reprezentarea pentru AND și OR la care se adaugă un mic cerculeț la ieșire, simbolizînd funcția de inversor (NOT). Desigur, orice poartă definită pentru două intrări, poate fi gîndită și utilizată cu un număr oricît de mare de intrări. Tehnicile de încapsulare limitează numărul de intrări ale diverselor porți, după cum urmează : *patru porți cu două intrări, trei porți cu trei intrări, două porți cu patru intrări, o poartă cu opt intrări pe capsulă.*

Pentru implementarea unei ecuații logice putem folosi porți AND și OR conectate ca în figura 3-4 *a* ; de observat corespondența directă dintre o poartă AND cu patru intrări și un produs de patru variabile din componenta ecuației logice. O dată obținute cele patru produse urmează sumarea lor cu ajutorul unei porți OR cu patru intrări, rezultînd astfel funcția dorită. Pentru generarea variabilelor complementate (negate) la intrarea celor patru porți AND se pot utiliza patru **inversoare.** Corespondența directă între o structură de porți și o ecuație logică face ușor de înțeles de ce ecuațiile logice sînt atît de utile în proiectarea cu porți.

3.7. SIMPLIFICAREA FUNCȚIILOR LOGICE

În figura 3-4 *b* este prezentată o altă implementare posibilă a funcției obținute în figura 3-4 *a*. Motivul pentru care diferența dintre cele două reprezentări este foarte mare este determinat de faptul că în figura 3-4 *a* a fost implementată o ecuație asupra căreia nu au fost aplicate metode de minimizare ; ecuația a fost utilizată sub forma în care a rezultat din definirea formală a problemei. Situații similare apar curent în calculul algebric obișnuit unde pot exista formule susceptibile de simplificare. Deci, prima formă rezultată din definirea problemei necesită transformări matematice simplificatoare înainte de implementarea într-o structură concretă cu porți logice. Cu toate că ecuațiile logice se pot simplifica și prin procedee algebrice elementare, se vor putea utiliza diagrame simple care conduc la proceduri rapide de simplificare.

Pentru început să utilizăm procedurile algebrice pentru simplificarea circuitului din figura 3-4, *a* pînă la structura din figura 3-4 *b* care cu numai cîteva porți simple face atît de multe. Din analiza celor două implementări remarcăm faptul că cele două porți AND cu două intrări din figura 3-4 *b* ignoră cîte două variabile de intrare. Aceasta înseamnă că ieșirea este adevărată pentru grupuri de *patru mintermeni diferiți.* În figura 3-4 *a* au fost utilizate porți AND cu patru intrări pentru definirea mintermenilor, unul pe fiecare poartă. Procedeul folosit presupune folosirea porților cu un număr n cît mai mic de intrări pentru a defini grupuri de 2^n mintermeni necesari definirii funcției logice. Nu contează dacă unii mintermeni aparțin la două grupuri sau dacă vom utiliza și mintermeni care nu vor apare niciodată la intrările circuitului. În problema noastră la intrare se vor aplica numai codurile BCD, în număr de 10 (de la 0 la 9), mintermenii corespunzători numerelor de la 10 la 15 putînd fi considerați nesemnificativi, dar utilizabili în simplificarea funcției logice.

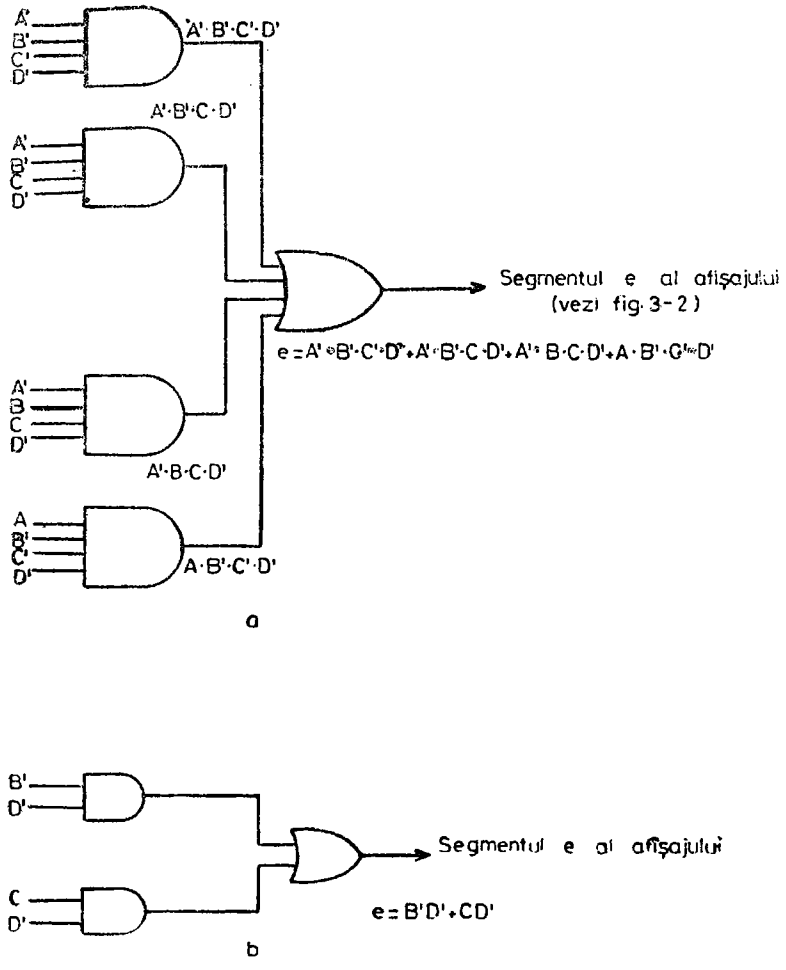


Fig. 3-4. *(a)* Implementarea cu circuite de tip AND/OR a ecuației 3-6. *(b)* Implementarea simplificată a aceleiași funcții.

Funcția $B' \cdot D'$ va genera cei patru mintermeni ce conțin $B' \cdot D'$ și cele patru combinații posibile ale variabilelor A și C:

$$B' \cdot D' = B' \cdot D'(A' \cdot C' + A' \cdot C + A \cdot C' + A \cdot C) =$$
$$= A' \cdot B' \cdot C' \cdot D' + A' \cdot B' \cdot C \cdot D' + A \cdot B' \cdot C' \cdot D' + A \cdot B' \cdot C \cdot D' =$$
$$= m_0 \quad + m_2 \quad + m_8 \quad + m_{10} \tag{3-18}$$

Forma canonică cu mintermeni a ecuației 3-6 este:

$$e = m_0 + m_2 + m_6 + m_8 \tag{3-19}$$

Observăm că $B' \cdot D'$ ne oferă trei din mintermenii doriți și, în plus, pe m_{10} care face parte din mintermenii nesemnificativi ce nu vor apare în mod normal la intrarea circuitului logic. Singurul mintermen ce mai trebuie obținut este m_6.

De asemenea, $C \cdot D'$ va mai genera încă patru mintermeni :

$$C \cdot D' = C \cdot D'(A' \cdot B' + A' \cdot B + A \cdot B' + A \cdot B) =$$

$$= A' \cdot B' \cdot C \cdot D' + A' \cdot B \cdot C \cdot D' + A \cdot B' \cdot C \cdot D' + A \cdot B \cdot C \cdot D' =$$

$$= m_2 + m_6 + m_{10} + m_{14} \qquad (3.20)$$

Am obținut mintermenul dorit, pe m_6, la care se adaugă încă o dată m_2 și mintermenii nesemnificativi m_{10} și m_{14}.

În cazul funcțiilor ce nu posedă mintermeni nesemnificativi vom putea face simplificări grupînd termeni care diferă numai printr-o variabilă. Spre exemplu :

$$A \cdot B \cdot C \cdot D + A \cdot B \cdot C \cdot D' = A \cdot B \cdot C \cdot (D + D') = A \cdot B \cdot C \qquad (3\text{-}21)$$

Prin scoaterea ca factor comun a produsului $A \cdot B \cdot C$, deoarece $D + D'$ este întotdeauna adevărat (dacă D este fals, D' este adevărat), vom putea scrie funcția simplificată sub forma $A \cdot B \cdot C$. De regulă este dificilă gruparea mintermenilor între ei sau adăugarea unora nesemnificativi pentru a se obține simplificări spectaculoase ale formei ecuațiilor logice. Pentru a simplifica această etapă a proiectării se utilizează tehnici ce presupun reprezentarea funcțiilor logice prin diagrame.

3.8. DESCRIEREA FUNCȚIILOR LOGICE III : DIAGRAME VEITCH

Diagramele Veitch sînt o formă particulară de tabel de adevăr, pentru o singură funcție logică, în așa fel constituite încît *mintermenii ce se pot grupa să fie adiacenți.* Diagrama Veitch pentru o funcție de patru variabile este reprezentată în figura 3-5 *a*. Diagrama este constituită din 16 pătrate, cîte unul pentru fiecare mintermen. Numărul înscris în pătrat indică numărul mintermenului reprezentat. Acoladele marcate prin A, B, C și D indică *zonele în care respectiva variabilă este adevărată ;* deci $B = 1$ în toate pătratele din jumătatea superioară a diagramei, în rest $B = 0$.

Utilizînd această reprezentare, putem defini orice funcție logică de patru variabile completînd cu 1 și 0 locațiile din diagramă. Spre exemplu, în figura 3-5 *b* este reprezentată funcția $f = A' \cdot B \cdot C' \cdot D$. Observăm că pătratul în care am înscris 1 se află în regiunea A' (deoarece se află în afara zonei notate prin A) *și* în zona B *și* în zona C' *și* în zona D. Diagrama reprezintă deci, la fel ca și ecuația logică sau tabelul de adevăr, funcția logică.

Utilitatea reală a reprezentării prin diagrame a funcțiilor logice rezultă și din exemplul din figura 3-5 *c*. Produsul a trei variabile de intrare apare reprezentat printr-o pereche *adiacentă* de locații. De fapt, *orice* produs de trei variabile (porți cu trei intrări) își găsește o reprezentare printr-o pereche de mintermeni adiacenți. În figura 3-5 *d* este ilustrată o situație limită a acestei reguli : *se poate extinde definiția „adiacenței" asupra marginilor opuse ale diagramei.* Dacă ne imaginăm diagrama înfășurată pe un cilindru, pe care-l putem gîndi atît vertical cît *și* orizontal simultan, vom putea obține o reprezentare completă a adiacenței în cadrul acestor diagrame.

Figurile 3-5 *e* și *f* ilustrează faptul că produsele de două variabile (porți cu două intrări) presupun gruparea a patru mintermeni adiacenți. Încă o dată

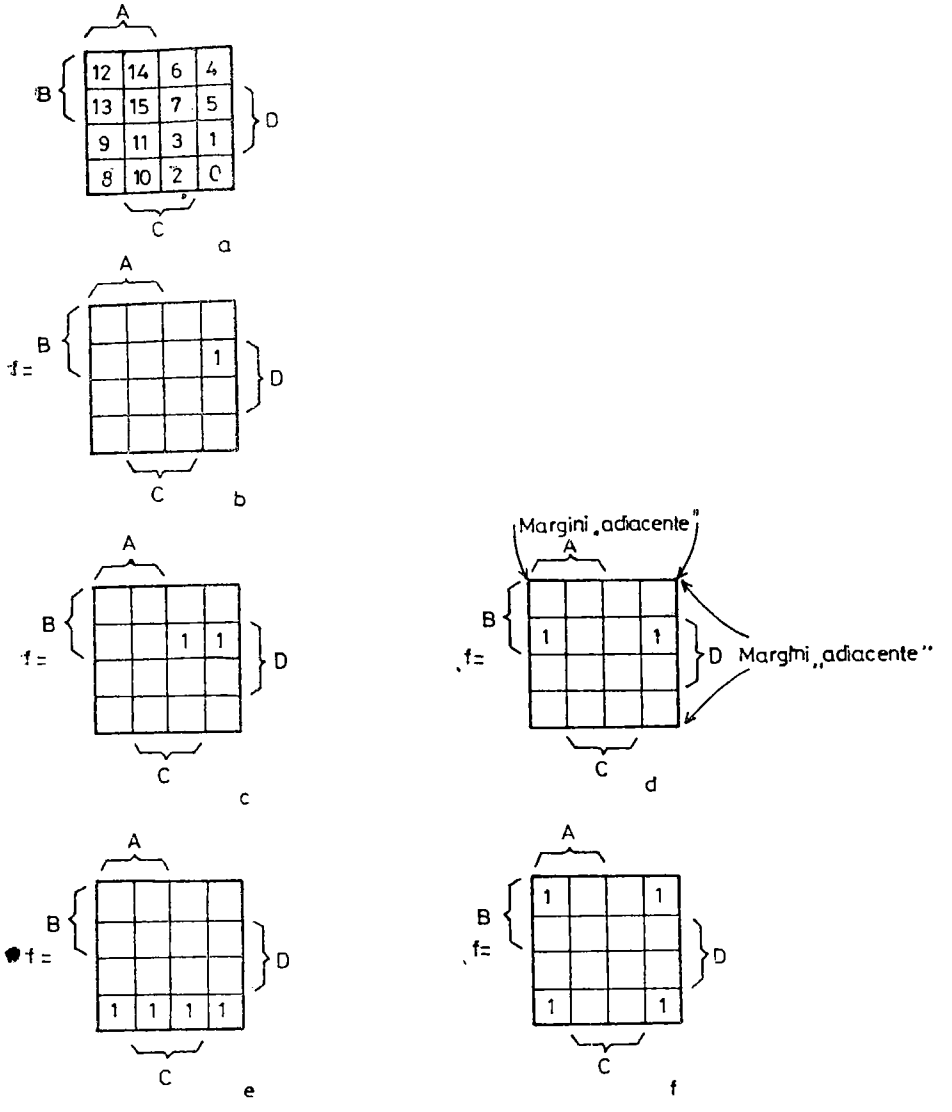


Fig. 3-5. Reprezentarea prin diagrame a funcțiilor de patru variabile : *(a)* numerotarea mintermenilor ; *(b)* funcția : $f=A'.B.C.'D'=M_5$; *(c)* funcția : $f=A'.BD$; *(d)* funcția : $f=B.C'.D$; *(e)* funcția : $f=B'.D$; *(f)* funcția : $f=C'.D'$.

trebuie să ne imaginăm că marginile opuse ale diagramei se ating pentru a forma (vezi fig 3-5 *f*) o zonă cu patru mintermeni adiacenți (cei patru 1). O funcție de o singură variabilă va presupune, desigur, adiacența a opt mintermeni (zona din diagramă denumită de aceea variabilă).

Pentru a demonstra utilitatea diagramelor Veitch ca instrumente de simplificare să construim diagrama următoarei funcții logice :

$$f = A \cdot B' \cdot C' \cdot D' + A \cdot B' \cdot C \cdot D' + A' \cdot B' \cdot D' \qquad (3\text{-}22)$$

Vom determina locațiile corecte (în ce pătrățele scriem 1 ?) asociate fiecărui termen procedînd prin eliminare. Spre exemplu, pentru primul termen A indică jumătatea stîngă a diagramei, B' restrînge zona de interes la cele patru locații ale colțului din stînga-jos, C' restrînge zona la două celule una deasupra celorlalte în colțul din stînga-jos, D limitează în final la pătratul din stînga-jos, deci m_8. Următorul termen mai adaugă un 1 în diagramă, iar ultimul impune completarea a două locații. Diagrama care rezultă este identică cu cea din figura 3-5 *e*, și imediat putem spune, dintr-o privire că ecuația 3-22 poate fi scrisă simplificat sub forma $f = B' \cdot D'$.

Simplificarea prin diagrame Veitch constă deci, în scrierea funcției în diagramă termen cu termen și apoi citirea ei ținînd cont de grupări corecte de termeni cît de mari este posibil. Atunci cînd prin simplificare rezultă mai mulți termeni, este utilă notarea mintermenilor eliminați prin încercuirea zonei ce i-a generat, pe măsură ce aceștia sînt extrași din diagramă.

În figura 3-6 sînt prezentate cîteva exemple. Menționăm faptul că este permisă introducerea aceluiași minterm în cadrul mai multor termeni pentru

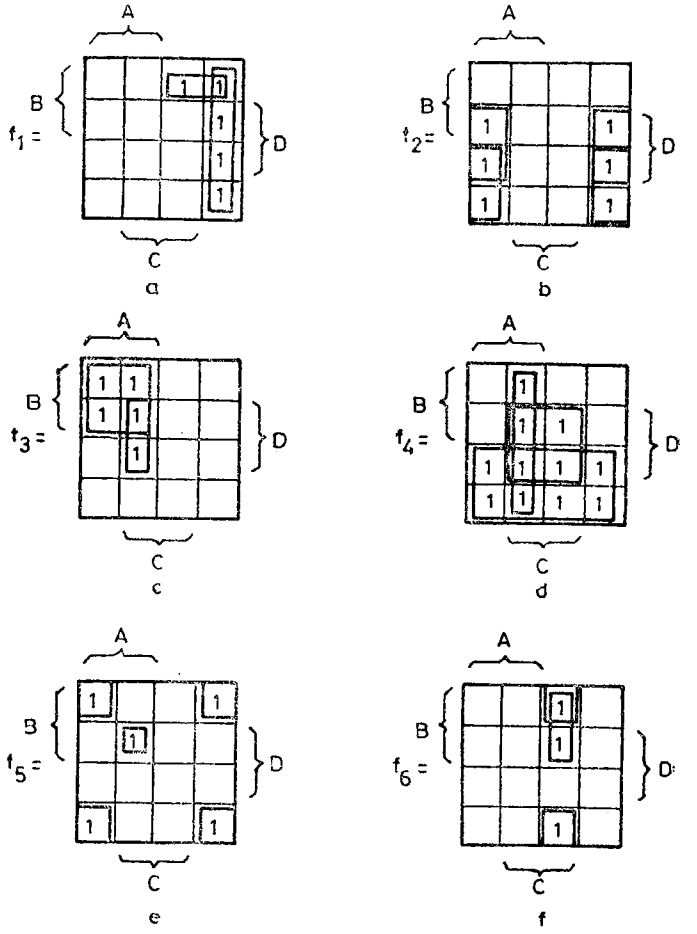


Fig. 3-6. Exemple de reprezentare prin diagrame: *(a)* $f_1 = A'.C' + A'.B.D'$; *(b)* $f_2 = C'.D + B'C'$; *(c)* $f_3 = A.B + A.C.D$; *(d)* $f_4 = B' + C.D + A.C$; *(e)* $f_5 = C'.D' + A.B.C.D$; *(f)* $f_6 = A'.C.D' + A'.B.C$.

a mări suprafața asociată termenilor respectivi. Observăm de asemenea, faptul că încercuirea ce include margini opuse (figurile 3-6 *b*, *e* și *f*) se poate imagina numai presupunînd atingerea marginilor opuse ale diagramei. Putem realiza grupuri de doi, patru sau opt mintermeni prin ignorarea unei variabile, a două variabile sau a trei variabile de intrare pentru termenul respectiv.

3.8.1. Utilizarea configurațiilor de intrare nesemnificative

De multe ori se întîlnesc situații în care o parte din posibilele combinații de intrare nu apar niciodată la intrarea circuitului. În acest caz putem considera că aceste combinații sînt nesemnificative pentru funcționarea circuitului, iar acesta poate avea orice comportare la ieșire. Putem folosi în avantajul nostru aceste situații punînd mintermenii nesemnificativi cu valoarea 1 sau 0, în funcție de modul în care putem obține o formă mai simplă pentru funcția logică.

În figura 3-7 *a* este prezentat un exemplu de folosire a mintermenilor nesemnificativi (redundanți) pentru a simplifica funcția *e* din afișajul cu șapte segmente (vezi fig. 3-2) la care ne-am referit anterior. Întrucît mintermenii de la 10 la 15 nu vor apare niciodată diagrama se completează în aceste locații cu *X*-uri. Vom introduce 1 în diagramă pentru mintermenii m_0, m_2, m_6 și m_8, conform definiției funcției *e* (segmentul *e* este aprins pentru cifrele 0, 2, 6 și 8). Vom pune în evidență doi termeni de două variabile considerînd că

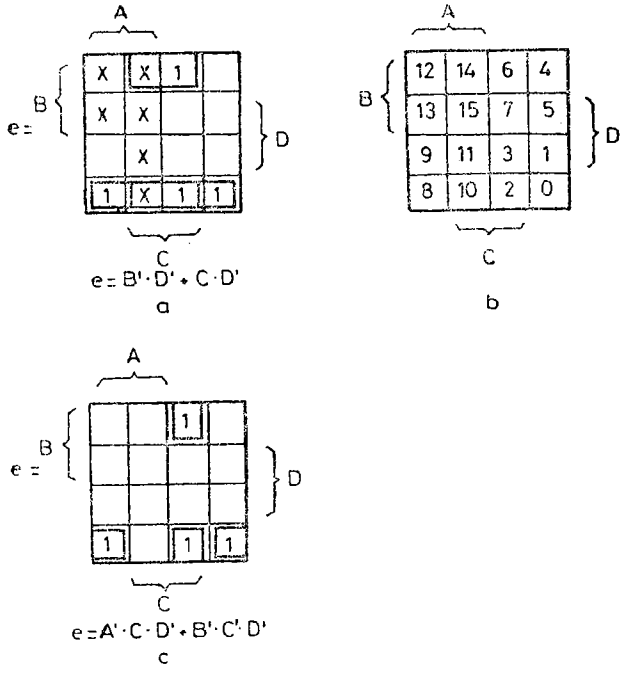


Fig. 3-7. Exemple de diagrame cu mintermeni nesemnificativi (don't care) : *(a)* segmentul *e* cu mintermeni nesemnificativi ; *(b)* numerotarea mintermenilor (pentru referință) ; *(c)* segmentul *e* definit fără a se ține seama de mintermenii nesemnificativi.

$m_{10} = m_{14} = 1$, iar restul de X-uri vor fi desemnate ca 0-uri. Funcţia rezultată este considerabil mai simplă faţă de cazul în care, nefolosind mintermenii redundanţi, am obţine doi termeni de trei variabile, după cum rezultă şi din figura 3-7 *c*.

3.8.2. Un exemplu de proiectare

Această modalitate de proiectare poate fi aplicată pentru afişajul cu şapte segmente definit în figura 3-2 soluţia problemei rezultînd pe o singură foaie de hîrtie cu pătrăţele, în cîteva minute după cum urmează (vezi fig. 3-8) :

1. Se începe prin a se desena o diagramă Veitch cu numerotarea mintermenilor marcată ca referinţă pentru întocmirea diagramelor propriu-zise, şi şapte diagrame necompletate, cîte una pentru fiecare segment.

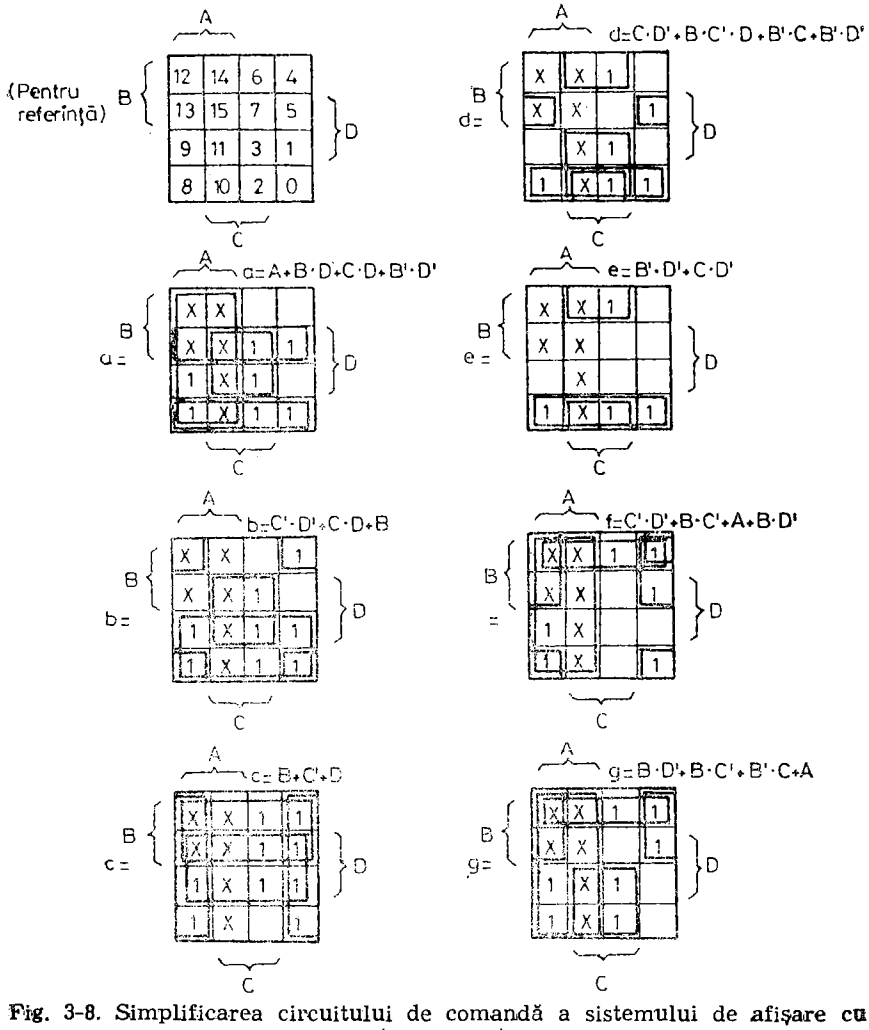


Fig. 3-8. Simplificarea circuitului de comandă a sistemului de afişare cu şapte segmente.

2. Urmează completarea cu X-uri a mintermenilor de la 10 la 15 în fiecare din cele şapte diagrame.

3. Se completează cu 1 în poziţia mintermenilor care definesc fiecare din cele şapte funcţii. Poate fi utilizat tabelul de adevăr (vezi fig. 3-2 *d*) sau definirea conform figurii 3-2 *b*.

4. Pe fiecare diagramă sînt încercuite blocuri de mintermeni pînă cînd fiecare este închis în cel puţin un bloc.

5. Din fiecare diagramă se extrag ecuaţiile conţinînd termenii definiţi prin gruparea mintermenilor în blocuri.

Dacă vom compara ecuaţiile rezultate cu cele nesimplificate de la 3—6 pînă la 3-12, scrise anterior vom observa reducerea considerabilă a numărului şi dimensiunii termenilor logici, cu consecinţe directe asupra numărului şi dimensiunii porţilor utilizate pentru implementare.

3.8.3. Utilizarea în comun a termenilor

Dacă analizăm grupul celor şapte funcţii rezultate, remarcăm faptul că există termeni ce sînt utilizaţi de mai multe funcţii. Spre exemplu, $C \cdot D$ este inclus atît în funcţia *a* cît şi în *b*. Poarta cu două intrări necesară pentru a genera $C \cdot D$ poate fi deci utilizată de ambele funcţii. Trebuie observat că este mult mai simplu să evidenţiem aceşti termeni comuni inspectînd diagramele decît analizînd ecuaţiile logice. Este suficient să căutăm zone încercuite, cu forme identice.

Pentru funcţii cu multe ieşiri ca în aplicaţia de faţă, este important să reţinem posibilitatea apariţiei termenilor comuni pentru a alege acele suprafeţe, încercuiri, ce permit *punerea în evidenţă* a cît mai multor termeni de acest fel. Spre exemplu, segmentul *a* putea fi simplificat optînd pentru o alegere de tipul celei din figura 3-9. În acest caz cel de al treilea termen al celor două ecuaţii rezultate este diferit, dar ambele sînt corecte şi conduc la acelaşi rezultat. Cu toate că în acest caz utilizarea în comun a unei porţi este posibilă pentru ambele alegeri, economisirea de porţi prin acest procedeu se obţine de multe ori numai printr-o anumită alegere.

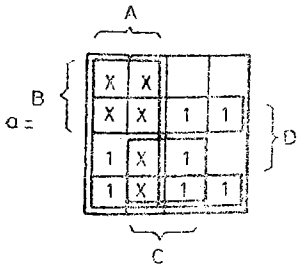
3.8.4. Implementarea funcţiei negate

Deoarece termenii produsului trebuie să includă toţi mintermenii funcţiei, deseori funcţiile pe care le definim au o formă dificilă de implementat. Spre exemplu, segmentul *c* este *activat (aprins)* pentru afişarea cifrelor 0, 1, 3, 4, 5, 6, 7, 8 şi 9. Putem exprima acest lucru mai simplu spunînd că segmentul *c* este *stins* (neselectat) atunci cînd este afişată cifra 2.

în figura 3-10 este redată diagrama funcţiei c' în loc de *c*, din care rezultă aceeaşi ecuaţie pentru *c* ca în figura 3-8, dar scrisă în alt mod.

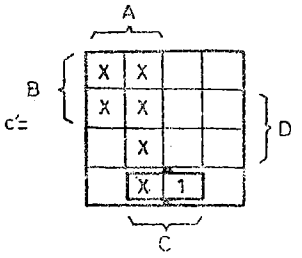
3.8.5. Alte tipuri de diagrame

Diagramele Veitch utilizate în această carte constituie o formă posibilă dintr-o clasă mai largă de diagrame. Aceste au fost alese în principal datorită faptului că sînt uşor de utilizat pentru scoaterea termenilor formei minimizate. O altă metodă de marcare a mintermenilor este prezentată în figura 3-11. Acestea se numesc diagrame Karnaugh.



$a = A + B \cdot D + \bar{B} \cdot C + B' \cdot D$

Fig. 3-9.



$c' = B' \cdot C \cdot D'$

$c = (B' \cdot C \cdot D')'$

Fig. 3-10.

AB \ CD	00	01	11	10
00	0	1	2	3
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Fig. 3-11. Diagramă Karnaugh cu numerotatea mintermenilor.

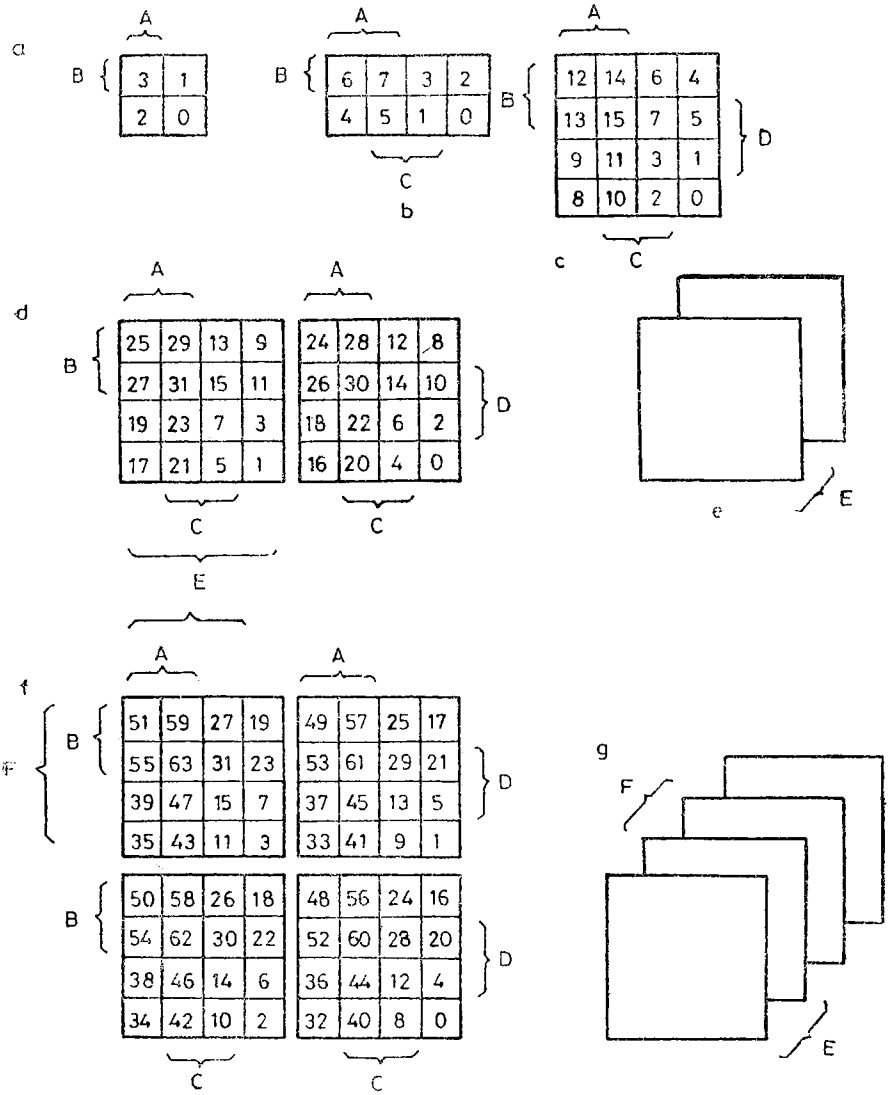


Fig. 3-12. Diagrame Veitch pentru două pînă la șase variabile (cu numerotarea mintermenilor): *(a)* două variabile; *(b)* trei variabile; *(c)* patru variabile; *(d)* cinci variabile; *(e)* reprezentarea spațială a *(d)*; *(f)* șase variabile; *(g)* reprezentarea spațială a *(f)*.

Pînă acum în toate exemplele s-au descris diagrame Veitch pentru patru variabile ; în figura 3-12 sînt reprezentate diagrame pentru două pînă la șase variabile. Pentru mai mult de patru variabile nu există o reprezentare în plan care să permită combinarea mintermenilor adiacenți. Putem obține o reprezentare spațială a diagramelor pînă la șase variabile distribuind în spațiu diagrame de patru variabile ca în figurile 3-12*e* și *g*. Putem obține un grup compact de mintermeni pentru o reprezentare cu unul, două sau patru planuri în adîncime, dacă aceștia sînt aliniați între planuri. Pentru a evita confuziile datorate suprapunerii diagramelor, acestea pot fi distribuite într-un plan, ca în figurii 3-12*d* și *f*. Gruparea mintermenilor în diagramele de patru variabile cu mintermeni din planuri diferite va fi marcată prin linii curbe trasate între planuri. Dacă desenul poate fi realizat la o scară suficient de mare, metoda este uneori aplicabilă pînă la opt variabile. Menționăm că pentru opt variabile diagrama rezultă ca o diagramă Veitch formată din 16 diagrame Veitch.

3.9. FACTORIZAREA ECUAȚIILOR LOGICE

Ecuațiile logice pot fi factorizate la fel ca și expresiile algebrice normale, avînd uneori ca rezultat micșorarea numărului de porți utilizate. De exemplu, ecuația logică simplificată pentru segmentul *e*

$$e = B' \cdot D' + C \cdot D' \tag{3-22}$$

poate fi scrisă

$$e = D' \cdot (B' + C) \tag{3-23}$$

În figura 3.13 este prezentată implementarea cu porți, care este oarecum mai simplă decît cea nefactorizată prezentată în figura 3-4*b*.

Avantajele sînt mai mari atunci cînd se poate da în factor o expresie mai complexă. De exemplu :

$$f = A \cdot B \cdot C \cdot D + A \cdot B \cdot C \cdot E$$

poate fi scrisă

$$f = A \cdot B \cdot C \cdot (D + E),$$

ceea ce presupune folosirea unei porți cu patru intrări și a uneia cu două intrări, față de expresia nefactorizată ce necesită două porți cu patru intrări și una cu două.

Factorizarea își dovedește utilitatea și atunci cînd putem separa prin acest procedeu o serie de variabile dintr-o funcție, rămînînd ca restul funcției să fie minimizat cu diagrame Veitch de mai puține variabile, deci mai simple.

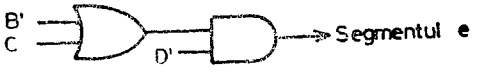


Fig. 3-13.

3.10. TEOREMA LUI DE MORGAN

Majoritatea identităţilor fundamentale ale algebrei Boole sînt aproape evidente şi toate pot fi verificate uşor utilizînd tabele de adevăr. Aceste identităţi sînt listate în Tabelul 3-1.

Tabelul 3-1. **Identităţi booleene**

1. $A \cdot B = B \cdot A$	legea comutativităţii
2. $A + B = B + A$	
3. $(A + B) + C = A + (B + C)$	(legea asociativităţii)
4. $(A \cdot B) \cdot C = A(B \cdot C)$	
5. $A(B + C) = AB + AC$	(legea distributivităţii)
6. $A + B \cdot C = (A + B) \cdot (A + C)$	
7. $A + A' = 1$	
8. $A \cdot A' = 0$	
9. $A + A = A$	
10. $A \cdot A = A$	
11. $(A')' = A$	
12. $A + 1 = 1$	
13. $A \cdot 0 = 0$	
14. $(A + B)' = A' \cdot B'$	(teorema lui De Morgan)
15. $(A \cdot B)' = A' + B'$	

Deoarece există o relaţie de corespondenţă directă între ecuaţiile logice şi reţelele de porţi logice, este utilă considerarea acestor identităţi ca fiind asertiuni referitoare la porţile logice. De exemplu, prima şi a doua identitate din Tabelul 3-1 semnifică faptul că intrările porţilor logice pot fi conectate în orice ordine. Multe din următoarele identităţi nu sînt deosebit de interesante, dar ultimele două sînt deosebit de importante în condiţiile raportării la reţelele de porţi logice. A 15-a identitate, spre exemplu, semnifică faptul că *o poartă NAND* (de exemplu ca în fig. 3-3 *d*) *este echivalentă cu o poartă OR cu intrările negate.* Acest lucru poate fi uşor demonstrat folosind tabelul de adevăr prezentat în Tabelul 3-2.

Tabelul 3-2. **Teorema lui de Morgan**

A	B	A,B	$(A \cdot B)'$	A'	B'	$A' + B'$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Teoremele lui De Morgan pot fi înţelese uşor şi direct, intuitiv. De exemplu, dacă vom gîndi variabilele A şi B ca semnificînd prezenţa a două persoane într-o cameră, vom putea afirma în egală măsură „A şi B nu sînt aici" sau „A nu este sau B nu este aici". Deci $(A \cdot B)'$ şi $A' + B'$ sînt două moduri distincte prin care putem exprima acelaşi lucru.

3.11. LOGICĂ DE TIP NAND/NAND

O consecință importantă a teoremelor lui De Morgan o constituie faptul că *aceeași structură de poartă logică poate realiza funcția logică de NAND sau NOR în funcție de modul în care definim nivelele logice.* Cînd vorbim de un circuit NAND în mod uzual considerăm că 1 logic este reprezentat de *nivelul înalt* al semnalului (logică *pozitivă*). Dacă redefinim semnificația semnalului electric astfel încît nivelul coborît este 1 (logică *negativă*) același circuit logic va realiza funcția NOR. Pentru a defini univoc funcția circuitului fizic este necesar să indicăm semnificația nivelelor semnalului. Putem prezenta aceste semnificații prin completarea tabelelor de adevăr cu simboluri ce reprezintă nivele electrice, folosind notațiile H = nivel înalt (high) de tensiune și L = nivel coborît (low) de tensiune. Se pot defini astfel două tabele de adevăr pentru fiecare circuit logic depinzînd de modul în care se stabilesc nivelele logice ; adevărat pentru nivel electric înalt sau pentru nivel electric coborît. Un exemplu este dat în figura 3-14. Se observă că același tip de circuit poate îndeplini fie funcție logică NAND, fie NOR.

De fapt teorema lui De Morgan pune în evidență faptul că funcțiile NAND și OR sînt complementare : OR evidențiază apariția la una din intrări a semnalului logic 1, pe cînd NAND indică apariția la intrări a semnalului 0 logic. Inversînd semnificația logică a semnalelor de intrare se schimbă funcția NAND în OR.

De observat că *micile cerculețe atașate simbolurilor logice semnifică faptul că valoarea de adevăr o au semnalele de nivel coborît,* iar absența lor indică valoarea logică de adevăr marcată prin nivelul ridicat al semnalului. Simbolul din stînga figurii 3-14 este o poartă AND cu intrări active pe semnal înalt și ieșire activă pentru semnal coborît (adică NAND pentru semnale active pe nivel înalt). Simbolul din dreapta este o poartă OR cu semnale active pe nivel coborît la intrare și active pe nivel înalt la ieșire (adică o poartă NOR pentru

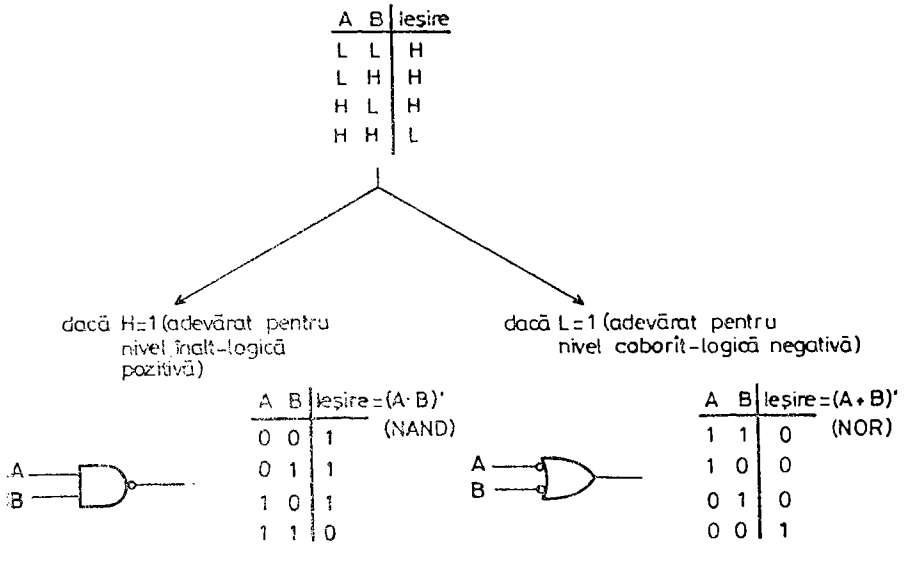


Fig. 3-14.

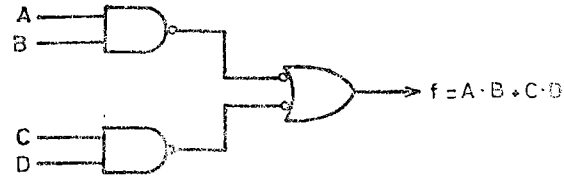


Fig. 3-15. Implementarea cu logică NAND/NAND — în logică pozitivă.

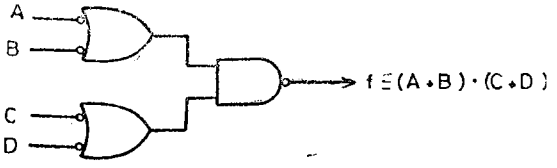


Fig. 3-16. Implementarea cu logică NAND/NAND — în logică negativă.

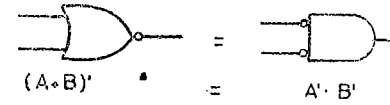


Fig. 3-17. Teorema lui De Morgan pentru funcția NAND.

semnale active pe nivel coborît). Dacă conectăm acest tip de poartă într-o structură cu două nivele, ca în figura 3-15, *obținem o structură echivalentă AND-OR cu toate că cele trei porți sînt identice, fiind fizic același tip de circuit.*

Semnalele active pe nivel coborît de la ieșirea primului nivel AND determină ca cel de al doilea nivel să acționeze la ieșire ca un OR cu semnale active pe nivel înalt.

Dacă vrem să definim nivele logice active pentru semnal coborît, atunci aceeași structură cu două nivele NAND/NAND devine o funcție OR-AND, ca în figura 3-16. Și în acest caz tabelele de adevăr construite cu simboluri *H* și *L* vor arăta că circuitul realizează de fapt în ambele cazuri aceeași funcție. Singura diferență ce apare este dată de interpretarea pe care o dăm nivelelor logice. Transformări similare în ecuațiile logice se pot obține prin utilizarea teoremei lui De Morgan.

De fapt, formularea egalității între două simboluri de porți în figura 3-14 este o exprimare, în termeni de porți mai curînd decît de ecuații logice, a teoremei lui De Morgan (vezi fig. 3-17).

Este deci ușor de înțeles de ce utilizarea porților NAND a devenit atît de răspîndită : *orice* funcție logică poate fi implementată utilizînd un *singur tip de circuit logic.*

3.12. CÎTEVA EXEMPLE DE LOGICĂ NAND/NAND

Să prezentăm cîteva exemple concrete de logică NAND/NAND. Ecuația logică din figura 3-8 pentru segmentul *b* este implemetată cu porți NAND în figura 3-18. Se observă că termenul *B'* este de fapt conectat la *B* deoarece

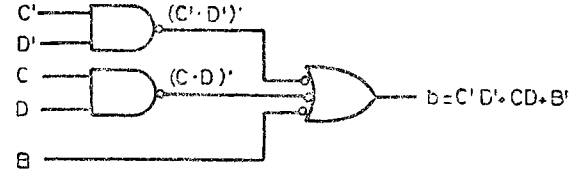


Fig. 3-18. Segmentul *b*.

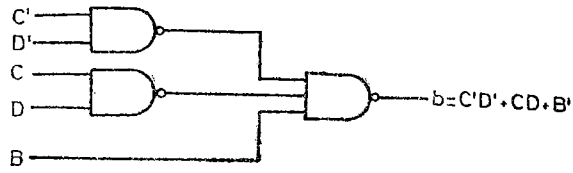


Fig. 3-19. Segmentul b.

convenţia logică pe care o utilizăm este semnalul de nivel înalt adevărat iar intrările în poarta ce corespunde nivelului secund trebuie să fie adevărate pentru nivelul coborît al semnalului pentru a realiza funcţia OR. De observat de asemenea că în acest caz poarta cu trei intrări a fost reprezentată ca OR cu intrări active pe nivel coborît numai şi numai pentru a se obţine o structură uşor de înţeles. Deoarece ea este de fapt tot o poartă de tip NAND reprezentarea din figura 3-19 este la fel de corectă.

Este important ca să se înţeleagă că cele două moduri de a desena o poartă NAND sînt echivalente deoarece uneori este imposibil să se deseneze întreg circuitul în forma uşor de citit din figura 3-18. Putem cerceta un circuit şi scrie o ecuaţie logică asociată fiecărei ieşiri *mergînd cu analiza de la ieşire spre intrare*. Pentru logica NAND cu semnalul de nivel înalt adevărat secţiunea de ieşire este întotdeauna o funcţie OR cu întrări active pe semnal coborît. Pe măsură ce parcurgem circuitul de la ieşire spre intrare secţiunile sînt alternative funcţiilor OR pentru nivelele impare şi AND pentru nivelele pare. În acelaşi mod pentru convenţia logică putem gîndi că la ieşire este semnal de nivel înalt adevărat şi alternează între semnal înalt adevărat-semnal coborît adevărat cu fiecare nivel.

În figura 3-20 se prezintă ca exemplu un circuit cu patru nivele (secţiuni). Pentru acest circuit ecuaţia logică pentru f_1 se poate scrie direct începînd cu poarta finală OR (intrări active pe semnal coborît), continuînd apoi spre intrare nivel cu nivel. Desenarea porţii NAND alternativ ca NAND sau ca OR cu intrări active pe semnal coborît este extrem de utilă în acest caz. Cealaltă funcţie, f_2, este un exemplu pentru situaţiile în care avînd funcţii logice cu mai multe ieşiri este imposibil de obţinut pentru toate ieşirile o reprezentare uşor de utilizat pentru scrierea funcţiei logice asociate. Semnalul $C' + A \cdot B$ este pe nivel impar pentru funcţia f_1 şi pe unul par pentru funcţia f_2. Ca urmare pentru a scrie funcţia f_2 direct cele două porţi care generează semnalul $C' + A \cdot B$ trebuie imaginate ca fiind desenate în forma alternativă. Dacă

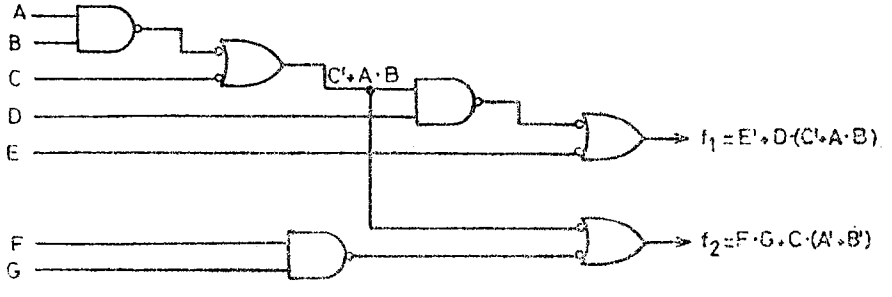


Fig. 3-20.

această afirmație pare confuză se observă că pur și simplu putem transforma funcția $[C' + A \cdot B]'$ considerînd-o o ecuație căreia îi aplicăm teorema lui De Morgan :

$$[C' + (A \cdot B)]' = C \cdot (A \cdot B)' = C \cdot (A' + B') \tag{3.26}$$

Utilizarea a două reprezentări pentru poarta NAND este strict legată de convenții de proiectare și de analiză a circuitelor logice. Este important să înțelegem că de fapt sînt două căi de a descrie aceeași funcție.

3.13. LOGICA DE TIP NOR/NOR

Întrucît este mult mai natural să gîndim în descrierea funcțiilor logice în termeni AND/OR, structurile de tip NAND/NAND s-au dovedit a fi cele mai comod de utilizat. Unele tipuri de structuri logice produc însă într-o modalitate mult mai naturală funcția NOR. Este important, totuși să ne reamintim faptul că aceste circuite sînt desemnate cu funcția NOR în virtutea unei convenții asupra semnificației nivelelor semnalelor de intrare și ieșire (semnalul de nivel înalt adevărat). Schimbînd această convenție (semnalul de nivel coborît să fie adevărat) ele devin circuite de tip NAND.

În anumite situații poate fi utilă însă structura de tip NOR/NOR. Este cazul definirii funcțiilor logice ca produse de maxtermeni, deoarece două nivele NOR sînt echivalente cu o structură de tip OR/AND. Un argument în plus este cunoscuta dualitate ce caracterizează funcțiile NOR și NAND. Spre exemplu, o poartă NOR cu semnal activ pe nivel înalt poate fi reprezentată în două moduri echivalente, conform teoremei lui De Morgan (vezi fig. 3-21).

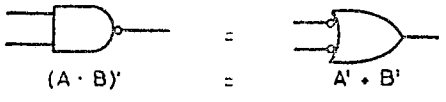


Fig. 3-21. Teorema lui De Morgan pentru funcția NOR.

De asemenea funcția logică a unei rețele de porți poate fi analizată similar logicii NAND/NAND parcurgînd, începînd de la ieșire, nivelele de porți și considerîndu-le ca porți AND cu intrări active pe semnal coborît pentru nivelele impuse și ca porți OR pentru nivelele pare. Un exemplu apare în figura 3-22.

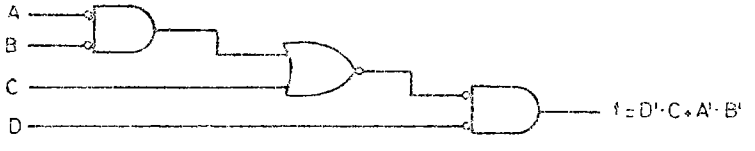


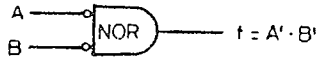
Fig. 3-22. Logică de tip NOR/NOR.

3.14. MIXAREA TIPURILOR DE LOGICA

Ținînd cont de avantajele standardizării este, în general, recomandabil să se folosească structuri de tip NAND/NAND. În majoritatea cazurilor folosirea unor structuri mixte este nerecomandabilă deoarece conduce la structuri

ineficiente. În foarte puține cazuri însă unele scheme pot fi optimizate prin folosirea unui număr limitat de porți NOR (de exemplu, numai porți cu două intrări). În general prin aplicarea acestui procedeu singurul avantaj care se obține este că uneori pot fi eliminate din schemă inversoare. Spre exemplu, dacă dorim să realizăm numai funcția logică AND (și nu AND/OR), este necesar să inversăm ieșirea unei porți NAND. În figura 3-23 este prezentat modul în care utilizarea unei porți NOR poate rezolva problema cu un singur nivel logic.

Fig. 3-23. Utilizarea porților NOR pentru a economisi inversoare.



Deseori apar situații în care pentru decodarea unei configurații binare nu dispunem de variabilele de intrare complementare. Pentru astfel de situații sînt uneori utile structurile de tip NOR/NAND care degenerează într-un NAND (vezi fig. 3-24 *a* și *b*). O altă combinație utilă este reprezentată în figura 3-24 *c*. Această combinație permite obținerea unei funcții AND cu 16 intrări utilizînd două NAND-uri cu 8 intrări.

O altă categorie de porți utile în seria SSI o constituie cele de tip NAND/OR sau AND/OR/INVERT. Deoarece în cazul acestor structuri conexiunile între nivelele logice sînt realizate interior în cursul fabricației lor se poate obține mai multă „logică" raportată la o capsulă de circuit integrat. În mod obișnuit cea mai bună strategie pentru utilizarea acestor circuite speciale (inclusiv NOR) este de a se examina schema logică finală pentru a găsi locurile în care se pot folosi.

Grupurile de 4 sau 2 porți NAND/NAND cu două intrări candidează pentru înlocuirea prin circuite de tip AND/OR. Trebuie avut totuși, în vedere faptul că circuitele de tipul AND/OR cu cîte două intrări, se încapsulează cîte două ; este deci lipsit de sens să utilizăm doar unul din circuite (utilizînd astfel o capsulă!) pentru a înlocui, de exemplu, 3/4 dintr-o capsulă de 4 porți cu două intrări ! Aceeași problemă apare și în cazul circuitelor NOR cu două intrări care sînt încapsulate cîte patru împreună. Este ineficient să utilizăm numai unul sau două dintre acestea.

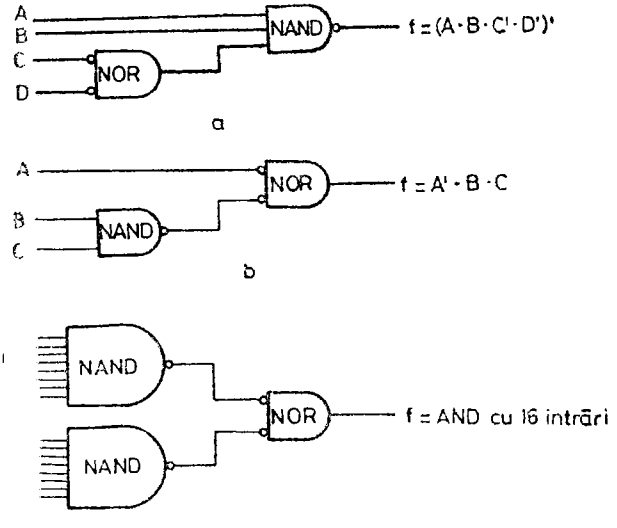
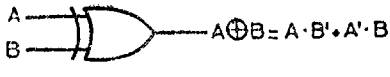


Fig. 3-24. Combinații uzuale de tip NOR/NAND : *(a), (b)* în cazul în care nu dispunem de variabile de intrare negate ; *(c)* pentru construirea unor porți AND cu un număr mare de intrări.

3.15. SAU EXCLUSIV

O altă funcție logică utilă uneori este funcția SAU EXCUSIV (EXCLUSIVE OR, XOR). Funcția XOR este definită în figura 3-25; la ieșire se obține 1 logic dacă *una și numai una* dintre intrări are semnal 1 aplicat. Deși această funcție se poate obține ușor utilizînd trei porți NAND cu două intrări, existența unor cipuri care conțin patru astfel de circuite cu cîte două intrări fiecare face ca utilizarea lor să merite osteneala deoarece poate duce la o economie de cipuri. Deoarece $A \oplus B$ poate fi gîndit ca $A \neq B$, acest circuit este folosit și pentru implementarea funcției de comparare.

Putem să privim acest tip de circuit și ca pe un *„inversor controlabil"*. Vom considera una din intrări ca o intrare de comandă, iar pe cealaltă vom aplica semnalul, ce trebuie sau nu inversat. Dacă intrarea de control este 0 (*fals*) circuitul nu inversează, iar dacă este 1 (*adevărat*) se comportă ca un inversor.



A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 3-25. Funcția SAU-EXCLUSIV (XOR).

3.16. CIRCUITE OR ȘI AND VIRTUALE

Dacă ieșirile porților logice nu sînt active în ambele direcții (numai pentru *L* pe ieșire nu și pentru *H*), acestea se pot conecta împreună pentru a forma noi funcții logice. Spre exemplu, circuitele TTL cu colectorul în gol, generează corect potențialul coborît la ieșire (zero logic), dar necesită o rezistență conectată la sursa de alimentare („pull-up" resistor) pentru a genera nivelul logic înalt. Dacă mai multe astfel de porți au ieșirile conectate prin aceeași rezistență la sursa de alimentare, este suficient ca una singură să genereze un semnal coborît pentru ca la ieșire să obținem zero logic. Avem de a face cu un veritabil OR cu ieșirea activă pe zero, realizat fără a exista fizic o poartă OR. Din acest motiv este denumit OR virtual. În literatură mai apar denumiri ca Implied OR, Collector OR, Bus OR sau Dot OR.

În figura 3-26 *a* este prezentat modul în care, utilizînd două circuite NAND cu colectorul în gol, se poate obține funcția AND/OR/INVERT prin conectarea în paralel a ieșirilor. Dacă $A \cdot B$ sau $C \cdot D$ sînt adevărate, una din cele două porți determină tranziția în zero a ieșirii. Circuitul OR apare în desen figurat punctat pentru a clarifica atît efectul de poartă introdus de conectarea circuitelor cu colectorul în gol cît și faptul că nu există fizic o poartă. Pentru porțile TTL sau **CMOS (complementary metal-oxide-semiconductor)** normale, conectarea împreună a ieșirilor nu este permisă deoarece dacă una din porți generează zero logic pe ieșire iar cealaltă generează unu logic se va obține o stare incertă, fără semnificație logică.

Există și alte familii de circuite logice, cum ar fi de exemplu **ECL (emitter-coupled logic)** sau **DTL (diode-transistor logic)** la care în mod normal ieșirea este unilateral activă. Cablarea în comun a ieșirilor acestor porți se poate face fără restricții. Așa cum un circuit logic normal poate îndeplini

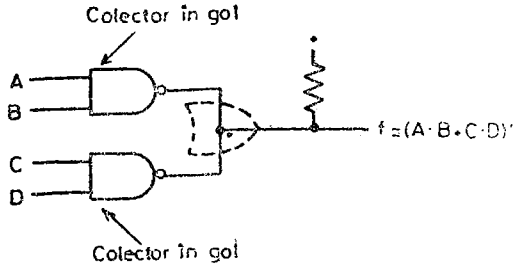
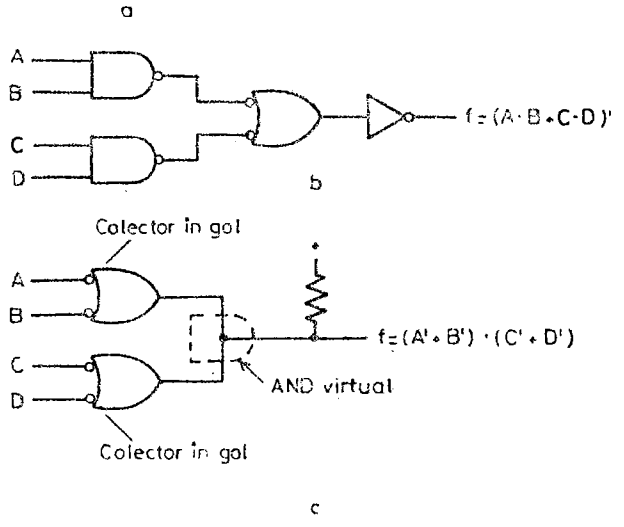


Fig. 3-26. OR virtual : *(a)* logică de tip NANO/OR virtual ; *(b)* aceeași funcție implementată cu logică NAND/NAND ; (c) aceeași funcție în logică negativă.



funcția logică OR sau AND în funcție de semnificația nivelelor electrice de la intrare și circuitul OR virtual poate fi privit ca un AND virtual în aceleași condiții. Deci, de exemplu, considerînd semnalul activ pe nivel coborît circuitul din figura 3-26 *a* poate fi redesenat ca în figura 3-26 *c*, realizînd funcția AND virtual.

3.17. CONECTAREA PE BUS

Deoarece porțile virtuale au un număr nedefinit de intrări ele sînt folosite pentru realizarea *bus*-urilor (magistralelor), contîndu-se pe proprietatea lor de a putea fi expandate comod. Spre exemplu, dacă avem semnale care trebuie să vină de la un număr oarecare de module se poate folosi un singur fir pentru a realiza această conexiune. Pe bus se formează un OR virtual deoarece oricare modul își poate introduce semnalul pe bus în mod independent.

O serie întreagă de aplicații presupun ca un singur modul să genereze într-un interval de timp dat semnal pe bus. În aceste cazuri devin deosebit de utile, în special datorită vitezei mari de transfer ce se poate obține, circuitele „**tristate**". Dacă circuitul tristate este activat ieșirea va genera semnal bidirecțional atît pentru nivelul coborît cît și pentru cel înalt. Dacă este inactiv poate fi privit ca un circuit deconectat de pe bus. Deci acest circuit posedă trei stări : nivel ridicat, nivel coborît și deconectat.

EXERCIȚII

1. Construiți tabelul de adevăr al funcției realizate de circuitele prezentate în figurile 3-4 *a* și *b* cunoscînd faptul că reprezintă funcții identice. Dacă vor apare diferențe, acest fapt va afecta funcționarea afișajului cu șapte segmente ?
2. (*a*) Construiți tabelul de adevăr similar celui din figura 3-2 *d* pentru un decodor de șapte segmente atacat cu semnale codate **biquinar.**
   (*b*) Simplificați cele șapte funcții folosind diagrama Veitch.
   (c) Implementați ecuațiile simplificate utilizînd porți AND și OR. Numărați capsulele integrate utilizate (capsule cu 14 terminale).
   (*d*) Implementați aceleași funcții utilizînd logica de tip NAND/NAND.
   (*e*) Realizați aceeași operație utilizînd logica de tip NOR/NOR.
   (*f*) Implementați cu logica de tip NAND/NAND folosind factorizarea de cîte ori este posibil.
3. (*a*) Construiți tabelul de adevăr pentru circuitul ce comandă aprinderea și stingerea independentă a unui bec din două locuri distincte (două comutatoare A și B distincte, de exemplu A este la sfîrșitul unei scări, iar B la începutul ei).
   (*b*) Extrageți ecuația logică pentru semnalul care comandă lampa.
   (*c*) Implementați circuitul folosind porți de tip AND și OR.
   (*d*) Implementați circuitul utilizînd logica de tip NAND/NAND.
   (*e*) Implementați circuitul utilizînd porțile speciale descrise la sfîrșitul acestui capitole.
4. (*a*) Scrieți ecuația logică pentru un sistem de alarmă împotriva hoților ce are următoarele intrări :

   A = comutatorul de activare,
   F = ușa din față închisă,
   B = ușa din spate închisă,
   W = fereastra închisă,
   M = mișcare detectată în cameră.

   (*b*) Implementați funcția logică utilizînd porți AND și OR.
   (*c*) Implementați funcție logică folosind rețele NAND/NAND.
   (*d*) Implementați funcția utilizînd logica de tipul NOR/NOR.
5. (*a*) Scrieți ecuațiile logice (eventual utilizînd și un tabel de adevăr) pentru un circuit de codare cu 10 intrări și 4 ieșiri, care generează la ieșire codul BCD cînd una din intrări ($I_0 \ldots I_9$) este activată (pe 1 logic). Menționăm că în același interval de timp este activă numai o singură intrare.
   (*b*) Implementați cu porți AND și OR circuitul și numărați capsulele folosite.
   (*c*) Implementați circuitul utilizînd logica NAND/NAND.
   (*d*) Implementați circuitul utilizînd structuri NOR/NOR.
6. (*a*) Construiți diagrama Veitch pentru funcția

$$f = A \cdot B \cdot C + B \cdot C' \cdot D + A' \cdot B \cdot C$$

   (*b*) Scrieți ecuația simplificată a funcției f.
   (*c*) Implementați funcția f folosind logica NAND/NAND.
7. (*a*) Scrieți ecuațiile simplificate ale funcțiilor reprezentate în figura 3-27.
   (*b*) Implementați funcțiile simplificate utilizînd circuite AND și OR.
   (*c*) Implementați funcțiile simplificate utilizînd logica de tip NAND/NAND.
   (*d*) Implementați funcțiile simplificate utilizînd logica de tip NOR/NOR.
8. (*a*) Construiți diagrama Veitch pentru următoarele ecuații scrise ca mintermeni de patru variabile :

$$f_1 = m_1 + m_2 + m_8 + m_13$$
$$f_2 = m_0 + m_1 + m_2 + m_0 + m_8$$
$$f_3 = m_5 + m_7 + m_9 + m_{10} + m_{14}$$
$$f_4 = m_0 + m_2 + m_4$$

   (*b*) Scrieți tabelul de adevăr pentru cele patru funcții.
   (*c*) Extrageți din diagramă cele patru ecuații logice simplificate.
   (*d*) Implementați-le folosind porți NAND.
9. (*a*) Construiți diagramele Veitch pentru cele patru funcții logice din tabelul reprezentat în figura 3-28.
   (*b*) Scrieți ecuațiile simplificate.
   (*c*) Implementați ecuațiile folosind logica de tip AND/OR și numărați capsulele integrate folosite.
   (*d*) Implementați ecuațiile folosind circuite NAND.

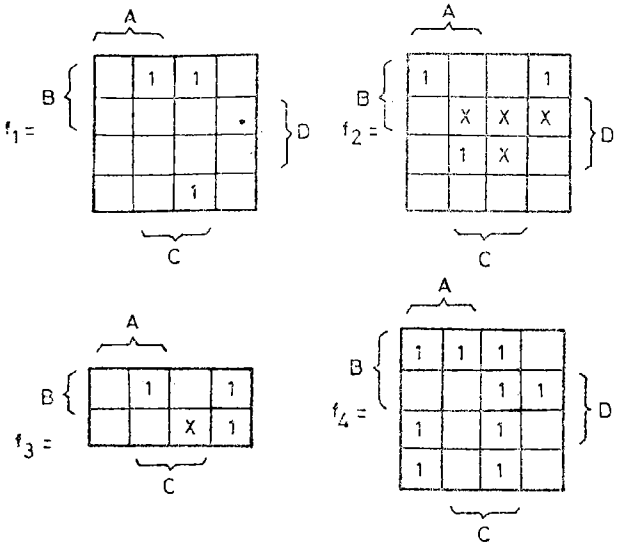


Fig. 3-27,

Intrare				Ieşiri			
A	B	C	D	F_1	F_2	F_3	F_4
0	0	0	0	1	0	0	1
1	0	0	0	0	1	0	1
1	1	0	0	0	1	1	1
1	1	1	0	1	1	0	1
1	1	1	1	1	1	1	0
0	1	1	1	0	0	0	0
0	0	1	1	0	0	1	1
0	0	0	1	0	0	1	0

Fig.3-28,

10. (*a*) Construiţi tabelul de adevăr pentru funcţia majoritate cu patru variabile. Această funcţie este adevărată dacă trei sau mai multe variabile de intrare sînt adevărate.
(*b*) Construiţi diagrama Veitch a funcţiei.
(*c*) Implementaţi cu porţi NAND.
11. (*a*) Scrieţi ecuaţiile simplificate ale funcţiilor din figura 3-29.
(*b*) Implementaţi-le cu porţi NAND.
(*c*) Număraţi capsulele integrate folosite.
12. (*a*) Scrieţi tabelul de adevăr al funcţiei care este adevărată atunci cînd două sau mai multe (din patru) variabile de intrare sînt adevărate.
(*b*) Construiţi diagramele funcţiei.
(*c*) Realizaţi implementarea cu porţi NAND.
(*d*) Număraţi capsulele integrate utilizate.
13. (*a*) Simplificaţi următoarea ecuaţie:

$$f = SH1 \cdot WH2 \cdot AM' + AM \cdot WO' + WH1' \cdot WO + \\ + WH2' \cdot WO + WH1 \cdot WH2' \cdot AM'$$

(Cele patru variabile sînt *WH1*, *WH2*, *AM* şi *WO*).
(*b*) Implementaţi funcţia cu porţi AND şi OR.
14. (*a*) Construiţi diagrama Veitch pentru următoarea funcţie:

$$f = A \cdot (B + C + D) + A \cdot B' \cdot C + A \cdot C'$$

(*b*) Scrieţi ecuaţia simplificată.
(*c*) Implementaţi cu porţi NAND.

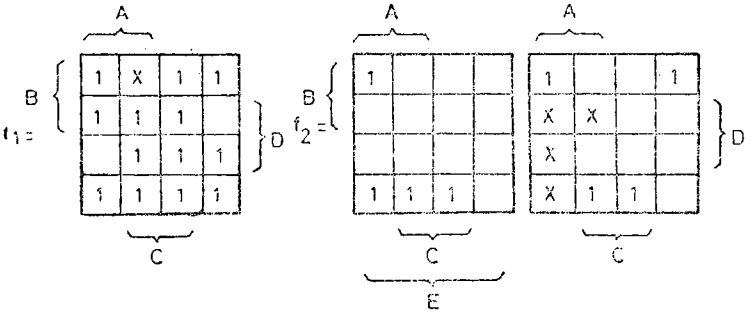


Fig. 3-29,

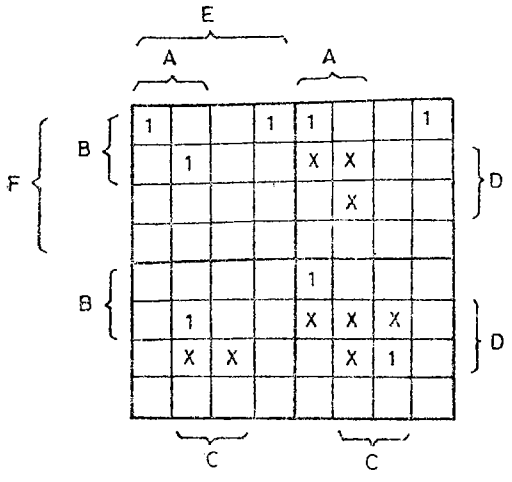


Fig. 3-30.

15. (*a*) Scrieţi forma simplificată a funcţiei reprezentate în figura 3-30.
(*b*) Implementaţi cu logica AND/OR şi număraţi capsulele integrate folosite.
(*c*) Implementaţi cu logica NAND/NAND
(*d*) Implementaţi cu logica NOR/NOR.

16. (*a*) Construiţi tabelul de adevăr al funcţiei de patru variabile (*A*, *B*, *C*, *D*) care este adevărată atunci cînd două variabile adiacente (*A* şi *B*', *B* şi *C*, *C* şi *D*, *D* şi *A*) sînt adevărate. Se consideră la intrare numai cele 10 combinaţii corespunzătoare codului zecimal codat binar; celelalte vor fi interpretate ca nesemnificative, redundante.
(*b*) Diagrama funcţiei.
(*c*) Scrieţi forma simplificată a funcţiei.
(*d*) Implementaţi circuitul folosind logica NAND/NAND.

17. (*a*) Implementaţi cu porţi NAND funcţiile afişajului cu şapte segmente, definite în figura 3-8.
(*b*) Rescrieţi ecuaţiile din figura 3-8, factorizînd termenii comuni.
(*c*) Implementaţi ecuaţiile logice factorizate folosind porţi NAND.

18. (*a*) Implementaţi ecuaţia 3-13 cu logica NOR/NOR.
(*b*) Refaceţi punctul anterior cu logica NAND/NAND.

19. Enumeraţi sub formă de desene toate identităţile logice din tabelul 3-1 (similar modului în care în figura 3-17 s-a explicat a 15-a identitate).

20. (*a*) Implementaţi funcţia XOR folosind porţi NAND.
(*b*) Presupunînd că nu dispunem de variabilele de intrare negate, să se implementeze funcţia XOR folosind patru NAND-uri cu două intrări.

21. (*a*) Construiţi tabelul de adevăr pentru funcţia XOR de patru variabile.
(*b*) Diagrama funcţiei.
(*c*) Implementaţi funcţia folosind porţi NAND.
(*d*) Implementaţi funcţia folosind circuite NAND cu colectorul în gol şi inversînd ieşirea OR-ului virtual obţinut.
(*e*) Implementaţi funcţia utilizînd numai trei circuite XOR cu două intrări, considerînd că nu sînt disponibile variabilele de intrare inversate.

22. (*a*) Construiţi tabelul de adevăr pentru funcţia ce ia valoarea logică de adevăr atunci cînd cele două numere de doi biţi de la intrare sînt egale ($AB = CD$).
(*b*) Scrieţi ecuaţiile logice ale funcţiei.
(*c*) Implementaţi funcţia folosind porţi NAND.
(*d*) Număraţi capsulele cu circuite integrate necesare.

BIBLIOGRAFIE

Calebotta, Steven, „CMOS-The Nearly Ideal Logic Family", *Digital Design*, July 1973, pag. 34—44.

Froeck, Donald, „Ring Map Minimizes Logic Circuit", *Electronic Design*, August 17, 1972, pag. 80—81.

Phister, M., *Logical Design of Digital Computers*, Wiley, New York, 1958, Capitolul 31, „Boolean Algebra", şi Capitolul 4, „The Simplification of Boolean Functions".

Richards, R. K., *Digital Design*, Wiley, New York, 1971.

Su and Nam, „Computer Aided Synthesis of Multiple Output, Multi-Level NAND Networks With Fan-In and Fan-Out Constraints", *IEEE Transactions on Computers*, December 1971, pag. 1445—1455.

Texas Instruments Staff, *Designing With TTL Integrated Circuits*, McGraw-Hill, New York, 1971.

Texas Instruments Staff, *The Integrated Circuits Catalog*, Texas Instruments Inc., Dallas, Tex., 1972.

Veitch, E. W., „A Chart Method for Simplifying Truth Functions", *American Mathematical Monthly* 59, 1952, pag. 521—531.

4

LOGICA COMBINAȚIONALĂ II

Proiectarea cu circuite MSI și LSI

4.1. PROIECTAREA CU CIRCUITE MSI ȘI LSI

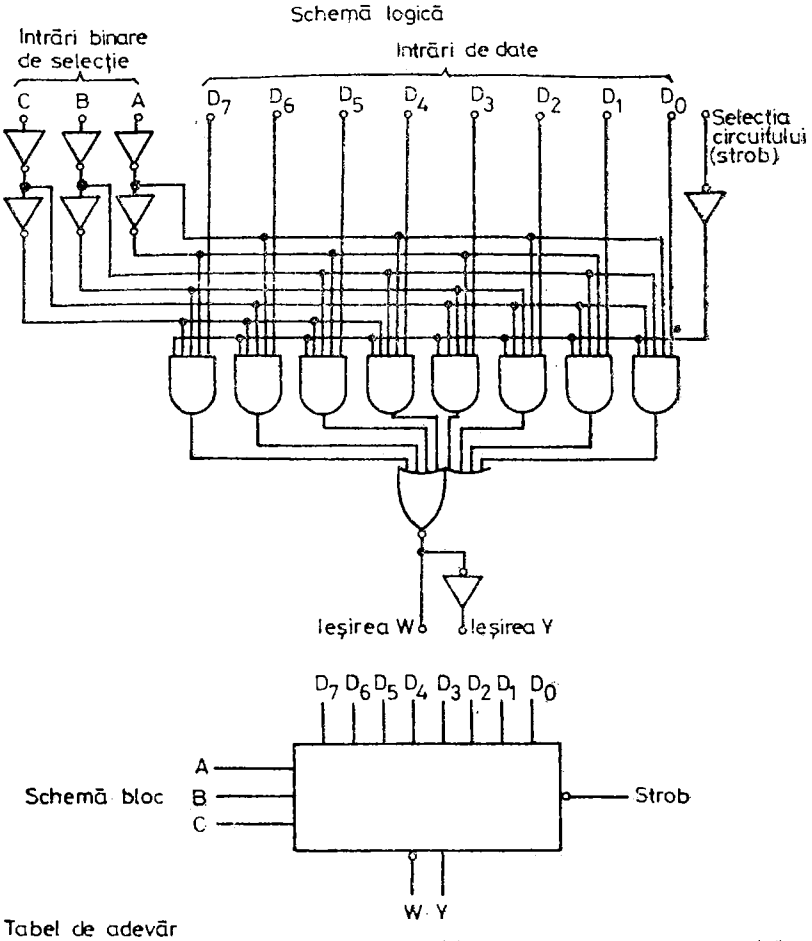
În capitolul 3 am dezvoltat acele tehnici tradiționale ale proiectării logice care par a fi încă utile. Înainte de apariția circuitelor integrate proiectarea logică presupunea controlul structurii pînă la nivel de poartă elementară (proiectantul opera cu cîte o poartă o dată). Problemele de proiectare logică pe care le avem de rezolvat în etapa actuală sînt mult prea complexe pentru a ne mai putea permite o astfel de abordare. În acest sens fabricanții de circuite integrate oferă structuri mai complexe disponibile ca structuri standard și care sînt în mod cert circuite mai „ieftine" (în sensul descris în capitolul 2) decît porțile simple. Din acest motiv — cu toate că trebuie să facem deseori apel la ele — *tehnicile de proiectare prezentate în capitolul anterior pot fi considerate ca o ultimă resursă* fiind utilizate acolo unde circuitele cu un nivel înalt de integare nu pot fi folosite. În mod obișnuit circuitele logice elementare sînt necesare pentru a adapta la aplicație circuitele MSI și LSI standardizate care rareori satisfac cu exactitate cerințele proiectantului.

Vom trece în revistă unele tehnici de utilizare a circuitelor MSI și LSI pentru implementarea ecuațiilor logice, menționînd de la început, că o bună utilizare a acestora poate fi obținută numai la nivelul proiectării de sistem, înainte de a ajunge la detalii cum ar fi ecuațiile logice.

Întotdeauna forma ecuațiilor logice ce dorim să fie implementate cu circuite MSI trebuie să fie corelată cu circuitele MSI *disponibile* pentru a nu risca, în final, implementarea lor tot cu porți discrete.

Din acest motiv, *este esențial să cunoaștem în amănunțime conținutul cataloagelor producătorilor de circuite integrate*. Pentru a obține o proiectare performantă trebuie să cunoaștem bine circuitele MSI și LSI disponibile. Numai după ce am detaliat schema bloc a sistemului, am ales circuitele MSI și LSI necesare, putem demara proiectarea scriind ecuațiile logice.

Scrierea într-o primă fază a proiectării a ecuațiilor logice, (desigur folositoare) tinde să facă obscură funcția care trebuie realizată. Spre exemplu, circuitul pentru comanda afișajului cu șapte segmente, utilizat în mod repetat ca exemplu în capitolul anterior, este disponibil ca circuit integrat MSI standard. Șansa ca să descoperim acest fapt începînd să scriem ecuațiile logice este nulă. Ecuațiile din figura 3-8 nu sugerează în nici un fel existența unui circuit de comandă pentru un afișaj cu șapte segmente. Numai enunțul clar al problemei — la nivelul proiectării de sistem : „circuit de comandă pentru afișaj cu șapte segmente", este necesar și suficient pentru aceasta. Din acest motiv întregul sistem trebuie definit mai întîi sub formă de blocuri MSI și LSI iar momentul în care trecem la scrierea ecuațiilor logice trebuie amînat cît mai mult.



Tabel de adevăr

Intrări											Ieșiri	
C	B	A	Selecție	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	Y	W
X	X	X	1	X	X	X	X	X	X	X	X	0	1
0	0	0	0	0	X	X	X	X	X	X	X	0	1
0	0	0	0	1	X	X	X	X	X	X	X	1	0
0	0	1	0	X	0	X	X	X	X	X	X	0	1
0	0	1	0	X	1	X	X	X	X	X	X	1	0
0	1	0	0	X	X	0	X	X	X	X	X	0	1
0	1	0	0	X	X	1	X	X	X	X	X	1	0
0	1	1	0	X	X	X	0	X	X	X	X	0	1
0	1	1	0	X	X	X	1	X	X	X	X	1	0
1	0	0	0	X	X	X	X	0	X	X	X	0	1
1	0	0	0	X	X	X	X	1	X	X	X	1	0
1	0	1	0	X	X	X	X	X	0	X	X	0	1
1	0	1	0	X	X	X	X	X	1	X	X	1	0
1	1	0	0	X	X	X	X	X	X	0	X	0	1
1	1	0	0	X	X	X	X	X	X	1	X	1	0
1	1	1	0	X	X	X	X	X	X	X	0	0	1
1	1	1	0	X	X	X	X	X	X	X	1	1	0

Utilizarea simbolului X indică faptul că intrarea este nesemnificativă

Fig. 4-1. Multiplexor/selector de opt intrări (74151).

4.2. MULTIPLEXORUL (SELECTORUL) DIGITAL

Circuitul de multiplexare este unul din cele mai importante circuite MSI. Este denumit uneori „selector" deoarece este utilizat și ca un comutator de selectare a anumitor căi. De exemplu, un multiplexor cu opt intrări selectează și transferă la ieșire una din cele opt intrări, în funcție de un cod de trei biți. În figura 4-1 este reprezentată schema logică a unui astfel de circuit și tabelul său de adevăr. Pe lîngă acest tip de multiplexor există și multiplexoare cu două intrări, cîte patru într-o capsulă, multiplexoare de 4 căi, două într-o capsulă și multiplexoare de 16 căi, unul într-o capsulă cu 24 de terminale.

Putem sesiza acum avantajul de a cunoaște circuitele integrate standard și de a le utiliza în proiectare. De exemplu dacă sistemul pe care-l proiectăm presupune accesul din patru surse a unor cuvinte de opt biți, în schema bloc vom introduce patru capsule ce conțin cîte două multiplexoare de patru căi. Dacă dorim să scriem ecuațiile logice, va trebui să scriem de opt ori ecuații de forma :

$$Y = I_0 \cdot A' \cdot B' + I_1 \cdot A' \cdot B + I_2 \cdot A \cdot B' + I_3 \cdot A \cdot B \qquad (4.1)$$

Dacă vom implementa aceste ecuații cu porți, vom utiliza pe puțin 10 capsule integrate. Va fi mult mai greu să recunoaștem după forma ecuațiilor că circuitul rezultant funcționează ca multiplexor decît după sugestia pe care ne-o oferă expresia : „de la una din cele patru surse".

O bună cunoaștere a circuitelor MSI ne asigură posibilitatea de a proceda inteligent și eficient la nivelul sistemului. Spre exemplu, știind că este la fel de ieftin să selectăm o intrare din trei sau o intrare din patru, vom putea introduce facilități suplimentare într-un sistem fără nici un cost suplimentar. Din acest motiv „specificațiile de proiectare" inițiale vor trebui să includă schema bloc detaliată la nivel de circuit MSI. Zonele realizate cu porți discrete pot fi, pentru început, incluse în blocuri marcate prin eticheta generală „logică de comandă".

4.2.1. Folosirea multiplexorului pentru implementarea funcțiilor logice

Pe lîngă rolul de a selecta diverse surse de semnal, multiplexorul poate fi folosit și pentru generarea și simplificarea funcțiilor logice. Dacă vom analiza ecuația unui multiplexor cu patru intrări, spre exemplu,

$$Y = I_0 \cdot A' \cdot B' + I_1 \cdot A' \cdot B + I_2 \cdot A \cdot B' + I_3 \cdot A \cdot B \qquad (4\text{-}2)$$

vom remarca faptul că avem intrări separate pentru fiecare din cele patru combinații ale intrărilor A și B. Deci vom putea „da în factor" pe A și B dintr-o funcție cu n variabile, menținînd patru funcții de n-2 variabile pentru a le conecta la intrările ($I_0, \ldots, I_3$) ale multiplexorului. Similar un multiplexor cu opt intrări poate elimina trei variabile iar unul cu 16 intrări patru variabile, ce sînt conectate la intrările de adresare (selecție).

Dacă avem o funcție cu patru variabile vom elimina trei prin aplicarea lor la intrările de selecție și vom conecta la celelalte intrări opt funcții de numai *o variabilă*. Singurele funcții posibile de o variabilă C sînt : C, C',

1, 0. Întrucît acestea sînt disponibile fără circuite suplimentare *vom putea implementa orice funcție cu patru variabile folosind un singur multiplexor de opt căi.*

Ca exemplu să considerăm funcția :

$$f_1 = A' \cdot B' \cdot C' \cdot D' + A' \cdot B' \cdot C \cdot D + A' \cdot B \cdot C' \cdot D + A' \cdot B \cdot C' \cdot D' + A \cdot B' \cdot C' \cdot D + A \cdot B' \cdot C \cdot D' + A \cdot B \cdot C' \cdot D' + A \cdot B \cdot C' \cdot D \quad (4\text{-}3)$$

Vom folosi un multiplexor de opt căi și vom aplica variabilele A, B și C pe intrările de adresare (selecție). Pentru determinarea funcțiilor aplicate pe intrările $I_0 \ldots I_7$ vom întocmi un tabel de forma celui din Tabelul 4-1. Coloana

Tabelul 4-1. **Intrările multiplexorului ce generează F_1**

Intrarea	Adresa	Alte variabile (Reziduuri)
I_0	$A' \cdot B' \cdot C'$	D'
I_1	$A' \cdot B' \cdot C$	D
I_2	$A' \cdot B \cdot C'$	$D + D'$
I_3	$A' \cdot B \cdot C$	
I_4	$A \cdot B' \cdot C'$	D
I_5	$A \cdot B' \cdot C$	D'
I_6	$A \cdot B \cdot C'$	$D' + D$
I_7	$A \cdot B \cdot C$	

„Alte variabile" din tabel o putem construi direct pornind de la ecuația logică (4-3), dînd în factor pe rînd $A' \cdot B' \cdot C'$, $A' \cdot B' \cdot C$, pînă la $A \cdot BC$ și obținînd D', D, $D + D'$ și 0.

Modul de conectare a intrărilor $I_0, \ldots I_7$ ale multiplexorului se obține direct din tabel, deoarece fiecare linie corespunde unei intrări (vezi fig. 4-2).

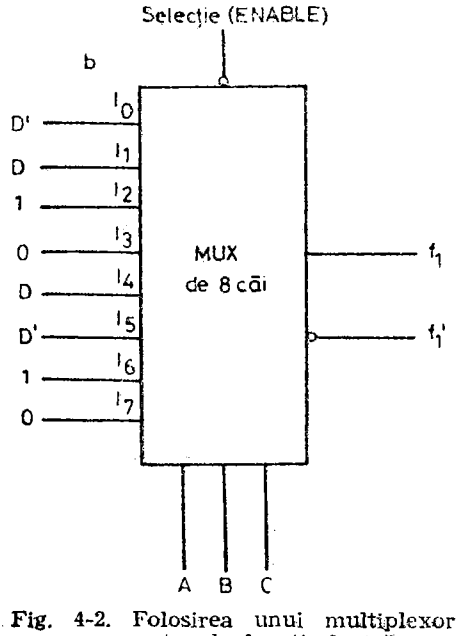


Fig. 4-2. Folosirea unui multiplexor ca generator de funcție logică.

Deoarece $D + D' = 1$ vom conecta la nivelul logic înalt (plusul sursei de alimentare) intrările I_2 și I_6. Vom conecta la masă (zero logic) intrările I_3 și I_7 deoarece pe liniile respective este 0. Circuitul rezultat este dat în figura 4-2. Remarcăm faptul că *o singură capsulă cu un multiplexor generează o funcție logică care ar fi necesitat patru capsule integrate pentru a fi implementată cu porți discrete,* chiar după simplificarea ecuației logice. În plus, intrarea de selecție a întregului multiplexor (enable) poate fi folosită cu un AND final cu întreaga funcție. Circuitul dispune de asemenea și de o ieșire inversată, generînd deci f_1 și f_1'. De asemenea numai una din variabile, D, este necesar să fie disponibilă și sub formă complementată.

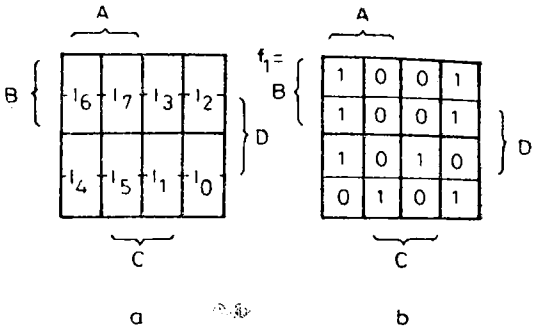


Fig. 4-3. Metoda diagramei Veitch : *(a)* corespondența cu intrările multiplexorului ; *(b)* ecuația 3-3.

În figura 4-3 este prezentată procedura de implementare cu multiplexor a unei funcții pornind direct de la diagrama Veitch și nu de la ecuația logică. Numerele din figura 4-3 *a* corespund intrărilor multiplexorului. Configurațiile de 1 din fiecare dintre cele opt dreptunghiuri indică modul de conectare a intrărilor multiplexorului, la D, D', 1 sau 0.

Faptul că, în cazul acestui exemplu, prin folosirea multiplexorului, au fost economisite trei capsule, constituie o situație particulară. Trebuie să reținem faptul că există multe funcții de patru variabile ce pot fi comod implementate — folosind o capsulă de circuite integrate sau chiar mai puțin — prin utilizarea de porți. Utilizarea multiplexorului este indicată numai în cazul funcțiilor mai dificile. De asemenea, prin acest procedeu nu putem utiliza în comun pentru mai multe funcții aceeași termeni. Deoarece în cazul în care trebuie să generăm mai multe funcții de aceleași variabile este posibil ca numărul termenilor utilizabili în comun să fie considerabil ; ca urmare nu trebuie să renunțăm cu ușurință la abordarea cu porți.

În general nu putem alege soluția cu multiplexoare fără a examina și soluția cu porți discrete avînd însă în minte și faptul că *oricare* din funcțiile indicate în Tabelul 4-2 se poate implementa cu o singură capsulă de circuit integrat. Găsim astfel numărul maxim de capsule cu porți discrete pe care trebuie să le folosim pentru a implementa funcția dată.

Tabelul 4-2. **Funcții ce necesită numai un singur multiplexor**

Funcția	Tipul de multiplexor
Oricare patru funcții de aceleași două variabile	Cuadruplu de două intrări
Oricare două funcții de aceleași trei variabile	Dual de patru intrări
Orice funcție de patru variabile	De opt intrări
Orice funcție de cinci variabile	16 intrări (sau două de opt intrări)

4.2.2. Simplificarea funcțiilor implementate cu multiplexoare

Simplificarea funcțiilor logice în cazul implementării cu multiplexoare se reduce, în principal, la conectarea unui număr cît mai mare de variabile la intrările de adresă. Pentru implementarea „reziduurilor" ce se conectează la celelalte intrări sînt folosite porți logice elementare.

Minimizarea acestor reziduuri funcționale se poate face printr-o alegere adecvată a variabilelor conectate la intrările de adresare, în așa fel încît aceste funcții să se reducă la 1 logic sau la funcții ce apar de mai multe ori. Deoarece funcțiile reziduale pot fi utilizate în generarea mai multor funcții, vom alege ca variabile aplicate pe intrările de adresare ale multiplexoarelor pe cele ce apar în cît mai multe funcții logice.

Vom folosi pentru exemplificare tot afișajul cu șapte segmente. Cele șapte funcții de patru variabile sînt definite de relațiile 3-6 pînă la 3-12 din Capitolul 3. Desigur, putem implementa aceste funcții cu șapte multiplexoare de opt căi în modalitatea descrisă anterior. Pe de altă parte funcțiile simplificate din figura 3-8 pot fi obținute utilizînd $5^1/_4$ capsule cu porți logice uzuale (nouă cu 2 intrări, trei cu 3 intrări și patru cu 4 intrări). Soluția cu șapte multiplexoare nu se mai justifică în acest caz.

Dacă vom încerca să utilizăm multiplexoare de patru căi într-o variantă *simplificată*, vom putea obține o reducere semnificativă a numărului de capsule integrate. Pentru a utiliza cît mai mult în comun funcțiile reziduale vom „da în factor" aceleași două variabile de la toate cele șapte funcții. Vom alege cele două variabile pornind de la diagramele din figura 3-8. Pot fi obținute diverse subdiviziuni ale diagramelor Veitch, pentru a fi realizată asocierea cu intrările multiplexoarelor de patru căi, în funcție de cele două variabile alese pentru conectarea la adresele multiplexoarelor (conform figurii 4-4). Toate subdiviziunile obținute pot fi gîndite ca mici diagrame Veitch de două variabile pentru funcțiile reziduale. Dacă vom analiza diagramele din figura 3-8 în funcție de subdiviziunile posibile din figura 4-4, vom ajunge la concluzia că cea mai bună alegere pentru eliminarea variabilelor o constituie B și D. În acest caz fiecare funcție reziduală este reprezentată în diagrama

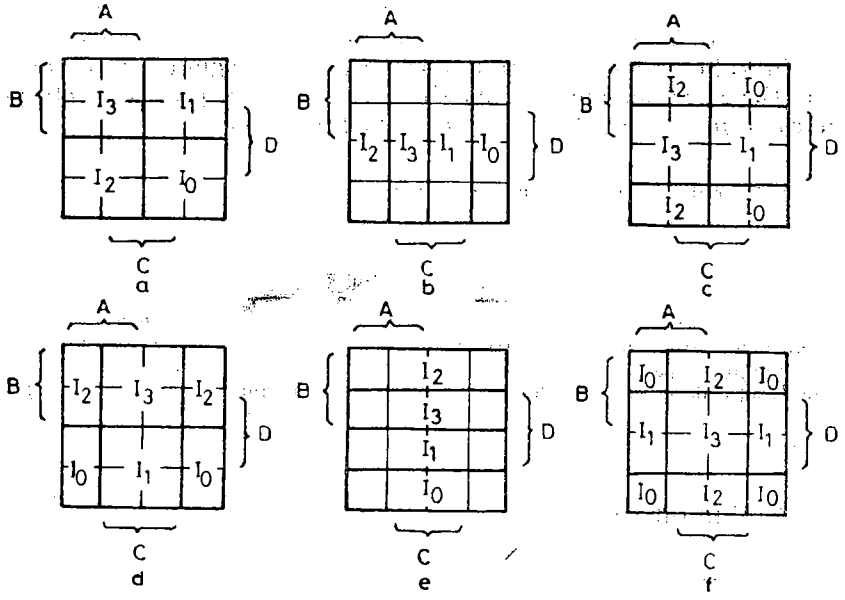
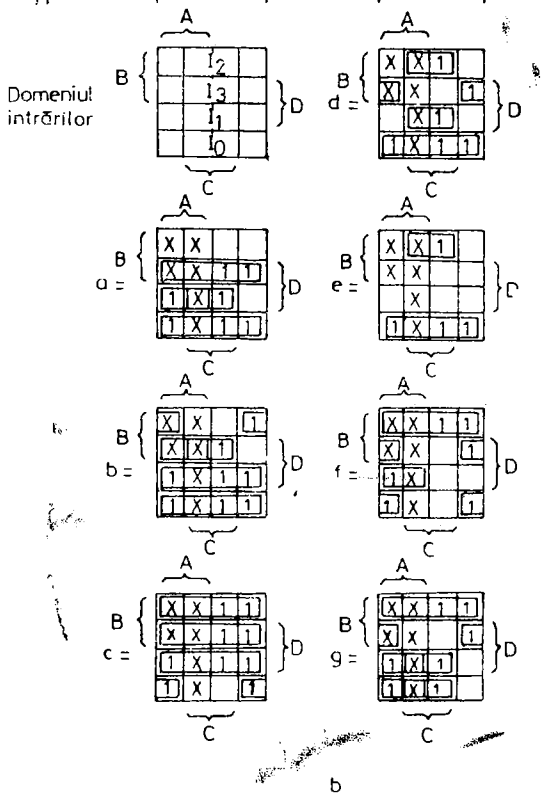


Fig. 4-4. Șase căi posibile de simplificare a unei funcții cu două variabile (I_n = numărul intrării multiplexorului); *(a)* A și B; *(b)* A și C; *(c)* A și D; *(d)* B și C; *(e)* B și D; *(f)* C și D.

Intrare	MUX ADR	a	b	c	d	e	f	g
I_0	B'·D'	1	1	C'	1	1	C	A • C
I_1	B'·D	A • C	1	1	C	0	A	A • C
I_2	B·D'	0	C'	1	C	C	1	1
I_3	B·D	1	A • C	1	C'	0	C'	C'

a



b

Fig. 4-5. *(a)* **Funcţii reziduale.** *(b)* **Diagramele Veitch sînt folosite pentru definirea funcţiilor.**

Veitch printr-o bară orizontală. Deoarece fiecare din aceste funcţii conţine unul sau două X-uri (termeni reziduali) există mari şanse ca forma lor să fie foarte simplă.

În această etapă vom construi tabelul de adevăr pentru funcţiile căutate direct din diagramele Veitch (vezi fig. 4-5*a*). Diagramele Veitch din figura 3-8 sînt redesenate, cu încercuirile cerute de această metodă, în figura 4-5*b*. Funcţiile reziduale sînt citite direct din liniile diagramei Veitch — (ca şi cum fiecare linie ar fi o diagramă Veitch de două variabile). Ca în orice problemă de funcţii multiple vom căuta să evidenţiem cît mai mulţi termeni comuni.

Remarcăm faptul că singura funcţie reziduală netrivială este $(A + C)$, ce poate fi implementată cu o poartă NAND. Forma finală a circuitului este reprezentată în figura 4-6. Deoarece multiplexoarele de patru căi sînt încapsulate cîte două, pentru a nu rămîne unul nefolosit, vom implementa funcţia *c*

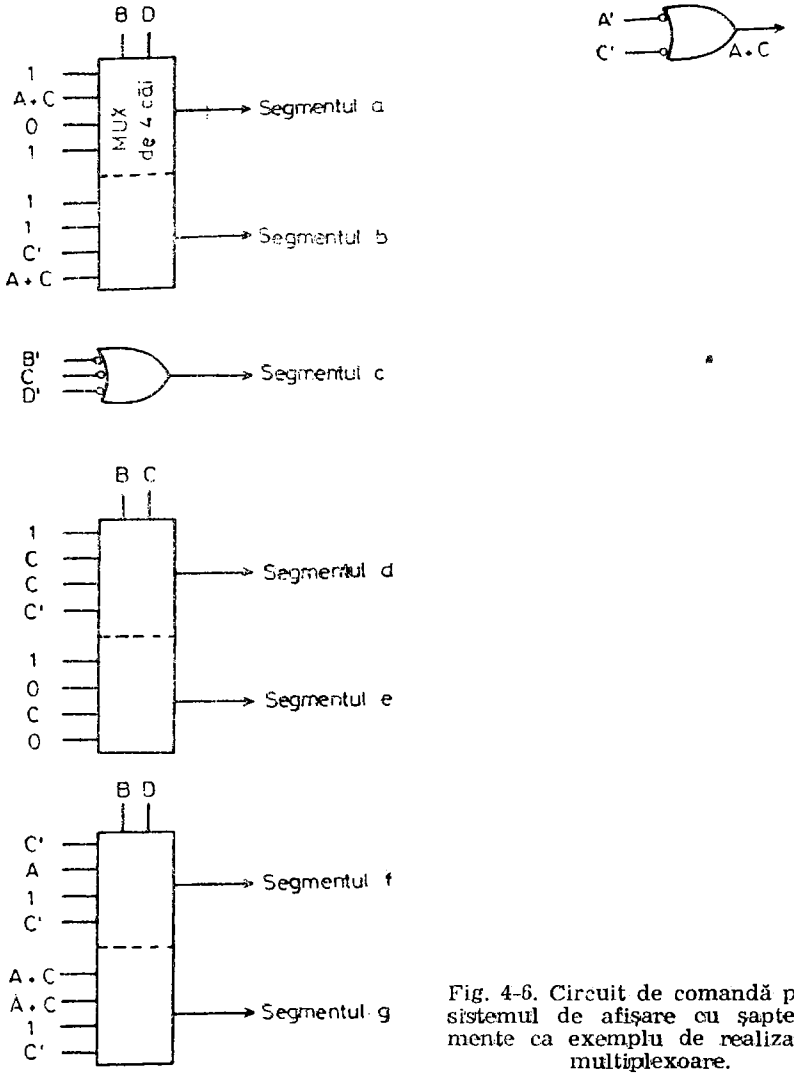


Fig. 4-6. Circuit de comandă pentru sistemul de afişare cu şapte segmente ca exemplu de realizare cu multiplexoare.

ca o poartă cu trei intrări. În final întreaga schemă conţine $3 + 1/3 + 1/4 = 3\ 7/12$ capsule integrate, mult mai puţine decît necesită soluţia convenţională cu porţi (5 1/4 capsule).

4.2.3. Simplificarea funcţiilor cu multe variabile

Metoda prezentată în exemplul precedent este de asemenea utilizabilă pentru simplificarea implementării funcţiilor cu un număr mare de variabile. Aşa cum am văzut în Capitolul 3 (vezi fig. 3-13) metoda diagramei devine greu de folosit pentru funcţii cu mai mult de patru variabile şi imposibil de utilizat pentru mai mult de şase. Utilizarea multiplexoarelor face ca amploarea problemei să scadă simţitor. Spre exemplu, utilizînd multiplexorul

de 16 căi pentru a „înlătura" patru variabile, lucrul cu o funcție de opt variabile de intrare se reduce la problema manipulării unor funcții cu patru intrări. Pe lîngă faptul că putem găsi *mult mai ușor* soluția, și numărul de circuite integrate utilizate scade simțitor. Desigur, multiplexoarele pot fi utilizate și pentru funcțiile reziduale; în cazul funcțiilor deosebit de complicate se poate folosi un al doilea nivel de multiplexoare pentru realizarea funcțiilor reziduale.

4.2.4. Determinarea numărului maxim absolut de capsule

În exemplele anterioare funcțiile reziduale cu două variabile au fost soluționate cu cîteva porți logice. Vom pune problema calculării numărului maxim de porți ce ar putea fi utilizate pentru implementarea într-un caz particular *limită, cel mai nefavorabil.* Există 16 funcții de două variabile. Șase din acestea sînt triviale : A, A', B, B', 1, și 0. Restul sînt : $A' \cdot B'$, $A' \cdot B$, $A \cdot B'$, $A \cdot B$ și $(A' \cdot B) + (A \cdot B')$ (SAU EXCLUSIV = XOR), la care se adaugă complementele lor. Toate aceste 10 funcții pot fi implementate folosind 5 circuite NAND cu două intrări și cinci inversoare, deci un total de 1 1/4 + 5/6 = 2 1/12 capsule integrate.

Desigur, în majoritatea aplicațiilor concrete nu vor fi utilizate *toate* aceste funcții, dar este important de marcat această *limită superioară absolută.* Vom putea, deci, implementa *orice* funcție cu șase variabile de intrare, utilizînd un multiplexor de 16 căi plus maximum 2 1/12 capsule integrate cu porți. Acest mod de abordare trebuie luat în considerație ori de cîte ori vom avea de rezolvat problema unei funcții de șase variabile ce presupune pentru a fi implementată prin metoda tradițională mai mult de trei capsule integrate cu porți.

4.2.5. Extinderea multiplexorului

Deși numărul maxim de intrări a multiplexoarelor integrate standard este de 16 (într-o capsulă cu 24 de terminale) putem construi multiplexoare cu dimensiuni oricît de mari interconectînd mai multe astfel de circuite în rețele arborescente. Un exemplu este dat în figura 4-7, unde este construit un multiplexor cu 64 de intrări folosind nouă multiplexoare cu 8 intrări. Fiecare multiplexor de pe primul nivel selectează o intrare din 8 în funcție de biții D, E și F utilizați în comun pentru adresare. Cel de al doilea nivel de multiplexare selectează ieșirile primului nivel în funcție de codul format de A, B și C. *În final rezultă selectarea uneia din cele 64 de intrări* în funcție de codul de șase biți format cu A, B, C, D, E și F.

O configurație de dimensiuni mai mici poate fi obținută prin eliminarea unor multiplexoare de pe primul nivel. *De asemenea arborele poate fi mărit oricît* prin adăugarea unor nivele suplimentare de multiplexoare. Spre exemplu, opt multiplexoare de 64 de biți de tipul aceluia din figura 4-7 pot genera semnal pentru un multiplexor de 8 căi formînd în final un multiplexor cu $8 \times 64 = 512$ intrări.

Facilitatea de a construi multiplexoare de mare capacitate extinde și posibilitatea folosirii lor în implementarea funcțiilor logice cu multe variabile. Spre exemplu, un multiplexor cu 64 de intrări (9 capsule) poate fi folosit

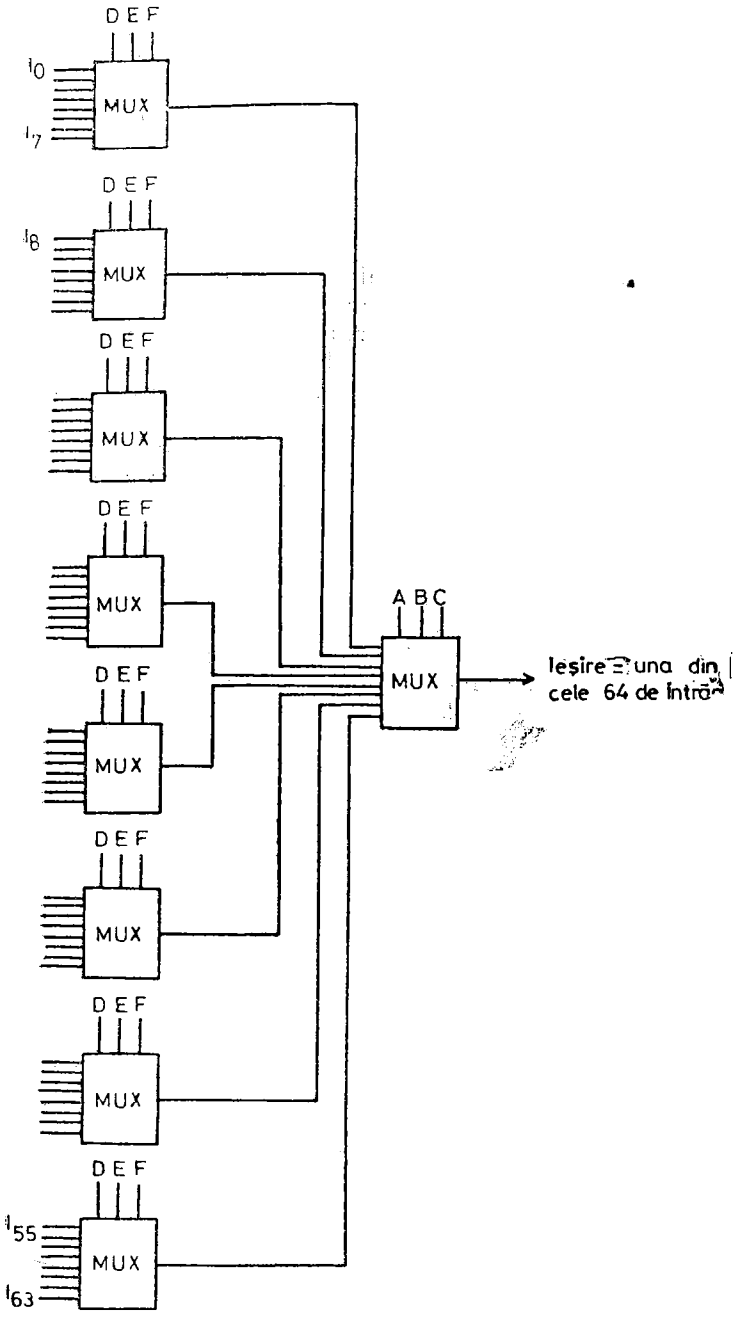


Fig. 4-7. Multiplexor de 64 de căi realizat cu o structură arborescentă.

pentru implementarea unor funcţii de şapte variabile. Utilizînd metoda expusă în secţiunea 4.2.4 se poate genera o funcţie logică cu şapte intrări cu o configuraţie şi mai economică. Deoarece putem genera funcţiile reziduale de 2 variabile cu maximum 2 1/12 cipuri, vom utiliza un multiplexor de 32 de căi (4 + 1 = 5 cipuri) şi *vom implementa orice functie logică cu şapte variabile de intrare* cu maximum 7 1/12 capsule integrate faţă de 9 capsule cît ar presupune utilizarea multiplexorului de 64 de căi. Utilizînd multiplexorul cu 64 de intrări (9 capsule) şi maximum 2 1/12 capsule pentru funcţiile reziduale vom putea implementa *orice funcţie cu opt intrări folosind maximum 11 1/12 capsule integrate.*

Aceste tehnici vor fi folosite numai în cazul în care soluţia nu poate fi obţinută mai simplu cu porţi elementare. Ca şi în cazul exemplului afişajului cu şapte segmente prezentat anterior se pot obţine adeseori rezultate mai bune decît cele indicate în Tabelul 4-3 prin factorizarea anumitor variabile, care „duc" mai multă informaţie.

Tabelul 4-3. **Numărul maxim de capsule de circuite integrate pentru orice funcţie**

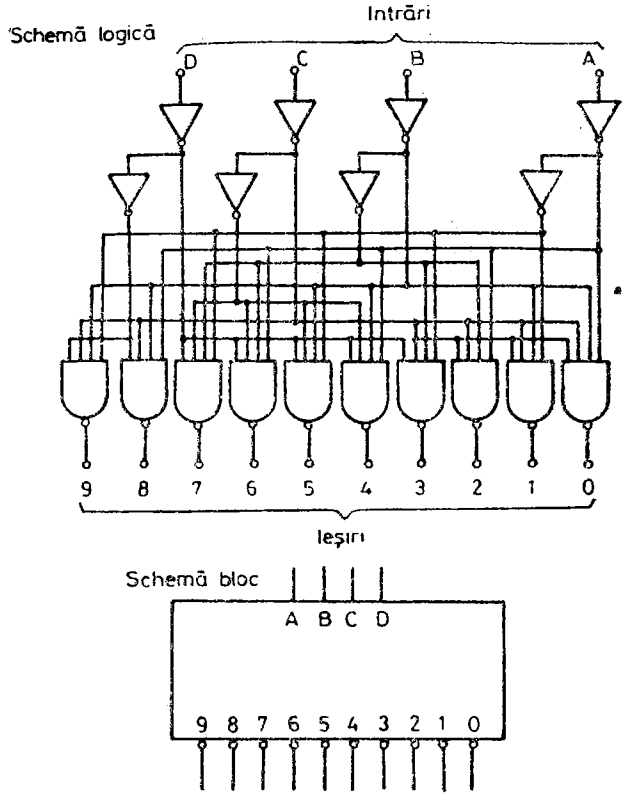
Numărul variabilelor	Numărul maxim al capsulelor integrate	Tipul de multiplexor
2 (patru funcţii)	1	Cuadruplu de două intrări
3 (două funcţii)	1	Dual de patru intrări
4	1	Opt intrări
5	2[a]	16 intrări
6	4 1/12[a]	16 intrări + porţi reziduale
7	6 1/3[a]	32 intrări + porţi reziduale
8	11 1/12	64 intrări + porţi reziduale

a Deoarece multiplexoarele de 16 biţi necesită un spaţiu dublu pe cablajul imprimat ele vor fi considerate ca două cipuri.

De multe ori prin această tehnică obţinem simplificări importante (într-un caz limită, cel mai defavorabil, o funcţie cu 5 variabile se implementează cu *18 cipuri cu porţi*). Cu toate acestea, o serie de funcţii ce apar în aplicaţii concrete sînt de obicei atît de simple încît implementarea cu porţi elementare constituie soluţia cea mai economică. Experienţa este cea care permite identificarea funcţiilor care pot fi implementate cu multiplexoare, prin simpla inspecţie a diagramelor Veitch sau a ecuaţiilor logice.

4.3. DECODIFICATOARE/DEMULTIPLEXOARE

Funcţia inversă multiplexării este demultiplexarea. Circuitele ce realizează această funcţie pot fi utilizate şi ca decodicicatoare. *În cazul unui decodificator semnalele de intrare aplicate căilor de adresare determină tranziţia uneia din ieşiri în 0 logic.* Figura 4-8 defineşte decodificatorul zecimal. Decodificatoarele MSI sînt disponibile şi în alte variante: două decodificatoare cu 4 ieşiri (2 biţi de adresă comuni) sau decodificator de patru biţi (cu 16 ieşiri) în capsulă cu 24 de terminale. Există şi alte tipuri de decodificatoare pentru aplicaţii speciale; decodificator pentru cod de intrare binar în exces trei, decodificator cu ieşire de tensiune ridicată sau curent mare, decodificator pentru afişajele cu şapte segmente etc.



S5442/N7442

Tabelă de adevăr

Intrare în cod binar zecimal(BCD)

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

Ieşire zecimală

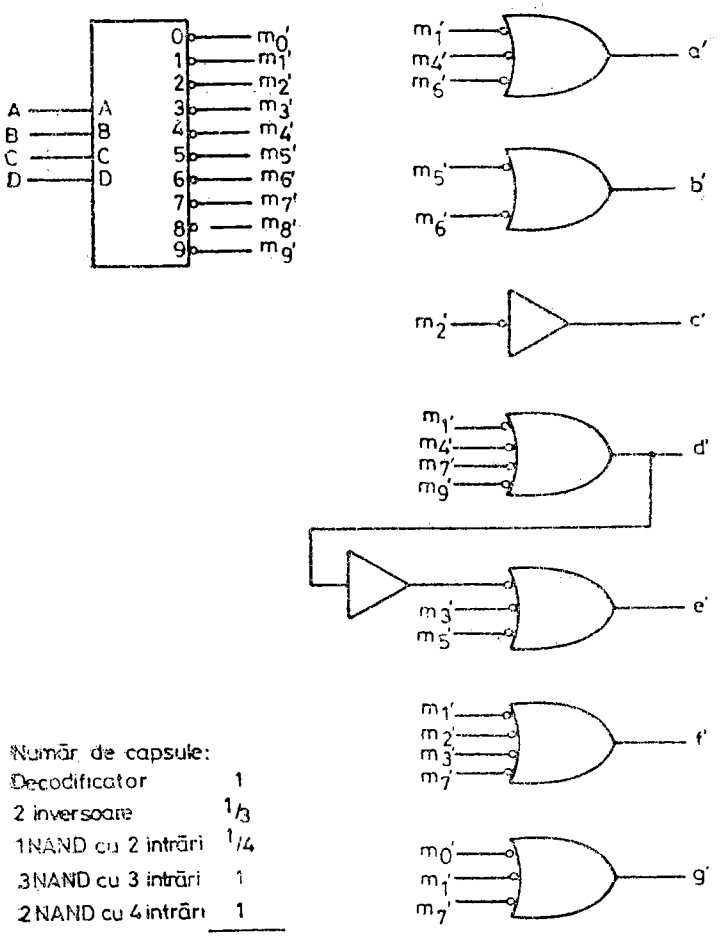
0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1
1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	0	1	1	1
1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

Fig. 4-8. Decodificator zecimal.

Tabelul de adevăr din figura 4-8 arată că decodificatorul zecimal generează 10 mintermeni, activi pe semnal coborît, de patru variabile active pe semnal ridicat. Dacă avem multe funcții de patru variabile putem utiliza un decodificator MSI pentru „predecodarea" variabilelor de intrare, înlocuind prin acestea un număr considerabil de porți logice.

Afișajul cu șapte segmente pe care l-am utilizat ca exemplu poate fi rezolvat (cu ieșirile active pe semnal coborît utilizînd un decodificator plus numai 2 7/12 capsule cu porți, deci un total de 3 7/12 cipuri.

Circuitele necesare sînt reprezentate în figura 4-9. Implementarea se face direct pornind de la tabelul de adevăr din figura 3-2 *d*, conectînd la intrările porților ieșirile mintermenilor din tabel, corespunzători lui 0 logic. De observat că deoarece numărul de 1 din tabel este mult mai mare decît cel de 0 atunci vor fi necesare un număr mai mare de porți logice pentru implementarea funcției utilizînd o intrare pentru fiecare 1 din tabelul de adevăr. Utilizarea zerourilor din tabel micșorează rețeaua de porți necesară, implementînd funcțiile complementare. În acest mod implementăm funcțiile com-



Număr de capsule:	
Decodificator	1
2 inversoare	1/3
1 NAND cu 2 intrări	1/4
3 NAND cu 3 intrări	1
2 NAND cu 4 intrări	1
Total capsule	3 7/12

Fig. 4-9. Decodificator pentru șapte segmente ca exemplu de realizare cu predecodificare.

plementate utilizînd forma canonică cu mintermeni, mintermenii fiind generați de decodificatorul MSI. A fost făcută o singură excepție, pentru funcția e', caz în care am folosit funcția d' ce utilizează o parte din mintermenii necesari pentru e'. Dacă nu am fi ținut seama de această observație ar fi trebuit să acordăm funcției e' o capsulă completă, conținînd o poartă standard cu opt intrări de la care am fi folosit numai șase intrări (deoarece porțile standard au numai două, trei, patru și opt intrări).

Această soluție necesită întîmplător, același număr de capsule ca și soluția anterioară în care s-au folosit multiplexoarele. Ea are totuși un ușor avantaj. Circuitele integrate sînt cu ceva mai ieftine deoarece cu excepția unuia, toate celelalte conțin numai porți simple. De asemenea, nu sînt necesare variabile de intrare complementate, iar încărcarea electrică a semnalelor de intrare este minimă deoarece sînt conectate numai la intrările decodificatorului, (în cazul soluției cu multiplexoare variabilele de intrare erau necesare și complementate, iar semnalele de intrare erau aplicate mai multor intrări).

Dacă dorim ieșiri active pe semnal ridicat va trebui să adăugăm în schemă șapte inversoare.

În general, predecodarea este un instrument util pentru simplificare în situația existenței *mai multor funcții de aceleași variabile*. Predecodarea înlătură și inconvenientul încărcării electrice a semnalelor de intrare cu un număr prea mare de porți, încărcare ce constituie deseori o problemă în astfel de situații. În loc de a utiliza abordarea prin sumarea strictă a mintermenilor predecodarea poate fi folosită în conjuncție cu simplificarea prin diagrame Veitch. Cînd extragem din diagramă ecuațiile logice simplificate putem recunoaște direct produsele ce au mai fost folosite pentru a le utiliza în continuare în mod convenabil. Acest mod de abordare este utilizabil numai pentru anumite tipuri de funcții ce pot fi recunoscute o dată cu trierea ecuațiilor pornind de la diagramele Veitch.

Dacă par a fi necesare un număr foarte mare de produse mari depinzînd de aceleași variabile, este convenabil să ne orientăm către o soluție ce presupune un decodificator căruia i se aplică la intrare aceste variabile. Funcții de această formă rezultă de obicei în cazurile în care unul sau mai multe moduri de operare sînt indicate prin intermediul a 3 sau 4 biți. Utilizarea multiplexoarelor sau a decodificatoarelor pentru manipularea biților de mod depinde esențial de cît de multe funcții sînt.

Se preferă utilizarea multiplexorului în cazul în care nu sînt prea multe funcții, deoarece deși folosirea multiplexorului necesită mai puține porți suplimentare, pentru fiecare funcție este necesar un multiplexor. Utilizarea unui decodificator se preferă în cazul existenței multor funcții deoarece *ieșirile* unui singur decodificator pot fi folosite pentru generarea unui număr mare de funcții.

Există și unele cazuri în care sînt folosite ambele tipuri de circuite: multiplexoare pentru eliminarea variabilelor și decodificatoare pentru funcțiile reziduale mai complexe.

4.3.1. Folosirea decodificatorului zecimal ca demultiplexor

Decodificatorul zecimal (vezi fig. 4-8) este deseori folosit ca demultiplexor pe opt căi. Dacă intrarea D este corectată la zero logic, circuitul poate fi gîndit ca un decodificator cu opt ieșiri (în acest mod de lucru ieșirile opt

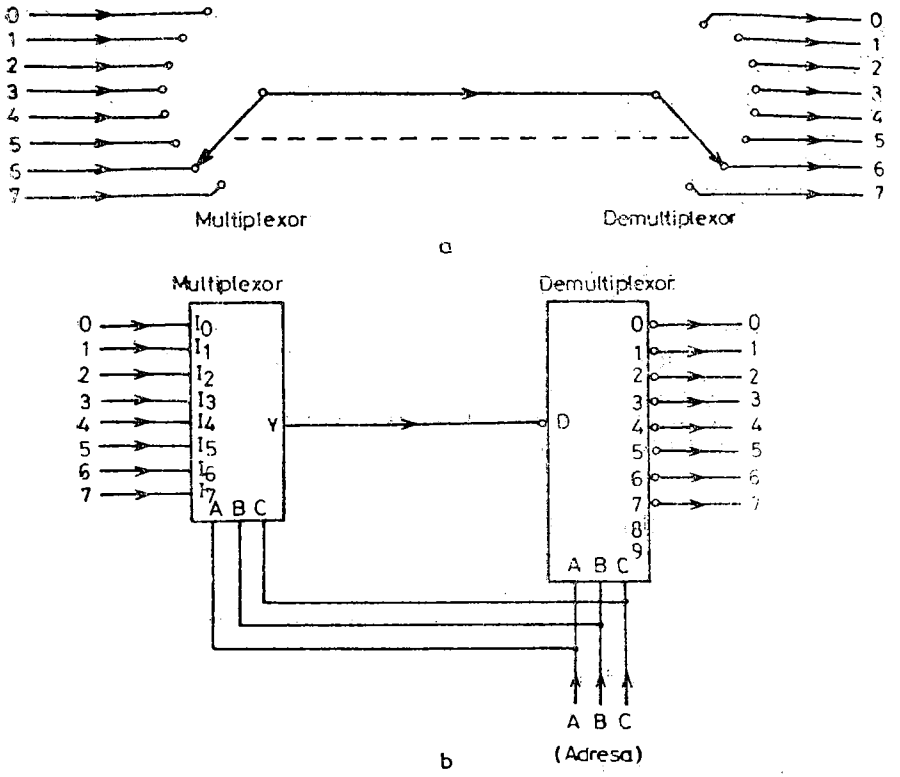


Fig. 4-10. Multiplexarea a opt semnale pe o singură cale: *(a)* utilizînd comutatoare mecanice; *(b)* utilizînd circuite logice MSI.

şi nouă nu sînt utilizate). Adresa de trei biţi formată din A, B şi C selectează o ieşire care va fi adevărată sau falsă în funcţie de intrarea D. Circuitul poate fi gîndit ca îndeplinind funcţia inversă multiplexării — demultiplexorul funcţionînd ca un selector conectat invers.

Figura 4-10 arată de ce aceste circuite sînt denumite multiplexoare şi demultiplexoare. Dacă dorim să transferăm mai multe semnale pe o singură cale, vom folosi la fiecare extremitate cîte un comutator în aşa fel încît la un moment dat să fie transferat un singur semnal. Acest procedeu este denumit multiplexare şi este ilustrat prin comutatoare în figura 4-10 *a*, iar circuitele MSI, în figura 4-10 *b*.

Se observă că în schema bloc a decodificatorului zecimal intrările de adresă A, B, C sînt desenate pentru această aplicaţie separat, tocmai pentru a-i pune mai clar în evidenţă modul de lucru. Cerculeţul pe intrarea D semnifică faptul că intrarea este activă pe zero (un semnal de intrare pe nivel coborît, determină pe una din cele opt ieşiri un semnal pe nivel coborît).

4.3.2. Construirea demultiplexoarelor/decodificatoarelor în arbore

Deoarece am evidenţiat dualitatea multiplexoarelor şi demultiplexoarelor, să reconsiderăm arborele de multiplexoare din figura 4-7 şi să folosim un principiu similar pentru a extinde, sub forma unui arbore, capacitatea demul-

tiplexorului, conform reprezentării din figura 4-11. Dacă intrarea D a demultiplexorului din stînga este menținută la nivel coborît, unul din cele opt decodificatoare de ieșire va fi activat, deoarece una din ieșirile primului decodificator va fi pe nivel ridicat. Biții de adresă D, E și F vor determina care ieșire a decodificatorului selectat va fi trecută în zero. Ca urmare a selectării cu 6 biți, una din cele 64 de ieșiri va trece în zero. Dacă dorim construirea unui decodificator cu 80 de ieșiri va trebui să adăugăm două decodificatoare de ieșire și să folosim intrarea D a primului decodificator ca o intrare de adresă. Vom obține 10 semnale de condiționare pentru cel de al doilea nivel. De observat că această tehnică este imposibil de aplicat dacă vrem să demultiplexăm, căci în acest caz avem nevoie de intrarea D pentru date.

Ca și în cazul multiplexoarelor, putem construi demultiplexoare de dimensiuni oricît de mari adăugînd niveluri suplimentare. Spre exemplu, dacă fiecare ieșire a demultiplexorului de 64 de biți va fi folosită pentru a condiționa intrarea D a 64 de decodificatoare, vom construi un demultiplexor/decodificator cu $64 \times 8 = 512$ ieșiri.

4.3.3. Submultiplexare/demultiplexare

O aplicație practică a decodoarelor și demultiplexoarelor mari este aceea de a activa sau de a sesiza starea unui dispozitiv oarecare, unul din multiplele dispozitive ale unui sistem mare.

Un exemplu poate fi un multiplexor cu 1 000 de intrări pentru sesizarea stării a 1 000 de linii telefonice. În loc să utilizăm un multiplexor cu 1 000 de intrări la care sosesc din 1 000 de locuri 1 000 de fire este mult mai eficient să dispersăm circuitele propriu zise, pe mai multe plăci cu circuite imprimate. Spre exemplu, circuitele telefonice sînt distribuite cîte 16 pe o placă și vom adăuga cîte un multiplexor de 16 căi pe fiecare. Aceeași adresă de patru biți este trimisă către fiecare modul și o singură ieșire „submultiplexată" va merge către un circuit central ce constă dintr-un multiplexor de 64 de căi precum cel din figura 4.7. Vom putea, deci, colecta date de la $64 \times 16 = 1\,024$ puncte utilizînd numai 64 de fire. Se spune că fiecare din cele 64 de fire este submultiplexat. Similar, pot fi distribuite comenzi către 1 024 de destinații pe cele 64 de fire ale unui decodificator.

Trebuie să reținem ideea : nu este nevoie ca toate circuitele să fie plasate în același loc, ele putînd fi distribuite în întregul sistem. Pe lîngă micșorarea numărului de conexiuni, apare și posibilitatea conceperii unui sistem modular, extensibil prin module de 16 circuite, în sistemul de bază existînd o rețea de multiplexoare de dimensiuni moderate.

4.4. ADRESARE MULTIDIMENSIONALĂ

Unele aplicații presupun adresarea unui număr foarte mare de puncte iar modulul de bază nu posedă un decodificator de 8 sau mai multe adrese realizat cu circuite MSI. Dacă, de exemplu, există o singură adresă pe modul, este nepractic să plasăm nivelul final al decodificatorului pe modul. De asemenea este nepractic să avem un decodificator centralizat cu 1 000 de ieșiri legate cu cîte un fir de fiecare modul. Altă abordare ar putea fi decodarea pe fiecare modul a unui cuvînt de adresă comun de 10 biți cu o poartă cu 10

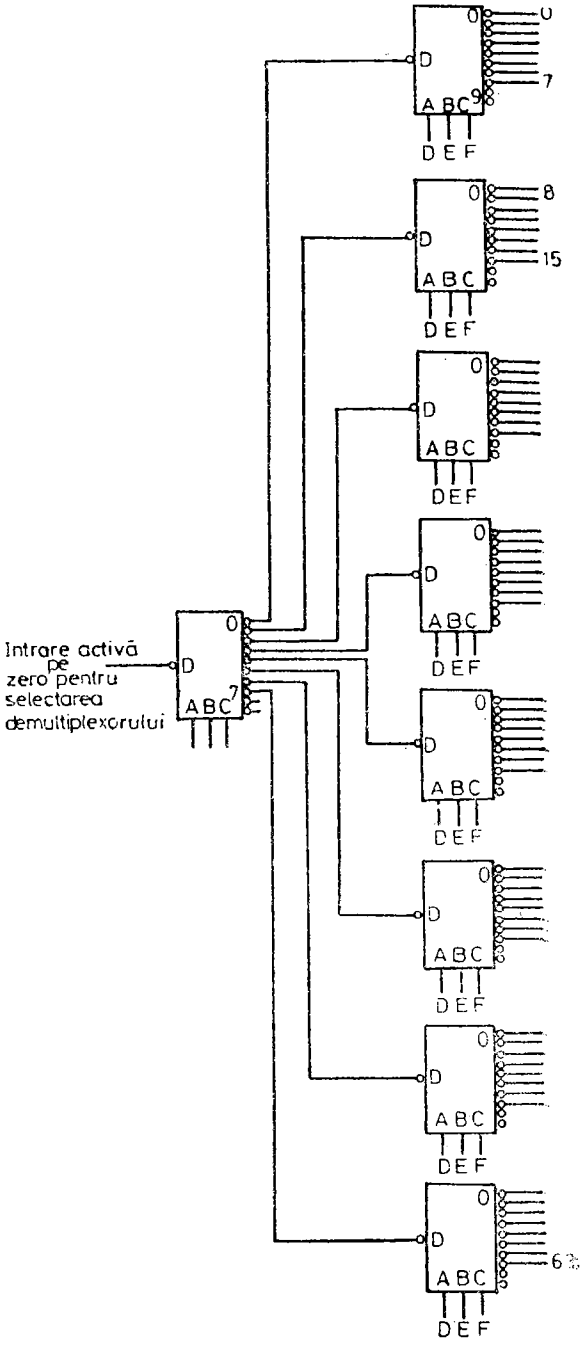


Fig. 4-11. Demultiplexor/decodificator de 64 de căi, structurat arborescent.

intrări. Și această abordare este ineficientă deoarece ajungem să folosim pentru decodare 1000 de porți cu 10 intrări și în plus cele 10 linii de adresă vor avea în sarcină 1 000 de porți.

Soluția cea mai bună este de a folosi pe fiecare modul o poartă cu un număr mic de intrări și o predecodificare cu circuite MSI. Dacă vom utiliza, de exemplu, pe fiecare modul porți cu trei intrări, vor fi necesare trei decodificatoare zecimale pentru a predecodifica cele 1 000 de adrese. Predecodificatorul va avea un număr de numai $3 \times 10 = 30$ conexiuni de ieșire. Vom considera cele trei decodificatoare asociate unităților, zecilor și sutelor numărului zecimal. Fiecare din cele 1 000 de module va avea o adresă cuprinsă între 000 și 999. Spre exemplu, modulul cu numărul 132 folosește o poartă cu trei intrări conectată la ieșirea 1 de la „sute" la ieșirea 3 de „zeci" și la ieșirea 2 de la „unități". Aceste trei semnale vor fi active numai atunci cînd la intrarea predecodificatorului se va afla numărul 132. Fiecare ieșire a predecodificatorului va genera semnal pentru 100 de intrări, deci vor trebui folosite circuite de comandă puternice la ieșirea decodificatoarelor. Această schemă de decodificare poate fi denumită tridimensională. Deoarece există 10 semnale pe fiecare din cele trei dimensiuni putem adresa $10 \times 10 \times 10 = 10^3$ puncte. Dacă vom descreste numărul dimensiunilor la două vor fi necesare $\sqrt{1\,000} = 32$ de semnale pe fiecare dimensiune. Vor exista 32 de fire de selecție pe dimensiunea X și tot atîtea pe Y. Se simplifică, în acest caz, porțile de selecție dar se complică logica la preselecție. (Figura 4-12 *a* prezintă o matrice bidimensională cu $8 \times 8 = 64$ de ieșiri). Fiecare ieșire de preselecție atacă acum 32 de porți, reducîndu-se încărcarea ieșirilor precodificatorului de 3 ori. Cu toate că sînt necesare 10 capsule integrate pentru predecodificator față de numai 3 capsule în cazul tridimensional, utilizarea a 1 000 de porți cu două intrări (250 cipuri*) față de 1 000 de porți cu trei intrări (333 cipuri)*, face varianta bidimensională mai avantajoasă. De fapt abordarea cea mai economică din punctul de vedere al numărului de capsule ar consta în utilizarea unui decodificator cu 1 024 de ieșiri. Această abordare nu este totuși de dorit din cauza problemelor de interconectare.

Rezultă, deci, că abordarea bidimensională este cea mai bună soluție deoarece $2 \times 32 = 64$ conexiuni și 10 circuite integrate pot fi plasate cu ușurință într-un modul central.

Pentru un număr mai mare de puncte selectate, varianta tridimensională poate deveni avantajoasă prin numărul mai mic de conexiuni ce-l presupune. Spre exemplu cu $3 \times 24 = 72$ de fire putem selecta $24^3 = 13\,824$ puncte.

Figura 4-12 *b* prezintă problema inversă, aceea a selectării unor intrări. O poartă cu colectorul în gol este folosită pentru generarea pe Y a semnalelor selectate prin decodificatorul pe X. Selectorul pe Y (un multiplexor) alege intrarea corectă. În figura 4-12 *b* este reprezentat un selector de $8 \times 8 = 64$ de intrări ; dar utilizînd arbori, prezentați în secțiunea anterioară, pot fi selectate 1024 de puncte cu o matrice de 32×32.

Desigur, pot fi gîndite și matrici cu dimeniuni X, Y inegale, precum $X = 128$, $Y = 8$ ($128 \times 8 = 1\,024$) dar sînt, de regulă neutilizate datorită numărului mare de conexiuni pe care îl presupun (se observă că ar fi $128 + 8 = 136$ conexiuni). Putem enunța regula : *dimensiunile trebuie astfel determinate încît costul implementării fiecărei dimensiuni să fie egal.*

Această regulă este aplicabilă în multe situații în care capacitatea sistemului variază cu produsul dimensiunilor iar costul cu suma lor. În cazul

---

* Se presupune că putem utiliza toate porțile dintr-o capsulă.

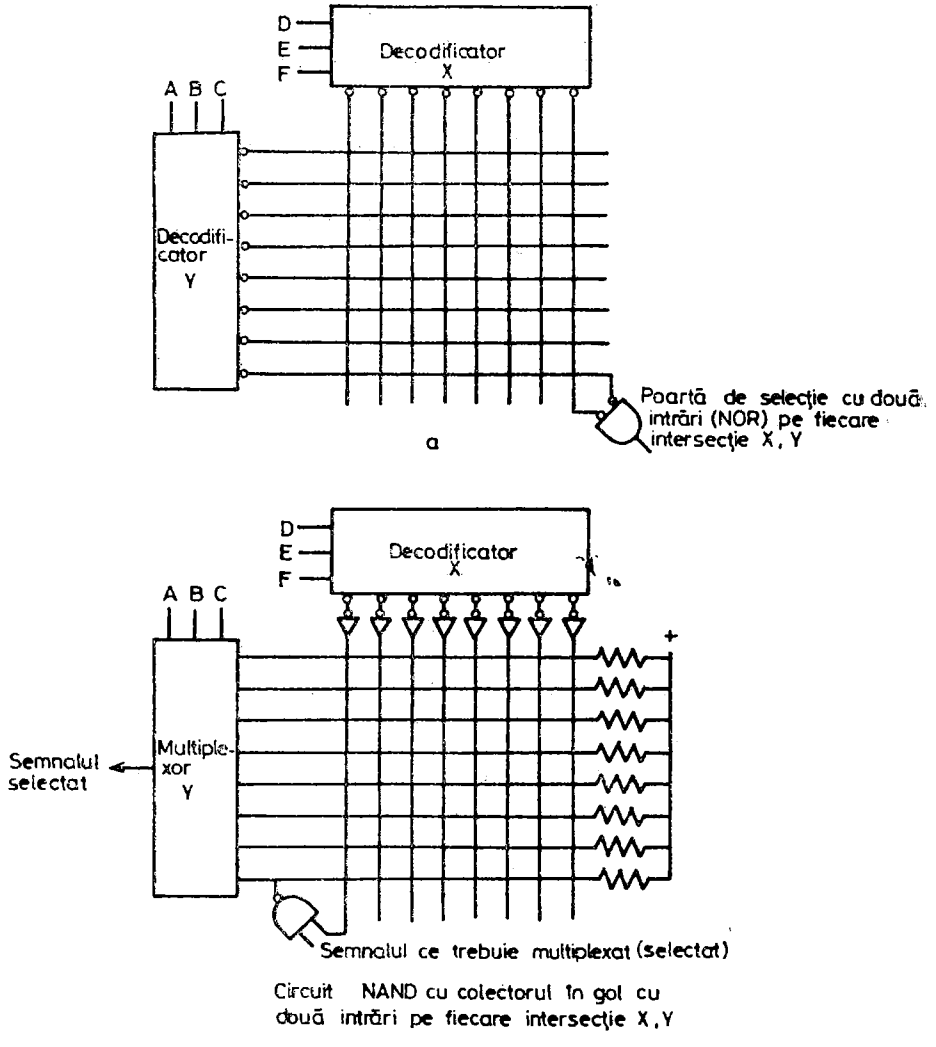


Fig. 4-12. Matrice de selecţie X-Y: *(a)* selector de ieşiri (decodificator); *(b)* selector de intrări (multiplexor).

bidimensional, costul dimensiunilor X şi Y este acelaşi dacă şi mărimea lor este aceeaşi deoarece circuitele utilizate în fiecare dimensiune sînt identice. În situaţia în care pe o anumită dimensiune sînt folosite circuite speciale mai scumpe, ca amplificatoare de linie şi ieşiri simple de decodificator pe alta (situaţie des întîlnită în memoriile de tip RAM), se va opta pentru dimensiuni inegale, invers proporţionale cu preţul de cost pe circuit. De asemenea modularizarea sau limitări date de numărul de conexiuni, pot determina abateri de la stricta egalitate a costului dimensiunilor.

Cu toate că circuitele de multiplexare şi demultiplexare sînt folosite uneori pentru implementarea funcţiilor logice, marea lor utilitate este probată la nivelul proiectării de sisteme. Ori de cîte ori semnalele trebuie selectate

de la, sau trimise la o multitudine de surse, sau atunci cînd modul de operare se comută între aceste două situații proiectarea de sistem trebuie *„adaptată"* la posibilitățile oferite de multiplexoarele și demultiplexoarele MSI.

4.5. ALTE TIPURI DE CIRCUITE COMBINAȚIONALE MSI

Cataloagele producătorilor de circuite integrate sînt pline cu descrieri detaliate pentru foarte multe alte tipuri de circuite MSI. *Aceste cataloage constituie cele mai importante instrumente de lucru ale proiectantului de sisteme digitale**. Circuitele disponibile trebuie bine studiate și revăzute continuu pe toată durata proiectării. Cu toate că vom utiliza la maximum circuitele complexe disponibile, nu trebuie să pierdem din vedere nici un moment tehnicile tradiționale de proiectare.

Proiectarea schemelor ce realizează operații aritmetice binare constituie una din preocupările deosebit de importante legată de sistemele digitale. Aceasta este una din direcțiile importante pe care se plasează preocupările producătorilor de circuite integrate. Un procesor programabil realizat într-o capsulă (vezi Capitolul 7) poate manipula toate operațiile aritmetice și poate fi programat pentru a le combina în funcții complexe. Dacă procesorul programabil se dovedește a fi prea lent, va trebui să procurăm o serie de circuite aritmetice MSI pentru a rezolva problemele numerice. Dăm în continuare cîteva exemple:

1. *Sumatorul de patru biți* ce poate genera suma binară a două numere de 4 biți. Expandarea funcției pentru cuvinte de orice lungime este posibilă, datorită intrării și ieșirii de transport (carry), Spre exemplu, cu patru astfel de sumatoare se obține suma de două numere de 16 biți (ex: 74283).

2. *Generatorul de transport anticipat* (look-ahead carry generator) este folosit împreună patru sumatoare de 4 biți pentru mărirea vitezei de operare pe 16 biți.Pentru cuvinte mai mari, la fiecare 16 biți, se adaugă un astfel de circuit (ex: 74182).

3. *Comparatorul de patru biți*, compară două numere binare A și B de 4 biți generînd la ieșire: $A = B$, $A > B$ și $A < B$. Posedă intrări pentru expandare, ce se poate realiza pentru o lungime oricît de mare a cuvîntului dacă se generează o rețea arborescentă. Spre exemplu, pentru cuvinte de 24 de biți se construiește o structură cu două nivele folosind șase comparatoare. Un arbore cu trei nivele poate compara pînă la 120 de biți; într-o variantă redusă pot compara cuvinte de 32 de biți cu o structură ce conține 10 capsule (ex: 7485).

4. *Unitatea logico-aritmetică* (ALU) multifuncțională poate realiza cu cuvinte de patru biți sumări, scăderi, XOR, OR, NOR, AND, NAND, comparație ș.a., comandate printr-un cuvînt de cod. De asemenea poate oferi rezultatul, deplasat la stînga sau la dreapta cu sau fără transport (carry). Folosind un circuit pentru fiecare patru biți se poate expanda pînă la orice lungime de cuvînt. Pentru obținerea unei viteze mari de operare va trebui utilizat circuitul pentru generarea transportului anticipat (ex: 74281)**

---

* Excelentul catalog al firmei Texas Instruments poate fi obținut la un preț foarte mic. Cataloage excelente au și firmele Signetics și Advanced Micro-Devices.

** Este interesant de observat că acest circuit costă aproximativ atît cît costă un circuit integrat cu porți de acum 12 ani și că realizează adunarea a două cuvinte de 16 biți în aproximativ timpul necesar semnalului să se propage printr-o poartă în acea epocă.

5. *Multiplicatorul binar* de 4 × 4 generează produsul binar a două numere de patru biți și oferă la ieșire patru biți pentru extensie. Două astfel de circuite permit realizarea unui multiplicator de opt biți. Expandarea funcției de multiplicare cu astfel de circuite nu este directă fiind necesare circuite adiționale. Spre exemplu, un multiplicator de numere de opt biți cu 16 ieșiri necesită 16 capsule, dintre care numai opt sînt multiplicatoare (ex : 74284).

Pe lîngă funcții aritmetice, circuitele standard MSI pot realiza și alte funcții combinaționale uzuale. Listăm, cîteva exemple :

1. *Codificatorul cu prioritate*, este un circuit cu opt intrări și trei ieșiri ce generează la ieșire codul binar al intrării selectate. Dacă este selectată mai mult de o singură intrare este luată în considerație (are prioritate) cea notată cu numărul mai mare. Circuitul are intrări și ieșiri de selecție ce permit cascadarea acestora pentru obținerea unui număr mai mare de intrări. Aceste scheme de extindere necesită, însă, porți adiționale pentru generarea biților de ieșire. Spre exemplu, pentru un număr de 16 intrări sînt necesare două codificatoare și patru porți NAND cu două intrări. Versiunea pentru 10 intrări ce generează cod BCD, nu este expandabilă. Acest circuit este utilizat pentru codarea matricilor cu taste de mici dimensiuni, pentru manipularea întreruperilor în calculatoare și pentru alte aplicații (ex. 74148).

2. **Testorul/*generatorul* de paritate** de nouă biți, care generează sau verifică bitul de paritate sau de imparitate. Acest circuit poate fi conectat într-o rețea arborescentă cu două niveluri pentru a obține o structură pentru 9 × × 9 = 81 biți, utilizînd 9 + 1 = 10 capsule integrate. Cu o rețea parțială, ce nu folosește complet intrările celui de al doilea nivel, se poate obține un circuit pentru 9 + 9 + 7 = 25 de biți, numai cu trei capsule. În plus, cu aceleași tipuri de circuite poate fi realizată o rețea care să genereze **coduri „Hamming"**, ce indică nu numai apariția unei erori dar și bitul eronat (ex : 74280).

3. *Convertorul de la șase biți la BCD*, generează un cuvînt de șapte biți codat BCD pornind de la un număr binar de șase biți. Pentru conversia a opt biți trebuie interconectate trei astfel de circuite iar 16 capsule sînt necesare pentru 16 biți. Ca și convertorul din BCD în binar prezentat mai jos, extinderea structurii conduce la o structură complexă pe mai multe nivele (ex : 74185).

4. *Convertorul din BCD în binar de sase biți* constituie jumătate din schema necesară convertirii a două numere de 4 biți codate în BCD într-un număr binar de șapte biți. Prin interconectarea a șase atfel de circuite se obține conversia a trei numere BCD la un număr binar de nouă biți. Fiecare extindere peste acest nivel presupune dublarea numărului circuitelor folosite (ex: 74184).

Listarea făcută anterior a indicat tipurile de circuite MSI disponibile. Deoarece ecuațiile logice ce descriu funcționarea lor sînt foarte complexe, *determinarea aplicabiltății lor poate fi făcută numai la nivelul proiectării de sistem.* Odată ce am început să facem tabele de adevăr sau să scriem ecuații logice este deja prea tîrziu pentru a mai utiliza aceste circuite. De multe ori chiar specificațiile la nivel de sistem sînt afectate de cunoașterea inițială a circuitelor standard disponibile.

4.6. MEMORII FIXE (ROM)

Memoriile fixe (read-only memory, ROM) constituie un bun exemplu pentru ceea ce urmează standardizării ; am putea spune „standardizarea pentru utilizator" („customized standardization"). Odată cu lansarea de către Henry Ford a Modelului T, coborîrea prețului a fost posibilă prin fabricarea a mii de automobile identice. El spunea că poți avea un automobil Ford „în orice culoare dorești — atît timp cît ea este neagră". Astăzi nivelul de sofisticare pe care-l generează calculatoarele și mașinile automate ne permit comandarea unui automobil, în orice opțiune dorim (așa cum îl vrea clientul !) Beneficiem încă de pe urma procesului de stadardizare, *dar a unui proces de standardizare mult mai sofisticat.*

Memoriile fixe reprezintă o culme a standardizării pentru utilizator ; spre exemplu putem specifica configurația de 1 și 0 într-o structură cu 16 384 de biți. Numărul combinațiilor posibile este de $2^{16.384}$ ceea ce reprezintă aproximativ 10^{5000}. Pentru comparație, a fost calculat că 10^{63} electroni pot ocupa spațiul unei sfere de dimensiunea globului pămîntesc.

Această fantastică posibilitate de adaptare la cerințele utilizatorilor a fost posibilă prin folosirea calculatorului în generarea automată a **măștilor** într-o anumită fază a procesului tehnologic de realizare a circuitelor integrate. Deoarece generarea măștilor este automată, configurația dorită de utilizator este introdusă prin cartele perforate, iar prin utilizarea unor iluminări selective în procesul tehnologic se înlătură sau nu oxidul din anumite zone ale circuitului integrat obținîndu-se configurația de 0 și 1 dorită.

Memoria fixă de 16 384 biți (16 kb) utilizată pentru exemplificare este formată dintr-o matrice $X - Y$ de 128 × 128 de puncte care sînt sau nu „conductoare" (vezi fig. 4-13). Circuitul are opt ieșiri și 11 intrări de adresă și este organizat în 2 048 de cuvinte de opt biți (2 kocteți). Pentru fiecare din combinațiile de intrare posibile este generată o combinație de ieșire, conform specificației utilizatorului. Funcțional, o astfel de schemă este identică cu aceea din figura 4-12 *b*, exceptînd ieșirea care este selectată în opt grupuri de cîte 16.

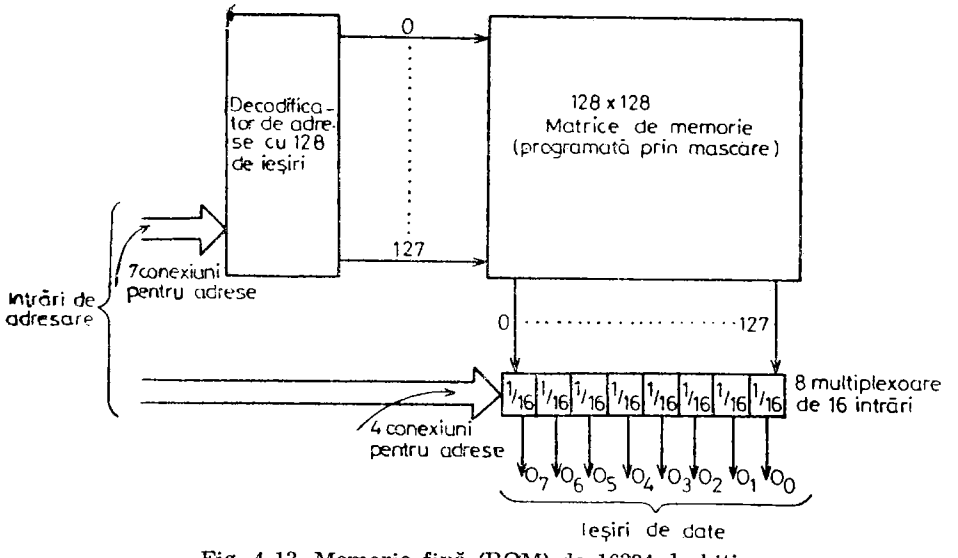


Fig. 4-13. Memorie fixă (ROM) de 16384 de biți.

În tabelul 4.4 este dată o listă ce completează tabelul 4-3 cu numărul maxim de capsule asociat implementării unor funcții logice. Nu ar fi corect să considerăm în contribuția la numărul de cipuri ROM-ul ca pe o singură capsulă, deoarece, în primul rînd, majoritatea acestor tipuri de circuite folosesc capsule cu 24 de terminale ce ocupă o suprafață dublă față de una obișnuită.

Tabelul 4-4. **Memoriile de timp ROM ca generatoare de funcții**

Funcțiile logice/Variabile	Coeficient de multiplicare a prețului de cost față de implementarea cu porți (montate în sistem) — 1978	Organizarea memoriei ROM
Oricare 4 de 8 variabile	~2	256×4 biți = 1000
Oricare 8 de 8 variabile	~2	256×8 biți = 2000
Oricare 4 de 9 variabile	~2	512×4 biți = 2000
Oricare 8 de 9 variabile	~2	512×8 biți = 4000
Oricare 4 de 10 variabile	~2	1024×4 biți = 4000
Oricare 8 de 10 variabile	~2	1024×8 biți = 8000
Oricare 4 de 11 variabile	~2	2048×4 biți = 8000
Oricare 8 de 11 variabile	~3	2048×8 biți = 16000
Oricare 4 de 12 variabile	~3	4096×4 biți = 16000

În al doilea rînd costul ROM-urilor este de cîteva ori mai mare decît cel al circuitelor obișnuite; dacă includem prețurile „ascunse" discutate în Capitolul 2, un ROM montat în sistem costă de 2 pînă la 4 ori mai mult decît o capsulă cu porți montată în sistem. În viitorul apropiat pot apare reduceri ale acestui raport. ROM-urile sînt disponibile în formate de diverse dimensiuni ca organizări, listate în tabelul 4-4 împreună cu posibilitățile pe care le oferă pentru implementarea funcțiilor logice.

Toate memoriile fixe din tabel (exceptînd pe cele de 1 024 și 2 048 de biți ce au 16 terminale sînt disponibile în capsule de 24 de terminale, fiind clar că *funcțiile logice deosebit de complexe nu pot fi implementate utilizînd porți discrete.*

4.7. MEMORII FIXE PROGRAMABILE

Chiar dacă confecționarea măștilor pentru ROM-uri la cererea utilizatorului este automată, acest procedeu este recomandabil numai dacă se pot comanda minimum 100 de circuite identice. Această limitare apare datorită faptului că se declanșează un ciclu special de producție pentru fiecare tip de ROM chiar dacă este specifică numai o etapă de mascare. Un alt dezavantaj al memoriilor fixe programabile prin mascare este acela că o eroare odată făcută nu mai poate fi corectată*. Din aceste motive pentru loturi mici de producție sau pentru prototipuri este esențială folosirea altor tipuri de ROM-uri.

* Totuși este posibilă adăugarea de porți AND — pentru înlăturarea — sau de porți OR — pentru includerea — de mintermeni adiționali.

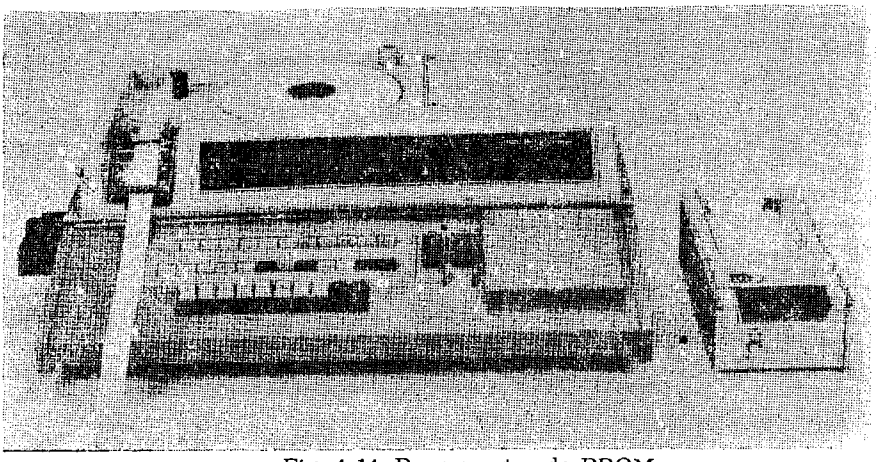


Fig. 4-14. Programator de PROM.

Un astfel de dispozitiv standard este **memoria fixă programabilă (programmable read-only memory-PROM)**. Pentru a programa acest dispozitiv utilizatorul (uneori chiar distribuitorul) folosește un „programator de PROM" introducînd configurația dorită de biți în memorie. Unele PROM-uri sînt *reprogramabile* în sensul că pot fi șterse electric sau prin expunerea la lumină ultravioletă și reprogramate în mod repetat. Deoarece PROM-urile sînt deobicei incompatibile cu circuitele echivalente programabile prin mascare, *sistemele produse în cantitate mare vor utiliza PROM-uri pînă în momentul în care punerea la punct a producției și experiența acumulată justifică înlocuirea lor cu ROM-uri programabile prin mascare.* Masca pentru ROM-uri va fi generată folosind *aceeași* bandă de hîrtie perforată sau aceleași cartele utilizate la programarea PROM-urilor, fiind introduse în același loc pe placheta cu circuit imprimat.

Sistemele de punere la punct a unei scheme pot fi simplificate și mai mult prin folosirea simulatoarelor de ROM. Acest aparat posedă modalități de conectare în soclurile ROM-urilor din sistemul de pus la punct. Acest simulator

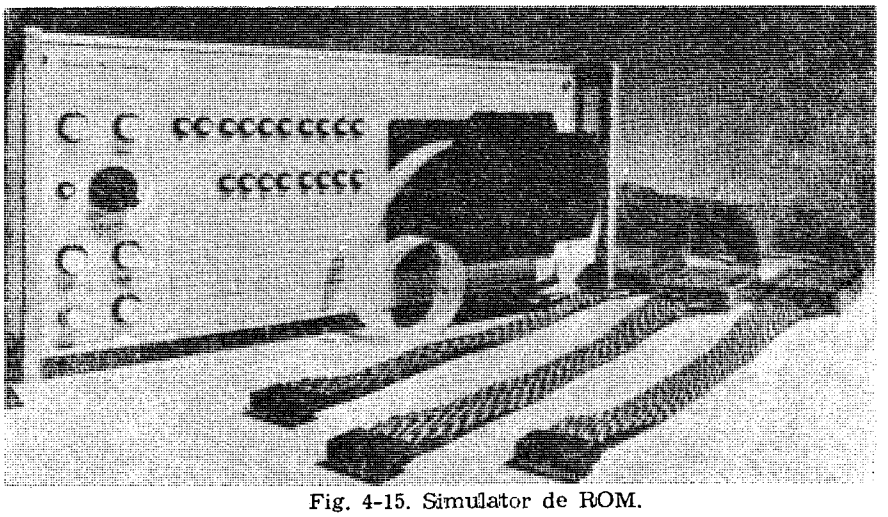


Fig. 4-15. Simulator de ROM.

folosește memorii de tip RAM pentru a simula memoriile fixe. Orice cuvînt în această memorare poate fi modificat, utilizînd comenzile de pe panou, fără a fi necesară reprogramarea întregii memorii ROM. Programul este introdus, ca și în programatorul de PROM, prin intermediul unei benzi perforate de hîrtie.

Dacă comparăm metoda de implementare prin porți a funcțiilor logice cu cea utilizînd PROM-uri, observăm că acestea din urmă pot realiza funcția unei rețele complexe de porți. Diferența constă în faptul că utilizînd un programator de PROM relativ simplu „interconexiunile" rețelei de porți simulate pot fi manipulate automat. Prin aceste procedee se înlătură necesitatea unor echipamente greoaie și costisitoare care să permită realizarea, modificarea și testarea sistemelor bazate pe realizarea exclusivă a funcțiilor logice prin porți.

4.8. MINIMIZAREA CIRCUITELOR LOGICE CE UTILIZEAZĂ ROM-URI

Procedeele de simplificare a ecuațiilor logice, discutate în Capitolul 3, sînt lipsite de sens în cazul utilizării ROM-urilor. Pentru a aduce funcția logică la forma utilizabilă pentru programarea ROM-urilor este necesară scrierea ei utilizînd forma canonică cu mintermeni. Se pune deci de multe ori problema expandării modului în care este scrisă funcția logică. Această formă poate fi obținută direct din tabelul de adevăr (vezi fig. 4-16). Deoarece fiecare termen reprezintă o funcție vom lucra cu cîte o funcție o dată. Pentru fiecare termen al ecuației logice vom scrie unul sau mai multe 1-uri în tabelul de adevăr. Unul singur dacă toate variabilele de intrare fac parte din termen (mintermeni). Dacă lipsesc din termen n variabile aceasta va presupune introducerea a 2^n 1-uri în tabelul de adevăr (unii din 1 pot fi deja în tabel, provenind de la ceilalți termeni). Figura 4-16 ilustrează implementarea cu un ROM a următoarelor ecuații logice:

$$f_1 = O_1 = A \cdot B' \cdot C' \cdot D \cdot E \cdot F' \cdot G \tag{4.4}$$

$$f_2 = O_2 = B \cdot C' \cdot F \cdot G' \tag{4.5}$$

$$f_3 = O_3 = A \cdot C' \cdot G \tag{4.6}$$

$$\begin{aligned} f_4 = O_4 = {} & A' \cdot B' \cdot C \cdot D' \cdot E \cdot F \cdot G \cdot H + A \cdot B' \cdot C' \cdot F + E \cdot F' \cdot G \cdot H + \\ & + B \cdot C \cdot D \cdot F' + A \cdot B \cdot C + D \cdot E' \cdot G \cdot H + A \cdot C \cdot D \cdot F' + \\ & + B' \cdot C' \cdot D' \cdot G' + A' \cdot C \cdot D \cdot H \end{aligned} \tag{4.7}$$

Primele trei funcții sînt folosite, în acest exemplu, numai pentru a ilustra modul în care se face asocierea între termeni și zonele completate din tabelul de adevăr (ele sînt prea simple pentru a fi implementate de fapt cu ROM). Variabilele sînt marcate în coloana din stînga tabelului și în partea de sus a acestuia, permițînd localizarea mintermenilor. Spre exemplu, f_1 este formată dintr-un singur mintermen: cuvîntul cu numărul 230. Pe de altă parte, f_2, care este format numai din 4 variabile, va fi marcat prin $2^4 = 16$ mintermeni. Se vor identifica cele 8 situații în care $B = 1$, $C = 0$ și $F = 0$ și se vor completa cu 1 în cele două coloane unde $G = 0$.

Utilizarea proiectării automate folosind calculatoare programabile este foarte indicată în acest caz în care corespondența între funcția logică și reprezentarea prin conținutul ROM-ului este directă, dată de tabelul de adevăr. Un program relativ simplu poate converti ecuațiile logice în benzi sau cartele perforate utilizabile pentru programarea circuitelor ROM sau PROM.

CUSTOMER: _____ THIS PORTION TO BE COMPLETED BY SIGNETICS

P.O. NO.: _____ PART NO.: _____

YOUR PART NO.: _____ S.D. NO.: _____

DATE: _____ DATE RECEIVED: _____

	GH = 00					GH = 01					GH = 10					GH = 11				
ABCDEF	Word	OUTPUT 04	03	02	01	Word	OUTPUT 04	03	02	01	Word	OUTPUT 04	03	02	01	Word	OUTPUT 04	03	02	01
000000	0	1	0	0	0	64	1				128					192				
000001	1	1	0	0	0	65	1				129					193				
000010	2	1	0	0	0	66	1				130					194	1			
000011	3	1	0	0	0	67	1				131					195				
000100	4	0	0	0	0	68					132					196	1			
000101	5	0	0	0	0	69					133					197	1			
000110	6	0	0	0	0	70					134					198	1			
000111	7	0	0	0	0	71					135					199				
001000	8	0	0	0	0	72					136					200				
001001	9	0	0	0	0	73					137					201				
001010	10	0	0	0	0	74					138					202	1			
001011	11	0	0	0	0	75					139					203	1			
001100	12	0	0	0	0	76	1				140					204	1			
001101	13	0	0	0	0	77	1				141					205	1			
001110	14	0	0	0	0	78	1				142					206	1			
001111	15	0	0	0	0	79	1				143					207	1			
010000	16	0	0	0	0	80					144					208				
010001	17	0	0	1	0	81				1	145					209				
010010	18	0	0	0	0	82					146					210	1			
010011	19	0	0	1	0	83				1	147					211				
010100	20	0	0	0	0	84					148					212	1			
010101	21	0	0	1	0	85				1	149					213	1			
010110	22	0	0	0	0	86					150					214	1			
010111	23	0	0	1	0	87				1	151					215				
011000	24	0	0	0	0	88					152					216				
011001	25	0	0	0	0	89					153					217				
011010	26	0	0	0	0	90					154					218	1			
011011	27	0	0	0	0	91					155					219				
011100	28	1	0	0	0	92	1				156	1				220	1			
011101	29	0				93	1				157					221	1			
011110	30	1				94	1				158	1				222	1			
011111	31	0				95	1				159					223	1			
100000	32	1				96	1				160	1	1			224	1	1		
100001	33	1				97	1				161	1	1			225	1	1		
100010	34	1				98	1				162	1	1			226	1	1		
100011	35	1				99	1				163	1	1			227	1	1		
100100	36	0				100					164		1			228	1	1		
100101	37	1				101	1				165	1	1			229	1	1		
100110	38	0				102					166		1			230	1	1		1
100111	39	1				103	1				167	1	1			231	1	1		
101000	40	0				104					168					232				
101001	41	0				105					169					233				
101010	42	0				106					170					234	1			
101011	43	0				107					171					235				
101100	44	1				108	1				172	1				236	1			
101101	45	0				109					173					237	1			
101110	46	1				110	1				174	1				238	1			
101111	47	0				111					175					239				
110000	48	0				112					176					240				
110001	49	0			1	113				1	177					241				
110010	50	0				114					178					242	1			
110011	51	0			1	115				1	179					243				
110100	52	0				116					180					244	1			
110101	53	0			1	117				1	181					245	1			
110110	54	0				118					182					246	1			
110111	55	0			1	119				1	183					247				
111000	56	1				120	1				184	1				248	1			
111001	57	1				121	1				185	1				249	1			
111010	58	1				122	1				186	1				250	1			
111011	59	1				123	1				187	1				251	1			
111100	60	1				124	1				188	1				252	1			
111101	61	1				125	1				189	1				253	1			
111110	62	1				126	1				190	1				254	1			
111111	63	1				127	1				191	1				255	1			

Fig. 4-16. Tabel de adevăr pentru un ROM de 1024 de biţi/formular de comandă.

Deoarece ROM-urile permit implementarea *oricărei* funcţii logice, ele reprezintă un procedeu extraordinar de puternic pentru realizarea tuturor funcţiilor ce pot apare în problemele concrete şi pot fi convertite într-o reprezentare logică. Pentru a utiliza cu maximă eficienţă o memorie de tip ROM în implementări de funcţii logice variabilele de intrare şi ieşire trebuie codate astfel încît să conţină cît mai multă informaţie. Cu cît codarea intrărilor şi ieşirilor unui ROM este mai compactă, cu atît mai complicate vor fi şi ecuaţiile logice. Totuşi, deoarece mărimea ROM este determinată *numai* de numărul de intrări şi ieşiri, rezultatul este în cele din urmă economisirea de hardware.

Şi în acest caz decodorul pentru un afişaj cu şapte segmente poate fi folosit ca un exemplu foarte bun. Deoarece toate cele şapte ieşiri se pot obţine cu uşurinţă pornind de la codarea cu patru biţi a numerelor zecimale (BCD)

cantitatea de informație conținută de cele șapte semnale este mai mică decît cea conținută de cei patru biți ai codului BCD. Deci, o cale de a reduce numărul ieșirilor unui ROM dintr-un sistem ce folosește un afișaj cu 7 segmente este de a utiliza un decodor BCD-7 segmente conectat la patru ieșiri ale ROM și nu de a utiliza șapte ieșiri, făcînd și decodarea în ROM.

Un caz extrem îl poate constitui situația în care o funcție logică presupune și trimiterea de exemplu a opt biți către una din patru destinații distincte. În loc de a utiliza un ROM cu 8 intrări și $8 \times 4 = 32$ ieșiri vom folosi doi din biții de adresă, ce ar apare într-o astfel de schemă, pentru a comanda patru demultiplexoare duale cuplate la ieșirea unui ROM cu 8 ieșiri. *Vom concentra deci în ROM acea parte a funcției logice care este nestandard și vom implementa cu circuite MSI sau cu porți elementare funcțiile speciale simple.*

Acumularea de experiență ne permite să determinăm grupuri de intrări care pot fi codate extern și grupuri de ieșire care pot fi decodate extern. *Poate fi codificat orice grup de variabile ce sînt mutual exclusive* (adică una singură poate fi activă într-un interval de timp). De exemplu, cele 16 taste ale unei claviaturi nu vor fi apăsate decît una o dată și succesiv; din acest motiv putem reduce numărul de intrări în ROM de la 16 la numai 4 prin codarea lor externă.

Un alt exemplu este dat de cele 12 intrări generate de o cartelă perforată utilizată pentru introducerea datelor într-un computer. Pe primele șapte rînduri de pe cartelă pe o coloană există cel mult o perforație. Putem deci codifica aceste șapte intrări utilizînd un codificator MSI. În figura 4-17 sînt prezentate două soluții pentru convertirea celor 12 biți de pe cartela perforată în cod **ASCII.** Fără codificare, cele 12 intrări presupun $2^{12} = 4\,096$ cuvinte, fiind nevoie deci de un ROM de $4\,096 \times 8 = 32\,768$ biți. Prin utilizarea unui codificator MSI (vezi fig. 4-17 *b*) mărimea memoriei se poate reduce la $2^8 = 256$

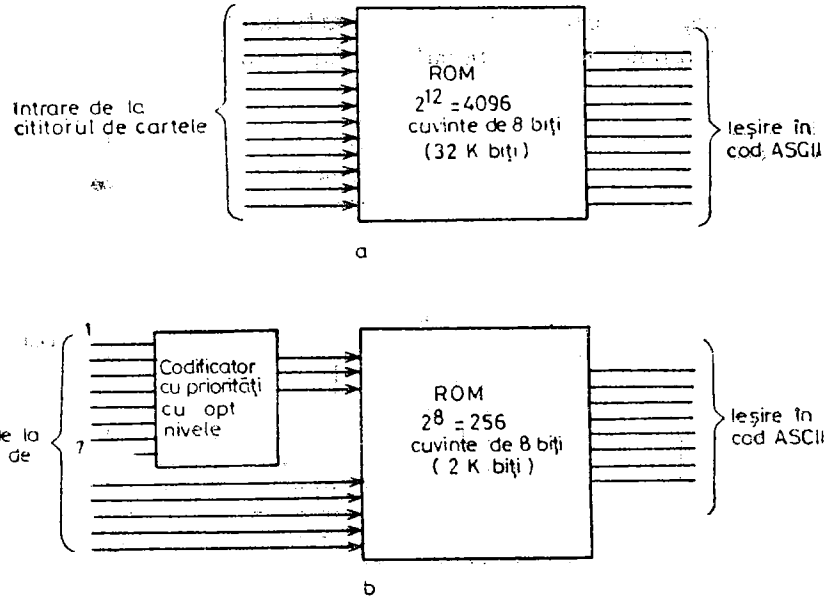


Fig. 4-17. Codificarea intrărilor în memoria fixă (ROM) — convertor pentru cartele perforate; *(a)* fără codificare; *(b)* cu codificare.

de cuvinte sau 256 × 8 = 2 048 biți. În realitate deoarece numai 128 de combinații de cod sînt semnificative codul se poate comprima suplimentar pentru a utiliza o memorie de numai 1 024 de biți. Deci utilizarea logicii de codificare a intrării scade semnificativ dimensiunea memoriei fixe. Totuși, în acest caz, este de remarcat faptul că o memorie mai mare costă mai puțin decît circuitele de codare. Dacă dorim să construim un convertor în sens invers (ASCII-Hollerith) se va folosi, similar, o decodificare la ieșirea ROM-ului pentru generarea primilor șapte biți. În acest caz minimizarea nu va fi atît de mare deoarece dacă *dimensiunea unui ROM este direct proporțională cu numărul de ieșiri ea se dublează pentru ficcare intrare suplimentară.*

Pentru reducerea numărului de intrări într-un ROM se utilizează adeseori multiplexoare. Dacă, spre exemplu, există mai multe moduri de operare și fiecărui mod îi corespunde o anumită configurație de intrare se pot utiliza circuite de selecție (multiplexoare) pentru biții corespunzători configurației de interes în funcție de mod. Dacă modurile de operare nu sînt codificate într-o modalitate direct utilizabilă putem folosi cîteodată pentru generarea adreselor multiplexorului chiar un ROM suplimentar (vezi fig. 4.18). Uneori putem folosi chiar ieșirile ROM-ului pentru selecția modului, dar numai într-o schemă secvențială.

Densitatea funcțională ce se obține prin folosirea ROM-urilor sau PROM-urilor este incomparabil mai mare decît cea obținută în abordarea cu „porți și bistabile". Din acest motiv vom defini inițial orice sistem în ideea că se concentrează cît mai mult din funcțiile combinaționale în ROM-uri. Deoarece putem realiza în această manieră orice funcții, scopul principal în această fază este de a minimiza numărul intrărilor și al ieșirilor circuitelor combinaționale prevăzute a fi implementate cu ROM-uri. Acest mod de gîndire ne conduce către abordarea „microprogramată", unde memoriile fixe sînt folosite pentru a înmagazina ceva ce se apropie foarte mult de programul unui calculator. Aceste probleme vor fi reluate în Capitolul 6.

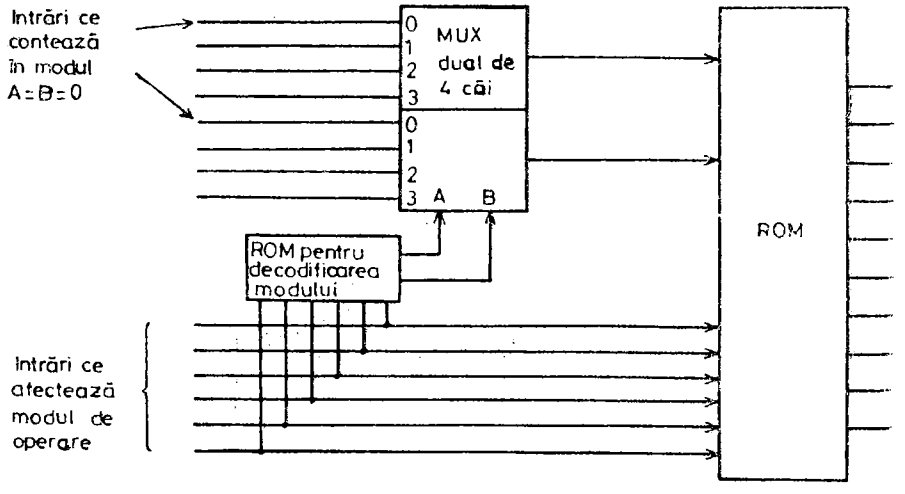


Fig. 4-18. Utilizarea unei memorii fixe (ROM) pentru selectarea intrărilor pertinente.

4.9. MATRICI LOGICE PROGRAMABILE

Matricile logice programabile (programmable logic array-PLA) constituie o soluție la problema codificării intrărilor discutată mai înainte în cazul ROM-urilor. *La un PLA vom specifica nu numai configurația biților fiecărui cuvînt de ieșire, ci și adresa acestuia.* Acest lucru permite un număr mai mare de intrări pentru o dimensiune echivalentă de ROM. Spre exemplu, un PLA de 96 cuvinte × 8 biți = 768 de biți poate avea un număr de 14 intrări*. Dacă toate combinațiile de intrare ar selecta un cuvînt (ca în cazul ROM-ului), atunci cele 14 intrări ar defini 2^{16} = 16 384 de cuvinte sau 16 384 × 8 = = 131 072 biți. Deoarece putem specifica cu 1, 0 sau *X* (nesemnificativ) fiecare din cele 14 intrări pentru fiecare din cele 96 adrese definite, se realizează de fapt o compresie a codului de intrare în zona de decodare de pe cip.

La fel ca și pentru ROM-uri, *aplicațiile PLA-ului pot fi legate de funcția de memorie, de conversia de cod, sau de generarea de funcții logice.* În calitate de *convertor de cod* pentru cartele perforate (vezi exemplul anterior din figura 4-17 *b*) poate fi utilizat un PLA de 96 de cuvinte cu 12 intrări și 8 ieșiri. Fiecărui cod de intrare i se asociază una din cele 96 de adrese programabile generîndu-se pe ieșire codul corespunzător.

Ca memorie circuitul PLA poate înlocui întreaga structură (ROM și multiplexor) din figura 4-18. Deoarece nu sîntem limitați numai la funcții, care reduc numărul de intrări, simple și ușor de implementat, putem realiza implementări extrem de eficiente făcînd uz de redundanța care apare în intrări.

Gîndind PLA-ul ca pe un generator de funcții logice** el poate fi asimilat cu o structură AND/OR, în care cele 96 de adrese sînt decodificate cu 96 de AND-uri cu 14 intrări, iar fiecare ieșire este a unui OR ce poate fi conectat la ieșirea oricăruia din cele 96 de AND-uri. Fiecare ieșire poate fi, deci, definită ca o *sumă de produse de termeni.*

Ca un exemplu simplu poate fi dat PLA-ul de mici dimensiuni din figura 4-19 *a*, ce posedă șase intrări, 13 termeni de produse (cuvinte) și patru ieșiri. Punctele din matricea de sus pot fi considerate intrări în AND-ul, reprezentat de linia verticală, ce calculează produsele necesare funcției. Aceste conexiuni pot fi făcute de la intrări, de la ieșirea inversoarelor conectate la intrări sau pot rămîne neconectate, după specificația utilizatorului. Matricea de mai mici dimensiuni din partea de jos a figurii este matricea OR de la ieșire. Fiecare punct în această matrice reprezintă intrarea unui OR care generează o ieșire. Ambele matrici sînt configurate în funcție de cerințele utilizatorului specificate într-un tabel de adevăr de tipul celui din figura 4-19 *b*. Ca și în cazul ROM-ului aceste tabele de adevăr se transferă pe cartele perforate care sînt folosite apoi pentru generarea automată a unei măști în cadrul procesului de fabricație a PLA-ului.

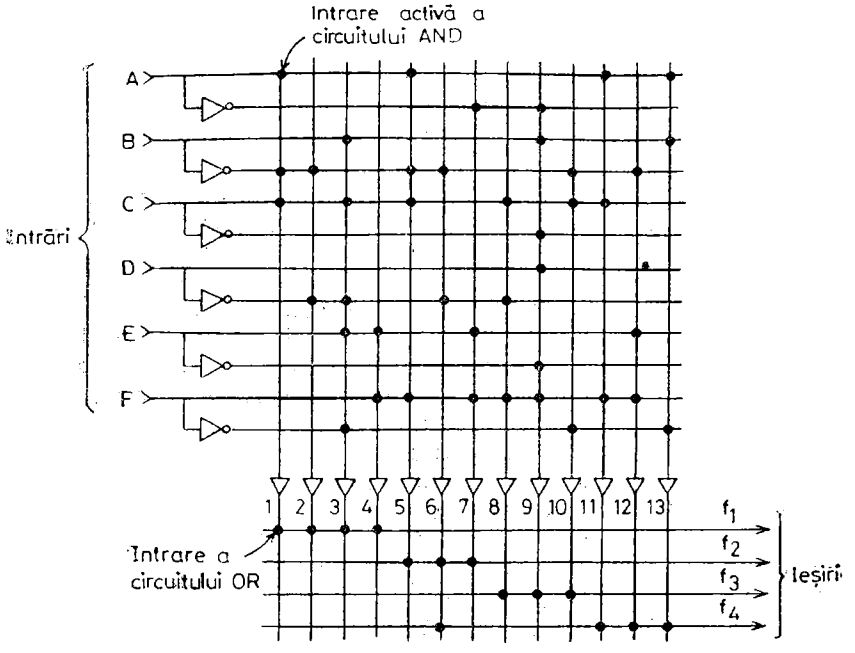
Din figura 4-19 *a* rezultă limpede modul în care structura de matrici conduce la o utilizare extrem de eficientă a suprafeței cipului.

În cadrul circuitelor LSI realizate la comanda utilizatorului un spațiu foarte mare este folosit pentru realizarea interconexiunilor (de exemplu, vezi fig. 7-4) ce definesc particularitatea structurii. Suprafața folosită de cip pentru o poartă este de 100 de ori mai mare decît cea folosită pentru o inter-

---

* Spre exemplu circuitul DM 7575 produs de National Semiconductor.

** Circuitele PLA sînt atît de versatile încît circuitul cu 14 intrări și 8 ieșiri utilizat ca exemplu poate fi programat să realizeze funcția oricărui circuit logic MSI din cele descrise mai înainte în acest capitol.



a

Numărul produsului	Intrări							Ieşiri			
	A	B	C	D	E	F		f_1	f_2	f_3	f_4
1	1	0	1	X	X	X		1	0	0	0
2	X	0	X	0	X	X		1	0	0	0
3	X	1	1	0	1	0		1	0	0	0
4	X	X	X	X	1	1		1	0	0	0
5	1	0	1	X	X	1		0	1	0	1
6	X	0	X	0	X	X		0	1	0	1
7	X	0	X	X	1	1		0	1	0	0
8	X	X	1	0	X	1		0	0	1	0
9	0	1	0	1	0	1		0	0	1	0
10	X	0	1	X	X	0		0	0	1	0
11	1	X	1	X	X	1		0	0	0	1
12	X	0	X	X	1	1		0	0	0	1
13	1	X	1	X	X	0		0	0	0	1

b

Fig. 4-19. (a) Reprezentarea simplificată a unei matrici logice programabile (PLA). *(b)* Tabelul de adevăr asociat pentru comanda unei astfel de matrici.

secție într-o matrice PLA. Desigur și ROM-urile au acest avantaj ; de fapt decodificarea unui cuvînt într-un ROM este realizată exhaustiv cu o matrice asemănătoare cu matricea din figura 4-19 *a* dar cu mai puține intrări.

Pentru a arăta cum trebuie specificat un PLA pentru a implementa funcții logice, vom utiliza PLA-ul „de buzunar" din figura 4-19 pentru a implementa următoarele ecuații :

$$f_1 = A \cdot B' \cdot C + B' \cdot D' + B \cdot C \cdot D' \cdot E \cdot F' + E \cdot F \quad (4.8)$$

$$f_2 = A \cdot B' \cdot C \cdot F + B' \cdot D' + B' \cdot E \cdot F \quad (4.9)$$

$$f_3 = C \cdot D' \cdot F + A' \cdot B \cdot C' \cdot D \cdot E' \cdot F + B' \cdot C \cdot F' \quad (4.10)$$

$$f_4 = A \cdot C \cdot F + B' \cdot E \cdot F + B' \cdot D' + A \cdot C \cdot F' \quad (4.11)$$

Deoarece sumele de produse sînt introduse direct în PLA prin programare, tabelul de adevăr se scrie pornind direct de la ecuațiile logice (vezi fig. 4-19 *b*). Vom începe prin a număra produsele pentru a fi siguri că numărul de produse calculabile pe primul nivel al PLA-ului este suficient. Sînt patru termeni în f_1, trei în f_2, trei în f_3 și patru în f_4. Rezultă un număr de 14 produse, cu unu mai mult decît capacitatea circuitului. Va trebui să utilizăm în comun, dacă este posibil, unele produse pentru mai multe funcții. Observînd că al doilea produs din f_2 și al treilea din f_4 sînt identice ($B' \cdot D'$) vom utiliza acest produs pentru ambele funcții. Putem în acest moment începe implementarea introducînd fiecare termen în ordine în tabelul de adevăr (vezi fig. 4-19 *b*). Pentru primul produs $A \cdot B' \cdot C$, vom scrie în tabel 101XXX, în prima linie, indicînd astfel că intrările primei porți AND se vor conecta la $A \cdot B'$ și C și nu se vor conecta în nici un fel (direct sau prin inversor) la intrările D, E și F (X = nesemnificativ). Un 1 în coloana f_1, pe aceeași linie, indică conectarea ieșirii AND-ului la intrarea primului OR. Se continuă în același fel pînă la completarea tabelului. Punctele marcate în figura 4-19 *a* corespund tabelului de adevăr din figura 4-19 *b*.

În realitate ecuațiile 4—8 pînă la 4—11 se pot implementa numai cu 10 produse, după cum este prezentat în tabelul de adevăr din figura 4-20. Deoarece reducerea numărului de termeni sub valoarea asigurată de PLA-ul utilizat nu oferă nici un avantaj, în cadrul acestui exemplu ne-am oprit la 13 termeni. Sînt situații în care este necesară o reducere mai drastică a termenilor pentru a ne putea încadra în numărul limitat de termeni ai PLA-ului. Reducerea în continuare se poate face aplicînd două procedee. *În primul rînd depistăm produsele identice* ; spre exemplu al doilea termen din f_1 și f_2, și cel de al treilea din f_4 sînt de forma $B' \cdot D'$, deci acest produs nu va fi generat de trei ori.

Numărul produsului	Intrări						Ieșiri			
	A	B	C	D	E	F	t_1	t_2	t_3	t_4
1	1	0	1	X	X	X	1	0	0	0
2	X	0	X	0	X	X	1	1	0	1
3	X	1	1	0	1	0	1	0	0	0
4	X	X	X	X	1	1	1	0	0	0
5	1	0	1	X	X	1	0	1	0	1
6	X	0	X	X	1	1	0	1	0	0
7	X	X	1	0	X	1	0	0	1	0
8	0	1	0	1	0	1	0	0	1	0
9	X	0	1	X	X	1	0	0	1	0
10	1	X	1	X	X	X	0	0	0	1

Fig. 4-20. Tabel de adevăr simplificat.

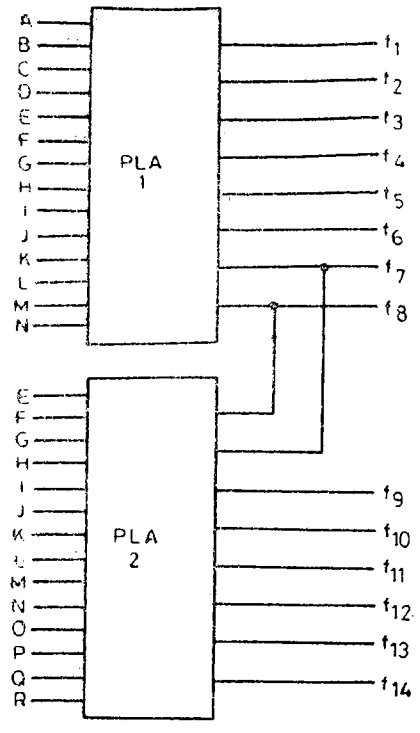


Fig. 4-21. Interconectarea PLA-urilor.

Deci același termen va fi folosit pentru toate cele trei ieșiri, conform figurii 4-20. Dacă în continuare numărul de produse este prea mare, *vom examina fiecare funcție pentru a pune în evidență perechile de termeni ce diferă numai prin negarea unei variabile.* Spre exemplu, primul și ultimul termen f_4 permit următoarea reducere:

$$A \cdot C \cdot F + A \cdot C \cdot F' = A \cdot C(F + F') = A \cdot C \tag{4.12}$$

În cazul acestor circuite trebuie urmărită numai reducerea numărului de termeni ai funcțiilor, fiind inutilă reducerea variabilelor fiecărui produs. De asemenea reducerea numărului de termeni este necesară numai pînă la încadrarea în numărul permis de structura PLA-ului, deoarece reducerea sub această limită nu oferă nici un avantaj suplimentar.

Dacă sînt utilizate mai multe circuite PLA atunci se vor putea implementa funcții logice mai complexe, precum în exemplul din figura 4-21 unde, folosindu-se două PLA-uri cu 14 intrări se generează 14 funcții cu 18 variabile. Examinarea configurației arată că nici unul din produsele utilizate nu conține toate variabilele. Concret, intrările A, B, C, D, O, P, Q și R apar numai în structura unor funcții, iar A, B, C și D nu apar niciodată în produse împreună cu O, P, Q și R. Vor putea fi generate produse ce conțin pe A, B, C și D numai în PLA 1, iar cele ce conțin pe O, P, Q și R numai în PLA 2. Deoarece f_7 și f_8 conțin termeni din ambele grupuri evidențiate anterior, *vor fi conectate împreună cîte două ieșiri de la fiecare* PLA *pentru a forma un* OR *virtual.* Acest fapt va permite utilizarea termenilor construiți în ambele PLA-uri. Celelalte funcții conțin numai termeni ce aparțin strict numai uneia din categorii, deci nu este necesară conectarea altor ieșiri în paralel. Menționăm că acest artificiu poate fi folosit și în cazul schemelor realizate cu ROM-uri (unele variabile apar numai în cîțiva termeni).

Atunci cînd sînt utilizate în implementări de funcții logice mai multe PLA-uri este util ca în același circuit să fie realizate toate funcțiile ce folosesc în comun o serie de termeni, chiar dacă intrările PLA-urilor sînt aceleași. În plus, conectarea în paralel a ieșirilor (OR virtual) poate fi practicată pentru funcțiile cu un număr foarte mare de termeni, care trebuie, în consecință, plasați în circuite diferite.

Ca și în cazul ROM-urilor, este deseori mai economică folosirea decodării anumitor ieșiri cu circuite adiționale. Cu toate că principial aceste tipuri de circuite elimină în mare măsură necesitatea codării intrărilor, fiind capabile să lucreze cu o codare pe intrare foarte redundantă, pot apare situații limită în care este de dorit ca o parte din intrări să fie suplimentar codificate și apoi

conectate la intrarea PLA-ului. În situații favorabile, aproape toate produsele programabile din PLA sînt complet folosite, fiecare definind o configurație de ieșire. Dacă intrările sînt codificate foarte redundant, cum este cazul unei tastaturi necodate, o astfel de utilizare integrală nu este posibilă. Din acest motiv este uneori necesară folosirea unor configurații de multiplexoare de tipul celeia din figura 4-18 pentru intrările în PLA. Cu circuite logice suplimentare de codificare a intrărilor PLA-ului se elimină mult contribuția variabilelor nesemnificative (X-uri) în definirea tabelului de adevăr. Acest lucru face posibilă folosirea acelorași produse (cuvinte) pentru mai multe configurații de intrare și nu a unui cuvînt separat pentru fiecare configurație.

4.10. CONCLUZII

În Capitolele 3 și 4 au fost dezvoltate o gamă largă de tehnici pentru implementarea funcțiilor logice. Fiecare tehnică poate fi gîndită ca un instrument distinct, utilizabil în situații distincte, destul de bine precizate. Orice inginer trebuie să aibă în minte o cît mai largă gamă de artificii și metode pe care să le pună în valoare în funcție de aplicația concretă. Este foarte important ca în fiecare etapă să fie utilizate acele tehnici ce se dovedesc cele mai eficiente. O problemă poate fi rezolvată prin orice metodă, este însă foarte important ca pentru fiecare caz în parte să fie aplicată metoda cea mai eficientă.

Dacă, spre exemplu, trebuie să selectăm una din opt căi, vom putea utiliza porți discrete, sau ROM sau PLA, dar cel mai corect și eficient este să folosim un multiplexor/selector MSI. De asemenea, dacă dorim să implementăm o serie de funcții neuzuale și complexe cel mai bine este să ne orientăm către circuite ROM sau PLA. Funcții logice simple și cele nestandard dar simple vor presupune porți discrete. Pentru fiecare tip de circuit există o clasă de funcții cea mai potrivită.

Chiar dacă vom orienta proiectarea de sistem către utilizarea unor circuite cu un nivel înalt de integrare, vor rămîne întotdeauna mici probleme legate spre exemplu, de unele aspecte de interconectare, care vor presupune folosirea porților discrete. *Pentru a determina care este limita pînă la care putem utiliza implementări cu porți logice SSI este foarte util Tabelul 4-5.* Putem utiliza informația oferită de acest tabel pentru a compara numărul de capsule integrate ce rezultă prin implementarea cu porți, cu cel indicat pentru alte procedee.

Tabelul 4-5. **Numărul de capsule de circuite integrate necesare pentru implementarea funcțiilor logice (rezumat)**

Numărul variabilelor	Capsule echivalente	Structura
2 (oricare 4 funcții)	1	Multiplexor cuadruplu de 2 biți
3 (oricare 2 funcții)	1	Multiplexor dual de 4 biți
4	1	Multiplexor de 8 biți
5	2	Multiplexor de 16 biți
6	2 pînă la 4 1/2[a]	Multiplexor de 16 biți + porți reziduale
7	4 1/3 pînă la 6 1/3[a]	Multiplexor de 32 biți + porți reziduale
8	9 1/2 pînă la 11 1/2	Multiplexoare de 64 biți + porți reziduale

Tabelul 4-5 (continuare)

Numărul variabilelor	Capsule echivalente	Structura
8 (oricare 4 funcții)	1 (cost ≈ 2)[b]	1024 biți (256×4) PROM sau ROM
8 (oricare 8 funcții)	2[a] (cost ≈ 2)[b]	2048 biți (256×8) PROM sau ROM
9 (oricare 4 funcții)	1 (cost ≈ 2)[b]	2048 biți (512×4) PROM sau ROM
9 (oricare 8 funcții)	2[a] (cost ≈ 2)[b]	4096 biți (512×8) PROM sau ROM
10 (oricare 4 funcții)	2[a] (cost ≈ 2)[b]	4096 biți (1024×4) PROM sau ROM
10 (oricare 8 funcții)	2[a] (cost ≈ 2)[b]	8192 biți (1024×8) PROM sau ROM
11 (oricare 4 funcții)	2[a] (cost ≈ 2)[b]	8192 biți (2048×4) PROM sau ROM
11 (oricare 8 funcții)	2[a] (cost ≈ 3)[b]	16384 biți (2048×8) PROM sau ROM
12 (oricare 4 funcții)	2[a] (cost ≈ 3)[b]	16384 biți (4096×4) PROM sau ROM
12 (cel mult 12 funcții)	2[a] (cost ≈ 3)[b]	FPLA sau PLA cu 12 intrări și 12 ieșiri
13 (cel mult 10 funcții)	2[a] (cost ≈ 3)[b]	FPLA sau PLA cu 13 intrări și 10 ieșiri
14 (cel mult 8 funcții)	2[a] (cost ≈ 3)[b]	FPLA sau PLA cu 14 intrări și 8 ieșiri
17 (cel mult 10 funcții)	4[a] (cost ≈ 3)[b]	FPLA sau PLA cu 17 intrări și 10 ieșiri

[a] O capsulă cu 24 de terminale ocupă locul a două capsule cu 16 terminale.
[b] În baza costului pentru loturi de 100 de bucăți în 1978.

EXERCIȚII

1. (*a*) Implementați următoarea funcție de patru variabile folosind numai un multiplexor de opt căi:

$$f = m_0 + m_3 + m_5 + m_6 + m_9 + m_{10} + m_{12} + m_{15}$$

(*b*) Implementați aceeași funcție, în cea mai simplă formă, folosind logica de tip NAND/NAND.
(*c*) Comparați numărul de capsule folosit în cele două cazuri.

2. (*a*) Implementați următoarea funcție de patru variabile folosind numai un multiplexor de opt căi:

$$f = m_0 + m_1 + m_2 + m_3 + m_{11} + m_{12} + m_{14} + m_{15}.$$

Intrări				Ieșiri			
A	B	C	D	f_1	f_2	f_3	f_4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1

Fig. 4-22.

(*b*) Implementați aceiași funcție, în cea mai simplă formă, folosind logica de tip NAND/NAND.
(*c*) Comparați numărul de capsule integrate folosite în cele două cazuri.

3. (*a*) Implementați un convertor din **cod Gray** în BCD, conform definirii din tabelul de adevăr din figura 4-22, folosind două multiplexoare duble de patru căi și porți suplimentare. (Ieșirile pentru codurile ce lipsesc din tabelul de adevăr sînt nesemnificative; utilizați diagramele Veitch).
(*b*) Implementați aceeași funcție cu patru multiplexoare de 8 căi.
(*c*) Implementați aceeași funcție cu patru multiplexoare de două căi plus porți pentru funcțiile reziduale.
(*d*) Implementați aceeași funcție cu porți logice NAND.

4. (*a*) Implementați decodificatorul de șapte segmente având la intrare cod biquinar (ca în Exercițiul 2 de la Capitolul 3) utilizînd patru multiplexoare de 8 căi.
(*b*) Cu două multiplexoare duale de patru biți și porți suplimentare.

(c) Cu patru multiplexoare de două căi şi porţi reziduale.
(d) Cu un decodificator zecimal şi porţi NAND.
(e) Comparaţi numărul de capsule rezultate.

5. (a) Implementaţi un codor de priorităţi zecimal (Exerciţiul 5, Capitolul 3) cu patru multiplexoare de 8 căi.
(b) Cu două multiplexoare duale de patru căi şi porţi suplimentare.
(c) Cu patru multiplexoare de două căi şi porţi reziduale.
(d) Cu un decodificator zecimal şi porţi NAND.
(e) Comparaţi numărul de capsule folosite pentru diverse soluţii.

6. (a) Implementaţi funcţiile f_2 şi f_4 din Exerciţiul 7, Capitolul 3, cu două multiplexoare de 8 căi.
(b) Cu un multiplexor dual de patru căi.

7. (a) Implementaţi funcţia descrisă în Exerciţiul 9, Capitolul 3, cu patru multiplexoare de 8 căi.
(b) Cu două multiplexoare duale de patru căi.
(c) Cu patru multiplexoare de două căi.
(d) Comparaţi numărul de capsule folosite.

8. Implementaţi funcţia descrisă în Exerciţiul 12, Capitolul 3, cu un multiplexor cu 8 intrări.
9. Implementaţi funcţia descrisă în Exerciţiul 13, Capitolul 3, cu un multiplexor cu 8 intrări.
10. (a) Implementaţi funcţia de la Exerciţiul 15, Capitolul 3, cu un multiplexor de 16 căi.
(b) Cu un multiplexor de 8 căi şi porţi pentru reziduuri.
11. Implementaţi funcţia din Exerciţiul 22, Capitolul 3, cu un multiplexor de 8 căi.
12. Construiţi diagrame, similare celor din figura 4-4, pentru patru căi posibile de reducere a funcţiilor de patru variabile utilizînd un multiplexor de 8 căi.
13. (a) Implementaţi un convertor BCD — cod Gray (prin schimbarea coloanelor de intrare cu cele de ieşire în tabelul de adevăr din Exerciţiul 3) folosind patru multiplexoare de 8 căi.
(b) Cu două multiplexoare duale de patru căi şi porţi porţi suplimentare.
(c) Cu patru multiplexoare de două căi şi porţi porţi suplimentare.
(d) Cu decodificator zecimal şi porţi NAND.
(e) Cu logica de tip NAND/NAND.
(f) Comparaţi numărul de capsule integrate rezultate în fiecare implementare.

14. (a) Implementaţi convertorul de la şapte segmente la BCD (comutînd intrările cu ieşirile în tabelul de adevăr din figura 3-2 *d*) folosind patru multiplexoare de 8 căi şi porţi pentru funcţiile reziduale (construiţi patru diagrame Veitch). Dacă apar simplificări notabile folosiţi şi pentru funcţiile reziduale multiplexoare.
(b) Ce dimensiune ar trebui să aibă un ROM ce implementează această funcţie (dimensiunea ce rezultă din calcul, ignorînd tipurile standard existente)?
(c) Ce dimensiune de PLA este necesară?
(d) Construiţi tabelul de adevăr pentru PLA.

15. Desenaţi un multiplexor de 16 căi folosind două multiplexoare de 8 căi şi o poartă NAND cu două intrări.
16. Desenaţi un multiplexor cu 40 de intrări folosind o reţea arborescentă de multiplexoare cu 8 intrări.
17. Desenaţi un demultiplexor pe 40 de căi utilizînd o reţea arborescentă cu şase decodificatoare zecimale.
18. (a) Desenaţi un decodificator cu 40 de ieşiri folosind cinci decodificatoare zecimale.
(b) Cu patru decodificatoare zecimale şi patru NAND-uri cu două intrări.
19. Proiectaţi un multiplexor cu 16 384 de intrări folosind matrici $X-Y$, cu submultiplexare pe 1 024 de module, fiecare cu multiplexoare de 16 căi cu ieşire tristate. Multiplexorul central are două decodificatoare zecimale; şi nouă multiplexoare de 8 căi, dispunînd de 100 de conexiuni externe.
20. Construiţi, pentru un decodificator/demultiplexor matricial cu 2 048 de ieşiri, următorul tabel pentru una, două, trei şi patru dimensiuni egale:

Numărul de dimensiuni a matricii	1	2	3	4
Mărimea fiecărei dimensiuni				
Numărul de capsule cu porţi (total)				
Numărul de capsule pentru precodare				
Numărul de conexiuni de la precodare				
Încărcarea ieşirilor precodorului				

21. (*a*) Construiţi un tabel similar pentru cazul în care dimensiunea X este de două ori mai mare decît dimensiunea Y.
    (*b*) Pentru o matrice cu 3 dimensiuni în raporturile 2 : 2 : 1.
22. Construiţi un circuit cu opt intrări şi opt ieşiri care are una din cele opt ieşiri pe zero în corespondenţă cu cea mai prioritară intrare pe zero. Utilizaţi numai două din circuitele speciale discutate în acest capitol.
23. (*a*) Construiţi tabelul de adevăr pentru un ROM (nerealist de mic) ce converteşte BCD — şapte segmente.
    (*b*) Care este mărimea, în biţi, a ROM-ului ?
    (*c*) Desenaţi schema unui ROM, similar celui din figura 4-19 *a*, folosind puncte pentru indicarea celor $2n$ cuvinte decodificate şi de asemenea puncte pentru specificarea cuvintelor de ieşire.
24. Ce mărime de ROM este necesară pentru a construi :
    (*a*) Patru multiplexoare de două căi ?
    (*b*) Două multiplexoare de patru căi ?
    (*c*) Un codificator cu prioritate pentru opt intrări, cu trei ieşiri ?
25. Ce mărime trebuie să posede un PLA pentru a realiza :
    (*a*) Patru multiplexoare de două căi ?
    (*b*) Două multiplexoare de patru căi ?
    (*c*) Un codificator cu prioritate pentru opt intrări cu trei ieşiri ?
    (Includeţi numărul de intrări, de ieşiri şi de produse).
26. (*a*) Cîţi biţi are memoria ROM din figura 4-18 ?
    (*b*) Ce număr de biţi ar fi necesari în cazul în care nu ar fi folosită în schemă o preselectare cu multiplexor ?

BIBLIOGRAFIE

Carr, W. N şi J. P. Mize, *MOS/LSI Design and Application*, McGraw-Hill, New York, 1972, Capitolul 8, „Programmable Logic Arrays", Capitolul 7 „Memory Applications".

Fairchild Semiconductor Staff, *The TTL Applications Handbook*, Fairchild Semiconductor, Mountain View, Calif., 1973.

Intel Staff, *The Intel Memory Design Handbook*, Intel Corp., Santa Clara, Calif., 1973, Capitolul 2, „Read Only Memories".

National Semiconductor, *How to Design With Programmable Logic Arrays*, Application Note # AN-89, National Semiconductor Corp, Santa Clara, Calif., 1973.

Texas Instruments Staff, *The TTL Data Book*, Texas Instruments, Dallas, Tex., 1973.

Uimari, David, „PROM's-a Practical Alternative to Random Logic", *Electronic Products*, January 21, 1974, pag. 75—91.

Priel, Vry, şi Phil Holland, „Aplication of a High Speed Programmable Logic Array", *Computer Design*, December 1973, pag. 34—96.

Yau şi Tang, „Universal Logic Modules and Their Application", *IEEE Transactions on Computers*, Vol. C-19, No. 2, February 1970.

5

LOGICA SECVENȚIALĂ

Proiectare

5.1. ELEMENTE DE MEMORARE

În Capitolul 4 am văzut că logica combinațională și memoria fixă (ROM) sînt de fapt forme diferite ale aceluiași lucru. Un circuit AND cu opt intrări, spre exemplu, poate fi gîndit ca o memorie ROM de $2^8 = 256$ cuvinte $\times$ $\times$ 1 bit astfel programată încît să aibă ieșirea falsă pentru toate combinațiile de intrare exceptînd una singură, corespunzînd aplicării lui 1 logic pe toate intrările. Din punct de vedere funcțional porțile și ROM-urile constituie același lucru.

În acest capitol vom introduce conceptul de memorie cu conținut variabil. Dacă am putea modifica structura ROM-ului în așa fel încît *să putem schimba configurația biților după dorință*, am putea utiliza acest circuit pentru memorare. Memoria PROM este modificabilă dar nu așa ușor ca o adevărată memorie activă de tip RAM* (read/write RAM). Pentru a modifica conținutul RAM-ului, trebuie generată la intrare configurația de biți dorită și un *impuls pe intrarea de scriere (write clock)* ; cuvîntul adresat va fi modificat imediat. Capacitatea de a memora este, desigur, foarte importantă în soluționarea unor clase largi de probleme concrete. Multe probleme presupun realizarea unor operații combinaționale asupra unor date de intrare, dar numai o mică parte folosesc datele de intrare numai în momentul apariției lor. Funcția de memorare ne permite să reținem datele pînă în momentul în care este necesară prelucrarea lor împreună cu alte date de intrare. De asemenea, deoarece RAM-urile pot fi utilizate precum ROM-urile în probleme de funcții logice, vom putea rezolva probleme diferite cu aceeași structură hardware, putînd face, prin aceasta, ca hardware-ul să se adapteze el însuși diverselor probleme.

Așa cum avem poarta ca element combinațional de bază și ROM-ul pentru rețele combinaționale complexe, așa vom avea bistabilul ca element de memorare (RAM de un bit) și RAM-ul pentru capacități mari de memorare. În esență, porțile și ROM-urile corespund bistabilului și memoriei RAM, dar bistabilii și memoriile active posedă în plus și dimensiunea *temporală*. Dispozitivele din această categorie au toate o intrare de ceas (tact) și *starea lor se schimbă numai drept consecință a tranziției acestuia.*

---

* De fapt și ROM-urile și RAM-uri (random-acces memories) sînt memorii cu acces aleator, dar prin RAM se înțelege uzual o memorie activă de tip *citește/scrie*.

5.2. TIPURI DE BISTABILI

În baza corespondenței dintre elementele de memorie și circuitele combinaționale elementare nu trebuie să surprindă utilizarea în procedeul de proiectare a tabelelor de adevăr, a diagramelor Veitch și a ecuațiilor logice. De fapt, vom defini circuitele elementare de memorie utilizînd un formalism similar aceluia folosit pentru circuitele combinaționale de bază. Figura 5-1 prezintă circuitele bistabile de bază într-o configurație utilizată pentru definirea porților similară celei din figura 3-3.

În toate cazurile ieșirea este definită prin ecuații logice și tabele de adevăr. Elementele de memorare își pot modifica ieșirea (conținutul) în funcție de intrări *numai ca urmare a tranziției ceasului*. În consecință, este utilizată o scriere cu indici superiori pentru a indica faptul că ieșirea la momentul $n+1$ este o consecință a intrării din momentul anterior n.

Pentru bistabilul de tip D (delay), ecuația este foarte simplă, și ne spune faptul că intrarea (D) *este „stocată" în bistabil la apariția impulsului de ceas*,

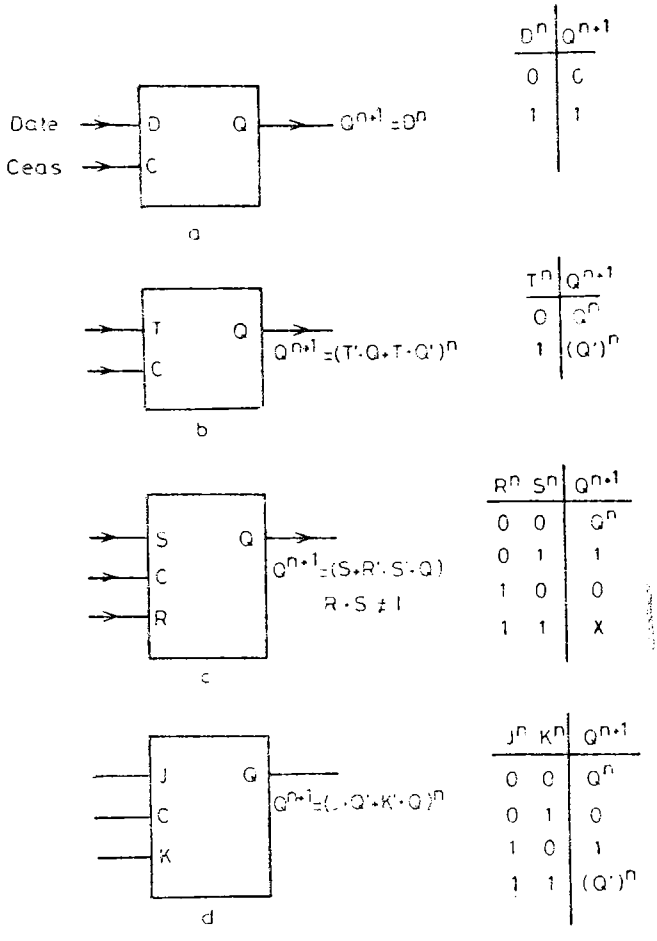


Fig. 5-1. *(a)* Bistabil de tip D (delay). *(b)* Bistabil de tip T (toggle). *(c)* Bistabil de tip R-S (reset-set). *(d)* Bistabil de tip J-K.

fiind generată la ieșire pe perioada următoare $(n+1)$ *a ceasului*. Bistabilul de tip D se comportă deci foarte asemănător cu un RAM de un bit. Este deosebit de util pentru stocări de date și alte aplicații speciale.

În figura 5-1 mai sînt definite trei tipuri de bistabili, ca elemente de memorare de un bit. Aceste tipuri de bistabili nu sînt însă numai simple elemente de memorare, starea lor modificîndu-se după diferite reguli în funcție de intrări. Utilitatea acestor tipuri de bistabili este datorată simplificării funcțiilor combinaționale de comandă a intrărilor în diverse aplicații secvențiale.

Bistabilul T (toggle), spre exemplu, nu-și modifică starea anterioară atît timp cît intrarea T este falsă înainte de aplicarea ceasului. Dacă T este adevărat, ieșirea se modifică trecînd în starea complementară, odată cu impulsul de ceas. Bistabilul de tip T este util în aplicațiile de numărătoare binare, unde fiecare bit trebuie să comute de fiecare dată cînd este semnalată depășirea de la ordinul binar anterior (mai puțin semnificativ). Dacă vom compara ecuația ce definește bistabilul T cu cea pentru bistabilul D, vom remarca faptul că bistabilul T poate fi construit pornind de la un bistabil D și adăugînd cun XOR conectat ca în figura 5-2. De fapt, *orice tip de bistabil poate fi construit pornind de la bistabilul de tip D adăugînd circuite combinaționale adecvate*, conform ecuațiilor din figura 5-1.

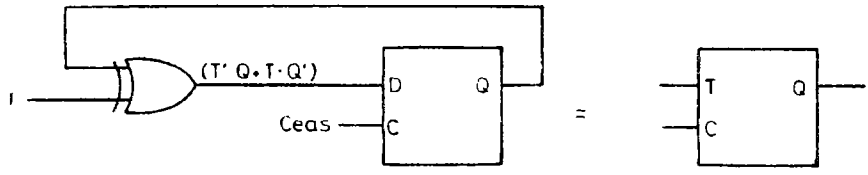


Fig. 5-2. Realizarea unui bistabil de tip T.

Bistabilul R—S (reset-set) este setat ($Q = 1$) ca urmare a trecerii intrării S în 1 logic și este resetat ($Q = 0$) după activarea intrării R. Tranziția este indefinită dacă sînt activate ambele intrări. Este posibilă definirea unor tipuri de bistabili speciali pentru înlăturarea acestui efect neplăcut. Pentru ambele intrări, R și S, active astfel de bistabili trec forțat în starea $Q = 1$ (SOR — Set Overrides Reset) sau în starea $Q = 0$ (ROS — Reset Overrides Set).

Bistabilul J—K (figura 5-1*d*) este setat după ce $J = 1$ și resetat după ce $K = 1$. Este deci similar bistabilului R—S cu deosebire că *pentru* $J = K = 1$ *ieșirea se complementează* (ca la bistabilul de tip T). Acest bistabil poate fi folosit ca un bistabil de tip T prin simpla interconectare a celor două intrări J, K. Deoarece bistabilul de tip J—K poate îndeplini ușor și funcțiile bistabilelor R—S și T, acestea din urmă nu sînt construite în mod uzual. În mod curent dispune de bistabili de tip J—K pentru numărătoare de capacitate mică și de bistabili de tip D pentru aplicații ce presupun memorări. De altfel, bistabilul D poate fi obținut dintr-un bistabil J—K interconectînd intrările acestuia printr-un inversor, ca în figura 5-3.

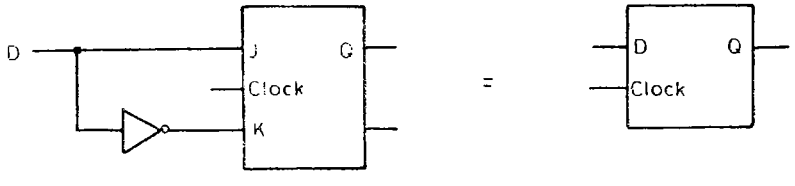


Fig. 5-3. Realizarea unui bistabil de tip D.

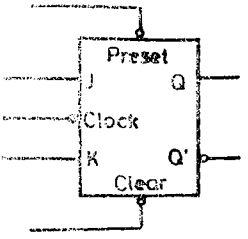


Fig. 5-4. Bistabil de tip J-K.

Bistabilele integrate sînt ceva mai complicate decît cele din figura 5-1. Bistabilele D și J—K sînt incapsulate cîte două într-un cip. De asemenea posedă intrări *asincrone* pentru *PRESET* ($Q = 1$) și **CLEAR** ($Q = 0$) ce determină setarea sau resetarea necondiționată de către impulsurile de ceas. Aceste intrări sînt folosite numai pentru unele aplicații, unde sînt deosebit de importante condițiile inițiale ale circuitului pentru a elimina „condițiile de blocare" discutate în Capitolul 6. Ca și în cazul porților logice, un cerculeț pe o intrare semnifică faptul că semnalul este activ pe zero. *În cazul bistabilelor, însă, nu este vorba de o convenție arbitrară, așa cum a fost în cazul porților.*

Spre exemplu, la bistabilul prezentat în figura 5-4 intrările CLEAR și PRESET sînt active pe zero, generînd comutarea bistabilului nu în funcție de convenția pe care o facem pentru nivelele logice. De asemenea, cerculețul pe intrarea de ceas indică faptul că bistabilul va comuta în funcție de *tranzitia negativă* a impulsului de ceas, indiferent de convenția făcută pentru nivelele logice.

(Bistabilele din figura 5-1 comută toate, pe frontul pozitiv al ceasului deoarece nu este desenat un cerculeț pe intrarea de tact).

Dacă bistabilele vor fi desenate întotdeauna cu intrările și ieșirile ca în figura 5-4, schemele logice vor fi mai inteligibile. Menționăm că majoritatea bistabilelor au și o ieșire inversată, Q', care se desenează în dreapta jos. Deoarece intrările de ceas ale tuturor bistabilelor dintr-un sistem sînt întotdeauna conectate la ceasul sistemului, adesea vor fi omise din reprezentare pentru a nu încărca inutil desenul. Conexiunile de alimentare (masă și sursă) sînt de asemenea omise în reprezentările uzuale.

Un alt element de memorie, denumit *latch (bistabil asincron)*, este descris în foile de catalog printr-un tabel de adevăr ca acela al bistabilului D din figura 5-1*a*. Acest bistabil diferă însă de unul de tip D, deoarece modificarea ieșirii se face pe toată durata palierului definitit activ al impulsului de ceas și nu numai sincron cu unul din fronturile tactului. În intervalul de timp în care semnalul de ceas este activ ieșirea urmărește orice modificare a intrării. Din acest motiv, *bistabilul de tip latch (asincron) nu poate fi utilizat în structurile* **sincrone** *ce vor fi prezentate ulterior.*

5.3. REGISTRE ȘI NUMĂRĂTOARE

Bistabilele sînt deseori utilizate grupate în *registre* sau *numărătoare*. Un *registru* este gîndit ca un dispozitiv de *stocare de date*. De multe ori conținutul său poate avea o semnificație fizică, cum ar fi de exemplu cea de cod pentru un caracter sau de număr folosit pentru diverse calcule. Registrul propriu-zis nu conține, în esență „logică" nefiind altceva decît o grupare de circuite bistabile. De obicei la intrarea registrului se folosesc circuite combinaționale pentru selectarea sursei de date care urmează a fi stocate, pentru realizarea unor conversii de cod, sau pentru funcții aritmetice.

De asemenea bistabilele pot forma o configurație de *numărător*. Numărătoarele sînt folosite pentru controlul secvențării operatiilor. Ieșirile bistabilelor ce formează numărătorul sînt interpretate împreună ca *stare* a numără-

torului. Succesiunea de stări poate fi directă, liniară, formată prin numărarea unor impulsuri de ceas, poate fi repetitivă, ciclică, sau poate fi foarte complexă. Secvențele complexe sînt, de regulă, controlate în diverse puncte, prin condiții de intrare variabile. În toate cazurile, numărătorul poate fi utilizat cu o logică combinațională suplimentară pe intrările fiecărui bistabil. Fiecare stare $(n+1)$ va fi definită de intrările (n) anterioare aplicării ceasului. Avem deci o problemă cu două etape:

1. Definirea numărătorului, cerut de sistem și a secvenței de stări necesare.

2. Proiectarea circuitelor logice combinaționale necesare pe intrările bistabilelor pentru realizarea secvenței de stări dorite.

Așa cum am arătat și mai înainte, partea cea mai creativă a proiectării se află în baza sa inițială. Odată definită structura generală a numărătorului în sistem și stările acestuia, urmează o etapă, caracterizată prin mai puțină creativitate, în care este găsită prin metode de rutină rețeaua logică combinațională asociată numărătorului.

Un instrument important ce poate fi utilizat la nivelul proiectării de sisteme este ideea de a fragmenta un sistem complex în subsisteme mai mici. Numărătoarele și registrele din sistem pot fi reprezentate ca simple blocuri într-o schemă de ansamblu. Aceste blocuri pot fi astfel concepute încît acolo unde este posibil să corespundă unor circuite MSI ; chiar dacă sînt folosite circuite bistabile discrete, blocurile vor fi denumite corespunzător funcției ce o execută. În loc de a concepe un singur numărător controlor de secvențe, în mod uzual este posibilă *fragmentarea problemei după cîteva „dimensiuni"* — fiecare controlată de un numărător separat.

De obicei numai unul din numărătoare are un rol mai semnificativ în evoluția secvențelor. Acest numărător este de regulă numit *numărător de stare* și poate fi reprezentat în schemă ca un simplu bloc. Celelalte numărătoare pot fi în general numărătoare binare simple, resetabile. Singura cale de a determina secvențele ce pot avea mai multe „dimensiuni" este de a analiza global problema și de a evidenția unele similitudini.

Spre exemplu, dacă sistemul manipulează bit cu bit, „cuvinte" de date, *numărătorul de cuvînt* poate determina o dimensiune iar *numărătorul pe bit* o altă dimensiune. Dacă sînt executate cîteva operații diferite asupra datelor în aceeași secvență, atunci un *numărător de stare* poate coordona toate aceste secvențe. Controlul secvenței rezultate va fi divizat în *trei dimensiuni* (vezi fig. 5-5) manipulate de către trei numărătoare: numărătorul de bit, numărătorul de cuvînt și numărătorul de stare. Pe lîngă faptul că problema se împarte în trei probleme mai simple, această divizare aduce avantajul suplimentar al economisirii de hardware (similară aceleia realizate pentru decodificatoare în Secțiunea 4.4). După cum este de așteptat un controlor de stări determină realizarea unor operații logice. În mod obișnuit aceste operații afectează în egală măsură toți biții cuvîntului. Ca urmare *logica ce comandă aceste operații nu este strict condiționată de numărătorul de stare pentru biți și, deci, îl poate ignora.* Sistemul mai conține și circuite de comandă pentru scrierea sau citirea corectă a biților din memorii sau registre. Această logică este independentă de tipul operațiilor logice efectuate pe care, deci, le poate ignora. *Divizînd problema pe mai multe dimensiuni, logica combinațională se reduce în diferite părți ale sistemului la funcții de cîteva variabile.* O astfel de logică va fi mai simplă și mai ușor de implementat.

Abordarea secvențării pe mai multe dimensiuni are încă o consecință: apare posibilitatea micșorării numărului de bistabili necesari. Va rezulta o

codificare mutual exclusivă în grupuri de posibilități, similară codificării mutual exclusive a intrărilor ROM-ului din Secțiunea 4-16. Așa cum în Secțiunea 4-16 a rezultat o micșorare a numărului de intrări în ROM, în acest caz va rezulta un număr mai mic de bistabili. La fel ca și în cazul intrărilor în ROM, în mod ideal am dori să avem un mod de operare logic diferit pentru fiecare combinație a stărilor bistabililor din sistem. Dacă vom avea definite cîteva numărătoare, aceasta va însemna că starea fiecărui numărător este semnificativă numai pentru unul din modurile de operare. Ca urmare, întotdeauna trebuie să încercăm să folosim același numărător pentru mai multe funcții, cu toate că această metodă poate să nu fie practică în toate situațiile.

Ca un exemplu de utilizare multiplă a unui numărător, să presupunem că sistemul din figura 5-5 are la intrare cîteva cuvinte, calculează cu ele niște funcții matematice și generează la ieșire alte cuvinte. Dacă executarea calculului presupune trei pași, numărătorul de stare va genera următoarele stări :

Intrarea datelor
Pasul 1 al calculului
Pasul 2 al calculului
Pasul 3 al calculului
Ieșirea datelor.

Numărătorul de stare va necesita trei bistabili pentru reprezentarea celor cinci stări. Deoarece numărătorul de cuvînt nu are semnificație pe durata celor trei stări ce comandă calculul, el este de fapt prost utilizat. O soluție mai bună constă în *a utiliza numărătorul de cuvînt pentru comanda procesului de calcul ;* în acest fel se vor putea elimina două din cele trei stări folosite pentru calcul. Numărătorul de stare va avea, în consecință, numai trei stări : intrare, calcul, ieșire.

Aceste tipuri de numărătoare, folosite pentru control, pot avea cele mai diferite utilizări. Spre exemplu, executarea instrucțiunilor într-un computer este de multe ori controlată de numărătoare de „minim" și „maxim". Utilizarea exactă a acestor numărătoare depinde de instrucțiunea în curs de execuție ; ceea ce este important este faptul că reprezintă două dimensiuni ale executării instrucțiunilor, permițînd obținerea unor avantaje din asemănarea ce există între diferitele instrucțiuni.

Putem extinde cu încă un pas analogia, dintre micșorarea numărului bistabilelor de control și codificarea intrărilor ROM-ului, observînd faptul că utilizarea multiplă a aceluiași numărător este similară artificiului folosit în figura 4-18 ce permite aplicarea diferitelor semnale pe aceeași intrare a ROM. Desigur, utilizarea multiplă a acelorași bistabile, necesită și un număr de circuite logice adiționale. De obicei numărul de componente combinaționale discrete din circuit crește pe măsură ce stările de control sînt codificate mai

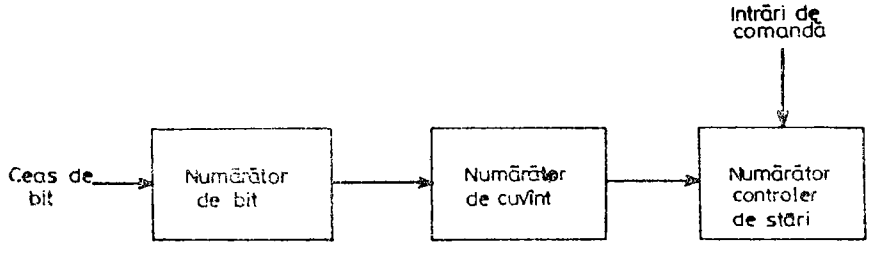


Fig. 5-5. Numărătoare conectate în cascadă.

compact. Din acest motiv trebuie să fim siguri că atunci cînd am scăpat cu multă dibăcie de un bistabil, n-am adăugat de fapt mai mult în logica combinațională ieșind pînă la urmă în pierdere. În plus trebuie să ținem cont și de faptul că numărătoarele MSI conțin cîte patru bistabili într-o capsulă. Din acest motiv nu există o diferență esențială între costul, de exemplu, al unui numărător cu 16 stări și costul unuia cu cinci stări.

5.4. PROIECTAREA NUMĂRĂTOARELOR CU BISTABILE

Cu toate că există circuite MSI pentru numărarea într-o succesiune binară și pentru alte cîteva tipuri de secvențe, de multe ori este de dorit proiectarea unor numărătoare speciale utilizînd bistabile și porți logice. Spre exemplu, secvența de numărare prezentată în figura 5-6*a* este cîteodată foarte utilă

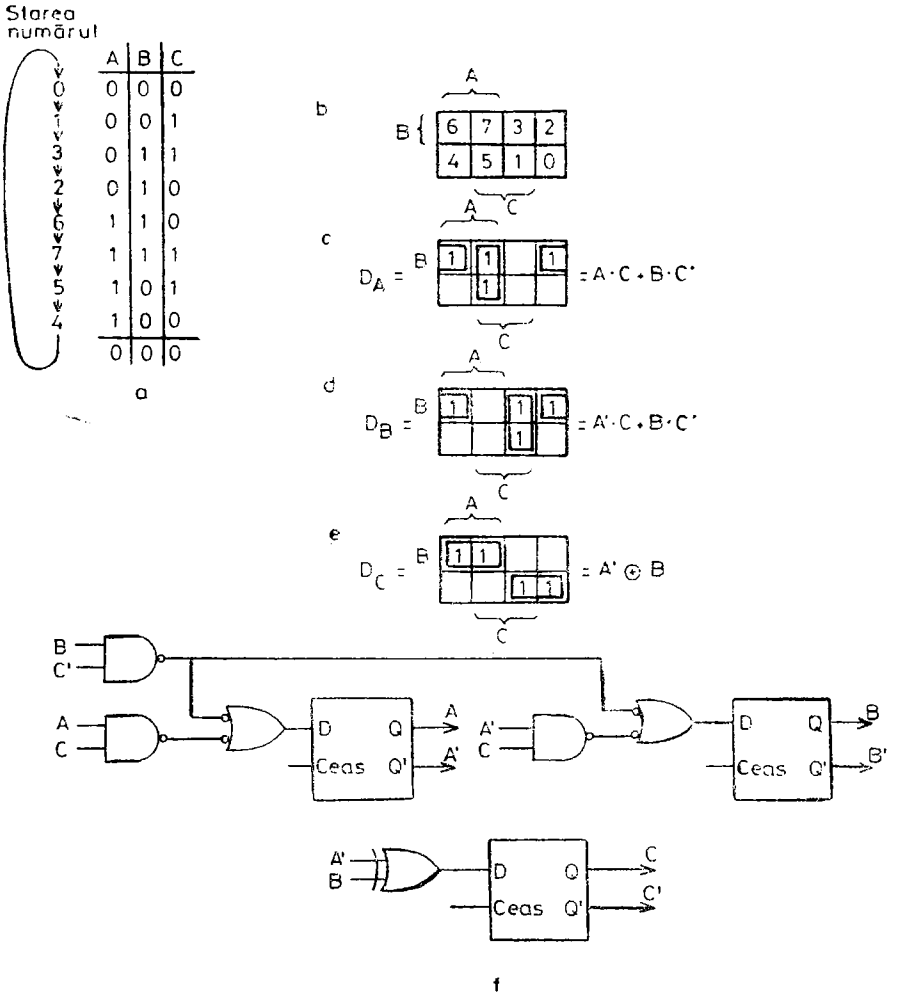


Fig. 5-6. *(a)* Numărător în cod Gray reflectat. *(b)* Numerotarea stărilor (pentru referință). *(c)* Diagrama pentru D_A. *(d)* Diagrama pentru D_B. *(e)* Diagrama pentru D_C *(f)* Implementarea cu bistabili de tip D.

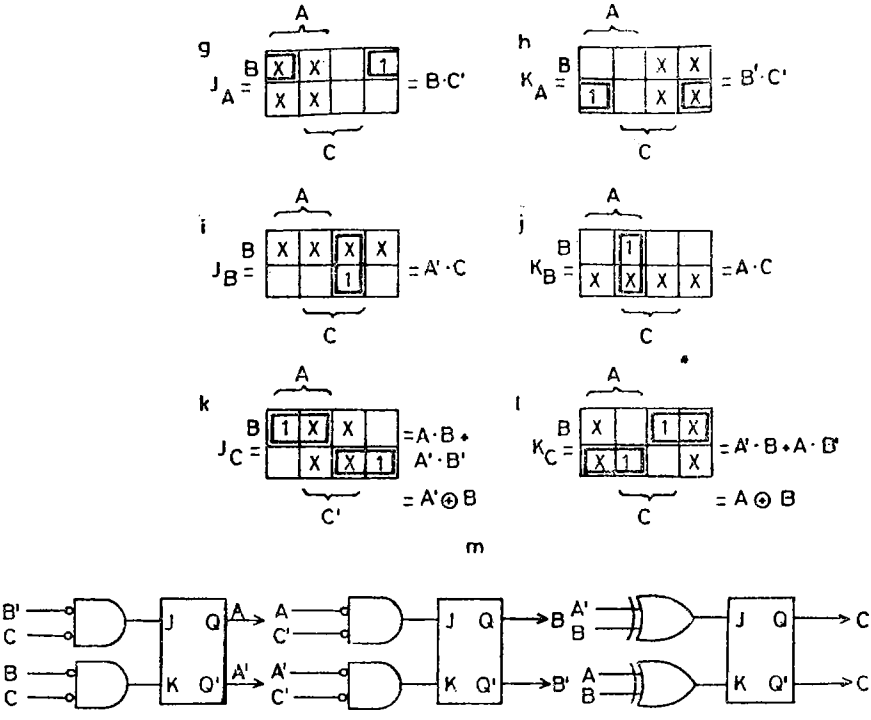


Fig. 5-6. (continuare). *(g—l)* Proiectarea cu bistabili de tip J-K. *(m)* Implementarea cu bistabili de tip J-K.

deoarece la fiecare tranziție se modifică doar un singur bit. Proiectarea numărătorului ce generează o astfel de secvență presupune obținerea ecuațiilor logice pentru intrările bistabililor A, B și C.

Dacă alegem pentru implementarea acestui numărător bistabile de tip D, trebuie să ținem cont de ecuația din figura 5-1*a* conform căreia impulsul de ceas din momentul n determină ieșirea în intervalul de timp $n+1$. Va fi deci o logică *care să genereze pe intrare* 1, *de fiecare dată cînd în starea următoare ieșirea trebuie să fie* 1. Spre exemplu, bistabilul A trebuie să fie pe 1 ca urmare a stărilor 2, 6, 7 și 5. Vom introduce, deci, 1 în aceste poziții în diagrama Veitch D_A din figura 5-6*c*. Aplicînd procedeele de simplificare cunoscute vom obține ecuația pentru bistabilul A. La fel se obțin ecuațiile pentru bistabilii B și C numărătorul implementîndu-se pe baza lor ca în figura 5-6*f*. Ecuația pentru D_C s-a întîmplat să fie XOR între A' și B' (vezi Secțiunea 3.15) așa că se implementează cu o poartă de tip XOR.

Proiectarea aceluiași numărător cu bistabili J—K se face utilizîndu-se aceeași metodă obținîndu-se de această dată ecuațiile pentru intrările J și K ale fiecărui bistabil.

Deoarece pentru intrarea J aplicarea semalului de 1 logic nu este semnificativă dacă în starea anterioară $Q = 1$, *vom putea introduce X-uri în jumătate din pătratele fiecărei diagrame Veitch.* Spre exemplu, dacă bistabilul A este setat ($Q = 1$), intrarea J_A nu mai este semnificativă, și vom putea introduce X-uri în jumătatea stîngă a diagramei, unde $A = 1$. Similar nu este semnificativă intrarea K atunci cînd bistabilul este resetat ($Q = 0$), deci vor putea fi intro-

duse X-uri în jumătatea dreaptă a diagramei pentru K_A, unde $A = 0$*. Aceste X-uri pot fi completate înainte de a începe rezolvarea problemei concrete propuse. Dacă există și stări cărora nu le-a fost atribuită nici o semnificație pot fi introduse X-uri suplimentare în aceste poziții.

Urmează examinarea stărilor, una cîte una, și completarea diagramei Veitch pentru definirea tranzițiilor fiecărui bistabil. *Pentru fiecare tranziție* $0 \rightarrow 1$ *(set) vom scrie un* 1 *în diagrama Veitch pentru* J *; pentru fiecare tranziție* $1 \rightarrow 0$ *(reset) vom scrie* 1 *în diagrama Veitch pentru* K *în pătratul corespunzător stării anterioare.* Spre exemplu, pentru a avansa *din* starea 0 va trebui să comutăm bistabilul C, completînd 1 în colțul din dreapta jos al diagramei J_C (starea 0). În continuare trebuie modificat B scriind 1 în diagrama J_B în pătratul corespunzător stării 1. Următoarea stare este 3 și este necesară resetarea bistabilului C, prin completarea cu 1 în diagrama K_C în poziția 3, pentru evoluția în continuare a numărătorului. Se continuă în acest mod pînă se obțin tranziția din toate stările.

Odată completate diagramele Veitch, ecuațiile pentru intrările J și K se scriu în forma cea mai simplă și se trece la implementarea reprezentată în figura 5-6*m*. În acest caz putem economisi cîteva porți prin folosirea unor porți NOR cu intrările inversate, pentru generarea funcțiilor AND necesare pentru J_A, K_A, J_B și K_B (conform analizei din Secțiunea 3.13). Deoarece pentru J_C și K_C se obțin funcții XOR (vezi Secțiunea 3.15), implementarea se face direct cu porți XOR.

Dacă am fi proiectat numărătorul folosind bistabili de tip T am fi urmat o procedură similară exceptînd faptul că pentru intrările T am fi completat cu 1 în toate pătratele din diagramele Veitch corespunzătoare stărilor care preced orice tranziție $0 \rightarrow 1$ sau $1 \rightarrow 0$.

5.5. NUMĂRĂTOARE MOEBIUS

Numărătorul în cod Gray pe care l-am proiectat mai înainte necesită pentru implementare un nunăr relativ mare de circuite combinaționale. Alte secvențe de numărare se pot obține prin folosirea unui număr mult mai mic de circuite combinaționale. Spre exemplu un numărător cu opt stări la care la fiecare tranziție se modifică numai un singur bit se poate construi utilizînd secvența prezentată în figura 5-7 *a*. Acest tip de numărător se numește *„numărător Moebius"***. Putem verifica funcționarea corectă a schemei din figura 5-7 *b* și *c* prin inspectarea directă a succesiunii de stări generate. Starea fiecărui bistabil este determinată de starea anterioară a bistabilului plasat în stînga sa. Starea primului bistabil din stînga este dată de starea anterioară a ieșirii *complementate* a ultimului bistabil din dreapta. Sînt disponibile circuite MSI (denumite registre de deplasare) cu cîte patru bistabili ce se pot conecta în acest mod.

---

* În general comanda intrărilor J și K se poate realiza numai cu una din variabilele de intrare ocalaltă variabilă fiind X. De exemplu pentru tranziția lui Q din starea 0 în starea 0 sînt posibile următoarele două situații: $J = 0$, $K = 0$ sau $J = 0$, $K = 1$; aceste situații sînt echivalente cu $J = 0$, $K = X$. Pentru celelalte tranziții rezultă: $0 \rightarrow 1$, $J = 1$, $K = X$; $1 \rightarrow 0$, $J = X$, $K = 1$; $1 \rightarrow 1$, $J = X$, $K = 0$. Trebuie observat că această regulă nu se aplică bistabilelor R—S.

** Se întîlnește și denumirea de *„twisted tail ring counter"* ceea ce într-o traducere cuvînt cu cuvînt înseamnă „numărător în inel cu coada răsucită". (n.t.).

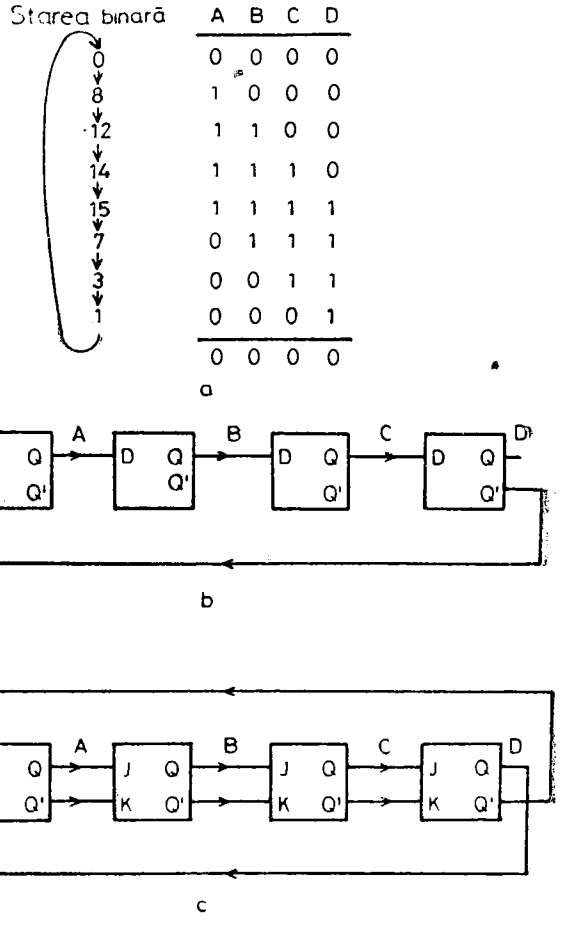


Fig. 5-7. *(a)* Numărător Moebius de patru biți (divizor cu 8). *(b)* Implementarea cu bistabili de tip D. *(c)* Implementarea cu bistabili de tip J-K *(d)* Secvența de numărare a unui numărător Moebius de trei biți (divizor prin 6). *(e)* Secvența de numărare a unui numărător Moebius de doi biți (divizor prin 4).

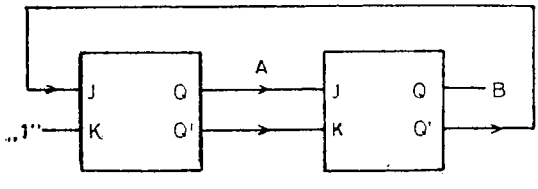
Obținem deci un numărător, cu caracteristicile impuse, folosind un singur circuit MSI față de cazul anterior (secvența din figura 5-6) ce a presupus folosirea a cel puțin trei capsule integrate. Chiar dacă am folosi bistabile discrete, vor fi necesare mai puține capsule cu toate că numărul de bistabile este mai mare cu unu. Folosind aceleași principii pot fi construite numărătoare Moebius de orice dimensiune. Secvențele pentru doi și trei biți sînt prezentate în figurile 5-7 *d* și *e*. De observat că pentru aceste dimensiuni nu se „risipesc" bistabile deoarece numărul de bistabile nu este mai mare decît ar fi pentru alte tipuri de numărătoare cu patru (două bistabile minimum), respectiv șase stări (trei bistabile minimum). Utilizarea unor astfel de numărătoare pentru un număr mare de stări devine însă nepractică deoarece cu n bistabile se obțin numai $2n$ stări distincte.

Numărătoarele Moebius sînt folosite deseori ca numărătoare de stare. Dacă numărătorul Moebius este implementat cu bistabile J—K, ca în figura 5-7 *c*, fiecare comutare a stării este controlată de cîte o intrare diferită. Din acest motiv modificarea oricărei stări poate fi controlată independent adăugînd un AND (sau NOR) pe intrarea care controlează schimbarea stării respective. Spre exemplu, în figura 5-7 *c* prima comutare ($0 \rightarrow 8$) poate fi blocată introducînd o poartă pe J_A. Următoarele stări se modifică controlînd prin porți J_B, J_C, J_D, K_A, K_B, K_C și respectiv K_D.

O altă aplicație a numărătorului Moebius este aceea de generare a ceasurilor cu multe faze. Cele opt ieșiri ale bistabililor din figura 5-7 *b* sau *c* generează opt semnale de ceas defazate în mod egal, cu factorul de umplere de 50%. Cele opt tranziții pozitive ($0 \rightarrow 1$) apar succesiv, o dată cu impulsul de ceas la următoarele ieșiri ; A, B, C, D, A', B', C', D', conform secvenței din figura 5-7 *a*. Vom putea deci obține un ceas cu opt faze, folosind un numărător Moebius de patru biți, atacat cu un ceas avînd frecvența de opt ori mai mare decît frecvența dorită. În general, un numărător Moebius de n biți generează $2n$ faze de ceas.

Un alt avantaj al numărătorului Moebius este dat de posibilitatea de a decodifica orice stare sau orice grup consecutiv de stări folosind o poartă cu două intrări. Pentru a decodifica o singură stare vom repera cei doi biți adiacenți distincți 1 și 0, spre exemplu pentru starea 14 = $C \cdot D'$. Diagrama Veitch va confirma că ceilalți trei minitermeni nesemnificativi pot fi incluși în această funcție. Decodificarea unui grup de stări consecutive se poate obține identificînd ultimul 1 al primei stări din grup și primul 0 al ultimei stări — spre exemplu pentru stările 8, 12 și 14 = $A \cdot D'$. Orice grup de stări consecutive poate fi deci decodificat cu o poartă cu două intrări.

Modificînd numai foarte puțin secvența de numărare Moebius putem obține alte secvențe cu minimum de efort. Spre exemplu, în figura 5-8 este prezentat un numărător cu trei stări obținut prin modificarea semnalului pe K_A a numărătorului cu patru stări pentru a-l face să se reseteze mai devreme.



0 0
1 0
0 1
0 0

Fig. 5-8. Numărător (divizor) cu 3.

5.6. DEFINIREA NUMĂRĂTOARELOR DE STARE

Cu excepția numărătoarelor de foarte mici dimensiuni ($\div 2$, $\div 3$, $\div 4$) sau a aplicațiilor foarte speciale, majoritatea numărătoarelor sînt construite prin simpla interconectare a unor numărătoare MSI în configurații impuse de lungimea ciclului dorit. În multe sisteme, însă cel puțin un numărător trebuie să fie mai mult decît un simplu numărător binar. Adeseori secvența sa este atît de complexă încît el seamănă mai mult cu un registru decît cu un numărător în sensul că răspunde mai mult semnalelor externe decît propriei sale stări.

Primul pas în proiectarea unui circuit secvențial îl constituie construirea *diagramei de stări*. Diagrama de stări este formată din cercuri ce corespund fiecărei stări în parte (vezi fig. 5-9). În interiorul cercurilor este înscris codul binar al stării și semnificația acesteia pentru sistemul proiectat. Stările sînt interconectate prin săgeți ce indică succesiunea lor. Funcția logică scrisă lîngă fiecare săgeată indică condiția în care poate avea loc tranziția. Diagrama de stări definește complet funcția numărătorului de stare.

Diagrama de stări este generată stare cu stare în funcție de specificațiile problemei. Asocierea codurilor binare fiecărei stări este făcută numai după ce diagrama a fost completată integral cu specificarea „semnificațiilor" fiecărei stări și a condițiilor de intrare. Prima stare este, de regulă, starea „inițială" a sistemului în care acesta trece odată cu alimentarea circuitului. Pornind din această stare se indică prin săgeți marcate de condiții logice „următoarele" stări posibile. Apoi, pornind din aceste stări vom desena alte săgeți marcate de condiții logice către alte stări impuse de funcționarea sistemului etc.

Odată cu apariția de noi stări în diagrmă, este necesar să se determine dacă toate trebuie într-adevăr *„memorate"* prin stări distincte. Spre exemplu, dacă un comutator bipozițional determină un anumit mod de operare în sistem, nu trebuie introdusă o stare specială deoarece logica sistemului poate fi acționată de ieșirea acestui comutator determinînd modul de funcționare. Dacă comutatorul nu realizează decît un contact momentan ce trebuie sesizat de sistem ca o tranziție, atunci va fi necesară o stare specială pentru memorarea apariției comenzii date de acest comutator. *Tranziția stării „memorează" deci condițiile tranzitorii ce nu sînt menținute pe toată durata funcționării sistemului și nici nu sînt memorate în nici o altă zonă a sistemului.*

Atunci cînd definim o stare nouă trebuie să fim atenți să nu fie de fapt un nume nou pentru o stare care a fost deja definită. Noua stare trebuie să definească *fie o operație nouă, fie cel puțin o stare care să aibă o condiție de ieșire, diferită de cele asociate altor stări.* De asemenea, trebuie să fim atenți la posibilitatea de *blocare* („hanging up") a sistemului. Dacă vom ajunge într-o stare pentru care condiția de tranziție nu va fi niciodată îndeplinită, vom bloca sistemul în această stare pentru totdeauna. Cele mai multe diagrame de stări includ legături înapoi la starea inițială odată cu încheierea unui ciclu de operații.

5.7. UN EXEMPLU

În figura 5-9 este definită diagrama de stări pentru un sistem de citire a datelor de pe o bandă magnetică, într-o memorie RAM și le transferă printr-o linie telefonică împreună cu caracterul de control. Sistemul așteaptă între citirea semnalului și recunoașterea corectitudinii recepției. Dacă nu este

recepționat un semnal de recunoaștere (ACKnowledge) timp de 1/3 secunde TIMOUT), sistemul trimite mesajul încă o dată. Acest ciclu este repetat pînăi (cînd este recepționat semnalul de recunoaștere (ACK). Acest sistem ma posedă numărătoare asociate prelucrării pe bit și cuvînt sau pentru alte operații necesare controlului benzii magnetice.

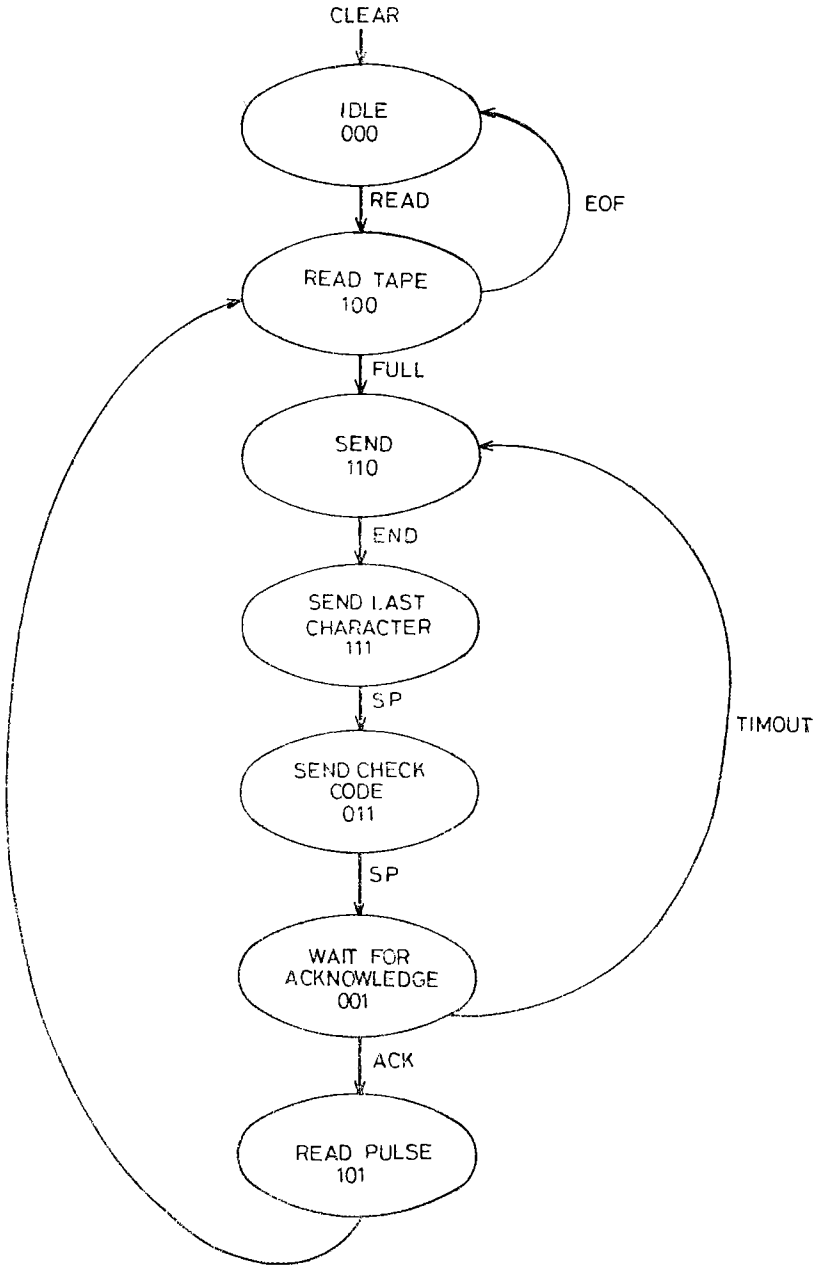


Fig. 5-9. Diagramă de stări pentru un transmițător de date.

Pornind din starea inițială, de repaus (IDLE), dacă este acționat butonul de citire (READ) se trece în starea de citire a benzii (READ TAPE). Dacă este detectat **sfîrșit de filă (End of file, EOF)**, indicînd sfîrșitul înregistrării, are loc tranziția în stare aIDLE. Dacă memoria s-a încărcat cu conținutul benzii (FULL), putem trece în starea de transmitere (SEND) pînă în momentul în care memoria este golită (END). Sistemul trece în starea de emisie a ultimului caracter (SEND LAST CHARACTER), necesară datorită faptului că datele sînt transferate bit cu bit dintr-un registru încărcat din RAM. Atunci cînd numărătorul de bit indică ultimul bit starea de stop (SP) a acestui numărător determină tranziția în starea de generare a codului de verificare (SEND CHECK CODE). Cînd ultimul bit al acestui cod a fost emis, aceeași condiție (SP) dată de numărătorul de bit permite avansarea în starea de așteptare a semnalului de recunoaștere (WAIT FOR ACKNOWLEDGE). După recepționarea semnalului ACKnowledge sistemul comută în starea READ PULSE pe durata unui impuls de ceas, pentru a simula apăsarea butonului READ, determinînd tranziția în starea READ TAPE (de pe bandă se citește o nouă înregistrare). Acest ciclu se repetă pînă se încheie citirea conținutului benzii și semnalul EOF readuce sistemul în starea inițială (IDLE). Dacă semnalul ACK nu este recepționat în 1/3 secunde după intrarea în starea de așteptare a semnalului de recunoaștere (WAIT FOR ACKNOWLEDGE) semnalul TIMOUT face ca înregistrarea să fie transmisă încă odată.

În figura 5-9 au fost definite toate stările și condițiile de tranziție în aceeași modalitate în care am face această descriere pentru înțelegerea noastră asupra procesului. Trebuie menționată, ca deosebit de importantă, introducerea buclelor în această diagramă, precum tranziția necondiționată în READ TAPE sau tranziția condiționată de TIMOUT în starea SEND (în acest fel ne reîntoarcem la stări definite anterior în loc să definim unele noi).

De asemenea, dacă nu am fi prevăzut semnalul TIMOUT, sistemul ar putea rămîne blocat în starea WAIT FOR ACKNOWLEDGE.

5.8. ALOCAREA STĂRILOR

Odată definite stările sistemului trebuie să le asociem coduri binare. Deoarece aceste coduri corespund combinațiilor binare înregistrate în cei trei bistabili ce formează numărătorul, stabilirea lor are implicații importante asupra circuitelor logice adiționale necesare implementării numărătorului de stări precum și asupra altor circuite din sistem.

Logica externă sistemului poate fi minimizată *făcînd o corespondență directă între bistabili și semnalele de comandă necesare* în exterior. Spre exemplu, conform alocării stărilor realizată în figura 5-9, bistabilul din mijloc este setat timp de trei stări în care este comandată emisia datelor (110, 111, 011). În consecință acest bistabil poate fi folosit *direct* pentru a permite emiterea datelor, în loc de a folosi porți suplimentare pentru a obține un semnal activ pe durata acestor trei stări.

Logica asociată direct numărătorului poate fi minimizată dacă *se ține cont de faptul că există unele secvențe ce pot fi generate mai ușor decît altele.* Toate tranzițiile, cu excepția celei condiționate de TIMOUT, presupun *modificarea stării cîte unui singur bistabil.* Acest lucru implică condiționarea prin porți logice a unei singure intrări de bistabil la fiecare tranziție Succesiunea de numărare folosită în figura 5-9 este liniară pornind din starea inițială,

exceptînd cele trei salturi. Pentru a simplifica logica numărătorului și cea de decodificare, primele șase stări constituie o secvență de numărare Moebius de tipul celei descrise în Secțiunea 5.5. Cu toate că a fost adăugată o stare în afara secvenței (READ PULSE) și există două ramuri în afara secvenței Moebius, ne putem aștepta la o configurație de porți mai simplă deoarece scheletul pe baza căruia am stabilit stările este constituit de această secvență. De asemenea, luînd ca stare inițială 000, apare posibilitatea inițializării corecte a numărătorului prin acționarea asupra intrărilor asincrone de ștergere (CLEAR) a bistabililor.

5.9. IMPLEMENTAREA CU BISTABILI A NUMĂRĂTORULUI DE STARE

Se poate implementa numărătorul de stare definit în figura 5-9 utilizînd același procedeu ca pentru un numărător necondiționat de semnale externe. *Diferența esențială ce apare este aceea că, în loc de a se completa în diagrama Veitch 1 pentru a determina tranziția stării dorite, se completează cu funcția logică ce definește condiția de tranziție a stării.* Vom apela la bistabile J—K deoarece utilizarea lor conduce la o formă mai simplă a implementării în aplicațiile în care numărarea, evoluția stărilor, trebuie oprită condiționat (deoarece absența semnalului menține bistabilul în starea anterioară).

Începem prin a construi o diagramă Veitch de mai mari dimensiuni (figura 5-10*a*) în care trecem toate stările conform denumirii date. Această diagramă va fi folosită ca referință pentru a localiza stările celorlalte șase diagrame utilizate, ce permit găsirea funcțiilor logice pentru cele șase intrări ale numărătorului. Deoarece starea 010 nu este folosită vom completa cu X-uri colțurile din dreapta sus ale celor șapte diagrame. În continuare, vom completa cu X-uri în patru din locațiile diagramelor ce definesc intrările J și K ale bistabililor conform regulilor anterior stabilite, după care, dacă $Q = 1$ intrarea J este nesemnificativă, iar dacă $Q = 0$ intrarea K nu este semnificativă pentru determinarea tranziției.

În acest moment putem începe completarea diagramelor pentru specificarea tranzițiilor. În starea IDLE bistabilul A va fi setat ($Q = 1$) dacă READ = 1, deci vom scrie READ în colțul din dreapta jos al diagramei J_A. Deoarece nu există alte tranziții din această stare, mergem mai departe la starea READ TAPE. Deoarece trebuie setat B dacă FULL = 1, vom completa cu FULL colțul din stînga jos al diagramei J_B. De asemenea trebuie resetat bistabilul A dacă EOF = 1, deci vom completa cu EOF în colțul din stînga jos al diagramei K_A. Se va continua pînă la completarea tuturor diagramelor ca urmare a parcurgerii întregii diagrame de stări. Dacă un bistabil este afectat de mai mult de o săgeată (condiție) vom utiliza simbolul „+" scriind în căsuța respectivă ambele condiții (exemplu : în diagrama pentru J_A).

Următorul pas îl constituie extragerea din cele șase diagrame a funcțiilor logice în forma lor cea mai simplă (se va utiliza X oriunde se poate). Vor rezulta șase funcții de *10 variabile* dar deoarece șapte dintre variabile interacționează foarte puțin se vor putea folosi pentru simplificare diagrame de trei variabile cu toate că în acest exemplu nu apar funcții comune pentru mai multe intrări deoarece stările au fost stabilite astfel încît cele mai multe tran-

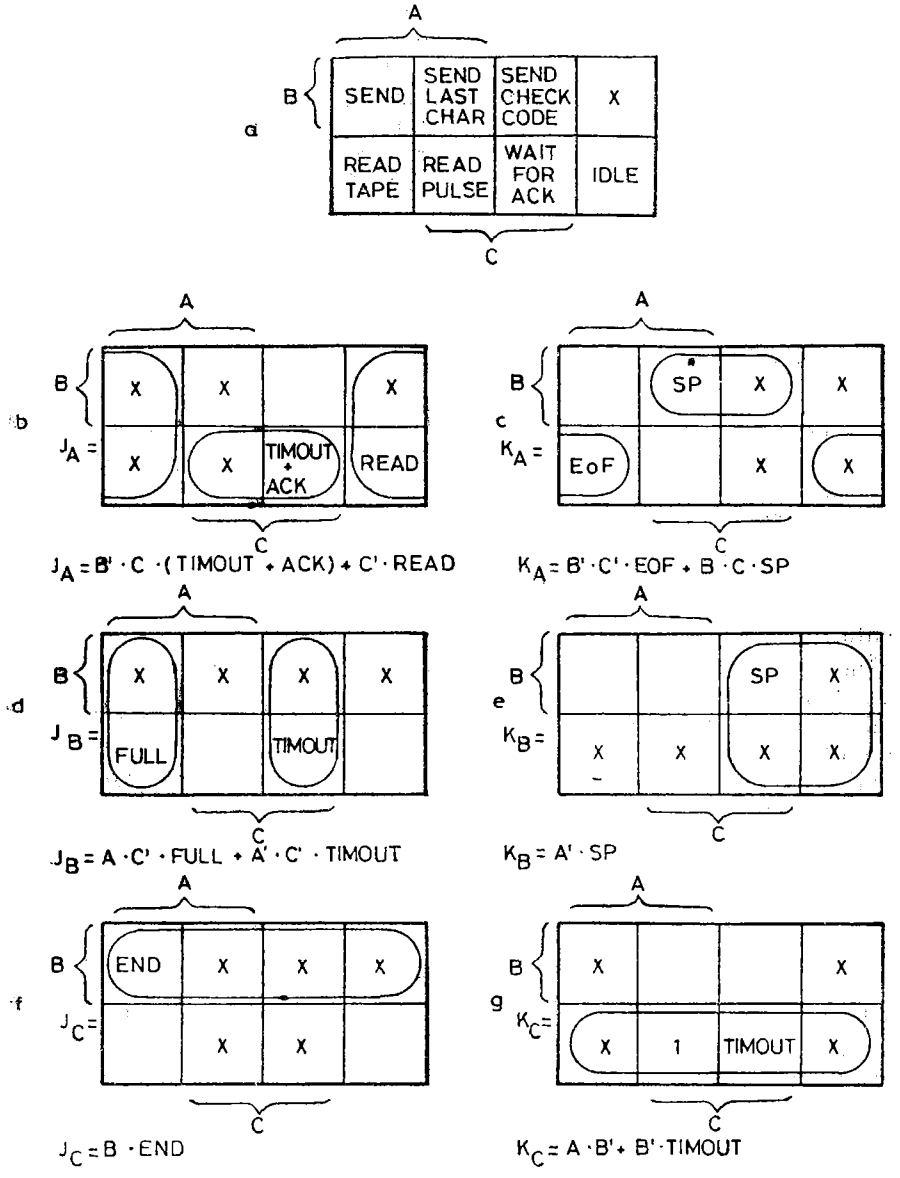


Fig. 5-10. Numărător controlor de stări pentru transmițătorul de date. *(a)* Notarea stărilor (pentru referință, vezi Secțiunea 5.7). *(b)* J_A. *(c)* K_A *(d)* J_B. *(e)* K_B. *(f)* J_C . *(g)* K .

ziții de stare să presupună comutarea doar a unui singur bistabil ; alegerea (modificarea) termenilor astfel încît aceeași poartă să fie folosită la implementarea mai multor funcții constituie o alternativă uneori posibilă.

În figura 5-11 este prezentată implementarea numărătorului folosind logica de tip NAND/NAND (cu două NOR-uri numai, unde a fost mai avantajoasă această soluție). Rezultă că sînt necesare 5 2/3 capsule integrate.

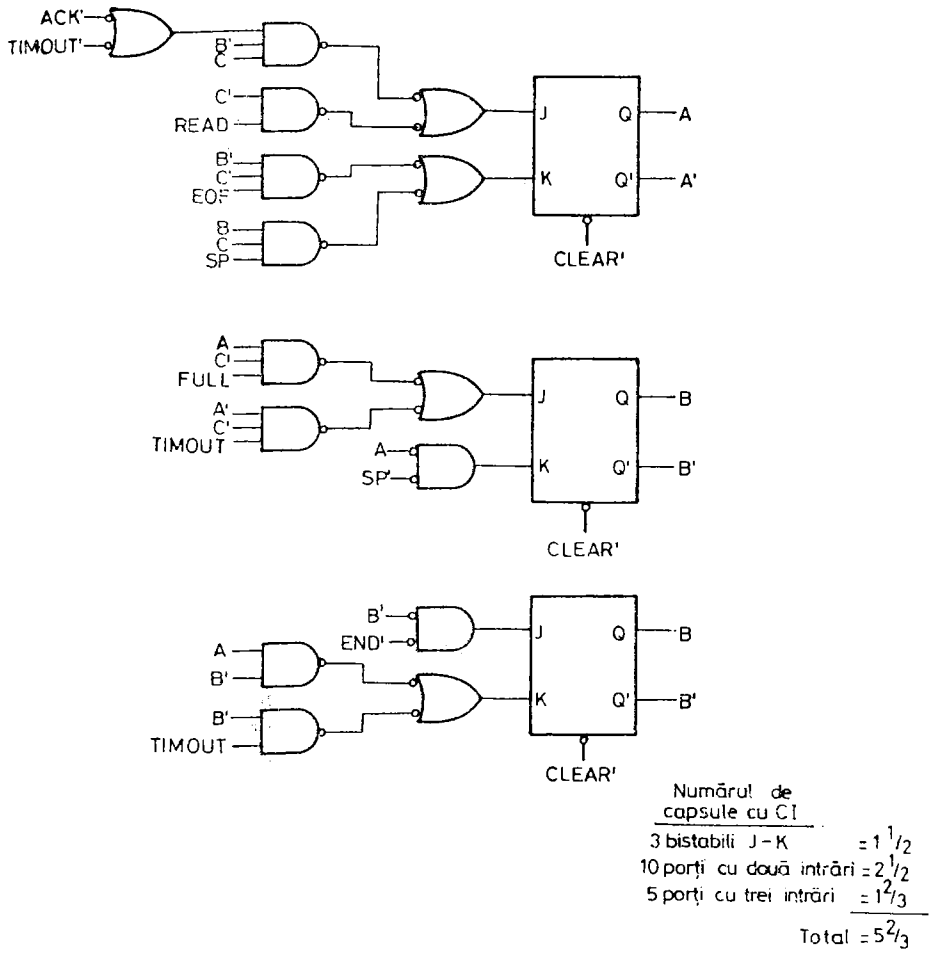


Fig. 5-11. Implementarea numărătorului controler de stări pentru transmițătorul de date.

Sub această formă structura mai poate fi micşorată, deoarece noi am presupus că orice semnal de intrare poate apare în *orice* moment. Această presupunere este acoperitoare pentru situaţiile în care nu cunoaştem foarte bine mecanismele de generare ale acestor semnale. Dacă, însă, putem folosi restricţiile impuse acestor semnale, de mecanismele de ansamblu ale sistemului din care face parte numărătorul proiectat, vom putea elimina o parte din porţile folosite la implementare. TIMOUT, spre exemplu, este un semnal generat la intrarea numărătorului *numai* atunci cînd acesta se află în starea WAIT FOR ACKNOWLEDGE. În aceste condiţii funcţiile J_A, J_B şi K_C se vor simplifica (spre exemplu : $J_B = A \cdot C' \cdot \text{FULL} + \text{TIMOUT}$). EOF apare numai cînd este citită banda magnetică, deci ecuaţia pentru K_A se va simplifica devenind $K_A = \text{EOF} + B \cdot C \cdot SP$. Deoarece semnalele de control pot veni din subsisteme proiectate de alţii în diferite perioade de timp, de multe ori însă este mai bine să implementăm schema cu condiţionări acoperitoare (risipind astfel cîteva porţi) pentru a elimina posibilitatea apariţiei unor tranziţii greşite din lipsa de corelare suficientă între schema noastră şi schemele proiectate de alţii.

5.10. FOLOSIREA NUMĂRĂTOARELOR MSI

Disponibilitatea numărătoarelor MSI, cu patru bistabile pe capsulă, simplifică puternic multe proiecte de numărătoare. Figura 5-12*a* prezintă un tip foarte general de numărător binar MSI. Atunci cînd intrarea LOAD trece în zero, numărătorul este presetat (încărcat) sincron, la următorul impuls de ceas, cu starea definită pe cele patru intrări de tip DATA (intrări de date). Ieşirea CARRY (transport) şi intrarea COUNT ENABLE (condiţionarea numărării) fac posibilă *conectarea în cascadă a mai multor numărătoare pentru a se obţine un numărător oricît de mare.* (Ieşirea CARRY se conectează la intrarea COUNT ENABLE a următorului numărător, permiţînd ca la 16 numărări ale primului să fie comandată o numărare a următorului. După 16 incrementări ale celui de al doilea numărător, ieşirea CARRY a acestuia comandă o numărare a celui de al treilea şi aşa mai departe). Prin conectarea

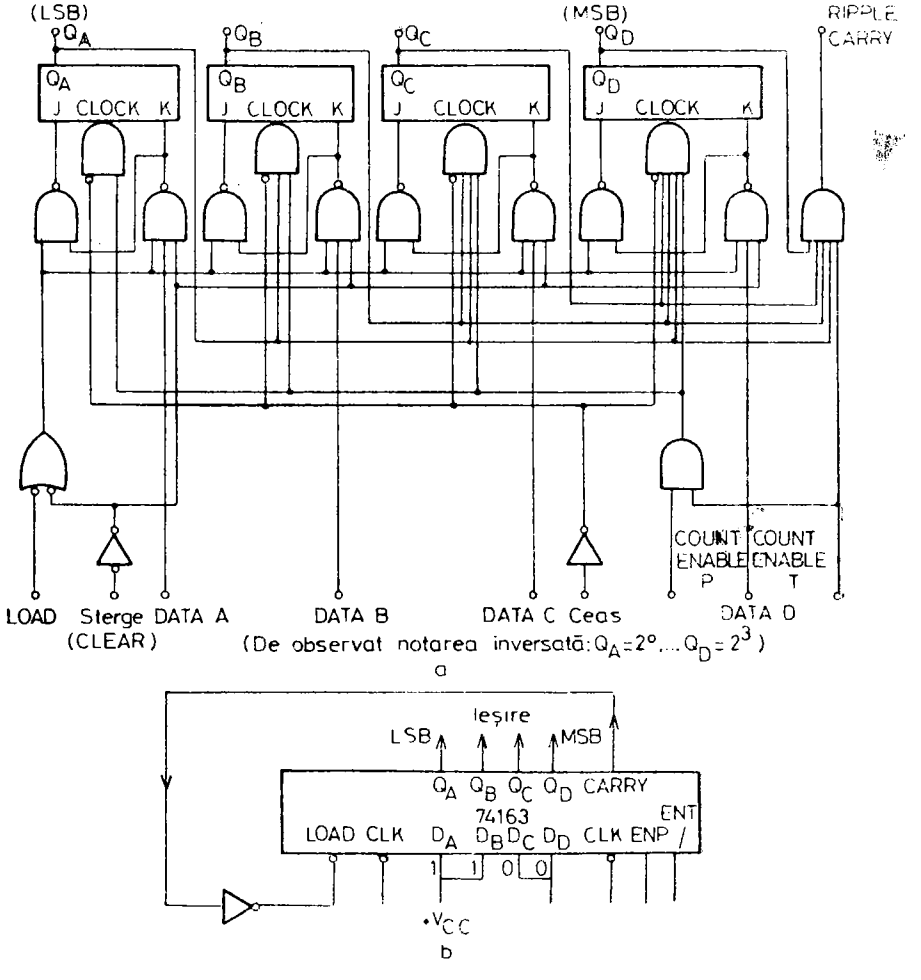


Fig. 5-12. Numărător binar sincron presetabil (74163); *(a)* schema logică; *(b)* ca numărător de stări divizor cu n ($n=13$).

ieşirii CARRY la intrarea LOAD', ca în figura 5-12*b*, se poate obţine un *numărător programabil* ce generează secvenţe de *ORICE* lungime. Spre exemplu, în figură este reprezentat un numărător conectat astfel încît să obţinem un ciclu de 13 stări ; deoarece pe intrările de date este generată configuraţia 0011 = 3, după tranziţia în starea 16 (CARRY) numărătorul trece în starea 3*. Lungimea ciclului va deveni 16 — 3 = 13. În această modalitate poate fi programată orice lungime de ciclu prin conectarea convenabilă a intrărilor numărătorului. Acest procedeu poate fi aplicat şi numărătoarelor mai mari, formate din mai multe astfel de capsule, utilizînd semnalul CARRY — inversat — de la ordinul cel mai semnificativ pentru comanda intrărilor LOAD' de la fiecare numărător.

Deoarece cu circuitele MSI descrise, se pot construi numărătoare sincrone de orice lungime *numărătoarele cu bistabile discrete se vor limita la aplicaţiile ce presupun un număr foarte mic de stări* (numai 2, 3 şi 4). Numărătoarele de stare pot fi de asemenea realizate cu un numărător MSI.

Orice numărător controler de stare poate fi gîndit ca un generator de secvenţă de numărare liniară cu „salturi" (jumps) sau „bifurcaţii" (branches). Spre exemplu, în figura 5-9 este reprezentată o secvenţă de numărare liniară de sus pînă jos, cu trei salturi la stări anterioare. *Dacă vom redenumi stările sub forma unei succesiuni binare şi vom utiliza intrările LOAD' şi CLEAR'* pentru obţinerea salturilor, vom putea reproiecta numărătorul de stare folosind numărătorul MSI din figura 5-12. Desemnînd starea iniţială 0000, intrarea CLEAR' se va putea folosi pentru iniţializare şi tranziţie în starea iniţială. Intrarea LOAD' va putea fi folosită pentru saltul condiţionat în orice stare, dacă se aplică pe intrările DATA codul binar al acesteia.

În loc de a se scrie ecuaţiile pentru intrările *J—K* ale bistabililor, problema se va rezolva prin scrierea ecuaţiilor pentru :

1. secvenţa de numărare de bază (COUNT ENABLE) ;
2. condiţionarea salturilor (LOAD') ;
3. intrările DATA pentru a determina unde are loc tranziţia ;
4. intrarea CLEAR ce permite iniţializarea.

Logica pe intrările de date este de regulă simplă, deoarece numărul stărilor din care au loc salturi nu este foarte mare iar celelalte stări introduc variabile nesemnificativ de puţine în diagrama logicii de tranziţie. O atenţie deosebită trebuie acordată funcţiilor de numărare (COUNT) şi încărcare (LOAD). Am văzut, în Capitolul 4, modul în care pot fi implementate cu multiplexoare MSI funcţii logice complexe. Vom utiliza această tehnică pentru a genera funcţiile COUNT şi LOAD, alocînd cîte un multiplexor pentru fiecare. Aceste multiplexoare vor *selecta semnalele de control necesare pentru tranziţia fiecărei stări.*

5.11. O REPROIECTARE CU CIRCUITE MSI A NUMĂRĂTORULUI DE STARE

Vom reproiecta cu circuite MSI numărătorul de stare pentru transmisii de date, care a fost proiectat anterior utilizînd bistabile *J—K*. Pentru început, remarcăm faptul că desemnarea stărilor din figura 5-9 nu mai este corespunzătoare, deoarece *dorim o succesiune binară de numărare pentru linia prin-*

* Atenţie la faptul că ieşirea bistabilului cel mai puţin semnificativ (LSB) este Q_A, (adică $Q_A = 2^0$ şi $Q_D = 2^3$) invers faţă de cum apare în cataloage.

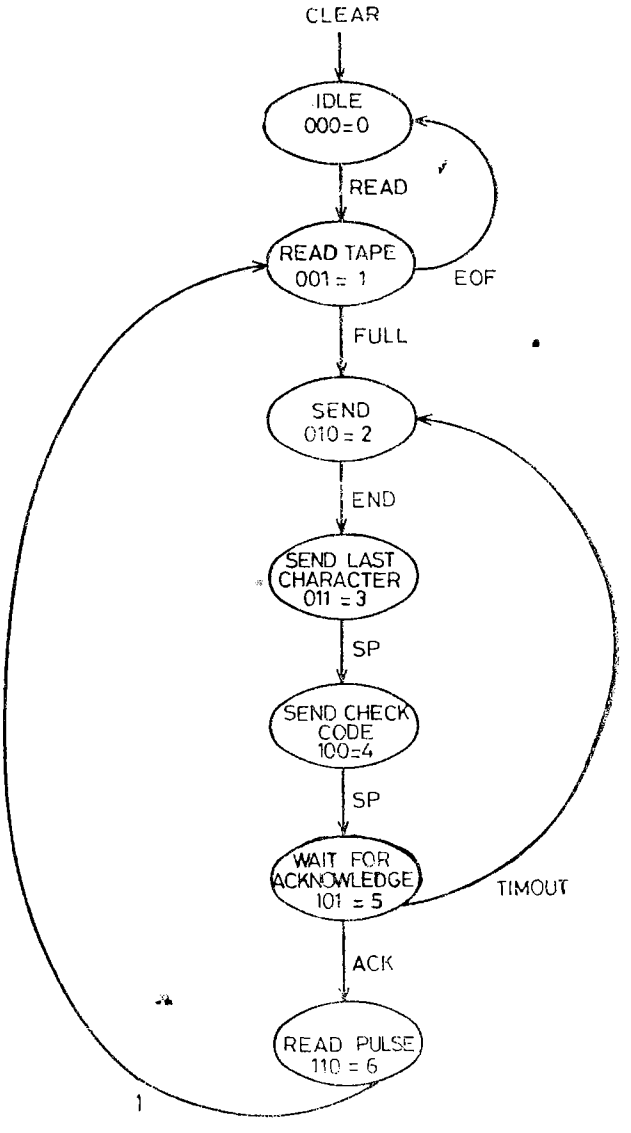


Fig. 5-13. Renotarea stărilor pentru controlerul transmițătorului de date. Numerele binare=Q_C, Q_B, Q_A (pentru notațiile din figură vezi Secțiunea 5.7).

cipală a diagramei de stări. În figura 5-13 este prezentată o diagramă de stări ce îndeplinește această condiție. Se observă faptul că, dacă diagrama de stări nu are o structură evident liniară, ca în cazul nostru, se poate defini în mod arbitrar una.

În figura 5-14*a* este prezentată implementarea directă ce rezultă pornind de la diagrama din figura 5-13. Remarcăm faptul că „multiplexorul pentru numărare" selectează, într-o manieră foarte simplă, semnalele de control potrivite pentru părăsirea fiecărei stări ce se află pe succesiunea binară liniară principală. *Fiecare intrare a multiplexorului corespunde unei „săgeți" din dia-*

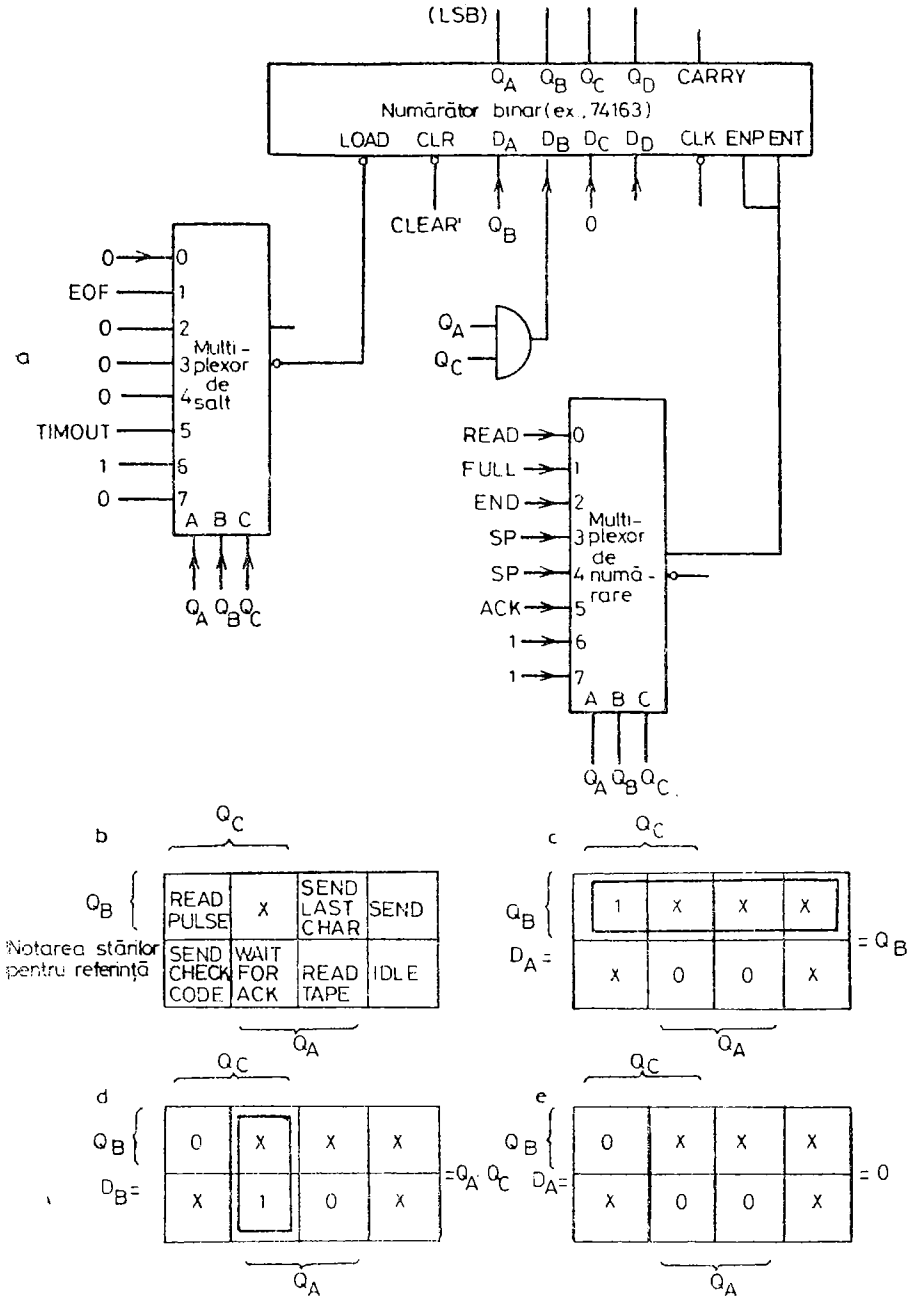


Fig. 5-14. Realizarea numărătorului controler de stări cu un numărător MSI ; *(a)* schema logică ; *(b)* notarea stărilor (pentru referință).

gramă, ce marchează tranzițiile pe succesiunea principlaă (liniară). Pe de altă parte fiecare intrare a „multiplexorului de salt" corespunde unei săgeți ce indică salturile în diagramă. Deoarece acest multiplexor selectează numai semnalele de control ce indică salturile din diagramă, la intrările corespunzătoare stărilor din care nu au loc salturi se conectează zero logic (masă).

Deoarece, modul de conectare a intrărilor multiplexoarelor de numărare și salt se poate citi direct din diagrama de stări, singura problemă reală ce apare în cadrul acestei proiectări este determinarea logicii pe intrările DATA, pentru realizarea salturilor condiționate. Pentru referință, în figura 5-14 *b* este reprezentată diagrama Veitch cu denumirile stărilor numărătorului. Figurile 5-14*c*, *d* și *e* reprezintă funcțiile pentru intrările DATA ale numărătorului. *Aceste funcții vor putea fi definite, pentru fiecare salt în parte, completînd codul pentru destinația dorită în pozițiile corespunzînd stărilor în care apare saltul.*

Spre exemplu, tranziția din starea READ PULSE presupune $D_C = 0$, $D_B = 0$ și $D_A = 1$ (pentru a trece în starea 001). Deci, vom introduce 0, 0 și 1 în colțul din stînga sus al diagramelor pentru D_C, D_B și respectiv D_A. Saltul din starea WAIT FOR ACKNOWLEDGE va fi efectuat, în cazul în care condiția de salt este îndeplinită, în starea 010, deci vom completa cu 0, 1 și 0 în poziția WAIT FOR ACKNOWLEGE a diagramelor D_C, D_B și respectiv D_A. Saltul din starea READ TAPE se face în starea 000, deci vom completa cu 0 în cele trei poziții corespunzătoare în diagrame. Deoarece salturile au loc numai din aceste trei stări, în celelalte poziții din diagramele Veitch se va completa cu X-uri, tranzițiile definibile în aceste situații nefiind niciodată efectuate. Urmează scrierea ecuațiilor logice pentru cele trei intrări DATA și implementarea cu porți logice ca în figura 5-14*a*. Deoarece, pentru a genera D_B este necesară o poartă logică AND (sau NAND și un inversor), putem lua în considerație observația făcută și în implementarea anterioară, conform căreia semnalul TIMOUT este adevărat numai în starea WAIT FOR ACKNOWLEDGE. În acest caz vom aplica direct pe intrarea D_B semnalul TIMOUT, deoarece Q_B este setat numai ca urmare a saltului inițiat de acest semnal.

Deoarece această aplicație presupune numai trei salturi utilizarea unui multiplexor pentru obținerea semnalului LOAD este greu de justificat. În acest caz utilizarea logicii convenționale este mai convenabilă. Utilizînd diagrama Veitch din figura 5-15 se ajunge la folosirea pentru implementare a 1 1/12 capsule integrate. Dacă vom presupune că TIMOUT AND EOF apar numai condiționate de stările în care acționează, ecuația logică poate fi redusă la

$$\text{LOAD} = Q_B \cdot Q_C + \text{TIMOUT} + \text{EOF} \qquad (5\text{-}1)$$

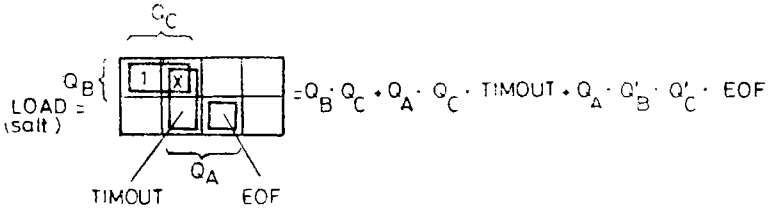


Fig. 5-15.

În acest caz utilizarea multiplexorului nu a fost absolut necesară. Cu toate acestea soluția cu multiplexor poate fi încă justificată și numai pentru faptul că este mai ușor de înțeles.

O altă posibilitate constă în utilizarea intrării CLEAR a numărătorului pentru generarea tranziției în starea 000. Vom putea, deci, omite EOF din ecuația pentru LOAD și introduce un X în poziție READ TAPE a diagramelor pentru D_A, D_B și D_C. Acest lucru va permite eliminarea porții pentru intrarea D_B, folosind $D_B = Q_A$.

Este interesant de remarcat faptul că implementarea cu MSI devine mai economică cu toate că sînt folosite numai trei din cele patru bistabile ale numărătorului și numai trei din cele opt intrări ale multiplexorului. Sîntem în fața unui exemplu elocvent în care deși „componentele ieftine" sînt utilizate ineficient, pe ansamblu obținem încă un cîștig.

5.12. ALTE CIRCUITE SECVENȚIALE MSI

Similare numărătorului prezentat în figura 5-12*a*, sînt disponibile și alte tipuri de numărătoare cum ar fi: numărătoare direct/invers (Up/Down), numărătoare zecimale, numărătoare Up/down zecimale, și numărătoare cu 12. Cu toate acestea, este bine de a se încerca, acolo unde este posibil, standardizarea unui tip general de numărător pentru o aplicație dată.

Un alt circuit MSI standard ce conține elemente de memorie este *registrul de stocare*. Acest circuit are intrări de ceas (CLOCK) și ștergere (CLEAR) comune pentru toate cele șase bistabile de tip D (ex. : 74147) din capsulă disponibile numai cu ieșirea Q sau patru bistabile D pe capsulă (cu Q și Q' disponibile de la fiecare bistabil). Desigur numărătorul din figura 5-12 poate fi folosit ca registru de stocare dacă este acționat pe intrările LOAD și DATA. Alte registre de patru biți sînt disponibile cu ieșiri de tip tristate (ex. : 74173) sau cu multiplexoare de doi biți încorporate pentru fiecare intrare în bistabili de tip D (ex. : 74298).

Un *registru asincron adresabil (addressable latch)* este echivalent cu un demultiplexor cu opt ieșiri ce au asociate fiecare cîte un element de memorie. O intrare de selecție activă pe zero și o adresă de trei biți selectează unul din cei opt bistabili asincroni în care se stochează data de la intrare. Utilizarea unui registru asincron adresabil pe ultimul nivel al unui demultiplexor structurat ca un arbore, de tipul celui din figura 4-11, permite obținerea unui arbore de demultiplexoare cu elemente de memorie pe fiecare ieșire. Folosirea unui astfel de demultiplexor va fi utilă acolo unde este necesară menținerea unui semnal la ieșirea selectată și după ce semnalul a dispărut de la intrare. Semnalul apare la ieșire ca urmare a aplicării semnalului de ceas activ pe toată durata palierului.

Într-o aplicație de tipul celei din figura 4-10*b* se poate obține *menținerea* unor semnale de scurtă durată, de pe intrare, la ieșire un timp nelimitat (ex. : 9334).

Registrele de deplasare sînt constituite din bistabile astfel conectate încît la fiecare impuls de ceas conținutul fiecăruia este încărcat în bistabilul vecin,

obținîndu-se o deplasare a conținutului întregului registru. Figura 5-16 conține principalele configurații de registre de deplasare:

1. Registru de opt biți cu intrare serie și ieșire paralel, (SIPO)*, în care datele sînt *încărcate bit cu bit.*

2. Registru de opt biți cu intrare paralel și ieșire serie (PISO)**, din care datele sînt *extrase bit cu bit.*

3. Registru de patru biți cu intrare și ieșire paralel (PIPO)***, în care datele sînt înscrise paralel *sau* serie și sînt citite de asemenea paralel *sau* serie. Registrul reprezentat în figura 5-16*c* este capabil să deplaseze în ambele direcții *și* să fie încărcat paralel.

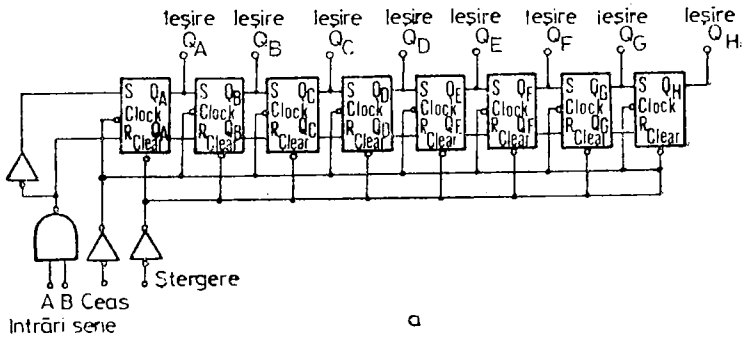


Fig. 5-16. *(a)* Registru de deplasare MSI; cu intrări serie și ieșiri paralel (74164).

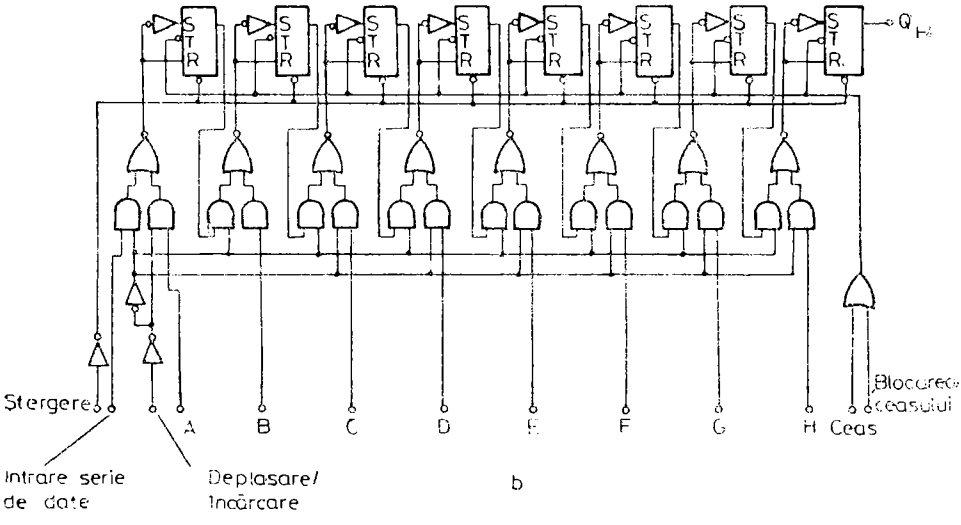


Fig. 5-16. *(b)* Registru de deplasare MSI; cu intrări paralel și ieșiri serie (74106).

* SIPO: *serial-in parallel-out* (n.t.)
** PISO: *parallel-in serial-out* (n.t.)
*** PIPO: *parallel-in parallel-out* (n.t.)

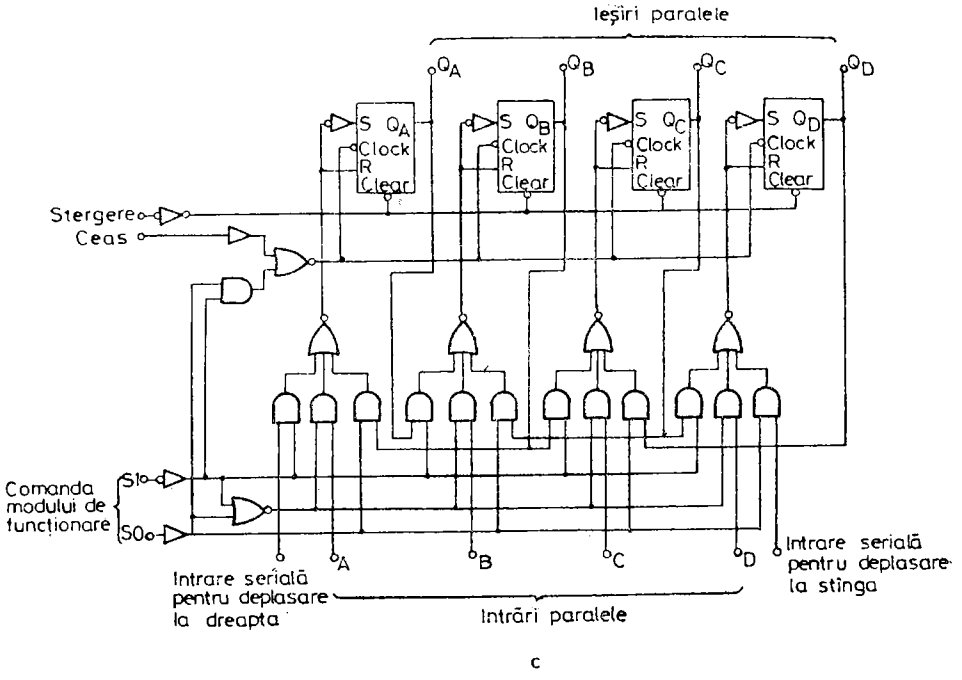


Fig. 5-16. *(c)* Registru de deplasare MSI ; cu intrări paralel şi ieşiri paralel — reversibil (74194).

Desigur, deplasarea poate fi obţinută cu orice registru presetabil (incluzînd numărătorul din figura 5-12), prin simpla conectare a intrărilor de date cu ieşirea bistabilului alăturat şi activitatea intrării LOAD.

O structură LSI tipică o constituie registrul cvadruplu conţinînd registre cu intrare şi ieşire serie (SISO)*. Lungimea registrelor poate fi de mii de biţi în această configuraţie deoarece sînt necesare numai o intrare şi o ieşire pentru fiecare. Deseori sînt prevăzute *porţi pentru recirculare* ce conectează, la o comandă externă, intrările cu ieşirile pentru fiecare registru. Această facilitate permite atît deplasarea datelor cît şi menţinerea lor. Astfel de circuite sînt disponibile în varianta conţinînd şase registre cu comandă comună de recirculare sau patru registre cu comenzi independente pentru recirculare.

EXERCIŢII

1. (*a*) Construiţi tabele de adevăr pentru definirea fiecărui tip de bistabil din figura 5-1 considerînd ca variabilă de intrare Q^n (Coloana Q^{n+1} va fi formată numai din 1 sau 0).
   (*b*) Construiţi diagramele Veitch asociate.
2. Proiectaţi reţelele de porţi necesare, similar cu figurile 5-2 şi 5-3, pentru :
   (*a*) Construirea bistabilului $R-S$, pornind de la bistabilul *D*.
   (*b*) Construirea bistabilului $J-K$, pornind de la bistabilul *D*.
   (*c*) Construirea bistabilului *D*, pornind de la bistabilul *T*.
   (*d*) Construirea bistabilului $J-K$, pornind de la bistabilul *T*.

* SISO : *serial-in serial-out* (n.t.)

3. (*a*) Scrieţi secvenţa de numărare în cod Gray reflectat pentru *patru biţi* similară aceleia din figura 5-6. (Se observă că secvenţa corespunzătoare la 3 biţi se obţine din cea pentru doi biţi. Pentru primii patru paşi A = 0 şi B, C au configuraţia codului Gray de 2 biţi. La următorii patru paşi $A = 1$ iar secvenţa de 2 biţi se generează în ordine inversă).
   (*b*) Proiectaţi generatorul definit, folosind bistabili $J-K$.
   (*c*) Proiectaţi cu bistabili D.
4. Proiectaţi un divizor binar cu 13 (0 . . .12) :
   (*a*) Cu bistabili D.
   (*b*) Cu bistabili $J-K$.
   (*c*) Cu bistabili T.
5. Proiectaţi un numărător binar cu 11 stări ce numără *invers* de la 15 la 5 :
   (*a*) Cu bistabili $J-K$.
   (*b*) Cu bistabili D.
   (*c*) Cu bistabili T.
6. (*a*) Proiectaţi o secvenţă de numărare Moebius de 5 biţi.
   (*b*) Implementaţi cu bistabili D.
   (*c*) Decodificaţi următoarele stări : 14, 15, 0, 7, 31.
   (*d*) Decodificaţi două, trei şi patru stări consecutive pornind din starea 0.
7. (*a*) Construiţi diagrama de tranziţii pentru un circuit secvenţial ce generează la ieşire 0 dacă intrarea A este menţinută pe 0 pentru mai mult de 15 impulsuri de tact (un circuit digital de semnalizare a depăşirii unui interval). Proiectaţi circuitul folosind :
   (*b*) Bistabile $J-K$ şi porţi NAND.
   (*c*) Bistabile D şi porţi NAND.
   (*d*) Numărătorul MSI prezentat în figura 5-12 şi porţi NAND.
8. (*a*) Construiţi diagrama de tranziţii a unui circuit ce generează 1 logic la ieşire timp de 13 impulsuri de ceas după ce intrarea A a trecut în 0 pentru un interval de timp egal cu perioada ceasului. Dacă A trece în 0 din nou pe durata celor 13 cicluri, această tranziţie este ignorată. (Acest circuit realizează funcţia unui monostabil neretrigerabil). Proiectaţi circuitul folosind :
   (*b*) Bistabili $J-K$ şi porţi NAND.
   (*c*) Bistabili D şi porţi NAND.
   (*d*) Numărătorul MSI din figura 5-12 şi porţi NAND.
9. Proiectaţi numărătorul de stare definit în figura 5-9. La *începutul* fiecărui caracter recepţionat ieşirea CHAR este activată (devine adevărată) pentru un ciclu de ceas. La *terminarea* verificării CHECK devine adevărat dacă nu a apărut nici o eroare. După transmiterea semnalului ACKnowledge, SCH este activat pentru un ciclu. Ca ieşiri sînt necesare semnalul STORE, pentru a indica începutul recepţiei şi semnalul SACK pentru a indica, că ACK va fi generat.
   (*a*) Construiţi diagrama de stări.
   (*b*) Proiectaţi circuitul folosind bistabile $J-K$ şi porţi NAND.
10. Implementaţi circuitul definit de diagrama de stări din figura 5-13, cu bistabile $J-K$ şi porţi NAND.
11. Reproiectaţi şi redesenaţi circuitul din figura 5-11 ţinînd cont de avantajele oferite de apariţia semnalelor TIMOUT, EOF şi ACKnowledge numai pe durata stărilor în care pot acţiona.
12. Reproiectaţi şi redresaţi circuitul din figura 5-14 folosind porţi NAND în loc de multiplexor, pentru funcţia LOAD, utilizînd şi avantajul apariţiei semnalelor TIMOUT, EOF şi ACK numai în stările dorite. De asemenea, utilizaţi intrarea CLEAR pentru a realiza tranziţia în funcţie de EOF.
13. Construiţi tabelul pentru intrările D_A, D_B, D_C, şi D_D astfel încît circuitul din figura 5-12*b* să poată fi folosit ca generator de cicluri cu lungimea variind între 1 şi 16.
14. Proiectaţi un numărător divizor cu 27 folosind două numărătoare MSI conectate similar reprezentării din figura 5-12*b*.
15. (*a*) Construiţi diagrama Veitch pentru intrarea COUNT ENABLE a circuitului din figura 5-14*a*.
   (*b*) Scrieţi ecuaţia logică sub forma cea mai simplă.
   (*c*) Implementaţi funcţia utilizînd porţi NAND.
16. Implementaţi un registru de deplasare folosind circuitul de numărare din figura 5-12*a*.

17. Conectați intrările circuitului din figura 5-16*c* în așa fel încît să execute una din următoarele funcții, comandate pe intrările S1, S0 :
   (*a*) Deplasare la dreapta.
   (*b*) Deplasare la stînga.
   (*c*) Inversarea ordinii biților : *ABCD*—*DCBA*.
18. Proiectați un ceas numeric pentru 24 de ore, folosind un oscilator de cuarț de 64 kHz, opt numărătoare MSI (figura 5-12*a*) și patru circuite de comandă pentru afișaj cu 7 segmente.

BIBLIOGRAFIE

Booth, Taylor, *Digital Networks and Computer Systems*, Wiley, New York, 1971, pag. 159—223. (se discută modul de construcție a diagramelor de stări).

Brock, *Desining With MSI*, Vol. 1, *Counters and Shift Registers*, Signetics Corp., Sunnyvale, California, 1970, Appendix 2, „Maximum Lenght ($2^n - 1$) Shift Counter Sequences".

Hamer, H., „Johnson Counter Decoder, „*Digital Design* November 1973, pag. 56—57.

Phister, M., *Logical Design of Digital Computers*, Wiley, New York, 1958, Capitolul 5, „Memory Element Input Equations", și Capitolul 6, „Huffman-Mealy Method of Minimizing Number of States".

Su, Stephen Y. H., „Logic Design and its Recent Development, Part 3 : Design of Sequential Networks", *Computer Design*, November 1973, pag. 85—92.

6

REALITĂȚI NEPLĂCUTE I :

Probleme de propagare și de blocare

6.1. INTRODUCERE

În capitolul anterior au fost tratate probleme legate de circuitele secvențiale, abordate strict formal la nivelul unor structuri ideale. A fost vorba de o simplificare ce a permis prezentarea tehnicilor de proiectare. Va trebui să ne îndreptăm, însă, atenția și asupra unor probleme concrete, suplimentare, ce apar în realizarea practică a acestor circuite, ce nu se comportă întotdeauna în modalitatea ideală dorită de noi. Acest capitol conține prezentarea unei serii de fapte concrete de care trebuie să țină cont proiectantul de circuite secvențiale pentru a putea concepe scheme demne de încredere.

Deși rar se recunoaște, multe defectări — în particular apărute în sistemele noi — sînt o consecință a greșelilor de proiectare. Deoarece majoritatea problemelor discutate în acest capitol se pun în evidență în *situații limită* de funcționare, aspectele legate de proiectarea logică *nu apar cu necesitate în primele faze ale procesului* de evaluare a bunei funcționări a schemei. În cazul unei variații normale a parametrilor componentelor folosite, problemele de acest tip nu apar decît ocazional în producția ulterioară. De regulă, înlocuirea unei componente poate soluționa această problemă — chiar dacă componenta înlocuită este în limita specificațiilor de catalog. Deoarece aceste probleme sînt atît de perfide, trebuie să fim extrem de atenți pentru a le putea elimina chiar din timpul etapelor inițiale ale proiectării.

6.2. LOGICA ASINCRONĂ ȘI PROPAGĂRILE

Majoritatea bistabilelor și circuitelor MSI ce conțin bistabili au intrări de *CLEAR* și/sau *PRESET* asincrone. Chiar dacă aceste intrări se dovedesc deosebit de utile pentru o serie de operații, ele pot totuși deveni foarte periculoase pentru buna funcționare a schemelor dacă sînt folosite incorect. În logica sincronă, toți bistabilii dintr-o schemă comută în același moment. Fiecare bistabil trece în noua stare în baza condițiilor stabilite strict *înaintea tranziției* active a ceasului. Atunci cînd bistabilele dintr-un montaj comută, datorită întîrzierilor diferite în comutare și în propagarea prin porți, *apar o serie de semnale false în circuit.*

În figura 6-1 este prezentat un exemplu în care ieșirea circuitului AND, căruia i se aplică la intrări ieșirile A și B a doi bistabili, *trece pentru scurt timp*, imediat după aplicarea impulsului de ceas bistabililor, printr-o stare ce nu corespunde conținutului acestora ca urmare a tranziției. În cazul unei

structuri sincrone acest puls scurt („glitch") nu mai poate avea nici un efect, deoarece ieşirea circuitului AND este luată în consideraţie numai la următorul front activ al impulsului de ceas. Dacă, însă spre exemplu, semnalul $A \cdot B$ este conectat la o intrare asincronă de ştergere (CLEAR) a unui bistabil, acesta va fi resetat, cu toate că dorim acest lucru numai în cazul în care rezultatul final (într-o structură ideală) ar fi $A \cdot B = 1$.

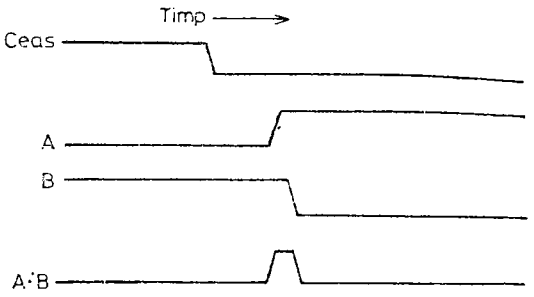


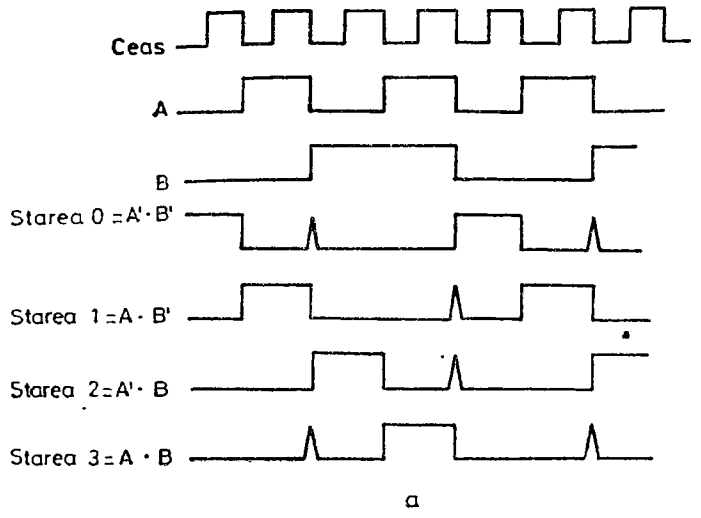
Fig. 6-1. Semnal la ieşire temporar incorect datorită coincidenţelor parazite.

Durata acestor impulsuri parazite poate fi atît de mică încît să nu reseteze bistabilul atunci cînd executăm punerea la punct a sistemului. Mai tîrziu, cînd schema va fi executată cu un bistabil A mai rapid şi cu un bistabil B mai lent, sau cu un bistabil mai sensibil pe intrarea CLEAR, sistemul va *comuta incorect de multe ori*. De multe ori modificarea temperaturii este suficientă pentru declanşarea tranziţiilor parazite. Funcţionarea incorectă pe de altă parte, poate dispare prin simpla deschidere a cutiei aparatului sau prin conectarea sondei osciloscopului în montaj.

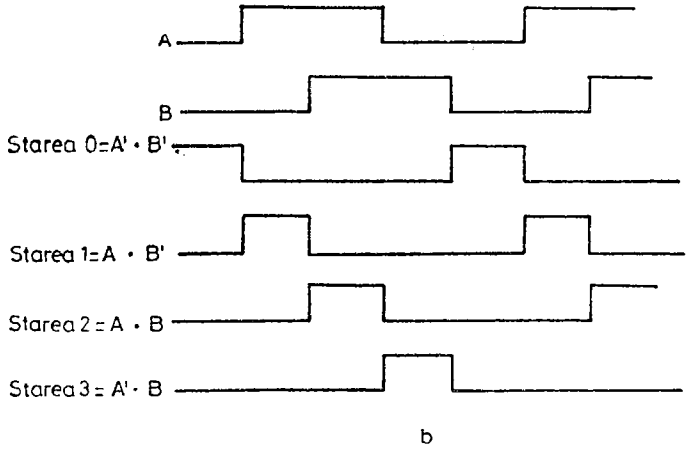
Aceste categorii de probleme pot fi evitate dacă procedăm cu foarte mare grijă în faza de proiectare. Cel mai corect este să se caute folosirea logicii sincrone şi *să se evite tentaţia de a folosi intrările asincrone de PRESET şi CLEAR cu excepţia cazurilor în care aţare funcţia simplă de ştergere sau presetare*. Deşi, în general prin uilizarea intrărilor asincrone se poate economisi destul de multă „logică", trebuie să procedăm cu multă prudenţă, deoarece logica este ieftină iar un fiasco scump. În mod normal, utilizarea comenzilor asincrone pentru *„iniţializarea"* sistemelor folosind semnele sigure, curate, ce nu rezultă din decodificări sau alte funcţii logice, oferă suficientă siguranţă deoarece în acest fel nu se introduc tranziţii parazite în sistem.

De remarcat faptul că diferenţa de întîrziere (figura 6-1) apare în cele mai multe cazuri atunci cînd cele două bistabile, A şi B, sînt discrete, deoarece, atunci cînd unul din bistabile comută cele două ieşiri Q şi Q', nu comută sincron, între ele existînd o diferenţă dată de propagarea printr-o poartă logică. Astfel, la un bistabil TTL, comutarea ieşirii Q' în **0** logic se produce după comutarea ieşirii Q în **1** logic cu timpul dat de propagarea printr-o poartă logică. În mod uzual, circuitele MSI nu au ieşiri şi pentru semnalele negate, astfel că pentru obţinerea lor sînt necesare circuite de inversare. Semnalele inversate vor comuta, deci, *după* cele neinversate. Deşi întîrzierile la ieşirile aceluiaşi circuit MSI sînt în general împerecheate între ele, vor apare totuşi diferenţe datorită sarcinilor diferite pe care lucrează aceste ieşiri, la care se adaugă şi variaţiile inerente în limitele specificaţiilor. La memoriile fixe (ROM) pot apare fenomene tranzitorii parazite, chiar dacă comută un singur bit de intrare, datorită măririi întîrzierilor ce apar prin generarea adresei complementate la intrarea decodificatorului de intrare prin intermediul unor inversoare.

Formele de undă din figurile 6-1, 6-2 şi 6-3 sînt exemple de *diagrame temporale (forme de undă)*, ce constituie un instrument deosebit de util în proiectare. Prin prezentarea semnalelor sub forma în care ar apare pe un osciloscop cu mai multe spoturi. putem obţine de obicei o mai uşoară înţelegere a opera-



a



b

Fig. 6-2. *(a)* Tranziții datorate decodificărilor parazite a ieșirilor unui numărător. *(b)* Decodificarea ieșirilor unui numărător Moebius (nu apar tranziții incorecte).

țiilor dintr-un sistem secvențial. Prin *accentuarea posibilelor întîrzieri între diverse semnale precum și a regimurilor tranzitorii parazite*, ce pot apare imediat după aplicarea impulsului de tact, vom putea evidenția posibilitatea apariției unor efecte disfuncționale.

Figura 6-2*a* prezintă formele de undă pentru un numărător divizor cu 4 și ieșirile unui decodificator conectat în continuare. Au fost reprezentate tranzițiile parazite ce pot apare la fiecare din cele patru ieșiri ale decodificatorului, cu toate că în realitate este puțin probabil, ca să apară toate. Semnalul parazit în starea zero, spre exemplu, va apare dacă A' trece în unu înainte cu B' să treacă în zero. Formele de undă presupuse de unele din celelalte semnale

parazite pot să nu prezinte acest aspect, dar esențial este faptul că trebuie să presupunem că tranzițiile parazite *pot* apare. Reprezentînd în formele de undă *posibilitățile* de apariție a tranzițiilor parazite, se evită o serie întreagă de probleme ce ar putea afecta acuratețea proiectării.

Deseori este necesar un semnal decodificat, curat (ce nu prezintă nici un fel de tranziții parazite), spre exemplu, pentru construirea unui ceas polifazic. În figura 6-2*b* este reprezentat un numărător Moebius, ales datorită faptului că tranziția dintr-o stare în alta presupune de fiecare dată *modificarea numai a unui singur bit.* Pentru secvențe de numărare mai lungi poate fi folosită o secvență Gray reflectată, de genul aceleia din figura 5-6.

O altă problemă care apare în mod curent în proiectarea logicii asincrone poate fi următoarea: dacă se alege soluția ștergerii (CLEAR) unui bistabil cu un semnal provenit de la un circuit combinațional a cărui funcție logică *este adevărată numai dacă bistabilul este setat* ($Q = 1$), semnalul CLEAR va dispare imediat după ce bistabilul a început să fie resetat. În mod normal, întîrzierea prin circuit va face ca CLEAR să rămînă suficient de mult pentru a desăvîrși resetarea, dar este posibil, în egală măsură, ca sistemul să nu se reseteze. Chiar dacă resetarea a fost efectuată, semnalul CLEAR va deveni foarte îngust, de forma unei tranzitii parazite. Dacă folosim același semnal și pentru acționarea *altor* bistabili. iar aceștia comută *mai încet* decît bistabilul care determină scurtarea impulsului CLEAR, ei nu vor mai fi resetați.

Înainte ca numărătoarele presetabile, complet sincrone, ieftine, să fie disponibile, se practica în mod curent construcția numărătoarelor cu „*n* stări" decodînd a $(n+1)$-a stare și conectînd semnalul obținut pe intrarea *asincronă* de CLEAR. Deoarece bistabilele din același cip au de regulă timpii de resetare de valori foarte apropiate (sînt bine împerecheați), o astfel de schemă funcționează de obicei corect. Sînt dese, însă, cazurile periculoase, în care împerecherea acestor bistabili fie că nu este perfectă (chiar dacă rămîn în limitele specificațiilor de catelog), fie că împerecherea se strică deoarece acești bistabili sînt foarte *neechilibrat încărcați.* În aceste situații o astfel de schemă nu este recomandabilă. Deoarece acum putem dispune la prețuri foarte convenabile de numărătoare presetabile complet asincrone (de exemplu ca acela din figura 5-12) este preferabil să nu ne asumăm riscurile unei astfel de scheme.

Deși tehnicile de proiectare *asincronă* sînt în general bine puse la punct (vezi Ref. 1), sînt de regulă atît de greu de aplicat, încît utilizarea lor este limitată în principal la proiectarea structurii interne a circuitelor integrate. Pentru sistemele digitale obișnuite *logica sincronă se va utiliza chiar și pentru cele mai simple funcții deoarece reduce în mod semnificativ incidența erorilor de proiectare.*

6.3. SISTEME LOGICE SINCRONE CU INTRĂRI ASINCRONE

Chiar dacă vom proiecta numai sisteme logice sincrone, *intrările* acestor sisteme vor comuta asincron, în momente de timp independente de ceasul intern. Există o foarte scurtă perioadă de timp, anterioară frontului activ al ceasului, în care intrările unui bistabil trebuie să fie stabile pentru a fi efectuată o tranziție corectă. Acest interval se numește **timp de stabilire (set-up time)** și este egal, aproximativ, cu întîrzierea unei porți logice. Dacă una din intrările bistabilului comută în acest interval starea finală a bistabilului va fi

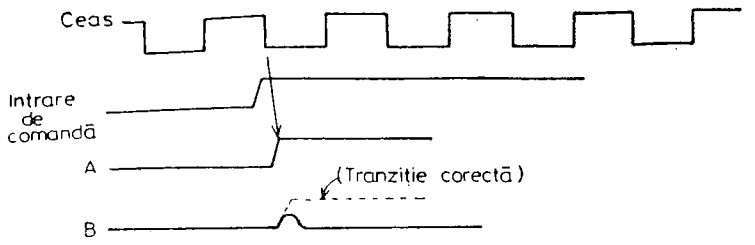


Fig. 6-3. Comutarea incorectă a stării dată de doi bistabili în cazul unei comenzi asincrone aplicate la intrare.

nedeterminată, raportată la stările intrărilor, depinzînd, parțial, de cît de rapid este exemplarul supus acestei experiențe. Dacă avem un sistem la care *mai mulți* bistabili trebuie să comute ca urmare a tranziției uneia din intrări, este posibil ca numai exemplarele cele mai rapide să comute, dacă intrarea se modifică cu foarte puțin timp înaintea impulsului de ceas . *Rezultă astfel o tranziție într-o stare eronată.*

Formele de undă din figura 6-3 prezintă un exemplu de tranziție eronată : atunci cînd intrarea de control trece în 1 logic, bistabilii *A* și *B* trebuie să comute în 1. Timpul de set-up al bistabilului *A* este mai mic decît al bistabilului *B*, astfel încît dacă intrarea de control comută foarte aproape de frontul activ al ceasului, *A* va fi setat iar *B* nu. Starea obținută va fi 10 în loc de 11. În sistem se introduce astfel o informație total incorectă. În plus, fenomenul este cu atît mai necontrolabil cu cît depinde și de diferența între cei doi timpi de set-up ai celor doi bistabili. Dacă diferența între cei doi timpi de set-up este de numai 2 ns (nanosecunde) și perioada ceasului este de 500 ns, statistic, *odată la 500/2 = 250 de tranziții corecte are loc și o tranziție incorectă*, ca urmare a comutării în 1 a intrării de control.

Dacă în cazul prototipurilor circuitele pot fi foarte bine selectate în așa fel încît acest efect să fie practic anulat, în cazul unei producții de serie acest lucru este imposibil. Cea mai bună soluție în acest caz este ca *NICIODATĂ SĂ NU MODIFICĂM STAREA A MAI MULT DE UN SINGUR BISTABIL CA URMARE A TRANZIȚIEI UNEI INTRĂRI ASINCRONE.*

Tendința de comutare (cocoașa) ce apare la ieșirea *B*, în figura 6-3 pune în evidență o altă problemă care apare în proiectarea sistemelor cu intrări asincrone. În acest caz bistabilul *B* a avut tendința de a comuta dar s-a reîntors în starea anterioară comenzii. Probabilitatea ca acest efect să genereze probleme neplăcute este foarte mică, exceptînd cazul în care întregul sistem funcționează la viteza maximă permisă de circuite. În acest caz această tranziție poate fi interpretată în restul sistemului ca o tranziție valabilă și nu ca una parazită, obținîndu-se o evoluție internă în neconcordanță cu procesele din circuitele de intrare. Problema poate fi soluționată prin introducerea unui bistabil suplimentar sau a unei stări suplimentare (din categoria celor ce presupun modificarea unui singur bit), pentru a întîrzia cu un tact acțiunea acestei intrări.

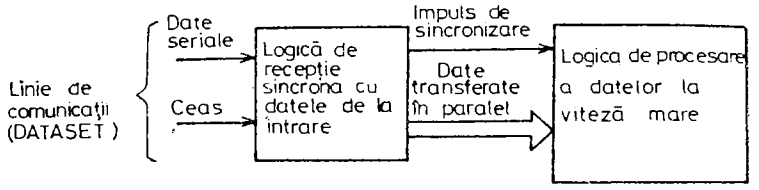


Fig. 6-4. Sistem cu ceasuri multiple.

Uneori este convenabil de adoptat soluţia folosirii mai multor ceasuri în sistem. Spre exemplu, în figura 6-4 este prezentat un sistem ce foloseşte un ceas de mică frecvenţă (lent), pentru a recepţiona bit cu bit datele serie într-un subsistem de intrare. După ce un cuvînt este complet asamblat, el este transferat în subsistemul de prelucrare ce funcţionează cu un ceas foarte rapid. Intrarea de *strob* în cel de al doilea subsistem este tratată asincron. Intrarea serie de date în subsistemul de intrare este tratată sincron cu ceasul subsistemului de recepţie a datelor.

De multe ori intrările provin de la dispozitive mecanice, cum ar fi comutatoarele, avînd o serie de caracteristici improprii tratării logice. În mod obişnuit, *un contact mecanic prezintă după comutare tranziţii necontrolate un interval de timp de 5 pînă la 50 ms* (vezi figura 6-5, *a*). Un astfel de fenomen poate produce perturbarea completă a sistemului dacă nu este tratat corespunzător. Dacă acest semnal, neprelucrat, este introdus într-un sistem ce reacţionează foarte rapid, operaţia declanşată se va termina înainte ca tranziţiile necontrolate să înceteze, determinînd repetarea nedorită a aceleiaşi comenzi. Acest lucru poate fi evitat prin introducerea comutatorului în sistem folosind circuite sau stări suplimentare. *O stare va fi introdusă pentru a evidenţia prima tranziţie a comutatorului, iar o altă stare pentru a marca revenirea comutatorului la poziţia iniţială.* Cea mai simplă soluţie o constituie circuitul bistabil asincron (latch) din figura 6-5 *b* construit din două circuite

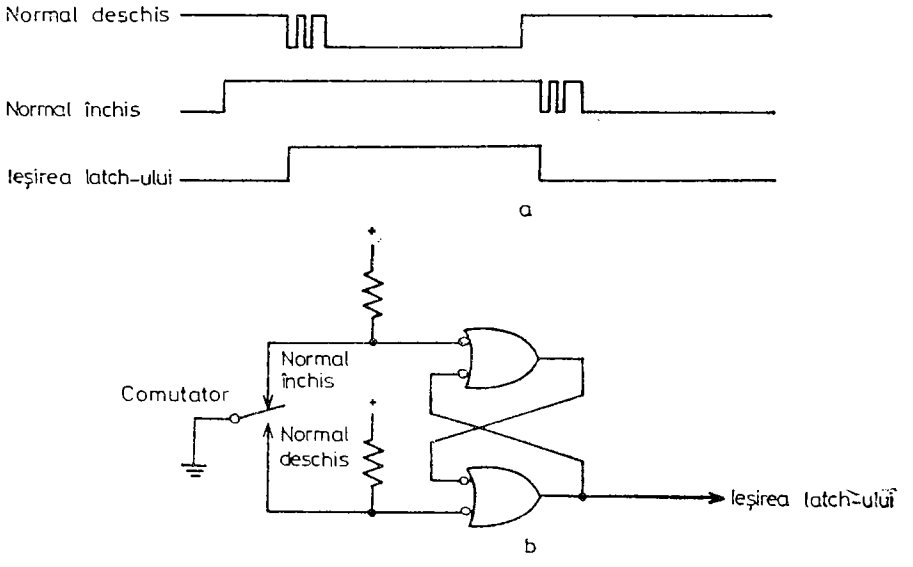


Fig. 6-5. Eliminarea tranziţiilor parazite ale unui contact mecanic.

NAND, ce înlătură oscilațiile suplimentare generînd la ieșire un semnal curat. Semnalele trebuie reprezentate în diagramele de timp cu oscilațiile suplimentare, ca în figura 6-5 *a*, cu excepția cazurilor în care s-au folosit latch-uri pentru „curățire". De regulă oscilațiile durează un interval de timp mai mic decît timpul în care contactul stă deschis fapt pentru care nu sînt dependente de acest timp.

6.4. DEFAZAREA CEASULUI

Orice sistem logic sincron se bazează pe faptul că fiecărei celule de stocare i se aplică semnalul de ceas în *același moment*. În practică acest principiu nu se poate fi respectat întotdeauna. În sistemele complexe sînt folosite mai multe circuite de comandă pentru distribuirea impulsurilor de ceas. Deoarece fiecare are timpul său de întîrziere, impulsurile de ceas nu mai sînt absolut sincrone la intrările tuturor bistabilelor. Acest tip de eroare este denumită *defazarea ceasului (clock skew)* și poate genera erori grave dacă este excesivă.

Figura 6-6 *a* ilustrează un transfer incorect de date între doi bistabili datorat unui defazaj prea mare între impulsurile de ceas. Circuitul reprezintă de fapt un registru de deplasare de doi biți cu 0 pe intrarea primului bistabil ($B1$) și cu starea inițială 11. Ne așteptăm, ca în mod normal, după primul impuls de ceas starea să devină 01 iar după cel de al doilea 00. Din cauza defazării celor două impulsuri de ceas peste o anumită limită starea registrului după primul impuls devine direct 00. Deficiența apare datorită faptului că ceasul la $B1$ este mult *anterior* ceasului aplicat lui $B2$ și *noua* stare a $B1$ este cea care este încărcată în $B2$.

De fapt, o ușoară defazare a ceasurilor poate fi tolerată de circuit datorită **timpului de propagare** prin $B1$ (întîrzierea între frontul activ al ceasului și modificarea ieșirii bistabilului). Pentru o funcționare corectă, defazajul cea-

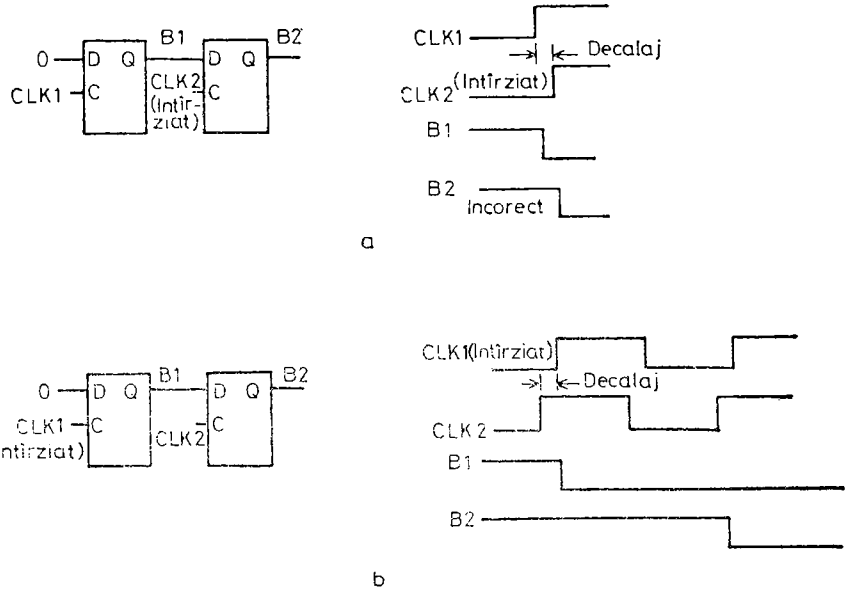


Fig. 6-6. *(a)* Transfer incorect cu ceasul întîrziat. *(b)* Transfer corect **de la** ceasul întîrziat.

sului trebuie să fie mai mic decît timpul de propagare prin $B1$, din care trebuie să scădem *timpul de menținere (hold time)* a intrării $B2$. Cele mai multe circuite sînt garantate pentru un timp de menținere (timpul cît trebuie menținută constantă intrarea după frontul activ al ceasului) nul, deci *vom putea considera în aceste cazuri că defazajul maxim admisibil este egal chiar cu timpul de propagare minim*. Din nefericire, specificațiile de catalog pentru majoritatea circuitelor integrate nu garantează timpul *minim* de propagare. Acest fapt face ca proiectarea pentru cazul cel mai defavorabil să fie posibilă numai dacă toate intrările de ceas sînt conectate efectiv la ieșirea aceluiași generator de ceas.

După cum se poate observa și din figura 6-6 *b*, problema defazajului ceasului apare numai într-un singur sens. Datele se transferă corect *într-una din direcții* chiar în prezența defazajului ceasului atît timp cît transferul se face de la ceasul care apare mai tîrziu. În multe cazuri, însă, este necesar transferul în ambele direcții. Singura soluție ar fi, deci, să se mențină defazajul între ceasuri foarte mic, folosind un singur generator (driver) de putere pentru ceas sau drivere foarte rapide bine împerecheate.

Defazarea ceasului constituie o mare problemă în cazul utilizării circuitelor TTL *deoarece cele mai multe circuite MSI comută, de regulă, pe frontul pozitiv al ceasului în timp ce bistabilele J—K comută pe frontul negativ* al ceasului. Din acest motiv folosirea unui ceas unic pentru toate circuitele este imposibilă (în afară de cazul în care transferul este unidirecțional, spre exemplu de la *J—K*-uri la MSI-uri). *O soluție poate fi oferită de utilizarea unor bistabile cu timpi foarte mici de propagare* și de generarea ceasului pentru celelalte circuite prin inversarea ceasului de la *J—K*-uri cu circuite ce au timpul de propagare garantat mai mic decît al bistabililor (ex. : bistabile *J—K* tip 74112 și inversoare din seria 74S). O soluție mai bună va fi utilizarea bistabilelor *J—K* ce comută pe frontul pozitiv (de exemplu 74109).

În structurile care folosesc circuite foarte rapide (cum ar fi **ECL** sau **Schottky TTL**) defazări semnificative ale ceasului pot fi date de întîrzierea determinată de propagarea pe firele de conexiuni. În figura 6-7 *a* se prezintă în

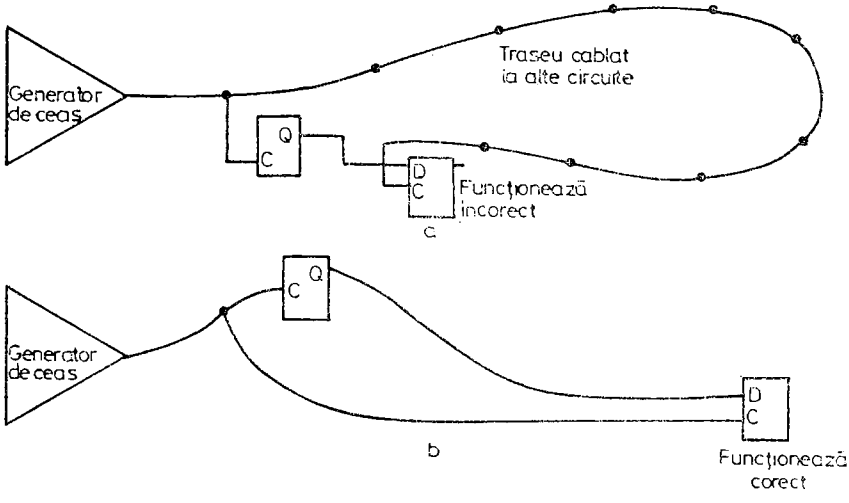


Fig. 6-7. Defazarea ceasului datorită cablajului. *(a)* Conectarea ceasului printr-un traseu ce realizează o buclă prea mare, poate cauza probleme de decalare : *(b)* Conectare corectă care elimină decalările.

calitate de exemplu o modalitate incorectă de distribuire a ceasului. Deoarece întîrzierea pe firele de conexiune este de ordinul a 5...6 ns/m, defazajul între cele două impulsuri de ceas poate deveni semnificativ într-o configurație a cablajului neglijent proiectată. O buclă de ordinul zecilor de centimetri devine semnificativă pentru bistabilii care comută în aproximativ 2 ns (timp minim de propagare pentru 74S112), iar pentru circuite ECL efectul va fi și mai puternic. După cum rezultă din figura 6-7 *b* logica sincronă poate funcționa corect chiar pentru lungimi diferite ale traseului de propagare a ceasului cu condiția de a fi similar cu traseul de propagare a datelor între cei doi bistabili, realizîndu-se astfel o împerechere a celor doi timp; de propagare. În acest fel întîrzierea pe conexiunile de date anulează efectul defazajului ceasului.

Trebuie observat faptul că în cazurile în care bistabilii sînt interconectați prin intermediul unor porți, întîrzierea semnalului crește, ceea ce permite mărirea defazajului minim admisibil între ceasurile lor. În sistemele de mare viteză, traseele impulsurilor de ceas trebuie însă optimizate în funcție de traseele de date în care însă nu se intercalează porți.

6.5. FRECVENȚA MAXIMĂ A CEASULUI

Majoritatea bistabililor și a circuitelor MSI au specificate în catalog frecvența maximă de lucru. Acest parametru nu poate fi folosit însă ca atare în estimarea structurilor proiectate. În primul rînd aceste mărimi nu sînt întotdeauna date ca valori limită ci numai ca valori tipice (ceea ce înseamnă de fapt medii). Cu alte cuvinte, *o parte din circuite vor lucra la frecvențele indicate iar altele nu.* Deoarece noi dorim ca toate schemele produse pe baza proiectului nostru să funcționeze corect, trebuie să luăm în considerație cazul *cel mai defavorabil,* al frecvenței *minime* garantate de producătorul de circuite.

Chiar valoarea minimă (în cazul cel mai defavorabil) a frecvenței limită de funcționare specificată de fabricant este de două pînă la trei ori mai mare decît frecvența la care vor funcționa în schemele practice din sistem. Motivarea acestui fapt o aflăm în circuitele combinaționale adiționale ce sînt utilizate pentru interconectarea diferitelor circuite secvențiale. Timpul de propagare prin porți se *sumează* la cel asociat bistabililor, rezultînd timpul minim între două tranziții. Frecvența maximă a ceasului se va calcula ținînd cont de : *întîrzierea prin bistabili + întîrizerea prin porți + timpul de set-up.*

În figura 6-8 este prezentat un sistem logic numai cu două nivele de porți intermediare, număr minim pentru sistemele practice. Dacă vom lua, spre exemplu, bistabili de tip 74S74 și porți de tip 74S00, perioada maximă va fi 9 ns (întîrzierea bistabilului) + 5 ns (prima poartă) + 4,5 ns (a doua poartă) + 3 ns (timpul de set-up) = 21,5 ns. Rezultă valoarea maximă a frecvenței ceasului : 1/21 ns = 46 MHz. Frecvența minimă pentru bistabilul 74S74 este *specificată în catalog ca fiind de 75 MHz.* Dacă vom considera circuitele logice combinaționale ca introducînd patru nivele logice, datorită factorizărilor, predecodificărilor, utilizării multiplexoarelor, vom micșora numărul circuitelor integrate utilizate dar vom crește perioada ceasului la 9 + 5 + + 4,5 + 5 + 4,5 + 3 = 31 ns ceea, ce reprezintă o frecvență de ceas de 1/31 ns = *32 MHz* adică mai puțin de jumătate din valoarea specificată. Rezultă că trebuie să realizăm un compromis între viteză și prețul de cost al logicii.

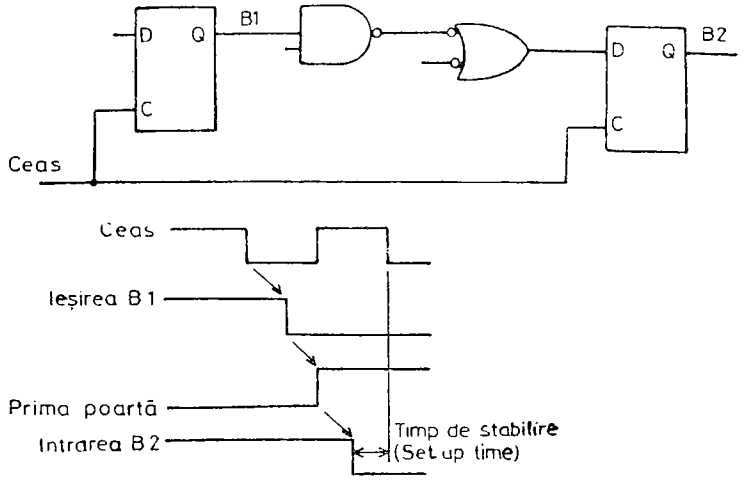


Fig. 6-8. Frecvenţa maximă de lucru.

Trebuie observat că am folosit în calculele anterioare pentru o poartă, ca timp de întîrziere, atît 5 ns cît şi 4,5 ns. Acest lucru este motivat de faptul că timpul de întîrziere depinde de sensul în care comută ieşirea. Deoarece fiecare nivel logic produce o inversare, vom presupune că semnalul se propagă alternativ prin tranziţii pozitive (4,5 ns) şi tranziţii negative (5 ns).

Unele din bistabilele *J—K* integrate *mai vechi* (ex. : 7470 pînă la 7374, 7476, 7478, 74107) *nu* comutau numai ca urmare a tranziţiei frontului (edge triggered), însemnînd că *pe toată durata palierului dinaintea frontului activ intrările trebuiau să rămînă constante.* Acest fapt afectează defavorabil frecvenţa maximă de lucru, deoarece *timpul de set-up ce trebuie luat în considerare este foarte mare, depinzînd şi de forma impulsului de ceas.* Cea mai bună cale de a ocoli această dificultate este aceea de a folosi noile tipuri de bistabile J—K ce nu prezintă această restricţie, timpul de set-up nedepinzînd de impulsul de ceas (ex. : 74112, 74109, sau 71S112).

În unele sisteme în care se doreşte o viteză mare de transfer a datelor, utilizarea a numai două nivele logice între circuitele secvenţiale este nepractică. În acest caz va fi utilizat procedeul *pipeline**. Spre exemplu, în figura 6-9 este reprezentat un sistem ce presupune patru nivele logice, funcţionînd la viteza unui sistem ce are numai două nivele logice. Procedeul se bazează pe *resincronizarea datelor* după cel de al doilea nivel logic prin introducerea unor bistabile suplimentare. Desigur, este introdus un nivel suplimentar care face

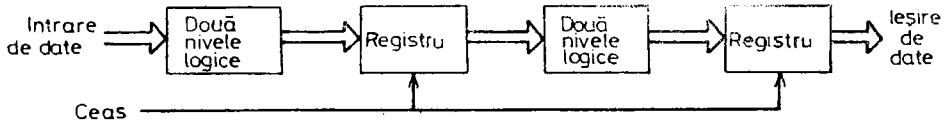


Fig. 6-9. Structură pipeline : patru niveluri combinaţionale funcţionînd la frecvenţa de lucru a două niveluri.

* conductă, în limba engleză (n.t.)

ca datele să apară la ieșire cu un impuls de ceas mai tîrziu, dar viteza de transfer prin sistem (prin „conductă") este echivalentă cu cea a unui sistem cu numai două nivele logice intermediare. Evident în exemplul nostru fragmentarea sistemului a fost arbitrară, în cazul unor structuri concrete fiind posibile oricîte nivele între bistabile și oricîte resincronizări suplimentare.

6.6. STĂRI DE BLOCARE ȘI SISTEME CU AUTOINIȚIALIZARE

Orice sistem ce conține elemente de memorie trebuie să fie prevăzut în mod normal cu un semnal de inițializare care la conectarea sursei de alimentare face ca sistemul să treacă într-o stare bine determinată. *Ștergerea generală*, sau *inițializarea*, sînt semnale generate automat prin intermediul unor circuite de întîrziere comandate de un semnal ce indică depășirea unui anumit nivel al sursei de alimentare. Fără acest mecanism de inițializare sistemul nu va porni din starea care trebuie, funcționînd prost de la început. Această problemă este însă minoră față de cele puse de *stările nedefinite* (nesemnificative) ale sistemului, din care acesta nu poate realiza o tranziție corectă, sau nici un fel de tranziție. Sistemul odată ajuns într-o astfel de stare poate continua să funcționeze necorect un timp indefinit (pînă *sistemul este din nou inițializat*).

Un exemplu foarte bun în acest sens îl constituie numărătorul Moebius descris în Secțiunea 5.5. În figura 5-7 *d* a fost prezentată secvența de numărare formată din șase stări pentru un numărător Moebius de trei biți. Deoarece cu trei biți se pot defini opt configurații, se pune întrebarea: ce se întîmplă în celelalte două stări ? Dacă presupunem că numărătorul se află în starea 101, următoarea va fi 010, care este o altă stare nedefinită, iar următoarea comutare se va face înapoi în 101 (vezi fig. 6-10 *b*). Numărătorul se va bloca în ciclul definit de aceste două stări. Desigur, sistemul intrat în acest ciclu prezintă simptomele unei pene. Este un lucru inevitabil, ca un *numărător Moebius de mai mult de doi biți* sau orice alt numărător de n bistabili cu mai puțin de 2^n stări să posede *secvențe de numărare suplimentare nedorite*. Aceasta nu înseamnă că orice numărător cu stări extraciclice prezintă aceste deficiențe.

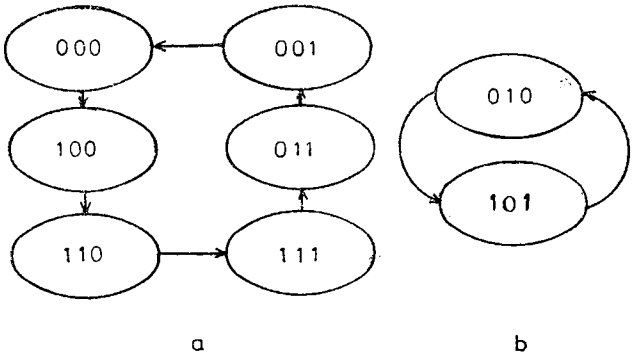


Fig. 6-10. Diagramă de stări completă pentru un numărător Moebius de trei biți ; *(a)* secvența utilă ; *(b)* secvența nedorită.

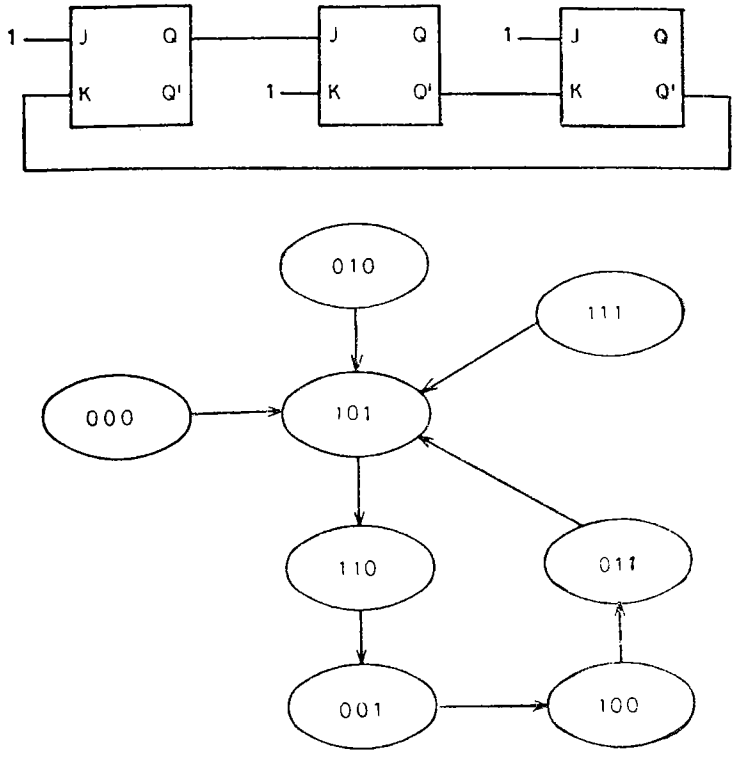


Fig. 6-11. Divizor cu 5 cu automorsarea corectă a funcționării.

Spre exemplu, divizorul cu 5 din figura 6-11 este un numărător cu *autoinițializare*, deoarece din *orice stare* se poate ajunge după un număr finit de tranziții într-o stare a ciclului util. Diagramele de stări utilizate în proiectarea numărătoarelor vor fi astfel concepute încît *să fie cuprinse toate stările* (ex.: figurile 6-10 și 6-11) în așa fel încît să nu fie permisă apariția stărilor de blocare descrise anterior.

Dacă ne vom baza pe semnalul de inițializare pentru a evita „agățarea" sistemului în cicluri nesemnificative *ne vom baza de fapt pe un procedeu foarte riscant*. Orice bistabil se poate comuta accidental în timpul funcționării ca urmare a introducerii în montaj a sondelor de testare, datorită semnalelor parazite, datorită conectării în montaj a unor circuite suplimentare. Aceste comutări vor produce o funcționare *temporar* necorectă *ce se va putea autocorecta numai dacă am prevăzut mecanisme de autoinițializare* și nu ne-am limitat la inițializarea schemei odată cu alimentarea în curent continuu. În cazul unui numărător MSI binar autocorectarea este automată deoarece orice stare a acestuia face parte din secvență. Dacă proiectăm un numărător controler de stare folosind un numărător MSI trebuie să fim siguri că semnalul de control al numărării este activ în stările nesemnificative în mod necondiționat, ceea ce se reduce la conectarea la 1 a intrărilor corespunzătoare stărilor

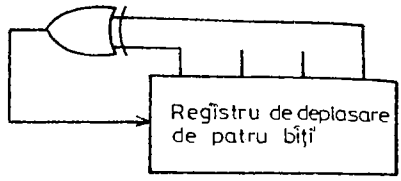
nesemnificative (fig. 5-14 *a*). În exemplul în care am folosit bistabile discrete (fig. 5-9) ocolim blocarea în stările nesemnificative utilizînd *X*-urile din diagrama J_A (fig. 5-10 *b*).

Acest procedeu care permite autoinițializarea sistemelor poate fi extins și dincolo de proiectarea numărătoarelor. *Spre exemplu experiența arată că într-o centrală telefonică informația memorată se poate altera în timp* dacă nu există un proces prin care să se înlăture datele incorecte. De exemplu informația referitoare la un apel telefonic este menținută pînă la încheierea convorbirii după care trebuie anulată. Dacă o funcționare *incorectă* face ca terminarea convorbirii să nu fie detectată informația relativă la apel rămîne stocată. În cele din urmă se acumulează din ce în ce mai multe legături inexistente ceea ce duce la supraîncărcarea sistemului. Adăugînd o serie de circuite de „supervizare" care testează din cînd în cînd starea sistemului și corectează erorile în anumite zone, se introduce în sistem o funcție similară *autoinițializării* discutate anterior. În unele calculatoare care rulează continuu aceleași programe există mecanisme care reîncarcă toate programele ori de cîte ori apar erori în funcționare, căderi ale sistemului (system crash).

EXERCIȚII

1. Construiți diagramele de timp pentru numărătorul divizor cu 5 din figura 6-11, incluzînd și ieșirile decodificatorului pentru cele 5 stări. Reprezentați tranzițiile parazite posibile.
2. Proiectați un numărător de 5 stări cu decodificatorul asociat, astfel încît schema să poată fi utilizată ca un *ceas* cu 5 faze.
3. Care din tranzițiile parazite din figura 6-2*a* va dispare dacă A se va modifica întotdeuna înaintea lui B (această situație poate apare dacă A este folosit ca *ceas* pentru B)?
4. Care stări din figura 5-9 pot fi decodificate fără a exista pericolul tranzițiilor parazite?
5. Care stări din figura 5-13 pot fi decodificate fără a exista pericolul tranzițiilor parazite?
6. (*a*) Presupunînd că nici unul din semnalele de control de la intrarea numărătorului controler de stare din figura 5-9 nu este sincronizat cu ceasul sistemului, există posibilitatea unor tranziții incorecte în sistem?
   (*b*) Dar în cazul alocării stărilor din figura 5-13?
7. Dacă semnalul READ din figura 5-9 este introdus direct de la un comutator mecanic caracterizat printr-un regim instabil, după conectare, timp de 20 ms, pot apare disfuncționalități în sistem? (Se consideră că REA TAPE durează 50 ms).
8. Apar probleme suplimentare dacă în figura 5-9, în loc de a folosi intrarea J_C, acționăm direct asupra intrării PRESET′ de la bistabilul C, cu semnalul $(A \cdot B \cdot C' \cdot \text{END})'$?
9. (*a*) Apar probleme suplimentare dacă în sistemul definit în figura 5-9 înlăturăm termenul C'. $C' \cdot \text{READ}$ din J_A (figura 5-10) și înlocuim acțiunea lui prin presetarea directă a bistabilului C cu semnalul $(A' \cdot B' \cdot C' \cdot \text{READ})$?
   (*b*) Dacă am utiliza pentru aceasta $(B' \cdot C' \cdot \text{READ})'$, ce se va întîmpla dacă butonul READ ar fi menținut apăsat și după ce ultimul caracter a fost transmis?
10. Redesenați figura 6-3 în condițiile în care timpul de menținere (set-up) al bistabilului B este mai mic decît al bistabilului A.
11. (*a*) Care este probabilitatea de eroare pentru circuitul din figura 6-3 dacă diferența între timpii de set-up ai celor doi bistabili este de 1 ns iar perioada ceasului este de 10 μs?
    (*b*) Dacă semnalul de control este acționat numai o dată pe secundă, cît de des pot apare erori în sistem?
12. Apar probleme suplimentare în cazul numărătorului controler de stare din figura 5-13 dacă semnalul ACK nu este sincronizat?
13. Modificați diagrama de stări din figura 5-13 ținînd cont de faptul că semnalele READ și ACK pot fi asincrone.

Fig. 6-12.



14. Desenați celelalte secvențe de numărare ale numărătorului Moebius de patru biți definit în figura 5-7*a*.
15. Reproiectați numărătorul Mebius din figura 5-7*a* în așa fel încît să se poată autoinițializa.
    (*a*) Cu bistabili *D*.
    (*b*) Cu bistabili $J-K$.
    (*c*) Utilizînd intrările CLEAR fără riscul apariției unor tranziții parazite.
16. Refaceți exercițiul 15 pentru un numărător Moebius de trei biți.
17. Proiectați un circuit care generează un impuls cu lungimea de o perioadă a ceasului ori de cîte ori este acționat un contact mecanic.
18. Proiectați un sistem, folosind un numărător MSI, ce generează un plus cu lungimea de 13 perioade de ceas la fiecare acționare a unui comutator mecanic.
19. Proiectați un circuit ce va genera un impuls de ceas la fiecare acționare a butonului *STEP* și permite generarea continuă a ceasului dacă *RUN* este conectat, oprind ceasul pentru comutatorul *RUN* neconectat. Nu sînt permise trunchieri ale impulsurilor de ceas.
20. Pot fi transferate date în sens invers fără eroare în schema reprezentată în figura 6-7*a*?
21. (*a*) Desenați un numărător Moebius de doi biți, similar cu cel din figura 6-8, realizat cu un bistabil $J-K$ ce comută pe frontul negativ și un bistabil *D* ce comută pe frontul pozitiv.
    (*b*) Care este întîrzierea maximum admisibilă a inversorului folosit pentru negarea ceasului?
22. Desenați formele de undă, similare celor din figura 6-8, dar considerînd că B2 este un bistabil $J-K$ cu tranziția necondiționată numai de frontul impulsului de ceas, ci și de stabilitatea datelor pe durata palierului anterior frontului activ. Circuitul lucrează la viteza maximă cu un ceas ce are factorul de umplere de 50%.
23. Cît de lungă poate fi bucla cablajului pentru ceas în figura 6-7*a* dacă între cei doi bistabili (74S112) exsită două porți 74S00? (Presupunem 5 ns/m întîrzierea introdusă de firul de conexiune).
24. Cum trebuie gîndit un sistem construit cu circuite combinaționale din seria 74S pentru a realiza sumarea a doi octeți în 35 ns, dacă sînt necesare opt nivele de porți pentru propagarea transportului.
25. (*a*) Determinați secvența de numărare pentru circuitul din figura 6-12.
    (*b*) Modificați schema în așa fel încît să fie completată cu funcția de autoinițializare.

REFERINȚĂ

1. G. A. Maley și J. Earle, *The Logic Design of Transistor Digital Computers*, Prentice-Hall, Englewood Clissff, N. J., 1963.

BIBLIOGRAFIE

Maley, G. A. și Earle, J., *The Logic Design of Transistor Digital Computers*, Prentice-Hall, Englewood-Cliffs, N. J., 1963 (În capitolele 8 și 9 se dezvoltă tehnici de proiectare și analiză pentru sisteme asincrone).

Peatman, John, *The Design of Digital Systems*, McGraw-Hill, New York, 1972, Secțiunile 5—6...5—12.

Liu, Bede and Gallagher, „On the Metastable Region of Flip-Flop Circuits", *Proceedings of the IEEE*, April 1977, pag. 581—583.

7

LOGICA PROGRAMATĂ I
Microcalculatoare

7.1. UN CIRCUIT LOGIC UNIVERSAL

În tehnicile de proiectare a sistemelor digitale pe care le-am discutat pînă în prezent, funcția logică a fost definită de către *interconexiuni*. Elementele, porțile sau bistabilele, sînt cuplate, într-un aranjament ce produce caracteristicile de sistem dorite. În cazul circuitelor MSI, legături tipice sînt efectuate în circuitul integrat, dar funcția sistemului este totuși definită de către configurația conexiunilor externe.

Deoarece cipurile LSI conțin atît de multe elemente de circuit, a fost necesar să se conceapă un mod complet diferit de definire a funcției sistemului. În locul unei mulțimi de fire, caracteristicile logice ale unui circuit LSI sînt definite de către un *program*, ce poate fi stocat sub forma unui grup ordonat de biți în cip. Aceasta face posibil ca o singură componentă LSI standard să servească necesitățile diferite a mii de beneficiari. Similar modului în care calculatorul numeric poate fi programat pentru orice aplicație, de la calculul statelor de plată la ghidarea rachetelor, se poate acorda și computerelor LSI prin intermdiul programării, flexibilitatea necesară realizării unor componente *standardizate*, care să poată fi produse în serie mare, pentru o varietate largă de aplicații.

Microcomputerele monocip reprezintă cu adevărat categoria cea mai avansată de componente standardizate, deoarece ele constituie, în esență, o implementare LSI a subsistemului digital general de tip „cutie neagră" (vezi fig. 2-3). Un microcomputer are linii de intrare și ieșire care interacționează într-un mod ce este descris de un program intern. Sarcina proiectantului nu mai este legată de fapt de porți sau bistabile, ci de scrierea unor programe.

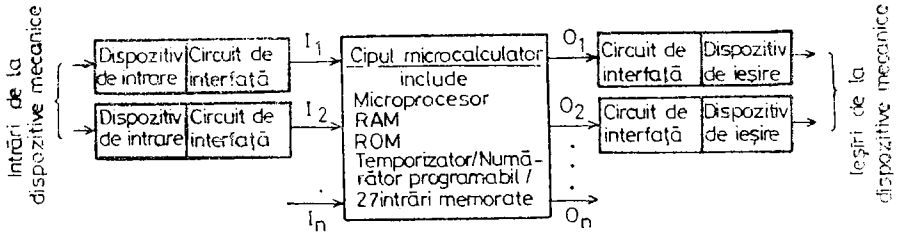


Fig. 7-1. Microcalculator într-un singur cip. Un subsistem digital complet ce poate fi gîndit ca o „cutie neagră" (black box) realizată pe un singur cip. Pentru cazul în care se concep sisteme cu memorie mare vor fi utilizate cipuri distincte pentru microprocesor și circuitele de intrare-ieșire.

Funcția care era efectuată anterior de bistabilii individuali este acum realizată de către o zonă comandată cu biții memoriei (RAM) inclusă în sistem. Programul însuși definește funcțiile logice, la fel cum o făcea interconectarea porților și bistabililor în proiectarea logică tradițională. Acest program este memorat într-un ROM, inclus, de asemenea, pe cip. Circuitele de temporizare, prezente pe cip, sînt utilizate de către program pentru a conduce operațiile dependente de timp. Programul include, de asemenea, instrucțiuni pentru inspecția liniilor de intrare și pentru modificarea stării bistabililor de ieșire, conform cerințelor aplicației.

Deoarece setul de instrucțiuni al microcomputerului include toate funcțiile logice, *se pot realiza orice funcții logice* dacă se alocă suficient *timp* și *memorie*. Cu cît sarcina este mai complexă, cu atît mai multe trepte de program sînt necesare pentru execuția ei. Deoarece fiecare treaptă de program solicită timp pentru execuție, se poate ajunge la un punct în care microcomputerul nu este suficient de rapid pentru sarcina ce trebuie executată.

Această limitare de viteză constituie motivul principal pentru care am folosit patru capitole pentru prezentarea tehnicilor tradiționale de proiectare logică. Multe procese din realitatea fizică au loc prea rapid pentru logica programată. Cînd un proces logic este divizat într-o secvență de pași, este normal să ia mai mult timp decît atunci cînd este tratat într-un singur pas, printr-un circuit special proiectat.

În cadrul tehnicilor de proiectare logică luate în considerare pînă acum, fiecare numărător și registru din sistem își poate schimba starea la *fiecare* impuls de ceas. În abordarea programată a problemei, maximul a ceea ce se poate realiza în fiecare ciclu de tact este citirea sau scrierea unui cuvînt de memorie (de obicei cu o lungime de 8 sau 16 biți). În plus, pe lîngă această limitare, mai mult decît jumătate din ciclurile de memorie sînt utilizate pentru a citi instrucțiunile programului. Astfel, *indiferent de tehnologia în care este realizat circuitul integrat, varianta programată va fi considerabil mai lentă decît cea obținută prin proiectarea logică obișnuită.*

Cu toate că multe probleme pot fi rezolvate folosind doar un singur microcomputer, sarcini mai ample și mai rapide pot fi tratate numai prin tehnicile ce presupun conexiuni hardware, pe care le-am discutat anterior. Deseori soluția cea mai bună este reprezentată de un „mariaj": circuite convenționale pentru a trata secțiunile problemei ce impun viteză și unul sau mai multe microcomputere pentru a manipula restul problemei.

Chiar și acolo unde viteza nu este importantă, logica convențională este adesea utilizată pentru a asambla sistemul și pentru a realiza funcții simple. În următorul capitol discutăm cîteva exemple în acest sens, dar, pentru început, să urmărim mai îndeaproape conceptele de programare.

7.2. PROGRAMAREA

Toate calculatoarele și microcomputerele au la bază conceptul de program. În loc să se proiecteze hard-ul pentru a implementa funcția solicitată de o anumită aplicație, el este conceput pentru a realiza cîteva *operații elementare*, sub conducerea unui *program*.

Programul constă dintr-o secvență de operații elementare, care vor realiza funcția dorită. Hard-ul de bază este în acest caz același pentru toate apli-

cațiile. Se scriu apoi programe pentru aplicația concretă și sînt stocate într-o memorie. Instrucțiunile necesare sînt depuse la locații succesive din memorie și sînt executate în ordine, în mod automat, așa cum impune *numărătorul de program*. Unele instrucțiuni produc *ramificații* în program, prin furnizarea unei noi adrese numărătorului de program. De obicei, aceste ramificații sînt *salturi condiționate*, în sensul că saltul se efectuează sau nu în funcție de rezultatul unei comparații sau al unei operații matematice anterioare.

Aceste ramificații în program (denumite uneori *salturi*) sînt similare ramificațiilor indicate pe diagrama de stări a emițătorului de date (vezi fig. 5-9). De fapt, secvența principală de numărare a respectivului numărător de stări corespunde la una sau mai multe instrucțiuni ale programului ce realizează aceeași funcție. De exemplu, starea READ TAPE (citește banda, vezi fig. 5-9) corespunde secvenței reale de instrucțiuni de citire a datelor de pe bandă pe memorie. Cîteva din instrucțiunile din această secvență compară datele citite de pe bandă cu un cod special EOF (end of file, sfîrșit de înregistrare). Dacă se detectează EOF, o instrucțiune de salt condiționat setează numărătorul de program în porțiunea IDLE (inactiv) a programului. Dacă nu se detectează EOF, programul continuă pînă cînd se depune în memorie ultimul caracter, după care se trece la programul de transmisie, care urmează programului READ TAPE în secvența normală a numărătorului de program.

7.3. BUCLE DE PROGRAM

Ramificațiile la stările anterioare din secvență (program), ca TIMOUT din figura 5-9, sînt denumite *bucle* de program deoarece se repetă o secvență mai mult decît o dată, formînd o buclă. Adesea dorim să parcurgem o buclă de un număr fixat de ori. În buclă vom include instrucțiuni de *incrementare a unui contor și instrucțiuni de testare a rezultatului incrementării, pentru a vedea dacă bucla a fost parcursă de un anumit număr de ori*. Bucla este produsă de o ramificație condiționată bazată pe acest test. Cînd contorul atinge valoarea corectă, nu se mai execută ramificația condiționată și *numărătorul de program continuă* cu următoarea porțiune de program.

Buclele de program pot reduce în mod considerabil numărul de instrucțiuni necesare pentru realizarea unei funcții. De exemplu, un program destinat citirii a 120 de caractere de pe o bandă magnetică și stocării lor în memorie trebuie să execute o funcție INPUT (intrare) și o instrucțiune STORE (memorează) pentru fiecare caracter, Fără bucle, programul ar cere 240 de instrucțiuni, ca mai jos :

```
INPUT ≠ 1
STORE ≠ 1
INPUT ≠ 2
STORE ≠ 2
   .
   .
   .
INPUT ≠ 120
STORE ≠ 120
```

Folosind o buclă de program, putem reduce programul *de la 240 la numai cinci* instrucțiuni, ca în continuare:

INIȚIALIZARE INDEX COUNT LA $n = 1$
BRANCH IF INDEX COUNT < 120
INPUT $\neq n$
STORE $\neq n$
INCREMENT INDEX COUNT $(n + 1)$

Buclele de program sînt atît de importante încît adesea calculatoarele au un registru special, *registrul index*, pentru a manipula mai rapid astfel de operații. De asemenea, multe calculatoare au o instrucțiune separată „increment index and branch if result zero" (incrementează registrul index și execută ramificația dacă rezultatul este zero), care realizează efectul ultimelor două instrucțiuni de mai sus.

7.4. COMPROMISUL PROGRAM – LOGICĂ

Datorită echivalenței dintre programe și logică, trebuie să hotărîm cît de departe putem merge în folosirea variantei programate. Programarea reduce costul sistemului, în dauna vitezei. Cel mai simplu microcalculator poate fi programat să realizeze *orice* prelucrare de date, dacă viteza de execuție nu este importantă. În realitate însă există întotdeauna constrîngeri în ceea ce privește viteza de lucru. Aceste constrîngeri pot fi depășite folosind mai multe circuite logice pentru a executa funcțiile impuse, putem să le realizăm mai repede reducînd ponderea programării.

La dispoziția utilizatorilor se află o gamă largă de procesoare, de la microcalculatoare de un dolar pînă la supercomputere în valoarea de milioane de dolari. În general, costul este legat de numărul și viteza circuitelor folosite. Cu mai mult hardware, putem realiza mai multe în fiecare pas de program. Mai mult hardware ne permite și o codare mai eficientă a instrucțiunilor, reducînd prin urmare efortul de programare și necesitățile de memorie. În general, tehnologiile ce produc o logică de mare viteză sînt mai costisitoare decît cele de mică viteză. Dar chiar cu circuite logice de aceeași viteză există o variație extrem de mare în ceea ce se poate face, într-un timp dat, în funcție de *numărul* porților folosite în structurarea unui calculator.

În principal, viteza de prelucrare este afectată de:

1. Durata ciclului;
2. Dimensiunea cuvîntului;
3. Setul de instrucțiuni și modurile de adresare.

Durata ciclului este timpul necesar pentru execuția unei instrucțiuni de bază. Pe calculatoarele mari ea este deseori egală cu ciclul de memorie, dar pe microcomputerele mici poate fi mult mai lungă. Durata ciclului depinde parțial de tehnologia circuitelor utilizate și, parțial, de cît anume din prelucrare se realizează în paralel. În mod normal instrucțiunile sînt executate ca o secvență de stări ale unui procesor intern mai mic. Mai multe trepte — înseamnă reducerea hard-ului procesorului dar, în același timp, o durată a ciclului mai mare.

Dimensiunea cuvîntului afectează direct viteza de prelucrare în mai multe moduri. Deoarece într-un ciclu se poate schimba doar un cuvînt din me-

memorie, dimensiunea cuvîntului impune numărul de biți care pot fi schimbați într-un ciclu. Cu toate că în acest fel se obține o limită superioară pentru viteza de prelucrare, această limită nu poate fi atinsă întotdeauna în aplicațiile curente.

De exemplu, dacă manipulăm caractere de 8 biți (bytes), folosirea unor cuvinte de 16 biți nu va dubla viteza de prelucrare. Totuși un cuvînt de 16 biți va crește viteza de transfer în mod considerabil, chiar atunci cînd se manipulează octeți, deoarece *execuția oricărei instrucțiuni de 16 biți produce mult mai multe efecte într-un singur ciclu, decît a uneia de 8 biți.* Operațiuni care iau o instrucțiune pe o mașină de 16 biți ocupă două sau trei instrucțiuni pe o mașină de 8 biți.

Setul de instrucțiuni al unui procesor este direct legat de dimensiunea cuvîntului, deoarece cu cît aceasta este mai mare, cu atît mai complexe sînt și operațiile ce pot fi definite într-un singur ciclu. Există, totuși, diferențe importante, printre procesoarele ce au o aceeași mărime a cuvîntului. Multe procesoare de 8 biți au lungimea cuvîntului instrucțiunii de 8, 16 sau 24 de biți, citiți în 1, 2 sau 3 cicluri de memorie. În general pentru a utiliza în mod eficient biții cuvîntului de instrucțiune este necesară mai multă logică. Răsplata este, totuși, triplă : (1) o codare eficientă a biților instrucțiunii înseamnă că se pot realiza mai multe efecte, în fiecare treaptă, cu alte cuvinte, se mărește viteza de prelucrare la aceeași lungime a cuvîntului ; (2) avînd la dispoziție mai multe efecte pe treaptă, pentru aceeași problemă sînt necesare mai puține instrucțiuni ; în acest mod se economisește memoria, compensînd creșterea costului hardware ; (3) la mai puține trepte de programare pentru o funcție, se reduce și efortul de scriere a programului.

7.5. SETURI DE INSTRUCȚIUNI ȘI MODURI DE ADRESARE

Se poate considera că un set de instrucțiuni este caracterizat prin doi parametri importanți : (1) tipul operației și (2) modul de adresare. Un set puternic de instrucțiuni definește multe tipuri de operații și, poate, un fapt mai important, multe moduri de adresare. *Chiar și cel mai slab set de instrucțiuni poate realiza orice operație,* la fel cum putem realiza orice funcție logică doar cu porți NAND. La fel cum pentru a realiza o operație complexă sînt necesare multe porți NAND, tot astfel o secvență lungă de instrucțiuni simple realizează funcția unei instrucțiuni complexe. Un procesor poate realiza multiplicarea, de exemplu, fără a avea o instrucțiune specială de multiplicare prin simpla deplasare și adunare a biților deînmulțitului.

Cele mai multe procesoare au unul sau mai multe registre interne denumite acumulatoare. *Toate operațiile aritmetice și logice sînt realizate de obicei, în acumulator.* De exemplu, instrucțiunea logică AND, ia un operand din memorie sau din alt registru face AND cu conținutul acumulatorului și plasează rezultatul în acumulator. De exemplu :

Operand din memorie sau alt registru :	10101010
Conținutul acumulatorului *înaintea* execuției instrucțiunii :	00001111
Conținutul acumulatorului *după* execuția instrucțiunii :	00001010

De observat, că fiecare bit al operandului este cuplat prin AND cu bitul corespunzător al acumulatorului.

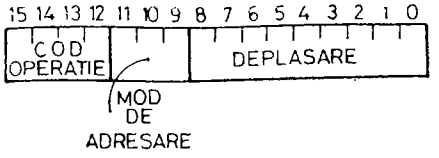


Fig. 7-2.

Un cuvînt de instrucțiune este divizat, în mod normal, în mai multe *cîmpuri*, fiecare cîmp avînd o semnificație distinctă. De exemplu, un cod de instrucțiune de 16 biți ar putea fi definit ca în figura 7-2.

Cîmpul codului operației definește tipul operației ce urmează a fi realizată. De exemplu AND, OR, XOR și ADD AND SUBSTRACT se execută la fel ca funcția AND exemplificată anterior, LOAD plasează pur și simplu operandul în acumulator, STORE depune conținutul acumulatorului la adresa operandului, BRANCH pune adresa operandului în contorul de program, transferînd, prin urmare controlul programului la locația respectivă, BRANCH ON CONDITION transferă controlul la adresa operandului numai dacă *bistabilii de condiție* au fost aduși în starea corespunzătoare de către precedentele operații logice sau matematice.

Adresa operandului (adresa de memorie unde poate fi găsit operandul) este determinată din cîmpul deplasării (*D*) prin diferite reguli, depinzînd de *modul de adresare* indicat de biții 9, 10 și 11. Cîteva moduri tipice sînt următoarele:

1. *Adresarea directă* folosește pur și simplu conținutul cîmpului *D* ca adresă a memoriei. Aceasta înseamnă că putem în acest exemplu, să adresăm direct locațiile 0-512. Aceste locații sînt prin urmare folosite pentru a memora constante importante. Aceasta se mai numește, de asemenea, adresare în *pagina de bază* sau pagina zero.

2. *Adresarea relativă* tratează cîmpul *D* ca un număr cu semn și îl adună la conținutul curent al contorului de program. Putem, adresa în acest mod orice cuvînt dintr-un interval de ± 256 cuvinte în raport cu locația curentă. Dacă programul rulat este relativ mic, aceasta poate fi suficient. Menționăm că referirea la cuvinte folosite de mai multe programe dispersate, se face prin alte moduri, ca adresarea *directă*.

3. În *adresarea indexată* se adună cîmpul *D* la conținutul unui registru, denumit *registru index* și se folosește rezultatul ca adresă a operandului. Aceasta permite adresarea oricărei locații din memorie, și simplifică realizarea unor operații repetate sau *bucle*. Unele instrucțiuni speciale fac posibilă incrementarea și testarea registrului index *și* definirea unei adrese de ramificație, (totul într-o singură instrucțiune).

4. *Adresarea indirectă* se utilizează împreună cu unul dintre celelalte moduri de adresare. *Conținutul* adresei operandului ce a fost definită de un alt mod, este folosit pentru a indica *adresa* operandului. Continutul memoriei poate fi, deci, interpretat ca un *indicator de adresă*. De exemplu, putem executa o altă ramificație (BRANCH) la alt program ce începe la locația 5 000, făcînd o ramificație *indirectă* la locația 100, din pagina-bază, dacă la aceasta se stochează 5000 (adresa „indicată"). Adresarea indirectă este folosită uneori împreună cu indexarea în modul denumit cu *post-indexare*, registrul index determină intrarea în tabela a cărei adresă de start este depusă ca adresa indicatorului. Deoarece adresarea indirectă este combinată cu alte moduri de adresare în formatul instrucțiunii se prevede adesea un *bit separat de adresare indirectă*.

5. *Adresarea imediată* folosește pur și simplu conținutul cîmpului *D* ca operand. De exemplu, dacă dorim să adăugăm „6" la conținutul acumulatorului, se va folosi o instrucțiune ADD, în modul imediat, cu „6" în cîmpul *D*.

Formatele instrucțiunilor pot să ia mai multe forme, care să nu prezinte asemănare cu cel din figura 7-2. Proiectanții unor procesoare diferite realizează compromisuri diferite. De exemplu, am putea defini de două ori mai multe coduri de operații dacă ne-am fixa la un cîmp de deplasare de numai opt biți. De asemenea, deoarece este de dorit să avem mai multe acumulatoare, am putea adăuga un cîmp pentru a desemna un acumulator din patru sau din opt, dacă am sacrifica mărimea cîmpului deplasării. Este de asemenea util să avem mai mult de un registru index, astfel încît am putea defini un cîmp al registrelor index prin sacrificarea altui cîmp. De fapt, există mai multe formate uzuale, fiecare fiind optimizat pentru un tip diferit de instrucțiune.

7.6. UN SET REAL DE INSTRUCȚIUNI

Ca un exemplu de set de instrucțiuni simplu, pe linia tradițională, putem urmări formatele din figura 7-3. Acest set de instrucțiuni a fost utilizat de National Semiconductor pe structura multicip IMP-16 în 1973. Ulterior, a fost reluat în microprocesorul monocip PACE și apoi, pentru microprocesorul mai rapid INS-8900. Procesorul are patru registre acumulator (adresate

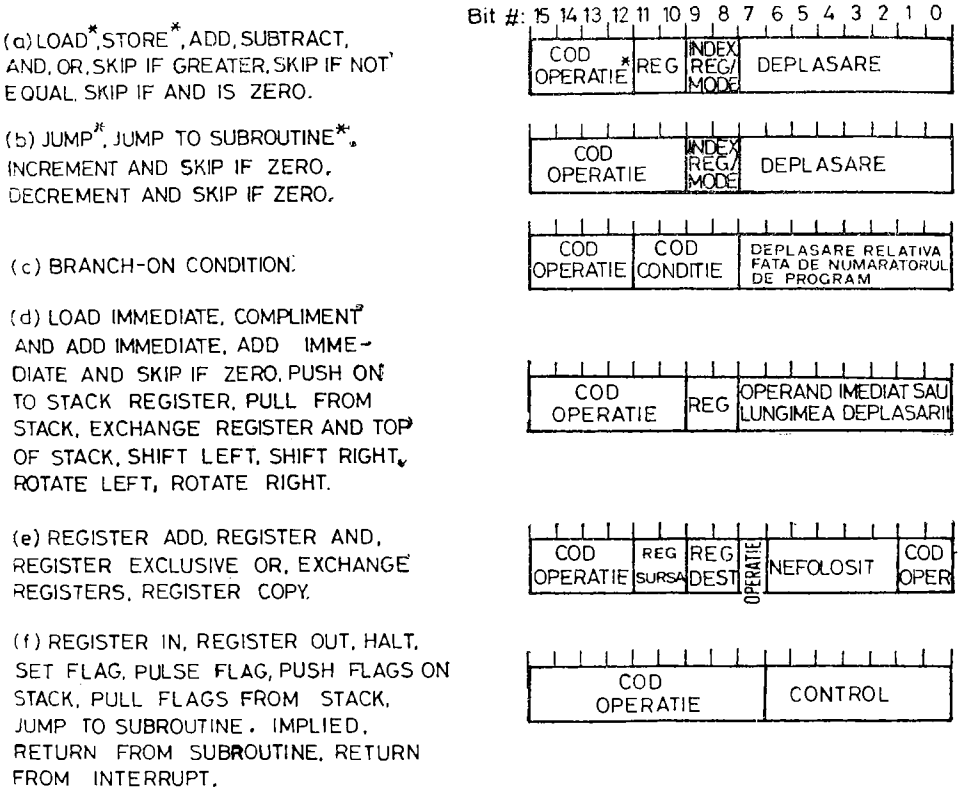


Fig. 7-3. Formatul instrucțiunilor pentru microprocesorul 8900. Asteriscul marchează instrucțiunile pentru care sînt disponibile două moduri de adresare: directă și indirectă.

prin cîmpul REG). Două dintre acestea pot fi folosite și ca registre index. Cîmpul INDEX REG/MODE definește patru moduri de adresare, conform Tabelului 7-1.

Tabelul 7-1. **Index REG/MODE (registru index/mod)**

Cîmpul reg/mod	Modul de adresare	Gama
0 0	Pagina bază (direct)	0-255
0 1	Relativ la contorul de program	PC ± 127
1 0	Relativ la acumulatorul 2	AC2 ± 127
1 1	Relativ la acumulatorul 3	AC 3 ± 127

În felul acesta putem să definim adresele de memorie în modul cel mai potrivit. Prima etapă din execuția instrucțiunii de către procesor constă în calculul *adresei efective* conform cu una din cele 4 reguli date în Tabelul 7-1. Dacă instrucțiunea este de tipul *indirect*, este necesar un ciclu suplimentar de memorie pentru a obține adresa efectivă din locația definită de către instrucțiune. În continuare, execuția reală a instrucțiunii folosește adresa efectivă.

Referindu-ne la figura 7-3 să observăm, pe scurt, instrucțiunile disponibile. LOAD și STORE produc fie *încărcarea* acumulatorului, fie respectiv *stocarea* conținutului său în adresa de memorie definită de adresa efectivă. Dacă instrucțiunea este JUMP, adresa efectivă este transferată contorului de program. ADD sumează conținutul locației de memorie definită de către adresa efectivă la conținutul acumulatorului indicat și plasează rezultatul în acumulatorul folosit. Instrucțiunile SUBSTRACT, AND și OR lucrează în același mod, cu excepția faptului că se realizează o operație diferită.

Instrucțiunea SKIP IF NOT EQUAL (omite instrucțiunea următoare dacă nu avem egalitate) compară conținutul adresei efective de memorie cu cea a registrului specificat. Datele din registru sau din memorie nu sînt modificate, dar dacă pe baza comparației se observă că cele două cuvinte sînt inegale, contorul de program este sumat cu doi în loc de unu. Dacă următoarea instrucțiune a fost, de exemplu, BRANCH, ramificația are loc numai dacă nu a fost indicată omiterea (skip). SKIP este astfel folosită pentru a executa în mod condiționat instrucțiunea ce urmează după ea.

Codul de condiție (stare) folosit de către instrucțiunea BRANCH ON CONDITION, reprezintă starea bistabililor din procesor care indică diferite condiții din sistem : dacă conținutul acumulatorului este *zero* („zero"), *pozitiv* („plus"), *odd* („impar"), sau *par* („even") și dacă în cea mai recentă operație a rezultat un *transport* („carry") sau o *depășire* („overflow"). Mai frecvent, codurile de condiție indică dacă ultima operație aritmetică sau logică executată a generat un rezultat care a fost zero, pozitiv sau a avut transport sau depășire.

Instrucțiunile IMMEDIATE au ultimii opt biți ai cuvîntului instrucțiunii drept operand. De exemplu, un ADD IMMEDIATE adună operandul imediat la unul din cei patru acumulatori și depune rezultatul în el.

Instrucțiunile de tip STACK, indicate în figura 7-3 *d*, sînt utilizate la memorarea temporară a conținutului registrelor pentru a fi utilizate ulterior, într-o așa-numită *stivă (stack)*. Putem să *introducem (push)* date în stivă sau să *extragem (pull, pop)* datele introduse anterior în stivă pentru a le plasa

într-un registru. Adresarea este inutilă, deoarece procesul de introducere/extragere din stivă are loc pe principiul *ultimul introdus/primul extras (last in/first out, LIFO)*. Pentru a întrerupe execuția unui program, salvăm conținutul registrelor în stivă într-o ordine oarecare, executăm celălalt program și apoi *extragem* registrele din stivă în ordine inversă pentru a relua programul inițial. Chiar conținutul numărătorului de program poate fi salvat în acest mod. Instrucțiunea JUMP TO SUBROUTINE (salt la subrutină, figura 7-3 *f*) introduce o nouă adresă în contorul de program (branch) și de asemenea salvează conținutul curent al numărătorului de program în stivă. O instrucțiune RETURN FROM SUBROUTINE (revenire din subrutină) scoate (pull) locația anterioară de program din stivă, adăugîndu-i în mod automat deplasarea. *Subrutinele* sînt folosite pentru funcții care sînt necesare în mai multe părți ale programului principal. În loc de a repeta aceeași secvență de instrucțiuni de fiecare dată, sărim pur și simplu la subrutină (în altă parte a memoriei) și ne întoarcem cînd o terminăm.

Revenind la figura 7.3 *d*, urmărim instrucțiunile SHIFT și ROTATE. În cazul instrucțiunii SHIFT (deplasare) conținutul registrului specificat este mutat la stînga sau la dreapta cu numărul de poziții definit în cîmpul deplasării. Dacă în cadrul instrucțiunii SHIFT cînd depășesc capătul, biții se pierd în cadrul instrucțiunii ROTATE (rotire) ei sînt introduși la celălalt capăt al registrului. Dacă anterior s-a executat o instrucțiune SET FLAG (setare indicator) sau PULSE FLAG (comandă indicatorul), registrul devine un registru de 17 biți, datorită adăugirii bistabilului de *legătură* la capătul cel mai semnificativ.

Instrucțiunea REGISTER ADD (adună registre, figura 7-3 *e*) adună conținutul unuia din cele patru registre acumulatoare (registrul sursă) la conținutul registrului destinație și plasează rezultatul în registrul destinație. *EXCHANGE REGISTERS* schimbă pur și simplu conținutul celor două registre specificate. REGISTER COPY încarcă conținutul registrului sursă în registrul destinație. REGISTER IN și REGISTER OUT sînt utilizate pentru introducerea și extragerea de date de la/spre dispozitive externe.

7.7. BUNICUL MICROPROCESOARELOR

Primul microprocesor ce a constituit un succes a fost introdus de Intel — în 1971. Deși a fost lent, în comparație cu standardul de azi (avînd un ciclu cu o durată de 20 μs), 8008 a fost suficient de rapid și eficace pentru a rezolva o gamă largă de aplicații practice. În 1974, în momentul în care începea să apară concurența, Intel a anunțat o variantă îmbunătățită, 8080 (vezi fig. 7-4) care a contribuit la continuarea dominației ce o exercita asupra pieții, într-un stil apropiat de cel al firmei IBM. Deși setul de instrucțiuni ale acestor produse este mai puțin direct decît al multor componente moderne similare, ele s-au arătat eficace. Dominația pe care o exercită asupra industriei le face imposibil de ignorat.

Deși 8008 este astăzi depășit, setul său de instrucțiuni a constituit baza pentru 8080 și 8085, apărute ulterior. Vom începe prin descrierea lui 8008 și vom dezvolta expunerea pe această bază.

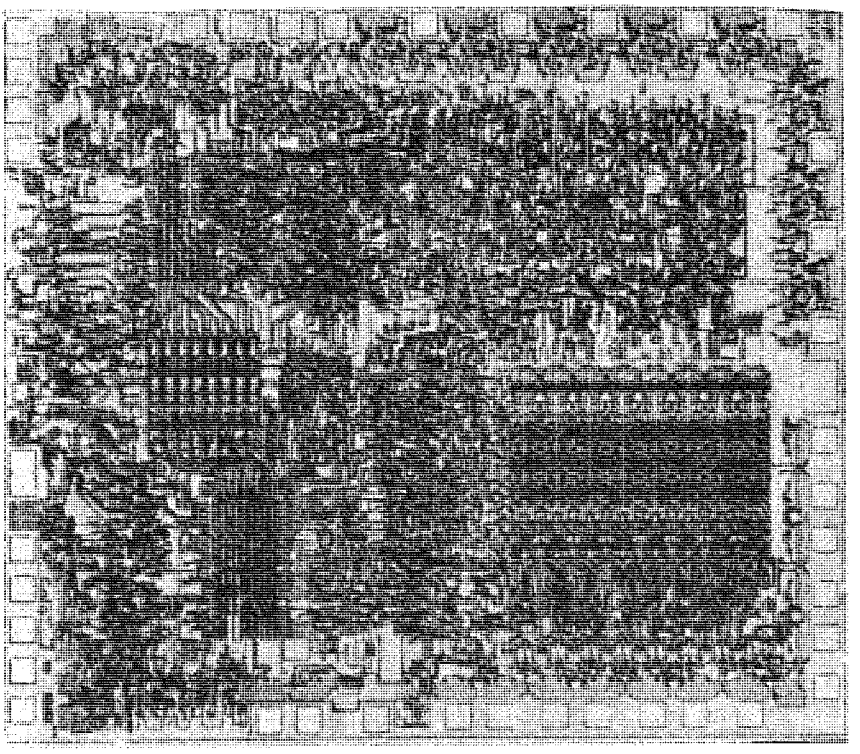


Fig. 7-4. Microfotografia microprocesorului 8080 (dimensiunea reală 3,2 mm×4,4 mm).

8008 are opt registre de 8 biți. Unul dintre acestea (*M*) este rezervat pentru operații cu memoria. Ori de cîte ori este adresat registrul *M*, datele vin dinspre sau merg spre locația de memorie adresată de alte două registre (*H* și *L*). Există, prin urmare, doar cinci registre disponibile pentru uz general denumite *A*, *B*, *C*, *D* și *E*. Așa cum o indică formatul din figura 7-5 *a*, conținutul oricărui registru poate fi *mutat* (move) în oricare alt registru prin instrucțiuni de un byte. Specificînd registrul *M* ca registru sursă sau destinație, putem executa operații de LOAD și STORE între *memorie* și oricare din celelalte registre. Totuși, înainte de a face aceasta este necesar să punem adresa lor în registrele *H* și *L*.

Aceste registre formează împreună *indicatorul* de adresă de memorie, avînd biții de adresă superiori în *H* iar pe cei inferiori în *L*. Putem realiza o *adresare similară celei directe* dacă folosim instrucțiunea de 2 bytes MOVE IMMEDIATE (vezi fig. 7-5 *d*) pentru a încărca adresa dorită în registrele *H* și *L*, înaintea executării instrucțiunii de referire la memorie. Dacă adresăm octeți consecutivi, de memorie, putem incrementa *L* cu instrucțiunea mono-byte INCREMENT. Aceasta este *similar* unei *adresări indexate*. De asemenea, putem realiza o *adresare similară celei indirecte* prin încărcarea registrului *L* din memorie înaintea executării unei instrucțiuni cu referire la memorie.

Există de asemenea facilități de *adresare directă adevărată*, datorită instrucțiunii JUMP (salt în program, similară BRANCH), ca și de chemare a

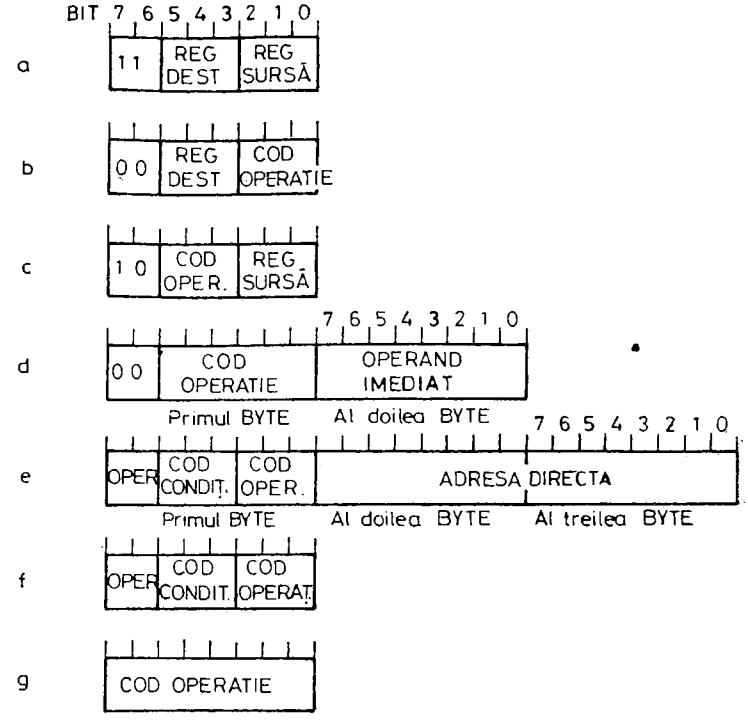


Fig. 7-5. Formatul instrucțiunilor microprocesorului Intel 8008: (a) MOVE REGISTER* TO REGISTER* (transfer registru-registru). *(b)* INCREMENT OR DECREMENT (increment sau decrement). (c) REGISTER* TO ACCUM : ADD**, SUBTRACT**, AND, EOR, OR, COMPARE (registru în acumulator : adunare, scădere, SI, SAU-EXOLUSIVO, SAU, compară) *(d)* IMMEDIATE : MOVE, ADD*, SUBSTRACT**, AND, EOR, OR, COMPARE (imediat : transfer, adunare, scădere, SI, SAU-EXCLUSIV, SAU, compară). *(e)* JUMP, (salt), CALL SUBROUTINE (apel subrutină). *(f)* RETURN FROM SUBROUTINE (revenire din subrutină). *(g)* ROTATE A RIGHT** (rotire A la dreapta), ROTATE A LEFT** (rotire A la stînga), RESTART (restart), HALT (stop), INPUT (transfer spre procesor), OUTPUT (transfer de la procesor).

* *Nota 1.* Pentru uz curent sînt disponibile numai primele cinci registre (A, B, C, D, E), deoarece atunci cînd este adresat al optulea registru (M) se inițiază o scriere sau o citire a memoriei la adresa specificată de conținutul registrelor H, L (al șaselea și al șaptelea).

** *Nota 2.* Cu sau fără CARRY (transport).

subrutinei (CALL SUBROUTINE), vezi figura 7-5*e*. Aceste instrucțiuni ocupă trei bytes succesivi în memorie și pot adresa direct pînă la 16.384 bytes de memorie. Execuția oricăreia din aceste instrucțiuni produce încărcarea contorului de program cu adresa directă specificată, dacă se îndeplinesc condițiile de salt. Instrucțiunea CALL SUBROUTINE introduce în stivă conținutul curent al contorului de program. Dacă condițiile specificate sînt îndeplinite, o instrucțiune RETURN readuce vîrful stivei (pop) în contorul de program, refăcîndu-se secvența programului principal. Instrucțiunile JUMP, CALL și RETURN pot fi fie necondiționate sau pot avea un cîmp al

codului de condiție care impune condițiile ce conduc la ramificații în program, așa cum se arată în Tabelul 7-2. Condițiile listate se referă la rezultatele celor mai recente operații aritmetice sau logice (mai puțin MOVE).

Tabelul 7-2. **Coduri de condiție**

Codul de condiție	Condiția cerută
000	No carry (fără transport)
001	Not zero (diferit de zero)
010	Positive (pozitiv)
011	Parity odd (imparitate)
100	Carry (transport)
101	Zero (zero)
110	Minus (negativ)
111	Parity even (paritate)

Toate operațiile logice și artimetice folosesc registrul *A* ca destinație. Putem, de exemplu, să sumăm (ADD), un operand imediat (fig. 7-5 *d*) sau conținutul oricărui registru (fig. 7.5 *c*), cu conținutul registrului *A*, rezultatul apărînd în *A*. Dacă realizăm ADD din registrul M, adunăm de fapt conținutul locației de memorie „indicată" de registrele *H* și *L* la registrul A. Desigur, celelalte operații, SUBSTRACT, AND, EXCLUSIVE OR și OR, lucrează similar. La fel COMPARE, folosită însă numai pentru a seta biții de condiție. În acest caz, se realizează o scădere, codul de condiție este inițializat în concordanță cu rezultatul, dar registrul A nu este modificat.

Instrucțiunile ROTATE (vezi fig. 7-5 *g*) realizează o deplasare stînga sau dreapta de un singur bit, incluzînd sau nu conținutul bistabilului *carry* ca al nouălea bit. Datele ce ies din registru pe la o extremitate sînt introduse în cealaltă, pentru a se preveni pierderea lor. Pentru o deplasare de mai mult de o poziție instrucțiunea trebuie repetată.

Instrucțiunea INPUT introduce date din dispozitive externe în registrul *A* prin pinii de intrare folosiți și pentru date din memorie. Trei biți ai instrucțiunii definesc pînă la 8 dispozitive (porturi) diferite de intrare. Instrucțiunea OUTPUT transferă date spre unul din 24 de dispozitive externe (porturi de ieșire-output ports). Patru biți ai instrucțiunii indică numărul de ordine al dispozitivului.

Pe lîngă toate instrucțiunile lui 8008, anterior descrise, 8080 și 8085 au toate *instrucțiunile suplimentare* prezentate în figura 7-6. Dacă 8008 este dotat cu adresare directă numai pentru instrucțiunile JUMP, 8080 are, de asemenea, adresare directă pentru LOAD A, STORE A, LOAD HL și STORE HL. În felul acesta putem încărca sau stoca un octet, o pereche de octeți, din oricare din cele 65.536 locații de memorie cu o singură instrucțiune formată din trei octeți.

Multe din instrucțiunile de 16 biți sînt adăugate pentru a permite manipularea comodă a cuvintelor și *adreselor* de 16 biți. De exemplu, instrucțiunea LOAD HL (vezi fig. 7.6*a*) încarcă în registrul *L* conținutul locației de memorie direct adresate, iar în registrul *H* conținutul următoarei locații de memorie. În felul acesta, *registrele H și L sînt efectiv tratate împreună, ca un registru de 16 biți*, încărcarea lor avînd loc dintr-o locație de memorie de 16 biți.

Așa cum se indică în figura 7-6 *b*, putem, de asemenea, încărca în perechile de registre BC, DE, HL, sau în indicatorul de stivă (stack pointer — SP),

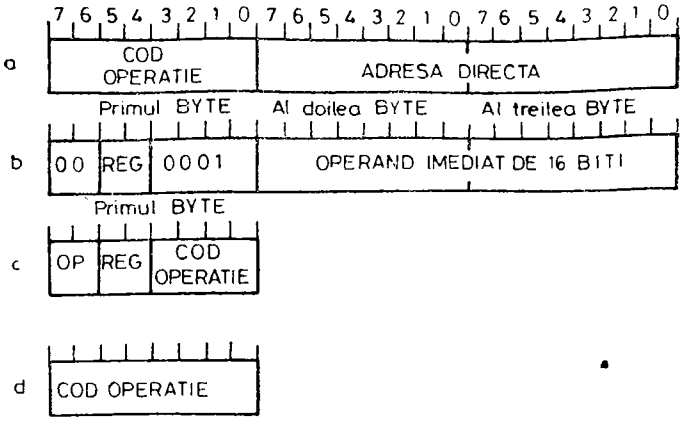


Fig. 7-6. Instrucțiuni adiționale ale microprocesorului 8080. *(a)* LOAD A din memorie, LOAD HL din memorie, STORE A în memorie, STORE A în memorie. *(b)* LOAD EXTENDED IMMED (registrele BC, DE, HL sau SP). *(c)* PUSH (BC, DE, HL sau AF), POP (BC, DE, HL, sau AF), ADD TO HL (BC, DE, HL, sau SP), STORE A (adresa de memorie se află în BC sau DE), LOAD A (adresa de memorie se află în BC sau DE), INCREMENT (BC, DE, HL, SP), DECREMENT (BC, DE, HL, SP). *(d)* EXCHANGE HL DE, EXCHANGE HL STACK, LOAD SP WITH HL, LOAD PC WITH HL, COMPLEMENT A, SET CARRY, COMPLEMENT CARRY, DECIMAL ADJUST A, ENABLE INTERRUPT, DISABLE INTERRUPT, NO OPERATION.
Notă: BC=registrele B și C compuse, formînd un registru de 16 biți (la fel pentru DE, HL și AF); SP=registru indicator de stivă; PC=numărător de program; F=registre de indicatori (flag-uri) (carry, sgn, etc.). Microprocesorul 8080 execută, de asemenea, toate instrucțiunile prezentate în figura 7-5.

operanzi imediați, de 16 biți. Definirea acestor operanzi imediați are loc în al doilea și al treilea octet al unei instrucțiuni de trei octeți. Celor patru perechi de registre li se pot aplica operații de salvare (PUSH) sau revenire (POP) din stivă. Instrucțiunile ADD extinse ne permit adunarea la conținutul lui *HL* a oricărui din cele 4 registre pereche. Acest fapt este desigur foarte util în calculul adreselor de memorie. Instrucțiunile INCREMENT și DECREMENT sînt extinse pentru a permite o incrementare sau o decrementare de 16 biți într-o singură instrucțiune. Instrucțiunile LOAD A și STORE A se pot executa și folosind registrele *B* și *C* sau *D* și *E* ca indicator de stivă, în locul lui *H* și *L*. Aceasta este echivalent cu a avea la dispoziție trei registre index separate, care pot fi folosite pentru a adresa trei liste diferite de operanzi. Putem incrementa fiecare index cu o instrucțiune de un singur byte; putem încărca fiecare operand cu o instrucțiune de un singur byte.

Așa cum se arată în figura 7-6*d*, putem calcula adrese de salt în *HL*, apoi putem executa saltul folosind LOAD PC WITH HL (încarcă PC cu HL). S-au adăugat instrucțiuni pentru lucrul cu indicatorul de stivă (SP), bistabilul de transport și bistabilul de activare a întreruperilor.

Stiva microprocesorului 8080 este realizată în memoria principală (în RAM), utilizînd conținutul indicatorului de stivă (SP) ca adresă de memorie,

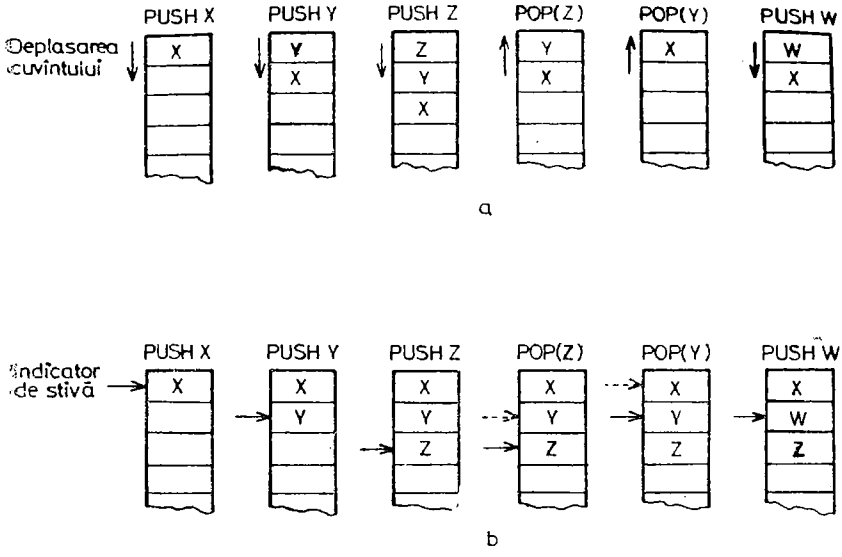


Fig. 7-7. Stive. *(a)* Stivă realizată prin registre de deplasare reversibile (ex.: microprocesorul 8080). (b) Stivă realizată cu registru indicator de stivă (stack pointer) în microprocesorul 8080. Săgeţile pline indică starea indicatorului de stivă; săgeţile întrerupte indică starea registrului indicator de stivă după executarea instrucţiunii.

pentru instrucţiunile de PUSH şi POP (vezi fig. 7-7 *b*). O instrucţiune PUSH decrementează SP cu doi şi apoi stochează datele în locaţia de memorie specificată de către registrul SP. O instrucţiune POP încarcă 16 biţi de date din locaţiile de memorie indicate de către SP, apoi incrementează SP cu doi.

Astfel, numărul de cuvinte care poate fi introdus în stivă este limitat numai de numărul de locaţii pe care dorim să-l rezervăm în RAM pentru constituirea stivei. În cazul lui 8008, stivele există în hardware, ca 8 registre de deplasare reversibile (vezi fig. 7-7 *a*). Cînd se introduce în stivă o adresă (prin CALL SUBROUTINE), ea este pur şi simplu trecută în registrul de deplasare. În momentul în care este extrasă, este deplasată în sens invers. Cu acest sistem, dimensiunea stivei este limitată numai de lungimea registrelor de deplasare disponibile (şapte cuvinte pentru 8008).

7.8. AVANTAJELE LOGICII PROGRAMATE

Urmărind formatele instrucţiunilor din figurile 7-3, 7-5 şi 7-6, observăm că ele codează eficient operaţiile ce trebuie realizate. Fiecare clasă de instrucţiuni are un format potrivit pentru definirea sa eficientă. Desigur, procesorul include o logică suplimentară, pentru a interpreta biţii instrucţiunii în funcţie de biţii codului operaţiei. Compensaţia rezidă în faptul că este necesară mai puţină memorie pentru stocarea programului. Acesta reprezintă acelaşi compromis ca acela întîlnit în Secţiunea 4-8, unde am găsit că prin adăugarea unei logici pentru codarea intrărilor în ROM, putem reduce considerabil mărimea memoriei ROM. Conceptul de cuvînt de instrucţiune conduce la o utilizare şi mai eficientă a ROM deoarece, în loc să încercăm să economisim

biți în ROM prin funcții de codare, ce rezultă dintr-o abordare bazată pe logică conbinațională, *pornim dintr-un început* să definim problema astfel încît memoria ROM să fie utilizată la maximum în cadrul abordării programate.

Desigur, nu este absolut necesar ca programele să fie depozitate în ROM. De fapt, procesoarele sînt aproape întotdeauna proiectate astfel încît *la bus-ul memoriei să poată fi cuplat orice tip de memorie, cu orice viteză.* În felul acesta, putem amesteca blocuri de dispozitive ROM, PROM și RAM, avînd viteze diferite. În cursul verificării programului este mai convenabil să-l avem în RAM, pentru a putea fi modificat ușor. Ulterior, dacă se dorește, porțiunile de memorie ocupate cu program pot fi trecute pe PROM. Dacă sistemul realizat urmează să fie produs în serie mare, programul poate fi eventual pus într-o memorie ROM programată prin mascare ; aceasta după ce a fost complet testat. În felul acesta, devine posibil să facem modificări rapide, pe teren, pentru a adăuga noi funcții prin simpla introducere a unor ROM noi, ce conțin noile facilități de program. Astfel de schimbări majore sînt imposibile în cazul logicii cablate convenționale.

7.9. ORGANIGRAMELE : SCHEMELE BLOC ALE PROGRAMATORULUI

În principiu, procedurile de proiectare a sistemelor logice programate și cablate sînt identice. *Schema bloc* folosită în proiectarea logicii cablate are un corespondent direct în programare, denumit *organigramă*. Ambele pot defini logica sistemului cu orice grad de detaliere și ambele ne permit reprezentarea sistemului și divizarea sa în reprezentări tot mai detaliate (vezi fig. 2-6) pînă cînd se atinge nivelul „componentei". *În cazul programelor, „componentele" sînt instrucțiunile.* Trebuie să ne cunoaștem bine „componentele" (setul de instrucțiuni), pentru a putea lua decizii corecte la toate nivelele proiectării. În mod aproximativ, instrucțiunile corespund ca nivel de complexitate logicii MSI.

Tot astfel cum într-o *schemă bloc* unim blocurile cu linii și săgeți ce indică *fluxul semnalului între componente*, într-o *organigramă* liniile specifică *evoluția contorului de program* între instrucțiuni. La nivelul sistemului general, printr-un dreptunghi se reprezintă multe instrucțiuni, tot astfel cum în schema bloc generală un bloc reprezintă multe circuite integrate. În cazul unui sistem mare, la nivelul cel mai general, fiecare dreptunghi poate reprezenta un program complet, ce urmează să fie scris de o singură persoană. De asemenea, programele sînt scrise adesea pe *„module"* cu intrări și ieșiri bine definite. Putem asambla ușor programe specializate, prin utilizarea a diferite combinații de module generale de program tot astfel cum sistemele specializate realizate cu logică cablată sînt asamblate prin conectarea unor module de uz general.

Diagrama de stări a emițătorului de date, prezentată în figura 5-13 este aproape identică cu o organigramă la nivelul sistemului general. Figura 7-8 o prezintă redesenată sub formă de organigramă. Toate punctele *importante* de decizie (ramificații condiționate) sînt prezentate ca romburi. Deoarece această organigramă corespunde nivelului ansamblului sistemului, dreptunghiurile și romburile reprezintă programe întregi. Deși *rutinele* (progra mele mici) reprezentate prin dreptunghiuri și romburi, pot conține multe rami-

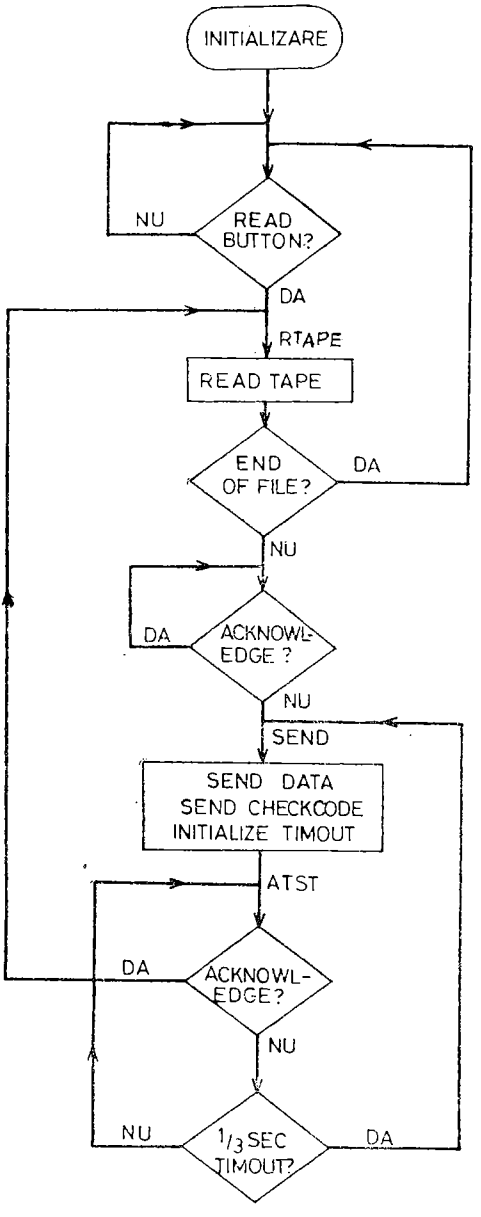


Fig. 7-8. Organigrama pentru transmițătorul de date (vezi figura 5-13).

ficații condiționate, acestea reprezintă decizii interne și nu trebuie să apară pe organigramă la acest nivel. Totuși ele vor apare pe organigramele detaliate pentru rutinele individuale.

Remarcați că în program există mai multe *bucle*, ce sînt indicate prin săgețile ce se închid spre porțiuni anterioare de program. De exemplu, după temporizarea de 1/3 secunde, TIMOUT, programul se ramifică spre înapoi, către începutul rutinei SEND DATA (emisie date). De asemenea, după ce se primește semnalul ACKNOWLEDGE, programul se ramifică către începutul rutinei READ TAPE (citește banda). În momentul în care programul este elaborat în varianta *finală fiecare săgeată reprezintă de fapt o stare a numărătorului de program* (adresa de memorie a primei instrucțiuni din rutina către care este îndreptată săgeata).

În general, contorul de program avansează prin stări consecutive de la începutul pînă la sfîrșitul organigramei, excepțiile fiind constituite numai de ramificații. Putem eticheta săgețile, pentru o identificare similară celei folosite în schemele bloc în proiectarea logicii cablate. Putem introduce denumiri mnemotehnice descriptive, pentru a sugera funcția instrucțiunilor ce încep la locația respectivă, tot astfel cum pe o diagramă bloc denumirile semnalelor sugerează funcțiilor lor. De exemplu, pe figura 7-8, denumim prima locație a rutinei READ TAPE, RTAPE. Ambele săgeți ce se îndreaptă spre dreptunghiul respectiv din organigramă reprezintă trecerea contorului de progam în starea respectivă. Una din săgeți reprezintă ramificația de program spre locația denumită RTAPE, ce are loc cînd se detectează semnalul ACKNOWLEDGE; o alta reprezintă instrucțiunea ce urmează în secvență (după ce testul READ BUTTON dă rezultat fals).

7.10. ECHIVALENȚA PROGRAMELOR CU LOGICA CABLATĂ

Există o echivalență directă între structura unei organigrame și funcțiile logice booleene. De exemplu, figurile 7-9 *a* și *b*, prezintă modul în care apar structurile AND și OR pe o organigramă. Ca și porțile, aceste structuri pot fi combinate pentru a produce orice funcție. Să observăm că funcțiile f_1 și f_2 sînt *generate la momente diferite* folosind *aceeași logică din procesor. Aceasta este cheia economiilor ce se realizează în materie de logică în varianta programată* și în același timp motivul principal pentru care ea este mai lentă. În loc să generăm f_1 și f_2 folosind simultan două porți, utilizăm aceleași porți și „memorăm" rezultatul cu o ramificație condiționată de program.

În figura 7-10 se prezintă, drept exemplu, o organigramă detaliată pentru generarea funcției logice :

$$f = A \cdot B' \cdot C' \cdot D' \cdot E \cdot G' \cdot H + A' \cdot B \cdot C \cdot D' \cdot E \cdot F \cdot G' \cdot H' \qquad (7.1)$$

unde A, B, C, D, E, F, G și H sînt biții registrului A din microprocesorul 8008 sau 8080. Mai întîi executăm o instrucțiune COMPARE IMMEDIATE 10001101 (vezi fig. 7-5 *d*). Dacă primul termen al ecuației este 1, conținutul registrului A va fi 10001101, astfel încît instrucțiunea COMPARE va inițializa bistabilii de condiție pentru a memora un rezultat nul. În continuare se va face un JUMP condiționat (salt dacă „zero" este setat), care va transfera controlul programului, dacă primul termen este „1", către programul care efectuează ceea ce dorim să aibă loc dacă funcția f este adevărată. Procedăm apoi la fel pentru al doilea termen. Dacă nici un termen nu este „1", contorul de program continuă cu ceea ce trebuie realizat dacă funcția f este „0".

Să remarcăm că aceasta este organigrama cu detalierea maximă, în sensul că fiecare treaptă reprezintă de fapt o instrucțiune. În mod normal, nu este necesar să detaliem atît de mult organigrama ; ne vom opri la ceva similar cu figura 7-9 *b* unde $A = 10001101$? pentru prima decizie și $A = 01101100$? pentru cea de a doua. După cum doar rareori realizăm schemele bloc la nivel de poartă, tot astfel, cele mai detaliate organigrame conțin mai multe instrucțiuni pentru fiecare dreptunghi sau romb decizie).

Pentru a defini logica corespunzătoare acestei funcții sînt necesari 10 octeți în ROM, deoarece instrucțiunea COMPARE IMMEDIATE (vezi fig. 7-5 *d*) necesită doi octeți, iar instrucțiunea JUMP (vezi fig. 7-5 *d*) trei. Implementăm astfel funcția cu 10 bytes, sau $10 \times 8 = 80$ biți în ROM. Pentru realizarea cu logică cablată ar fi necesare 2,25 circuite după cum se arată în figura 7-10 *b*. Cu alte cuvinte, *un circuit integrat conținînd porțile necesare este înlocuit de 80/2,25 = 36 biți în ROM*. Experiența demonstrează că aceasta este o valoare *medie* care corespunde multor situații în care logica cablată este înlocuită cu logică programată. Avînd în vedere că *prețul de cost al unui singur circuit integrat montaj pe cablaj este echivalent cu prețul a mii de biți în ROM*, economia care rezultă prin această abordare este fantastică. De exemplu, cu un singur cip de ROM cu 65.536 biți, se pot înlocui $65.536/36 = 1\,820$ de *circuite integrate cu porți*.

Ca alt exemplu de echivalare directă prin logică programată, putem să realizăm decodorul pentru afișajul cu 7 segmente din figura 3-2 ca o *tabelă programată*. Se generează, mai întîi o tabelă de 10 cuvinte în memorie, care arată exact ca tabela de adevăr din figura 3-2, *d*. Putem localiza această tabelă oriunde în memorie, dar să presupunem că am plasat-o în cele 10 locații

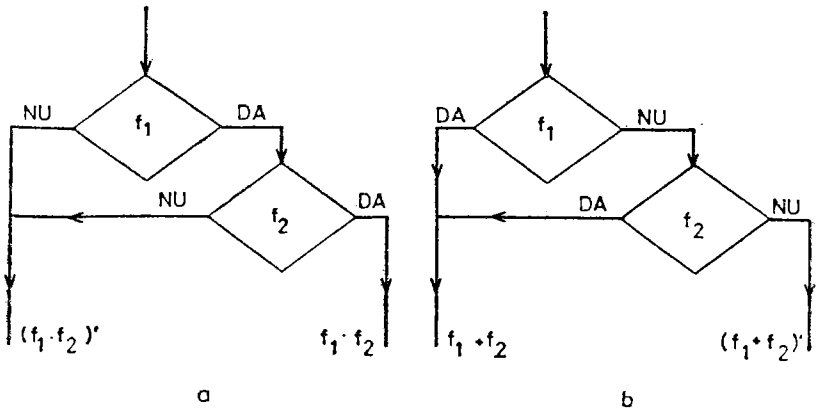


Fig. 7-9. Organigrama unei structuri Booleene : *(a)* structură de tip AND ; *(b)* structură de tip OR.

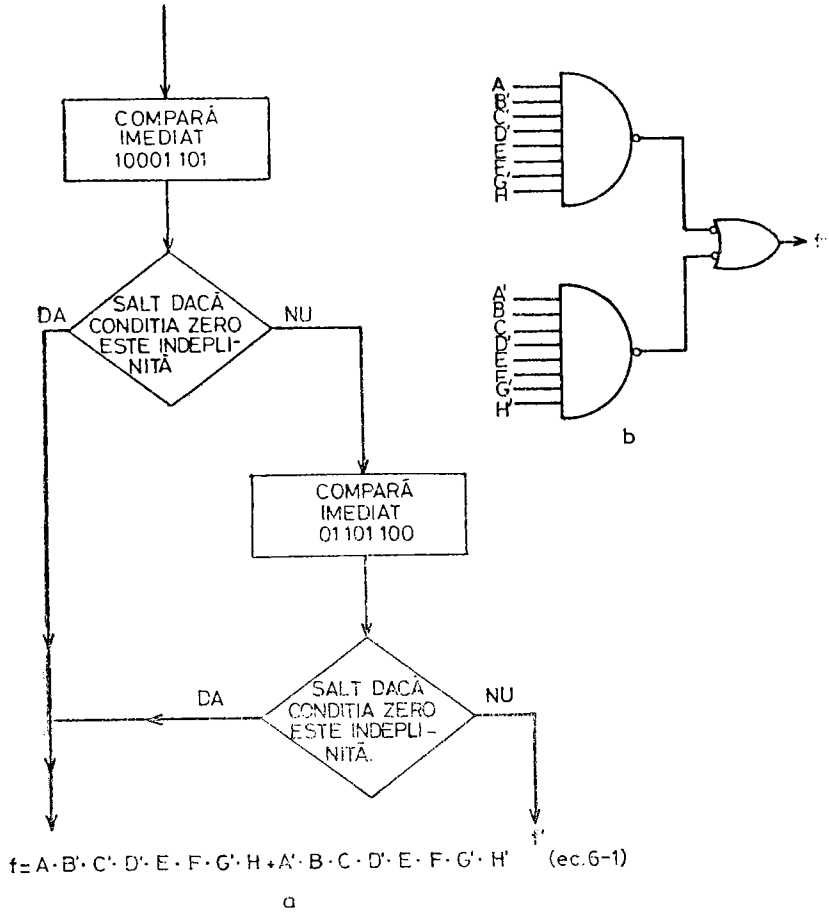


Fig. 7-10. Exemplu de logică programată : *(a)* programarea unui microprocesor (Registrul A conține A, B, C, D, E, F, G, H) ; *(b)* circuit logic ce realizează aceiași funcție (NAND).

ce încep cu 11110000 (FO în **hexadecimal**), vezi tabelul 7.3.

Tabelul 7-3. **Tabelul memorat pentru afişajul cu şapte segmente**

Locaţie de memorie	Conţinut
11110000	01111110
11110001	00110000
11110010	01101101
11110011	01111001
11110100	00110011
11110101	01011011
11110110	00011111
11110111	01110000
11111000	01111111
11111001	01110011

Presupunînd că registrul A (al unui microprocesor 8008 sau 8080) conţine codul unei cifre (ce trebuie să fie afişată), în loc să îl trecem unui decodor extern, putem să îl convertim în cod 7 segmente prin execuţia următorului program (presupunem că H conţine jumătatea superioară a adresei corecte) :

```
IMMEDIATE OR TO A  11110000
MOVE TO L FROM A
MOVE TO A FROM M
```

În felul acesta, convertim codul cifrei într-o adresă de tabelă, plasăm adresa în registrul L, apoi încărcăm din tabela de memorie (locaţia H, L) în registrul A codul corespunzător în 7 segmente. În consecinţă, registrul A va conţine în biţii $A_6 \ldots A_0$ codul de 7 segmente care poate fi transmis direct circuitelor de comandă a afişajului.

Presupunînd, de exemplu că A conţine 0011 (3), valoarea sa va fi 11110011 după IMMEDIATE OR. Plasînd această valoare în registrul L şi încărcînd conţinutul locaţiei respective în A, obţinem în A valoarea 01111001. Cei şapte biţi din dreapta ai acestui cuvînt reprezintă starea celor 7 segmente de afişaj „*a*...*f*".

Necesarul nostru de memorie pentru decodorul în şapte segmente este de 10 octeţi pentru tabelă şi 4 octeţi pentru program, adică un total de $14 \times 8 = 112$ biţi (să remarcăm că programul ar avea numai doi octeţi dacă tabela ar începe la locaţia 0, deoarece s-ar putea omite prima instrucţiune). Am găsit anterior (fig. 3-8) că decodorul în şapte segmente ar lua 5,25 cipuri pentru implementarea cu porţi discrete. În acest caz, raportul biţi-cipuri este $112/5{,}25 = 21{,}4$ *biţi/cip*, ceea ce este chiar mai avantajos decît valoarea precedentă de 36. Desigur, putem cumpăra un cip MSI care realizează aceeaşi funcţie şi include şi circuitele de comandă. Deşi în acest caz este posibil ca varianta MSI să reprezinte o alegere mai bună, ceea ce trebuie înţeles este faptul că programul poate fi un substitut mai economic pentru orice logică. Socotind numărul de capsule folosite în cazul trecerii programului pe un ROM de 65 kbiţi rezultă echivalentul a $112/65.536 = 1/585$ dintr-o capsulă !

Desigur, sistemele reale pot să solicite în continuare logică cablată, circuite integrate, pentru domeniile mai rapide, iar circuitele de interfaţă trebuie oricum să fie realizate cu circuite integrate discrete. Însă pentru logica de viteză moderată şi complexitate ridicată, varianta programată este imbatabilă.

În Capitolul 4 am discutat folosirea ROM pentru generarea funcţiilor booleene. Dacă am implementa ecuaţia 7-1 în acest mod, ar fi fost necesar un ROM de $1 \text{ bit} \times 2^8 = 256$ cuvinte. Am fi folosit astfel 256 de biţi pentru a putea realiza ceea ce se poate face şi cu numai 80 de biţi. Totuşi mai important este faptul că cei 80 de biţi pot fi *o parte redusă dintr-un ROM de orice dimensiune*, în timp ce cei 256 de biţi solicită o anumită structură particulară a memoriei fixe. Lecţia cu care rămînem este aceea că varianta programată conduce la utilizarea optimală a memoriei de tip ROM pentru logică. În mod

normal, *nu folosim niciodată ecuații logice în proiectarea programelor.* Funcțiile sînt definite în mod natural utilizînd organigrame ale funcțiilor dorite și scriind apoi o secvență de instrucțiuni pentru implementarea organigramei.

7.11. COMPARAREA SETURILOR DE INSTRUCȚIUNI

Pentru a ilustra în ce măsură setul de instrucțiuni afectează puterea de calcul, necesarul de memorie și ușurința programării, să luăm o singură instrucțiune a lui 8 900 și să realizăm aceeași funcție cu 8008 și cu 8080. Pentru a realiza o competiție corectă, să presupunem că lucrăm cu un operand de 8 biți. Desigur că microprocesorul 8900 ar fi mult avantajat dacă operandul ar avea 16 biți. Instrucțiunea ADD a lui 8900 adună conținutul oricărei locații de memorie la oricare din cele 4 acumulatoare și depune rezultatul în respectivul acumulator. Să presupunem că valoarea din AC0 (primul acumulator) trebuie păstrată pentru uz ulterior ; ca urmare pentru funcția ADD vom folosi acumulatorul AC1 al microprocesorului 8900.

Pentru a realiza același lucru cu 8008, trebuie mai întîi să salvăm conținutul registrului A și apoi să încărcăm *A* cu conținutul registrului B. Aceasta este necesar deoarece operațiile aritmetice se pot realiza numai în registrul *A*. Apoi inițializăm adresa (*H* și *L*) și facem ADD. Ne sînt necesare următoarele instrucțiuni :

```
MOVE C FROM A          (salvează A)
MOVE A FROM B          (pune operandul în A)
MOVE H IMMEDIATE       (jumătate de adresă)
MOVE I IMMEDIATE       (jumătate de adresă)
ADD MEMORY TO A
```

Deoarece instrucțiunile trei și patru necesită fiecare cîte doi octeți (formatul din figura 7-5*d*) vom avea nevoie la 8008 de 7 octeți pentru a duplica cei doi octeți necesari la 8900 pentru a stoca instrucțiunea ADD. Deoarece memoria de program și efortul de scriere a programului înseamnă bani, *utilizarea unui procesor mult mai scump poate conduce la un sistem care pe ansamblu să fie mai ieftin. De regulă constituie o falsă economie folosirea unui procesor cu un set anemic de instrucțiuni, în cazul în care costul memoriei este de cîteva ori mai mare decît cel al procesorului.*

Să vedem în continuare cum realizăm același lucru cu setul de instrucțiuni mai bogat al microprocesoarelor 8080 și 8085. În acest caz *A* poate fi încărcat direct din memorie și, apoi, adunat cu *B* :

```
MOVE C FROM A          (salvează A)
LOAD A                 (adresat direct din memorie)
ADD B TO A
```

Programul nostru este acum mai simplu, dar, deoarece a doua instrucțiune cere trei octeți (Formatul din figură 7-5 *d*), ocupă cinci octeți în memorie în raport cu cei doi necesari în varianta cu 8900.

Desigur, că lucrurile au fost puțin „aranjate" începînd cu o instrucțiune a lui 8900 (deși am fi putut să forțăm în acest sens utilizînd un operand de 16 biți). În realitate, există situații în care formatul de lungime variabilă al

lui 8008 și 8080 este mult mai eficient. De exemplu, adunarea registrului *B* la registrul *A* poate fi realizată în cazul lui 8008 și al lui 8080 cu o instrucțiune de un singur octet, însă necesită o instrucțiune de doi octeți cu 8900. De asemenea, apar frecvent adresări de locații consecutive din memorie, astfel încît putem folosi numai incrementarea registrului *L*, folosit ca indicator, pentru a obține adresa viitorului operand. Aceasta se realizează cu o instrucțiune de un octet în cazul lui 8080, dar solicită doi octeți în cazul lui 8900. Alte procesoare, cum sînt cele din seria 9900 produsă de Texas Instruments, pot să adune două numere din memorie indexate separat și să incrementeze ambii indicatori, într-o singură instrucțiune.

Deși seturile puternice de instrucțiuni conduc la o economie considerabilă de memorie în sistemele mari, este dificil să se optimizeze instrucțiunile atît pentru sisteme mari cît și pentru sisteme mici. Se observă uneori că flexibilitatea instrucțiunilor, ce îmbunătățește eficiența în sistemele mari produce risipă de biți în cele mici. Acest fapt este recunoscut prin orientarea unor produse mari recente: Intel a optimizat un set de instrucțiuni pentru sisteme mici, introducînd microcalculatorul 8048/9 și un set de instrucțiuni pentru sisteme foarte mari (microprocesorul 8086). Nici unul din aceste seturi nu este direct compatibil cu cel al lui 8080, dar amîndouă sînt suficient de apropiate de acesta pentru a putea fi ușor învățate de utilizatorii lui 8080.

O parte surprinzător de mare din programele reale este ocupată cu calculul, salvarea și inițializarea indicatorilor de adresă. Din acest motiv, facilitățile aritmetice în 16 biți sînt foarte importante chiar în aplicații în care se prelucrează date de 8 biți (cu excepția sistemelor foarte mici). Se relevă adesea că manipularea propriu-zisă a datelor ocupă o parte relativ redusă din program.

O altă problemă de programare care poate necesita un efort de programare și un timp de execuție importante este *schimbarea contextului*. Cele mai multe microcomputere trebuie să rezolve mai multe sarcini, prin programe separate. În cazul sistemelor reale de microcomputere controlul programului sare în permanență între diferite programe și subrutine. Seturile moderne de instrucțiuni permit ca aceste schimbări de context să se facă mai rapid și fără efort. De exemplu, microprocesoarele din seria 9900 produse de Texas Instruments realizează o schimbare completă de context prin simpla selecție a unui nou grup de „registre" de lucru prin intermediul unei singure instrucțiuni. Este posibil orice număr de contexte diferite, avînd în vedere că registrele sînt de fapt, în acest caz, locații în RAM.

Alegerea setului de instrucțiuni este o decizie foarte importantă deoarece este posibil ca întreaga firmă să se blocheze într-un anumit stil de lucru. Deși este ușor ca microcomputerele noi să includă seturile vechi de instrucțiuni, schimbarea setului de instrucțiuni înseamnă că programele vechi se vor rescrie, programatorii trebuie reciclați, sistemele de dezvoltare schimbate ș.a.m.d. Astfel, dificultatea de a schimba un set de instrucțiuni impune o previzionare atentă în momentul deciziei. Costul suplimentar al unui set de instrucțiuni mai puternic tinde să dispară în timp. De asemenea necesitățile firmei tind să crească în timp către sisteme mai complexe. Alegerea microprocesorului trebuie astfel să ia în mod serios în considerație nevoile viitoare ca și ambițiile și perspectivele producătorului.

EXERCIȚII

1. (*a*) Dintre cele două variante de program de la sfîrșitul Secțiunii 7-4 care este mai rapidă ?
   (*b*) Dacă fiecare instrucțiune ia 2 μs, găsiți timpul necesar pentru a stoca 100 de caractere în fiecare variantă ?
   (*c*) Cît timp ar lua dacă perioada ciclului este de 1 μs ?
   (*d*) Cît timp ar lua varianta cu buclă dacă ar fi disponibilă o instrucțiune „Incrementează registrul index și fără ramificație dacă rezultatul este zero" ?
   (*e*) Să se realizeze o organigramă detaliată a variantei cu buclă.
2. Cele 4 acumulatoare ale unui 8900 au următorul conținut :

   AC0 = 5, AC1 = 27, AC2 = 3, AC3 = 255.

   (*a*) Ce vor conține după următoarele instrucțiuni de lucru cu stiva :

```
PUSH      AC0
PUSH      AC1
PUSH      AC2
PULL TO   AC1
PULL TO   AC0
PUSH      AC3
PULL TO   AC2
PULL TO   AC3
```

   (*b*) Ce vor conține după execuție următoarelor instrucțiuni ?

```
REGISTER ADD AC0 TO AC1
REGISTER AND AC2 TO AC0
REGISTER EXCLUSIVE OR AC2 TO AC3
```

3. Ce va conține registrul *A* al unui microprocesor 8008 după următoarea secvență de instrucțiuni :

```
MOVE IMMEDIATE 11001000
MOVE TO B FROM A
INCREMENT B
ADD B TO ACCUMULATOR
```

4. Alcătuiți o organigramă pentru un program ce produce o ramificație la o subrutină din opt, în funcție de rezultatul celor mai puțin semnificativi trei biți din registrul *A*.
5. Faceți o organigramă ce generează o ramificație la o subrutină din patru, în funcție de conținutul (în binar) al registrului *A*, după cum urmează :

Valoarea	Go to
0— 60	SR1
61—130	SR2
131—138	SR3
139—255	SR4

6. Scrieți un program pentru 8080 care va face ca :
   (*a*) Registrul *A* să numere (în permanență) ca un numărător binar.
   (*b*) Registrul *A* să numere (în permanență) ca un numărător de 8 biți Mocbius.
7. (*a*) Realizați o organigramă a unui program ce va implementa ecuația 4-7.
   (*b*) Scrieți programul corespunzător pentru microprocesorul 8008.
   (*c*) Calculați numărul de circuite integrate cu porți necesare pentru implementarea funcției cu logică cablată și folosiți această valoare pentru a calcula raportul biți de program/număr de circuite integrate cu porți.
8. Realizați o organigramă pentru un program ce produce o ramificație la una din trei subrutine în funcție de registrul *A*, *B*, *C* care conține valoarea cea mai mare.
9. Ce va conține registrul *A* după ce se execută „Exclusive OR to A from A"
10. Realizați o organigramă a unui program ce se ramifică dacă conținutul lui *A* satisface :
    (*a*) Ecuația 4-5.
    (*b*) Ecuația 4-6.
11. Cîți biți de ROM sînt necesari pentru a implementa ecuațiile 4-5 și 4-6 în acest mod ?
12. Realizați organigrama unui program cu buclă ce se va ramifica dacă conținutul registrului *A* satisface ecuația 4-7. Folosiți o subrutină generală din problema 10 pentru a evalua fiecare mintermen.
13. Cîți biți de program în ROM sînt echivalenți cu ROM-ul de 1 024 biți ce a implementat ecuațiile (4-4)...(4-7) ?

BIBLIOGRAFIE

Microprocesoare

Intel, MCS-85 *Users Manual*, Intel Corp., Santa Clara, Calif., 1978, Pub. # 9800366D.

Katz, Jeffery, Morse, Stephen P., Pohlman, Williams B., Ravenel, Bruce W., „8086 Microcomputer Bridges the Gap Between 8 and 16 Bit Designs", *Electronics*, February 16, 1978, pag. 99—104.

Lofthus, Alan și Ogden, Deene, „16 Bit Processor Performs Like Minicomputer", *Electronics*, May 27, 1976, pag. 99—105.

Morse, Stephen, Pohlman, Wiliam B., and Ravenel, Bruce W., „The Intel 8086 Microprocessor: A 16-Bit Evolution of the 8080", *Computer*, June 1978, pag. 18—27.

Scrupski, Stephen, „Minis, Look Out: Here Comes a Powerful Family of 16-Bit C Chips", *Electronic Design* 12, June 7, 1978, pag. 54—48.

Texas Instruments, *990 Computer Family Systems Handbook*, Texas Instruments, Austin, Texas, 1975, Manual # 945250—9701.

Microcalculatoare

Bryant, John D. and Longley, Rick, „16-Bit Microcomputer Is Seeking a Big Bite of Low-Cost Controller Tasks", *Electronics*, June 23, 1977, pag. 118—123 (TI9940).

Intel, *Application Techniques for the MCS-48 Family*, Intel Corp., Santa Clara, Calif., 1977, Pub. # 98—413B.

Intel, *MCS-48 Users Manual*, Intel Corp., Santa Clara, Calif., 1978, Pub. # 98—270C.

Seturi de instrucțiuni

Osborne, Adam, *An Introduction to Microcomputers*, Osborne, Berkeley, Calif., 1975 (Două volume; al doilea stringe la un loc foarte multe foi de catalog).

Sisteme de intrare-ieșire și de întreruperi pentru diverse microprocesoare

Kingman, Edwing E., Microprocessor Systems Design, Prentice-Hall, Englewood Cliffs, N.J., 1977.

Concepte de programare

Booth, Taylor, *Digital Networks and Computer Systems*, Wiley, New York, 1971 (Informații despre programare, asamblare, compilatoare, algoritmi).

Computer Automation, The Value of Power, Doc. # 89A00064A-C, General Automation, Anaheim, Calif., 1972 (O excelentă trecere în revistă a seturilor de instrucțiuni și a modurilor de adresare).

Minicalculatoare și calculatoare mari

Bell and Newell, *Computer Structures: Readings and Examples*, McGraw-Hill, New York, 1971 (Descierea structurii și filozofiei de proiectare a 40 de calculatoare existente).

8

LOGICA PROGRAMATĂ II
Programarea asistată de calculator

8.1. ASAMBLOARE

Deşi uneori programele sînt folosite pentru a genera liste de interconexiuni sau pentru a ajuta în alt mod proiectantul de *logică cablată*, lipsa de standardizare a făcut ca proiectarea asistată de calculator în acest domeniu să aibă o răspîndire redusă. Spre deosebire de logica cablată, logica programată este perfect adecvată pentru a fi implementată cu ajutorul calculatorului. De fapt, *succesiunea de coduri asociată unui program este generată în mod automat de către programe specializate, denumite asambloare.*

Un asamblor transformă în mod automat un „limbaj uman" într-un limbaj binar, al calculatorului. Programatorul scrie pur şi simplu într-un limbaj apropiat de cel natural. Acest program, denumit **programul sursă** este apoi introdus într-un calculator ce este dotat cu programul asamblor. Acesta din urmă transformă programul sursă într-un **program obiect,** codat în binar, ce poate fi executat de către microcalculator. Se efectuează în paralel o imprimare (listare) a codului obiect în binar şi a programului sursă. Deoarece programul sursă este mai uşor de citit putem înţelege astfel fiecare linie a codului maşină, ce are o reprezentare binară, prin citirea **declaraţiei** de program sursă tipărită pe aceeaşi linie, la dreapta.

Programul asamblor verifică, de asemenea, formatul programului sursă şi rezolvă toate detaliile legate de alocarea adreselor. Pentru a face programul mai lizibil (cu „autoexplicitare") asamblorul permite efectuarea de *comentarii,* care nu au efect asupra programului. Aceste comentarii sînt tipărite pur şi simplu în partea dreaptă a listării programului, pentru a-l face mai inteligibil.

Programatorul (inginerul) scrie o linie a programului sursă pentru fiecare instrucţiune necesară. În loc să treacă numele întreg al instrucţiunii, el foloseşte o *abreviere mnemonică* (pe scurt mnemonică). De obicei, mnemonicile sînt constituite din trei pînă la patru litere esenţiale din numele instrucţiunii. Aceasta le face mai uşor de memorat şi mai simplu de scris. Tabelul 8-1 prezintă mnemonicile recunoscute de către asamblorul 8008 (MCS8) şi codurile de operaţie (în binar) pe care le produc. De exemplu, dacă dorim executarea unei instrucţiuni „Roteşte registrul *A* spre dreapta cu bitul de transport", („Rotate *A* register thru carry"), scriem pur şi simplu RAR şi asamblorul produce în mod automat 00011010, care este codul corect al operaţiei, pentru programul obiect.

Majoritatea instrucţiunilor cer programatorului informaţii suplimentare necesare pentru completarea cîmpurilor variabile, cum sînt : registrul sursă sau destinaţie, adresa sau operandul imediat (vezi fig. 7-5). Această informaţie este furnizată după mnemonica instrucţiunii, apărînd la unul sau mai multe

Tabelul 8-1. **Mnemonicele instrucțiunilor microprocesorului Intel 8008[a]**

Descrierea operației	Asamblor		Biții codului operației		
	Mnemonica	Operandul	76	543	210
Move data to reg/mem from reg/mem	MOV	dst, src	11	DDD	SSS
Move immediate data to reg/mem	MVI	dst, data	00	DDD	110
			BB	BBB	BBB
Increment register contents	INR	dst	00	DDD	000
Decrement register contents	DCR	dst	00	DDD	001
Add reg/mem to A register	ADD	src	10	000	SSS
Add immediate to A register	ADI	data	00	000	100
			BB	BBB	BBB
Add reg/mem to A register with carry	ADC	src	10	001	SSS
Add immediate data to A register with carry	ACI	data	00	001	100
			BB	BBB	BBB
Subtract reg/mem from A register	SUB	src	10	010	SSS
Subtract immediate data from A register	SUI	data	00	010	100
			BB	BBB	BBB
Subtract reg/mem from A register with borrow	SBB	src	10	011	SSS
Subtract immediate data from A register with borrow	SBI	data	00	011	100
			BB	BBB	BBB
Logical AND reg/mem to A register	ANA	src	10	100	SSS
Logical AND immediate data to register	ANI	data	00	100	100
			BB	BBB	BBB
Exclusive OR reg/mem to A register	XRA	src	10	101	SSS
Exclusive OR immediate data to A register	XRI	data	00	101	100
			BB	BBB	BBB
OR reg/mem to A register	ORA	src	10	110	SSS
OR immediate data to A register	ORI	data	00	110	100
			BB	BBB	BBB
Compare reg/mem to A register	CMP	src	10	111	SSS
Compare immediate data to A register. No change to A	CPI	data	00	111	100
			BB	BBB	BBB
Rotate A reg. Left (A_7 to A_0 and Carry)	RLC	—	00	000	000
Rotate A reg. Right (A_0 to A_7 and Carry)	RRC	—	00	001	010
Rotate A reg. Left thru Carry	RAL	—	00	010	010
Rotate A reg. Right thru Carry	RAR	—	00	011	010
Unconditional Jump to specified address	JMP	adr	01	XXX	100
			BB	BBB	BBB
			XX	BBB	BBB
Jump to specified address if condition satisfied:[a] (JNC, JNZ, JP, JPO, JC, JZ, JM, JPE)	J[a]	adr	01	CCC	000
			BB	BBB	BBB
			XX	BBB	BBB
Unconditional Call subroutine at specified address, push current address into stack for use by Return	CALL	adr	01	XXX	110
			BB	BBB	BBB
			XX	BBB	BBB
Call subroutine at specified address and push current address if condition:[a] (CNC, CNZ, CP, CPO, CC, CZ, CM, or CPE)	C[a]	adr	01	CCC	010
			BB	BBB	BBB
			XX	BBB	BBB
Unconditional Return from Subroutine	RET	—	00	XXX	111
Subroutine					
Return if:[a] (RNC, RNZ, RP, RPO, RC, RZ, RM, or RPE)	R[a]	—	00	CCC	011
Restart at mem adr AAA000, push prog counter	RST	adr	00	AAA	101
Input port 0—7 to A register	IN	port	01	00M	MM1

Tabelul 8-1 (continuare)

Descrierea operației	Asamblor		Biții codului operației		
	Mnemonica	Operandul	76	543	210
Output port 0—24 from A register	OUT	port	01	RRM	MM1
Stop program until interrupted	HALT	—	00	000	00X
Stop program until interrupted	HALT	—	11	111	111

a Nota 1 : Alte litere pentru mnemonica ramificațiilor de program condiționate, semnificația lor și codurile corespunzătoare de conducție : NC (No Carry-Transport nul) 000 NZ (Not Zero-Nenul) 001, P (Positive) 010, PO (Parity Odd-Paritate impară), C (Carry-transport) 100, Z (Zero) 101, M (Minus) 110, PE (Parity Even — Paritate pară) 111.

Nota 2 : src și dst indică sursa și destinația datelor, după cum urmează : registrul A (000), registrul B (001), registrul C (010), registrul D (011), registrul E (100), registrul N (101), registrul L (110), Adresa de memorie H' L (111).

Nota 3 : X = indiferent, octeții adiționali sînt indicați prin BB, BBB.

a Vezi și figura 7-5.

spații libere (blancuri) distanță după mnemionica operației. De exemplu, pentru a incrementa registrul *C*, vom scrie INR C, iar asamblorul „traduce" în 00010000, ceea ce reprezintă codul instrucțiunii de incrementare, unde 010 apare în cîmpul registrului de destinație pentru a indica registrul *C*.

Unele operații solicită două informații suplimentare, separate de o virgulă. De exemplu, pentru a transfera date în registrul *B* din registrul *C*, vom scrie simplu MOV B, C. Asamblorul transformă aceasta în 11001010, unde 001 indică registrul *B*, iar 010, registrul *C*.

Datele necesare pentru operanzii imediați sau pentru precizarea adreselor *(adr)* pot fi definite ca un număr zecimal, sau hexazecimal (urmate de *H*). De exemplu :

ADI 5	este tradusă în	10000111
		00000101
MVI M,5AH	este tradusă în	00111110
		01011010
MVI B,1	este tradusă în	00001110
		00000001
JNZ 127	este tradusă în	01001000
		01111111

Remarcați că în ultimul exemplu JNZ semnifică „sari dacă nu este zero" (acumulatorul) — Jump if Not Zero Condition. Asamblorul transformă NZ în codul de condiție 001. Mnemonicile instrucțiunilor JUMP, CALL, RETURN au fiecare opt forme posibile, așa cum se indică în Nota 1 la tabelul 8-1.

Deși uneori dorim să specificăm o adresă ca un număr, cum s-a făcut în ultimul exemplu, aceasta este adesea neconvenabil. De obicei, dorim să sărim la începutul unei alte părți a programului, a cărui adresă nu o cunoaștem în momentul scrierii programului. Pentru acest motiv, toate asambloarele fac posibilă o *adresare simbolică, care ne permite să asignăm un nume arbitrar oricărei adrese.* De exemplu, putem *eticheta* TAPE1 prima instrucțiune a rutinei READ TAPE. Pentru salt la rutina respectivă, scriem pur și simplu JMP TAPE1, iar asamblorul va produce codul binar al instrucțiunii (trei octeți).

Eticheta (numele) este asignată plasînd-o înaintea primei instrucțiuni din rutina READ TAPE. De exemplu, dacă prima instrucțiune a rutinei este INPUT (intrare) de la portul 3, vom scrie :

```
TAPE 1 : IN 3     ; READ TAPE ROUTINE
```

Această linie va genera o instrucțiune de un singur octet (01000111), dar asamblorul va „memora" simbolul RTAPE 1 și locația sa pentru utilizare ulterioară. READ TAPE ROUTINE este *total ignorată* de către asamblor deoarece apare după punctul și virgula ce urmează operandului (3). Aceste cuvinte, denumite *comentarii*, sînt introduse numai pentru a face programul mai explicit. De fapt, *fiecare linie de program poate fi concepută ca avînd patru „cîmpuri", după cum urmează :*

Etichetă	*Operație*	*Operand*	*Comentariu*
TAPE 1 :	IN	3	; READ TAPE ROUTINE

Pentru a separa cîmpul operației de cel al operandului, este necesar, din punctul de vedere al asamblorului, doar un caracter blanc (spațiu). Cîmpul *etichetei* (menționat uneori ca *nume* sau *locație)* începe întotdeauna cu primul caracter din stînga liniei și se încheie întotdeauna cu două puncte. Etichete se acordă numai liniilor la care vrem să ne referim ulterior, astfel încît *majoritatea liniilor vor începe cu unul sau mai multe blancuri, pentru a indica absența cîmpului etichetei.*

Cîmpul următor este cel al *operației*, care definește mnemonic instrucțiunea. Dacă instrucțiunea este de tipul ce presupune un *operand*, pentru a indica începutul cîmpului corespunzător se plasează unul sau mai multe blancuri după mnemonica instrucțiunii. Pentru operațiile care nu solicită operand (de ex. RLC), un punct și virgulă plasat după cîmpul operandului arată că începe cîmpul comentariilor.

Cîmpul operandului poate să indice sursa sau destinația *(src* sau *dst)* prin registrele *„A, B, C, D, E, H,* sau *L"* sau *„M"* pentru operații cu memoria. *Datele (data)* ce formează operandul imediat, adresele *(adr), porturile* de intrare sau ieșire pot fi definite în unul din următoarele moduri :

1. Constante zecimale, de exemplu : 1, 15, 255 sau 2586.
2. Constante hexazecimale, urmate de *H*, de exemplu 1H, FH, FFH sau A1AH.
3. Caractere ASCII în ghilimele ca, de exemplu, „M", „N", „2", „&".
4. Adrese simbolice definite ca etichete în alte părți, de exemplu RTAPE1.
5. Expresii aritmetice ce conțin etichete, de exemplu RTAPE1+4.
6. Expresii aritmetice ce conțin locația curentă, de e emplu $-3.

Acest ultim tip este necesar deoarece, avînd în vedere că asamblorul ține evidența locațiilor, nu știm în momentul scrierii programului unde va ajunge instrucțiunea curentă în memorie. Dacă, de exemplu, dorim să facem saltul cu trei octeți înaintea începutului instrucțiunii curent, vom scrie pur și simplu $-3 în cîmpul operandului, iar asmblorul va calcula adresa corectă. De observat că aici $ semnifică „adresa curentă". Dacă dorim să ajungem la al patrulea cuvînt (octet) de la adresa pe care am etichetat-o RTAPE1, scriem pur și simplu RTAPE1+4 și ansamblorul va calcula în cursul asamblării adresa solicitată.

Remarcați că expresiile aritmetice folosite în asambloare sînt transformate în coduri binare *în cursul asamblării*, astfel încît *vor include numai date*

ce sînt definite complet în cursul asamblării. Cîmpul de *comentarii* este strict opțional și este ignorat de către asamblor. El este utilizat doar pentru a face programul mai lizibil. Prin folosirea adecvată a comentariilor, programul se poate „auto-explica", în sensul că oricine poate să-i urmărească funcționarea fără alte completări. Dacă o linie de program începe cu *două puncte* (:) în poziția caracterului celui mai din stînga, asamblorul va ignora întreaga linie, astfel încît putem folosi întreaga linie drept comentariu. Este astfel posibil să includem „adnotări" asupra programului chiar în structura acestuia.

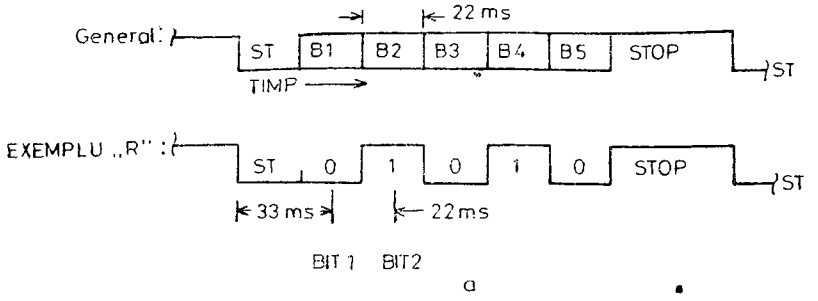
8.1.1. Un exemplu de programare

Înainte de a intra în alte detalii ale programelor de asamblare, să încercăm să folosim cîteva din principiile de bază pe care le-am descris pentru scrierea unui program real în limbajul de asamblare. Vom scrie un program de recepție a datelor serie de tipul celor generate de către un Teletype echipat pentru comunicație pe o rețea Telex sau Western Union. Această comunicație este similară modului în care se transmit datele de către un calculator, dar este mai lentă și fiecare caracter are numai cinci biți. Figura 8-1*a* ilustrează formatul semnalului în general și prezintă, ca un exemplu, un cod pentru *R* (sau 4 în modul „numere"). Biții sînt transmiși unul cîte unul cu o durată de 22 ms/bit. Cînd nu se transmit date, linia este în permanență în starea „1" (codul lui „stop"). Cînd se transmit date, bitul de 0 (bitul de start) este transmis primul. Sincronizarea este realizată măsurînd timpul de la prima tranziție negativă la centrul fiecărui bit. Intervalele sînt astfel egale cu 1,5 din durata unui bit, sau 33 ms (cel măsurat de la tranziția de start pînă la centrul primului bit) și cu 22 ms între fiecare bit, în continuare.

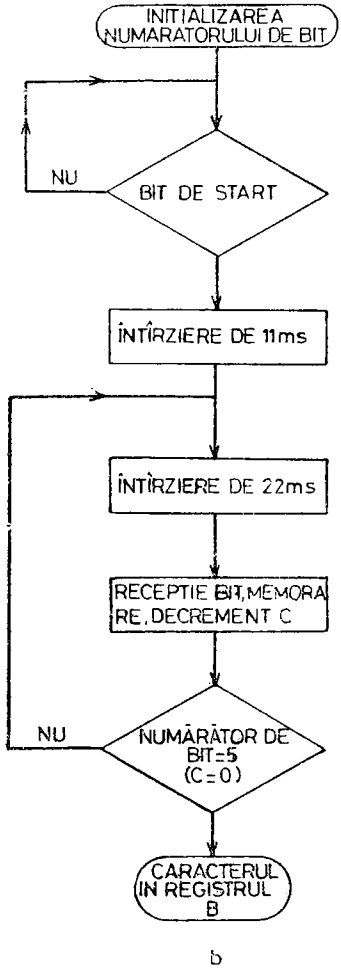
Majoritatea teleimprimatoarelor ce folosesc acest cod utilizează un comutator mecanic, declanșat de către bitul de start, ce se rotește cu 420 rot/min pentru a transforma acest semnal serie în cinci biți la care avem acces, paralel, memorați pe cale mecanică. Deși această abordare mecanică poate părea depășită, ea este încă și astăzi, de departe, cel mai răspîndit mod de recepție a acestui tip de semnal. Desigur, putem proiecta un circuit electronic pentru a face același lucru utilizînd un registru de deplasare, un numărător, un generator de tact și o anumită logică. Există, de asemenea, un circuit LSI special, denumit UART (Universal Asynchronaus ReceiverTransmitter-Receptor-transmițător asincron universal), care va realiza funcția folosind un releu sau un **izolator optic** pentru interfața cu linia și o anume logică pentru a-l cupla la bus-ul de intrare al procesorului.

Vom realiza aceeași logică prin programare, cuplînd la linie, printr-un optoizolator, *o singură* intrare a unui procesor. Această aplicație constituie astfel un exemplu direct de înlocuire a logicii cablate cu programarea. În Capitolul 11, vom ataca aceeași problemă, folosind însă un alt mod de abordare.

Figura 8-1*b* prezintă o organigramă destul de detaliată a programului necesar. Se observă că avem două blocuri ce reprezintă întîrzieri. Figura 8-2 prezintă o organigramă detaliată a rutinei „11 ms DELAY". Această rutină utilizează registrul *D* pentru a forma o buclă de contorizare ce obține întîrzierea necesară. În cazul microprocesorului Intel 8008, instrucțiunea INCREMENT ia 20 μs, iar JUMP, 44 μs, astfel că întîrzierea totală pe buclă este de 20 + 44 = 64 μs. În felul acesta, trebuie să parcurgem bucla de



a



b

Fig. 8-1. Recepția serială a datelor (cod Baudot); *(a)* formele de undă ale semnalului recepționat; *(b)* organigrama programului.

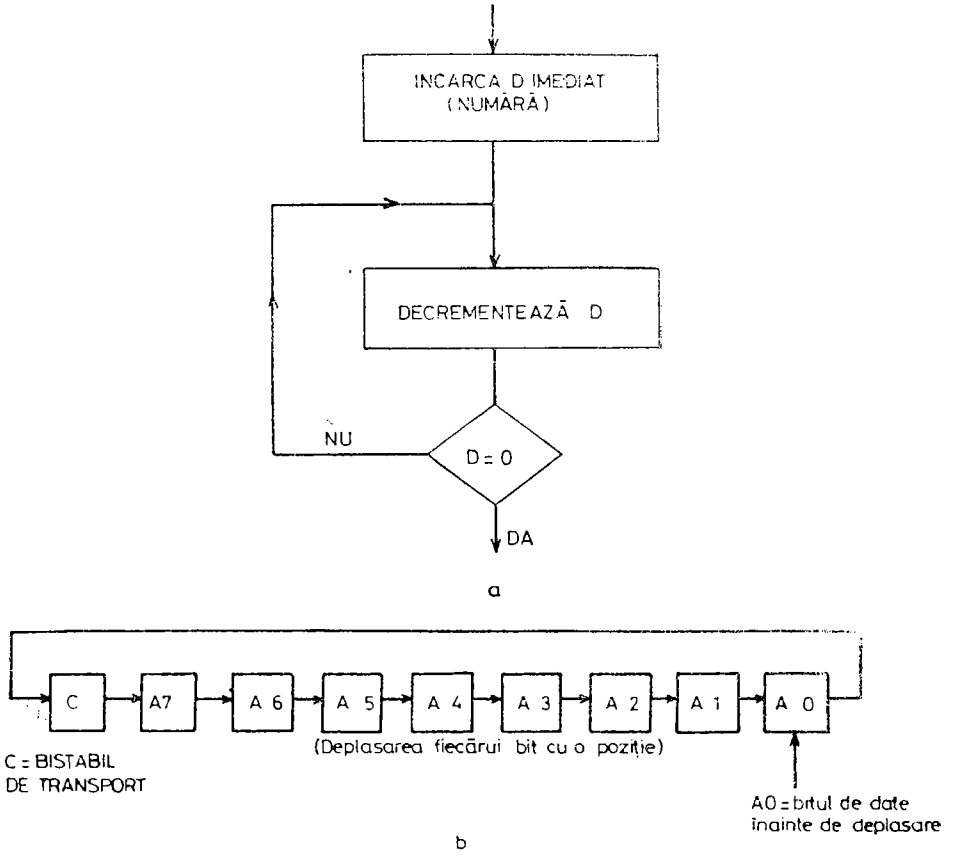


Fig. 8-2. *(a)* **Rutina pentru** întîrzierea de 11 ms (blocul din figura 6.12 *b*) *(b)* Instrucțiunea ROTATE A RIGHT (rotire la dreapta).

11000/64 = 172 ori pentru a genera o întîrziere de 11 ms. Programul nostru de realizare a întîrzierii este, în formă simbolică, următorul :

Etichetă	Operație	Operand	Comentariu
LOOP 1 :	MVI	D,172	; ÎNTÎRZIERE 11MS
	DCR	D	
	JNZ	LOOP1	

Aici putem observa utilitatea adreselor simbolice, deoarece, în acest stadiu, deşi nu avem nici o idee unde va fi localizată în memorie această parte a programului, totuşi trebuie să facem o referinţă la o adresă anterioară, pentru instrucţiunea JUMP. Alegînd o denumire arbitrară, LOOP1, putem să ne referim la ea fără să ne preocupăm de adresa efectivă şi să lăsăm programul asamblor să se îngrijească de detalii. Deoarece acesta este un JUMP simplu, am putea prefera să-l definim în termeni de adresă curentă (**$**), fără a mai defini nici un simbol :

```
MVI     D,172 ;   ÎNTÎRZIERE 11MS
DCR     D
JNZ     $-1
```

În ambele moduri, programul obiect, produs de către asamblor, va fi identic. A doua metodă este însă periculoasă în cazul unei mașini cu cuvînt de lungime variabilă, deoarece poate conduce la erori, datorită numărării biților.

„Întîrzierea de 22 ms" este o problemă ceva mai grea deoarece întîrzierea maximă pe care o putem genera cu bucla simplă este de $256 \times 64 = 16\,700 =$ $= 16{,}7$ ms. Modul uzual de generare a întîrzierilor mai mari este de a plasa *o buclă în interiorul altei bucle*. Această metodă poate produce o întîrziere de pînă la $16{,}7 \times 256 = 4{,}27$ s. Putem totuși obține o întîrziere suplimentară, în cazul nostru, doar prin introducerea unei alte instrucțiuni în buclă. Dacă de exemplu, adăugăm „Compară registrul A cu memoria" (CMP M) nu apare nimic nedorit, dimpotrivă mărim durata buclei noastre la $64 + 32 = 96$ μs. Numărul necesar este astfel $22\,000/96 = 230$. Rutina noastră "întîrziere 22 ms" devine astfel :

```
        MVI     D, 230      ; ÎNTÎRZIERE 22 MS
LOOP2   CMP     M
        DCR     D
        JNZ     LOOP2
```

La sfîrșitul fiecărui interval de 22 ms vom avea centrul unui nou bit, astfel încît vom introduce un bit în bitul 0 al registrului A. Vom plasa apoi bitul respectiv în bistabilul de transport, prin efectuarea unei rotații a acumulatorului spre dreapta (ROTATE A RIGHT- vezi fig. 8-2 *a*, apoi vom transfera (MOVE) biții acumulați din B în A și vom roti A spre stînga (ROTATE A LEFT) pentru a deplasa noul bit înapoi (din transport în A_0) deplasîndu-i pe ceilalți peste o poziție. Vom deplasa (MOVE) apoi datele acumulate din B în A, vom incrementa și testa contorul de biți. În cod simbolic (cu comentarii) aceste operații se prezintă după cum urmează :

```
IN      0       ; INTRODU NOUL BIT ÎN A0
RAR             ; PUNE NOUL BIT ÎN CARRY
MOV     A, B    ; PUNE BIȚII PRECEDENȚI ÎN A
RAL             ; DEPLASEAZĂ TOȚI BIȚII SPRE STÎNGA
MOV     B, A    ; PĂSTREAZĂ ÎN B
DCR     C       ; DECREMENTEAZĂ CONTORUL DE BIȚI
JNZ     LOOP2   ; REIA BUCLA DACĂ NU ESTE BITUL 5
```

Singura chestiune care mai rămîne este partea de început a organigramei din figura 8-1 *b*. Trebuie să inițializăm contorul de biți (registrul C) cu 5 și să executăm o instrucțiune HALT. Astfel, procesorul se va opri doar atunci cînd se primește o întrerupere. Deoarece linia de date (DATA) este legată la intrarea de întreruperi a procesorului (ÎNTRERUPT) și la bitul 0 din portul de intrare 0, *tranziția de start* va produce reluarea programului cu prima instrucțiune din întîrzierea de 11 ms. Un alt mod de a aștepta START-ul, ar fi rularea unei bucle ca :

```
IN      0
CPI     0
JNZ     $—3
```

Cît timp intrarea este 1 (stop), programul rămîne în această buclă. *Tranziția de start* produce realizarea condiției *zero*, permițînd programului să continue cu întîrzierea de 11 ms. Deoarece metoda descrisă mai sus pretinde trei octeți suplimentari în program, vom utiliza o întrerupere pentru

a detecta bitul de start. Prin gruparea segmentelor pe care le-am discutat, se poate scrie programul complet, ce este prezentat în Tabelul 8-2, așa cum este listat de un asamblor.

Tabelul 8-2. **Listingul asamblorului**

```
                ;RUTINĂ PENTRU RECEPTOR DE DATE, SERIE, TELEX, 45 BAUD
0000  1605             MVI C, 5        ;INIȚIALIZAREA CONTORULUI DE BIȚI
0002  00               HLT             ;AȘTEAPTĂ ÎNTRERUPEREA DE START
0003  1EAC             MVI D, 172      ;INIȚIALIZEAZĂ ÎNTÎRZIEREA DE 11MS
0005  19               DCRD            ;DECREMENTEAZĂ D
0006  480500           JNZ $-1         ;REIA BUCLA PINA CÎND D = 0  (11MS)
0009  1EE6       L00P2 : MVI D, 230    ;INIȚIALIZEAZĂ ÎNTÎRZIEREA DE 22MS
000B  BF         L00P1 : CMP M         ;ÎNTÎRZIERE ADIȚIONALĂ
000C  19               DCR D           ;DECREMENT D
000D  480B00           JNZ LOOP1       ;REIA BUCLA PÎNĂ CÎND D=0 (22MS)
0010  41               IN 0            ;PREIA NOUL BIT
0011  1A               RAR             ;PUNE BITUL ÎN CARRY
0012  C1               MOV A, B        ;PUNE BIȚII PRECEDENȚI ÎN A
0013  12               RAL             ;DEPLASEAZĂ TOȚI BIȚII SPRE STÎNGA
0014  C8               MOV B, A        ;PASTREAZĂ ÎN B
0015  11               DCR C           ;DECREMENTEAZĂ CONTORUL DE BIȚI
0016  480900           JNZ LOOP2       ;REIA BUCLA DACĂ NU ESTE BITUL 5
```

Remarcați că listarea de asamblor conține declarațiile programului sursă în același format (etichetă, operație, operand, comentariu) în care au fost scrise. La stînga textului sursă el imprimă locațiile de memorie repartizate și codurile instrucțiunilor în *hexazecimal*. Imprimarea conține patru coloane. Coloana din stînga conține adresa în hexadecimal a primului octet al instrucțiunii. În continuare apare codul instrucțiunii, folosind două cifre hexazecimale pentru fiecare octet al instrucțiunii (de la stînga la dreapta).

Prima linie a imprimării arată că prima instrucțiune (MVI C,5) este stocată în octeții de memorie 0 și 1. Primul octet este 16 în hexadecimal, sau 00 010 110, unde 010 reprezintă registrul *C*. Al doilea octet este operandul imediat, 5 în zecimal, sau 0000 0101. Următoarea instrucțiune (HLT) este depusă în locația 2 și conține numai zerouri. Asamblorul generează în mod automat codurile cerute ale operațiilor și alocă adrese consecutive. Remarcați că al doilea octet al JNZ $—1 este calculat de către asamblor, scăzînd 1 din locația curentă (6 — 1 = 5). Notați, de asemenea, că asamblorul urmărește adresa etichetată L00P2 și completează „9" drept cîmp de adresă pentru declarația din programul sursă JNZ LOOP2.

Programul final solicită 25 de octeți în memoria de instrucțiuni. Deoarece el realizează aceeași funcție ca și receptorul de date serie (implementat în logică cablată), sau ca receptorul din circuitul LSI UART, este interesant de comparat costul biților de program cu al celor din logica convențională. La începutul lui 1978 costul biților de ROM era de 0,05 cenți/bit, în situația unui ROM de tip MOS de 65 kbiți, achiziționat în cel puțin 100 de bucăți. Costul la care ar urca receptorul de date serie programat ar fi de 8 × 25 octeți × 0,05 = 10 cenți ! Acesta este un alt exemplu asupra invincibilității logicii programate.

Urmărind din nou programul propriu-zis, este clar că el se „auto-explică", în sensul că oricine este antrenat în programare poate urmări listarea. De asemenea, se pot aduce extrem de ușor modificări de format. De exemplu,

putem să-l transformăm într-un receptor de opt biți doar prin schimbarea primei instrucțiuni în MVI C,8. Pentru a modifica temporizarea, schimbăm operanzii instrucțiunilor MVI din cele două bucle de întîrziere. De fapt, dacă acești parametri sînt stocați într-o memorie operativă în care se poate scrie sau din care se poate citi (RWM-read/write memory) de la orice adresă în orice ordine (RAM-random acces memory), putem să-i inițializăm înaintea introducerii programului, obținînd astfel mai multe formate și viteze de transmisie cu același program.

Acum, după ce am văzut ce putem face cu un asamblor, să vedem ce alte resurse ne mai oferă un astfel de instrument.

8.1.2. Instrucțiuni de asamblor (Pseudo-op-uri)

Pînă în momentul de față, toate expresiile noastre erau convertite de către un asamblor în coduri de instrucțiuni. Instrucțiunile asamblorului, sau *pseudo-op-urile*, sînt mnemonice ce apar în cîmpul operației, care nu se convertesc direct în instrucțiuni mașină dar sînt folosite pentru controlul procesului de asamblare, pentru a defini simboluri, pentru a genera date, sau pentru a permite generarea cu o singură declarație a secvențelor de instrucțiuni frecvent utilizate.

Programul asamblor alocă în mod automat adrese de memorie consecutive fiecărei linii de program, în ordine. El începe cu adresa zero, dacă nu i se spune să facă altfel printr-o pseudooperație de tip *Origin* (Definește originea — *ORG*). Dacă, de exemplu, dorim ca programul să înceapă de la locația 1 000, vom scrie următoarea pseudo instrucțiune (pseudo-op), înaintea primei linii a programului sursă :

```
ORG   1 000
```

Putem folosi din nou ORG în program, dacă dorim ca o altă parte a programului să înceapă la o locație diferită. De observat că expresia ORG nu produce instrucțiuni ce vor fi executate de mașină (sau pe scurt instrucțiuni mașină). Putem acorda simbolurilor (etichetelor) orice valoare dorim prin folosirea pseudo-op-ului *Equate* (*EQU*) (Acordă valoare). De exemplu,

```
TADR EQU 2
```

definește simbolul TADR ca avînd valoarea 2. Prin această expresie nu sînt produse instrucțiuni mașină în programul final, însă odată definit, simbolul TADR poate fi folosit în cîmpul operandului, rezultatul fiind același ca în cazul în care am fi scris un 2. Acest fapt poate fi util în momentul în care un anumit parametru (cum ar fi de exemplu adresa unei unități de bandă) este folosit în mod repetat în program. Parametrul poate fi redefinit și programul reasamblat, prin modificarea unei singure declarații EQU, în locul parcurgerii întregului program și al modificării valorii parametrului de fiecare dată cînd apare.

Putem *defini constante și tabele de date* utilizînd pseudo-op-ul *Define byte* (Definește octetul, DB). De exemplu,

```
MESS1 DB "PRINTOUT"
```

definește opt locații consecutive de memorie, ce conțin codurile caracterelor *P, R, I, N, T, O, U* și *T*. Putem să ne referim simbolic la adresa primei litere, prin MESS1. Remarcați că am semnalizat date alfanumerice (ASCII) prin

folosirea ghilimelelor, plasate înainte și după date. Putem, de asemenea, defini tabele de date. De exemplu :

```
MASKS  DB  1,0FH,240,TADR + 2
```

definește patru cuvinte consecutive: 00000001, 00001111, 11110000 și 00000100. Adresa simbolică a primului cuvînt este MASKS (măști) încît putem, de exemplu, utiliza a treia mască punînd direct MASKS + 2 în cîmpul operandului. Remarcați că al patrulea cuvînt a fost calculat de către asamblor din definiția noastră anterioară a TADR.

Putem astfel genera tabelul de date corespunzător afișajului cu șapte segmente (Tabelul 7-3), folosit în exemplul din Capitolul 7

```
SEG7  DB  7 EH, 30H,GDH, 79H,33H, 5BH,1FH, 30H,7FH, 73H
```

Programul de conversie a codurilor BCD din registrul *A* în codul 7 segmente poate apoi utiliza această denumire simbolică, după cum urmează :

```
ORI   SEG7   ;GENEREAZĂ ADRESA (SEG7 + CONȚINUTUL A)
MOV   L,A    ;PUNE ADRESA ÎN REGISTRUL
MOV   A,M    ;ÎNCARCĂ CODUL 7 SEGMENTE
```

Pentru a genera tabele de adrese de 16 biți putem utiliza pseudo-op-ul *Define Word* (Definește cuvînt, *DW*). De exemplu :

```
LINK  DW  SUB1,SUB2,SUB3
```

va genera o tabelă de 6 octeți conținînd adresa binară a SUB1 în primii doi octeți (cel mai puțin semnificativ fiind primul), urmată de către SUB2 și SUB3. Pseudo-op-ul DW este necesar deoarece în timp ce adresele simbolice sînt utilizate cu ușurință în asamblor, ele apar finalmente în programul asamblat *sau ca locații ce conțin instrucțiunii sau doar în cîmpul de adrese al instrucțiunilor.* Definind o adresă ca dată, permitem programului obiect asamblat să folosească el însuși adresa (de exemplu pentru a calcula adresele, la urmărirea timpului de execuție). *Este important să fim conștienți de această distincție între ceea ce este disponibil în cursul execuției programului asamblor și ceea ce este disponibil în cursul execuției programului obiect asamblat.*

Uneori este de dorit ca să se asambleze mai multe versiuni ale unui program prin introducerea sau eliminarea unor secțiuni ale programului. De exemplu, dacă un sistem este vîndut cu diferite opțiuni privind echipamentele de I/O, putem dori să eliminăm acele porțiuni ale programului pentru care nu există echipament corespunzător. Desigur, aceasta se poate realiza prin scrierea programului în module separate de program sursă pe bandă magnetică sau cartele și realizînd gruparea fizică corespunzătoare a benzilor sau cartelelor. Un mod mult mai „curat" de a face același lucru constă în utilizarea pseudo-op-ului de *asamblare condiționată, IF.* Fiecare modul de program este precedat de o declarație *IF* și este urmat de o declarație *ENDIF.* Dacă expresia din cîmpul *operand* al declarației *IF* este nulă asamblorul ignoră întregul bloc în cauză din programul sursă. De exemplu, un program READ TAPE (citește banda) poate fi asamblat condițional ca mai jos :

```
IF        TAPE
:
(programul TAPE)
:
ENDIF
```

Adăugînd o singură declarație de control

```
TAPE    EQU 1
```

sau

```
TAPE    EQU 0
```

putem în mod efectiv include sau elimina întreg programul pentru bandă.

Constatăm adesea că în programele noastre folosim în mod repetat aceeași secvență de instrucțiuni. Pseudo-op-ul *MACRO* ne permite să acordăm un nume unei secvențe de instrucțiuni și apoi *să facem asamblorul să producă instrucțiunile respective de fiecare dată cînd plasăm MACRO în cîmpul operației din programul nostru sursă*. De exemplu, în cazul lui 8008 putem dori adesea să transferăm adrese de 14 biți în grupul de registre *HL*, pentru a defini o adresă înaintea unei operații cu memoria. În loc să utilizăm pentru aceasta două instrucțiuni, putem defini un MACRO denumit LXI (Load eXtended Immediate) după cum urmează :

```
LXI     MACRO   ADR
        MVI     H,ADR/256
        MVI     L,ADR
        ENDM
```

După ce am definit acest MACRO putem scrie, direct :

```
        LXI     3597
```

aceasta avînd același efect ca și cum am fi scris :

```
        MVI     H,3597/256
        MVI     L,3597
```

Asamblorul va produce în mod automat secvența corectă de 4 octeți pe baza unei singure declarații. Divizarea prin 256 ne dă valoarea octetului celui mai semnificativ al adresei, în timp ce *L* este încărcat cu cei mai puțin semnificativi opt biți ai ei.

Să observăm că un MACRO *nu este o subrutină* fiindcă secvența este inclusă în program integral, de fiecare dată cînd este menționat numele său. Am putea, de exemplu, să definim un DIVIDE MACRO (macro de divizare) și apoi să-l utilizăm în mod repetat în program, dar rezultatul ar fi mai puțin eficient decît dacă am sări de fiecare dată la aceeași subrutină DIVIDE (divide). Asambloarele includ adesea mai multe MACRO-uri gata definite. De exemplu, asamblorul MCS-8 include LXI. Acesta oferă de fapt programatorului un set de instrucțiuni extins pentru lucru.

Pseudo-op-ul END indică asamblorului sfîrșitul programului. El trebuie să fie inclus întotdeauna după ultima declarație din programul sursă.

O caracteristică importantă a asambloarelor este faptul că ele *verifică programul sursă*, pentru a detecta erorile. *Listarea programului* va conține un *indicator de eroare* (o literă) în fiecare linie unde s-a detectat o eroare. Uneori se poate solicita o imprimare separată care prezintă numai liniile cu erori și mesajele de eroare. În continuare se prezintă cîteva exemple de mesaje de eroare și tipul de erori care le provoacă :

ILLEGAL CHARACTER (caracter ilegal) : Un caracter special (cum ar fi de exemplu $, /.,) a apărut în declarație (nu însă în comentariu), sau poate lipsește un cîmp necesar de operand.

MULTIPLY DEFINED SYMBOL (simbol definit multiplu) : Simbolul a fost definit mai mult decît o dată.

UNDEFINED SYMBOL (simbol nedefinit) : Simbolul a fost utilizat dar nu a fost definit niciodată.

ILLEGAL NUMERIC CONTAINS CHARACTER (expresia numerică conține caracter) : Un număr hexazecimal conține o cifră interzisă (de exemplu, *G* sau *N*), sau expresia numerică conține caractere nenumerice.

ILLEGAL OPCODE (opcod ilegal) : Codul operației nu face parte din mnemonica acceptată.

MISSING OPERAND FIELD (cîmp de operand lipsă) : În cazul unui cod de operație ce impune operand, acesta lipsește.

ILLEGAL VALUE (valoare ilegală) : Valoarea numerică a unui număr hexazecimal sau zecimal dintr-o anumită expresie a depășit limita.

ILLEGAL SYMBOL (simbol ilegal) : Un cîmp de locație conține un simbol care are mai mult decît cinci caractere sau nu începe cu un caracter alfabetic.

MISSING LABEL (etichetă lipsă) : Eticheta, solicitată de către pseudo-op-ul EQU lipsește.

SYMBOL TABLE OVERFLOW (depășire a tabelei simbolurilor) : În programul sursă există prea multe simboluri pentru a încape în tabelul alocat pentru simboluri.

LINE OVERFLOW MAXIMUM (depășire de linie) : Linia de intrare depășește 48 de caractere sau lipsește „carriage return" (carul la început de rînd).

ERRONEOUS LABEL (etichetă eronată) : Opcodurile END și ORG pot nu au etichetă.

ILLEGAL ORIGIN (origine ilegală) : Valoarea noii origini este mai mică decît valoarea calculată de program.

ILLEGAL OPERAND (operand incorect) : Codul de operație DAD pretinde operand simbolic.

Pe lîngă generarea unei *listări* combinate a programelor sursă și obiect, un asamblor poate produce *cartele perforate sau bandă de hîrtie* conținînd codurile binare ale programului obiect. De obicei, această bandă are un format ce permite încărcarea sa în memoria scrie/citește (RWM sau RAM) a microprocesorului, folosind un *program de încărcare binar* stocat în microprocesor. Uneori banda începe cu un program de tip *bootstrap* (*autoîncărcător*), ce face posibilă încărcarea benzii chiar într-un procesor ce conține doar primele cîteva instrucțiuni ale încărcătorului. După acești octeți de început, controlul este preluat de către programul bootstrap ce se autoîncarcă și asigură în continuare și încărcarea programului obiect propriu-zis.

Prezența unui *încărcător relocatabil* în microreceptor face posibilă încărcarea programului începînd la orice locație datorită de memorie. Este posibil astfel să încărcăm mai multe module ale programului în părți diferite ale memoriei. Banda ce conține programul obiect trebuie să fie în *format relocatabil*, pentru ca programul încărcător să poată *calcula în timpul încărcării, adresele corecte ale tuturor cîmpurilor de program ce reprezintă referiri la memorie.*

Un alt format de bandă perforată este necesar dacă banda de program urmează să fie folosită pentru a programa un PROM pe un programator. Aceste benzi folosesc *un caracter pentru fiecare bit* ce urmează să fie programat. Formatul BNPF utilizează *B* pentru a defini începutul unui cuvînt, *N*-uri și *P*-uri pentru a indica 0 sau 1 și *F* pentru a indica sfîrșitul unui cuvînt.

8.1.3. Asamblorul microprocesorului 8900

Secțiunea precedentă s-a ocupat în principal cu asamblorul MCS-8 (pentru microprocesorul 8008). Deși toate asambloarele se bazează pe principii comune, ele diferă în privința unor detalii cum ar fi caracterele de control utilizate, abrevierile, mnemonicele folosite pentru pseudo-op-urile și formatele de date acceptate. Deși asamblorul MCS-8 este tipic din multe puncte de vedere, modurile sale limitate de adresare îl fac mai simplu decît majoritatea asambloarelor.

În unele privințe 8900 este mult mai tipic deoarece setul de instrucțiuni include o gamă completă de moduri de adresare. Datorită acestui motiv, cîmpul *operandului* din programul sursă este ceva mai complex. Tabelul 8-3 prezintă mnemonicile și formatul utilizat de către 8900. De exemplu :

LD 0,7(3)

înseamnă „Încarcă registrul 0 din adresa de memorie existentă în registrul index 3, sumată cu 7". Aceasta este tradusă de către asamblor în 1000001100000111 (vezi fig. 7-3. *a*). Adresarea *indirectă* este indicată prin precedarea adresei de către @. De exemplu,

LD 0,@7(ß)

constituie o instrucțiune LOAD INDIRECT (încarcă indirect), ce permite ca datele să fie încărcate în registrul 0, din adresa de memorie *indicată* de către *conținutul* locației de memorie 7, la care se *adună* conținutul registrului index ß.

Tabelul 8-3. **Mnemonicele și operanzii limbajului de asamblare pentru microprocesorul 8900**

Instrucțiune	Mnemonică	Operand[a]	Format (fig. 7-4)
Load	LD	r, disp(Xr)	a
Load Indirect	LD	r, @ disp(Xr)	a
Store	ST	r, disp(Xr)	a
Store Indirect	ST	r, @ disp(Xr)	a
Add	ADD	r, disp(Xr)	a
Subtract	SUB	r, disp(Xr)	a
Skip if AND is Zero	SKAZ	r, disp(Xr)	a
Skip if Greater	SKG	r, disp(Xr)	a
Skip if Not Equal	SKNE	r, disp(Xr)	a
And	AND	r, disp(Xr)	a
Or	OR	r, disp(Xr)	a
Jump	JMP	disp(Xr)	b
Jump Indirect	JMP	@ disp(Xr)	b
Jump to Subroutine	JSR	disp(Xr)	b
Jump to Subroutine Indirect	JSR	@ disp(Xr)	b
Increment and Skip if Zero	ISZ	disp(Xr)	b
Decrement and Skip if Zero	DSZ	disp(Xr)	b
Push on to Stack Register	PUSH	r	d
Pull from Stack	PULL	r	d
Add Immediate, Skip if Zero	AISZ	r, data	d

Tabelul 8-3 (continuare)

Instrucțiune	Mnemonică	Operand[a]	Format (fig. 7-4)
Load Immediate	LI	r, data	d
Complement and Add Immediate	CAI	r, data	d
Shift Left	SHL	r, ##	d
Shift Right	SHR	r, ##	d
Rotate Left	ROL	r, ##	d
Rotate Right	ROR	r, ##	d
Exchange Registers and Top of Stack	XCHRS	r	d
Register Copy	RCPY	sr, dr	e
Exchange Registers	RXCH	sr, dr	e
Register And	RAND	sr, dr	e
Register Exclusive OR	RXOR	sr, dr	e
Register Add	RADD	sr, dr	e
Branch-on Condition	BOC	cc, disp	c
Return from Subroutine	RTS	ctl	f
Return from Interrupt	RTI	ctl	f
Jump to Subroutine Implied	JSRI	ctl	f
Set Flag	SFLG	fc	f
Pulse Flag	PFLG	fc	f
Push Flags on Stack	PUSHF		f
Pull Flags from Stack	PULLF		f
Register in	RIN	port	f
Register out	ROUT	port	f
Halt	HALT		

[a] r = numărul registrului; Xr = numărul registrului index (3 sau 4) (relativ sau indirect folosit dacă nu este specificat); disp = deplasare; fc = codul indicatorului (flag-ului): L, OV, CY sau GFO-12 pentru Link, Overflow, Carry, sau General Flags; ctl = control (adăugat la adresa de revenire); port = port de intrare sau ieșire (se adună la conținutul lui AC3 pentru a se obține numărul portului de intrare sau ieșire).

Dacă nu se specifică un registru index, asamblorul calculează în mod automat deplasarea și folosește modul de adresare relativ la program. De exemplu:

```
LD   1,TABLE2
```

produce încărcarea registrului 1 cu conținutul locației etichetate TABLE2. Asamblorul scade adresa curentă din adresa etichetată TABLE2. Dacă rezultatul este cuprins în gama permisă de ±127, codul obiect rezultant specifică modul de adresare *relativ la amplasarea programului*, conținînd deplasarea corectă în cîmpul deplasării. Dacă rezultatul iese din această gamă, se generează o instrucțiune LOAD INDIRECT cu modul *pagină de bază*, ce adresează un *cuvînt indicator* ce este de asemenea produs de către asamblor. Acest cuvînt indicator conține adresa directă de 16 biți a locației TABLE2.

Mai există și alte diferențe minore între asambloarele 8900 și MCS-8. De exemplu, simbolul pentru *adresa prezentă* este „.” în loc de „$”. Datele binare pentru operanzi sînt reprezentate în formă hexazecimală fiind *precedate*, iar nu urmate de *H*. Imprimarea codului obiect 8900 este, de asemenea, diferită în sensul că locația este tipărită în formă zecimală în prima coloană și de asemenea în formă hexazecimală în a doua coloană. Codul operației este imprimat numai ca un număr hexazecimal în coloana 3.

Aceste diferențe minore apar la fiecare nou asamblor, deoarece nu există o standardizare reală. Scopul nostru este doar acela de a introduce conceptele generale. Manualele furnizate de către producători oferă informația necesară pentru urmărirea unor detalii ale asamblorului particular folosit. De fapt, un producător dispune adesea de mai multe versiuni diferite de asamblor, deoarece un asamblor mai elaborat solicită pentru a fi rulat un computer mai evoluat (adică cu mai multă memorie sau cu mai multe dispozitive periferice.)

8.2. COMPILATOARE

Compilatoarele sînt programe similare asambloarelor, dar ele ridică nivelul automatizării proiectării spre cel al organigramelor sau cel al declarațiilor de probleme. În cazul unui compilator, programul sursă nu seamănă cu „limbajul mașină" legat de setul de instrucțiuni. Limbajele compilatoarelor sînt *orientate pe probleme* : ele sînt proiectate pentru a exprima problema în modul cel mai eficient. Programul compilator traduce apoi această exprimare a problemei în *limbaj mașină*, producînd un cod obiect binar pentru procesorul pentru care a fost scris.

Unele limbaje de programare au devenit atît de standardizate încît compilatoarele pentru ele sînt disponibile practic pentru orice computer existent pe piață. FORTRAN constituie un exemplu de astfel de limbaj ; programele scrise în FORTRAN pot fi compilate și rulate pe aproape orice calculator (cu excepția microcomputerelor de astăzi). Acest fapt este utilizat într-un sens avantajos de către producătorii de microcomputere. Scriind în FORTRAN programele ce solicită un computer în configurație completă, aceste programe pot fi compilate și rulate pe practic orice tip de computer. Intel are de exemplu, un *cross assembler* (pentru asamblarea programelor microcomputerelor), un *simulator* (pentru a permite verificarea programelor pentru microcomputere pe o mașină mai mare prin folosirea unor programe de execuție a instrucțiunilor microcomputerelor) și chiar un compilator scris în limbajul FORTRAN.

Un alt avantaj al compilatoarelor este faptul că programele supraviețuiesc încă mult timp după ce computerul a devenit depășit. FORTRAN a fost introdus inițial de către IBM pentru a fi folosit pe calculatoarele sale 704 și 709. Astăzi, aceste calculatoare sînt depășite și chiar calculatoarele IBM actuale folosesc un set de instrucțiuni complet diferit. Totuși, programe scrise în FORTRAN pot fi încă compilate nu numai pe noile computere IMB ci și pe orice alt calculator ce este cel puțin un minicomputer.

Un dezavantaj al compilatoarelor este faptul că ele programează ca un programator lipsit de imaginație, dar exact. Deoarece cuplează doar subrutine pentru a realiza funcția cerută de către limbajul sursă, ele folosesc adesea multe instrucțiuni pentru a face lucruri pe care un programator inteligent le-ar putea rezolva cu mai puține instrucțiuni. Desigur, unele compilatoare sînt mai eficiente decît altele, dar, în general, ele produc programe ce utilizează mai multă memorie și/sau se rulează mai încet decît programele scrise în limbaj de asamblare. Timpul se află, totuși, de partea compilatorului ; pe măsură ce salariile programatorilor și inginerilor cresc, costul memoriei

se reduce, iar viteza procesoarelor crește. De pe acum, programarea sistemelor ce nu vor fi produse în serie mare se realizează cel mai bine în *limbaje evoluate* (cum sînt denumite limbajele compilatoarelor). Costul memoriei adiționale și al procesorului este depășit în mod substanțial de cel al programării.

O altă problemă este aceea că, în general, astăzi limbajele de programare standard sînt orientate pe soluționare de probleme economice (COBOL, RPG) sau științifice (FORTRAN, ALGOL, BASIC). Problemele științifice impun o precizie foarte mare, astfel încît limbajele științifice ca FORTRAN manipulează datele, în general, în notația cu virgulă mobilă. Pentru sistemele digitale simple acest fapt poate fi complet neeconomic. De asemenea, limbajele științifice sînt optimizate pentru introducerea de date formate (de exemplu, cartele perforate), prelucrarea lor în cadrul unor operații algebrice complexe și furnizarea de rezultate formate (de exemplu, imprimări). În același timp, limbajul BASIC este mai „cu picioarele de pămînt", dar reduce în mod substanțial viteza și eficiența de stocare pentru a permite o elaborare rapidă, interactivă de programe mici, orientate pe terminale.

8.2.1. PL/M

PL/M este, probabil, compilatorul cel mai folosit în cadrul microcomputerelor (alte denumiri : MPL, PL/I). El este de fapt un subset al limbajului PL1 (IMB) dar seamănă cu PASCAL și cu strămoșul lor comun ALGOL.

Un program în PL/M este scris într-o formă mult mai liberă decît un program scris în limbajul de asamblare. În loc de a avea doar o declarație pe linie într-o singură linie se pot face mai multe *declarații. Punctul și virgula*(;) *plasate după o declarație* indică sfîrșitul ei. *Etichetele sînt întotdeauna urmate de două puncte* (:), iar comentariile sînt întotdeauna marcate de /* și */ de fiecare parte (de exemplu, /* COMENTARIU*/). *Identificatorii* simbolici pot avea orice lungime și pot folosi orice caractere alfanumerice, în măsura în care încep cu o literă și nu conțin spații. În locul spațiilor, se pot folosi semne de dolar pentru a îmbunătăți lizibilitatea identificatorului (de exemplu, INPUT COUNT). *Cuvintele speciale, rezervate,* ce fac parte din limbaj, nu pot fi utilizate ca identificatori (de exemplu, OR, PLUS, DO, IF, etc.).

În general programatorul poate neglija existența adreselor. Problema este pusă prin simpla definire a variabilelor, apoi prin efectuarea unor declarații asupra variabilelor. Aceste *declarații* pot fi făcute într-o formă foarte lizibilă, ca :

LUNGIME = ARIE/LĂȚIME

Deși microprocesorul INTEL nu conține instrucțiunea de *împărțire,* această simplă declarație este suficientă pentru a defini ceea ce dorim să se facă. Compilatorul inserează în mod automat în programul obiect o subrutină *DIVIDE* (împarte) pentru a realiza operația, urmărind variabilele pe baza *identificatorilor* lor simbolici.

Mai înainte ca identificatorul să poată fi folosit într-o declarație, el trebuie definit cu un enunț de tip *DECLARE.* Declarația precizează dacă variabila trebuie să fie manipulată ca un *BYTE* (octet) sau ca o variabilă de adresă (ADDRESS, de 16 biți). De asemenea, precizează uneori o *valoare inițială.*

Datele, in declarații și enunțuri, pot fi specificate în oricare din următoarele forme :

212	*Zecimal* (se poate folosi D)
11010100B	*Binar*
324Q	*Octal* (se poate folosi O)
D4H	*Hexazecimal*
'T'	*Caracter ASCII*

Remarcați că cele cinci exemple de mai sus reprezintă *aceleași* date binare într-o formă diferită.

Pentru a economisi spațiul, în prezentarea PL/M vom enunța declarațiile și enunțurile permise într-o formă standard. Vom folosi *A*, *B*, *C*, *D* și *E* pentru a reprezenta *identificatorii*. Vom defini formele acceptabile prin exemple. Cîteva exemple de variații de la forma generală sînt prezentate decalate spre dreapta, sub formele generale, iar explicațiile apar în coloana din dreapta, acolo unde este necesar.

Mai întîi, *declarațiile* :

DECLARE (A, B, C) BYTE ;	*Identificatorii A. B* și *C* sînt variabile de opt biți
DECLARE A BYTE ;	*A* este o variabilă de opt biți
DECLARE (A, B) ADDRESS ;	*A* și *B* sînt variabile de 16 biți
DECLARE A(100) BYTE ;	Identificatorul *A* reprezintă o tabelă indiciată de variabile de opt biți A(0), A(1),...,A(99), la care în diverse declarații se pot face referiri sub această formă ; *A* este denumit un *vector*
DECLARE A(50) ADRESS ; D	Se formează un tabel de 100 de octeți, la care ne putem referi ca A(0), A(1),...,A(49)
DECLARE (A, B, C) (100) BYTE	Definește trei tabele de 100 de octeți.
DECLARE (A, B, C) BYTE INITIAL (1, 27, 42)	Variabilele de un octet *A*, *B* și *C* au valori inițiale de 1, 27 și 42 (în zecimal)
DECLARE A ADDRESS INITIAL 27H	Variabila de 16 biți *A* are valoarea inițială de 27 (hexadecimal)
DECLARE A DATA (1, 27, 575)	A(0), A(1) și A(2) sînt stocate în ROM ca 1, 27, 575 (în zecimal) ; dimensiunea variabilă este alocată în mod automat, de opt sau 16 biți, astfel că A(0) și A(1) vor fi octeți iar A(2) va fi adresă.
DECLARE A LITERALLY 2350H	Ori de cîte ori *A* va fi utilizat undeva în program, se va substitui valoarea hexadecimală 2350.
DECLARE A LITERALLY 'ERROR STOP'	Este acceptată orice lungime a șirului de caractere, pînă la 256 de octeți.
DECLARE A BYTE, (B, C) ADDRESS, D, DATA 27Q	După un „DECLARE" pot fi listate mai multe declarații.

În principiu, înșiruirea anterioară ține cont de toate tipurile generale de declarații posibile. Rețineți că literele *A*, *B*, *C*, ș.a.m.d., sînt folosite aici doar pentru simplitate. De exemplu, DECLARE A BYTE în mod normal ar avea un identificator mai descriptiv decît *A*, ca DECLARĂ LUNGIME BYTE. După ce am declarat variabila LUNGIME ca dată de 8 biți, o putem folosi liber în *enunțuri* pe care le prezentăm în continuare. Compliatorul calculează în mod automat funcțiile definite de enunțuri la precizia cea mai mare folosită în cazul variabilelor utilizate. Dacă însă apar multiplicări sau divizări, rezultatul este în mod automat o variabilă de 16 biți. În general, enunțurile pot folosi *expresii* similare ca formă cu expresiile algebrice și logice. Totuși, deoarece expresiile pot fi imprimate ca o secvență de caractere, poate apare o anumită confuzie dacă nu folosim paranteze. Cu excepția parantezelor, compilatorul PL/M realizează operațiile într-o ordine fixă de prioritate. *Lista de mai jos cuprinde toți operatorii permiși de către expresiile PL/M, în ordinea priorității :*

/,*, MOD
+, —, PLUS, MINUS
<, < =, < >, =, > =, >
NOT
AND
OR, XOR

Majoritatea acestor simboluri au o semnificație evidentă, dar, deoarece ele trebuie să fie constituite dintr-un set de caractere care nu include cîteva din simbolurile matematice standard, sînt necesare unele explicații :* indică înmulțire, MOD indică *restul împărțirii*, < = indică *mai mic decît* sau *egal*, <> indică inegal, > = indică *mai mare decît* sau *egal*, NOT *completează* toți biții dintr-o variabilă, iar XOR este pur și simplu *EXclusive OR*.

În mod normal, expresiile sînt evaluate de către PL/M prin efectuarea, în primul rînd a operațiilor din linia superioară a listei de mai sus și apoi a restului operațiilor, în ordine. Operatorii ce apar pe aceeași linie în lista menționată au aceeași prioritate și sînt evaluați de la *stînga spre dreapta*, pe măsură ce apar în expresie. Astfel, sînt necesare uneori paranteze pentru a defini corect expresia dorită. De exemplu, în :

VALOAREA $ MEDIE = (VALOAREA 1 + VALOAREA 2)/2

parantezele sînt necesare, altfel se va calcula mai întîi VALOAREA 2/2 ce va fi adunată la VALOAREA 1.

Să continuăm rezumatul nostru, indicînd exemple de posibile *declarații* :

A = B + C*D	Enunț simplu
A,E = (B + C)*D	Definește atît *A* cît și *E* ca fiind egale cu (*B* + *C***D*.
A = A + 1	Incrementează *A*
A = B(6) — 3	*A* = Al șaptelea element din tabela *B*, minus 3 (B(0) este primul element).
A = B(C + 27) + 5	Indicii pot să fie chiar o expresie
A = B AND 15	Maschează cei mai puțin semnificativi 4 biți ai *B* și trece rezultatul în *A*.

IF A > *B* THEN A = B	Enunțul condiționat $A = B$ este aplicabil numai dacă $A > B$
IF A > *B* THEN A = B ELSE A = 0 ;	Un enunț condiționat mai complex ; dacă A nu este mai mare decît B, se aplică $A = 0$
IF A = 1 THEN B = C/D IF A = B/C THEN D = 1 GO TO A	Va transfera controlul programului la locația A ; A este o etichetă definită de către „*A*" : plasat în fața unui enunț din altă zonă a programului. (Enunțurile GO TO fac ca un program să fie dificil de urmărit, încît se consideră nerecomandabilă utilizarea lor).
HALT	Oprește mașina pînă la întrerupere
DO WHILEA = 3 ; . . END	Un enunț de grup ; se formează o buclă, ce produce execuția repetată a elementelor între DO și END, atît timp cît expresia A = 3 este adevărată.
DO A = 1 TO 10 . . END;	Variabila A va avea valoarea inițială 1 și se va incrementa la fiecare trecere prin buclă ; cînd A va atinge 10, programul va continua.
DO A = 1 TO B + 1 : . END;	Atît valoarea inițială cît și cea finală pot fi *expresii*
DO A = 1 TO 10 BY 2 . . END;	La fiecare parcurgere a buclei, A va fi incrementat cu 2, încît bucla va fi executată numai de cinci ori.

IF A = B THEN
DO ;
.
.
.
END;
	Dacă $A = B$, se execută toate declarații plasate între DO și END ; altfel, trece la declarația de după END ; se poate astfel executa, în mod condiționat, un întreg bloc de program.

DO CASE A—5 declarația 0 declarația 1 declarația 2 . . . END ;	O ramificație multiplă. Va fi executată numai una dintre declarații, în funcție de expresia A-5 ; de exemplu, dacă A-5 este zero, va fi executată numai enunțul 0 ; dacă A-5 este 1 va fi executat numai enunțul 1 ; programul continuă cu enunțul ce urmează după END

Fiecare din aceste declarații definește o funcție care pentru a fi definită ar lua multe instrucțiuni de limbaj de asamblare. Compilatorul manipulează pentru noi toate detaliile, pe baza declarațiilor orientate pe probleme, care

descriu ceea ce noi dorim să se facă. Deoarece putem folosi *expresii* pentru a determina chestiuni ca indici, adrese, condiții de ramificare sau număr de iterații în buclă, foarte mulți pași de program pot fi combinați într-un singur enunț. Pentru a face limbajul și mai puternic, putem defini *PROCEDURI* (subrutine), similar modului în care am definit MACRO-urile în limbajul de asamblare, dar *pe care le putem utiliza aici în expresii.*

Mai întîi, trebuie să definim procedura :

```
A :   PROCEDURE (B, C) ADDRESS ;
      DEFINE (B, C, D) ADDRESS ;
      .
      .
      .
      RETURN D ;
      END ;
```

Declarațiile dintre DEFINE și RETURN definesc pe (*D*), rezultatul ce trebuie *returnat* ca funcție de variabilele *B* și *C*. Putem acum utiliza în orice expresie (*A*(*E*, *F*)) ca o variabilă. *E* și *F* sînt folosite de către subrutină, ca variabilele *B*, respectiv *C*, pentru a calcula *D* în cursul execuției programului. Rezultatul (*D*) este apoi substituit în mod automat în expresie, în locul lui *A* (*E*, *F*).

Pe lîngă *procedurile* pe care le putem scrie noi înșine, limbajul PL/M a fost dotat cu cîteva proceduri proprii :

SHL (A, B)	Deplasează spre stînga variabila *A*, de 8 sau 16 biți cu *B* poziții.
SHR (A, B)	Deplasează spre dreapta variabila *A*, de 8 sau 16 biți, cu *B* poziții
ROL (C, B)	Rotește variabila de octet *C* spre stînga cu *B* poziții
ROR (C, B)	Rotește variabila de octet *C* spre dreapta cu *B* poziții
CARRY ZERO SIGN PARITY	= 1 dacă condiția indicată este satisfăcută
INPUT (A)	Reprezintă octetul obținut printr-o încărcare de la portul *A*
OUTPUT (A)	Produce transferul unui octet spre portul de ieșire *A*

Putem utiliza aceste denumiri de *PROCEDURI* în expresii, ca și cum ar fi identificatori de variabilă.

IF CARRY = 1 THEN B = SHL (B, 1)	Va deplasa spre stînga datele reprezentate de către identificatorul *B*, cu o poziție, dacă bistabilul de condiție indică existența unui transport anterior.
OUTPUT (2) = A + 1	Va incrementa *A* și îl va transfera portului de ieșire 2.

OUTPUT(2)=INPUT(3)*INPUT(2)+A;	Va transfera spre portul de ieșire 2 produsul valorilor existente la porturile de intrare 2 și 3, plus valoarea lui *A*.

Un alt tip de *PROCEDURĂ* nu returnează date programului, astfel încît va fi doar chemată (CALL), în loc să fie utilizată în expresii. O astfel de procedură este definită ca parte din limbajul PL/M. Fie ca exemplu un program de întîrziere.

Putem să-l chemăm cu ușurință astfel :

CALL TIME (A);	Se va produce o întîrziere egală cu de *A* ori 0,1 ms ; ulterior, se reia programul
CALL TIME (25)	Va produce o întîrziere de 2,5 ms

Etichetele de adresă pot fi atașate la orice enunț din program, precedînd declarația cu o etichetă urmată de *două puncte* (:). De exemplu :

A : B = C + 1	Atribuie identificatorul *A* ca adresă a enunțului
247 : A = B + 1	Este similar unui ordin ORG dintr-un asamblor, prin aceea că atribuie adresa 247 respectivei declarații ; compilatorul va continua de aici cu adreselece urmează.
B = .A + 27	.A înseamnă „adresa etichetei *A*", în felul acesta adresele pot fi manipulate ca părți în expresii.
EOF	Este necesar la încheierea programelor PL/M pentru a indica sfîrșitul (End Of File)

Cu aceasta încheiem rezumatul nostru asupra limbajului PL/M. Pentru a-l folosi efectiv, este necesară cunoașterea mai multor detalii care sînt disponibile în manualul de PL/M al producătorului.

Și acum pentru a utiliza cele învățate despre PL/M, să încercăm să reprogramăm în PL/M programul de *Recepție a datelor serie* pe care l-am scris anterior în limbaj de asamblare. Referindu-ne la organigrama din figura 8-1 *b*, vedem că ea este, de fapt, *mult mai detaliată* decît trebuie să fie. Mai întîi, bucla prin care sînt numărați biții și inițializarea numărului de biți poate fi manipulată folosind pur și simplu un grup *DO*. De asemenea, nu este nevoie să ne preocupăm ce registre să utilizăm ; vom folosi doar un simbol CHAR, pentru a reprezenta datele. Putem, de asemenea, să uităm artificiul cu bitul de transport, din moment ce putem deplasa caracterul spre stînga și putem realiza *OR* cu noul bit. de la intrare. Programul PL/M propriu-zis este prezentat mai jos.

```
/* RUTINA PENTRU RECEPTOR DE DATE, SERIE, TELEX, 45 BAUD*/
   HALT; /* WAIT FOR START INTERRUPT*/
   DECLARE (CHAR, I) BYTE;
   CALL TIME (110); /* START DELAY*/
```

```
DO I = 1 TO 5 ;
CALL TIME (220) ;  /*BIT DELAY*/
CHAR = SHL (CHAR, 1) OR INPUT (0) ;
END ;
```

În principal, el urmărește același tipar ca și programul în limbaj de asamblare. În loc să scriem bucle de întîrziere, putem pur și simplu să chemăm procedura TIME pentru cele două întîrzieri. Remarcați că pentru a deplasa caracterul spre stînga după iterația precedentă *și* pentru a face OR-ul cu noul bit, este necesar doar un enunț. Faptul important este că acolo unde programul în limbaj de asamblare a luat 16 linii de cod (destul de abstract) PL/M ia doar șapte. În plus, programul PL/M este mult mai lizibil și prin urmare mai ușor de scris și cu mai puține greșeli. Totuși, fără îndoială, codul rezultant ia în plus cîțiva octeți de memorie.

În cazul programelor pentru sistemele mari, PL/M face posibilă reducerea substanțială a timpului de programare și a erorilor prin organizarea programului într-o *structură pe blocuri*. Structura pe blocuri este echivalentul în programare al împărțirii în „cutii negre" a hardware-ului, discutată în Secțiunea 2-9. Fiecare bloc de program începe pe o singură pagină și conține un program suficient pentru a fi pretențios, dar nu excesiv de mare. Acest nivel de asamblu ignoră detaliile și definește simplu interacțiunea blocurilor de program importante cărora li se dau nume descriptive de *PROCEDURI*. Următorul nivel de diviziune constă dintr-o pagină de cod pentru fiecare procedură. Într-un sistem simplu aceste blocuri de o pagină pot fi adecvate pentru a defini complet procedura, în mod direct. Dacă aceasta nu este posibil, detaliile sînt lăsate la o parte folosind din nou denumiri descriptive de proceduri. Aceste proceduri vor fi ulterior definite în blocuri de programare mai detaliate.

În felul acesta, este posibil să utilizăm *procedurile în interiorul procedurilor din interiorul procedurilor*, pentru a diviza un program într-o structură de arbore cu suficient de multe nivele pentru a defini un program de *orice* complexitate, fără a avea de-a face cu mai mult decît o pagină de cod la un moment dat. Deoarece toată munca este realizată „pe bucățele", se minimizează confuzia și apar mai puține erori. În cazul *programării structurate* complet, fiecare bloc trebuie să înceapă la capul paginii și să se încheie în josul ei, fără a fi permise intrări sau ieșiri intermediare. Din acest motiv, enunțurile GO TO nu sînt admise.

Figura 8-3 prezintă o mostră de program în PL/M constînd din cinci blocuri scurte. Scopul programului este imprimarea unui tabel cu rădăcinile pătrate ale numerelor cuprinse între 1 și 1000. Blocul la nivelul cel mai general al programului este pe liniile 52—69. Linia 68 este un exemplu perfect asupra compactității care poate fi realizată prin ignorarea detaliilor, ce sînt definite în blocurile de nivel mai scăzut. Linia 68 face să se tipărească numărul ce este rădăcina pătrată a lui *I*. Calculul efectiv al numărului este definit de o procedură denumită SQUARE$ROOT, ce este definită în detaliu de către blocul de nivel inferior de pe liniile 5—12. Detaliile imprimării propriu-zise sînt definite în blocul procedurii PRINT$NUMBER, aflat în liniile 37-50. În interiorul acestui bloc se face o referire la o altă procedură pe linia 49. Procedura PRINT$STRING definește, la rîndul său, mai multe detalii asupra operației de tipărire, dar întîrzie încă detaliile de reprezentare concretă a caracterelor individuale, referindu-se la procedura PRINT$CHAR de pe linia 33. Detaliile finale ale manipulării individuale serie a biților sînt definite în liniile 14—27. Aceasta reprezintă cel mai de jos nivel al diviziunii, la fel cum o face

```
PASS-1 BLOCK
       LEVEL

00001    2        2048:  /* IS THE OR;GIN OF THIS PROGRAM */
00002    2     DECLARE TTO LITERALLY '2', CR LITERALLY '150', LF LITERALLY '0AH
00003    2             TRUE LITERALLY '1', FALSE LITERALLY '0';
00004    2
00005    2     SQUARE$ROOT: PROCEDURE(X) BYTE;
00006    3         DECLARE (X,Y,Z) ADDRESS;
00007    3         Y = X; Z = SHR(X+1,1);
00008    3             DO WHILE Y <> Z;
00009    3             Y = Z; Z = SHR(X/Y + Y + 1, 1);
00010    4             END;
00011    3         RETURN Y;
00012    3         END SQUAREROOT;
00013    2
00014    2     PRINT$CHAR: PROCEDURE (CHAR);
00015    3       DECLARE BIT$CELL LITERALLY'691',
00016    3             (CHAR,I) BYTE;
00017    3         OUTPUT (TTO) = 0;
00018    3         CALL TIME (BIT$CELL);
00019    3             DO I = 0 TO 7;
00020    3             OUTPUT(TTO) = CHAR; /* DATA PULSES */

00021    4             CHAR = ROR(CHAR,1);
00022    4             CALL TIME(BIT$CELL);
00023    4             END;
00024    3         OUTPUT (TTO) = 1;
00025    3         CALL TIME (BIT$CELL+BITCELL);
00026    3         /* AUTOMATIC RETURN IS GENERATED */
00027    3         END PRINT$CHAR;
00028    2
00029    2     PRINT$STRING: PROCEDURE(NAME,LENGTH);
00030    3         DECLARE NAME ADDRESS,
00031    3             (LENGTH,I,CHAR BASED NAME) BYTE;
00032    3             DO I = 0 TO LENGTH - 1;
00033    3             CALL PRINT$CHAR(CHAR(I));
00034    4             END;
00035    3         END PRINT$STRING;
00036
00037    2     PRINT$NUMBER: PROCEDURE(NUMBER,BASE,CHARS,ZERO$SUPPRESS);
00038    3         DECLARE NUMBER ADDRESS, (BASE,CHARS,ZERO$SUPPRESS,I,J) BYTE;
00039    3         DECLARE TEMP (16) BYTE;
00040    3         IF CHARS > LAST(TEMP) THEN CHARS = LAST(TEMP);
00041    3             DO I = 1 TO CHARS;
00042    3             J = NUMBER MOD BASE + '0';
00043    4             IF J > '9' THEN J = J + 7;
00044    4             IF ZERO$SUPPRESS AND I <> 0 AND NUMBER = 0 THEN
00045    4                 J = ' ';
00046    4             TEMP(LENGTH(TEMP)-I) = J;
00047    4             NUMBER = NUMBER / BASE;
00048    4             END;

00049    3         CALL PRINT$STRING(.TEMP + LENGTH(TEMP) - CHARS, CHARS);
00050    3         END PRINT$NUMBER;
00051    2
00052    2     DECLARE I ADDRESS,
00053    2         CRLF LITERALLY 'CR,LF',
00054    2         HEADING DATA (CRLF,LF,LF,
00055    2                                   TABLE OF SQUARE ROOTS', CRLF,LF,
00056    2           VALUE  ROOT VALUE  ROOT VALUE  ROOT VALUE  ROOT VALUE  ROOT
00057    2         CRLF,LF);
00058    2
00059    2         /* SILENCE TTY AND PRINT COMPUTED VALUES */
00060    2         OUTPUT(TTO) = 1;
00061    2         DO I = 1 TO 1000;
00062    2         IF I MOD 5 = 1 THEN
00063    3             DO; IF I MOD 250 = 1 THEN
00064    4                 CALL PRINT$STRING(.HEADING,LENGTH(HEADING));
00065    4             END; ELSE
00066    3         CALL PRINTSTRING(.(CR,LF),2);
00067    3         CALL PRINT$NUMBER(I,10,6,TRUE /* TRUE SUPPRESSES LEADING
                  ZEROES */);
00068    3         CALL PRINT$NUMBER(SQUARE$ROOT(I), 10,6, TRUE);
00069    3         END;
00070    2
00071    2     DECLARE MONITOR$USES (10) BYTE;
00072    2     EOF
NO PROGRAM  ERRORS
```

Fig. 8-3. Exemplu de program scris în PL/M.

o schemă detaliată în cazul logicii cablate. Cînd facem o modificare sau localizăm o problemă, putem lucra de sus în jos, urmărind structura și forma de arbore pînă la blocul ce solicită atenție. Prin opoziție programele nestructurate, sînt ca o schemă detaliată uriașă a *întregului sistem* și pretind trecerea unor munți de detalii pentru a ajunge la problemă.

```
                        LINE NUMBER - ADDRESS CORRESPONDENCE

   2=0800H    6=0803H    7=080AH    8=0810H    9=0833H   10=089DH
  11=08A9H   12=0881H   13=08B2H   16=08B5H   17=08BAH   18=08BCH
  19=08C5H   20=08C8H   21=08D2H   22=08D7H   23=08D8H   24=08E8H
  25=08EBH   26=08EFH   27=08F7H   28=08F8H   30=08FBH   31=0904H
  32=0907H   33=090DH   34=0923H   36=092DH   37=092EH   38=0931H
  39=0938H   40=093CH   41=093FH   42=0952H   43=096FH   44=0972H
  45=0999H   46=099DH   47=09AFH   48=09CCH   49=09D1H   51=09F5H
  52=09F6H
S00064 02553   115
    0DH 0AH 0AH 0AH 20H 20H 20H 20H 20H 20H 20H 20H 20H 20H 20H 20H
20H 20H 20H 20H 20H 20H 20H 20H 20H 20H 20H 54H 41H 42H 4CH 45H 20H 4FH
46H 20H 53H 51H 55H 41H 52H 45H 20H 52H 4FH 4FH 54H 53H 0DH 0AH 0AH 20H
56H 41H 4CH 55H 45H 20H 20H 52H 4FH 4FH 54H 20H 56H 41H 4CH 55H 45H 20H
20H 52H 4FH 4FH 54H 20H 56H 41H 4CH 55H 45H 20H 20H 52H 4FH 4FH 54H 20H
56H 41H 4CH 55H 45H 20H 20H 52H 4FH 4FH 54H 20H 56H 41H 4CH 55H 45H 20H
20H 52H 4FH 4FH 54H 0DH 0AH 0AH
  60=0A6CH   61=0A6FH   62=0A78H   63=0AA4H   64=0AC3H
  65=0AC6H   66=0ACFH
S00081 02773     2
    0DH 0AH
  67=0AD7H   68=0AF7H   69=0B09H   70=0B28H
 S00001 00BCCH  S00002 00BCDH  S00003 00BCEH  S00004 00BCFH
 S00005 00C00H  S00021 00BD0H  S00023 00BD2H  S00024 00BD4H
 S00029 00BD6H  S00031 00BD7H  S00039 00BD8H  S00040 00BDAH
 S00042 00BDBH  S00047 00BDCH  S00048 00BDFH  S00049 00BE0H
 S00050 00BE1H  S00052 00BE2H  S00053 00BE3H  S00054 00BE4H
 S00063 00BF4H  S00078 00BF6H  S00079 00BCAH  S00080 00BC8H
0000H HLT HLT HLT HLT HLT HLT HLT HLT HLT HLT HLT HLT HLT HLT HLT HLT
```

(a)

Fig. 8-4. *(a)* Corespondența număr-adresă.

Deoarece nu există o corespondență unu-la-unu între liniile de program și codul mașină produs, PL/M asigură imprimarea separată a codului obiect, așa cum se arată în figura 8-4. În cazul PL/M, verificarea programelor se efectuează prin verificarea *rezultatelor* enunțului sursă, cu aceeași abordare sus-jos folosită și în scrierea programelor. Într-un program structurat, „fluxul" urmează direct listarea, de la cap la coadă. Dacă un program ajunge într-un anumit punct, știm că tot codul ce precede punctul respectiv a fost executat iar restul nu. Inițial, se pot neglija detaliile în blocurile de nivel inferior. Verificarea constă deci în urmărirea rezultatelor corecte, întîi la un nivel mai general și apoi la nivele tot mai detaliate.

8.3. COMPILATOARE, SISTEME DE OPERARE ȘI MAȘINI VIRTUALE

Interschimbabilitatea dintre programe și logica cablată produce o problemă filozofică interesantă în sensul că noi lucrăm, aproape întotdeauna, nu cu o „mașină reală", ci mai mult cu o abstracție programată. Undeva, în interiorul fiecărui calculator sau microprocesor, există un anumit set de instrucțiuni pe care utilizatorul nu îl vede niciodată. La acest nivel, al *microinstrucțiunilor*, instrucțiunile sînt executate în hard-ul realizat în logică cablată.

```
                    GENERATED OBJECT CODE

0800H JMP,B2H,00H LHI,00H LLI,D0H LMB INL LMC DCL LBM INL LCM INL LMB
0810H INL LMC LLI,D0H LAM INL LCM ADI,01H LBA LAC ACI,00H ORA RAR LCA
0820H LAB RAR LLI,D4H LMA INL LMC LHI,08H LLI,D2H LAM INL LCM INL SUM

0830H INL LBA LAC SBM ORB JTZ,A9H,08H DCL LBM INL LCM LLI,D2H LMB INL
0840H LMC DCL LBM INL LCM LLI,C8H LMB INL LMC LLI,D0H LBM INL LCM LLI
0850H,CAH LMB INL LMC JMP,8AH,08H LEM DCL LDM LMI,11H LBI,00H LCB LAD
0860H RAL LDA LAE RAL LEM DCE LME LEA RTZ LAB RAL LBA LAC RAL LCA DCL
0870H DCL LAB SUM LBA INL LAC SBM LCA JFC,83H,08H DCL LAB ADM LBA INL
0880H LAC ACM LCA INL SBA SBI,80H JMP,5FH,08H CAL,57H,08H LAD LLI,D2H
0890H ADM INL LDA LAE ACM LEA LAD ADI,01H LDA LAE ACI,00H ORA RAR LEA
08A0H LAD RAR INL LMA INL LME JMP,27H,09H LHI,08H LLI,D2H LAM INL LCM
08B0H RET RET JMP,F8H,08H LHI,0BH LLI,D6H LMB XRA 010 LBI,5BH DCB JTZ
08C0H,C5H,08H JMP,BEH,08H INL LMI,00H LAI,07H LHI,08H LLI,D7H SUM JTC
08D0H,E8H,08H DCL LAM 010 LAM RRC LMA LBI,5BH DCB JTZ,E1H,08H JMP,DAH
08E0H,08H INL LBM INB LMB JMP,C8H,08H LAI,01H 010 LAI,5BH ADI,5BH LBA
08F0H DCB JTZ,F7H,08H JMP,F0H,08H RET JMP,2EH,09H LHI,0BH LLI,D8H LMB
0900H INL LMC INL LMD INL LMI,00H LHI,0BH LLI,DAH LBM DCB LAB INL SUM
0910H JTC,2DH,09H LAM LLI,D8H ADM INL LBA LAI,00H ACM LLB LHA LAM LBA
0920H CAL,B5H,08H LHI,0BH LLI,DBH LBM INB LMB JMP,07H,09H RET JMP,F6H

0930H,09H LHI,0BH LLI,E0H LMB INL LMD LAI,0FH DCL SUM JFC,41H,09H LMI
0940H,0FH LHI,0BH LLI,E2H LMI,01H LHI,0BH LLI,E0H LAM LLI,E2H SUM JTC
0950H,D9H,09H LLI,DFH LBM LLI,C8H LMB INL LMI,00H LLI,DCH LBM INL LCM
0960H LLI,CAH LMB INL LMC CAL,57H,08H LAB ADI,30H LBA LAC ACI,00H LLI
0970H,E3H LMB LAI,30H SUM JFC,7CH,09H LAM ADI,07H LMA LHI,0BH LLI,E2H
0980H LAM SUI,00H ADI,FFH SBA DCL NDM LLI,DCH LBA LAM INL LDM SUI,00H
0990H LCA LAD SBI,00H ORC SUI,01H SBA NDB RRC JFC,A1H,09H LLI,E3H LMI
09A0H,20H LAI,10H LHI,0BH LLI,E2H SUM LLI,E4H ADL LBA LAH ACI,00H DCL
09B0H LDM LLB LHA LMD LHI,0BH LLI,DFH LBM LLI,C8H LMB INL LMI,00H LLI
09C0H,DCH LBM INL LCM LLI,CAH LMB INL LMC CAL,57H,08H LLI,DCH LMD INL
09D0H LME LLI,E2H LBM INB LMB JMP,47H,09H LHI,0BH LLI,E4H LCH LAL ADI
09E0H,10H LBA LAC ACI,00H LCA LAB LLI,E0H SUM LBA LAC SBI,00H LLI,E0H
09F0H LDM LCA CAL,FBH,08H RET JMP,6CH,0AH R01 RRC RRC RRC INE INE INE
0A00H INE INE INE INE INE INE INE INE INE INE INE INE INE INE INE INE
0A10H INE INE INE INE INE JMP,41H,42H JMP,45H,20H I07 CAL,20H,53H 008
0A20H 010 I00 CFS,45H,20H CFS,4FH,4FH JMP,53H,0DH RRC RRC INE CAL,41H
0A30H,4CH 010 I02 INE INE CFS,4FH,4FH JMP,20H,56H I00 JMP,55H,45H INE
0A40H INE CFS,4FH,4FH JMP,20H,56H I00 JMP,55H,45H INE INE CFS,4FH,4FH
0A50H JMP,20H,56H I00 JMP,55H,45H INE INE CFS,4FH,4FH JMP,20H,56H I00
0A60H JMP,55H,45H INE INE CFS,4FH,4FH JMP,0DH,0AH RRC LAI,01H 010 LHI
0A70H,0BH LLI,F4H LMI,01H INL LMI,00H LAI,E8H LCI,03H LHI,0BH LLI,F4H
0A80H SUM INL LBA LAC DBM JTC,28H,0BH LLI,C8H LMI,05H INL LMI,00H LLI
0A90H,F4H LBM INL LCM LLI,CAH LMB INL LMC CAL,57H,08H LAB SUI,01H LBA
0AA0H LAC SBI,00H ORB JFZ,D2H,0AH LLI,C8H LMI,FAH INL LMI,00H LLI,F4H
0AB0H LBM INL LCM LLI,CAH LMB INL LMC CAL,57H,08H LAB SUI,01H LBA LAC
0AC0H SBI,00H ORB JFZ,CFH,0AH LBI,F9H LCI,09H LDI,73H CAL,FBH,08H JMP
0AD0H,E0H,0AH JMP,D7H,0AH R01 RRC LBI,D5H LCI,0AH LDI,02H CAL,FBH,08H
0AE0H LHI,0BH LLI,F4H LBM INL LCM LLI,DCH LMB INL LMC LLI,DFH LMI,0AH
0AF0H LBI,06H LDI,01H CAL,31H,09H LHI,0BH LLI,F4H LBM INL LCM CAL,03H
0B00H,08H LHI,0BH LLI,DCH LMA INL LMI,00H LLI,DFH LMI,0AH LBI,06H LDI
0B10H,01H CAL,31H,09H LHI,0BH LLI,F4H LAM INL LCM ADI,01H LBA LAC ACI
0B20H,00H DCL LMB INL LMA JMP,78H,0AH HLT
```

(b)

Fig. 8-4. *(b)* Codul obiect generat.

Setul de instrucțiuni pe care îl denumim în mod normal „limbajul mașină" (vezi, de exemplu, figurile 7-3, 7-5 și 7-6) este în realitate o *mașină virtuală* creată de microprogramele din procesor. Deoarece este incomod să lucrăm cu limbajul mașină, am creat un alt nivel de abstractizare, denumit limbajul de asamblare. Cînd scriem și verificăm programe în limbajul de asamblare putem, în principiu să ignorăm existența limbajului mașină. Într-un anumit sens, lucrăm cu o mașină abstractă ce execută instrucțiuni în limbajul de asamblare. În cazul limbajelor evoluate, ca PL/M, este adevărat același lucru : scriem și depanăm programele în limbajul respectiv, de parcă am dispune de o mașină ce l-ar executa.

Atît în cazul asambloarelor cît și în cel al compilatoarelor, apare o etapă suplimentară între scrierea programului și rularea sa efectivă : programul

sursă trebuie să fie asamblat sau completat în program obiect, înainte de a fi rulat. Această etapă suplimentară devine foarte deranjantă în cursul verificării de program, deoarece atunci cînd se găsește o greșeală este adesea necesar să reasamblăm sau să recompilăm doar pentru a verifica soluția. Această întîrziere poate să devină importantă dacă există un număr mare de erori.

O altă abordare a limbajelor evoluate constă în utilizarea unui program denumit *interpretor* pentru a executa comenzile în limbajul evoluat în mod direct *la momentul execuției*. Programatorul poate astfel să introducă programe sau modificări și să le execute imediat fără să fie obligat să facă mai întîi o asamblare sau o compilare. Aceasta accelerează foarte mult verificarea deoarece ea devine un proces *interactiv*. BASIC și FORTH sînt exemple de astfel de programe interpretative.

Tot astfel cum setul de instrucțiuni din limbajul mașină este implementat de către programe scrise în microinstrucțiuni, interpretorul implementează comenzile din limbajul evoluat cu programe executate în limbaj mașină. Se crează astfel o mașină virtuală de nivel superior, ce execută direct programele scrise în limbajul de nivel evoluat.

Depanarea de programe este mult mai rapidă, în cazul limbajelor interpretative, deoarece nu se impune o reasamblare sau o recompilare de fiecare dată cînd se operează o modificare. Dezavantajul lor este că viteza de execuție este, în general, mult mai redusă deoarece fiecare comandă trebuie să fie interpretată *în momentul execuției*. Traducînd înainte programele, asambloarele și compilatoarele evită pierderea de timp în momentul execuției.

O altă noțiune abstractă folosită în programare este *sistemul de operare*. Creind o mașină virtuală ce este mai ușor de utilizat decît mașina pe care se lucrează, sistemele de operare simplifică programarea. De exemplu, un sistem de operare poate permite programatorului să programeze în mod separat mai multe sarcini, ca și cum fiecare ar avea repartizat un procesor separat. Sistemul de operare planifică rularea acestor sarcini astfel încît ele ajung să folosească, fiecare, mașina pe un anumit interval de timp. Resursele comune, cum sînt echipamentele de intrare/ieșire, sînt planificate în mod automat astfel încît fiecare sarcină să le poată folosi fără a se preocupa de celelalte. De asemenea, sarcinile pot să-și transmită, în mod reciproc mesaje.

La orice moment, fiecare sarcină este fie *în rulare, gata de prelucrare sau în așteptare*. Fiecărei sarcini îi revine o prioritate ce determină importanța sa relativă în sistem. Sistemul de operare execută întotdeauna sarcina de cea mai mare prioritate ce este gata de prelucrare. *Dintr-o singură mașină reală, sistemul de operare crează astfel un număr de mașini virtuale*. El simplifică mult programarea, preocupîndu-se de toate detaliile sarcinilor de rutină, cum sînt comunicarea cu dispozitivele de intrare/ieșire și răspunsul la întreruperi.

Și de această dată, prețul plătit este timpul de execuție. Fiecare nivel de abstractizare în programare este analog unui „intermediar" ce face lucrurile să fie mai ușoare, dar își extrage profitul în viteza de programare.

Într-un anumit sens, un microcomputer ce este programat pentru o aplicație particulară este o mașină virtuală ce execută comenzi într-un limbaj extrem de evoluat. De exemplu, apăsarea unei taste pe un terminal cu tub catodic cu microcomputer, poate fi concepută ca o comandă de nivel superior. Așa cum se arată în tabelul 8-4, programul de aplicații poate fi izolat prin (pînă la patru) nivele de abstractizare de logica cablată din procesor. Tot astfel după cum utilizatorul nu este, în general, în cunoștință de cauză cu

Tabelul 8-4. **Nivele de programare**

↑	Nivelul programului	Tradus de un	Stocat în
Nivel mai înalt	Intrările sistemului	Program de aplicații	Memoria principală
	Limbaj evoluat de programare	Compilator, interpretor, sistem de operare	Memoria principală
	Limbaj de asamblare	Asamblor	Memoria principală
	Limbaj mașină	Microprogram	Memoria rapidă internă
	Microinstrucțiuni	Logică cablată	Interconexiuni de porți și registre

funcționarea internă, programatorii pot, în general, să ignore nivelele inferioare celui în care lucrează. La fiecare nivel, lucrăm în mod efectiv pe o mașină virtuală și ignorăm toate nivelele inferioare.

8.4. MICROPROGRAMAREA

Deoarece am arătat că logica programată poate substitui în mod direct logica cablată, cu o economie importantă, nu este surprinzător faptul că *logica tuturor calculatoarelor moderne este realizată intern prin program.* Desigur, trebuie să plasăm undeva o linie de demarcație, fiindcă pe măsură ce înlocuim logica cablată cu programe, produsul rezultant devine tot mai lent. Deoarece circuitele logice ieftine pot lucra la o frecvență de tact de 10 ori mai mare decît a componentelor de memorie principală de preț moderat, putem realiza un compromis avantajos, cu o reducere neglijabilă a vitezei, asamblînd numai logica cablată absolut necesară execuției instrucțiunii în 5...10 pași.

În cazul abordării microprogramate tehnicile de proiectare logică convenționale sînt folosite pentru a proiecta un „calculator" intern foarte rapid, simplu din punct de vedere logic, ce adresează numai registre. *Microprogramul* calculatorului intern este stocat într-o memorie redusă (100...1 000 de cuvinte) de mare viteză denumită *„memoria de control"* (de tip ROM). Microprogramul citește *macroinstrucțiunile* din memoria principală, de mare capacitate, și realizează toate funcțiile impuse de setul de (macro) instrucțiuni al *mașinii virtuale* („calculatorul extern"). Mașina virtuală se comportă din toate punctele de vedere ca un calculator cu un set particular de (macro) instrucțiuni și moduri de adresare, determinate de către memoria de control. Acest set de instrucțiuni nu are asemănări cu *setul de microinstrucțiuni* ce este realizat în logica convențională. Într-un anumit sens, putem concepe microinstrucțiunile ca fiind componente ce sînt conectate de către microprogram pentru a realiza setul de instrucțiuni.

Ca un exemplu real, să urmărim setul de microinstrucțiuni folosit de către National Semiconductor centru a implementa setul de instrucțiuni al microprocesorului 8900, (vezi fig. 7-2) denumit, în versiunea bitslice, GPC/P. Se poate produce o mașină virtuală cu setul de instrucțiuni al lui 8900, printr-un anumit microprogram, rulat pe „computerul intern" GPC/P. Cuvîntul de microinstrucțiuni intern al GPC/P are o lungime de *24 de biți* și nu prezintă similitudini cu formatul macroinstrucțiunilor din figura 7-3. Așa cum arată figura 8-5, cuvîntul de 24 biți ai microinstrucțiunii este divizat în mai multe cîmpuri codate. Fiecare pas de microprogram poate combina logic sau arit-

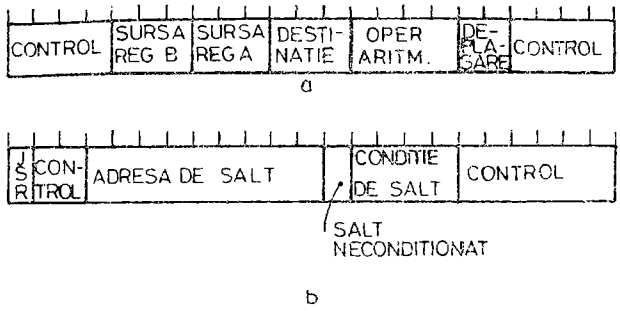


Fig. 8-5. Formatul microinstrucțiunilor pentru GPC/P : *(a)* instrucțiuni aritmetice ; *(b)* instrucțiuni de salt în microprogram.

metic conținutul a două *registre sursă* (*A* și *B*) și să depună rezultatul într-un *registru destinație*. Operația ce trebuie îndeplinită este definită în cîmpul de patru biți *Arith Operation*. Doi dintre biții acestui cîmp selectează operația : AND, OR, Exclusive OR sau ADD. Ceilalți doi determină dacă registrul sursă va fi completat și dacă va fi folosită o intrare de transport. Un alt cîmp controlează deplasările, iar biții ce rămîn formatul cuvîntului, setarea codului de condiție ș.a.m.d. Pentru salturi de microprogram, deplasări și intrare-ieșire se folosesc alte formate. Pentru a implementa mai direct funcțiile sistemului, se folosesc *microcontrolere*, ce au seturi de instrucțiuni similare acestora, avînd în vedere că în anumite aplicații acest format al instrucțiunilor este mai eficient.

Deși computerul *virtual* 8900 are numai patru registre de lucru, computerul intern ce execută microinstrucțiunile are opt (vezi fig. 8-6). Patru din

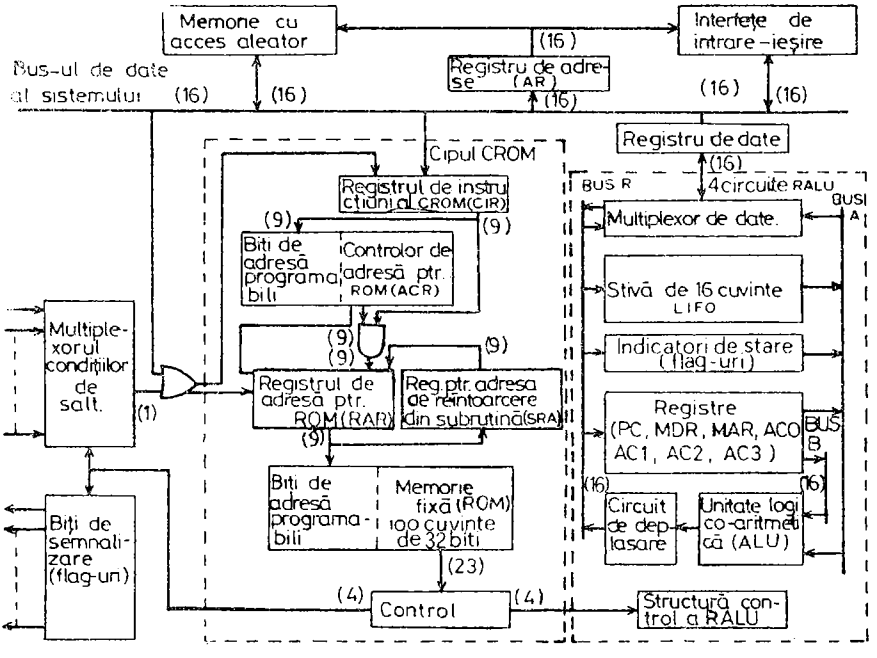


Fig. 8-6. Schema bloc a sistemului GPC/P.

este folosită pentru eliminarea biților indiferenți („dont care") ai macroinstrucțiunii. De exemplu, formatul din figura 7-3 *a*, impune ca legătură spre subrutina adecvată numai biții 12...15 (dacă s-ar include și biții 10 și 11, s-ar putea specifica o altă adresă de start în microprogram, pentru fiecare din cele patru registre posibile). Dimpotrivă formatul din figura 7-3 *b* impune ca biții 10...15 să fie utilizați ca un indicator. Fiecare cuvînt din PLA-ul de control al adreselor reprezintă o mască diferită, corespunzătoare altui format al macroinstrucțiunilor.

Deși această mascare, potrivit cu formatul, reduce numărul adreselor de start ale microprogramului indicate de către cei nouă biți ai codului operației, ea nu schimbă faptul că ele sînt împrăștiate peste $2^9 = 512$ locații diferite. Folosind un PLA pentru însăși memoria de control, se au în vedere numai acele cuvinte ce conțin cuvintele de microprogram utile. Rezultatele acestei metode sînt destul de impresionante, în sensul că pentru a realiza întregul set de instrucțiuni al lui 8900 (din figura 7-3) sînt necesare *numai 100 de cuvinte de microprogram de 23 de biți*. Un set de instrucțiuni atît de complex ar solicita în ROM cel puțin 256 de cuvinte pentru a realiza aceeași funcție. Cînd pentru stocarea microprogramului se folosesc ROM-uri, în locul mascării biților macroinstrucțiunii, pentru generarea adresei efective de microprogram din codurile de operație, *se folosește un ROM sau un PLA ce generează adresele de start*. În mod ideal, putem considera fiecare cuvînt de microprogram ca o microoperație, în sensul că biții de ieșire produși declanșează o anumită combinație de operații (de exemplu, incrementarea contorului de program). Realizarea macroinstrucțiunilor este simplă, din acest punct de vedere, ea constînd din parcurgerea unei anumite secvențe compusă din aceste microoperații.

Varianta microprogramată este folosită de asemenea, pentru a implementa o logică de foarte mare viteză de uz general. Pentru implementarea procesoarelor de mare viteză speciale cu cuvinte de orice lungime pot fi utilizate circuite LSI *bit-slice* (de exemplu 2901). De exemplu, un procesor de 86 de biți poate fi realizat prin conectarea a 9 „felii" a cîte 4 biți cu un ROM de control. Microprocesoarele ce au seturi de instrucțiuni la nivelul micro sînt denumite în mod general *microcontrolere*. Deși sînt mai dificil de programat, ele pot atinge o viteză foarte mare în execuția sarcinilor logice simple, fiindcă nu „achită penalizările" impuse de nivelele mai înalte de abstractizare.

8.5. PROGRAMAREA ÎNTRERUPERILOR

Sarcina majorității sistemelor digitale este producerea unor semnale de ieșire drept răspuns la semnalele de intrare recepționate. În sistemele foarte simple, programul poate să urmărească o buclă de testare a unor intrări, așteptînd apariția unui fenomen (de exemplu apăsarea unui buton). Deoarece în timp ce se parcurge o astfel de buclă nu pot fi realizate alte funcții utile, această tehnică conduce la risipă. *Întreruperile* permit dispozitivelor de intrare și ieșire să întrerupă un program ce realizează între timp o funcție utilă, doar în momentul în care dispozitivul este gata (ready).

În felul acesta, procesorul poate să *rezolve* cererile dispozitivelor de intrare/ieșire numai cînd ele sînt activate, în restul timpului realizînd o funcție utilă (și nu numai una de așteptare).

O întrerupere produce salvarea conținutului numărătorului de program în stivă și forțează setarea sa la o locație specificată. Mai înainte ca aceasta să aibă

loc, se permite instrucțiunii ce se execută în momentul în care apare întreruperea să-și completeze ciclul în mod normal. Locația la care este setat numărătorul de program se numește *locația de întrerupere* și conține, de obicei, un salt necondiționat la o *rutină de rezolvare a întreruperii*. Această rutină îndeplinește funcția dorită (de obicei o intrare și/sau o ieșire, plus prelucrarea asociată) și apoi readuce controlul programului în programul principal, prin execuția unei instrucțiuni RETURN (vezi fig. 8-8 *a*). Această instrucțiune extrage din stivă conținutul salvat anterior al numărătorului de program și îl reintroduce în numărătorul de program. Astfel, ultima instrucțiune a tuturor instrucțiunilor de rezolvare a întreruperilor este, întotdeauna, RETURN.

Folosirea întreruperilor generează o *întreagă serie de probleme*. Rutina de rezolvare a întreruperilor trebuie să fie scrisă astfel încît nimic să nu se modifice în momentul în care revenim în programul principal. Aceasta înseamnă în mod normal că rutina de rezolvare a întreruperilor trebuie să înceapă prin *salvarea conținutului tuturor registrelor* pe care le va folosi, inclusiv a stării biților de condiție (transport, zero, semn, paritate, etc.). La sfîrșitul

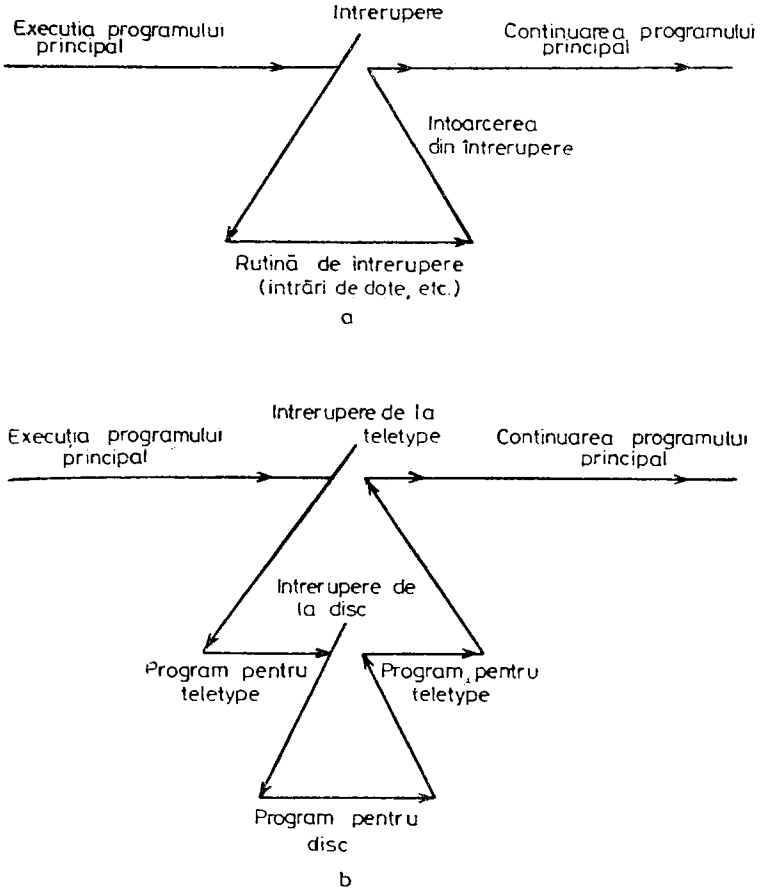


Fig. 8-8. *(a)* Program cu o singură întrerupere. *(b)* Două nivele de întrerupere (întreruperea rutinei de întrerupere).

rutinei de întrerupere, înaintea întoarcerii în programul originar, se reface starea tuturor registrelor și indicatorilor.

Stiva este ideală pentru această funcție, deoarece registrele și indicatorii pot fi salvați și refăcuți într-o ordine fixată (refacerea se face în ordine inversă față de salvare).

Această salvare și refacere a registrelor poate fi eliminată dacă toate registrele folosite în rutina de întrerupere sînt *rezervate* pentru acest scop și nu sînt folosite niciodată în alte părți ale programului. Chiar și în acest caz, salvarea și refacerea registrelor este totuși necesară : dacă un program este întrerupt între o operație aritmetică și una de salt condiționat ce ia în considerare starea biților de condiție generați de către operația aritmetică respectivă, programul *va face un salt greșit* dacă biții de condiție sînt alterați de către o rutină de întrerupere ce a intervenit între cele două operații. Notați că și subrutinele pot produce aceleași probleme, dar este mai ușor să evităm problemele în acest caz deoarece chemăm o subrutină în mod intenționat. Pe de altă parte, o întrerupere poate să apară la *orice punct* în program.

În cazul folosirii întreruperilor, programarea trebuie făcută cu mare atenție, deoarece greșelile pot produce dificultăți numai cînd întreruperea apare la o anumită locație și cu anumite combinații de date. *Eroarea de programare poate astfel să apară doar o dată pe lună sau chiar pe an*, cînd apare combinația în cauză. Dacă, de exemplu, uităm să salvăm biții de condiție, nu vor apare probleme pînă cînd întreruperea va apare între o operație aritmetică și o ramificație condiționată bazată pe rezultatele operației respective. Chiar și în acest caz, nu vor fi probleme dacă ultima operație aritmetică din rutina de întrerupere produce același rezultat (carry, zero, etc) ca și programul principal. Problema se va pune astfel în evidență cu totul întîmplător, cînd întreruperea apare în anumite părți-cheie ale programului și biții de condiție rezultați sînt diferiți. Pentru a face lucrurile și mai complicate, simptomele pot fi diferite de fiecare dată cînd apare comportamentul eronat. Singura soluție este să fim foarte atenți cînd scriem programele de întrerupere. În cazul întreruperilor mai sînt posibile multe alte greșeli de programare. Dacă de exemplu, rutina de întrerupere modifică o locație de memorie ce este, de asemenea folosită de către programul principal, vor apare probleme la reluarea acestuia din urmă. Aceasta se poate întîmpla dacă o subrutină distribuită între mai mulți utilizatori este întreruptă în timp ce este folosită de către programul *A* și folosită ulterior de către programul *B*. Dacă subrutina folosește locații de memorie drept contoare sau pentru stocare temporară, valorile originale vor fi pierdute cînd întreruperea se încheie. Dacă subrutinele urmează să fie împărțite între programe diferite, ele trebuie să fie *reentrante*, ceea ce înseamnă *eliminarea folosirii* RAM *în mod distribuit*. Registrele se pot folosi pentru stocare temporară de date (dacă sînt salvate și refăcute) dar dacă în acest scop, se folosesc locații de memorie, zonele repartizate fiecărui utilizator vor fi separate.

Cu excepția cazului în care un microcomputer dispune de mai mult de cîteva registre de lucru (de exemplu, TI 990 are 16), este dificil să se scrie cod reentrant. Unele seturi de instrucțiuni (de exemplu, cele pentru 8086) furnizează *registre bază* care pot fi utilizate pentru a decala locațiile de stocare temporară în zone destinate pentru fiecare utilizator sau program ce cheamă o subrutină. În mașini ca 8080, se poate folosi pentru o stocare temporară suplimentară stiva, fără a sacrifica condiția de reentranță. Cînd întreruperile sînt posibile, nu putem folosi buclele de întîrziere, cum sînt cele ce apar în programul de receptor serie din figura 8-1, deoarece durata întîrzierii va fi

mărită cu timpul de execuţie al rutinei de întrerupere. Cînd folosim întreruperile, putem degaja programul de sarcina buclelor de temporizare, folosind întreruperea generată de către un circuit de temporizare extern la sfîrşitul perioadei ce indică sfîrşitul duratei unui bit.

Pentru a controla toate operaţiunile dependente de timp dintr-un sistem, se foloseşte adesea un ceas de **timp real.** De exemplu, se poate genera o întrerupere la fiecare 20 ms de către reţeaua de alimentare (50 Hz). Rutina de întrerupere corespunzătoare poate ţine evidenţa ceasului de timp real într-o locaţie de memorie sau într-un registru ce poate fi folosit pentru temporizarea diverselor operaţii. Duratele de timp pot fi determinate prin simple diferenţe între valoarea *iniţială* şi cea *curentă* a contorului.

Uneori, unele operaţii trebuie să fie făcute periodic, la fiecare întrerupere de ceas sau poate la fiecare 10 întreruperi de ceas. Rutina de întreruperi de ceas de timp real poate ţine cont de aceste solicitări, executînd sarcinile la momentul adecvat. Uneori în sistem sînt necesare timpul absolut sau data. Ele pot fi introduse iniţial prin *tastatură* şi actualizate de către rutina de întrerupere de ceas. Precizia pe termen lung a acestui tip de ceas este excelentă, avînd în vedere că în centralele electrice se urmăreşte numărul total de perioade ale tensiunii reţelei pe zi, pentru a asigura corectitudinea indicaţiilor ceasurilor electrice.

O altă problemă potenţială poate apare în cazul întreruperilor cînd au loc operaţii ce trebuie să se desfăşoare în mare viteză odată ce sînt declanşate. De exemplu, dacă scriem o înregistrare pe disc sau pe bandă magnetică, o întrerupere, în timpul acestui transfer ar fi dezastruoasă. Pentru a preveni acest tip de problemă, se foloseşte un bistabil de *activare a întreruperilor* (ce este, de obicei, conţinut în microprocesor). Înainte de a începe o operaţie ce nu poate fi întreruptă, se dezactivează întreruperile cu o instrucţiune specială. Cînd operaţiunea este încheiată, se reactivează întreruperile.

Sistemele mai complexe utilizează *întreruperile cu priorităţi,* pentru a efectua diferite funcţii pe baza unor priorităţi. Funcţiilor cele mai critice în timp li se alocă prioritatea cea mai mare, astfel încît să poată întrerupe o rutină de întreruperi avînd o prioritate mai redusă, aşa cum se arată în figura 8-8 *b*. În timpul execuţiei întreruperii de nivel superior de prioritate se dezactivează întreruperile de ordin inferior, ce sînt reactivate la sfîrşitul rutinei. Întreruperile cu priorităţi pot fi folosite pentru a asigura transferul prin *separatoarele* (buffer-ele) de intrare şi ieşire ale memoriei principale. Programul principal îşi preia datele sale de intrare de la o zonă de separator (buffer) din memorie. Acest buffer poate fi scris în mod automat de către rutina de întrerupere, de fiecare dată cînd apare o întrerupere. Cele două procese — scriere în zona tampon şi preluarea de date din ea, pentru prelucrare pot decurge independent. Pentru a ţine evidenţa adresei următoare la care trebuie să scriem sau din care trebuie să citim datele, se poate folosi un registru sau o locaţie de memorie. În mod similar, rutinele de întrerupere de ieşire pot citi datele din buffer-ele din memorie, independent de programul principal, ori de cîte ori un dispozitiv de ieşire este gata.

Metoda de rezolvare a întreruperilor variază mult de la un microcomputer la altul. În general, apariţia unei întreruperi cauzează execuţia unei instrucţiuni microprogramate ce citeşte un indicator de adresă dintr-o locaţie de ROM particulară şi transferă programul la locaţia respectivă. Dacă procesorul posedă *întreruperi dirijate (vectorizate),* pentru fiecare nivel de întrerupere există un indicator de adresă (vector) diferit, astfel încît controlul este transferat imediat rutinei de întrerupere corespunzătoare. Dacă în cazul unei maşini

ce nu posedă întreruperi dirijate există întreruperi multiple, rutina unică de întrerupere trebuie să testeze diferitele dispozitive de intrare pentru a găsi spre care rutină de intrare trebuie să se producă ramificația.

Majoritatea microcalculatoarelor fac mai mult decît ramificație la locația de început indicată. În general, ele dezactivează întreruperile și execută aceeași instrucțiune ce este folosită în mod normal pentru a *chema* subrutinele. Această instrucțiune salvează în stivă contorul de program, cuvîntul de stare a programului și alte registre de lucru.

Timpul necesar pentru a răspunde la o întrerupere, inclusiv cel necesar pentru salvarea registrelor, este denumit *timp de latență al întreruperii*. În multe aplicații, el poate reprezenta o parte semnificativă din timpul de execuție al celor mai critice segmente de program, din punct de vedere al timpului. Adesea, cîteva circuite adiționale de separare externă pot elibera în mod remarcabil procesorul, reducînd frecvența întreruperilor.

EXERCIȚII

1. (*a*) Construiți organigrama unei rutine de întîrziere, sub forma unei „bucle într-o buclă", ce generează o întîrziere de 3 s cu ajutorul microprocesorului 8008.
   (*b*) Scrieți programul pentru buclă utilizînd limbajul de asamblare.
   (*c*) Construiți listingul programului, utilizînd adrese hexazecimale și codul obiect, ca în Tabelul 8-3.
   (*d*) Scrieți programul în limbajul de asamblare al microprocesorului 8900. Presupuneți că timpii de execuție ai instrucțiunilor sînt: AISZ fără salt = 5,6 μs, JMP = 4,2 μs.
   (*e*) Scrieți în PL/M programul pentru întîrzierea de 3 s.
2. Care este dimensiunea permisă a spațiului de adresare pentru instrucțiunile INP și OUT:
   (*a*) Pentru microprocesorul INTEL 8008?
   (*b*) Pentru microprocesorul INTEL 8080?
3. Scrieți cu o *singură* declarație în limbaj de asamblare următoarele coduri obiect pentru microprocesorul 8008 (folosiți Tabelul 8-1):
   (*a*) 00000010
   (*b*) 11000111
   (*c*) 00001100
   00010001
   (*d*) 01001100
   00010110
   00000000
4. (*a*) Scrieți utilizînd limbajul de asamblare MCS-8 un program pentru tipărirea cifrelor hexazecimale. Intrarea în acest tabel va fi *etichetată* cu HEXTAB.
   (*b*) Construiți organigrama programului de generare a caracterelor codului ASCII către portul 3. Se va emite întîi caracterul corespunzător biților 7, 6, 5 și 4 din registrul A iar apoi se va emite caracterul corespunzător biților 3, 2, 1 și 0.
   (*c*) Scrieți programul în limbajul de asamblare MCS-8.
   (*d*) Scrieți același program în PL/M.
5. Generați listarea de asamblor a următorului program, conform Tabelului 8-3.

```
; AVERAGING PROGRAM: OUTPUT 0 = AVG OF INP 1&2
START ORG 1000H
      INP  1000H
      MOV 1
      INP  B, A
      ADD 2
      RRC B
      OUT          (A + B)/2 = AVERAGE
      END 0
```

6. Rescrieți programul din Tabelul 8-2 pentru un caracter de opt biți cu o durată de 10 ms pe bit. Faceți ca programul să pornească de la adresa 2000 și chemați subrutina de întîrziere la adresa 100, în loc de a folosi două bucle de întîrziere în program. Scrieți subrutina de întîrziere astfel ca să fie asamblată în același timp.

7. (*a*) Scrieți programul asociat organigramei din figura 7-11 utilizînd asamblorul MCS-8. Presupuneți că datele (ABCDEFGH) sînt disponibile de la portul 3 ; dacă funcția este adevărată se va genera saltul la locația etichetată cu CONTROL. Programul va bucla atît timp cît funcția este adevărată.

   (*b*) Scrieți un program identic pentru microprocesorul 8900.

   (*c*) Scrieți un program identic pentru PL/M.

8. (*a*) Scrieți în limbajul de asamblare MCS-8 programul ce emite prin portul 2 caracterele ASCII corespunzătoare mesajului DATA INCORRECT. Programul va începe la locația 1100 și se va încheia cu instrucțiunea *Return*.

   (*b*) Scrieți două declarații în PL/M care să facă același lucru.

9. Definiți un MACRO numit DELAY în limbajul de asamblare MCS-8. Acest MACRO permite generarea unei întîrzieri cuprinse între o milisecundă și 256 ms, specificată în cîmpul operandului. Spre exemplu, DELAY 22 va produce o întîrziere de 22 ms.

10. Definiți în limbajul de asamblare MCS-8 MACRO-ul „RARN" ce generează rotirea registrului *A* cu una pînă la șapte poziții în funcție de cîmpul operandului.

11. Scrieți în limbajul de asamblare MCS-8 un program de testare a memoriilor ROM, ce permite compararea conținutului unui ROM corect programat implantat între adresele 1024 și 1380 cu un ROM supus testării, montat într-un soclu, între adresele 1381 și 1637. Dacă apare o eroare, programul se oprește reținînd în registrul *A* datele corecte, în registrul *B* datele incorecte și în registrul *L* adresa la care s-a notificat eroarea.

12. Scrieți în limbajul de asamblare MCS-8 un program ce urmărește continuu modificările apărute în 64 de puncte de test. Cele 64 de semnale sînt conectate cîte 8 la un port de 8 biți. Dacă apare o modificare se va genera 1 pe ieșirea corespunzătoare. Ieșirile sînt similar grupate cîte 8 pe 8 porturi distincte. Utilizați locațiile de memorie de la 256 la 264 pentru a menține valoarea cu care se face compararea.

13. Scrieți în limbajul de asamblare MCS-8 un program ce va fi apelat după cel descris în Tabelul 8-2 pentru a stoca la adresele 1024...1104 (în ordine inversă) o transmisie de 80 de caractere. În loc de a folosi instrucțiunea HALT și întreruperea pentru a aștepta bitul de start, scrieți o buclă de așteptare. Dacă nu este detectat bitul de start timp de 100 ms se resetează numărul de caractere. (Se menține, în acest fel, numărul de caractere, al buclei de așteptare, în „sincronism" cu șirul de caractere). Cînd cel de-al 80-lea caracter este recepționat, se sare la locația 2000.

14. (*a*) Scrieți în limbajul de asamblare al microprocesorului 80 80 un program ce adună conținutul locațiilor de memorie 100 pînă la 127 cu conținutul locațiilor 128 pînă la 155 și depuneți rezultatele la adresele 156 pînă la 182. (Folosiți pentru aceasta trei numărătoare de indexare simulate prin program).

   (*b*) Scrieți același program pentru 8900.

   (*c*) Scrieți programul în limbajul PL/M.

15. Rescrieți programul de la exercițiul 5 în PL/M.

16. Rescrieți programul de recepție serială din Tabelul 8-2 în limbajul de asamblare a microprocesorului 8900. Comparați numărul de instrucțiuni necesare și numărul de octeți necesari stocării programului (vezi exercițiul 1 pentru întîrzieri).

17. (*a*) Scrieți un program de diagnosticare a memoriei, în limbajul de asamblare MCS-8, ce ciclează continuu și trece în Halt cînd este detectată o eroare. În fiecare locație de memorie de la 4 096 la 3 120 se va scrie cîte un cuvînt ce conține cîte un singur bit 1 în cele nouă poziții posibile. Se vor executa citiri, pentru verificare, de la fiecare adresă cuprinsă între 4096 și 3120 de cîte nouă ori pentru a nu rămîne nici o poziție de bit neverificată.

   (*b*) Scrieți același program utilizînd instrucțiunile microprocesorului 8080.

   (*c*) Scrieți același program în PL/M.

   (*d*) Scrieți programul pentru microprocesorul 8900 (modificați programul pentru a testa locații de memorie ce conțin 17 biți).

18. *(a)* Modificaţi macroinstrucţiunea ADD, descrisă în Secţiunea 8—4, astfel încît să fie realizată în opt paşi în loc de patru.
    (b) Ce efect va avea această modificare asupra complexităţii logice a calculatorului ?
    (c) Dar asupra cerinţelor legate de controlul memoriei ?
    (d) Dar asupra timpului de execuţie a instrucţiunii ADD presupunînd că timpul necesar pentru o microinstrucţiune nu se modifică ?
19. *(a)* Scrieţi o rutină de întrerupere pentru microprocesorul 8080, ce începe la locaţia 8 şi salvează în stivă (*push*) toate registrele şi indicatorii (flag-urile) şi cheamă (*Call*) o subrutină de la locaţia E1F. Cînd subrutina se încheie programul readuce (*Pull*) din stivă mărimile salvate şi se reîntoarce (*Return*) în programul principal.
    (b) Scrieţi acelaşi program pentru microprocesorul 8900.
20. *(a)* Scrieţi o rutină de întreruperi, similară celei din problema anterioară, pentru microprocesorul 8080. Programul va diferi prin faptul că va fi chemată o subrutină, din opt disponibile, în funcţie de codul dat de biţii ce mai semnificativi citiţi de la portul 0 (unul din opt disponibile). Adresele de start disponibile sînt stocate succesiv la adresele 8 pînă la 15.
    (b) Scrieţi o rutină similară pentru microprocesorul 8900.
    (c) Comparaţi numărul de instrucţiuni utilizate la punctele *(a)* şi *(b)*.
21. Ce se va întîmpla dacă apare o întrerupere în timpul rulării programului descris în Tabelul 8-2 iar rutina de întrerupere a setat bitul de condiţie în stare *not zero* :
    (a) Dacă întreruperea apare cînd se rulează instrucţiunea de la locaţia 5 ?
    (b) Dacă întreruperea apare cînd este executată instrucţiunea de la adresa 15 (în hexazecimal şi primii trei biţi ai cuvîntului au fost deja recepţionaţi ?
    (c) Dacă întreruperea apare cînd este executată instrucţiunea de la adresa 15 (în hexazecimal) şi al cincelea bit al cuvîntului a fost deja recepţionat ?
22. Ce se va întîmpla dacă programul descris în Tabelul 8-2 este întrerupt la locaţia 15 (în hexazecimal) iar conţinutul registrului *C* este (ca urmare a unei rutine de întrerupere incorect scrise) accidental încărcat cu FF.

BIBLIOGRAFIE

Tehnici de programare

Adams şi Haden, *Computers*, Wiley, New York, 1973. (Capitolele 1...6 acoperă principiile generale ale programării).

Booth, Taylor, *Digital Networks and Computer Systems*, Wiley, New York, 1971. (Capitolul XI, „Assembler Languages", pag. 391., Programming Languages and Compilers).

Spencer, D. D., *Computers and Programming Guide for Engineers*, Howard W. Sams, Indianapolis, 1973.

Dahl, O. J., et al, *Structured Programming*, Academic Press, New York, 1972 (Asupra programării cu compilatoare fără a se utiliza GO TO).

Dalton, William, „Design Microcomputer Software Like Other Systems-Systematically", *Electronics*, January 19, 1978, pag. 97—101.

Yourdon, Edward, *Techniques of Program Structures and Design*, Prentice-Hall, Englewood Cliffs, N.J., 1975.

Compilatuare

Burns and Savitt, „Microprogramming Stack Architecture Ease Minicomputer Programmers Burden, „*Electronics*, February 15, 1973, pag. 95—101. (Se descrie Microdata 32/S, care a fost proiectat pentru un compilator MPL).

Heaps, H. S., *An Introduction to Computer Languages*, Prentice-Hall, Englewood Cliffs, N.J., 1973.

McCameron, Fritz, *Forlran Logic and Programming*, Irwin Inc., Homewood, Ill, 1968.

McCracken, Daniel O., *A Guide to PL/M Programming for Microcomputer Applications*, Addison-Wesley, Reading, Mass., 1978.

Microprogramare

Gorman and Kaufman, „Low Cost Minicomputer Opens Up Many New System Opportunities", *Electronics*, June 7, 1973. (Se descrie Computer Automation Alpha LSI 16, which uses a PLA controle store).

House, David L., „Micro Level Architecture in Minicomputer Design", *Computer Design*, October 1973, pag. 75—80. (Avantajele microprogramării și proiectarea minicalculatorului Microdata 3200).

Husson, *Microprogramming Principles and Practices*, Prentice Hall, Englewood Cliffs, N.J., 1970.

IEEE, *Transactions on Computers*, „Special Issue on Microprogramming", C-20, No. 7, July, 1971.

National Semiconductor, *GPC/P Microcoding Manual*, Pub. 4200007, National Semiconductor, Santa Clara, Calif. 1974.

Serkin, Baker, W. Davidow, and Allison, „Microprogramming Made Easy With a 4096 Bit Bipolar ROM, *„Electronics"*, March 15, 1971.

Signetics Memory Systems, „Design of Microprogrammable Systems", Signetics Memory Systems Application Note SMS0052AN, Sunnyvale, Calif. (O trecere în revistă, generală, foarte bună).

Tucker, S. G., „Microprogram Control for System/360, *IBM Systems Journal*, Vol. 6, No. 4, 1967, pag. 222.

Clymer, Jim, „Use 4-Bit Slices to Design Powerful Microprogrammed Processors. The 2900 Family of Circuits Lets You Build Machines with Cycle Times of 110 ns.", *Electronic Design*, 10 May, 1977, pag. 62—71.

Lau, Stephen, „Design High-Performance Processors With Bipolar Bit Slices", *Electronic Design*, 7, March 29, 1977, pag. 86—95.

Mick, John R. and Brick, Jim, *Microprogramming Handbook*, Advanced Micro Devices, Sunyvale, Calif., 1976. (Tehnici pentru seria bit-slice 2900).

Nemec, John et al., „A Primer on Bit-Slice Processors...", *Electronic Design*, 3, February 1, 1977, pag. 52—60.

Salisbury, Alan B., *Microprogrammable Computer Architecture*, Elsevier, New York, 1976

Interpretoare și sisteme de operare

Stiefel, Malcolm L., „Venturing Forth", *Mini-Micro Systems*, August 1976, pag. 46—47.

Forth Inc., *Mieroforth Primer*, Forth Inc., Manhattan Beach, Calif. 1977.

Kahn, Kelvin C., „A Small Scale Operating Systems Foundation for Microprocessor Applications", *Proceedings of the IEEE*, Vol. 66, No. 2, February 1978. (Descrie sistemul de operare Intel RMX-80).

Claggett, Edward, „Interpreters vs. Compilers for On-Line Microcomputers", *Control Engineering*. August 1977, pag. 24—27.

Burzio, Gus, „Operating Systems Enhance μCs...", *Electronic Design*, 13, June 21, 1978, pag. 94—99.

Intel, *RMX/80 Real-Time Multitasking Executive*, Pub. 9800577A. Intel Corp., Santa Clara, Calif., 1977.

Townzen, David A, „A Task-Scheduling Executive Program for Microcomputer Systems", *Computer Design*, June 1977, pag. 194—202.

Intreruperi

Klingman, Edwin E., *Microprocessor Systems Design*, Prentice-Hall, Englewood Cliffs, N.J., 1977, Chapter 12.

Rattner, Justin, „The Open Channel", *Computer*, March 1975, pag. 19—20. (Discuție asupra eliminării întreruperilor).

9

LOGICA PROGRAMATĂ III

Sisteme de dezvoltare

9.1. SISTEME DE DEZVOLTARE

Natura ordonată a logicii programate face ca întregul proces de programare şi verificare să se poată realiza cu ajutorul calculatorului. Instrumentul de bază pentru automatizarea întregului proces este denumit sistem de dezvoltare (vezi fig. 9-1). Pe lîngă asamblarea sau compilarea de programe, sistemul de dezvoltare realizează în mod automatizat procesul de stocare şi modificare a programelor sursă şi obiect. El simulează de asemenea ROM-ul, ce conţine eventual un program şi ajută în depanarea de program şi de hardware. Îndată ce verificarea programului este completă, sistemul de dezvoltare scrie şi testează PROM-ul ce va fi inclus în sistemul final.

Cel mai ieftin sistem de dezvoltare constă doar dintr-un microcomputer asamblat pe o singură placă de cablaj imprimat şi dotat cu o interfaţă serie pentru cuplarea unei imprimante şi cu două casetofoane audio. Sistemele profesionale măresc viteza de asamblare în mod considerabil utilizînd pentru stocare discuri flexibile. În orice caz, sînt necesare două dispozitive de memorie, astfel încît datele să poată fi copiate de pe un dispozitiv pe altul. Dacă nu posedăm această facilitate de a face copii de rezervă, o singură greşeală sau defecţiune poate să distrugă rezultatele unor luni de lucru.

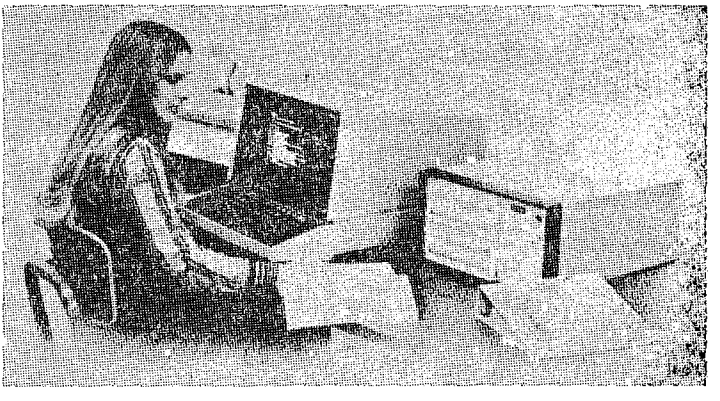


Fig. 9-1. Sistem de dezvoltare pentru microprocesor. Un terminal inteligent, o unitate dublă de disc flexibil şi o imprimantă reprezintă o structură ideală pentru dezvoltarea şi punerea la punct a programelor.

O altă economie importantă de timp se realizează în sistemele de dezvoltare profesionale prin utilizarea unei imprimante rapide și a unui terminal cu afișare pe tub catodic. Acesta din urmă face posibilă editarea și introducerea interactivă a programelor. Textul de pe ecran poate fi citit rapid și corectat pe loc. Programul *Editor* permite modificări sau inserții de caractere sau chiar de linii întregii din program. Pentru a efectua o modificare, programatorul selectează modul *replace* (înlocuiește), *insert character* (inserează caracter), *insert line* (inserează linie), *delete character* (elimină caracter), *delete line* (elimină linia) și poziționează un cursor acolo unde este necesară o modificare. Noile date și corecții sînt apoi introduse pe tastatură și modificarea rezultată este afișată imediat pentru reverificare.

Cînd un sistem de dezvoltare este cuplat la alimentare, el începe să execute un program denumit *executiv, monitor* sau *sistem de operare* (în funcție de producător)*. Acest program răspunde la comenzile introduse de programator prin tastatură. Fiecare din aceste comenzi produce execuția unui program diferit, ca *editorul de text* și *asamblorul*. Cînd programul chemat este terminat, controlul este transferat programului *executiv*.

Prima etapă în crearea unui nou program constă deci în chemarea programului editor de texte, prin emiterea unei comenzi ca EDIT, urmată de numele noului program. Programul sursă este apoi introdus prin tastatură, sub controlul programului editor de texte. Dacă programul este mai lung decît o pagină, editorul îl divizează în pagini pe care le stochează pe disc. Taste de control speciale permit *urmărirea pe pagini* (în sens direct sau invers) a programului prin scrierea automată a paginii afișate (cu corecții) pe disc și citirea paginii următoare de pe disc.

Cînd programul este terminat, programatorul îl parcurge pe pagini și corectează erorile evidente, apoi se întoarce la programul executiv. Următoarea etapă constă într-o cerere de asamblare, efectuată printr-o comandă ca ASMB, urmată de numele programului sursă, de numele fișierului obiect și de un cod ce indică dacă rezultatele trebuie tipărite. Programul asamblor începe apoi prelucrarea. Asamblarea unui program mare poate lua cîteva minute deoarece asamblorul folosește discul pentru a memora denumirile adreselor simbolice.

Deoarece imprimarea unei listări a programului ia mult timp, primul rezultat al asamblării este de obicei afișat doar pe ecran. Dacă sînt erori de asamblare (și de obicei există), editorul de texte este chemat din nou și este folosit să corecteze programul sursă. Cînd toate erorile sînt corectate, programul este asamblat din nou. Pentru a realiza o asamblare perfectă, pot fi necesare mai multe repetări ale acestui ciclu. În acest moment, se poate efectua o imprimare comandată cu PRINT, urmat de numele programului obiect.

Deoarece programul de asamblare și tipărire a listărilor de programe mari poate solicita mult timp, programele mari trebuie să fie divizate în module de dimensiuni convenabile. Această divizare este posibilă numai dacă asamblorul produce un cod obiect *relocatabil*. Un *program de legături (linker)* poate apoi combina modulele separate într-un program mare de sistem, depus în locații adiacente de memorie. Aceasta permite modificarea dimensiunilor unui modul individual de program, fără a fi obligați să reasamblăm sau să imprimăm o listare a întregului program de sistem.

---

* În mod normal, programul executiv este transferat de pe disc în RAM, de către un program aflat în ROM denumit program *bootstrap*. Aceasta are loc în mod automat la fiecare cuplare a rețelei.

9.2. DEPANAREA PROGRAMELOR

Asamblarea fără erori a unui program nu garantează cîtuşi de puţin funcţionarea sa corectă. Procesul de găsire şi îndepărtare a erorilor de program se numeşte depanare (debugging). În principiu, programul de *depanare* al sistemului de dezvoltare automatizează această sarcină. El permite execuţia pas cu pas a programelor şi afişează conţinutul registrelor după fiecare pas de program. El permite, de asemenea, ca urmare a introducerii unor comenzi prin tastatură, să modificăm locaţii de memorie sau registre.

Figura 9-2 prezintă aspectul ecranului la un moment dat în cursul depanării unui program pe un sistem de dezvoltare AMI 6800. Funcţionarea programului poate fi verificată prin parcurgerea sa pas cu pas şi urmărirea corectitudinii rezultatelor. Datele pot fi introduse prin tastatură, efectuînd scrieri în registre la momentul oportun. De exemplu, după ce în registrul *A* a fost introdus, prin tastatură, un anumit cuvînt, putem *urmări (trace)* evoluţia rezultatelor parţiale pe măsură ce se execută fiecare treaptă de program.

Pentru a economisi timp, segmentele mai lungi de program pot fi executate rapid, cu o oprire automată la o locaţie oarecare (*breakpoint*). Aceasta produce oprirea execuţiei programului în momentul în care adresa punctului de oprire apare pe busul de adrese. De exemplu, după ce am introdus date, putem urmări rezultatul prin fixarea punctului de oprire la locaţia instruc-

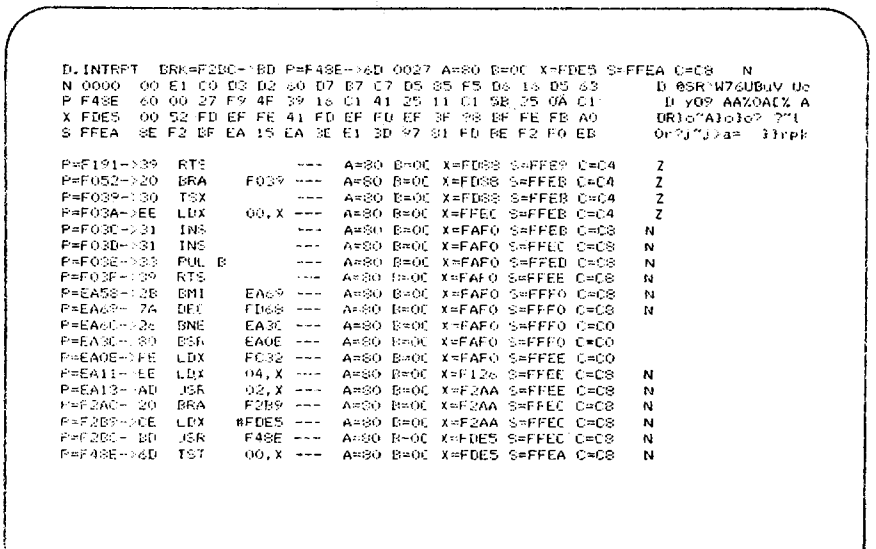


Fig. 9-2. Afişarea pe tub catodic în procesul de depanare a unui program. Prima linie indică oprirea programului ca urmare a unei întreruperi, punctul de oprire la adresa F2BC, cu valoarea numărătorului de program la F48E, precum şi conţinutul rgistrelor A, B, X, S şi C. Următoarele patru linii indică, fiecare, conţinutul a cîte 16 locaţii de memorie (în cod hexazecimal), cu specificarea primei adrese afişate la capăt de rînd. În partea dreaptă a acestor linii este indicat conţinutul aceloraşi locaţii cu caractere ASCII. Următoarele 19 linii afişate prezintă ultimele 19 instrucţiuni executate (începînd de sus cu cea mai recentă). Pe fiecare linie este indicat conţinutul numărătorului de program, transcrierea numerică a codului operaţiei (instrucţiunii), cîmpul de adresă al instrucţiunii, conţinutul registrelor A, B, X, S, C şi codul de condiţii.

țiunii ce transferă rezultatul. Afișajul prezintă rezultatul, ce urmează să fie extras din registrul *A*, și cei 19 pași de program ce au precedat pasul în cauză.

De asemenea, putem afișa în mod continuu conținutul unei anumite locații de memorie, prin simpla indicare a adresei sale în hexazecimal pe linia superioară a ecranului. Se pot scrie date în oricare din locațiile afișate prin plasarea în locul respectiv a cursorului și introducerea prin tastatură a valorii dorite (în hexazecimal). Deoarece intrările și ieșirile sistemului sînt tratate ca locații de memorie, în mod asemănător putem afișa sau modifica și starea tuturor registrelor de intrare sau ieșire.

Deoarece programul de depanare traduce în mod automat codurile operațiilor din „limbajul mașină" în mnemonicile de asamblor, programatorul poate să nu se mai gîndească la biții din limbajul mașină și poate opera cu mnemonicele instrucțiunilor. Dacă se detectează o eroare în program, se poate pune un „petec" temporar, fără reasamblare, prin simpla indicare a locației și a instrucțiunilor dorite *în formă mnemonică*. Cîmpurile de adrese ale instrucțiunilor pot fi indicate ca cifre hexazecimale sau prin denumirea lor simbolică (dacă a fost încărcat anterior un tabel de traducere a simbolurilor).

În cursul procesului de depanare a programului, acesta este depus în RAM, pentru ca modificările să poată fi ușor efectuate. Avînd în vedere că majoritatea microcomputerelor folosesc, în forma definitivă, pentru memoria lor de program PROM sau ROM, sistemele de dezvoltare includ un soclu pentru scrierea programului în PROM. Cînd verificarea programului în RAM este completă, se introduce un PROM în soclul de programare și se transferă controlul programului de programare a memoriilor fixe. Acest program scrie în PROM și verifică cu o copie a programului existent în RAM. Acest PROM poate fi inclus în varianta finală a sistemului sau poate fi folosit pentru a programa alte PROM-uri sau ROM-uri.

Deoarece PROM-urile sînt verificate în mod automat, bit cu bit, pentru a fi identice cu programul ce funcționează, în cazul logicii programate șansele de eroare umană și de funcționare incorectă sînt reduse în mod substanțial. Deoarece documentația programului se realizează de asemenea în mod automat, se elimină practic posibilitatea erorilor de documentație sau motivele de apariție a documentației depășite, corespunzătoare unei etape anterioare.

9.3. GESTIUNEA FIȘIERELOR

O altă funcție ce este automatizată, în cazul sistemelor de dezvoltare bazate pe disc flexibil, este funcția de păstrare a înregistrărilor. Fiecare disc are o evidență ce conține denumirile, dimensiunile și locațiile tuturor fișierelor depuse pe el. (vezi figura 9-3). Aceste fișiere conțin în general programe sursă sau obiect, dar pot fi în realitate orice tip de text (de exemplu, manuale sau specificații). Editorul de text poate fi folosit pentru a efectua modificări sau actualizări de orice natură a textelor, inclusiv a programelor. De fapt, sistemul de dezvoltare este și un „procesor de cuvinte".

În mod normal, fiecare inginer, programator sau secretară are propriile sale dischete (discuri flexibile), care conțin programele de sistem ca și orice alte programe ce se află în testare. Sistemul de operare are comenzi pentru

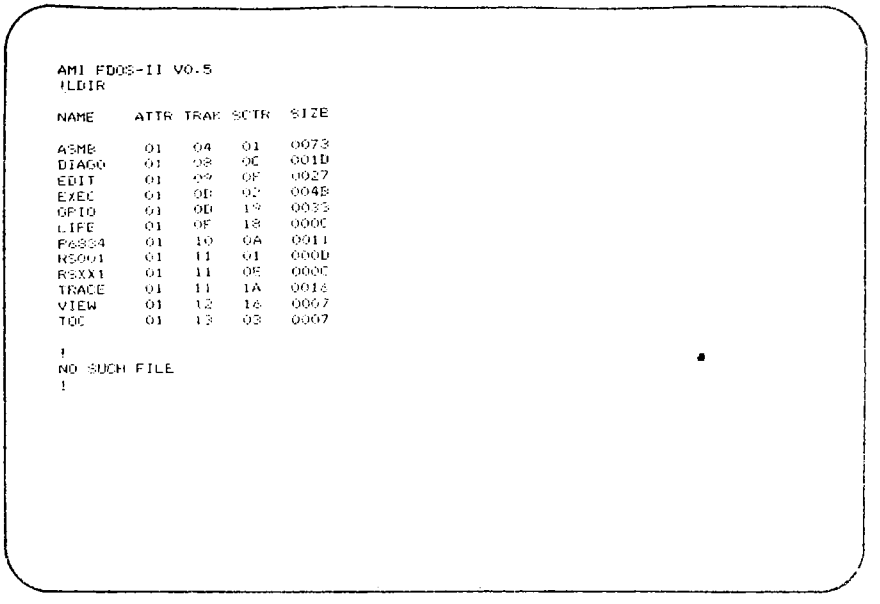


Fig. 9-3. Aşezarea pe tub catodic a numelor programelor stocate pe un disc flexibil. Acest catalog a fost afişat de către sistemul de operare FDOS ca răspuns la comanda LDIR, transmisă prin claviatură. Oricare din aceste nume poate fi editat, asamblat, înlăturat ş.a.m.d. prin tastarea unor comenzi ulterioare. Răspunsul : NO SUCH FILE (nu există o astfel de filă), afişat în partea de jos a ecranului, a apărut ca urmare a unei comenzi incorecte, accidentale.

crearea de noi fişiere (CREATE), anularea unor fişiere vechi (DELETE), modificarea denumirii unor fişiere (RENAME), afişare a unui fişier pe ecran (VIEW), tipărirea unui fişier (PRINT), copierea (COPY) selectivă de fişiere de pe un disc pe altul, şi gruparea (MERGE) a mai multor fişiere vechi sub o singură denumire nouă. O listă tipică a comenzilor sistemului de operare este dat în Tabelul 9-1.

Este de remarcat că listele de parametri ce apar după comenzile din Tabelul 9-1 nu trebuie să fie mereu completate în întregime. Cînd comanda este terminată cu CR (carriage return), programul va lucra pe parametrii ce au fost introduşi pînă în momentul respectiv.

Folosirea, de exemplu, a comenzii EDIT pentru a corecta un program denumit STERM 2, presupune următorul mesaj :

EDIT, STEMR2, STEMR3 (carriage return)

Programul editor de texte va încărca în mod automat vechiul program (STERM2) şi va crea un nou fişier, denumit STERM3. Orice corecţii ce se introduc vor deveni o parte a noii versiuni, (3), a programului în timp ce versiunea STERM2 va fi păstrată în fişier pînă va fi anulată prin comanda DELETE, STERM2 (carriage return).

În general, este o idee bună să faceţi o copie a discului, pentru o refacere rapidă după o funcţionare defectuoasă sau un accident. Lista din figura 9-3 conţine doar programele ce sînt livrate odată cu sistemul de dezvoltare. Cînd un program este în curs de dezvoltare, pe disc sînt prezente de obicei versiu-

Tabelul 9-1 **Comenzile sistemului de oprire AMI**

ASMB, denumire fişier sursă, denumire fişier destinaţie, p, lista dispozitivelor
p = 2 obiect + listare
p = 3 numai listare
p = 4 numai obiect
CHGAT, denumire fişier, atribute noi
COPY (de pe discul 1 pe discul 2)
CREATE, denumire fişier, dimensiune
DEBUG
DELET : u, denumire fişier 1, denumire fişier 2, ... denumire fişier n
DUMP, denumire fişier (plasează fişier pe imprimantă)
EDIT, denumirea registrului de intrare, denumirea registrului de ieşire
EDIT, denumirea registrului de ieşire
HOME:u (poziţionează capul de lectură al dischetei pe pista 0)
INIT:u (u = 1, 2, 3 : iniţializează discheta utilizatorului)
INITX (implică u = 0 : iniţializează discheta sistem)
LDIR:u, lista dispozitivelor
LOAD, denumirea fişierului, decalajul (opţional)
MERGE, noua denumire a fişierului, denumirea fişierului 1, denumirea fişierului 2, denumirea fişierului n
PRINT, denumirea fişierului, lista dispozitivelor
RENAM, denumirea fişierului vechi, denumirea fişierului nou
RUN, fişierul obiect, denumirea fişierului de intrare, denumirea fişierului de ieşire, parametru
STORE, denumirea fişierului destinaţie (de pe dispozitivul lector)
VIEW, denumirea fişier, n linii, linia de start
XGEN.

: u este numărul unităţii u = 0, 1, 2 sau 3
denumirea fişierului poate apare ca denumirea fişierului : u
toate numerele apar în hexazecimal
lista dispozitivelor : disc, perforator, imprimantă, consolă (în absenţa unei specificaţii se ia automat consola)

nile vechi şi cele curente ale programelor sursă şi obiect, ca şi o selecţie din programele de sistem.

Denumirile programelor trebuie să fie scurte, cu un „S" sau un „O", ataşate pentru a indica „sursă" sau „obiect". La sfîrşitul denumirii programului va fi introdus şi un indice, pentru a putea distinge versiunile succesive. Se recomandă întotdeauna păstrarea versiunii precedente, pentru a putea reveni la ea dacă în noua variantă se dovedeşte a fi o greşeală.

Paginile care urmează ilustrează o dezvoltare tipică de program aşa cum este văzută pe ecranul unui sistem de dezvoltare AMI 6800. Programul în cauză trebuie să producă afişarea alfabetului ; el este scris, editat, asamblat, depanat, corectat, reasamblat şi trecut în evidenţă. Deşi nu am specificat suficiente detalii pentru ca cititorul să-l poată urmări exact, el poate să-şi formeze o idee corectă asupra procesului global.

9.4. REALIZAREA TESTĂRII HARDWARE-ULUI CU UN SISTEM DE DEZVOLTARE

Deşi depanarea iniţială a programelor poate fi făcută şi manual efectuînd introduceri de date şi urmărind ieşirile, testarea reală a sistemului pretinde testarea circuitelor de interfaţă şi a dispozitivelor de intrare şi ieşire. O opţiune puternică, prezentă în majoritatea sistemelor de dezvoltare, denumită *emularea în circuit*, face posibilă *cuplarea sistemului de dezvoltare în soclul*

```
AMI FDOS-II V0.5

!EDIT..TEST1

FDOS-II/AMI EDITOR VER. 1.1
@INAM TEST -- PRINT THE ALPHABET
 OPT O
 ORG $100
DPYOUT EQU $E006 MONITOR DISPLAY OUTPUT ROUTINE
*
START LDA A #13 CARRIAGE RETURN
 JSR DPYOUT
 LDA A #10 LINEFEED
 JSR DPYOUT
 LDA A #'A FIRST LETTER TO PRINT
 LDA B #26 NUMBER OF LETTERS
LOOP JSR DPYOUT DISPLAY A LETTER
 NOP
 DECB COUNT
 BNE LOOP CONTINUE UNTIL COUNT IS ZERO
 WAI THEN STOP
 END
$$
@
```

(a) Apelarea editorului; introducerea programului de la claviatură

```
!EDIT,,TEST1

FDOS-II/AMI EDITOR VER. 1.1
@I NAM TEST -- PRINT THE ALPHABET
 OPT O
 ORG $100
DPYOUT EQU $E006 MONITOR DISPLAY OUTPUT ROUTINE
*
START LDA A $13 CARRIAGE RETURN
 JSR DPYOUT
 LDA A $10 LINEFEED
 JSR DPYOUT
 LDA A #'A FIRST LETTER TO PRINT
 LDA B #26 NUMBER OF LETTERS
LOOP JSR DPYOUT DISPLAY A LETTER
 NOP
 DECB COUNT
 BNE LOOP CONTINUE UNTIL COUNT IS ZERO
 WAI THEN STOP
 END
$$
@BCPRINT$DISPLAY$$
@BT$$
 NAM TEST -- DISPLAY THE ALPHABET
@
```

(b) Se modifică linia de comentarii; în loc de „DISPLAY" se scrie „PRINT"

```
 OPT O
 ORG $100
DPYOUT EQU E006 MONITOR DISPLAY OUTPUT ROUTINE
*
START LDA A13 CARRIAGE RETURN
 JSR DPYOUT
 LDA A10 LINEFEED
 JSR DPYOUT
 LDA A'A FIRST LETTER TO PRINT
 LDA B26 NUMBER OF LETTERS
LOOP JSR DPYOUT DISPLAY A LETTER
 NOP
 DECB COUNT
 BNE LOOP CONTINUE UNTIL COUNT IS ZERO
 WAI THEN STOP
 END
$$
@BCPRINT$DISPLAY$$
@BT$$
 NAM TEST -- DISPLAY THE ALPHABET
@E$$

!ASMB.TEST1,TESTL,3,0
!3
!
```

(c) Ieşirea din editor, lansarea asamblorului pentru a produce pe dischetă un fişier de tip listing cu numele TEST L

Fig. 9-4. Secvenţele unui program tipic de dezvoltare *(a—f)*. Programul este generat, asamblat, testat, depanat şi verificat că merge bine.

```
!ASMB,TEST1,TESTL,3,0
!3
!VIEW,TESTL

$---

PAGE  001  TEST ==

00001                       NAM    TEST     -- DISPLAY THE ALPHABET
00002                       OPT    0
00003 0100                  ORG    $100
0000       E006     DPYOUT  EQU    $E006    MONITOR DISPLAY OUTPUT ROUTI
00005                       *
00006 0100 86 0D    START   LDA A  #13       CARRIAGE RETURN
00007 0102 BD E006          JSR    DPYOUT
00008 0105 86 0             LDA A  #10       LINEFEED
00009 0107 BD E006          JSR    DPYOUT
00010 010A 86 41            LDA A  #'A       FIRST LETTER TO PRINT
00011 010C C6 1A            LDA B  #26       NUMBER OF LETTERS
00012 010E BD E006 LOOP     JSR    DPYOUT   DISPLAY A LETTER
00013 0111 01               NOP
00014 0112 5A               DEC B  COUNT
```

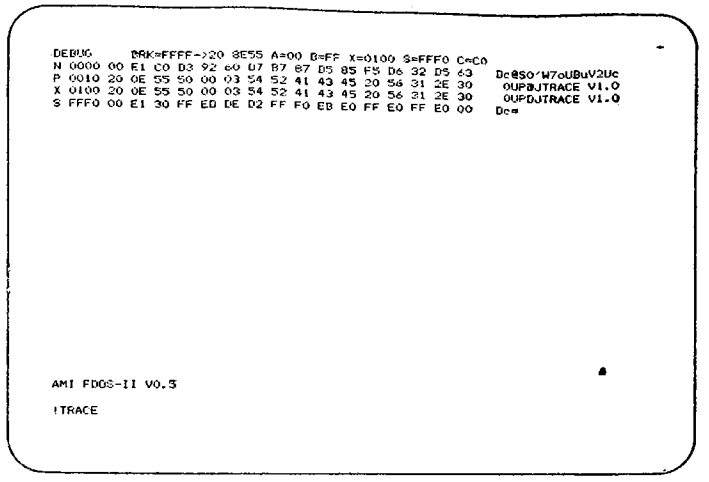
(d) Afişează, cite un ecran odată, conţinutul fişierului TEST L

```
00001                       NAM    TEST     -- DISPLAY THE ALPHABET
00002                       OPT    0
00003 0100                  ORG    $100
00004      E006     DPYOUT  EQU    E006     MONITOR DISPLAY OUTPUT ROUTI
00005                       *
00006 0100 86 0D    START   LDA A  #13       CARRIAGE RETURN
00007 0102 BD E006          JSR    DPYOUT
00008 0105 86 0A            LDA A  #10       LINEFEED
00009 0107 BD E006          JSR    DPYOUT
00010 010A 86 41            LDA A  #'A       FIRST LETTER TO PRINT
00011 010C C6 1A            LDA B  #26       NUMBER OF LETTERS
00012 010E BD E006 LOOP     JSR    DPYOUT   DISPLAY A LETTER
00013 0111 01               NOP
00014 0112 5A               DEC B  COUNT
00015 0113 26 F9            BNE    LOOP     CONTINUE UNTIL COUNT IS ZERO
00016 0115 3E               WAI              THEN STOP
00017                       END

TOTAL ERRORS 00000

!ASMB,TEST1,TESTX,4
!4
!
```

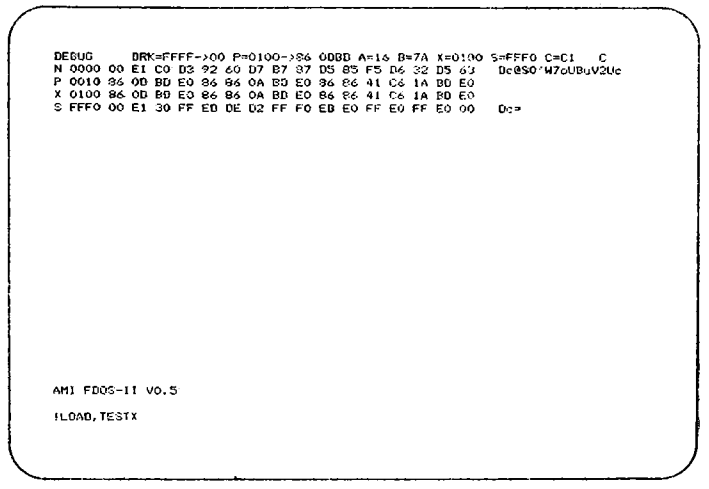
(e) Asamblare pentru a produce un fişier de tip obiect

```
00001                       NAM    TEST     -- DISPLAY THE ALPHABET
00002                       OPT    0
00003 0100                  ORG    $100
00004      E006     DPYOUT  EQU    $E006    MONITOR DISPLAY OUTPUT ROUTI
00005                       *
00006 0100 86 0D    START   LDA A  #13       CARRIAGE RETURN
00007 0102 BD E006          JSR    DPYOUT
00008 0105 86 0A            LDA A  #10       LINEFEED
00009 0107 BD E006          JSR    DPYOUT
00010 010A 86 41            LDA A  #'A       FIRST LETTER TO PRINT
00011 010C C6 1A            LDA B  #26       NUMBER OF LETTERS
00012 010E BD E006 LOOP     JSR    DPYOUT   DISPLAY A LETTER
00013 0111 01               NOP
00014 0112 5A               DEC B  COUNT
00015 0113 26 F9            BNE    LOOP     CONTINUE UNTIL COUNT IS ZERO
00016 0115 3E               WAI              THEN STOP

TOTAL ERRORS 00000

!ASMB,TEST1,TESTX,4
!4
!RUN,TESTX

AAAAAAAAAAAAAAAAAAAAAAAAAA
```

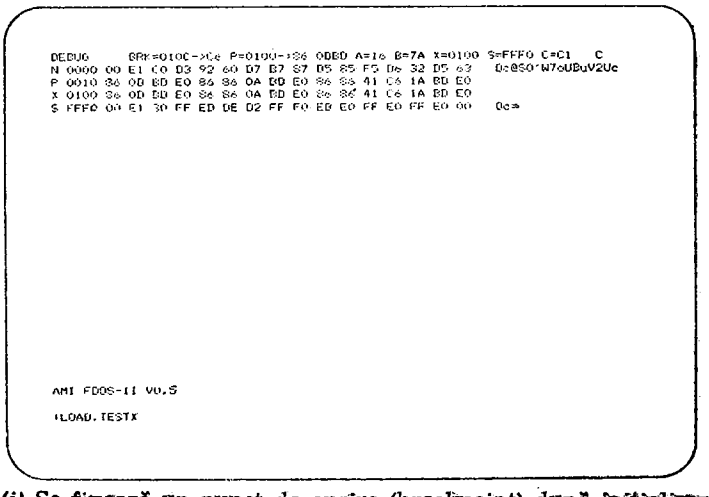
(f) Se lansează execuţia programului. Se constată că programul nu funcţionează corect (nu tipăreşte alfabetul ci numai litera A)



(g) Se încarcă programul pentru urmărirea execuției (TRACE)

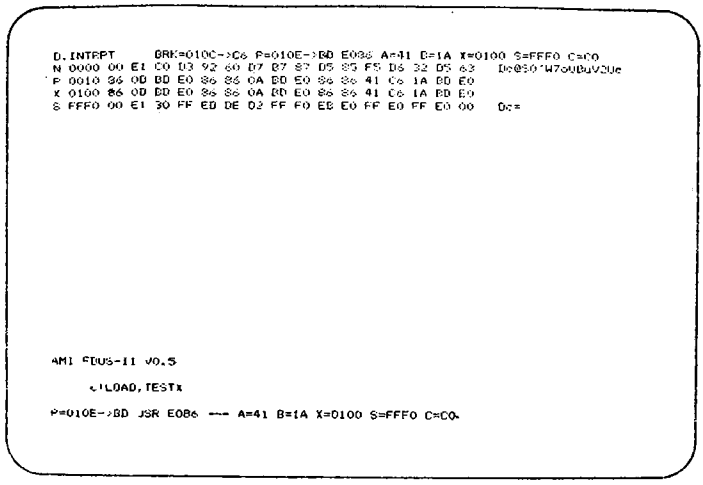


(h) Se încarcă programul ce trebuie depanat

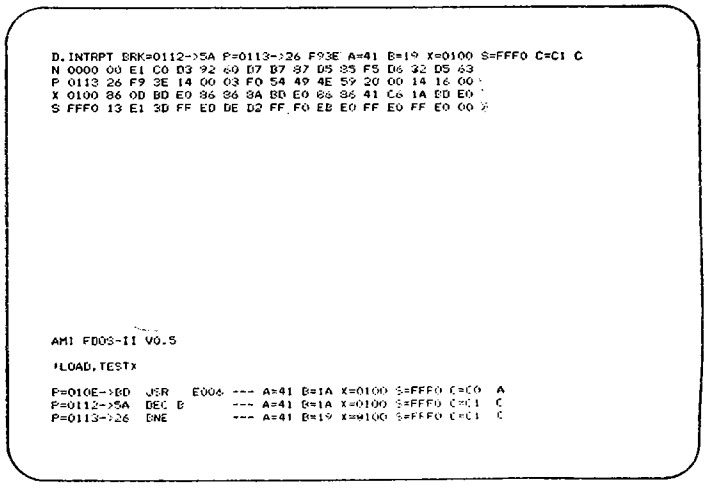


(i) Se fixează un punct de oprire (breakpoint) după inițializare

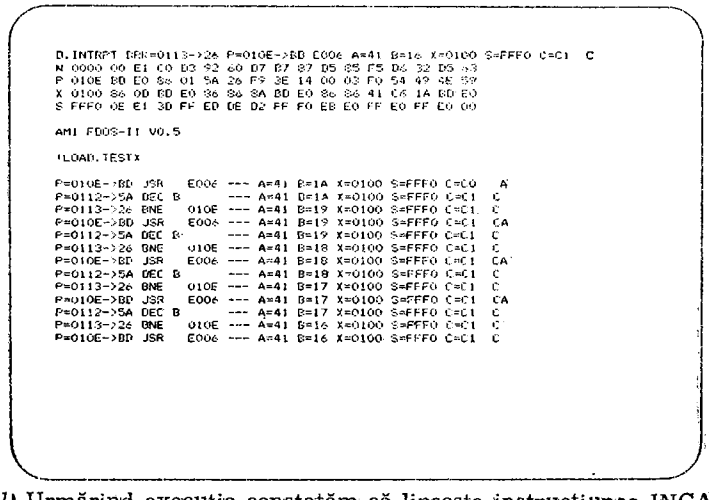
Fig. 9-4. *(g-l)*. Depanarea programului.



(j) Se lansează programul. Se începe urmărirea execuției de la punctul de oprire



(k) Se urmărește programul principal. Se inhibă urmărirea în subrutine



(l) Urmărind execuția constatăm că lipsește instrucțiunea INCA

```
!EDIT,TEST1,TEST2

EDOS-II/AMI EDITOR VER. 2.1
@ASMT$$
 NAM TEST -- DISPLAY THE ALPHABET
 OPT O
 ORG $100
DPYOUT EQU $E006 MONITOR DISPLAY OUTPUT ROUTINE
$
START LDA A #13 CARRIAGE RETURN
 JSR DPYOUT
 LDA A #10 LINEFEED
 JSR DPYOUT
 LDA A #'A FIRST LETTER TO PRINT
 LDA B #26 NUMBER OF LETTERS
LOOP JSR DPYOUT DISPLAY A LETTER
 NOP
 DECB COUNT
 BNE LOOP CONTINUE UNTIL COUNT IS ZERO
 WAI THEN STOP
 END
@SDECB$OLT$$
 DECB COUNT
```

(m) Apelează editorul. Se listează programul, se caută locul unde trebuie făcută corecția

```
 END
@SDECB$OLT$$
 DECB COUNT
@IINCA
$$
@@99T$$
 NAM TEST --- DISPLAY THE ALPHABET
 OPT O
 ORG $100
DPYOUT EQU $E006 MONITOR DISPLAY OUTPUT ROUTINE
$
START LDA A #13 CARRIAGE RETURN
 JSR DPYOUT
 LDA A #10 LINEFEED
 JSR DPYOUT
 LDA A #'A FIRST LETTER TO PRINT
 LDA B #26 NUMBER OF LETTERS
LOOP JSR DPYOUT DISPLAY A LETTER
 NOP
INCA
 DECB COUNT
 BNE LOOP CONTINUE UNTIL COUNT IS ZERO
 WAI THEN STOP
 END
```

(n) Se inserează o nouă linie, se afișează programul

```
 ORG $100
DPYOUT EQU $E006 MONITOR DISPLAY OUTPUT ROUTINE
$
 START LDA A #13 CARRIAGE RETURN
 JSR DPYOUT
 LDA A #10 LINEFEED
 JSR DPYOUT
 LDA A #'A FIRST LETTER TO PRINT
 LDA B #26 NUMBER OF LETTERS
LOOP JSR DPYOUT DISPLAY A LETTER
 NOP
INCA
 DECB COUNT
 BNE LOOP CONTINUE UNTIL COUNT IS ZERO
 WAI THEN STOP
 END
@SINCA$OL1 $OLI$$
 INCA
@E$$

!ASMB,TEST2,TESTY,0
!0
!RUN,TESTY

ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

(o) Se corectează eroarea pe noua linie, se iese din editor, se asamblează și se rulează. De data aceasta programul funcționează

Fig. 9-4. *(m-r)*. Editarea și manipularea fișierelor.

```
*DIR

NAME    ATTR  TRAK  SCTR  SIZE

ASMB    01    04    01    0073
DIAGO   01    08    0C    0010
EDIT    01    09    0F    0027
EXEC    01    0B    02    0048
GPIO    01    0D    19    0033
LIFE    01    0F    18    000C
P6834   01    10    0A    0011
RSOO1   01    11    01    0000
RSXX1   01    11    0E    000C
TRACE   01    11    1A    0016
VIEW    01    12    16    0007
TOC     01    13    03    0007
TEST1   00    13    0A    0003
TESTL   00    13    09    0008
TESTX   00    13    15    0001
TEST2   00    13    16    0004
TESTY   00    13    1A    0001
```

(p) Se afişează catalogul (directory) fişierelor de pe disc

```
LIFE   01 0F 18 000C
P6834  01 10 0A 0011
RSOO1  01 11 01 0000
RSXX1  01 11 0E 000C
TRACE  01 11 1A 0016
VIEW   01 12 16 0007
TOC    01 13 03 0007
TEST1  00 13 0A 0003
TESTL  00 13 00 0008
TESTX  00 13 00 0008
TEST2  00 13 16 0004
TESTY  00 13 1A 0001

*DELETC,TEST1,TESTL,TEST*
TEST1 DELETED
TESTL DELETED
TESTX DELETED
PACKING DISK

*RENAME.TEST2,TEST6

*RENAM,TESTY,TEST
```

(q) Se şterg vechile versiuni şi se renumesc cele noi

```
*RENAM,TESTY,TEST

*DIR

NAME    ATTR  TRAK  SCTR  SIZE

ASMB    01    04    01    0073
DIAGO   01    08    0C    0010
EDIT    01    09    0F    0027
EXEC    01    0B    02    0048
GPIO    01    0D    19    0033
LIFE    01    0F    18    000C
P6834   01    10    0A    0011
RSOO1   01    11    01    0000
RSXX1   01    11    0E    000C
TRACE   01    11    1A    0016
VIEW    01    12    16    0007
TOC     01    13    03    0007
TEST6   00    13    0A    0004
TEST    00    13    0E    0001

*TEST

ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

(r) Se afişează catalogul, se execută fişierul obiect (cu noul nume).

microprocesorului, din produsul real. Aceasta permite rularea programului la viteza maximă și urmărirea modului în care interacționează cu dispozitivele de intrare/ieșire ale produsului.

Pentru produs, conectorul cu 40 de terminale al emulatorului se prezintă cu un microcomputer sau un microprocesor în stare de funcționare. Chiar dacă produsul este dotat cu o memorie separată de program, emulatorul poate să o ignore și să folosească în locul ei RAM-ul din sistemul de dezvoltare. Modificările de program pot fi astfel efectuate simplu, fiind disponibile facilitățile de *debug* și *trace*.

Cu programul rulînd la viteza maximă, stările succesive ale microcomputerului sînt înregistrate într-o memorie de *urmărire*, împreună cu stările altor puncte de test ce prezintă interes. Cînd este detectată adresa specificată sau alt semnal de declanșare, procesul de înregistrare este oprit, păstrînd în memorie cele 1023 de stări ale mașinii ce preced semnalul de sincronizare. În acest mod, este posibil să urmărim evoluția stărilor *anterioare* ale microprocesorului și punctelor de test pentru a determina cauza unei eventuale funcționări defectuoase. Stările mașinii pot fi vizualizate în diferite forme, ce includ și un format „dezasamblat" care transformă codurile instrucțiunilor în mnemonică, astfel încît funcționarea programului poate fi urmărită la fel ca în cazul lecturii în limbaj de asamblare. Cuvintele ce corespund grupurilor de probe logice pot fi decodate și afișate în zecimal, ASCII, hexazecimal, ș.a.m.d.

În modul de lucru pas cu pas este posibil să urmărim efectiv citirea datelor de intrare de către program. Putem, de asemenea, executa instrucțiuni de ieșire, urmărind rezultatele lor pe dispozitivele de ieșire din produsul final. Tehnica emulării în circuit permite astfel conectarea tuturor dispozitivelor de intrare și de ieșire la sistemul de dezvoltare la fel cum sînt conectate la microcalculator în produsul real. În felul acesta, problemele subtile care implică interacțiunea intrare/ieșire sau/și temporizarea pot fi rezolvate în mediul automatizat al sistemului de dezvoltare.

Emulatorul în circuit poate fi folosit nu numai la testarea dispozitivelor de intrare/ieșire, ci și a cipurilor ROM și RAM din produs. Folosind programul de depanare (debug) se pot scrie și citi locații individuale ; de asemenea, se pot rula programe speciale de diagnoză. Programul de diagnoză al RAM-ului (ce face parte din softul de sistem) scrie și citește configurațiile de biți ce trebuie testați în RAM-ul din produs cu un PROM cunoscut a fi corect. Harta de memorie operativă (RWM) din dotarea sistemului de dezvoltare determină dacă dispozitivele din produs sau echivalentele lor din sistemul de dezvoltare vor răspunde la o anumită gamă de adrese. În consecință sistemul de dezvoltare este un instrument puternic atît în testarea de hard cît și în prepararea programelor.

EXERCIȚII

1. Care este conținutul (exprimat în hexazecimal) al locației de memorie 0003 în reprezentarea din figura 9-2 ?
2. Care este codul în hexazecimal al următoarei instrucțiuni de executat în figura 9-2 ?
3. Care filă din figura 9-3 este cea mai mare ? Care este cea mai mică ?
4. Ce zonă de memorie ocupă programul de scriere a alfabetului, din figura 9-4 (înainte de corecturi) ?
5. Pentru sistemul de dezvoltare descris de Tabelul 9-1 care sînt tastele ce trebuie acționate pentru a se face :
   (*a*) Înlăturarea filei numită ALPHA3 ?
   (*b*) Editarea filei numită ALPHA3 și generarea unei noi versiuni #4 ?
   (*c*) Asamblarea filei editate și afișarea listingului pe consolă ?
   (*d*) Rularea programului obiect ?

10

LOGICA PROGRAMATĂ IV

Proiectarea hardware-ului microcalculatoarelor

Procesul de proiectare a sistemelor reprezintă o trecere gradată de la definirea cerințelor generale la definirea unor detalii concrete. În sistemele centrate pe microprocesoare, majoritatea detaliilor de funcționare sînt definite prin programe. Totuși, înainte de a începe programarea propriu-zisă, este necesară stabilirea unei configurații de hardware care să fie capabilă (cel puțin cu titlu de încercare) să satisfacă specificațiile sistemului.

De asemenea trebuie proiectate și circuite de interfață, pentru a converti semnalele (digitale) ale microcalculatorului în semnale compatibile cu dispozitivele electromecanice de intrare și ieșire. Este posibil ca în cazul sistemelor simple aceasta să fie unica problemă în materie de proiectare hardware. Microcalculatoarele realizate pe un singur cip (vezi fig. 8-1) includ un microprocesor, memorie, temporizator și linii de intrare-ieșire prevăzute cu latch-uri. Dacă parametrii microcalculatorului sînt adecvați problemei, proiectarea de hardware se reduce la conceperea unui cablaj imprimat pentru a ne cupla la circuitele de interfață și la punerea sa într-o cutie împreună cu o sursă de alimentare.

Adesea, se impune prezența unei logici externe simple, pentru a manipula sarcini care altfel ar mări în mod serios timpul de execuție al procesorului. De exemplu, datele prezente la o intrare asincronă pot fi prelucrate direct de către program, dacă programul eșantionează intrarea cel puțin la un interval de timp egal cu 1/8 din *durata unui bit*. Dacă se folosește o logică externă, de sincronizare, aceasta se poate reduce la o întrerupere pe durata unui bit, iar dacă utilizăm un registru extern de deplasare, programul trebuie întrerupt doar o dată la fiecare *byte*.

Pentru a face o alegere inteligentă asupra demarcației între implementarea în hardware și cea în software, este adesea necesar să scriem versiuni preliminare ale programelor necesare. De exemplu, dacă viteza de transmisie serie este de 1200 baud, întreruperile la 1/8 din durata unui bit ar fi decalate la numai 104 μs. Deoarece programul de rezolvare a întreruperii ia cel puțin tot atîta timp putem exclude această posibilitate. Varianta de sincronizare externă mărește intervalul de întrerupere la 833 μs. Dacă subrutina întreruperii ia pentru execuție 300 μs, sarcina de urmărire a intrării asincrone ia 300/833 = 36% din timpul procesorului. Dacă intrările se fac la nivel de bytes, timpul de execuție al rutinei de intrare este redus la aproximativ 150 μs, la fiecare 6664μs, adică la numai 2,25% din timpul procesorului.

Dacă aplicația impune și alte prelucrări, cu o pondere importantă în timp, este necesar, în mod clar, să operăm la nivel de byte. Există cipuri

receptoare serie specializate pentru această problemă. Dacă aplicația nu solicită într-o măsură serioasă alte prelucrări în timpul recepției, prelucrarea datelor de intrare la nivel de bit poate fi satisfăcătoare. Este de reținut că, în fazele preliminare ale proiectării sînt necesare estimări atît ale costului hardware cît și ale *încărcării* programului, pentru a face compromisuri hardware-software convenabile.

Cipurile LSI pentru cuplarea cu dispozitive periferice, cum este receptorul de date serie, sau cele suplimentare de RAM ori ROM se pot adăuga la orice microcalculator sau sistem cu microprocesor prin simpla lor conectare în paralel la un *bus* comun.

10.1. CONCEPTUL DE BUS

Un bus este pur și simplu un grup de linii de semnal, (sîrme de conexiune sau trasee de cablaj imprimat). La un bus se cuplează numeroase dispozitive, în paralel (însă, în așa fel încît la un moment dat numai un dispozitiv poate transmite semnale pe bus). *Busul de sistem, întîlnit în cazul microprocesoarelor, este divizat, în mod conceptual în trei busuri.*

1. Busul de *adrese* (8, 12, 16 sau 20 de biți)
2. Un bus de *date* (8 sau 16 biți)
3. Un bus de *control* (4 sau mai multe semnale: READ', WRITE, RESET, etc.).

În general, toate dispozitivele din sistem sînt cuplate în paralel la busul de date și la cel puțin o parte din cel de adrese sau de control. Figura 10-1 prezintă busul unui sistem cu microprocesor. Microprocesorul este conectat la toate semnalele busului. Microprocesorul poate citi sau scrie din/în fiecare din dispozitivele cuplate la bus, *plasînd adresa dispozitivului pe busul de adrese* și transmițînd semnalele adecvate de control pe busul de control. Cînd procesorul scrie date într-un dispozitiv, el transmite datele pe busul de date însoțindu-le cu un impuls de WRITE (scriere) pe busul de control. Cînd procesorul citește datele aflate într-un dispozitiv, își dezactivează propriile drivere tristate de la busul de date, transmite un impuls de READ (citire) pe busul de control și urmărește datele plasate pe bus de către driverele tri-state ale dispozitivului exterior selectat. Celelalte dispozitive de pe bus pot fi ROM RAM, alte microprocesoare, sau alte circuite periferice programabile.

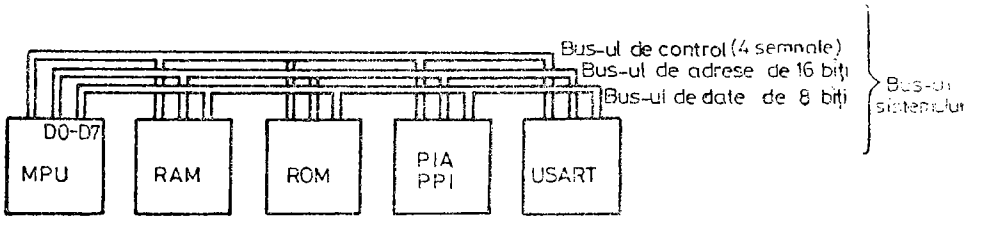


Fig. 10-1. Bus-ul unui sistem cu microprocesor.

10.2. CIRCUITE DE INTERFAȚĂ CU PERIFERICELE

În paginile care urmează se examinează tipurile de circuite LSI periferice ce realizează interfața microprocesorului cu lumea exterioară. Unele din aceste dispozitive sînt specializate, fiind dedicate interfeței cu un anumit periferic (de exemplu, un afișaj cu tub catodic, un disc flexibil sau o linie de comunicație serie). Cîteva din aceste dispozitive sînt atît de generale încît sînt realizate de multe ori pe același cip cu microprocesorul, RAM-ul și ROM-ul, formînd cipul *microcalculatorului.*

Chiar dacă un microcalculator folosește un bus intern pentru a interconecta circuite de acest tip pe cip, el este în general dotat și cu un bus extern, pentru extensie. Indiferent dacă dispozitivele sînt pe cip sau se cuplează la busul extern, funcționarea lor este în principiu aceeași.

În general, circuitele de interfață periferice conțin atît *registre de date* cît și *registre de control/stare* pe care procesorul le scrie sau le citește ca și cum ar fi locații de memorie (sau porturi de I/O). Registrele de control permit ca un singur circuit LSI să fie folosit într-o varietate largă de aplicații.

După alimentarea microcalculatorului programul începe în mod automat la o *locație de restart.* Prima acțiune, a unui program constă în *inițializarea* tuturor registrelor de date și control din sistem. Configurațiile de biți înscrise în registrele de control ale circuitelor LSI periferice *realizează în special adecvarea acestor circuite la sarcinile curente.*

De exemplu, biții din registrul de control al unui controler de afișaj pe tub catodic (de exemplu 6845) specifică fiecare detaliu din formatul ecranului inclusiv numărul de linii, numărul de caractere pe linie, mărimea caracterelor, tipul cursorului, întrețeserea și caracteristicile impulsului de sincronizare. În felul acesta, același cip de controler poate fi folosit de utilizatori cu o gamă largă de pretenții. Deoarece caracteristicile afișajului sînt determinate de biții din ROM-ul de program principal, schimbările de format de afișare pot fi făcute la fel de simplu ca orice modificare de program.

Această idee (adaptarea funcțiilor circuitului LSI prin biții registrului de control) constituie un alt exemplu pentru modul în care aceste circuite pot fi făcute să satisfacă o gamă largă de sarcini prin simpla programare. (Totuși această programare este foarte deosebită, în sensul că starea registrului de control lucrează ca o singură instrucțiune, foarte specială, ce se execută în permanență).

Registrele de control sînt de asemenea, modificate și *după* inițializare, în momentele în care se schimbă modul de lucru al circuitului în cursul funcționării sistemului. Poate fi necesar de exemplu, să se utilizeze un format diferit al afișării pe tubul catodic în timpul diferitelor moduri de funcționare ale sistemului.

10.2.1. Circuite de interfață paralel de uz general

În mod sigur, circuitul de interfață de intrare/ieșire cel mai utilizat este *interfața programabilă pentru periferice* (PPI — *Programmable Peripheral Interface*), denumită și *adaptor de interfață cu perifericele* (PIA — *Peripheral Interface Adapter*). Aceste circuite au 24 de pini de intrare/ieșire de uz general, care se cuplează la busul de date astfel încît semnalele de intrare și ieșire

pot fi tratate de către program ca biți din memorie (sau din porturile de I/O). Registrele de control de pe cip se pot inițializa prin programul de restart pentru a specifica care linii vor fi intrări și care anume ieșiri.

Cele mai puțin semnificative două linii ale busului de adrese sînt decodate de către cip, astfel încît cele 24 de linii de I/O apar ca trei bytes (octeți) succesivi în memorie iar registrul de control ca al patrulea. Adresa efectivă a acestor patru locații de memorie este determinată de logica externă de decodare a semnalelor.

Pentru a observa o linie de intrare, programul face lectura locației respective de memorie, apoi testează bitul care corespunde intrării de interes. Cînd se efectuează, prin program o scriere a uneia din cele trei locații de memorie, se actualizează toți cei opt biți ai registrului de ieșire. Dacă trebuie schimbată doar starea unui bit, programul poate transfera conținutul curent al registrului de ieșire în acumulator, unde se modifică bitul de interes; transferul este executat apoi în sens invers. În cazul unor circuite de interfață de uz general este posibilă setarea sau resetarea unui singur bit de ieșire, fără a mai fi nevoie să-i urmărim și pe ceilalți.

Unele microcomputere (de exemplu 8048) conțin un adaptor de interfață cu perifericele simplificat; ele folosesc un circuit *cvasibidirecțional*, care elimină necesitatea unui registru de control pentru a specifica dacă pinii sînt intrări sau ieșiri. *Toate* liniile pot fi atît intrări cît *și* ieșiri. Dacă o ieșire este comandată să treacă în 1, ea poate fi totuși ușor adusă în 0 de un semnal de intrare deoarece circuitul de ieșire poate furniza numai o fracțiune de mA, (ieșirea respectivă poate furniza mai mult cînd „urcă" pentru prima dată). Ieșirile adevărate arată ca o intrare prevăzută cu o rezistență mare spre sursa de alimentare.

10.2.2. **Alte dispozitive programabile de interfață**

Dacă viteza nu ar ridica probleme, toată interfața de intrare/ieșire ar putea fi realizată prin controlul, cu ajutorul programului, al liniilor de intrare/ieșire de uz general.

Apar însă adesea procese prea rapide pentru a fi manipulate prin program. În această situație, este necesară introducerea de logică externă pentru a realiza o prelucrare primară a intrărilor și o reducere suficientă a cerințelor la ieșire pentru ca microcalculatorul să poată lucra corespunzător. Din fericire, multe din funcțiile periferice rapid variabile sînt larg utilizate încît pentru a le manipula au fost realizate circuite LSI rapide.

În aplicațiile de serie mare, în care extinderea sistemului este improbabilă, cipurile periferice trebuie să fie utilizate numai în măsura în care dorim să folosim la maximum performanțele procesorului. Dimpotrivă, în sistemele produse în serie mică sau în care se prevede o extindere a sistemului, ele trebuie folosite ori de cîte ori sînt disponibile. Aceasta reduce costul cercetării și eliberează procesorul pentru necesități viitoare. În paginile care urmează se listează cîteva din circuitele periferice disponibile.

Receptoare/Transmițătoare serie. Ele includ circuite de tip UART (*Universal Asynchronous Receiver/Transmitter — receptor/transmițător asincron universal*), USART (la fel, dar *atît sincron cît și asincron*, vezi fig. 10-2), ACIA (*Asynchronous Communications Interface Adapter — adaptor de interfață pentru comunicații asincrone*), (PCI *Programmable Communications interface — interfață programabilă de comunicații*), etc. Ele primesc și trimit biții de date în format serie, conținînd și memoriile tampon necesare pentru ca microcalculatorul să realizeze numai scrieri sau citiri în registrele de date. Conținutul registrului de control definește lungimea cuvîntului, tipul de paritate, numărul biților de stop, cuvintele de sincronizare ș.a.m.d. Pe unele cipuri există și un registru de control al vitezei de transmisie. La viteze reduse de transmisie funcția de receptor/emițător poate fi realizată prin program (vezi fig. 8-1). Unele microcomputere (de exemplu 6801) conțin un receptor/emițător serie pe cip.

Controlere de comunicații serie. Aceste controlere realizează toate funcțiile descrise mai sus, ca și cele impuse de protocoalele de comunicație în „blocuri": SDLC (*synchronous data link control — controlul liniei de comunicație sincrone*) și HDLC (*high-level data link control — controlul de nivel înalt al liniei de comunicație*). Protocolul concret este indicat de biții registrului de

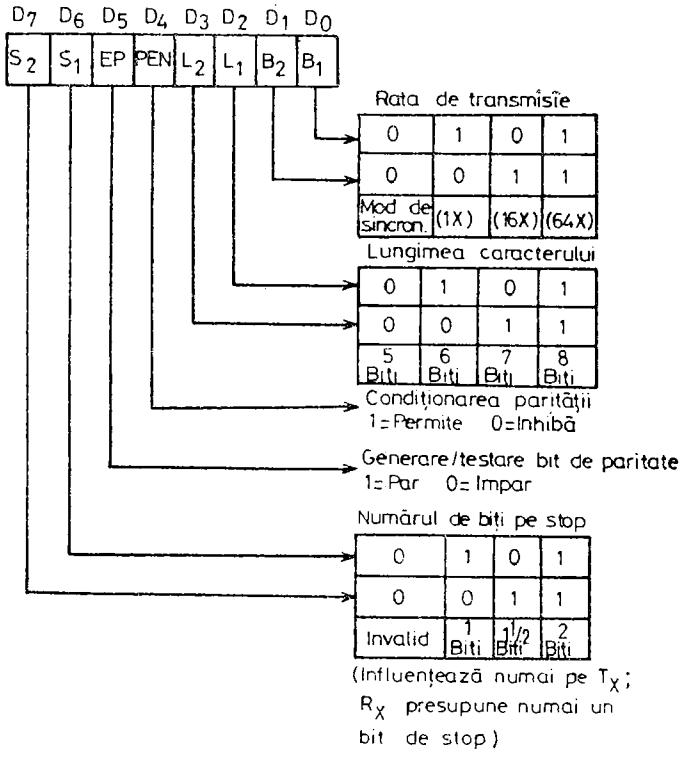


Fig. 10-2. Formatul instrucțiunii în registrul de control al unui USART (receptor — emițător universal sincron-asincron) funcționînd în modul sincron.

control, realizîndu-se de asemenea și funcții ca detecția de indicator (flag), sincronizarea, verificarea erorilor prin polinomul de 16 biți tip *CRC* (*cyclical redundancy checking — verificarea prin redundanță ciclică*); ca și introducerea/extragerea de zero. Dispunînd numai de un USART aceste chestiuni *s-ar putea* rezolva prin program (dar sînt neplăcute). De fapt, majoritatea logicii mai lente de pe aceste cipuri este programată.

Controlere de întreruperi (de exemplu 8259). Aceste cipuri, ce pot fi legate în cascadă, rezolvă fiecare opt intrări de întrerupere, pe bază de priorități. Fiecare întrerupere poate fi dezactivată în mod independent, modificînd bitul corespunzător din registrul de control. În mod automat, se generează și *vectorul* corespunzător, în sensul că fiecare întrerupere produce un salt la o adresă (vector) diferită din ROM. Majoritatea microcomputerelor și cîteva circuite de interfață paralelă de uz general includ două sau trei nivele de întreruperi vectorizate (dirijate).

Circuitele de temporizare/numărătoare de evenimente, numără tactele sistemului pentru a genera întreruperi temporizate (la intervale prestabilite). Pot, de asemenea, să numere diverse evenimente. Procesorul fixează conținutul numărătorului și recepționează o întrerupere în momentul în care numărătorul a ajuns în zero. Starea acestui contor poate fi citită în orice moment. Deoarece circuitul este simplu și pretinde numai un pin de intrare, este inclus pe majoritatea microcalculatoarelor.

Procesoare aritmetice (de exemplu AMD 9511). Realizează operații aritmetice în virgulă fixă și mobilă, în 16 sau 32 de biți, cu mare viteză. Operațiile includ înmulțirea, împărțirea, funcții logaritmice, trigonometrice, puteri, etc. Procesorul doar introduce operanzii în stiva cipului, scrie comanda și citește, ulterior, rezultatul. Pentru a accelera aceste transferuri poate fi utilizat cu DMA (direct memory access- acces direct la memorie).

Controlere de disc flexibil (floppy disck). Rezolvă toate detaliile de format, date, control de erori, adresare, etc. pentru blocul de comandă al discului. Asigură inițializarea registrelor, adaptează unii parametrii, ca viteza de comutare a pistei, timpul de stabilire a capului, timpul de încărcare a capului etc. Ca răspuns la comenzi plasate în registrul de control, execută (pe baza programelor sale interne) macro-comenzi cum sînt cele de citire sau scriere a unui singur sector sau a unui sector multiplu. Execută, dacă este necesar, transferul de tip DMA.

Circuite de interfață cu semnale analogice. Pînă la 6 intrări analogice pot fi convertite în eșantioane digitale printr-un cip de intrare specializat (de exemplu μA 9708). Prin utilizarea unor convertoare digital-analogice (de exemplu 5018), se pot produce semnale de ieșire analogice. Există cipuri de microcalculator complete (de exemplu 8022, produs de INTEL), care pe lîngă pinii uzuali de intrare și ieșire digitală au și intrări analogice. Semnalele analogice pot fi astfel filtrate, comparate, memorate sau supuse unei alte prelucrări numerice programate.

Interfețe universale cu perifericele (*universal peripheral interfaces* — UPI, vezi fig. 10-3). Reprezintă de fapt, un microcalculator, cu un registru special de interfață ce poate fi folosit de către procesorul central atît ca registru de control cît și ca registru de date. Utilizatorul programează acest microcomputer pentru a realiza orice funcție de interfață, definind, în același timp, formatele de date și control. Rezultatul este de fapt, un sistem multicomputer. Dar concepînd microcalculatoarele periferice drept controlere de interfață

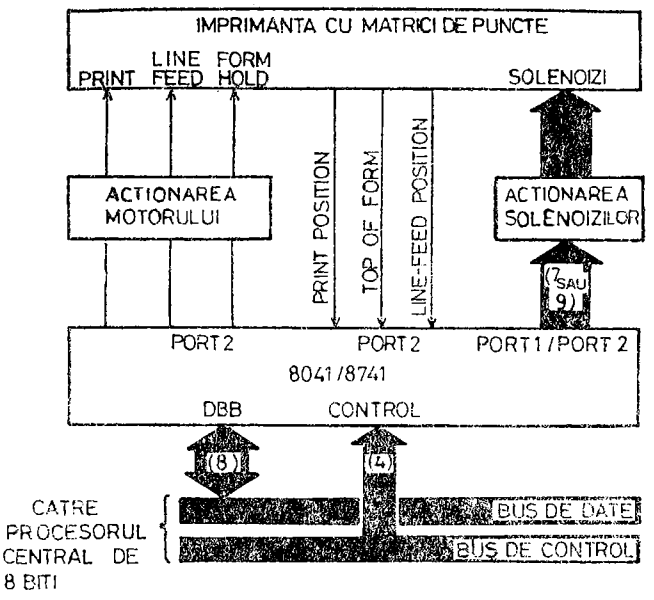


Fig. 10-3. Interfața programabilă universală (IPU) ce permite conectarea la o imprimantă. Microprocesorul central poate transfera ușor mesaje complexe către acest controler, și poate continua executarea altor teste în timp ce IPU controlează prin mesaje detaliate imprimarea în matrici de puncte. Deoarece IPU este un microcalculator complet, sistemul rezultat poate fi gîndit ca un sistem multicalculator.

specializate, sistemul este mai ușor înțeles, programat și testat. Unele produse vîndute drept controlere de interfață specializate (de exemplu, circuitul pentru cifrare de date 8294) sînt de fapt UPI cu un program special depus în ROM-ul realizat prin mascare.

10.2.3. Controlere pentru acces direct la memorie (direct memory access — DMA)

În mod normal, circuitele periferice precedente sînt pur și simplu conectate în paralel la busul principal al memoriei. De fiecare dată cînd dispun sau au nevoie de un cuvînt de date, ele trebuie „servite" de către program. Acest proces ia timp, deoarece programul trebuie să răspundă unei întreruperi, să introducă sau să extragă un cuvînt, să incrementeze un contor și să verifice dacă mai există „spațiu" pentru stocarea altor caractere.

Dispunînd de un circuit controler de DMA, programul este eliberat de această sarcină ce consumă timp. Programului îi rămîne doar să definească o zonă tampon pentru fiecare dispozitiv, iar restul activităţilor este realizat de către controlerul de DMA. De fiecare dată cînd dispozitivul periferic are

de transferat un nou cuvînt, controlerul de DMA intră în posesia busului, „reținînd" procesorul (trecut în stare „hold", în care ieșirile sale devin inactive), plasează adresa corectă de memorie pe busul de adrese, realizează ciclul de memorie, și apoi „eliberează" din nou procesorul. Pentru fiecare dispozitiv periferic, se menține un *contor de adresă curentă* ce este incrementat și comparat cu adresa finală la fiecare transfer. Fiecare controler de periferic transferă datele solicitate pe *conexiunile sale obișnuite la busul de date*, după ce *solicitarea* sa de DMA (*DMA request*) a fost *acceptată* (*DMA acknowledge*).

În general, un cip controler de DMA supervizează patru canale de DMA, cărora li se alocă anumite priorități ; fiecare canal este dotat cu propriul său contor și cu logică de cerere/acceptare de DMA separate. Se pot plasa în cascadă mai multe cipuri de DMA pentru a forma un lanț de priorități procesorul luînd-o întotdeauna pe ultima. Circuitul poate avea o opțiune prin care se selectează modul de prioritate „turnantă" în care cererile simultane de DMA sînt rezolvate prin rotație.

Creșterea performanțelor unui sistem ca urmare a introducerii unui controler de DMA este semnificativă doar dacă sînt necesare multe operații de I/O. De obicei, se rezolvă prin DMA numai dialogul cu dispozitivele rapide, ce operează transferuri în blocuri, în timp ce problemele dispozitivelor mai lente sînt rezolvate de programe de intrare/ieșire obișnuite. Pentru dispozitivele cu adevărat rapide, este adesea necesar să se utilizeze fie DMA fie o memorie tampon externă (sau ambele).

10.3. PROIECTAREA DECODORULUI DE ADRESE

Una din sarcinile de bază ale hardware-ului atașat unui microprocesor este decodarea adreselor astfel încît să se genereze un semnal unic de *selecție (chip select) pentru fiecare chip* de pe busul sistemului. Cînd microprocesorul intenționează să facă o operație de citire sau de scriere la o anumită locație de memorie, va trimite un impuls de READ sau WRITE pe busul de control, plasînd adresa corespunzătoare pe busul de adrese. Schema de decodare pentru selecția de cip trebuie să aleagă doar unul din cipurile de pe bus. Proiectantul de hard poate minimiza logica „chip select" printr-o alocare de adrese compatibilă cu o decodare simplă.

Punctul de pornire în alocarea adreselor trebuie să fie zonele de memorie din RAM și/sau ROM. Cînd la cuplarea surselor de alimentare procesorul este activat, el își inițializează contorul de program cu o adresă pe care o găsește prin citirea unei locații de memorie particulare. Deoarece adresa de restart trebuie stocată în ROM, acest fapt impune domeniul de adrese ce trebuie alocat ROM-ului. Dacă pentru stocarea programului este necesar mai mult decît un chip de ROM, generarea semnalelor de „chip select" trebuie făcută astfel încît să se realizeze domenii de memorie adiacente.

De exemplu, un ROM cu organizarea 2 k × 8 biți va decoda intern cei mai puțin semnificativi 11 biți ai busului de adrese (A0—A10) pentru a determina care cuvînt trebuie plasat pe busul de adrese, în momentul în care pri-

meşte semnalele de „chip select" şi de „read". Astfel, adresa celui de al doilea ROM diferă de a primului numai prin A11 (următorul bit de pe busul de adrese). Dacă sînt necesare mai mult de două ROM-uri, se poate folosi un decodor (de exemplu 8205, 74LS138) pentru a genera un total de opt semnale de selecţie de cip. Dacă intrările de adrese ale decodorului sînt conectate la A11, A12 şi A13 se pot cupla, la bus pînă la opt memorii de tip ROM, pentru un spaţiu de adrese continuu în limita a 16k bytes. Se observă că acest tip de decodare de memorie este similar în principiu cu arborele de decodare din figura 4-11, cu excepţia ultimilor 11 biţi ce sînt decodaţi în cip. Pentru a realiza decodarea completă a adreselor din ROM, putem lega biţii de adresă rămaşi (A14 şi A15) la intrările de activare (enable) ale decodorului.

Semnalele de selecţie pentru RAM (RWM) pot fi rezolvate în acelaşi mod. În general, aceste tipuri de memorii (RAM) au mai puţin decît opt pini de date, astfel încît un semnal de selecţie trebuie să comande mai mult decît un RAM. De exemplu, un RAM cu organizarea 1k × 4 bit este conectat la 10 biţi ai busului de adrese (A0—A9) şi numai la 4 biţi ai busului de date. Prin urmare, pentru a forma un byte de date se vor cupla două circuite, cu un semnal comun de selecţie.

O modalitate simplă de a urmări alocarea aderselor şi de a ne asigura că toate activările de cipuri sînt reciproc exclusive, este să le tabelăm, ca în Tabelul 10.1. Cît timp între două semnale de activare există cel puţin un bit diferenţă nu va apare un conflict de adrese. De exemplu, generarea semnalelor de activare a RAM şi a ROM solicită valori diferite ale lui A15. Activarea cipului periferic impune A14 = 1, în timp ce pentru RAM şi ROM, A14 = 0.

Tabelul 10-1. **Tabelul pentru decodificarea adreselor unui sistem cu 16 k-cuvinte de ROM, 8 k-cuvinte de RAM şi un cip de periferic**

Semnale pe BUS	A15 A14 A13 A12	A11 A10 A9 A8	A7 A6 A5 A4	A3 A2 A1 A0
ROM-uri		(A10———	———	———A0)
Decodificator de ROM	E1′ E2′ A2 A1	A0		
RAM-uri		(A9———	———	———A0)
Decodificator de RAM	E3 E2′ E1′ A2	A1 A0		
Cip periferic	CE			(RS1 RS0)

Adresele hexazecimale ale circuitelor pot fi citite prin divizarea tabelului în grupe de cîte 4 biţi (ca în tabelul 10-1). De exemplu, memoriile ROM acoperă toate cuvintele de adresă cu A14 = A15 = 0 sau adresele dintre 0000 şi 3FFF. Analog, memoriile RAM acoperă toate stările cu A15·A14′·A13′, adică o gamă cuprinsă între 8 000 şi 9 FFF. Adresele pentru cipurile periferice sînt decodate incomplet, deoarece dispunem de 32 768 de adrese distincte pentru

cazul cînd A14 = 1. Putem în mod arbitrar, să considerăm adresele dintre 4 000 şi 4 003 (hexazecimal) ca adrese de program pentru acest circuit. Să observăm că acest cip decodează intern A0 şi A1 pentru a adresa cele patru registre de date şi control ale sale. Programul poate astfel citi sau scrie în fiecare din aceste patru registre de parcă ar fi locaţii de memorie.

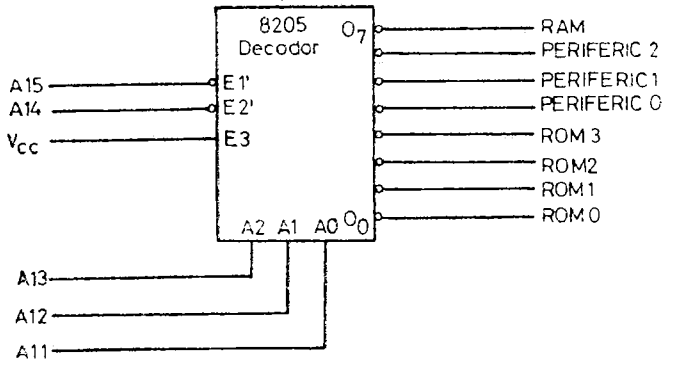
10.3.1. Decodarea adreselor în sistemele mai mici

Pe măsură ce sistemele devin mai mici, decodarea adreselor se poate simplifica progresiv prin decodarea incompletă a adreselor. Şi în acest caz regulile de bază sînt:

1. Toate semnalele de selecţie trebuie să fie reciproc exclusive.
2. Domeniul de ROM trebuie să fie continuu şi să includă adresa de restart.
3. Domeniul de RAM trebuie să fie continuu.

Figura 10-4 prezintă decodarea pentru un sistem ce are doar un cip de RAM. În felul acesta, RAM-ul şi circuitele de control ale perifericelor pot să fie selectate de alte ieşiri ale aceluiaşi decodor folosit pentru ROM.

Pentru sisteme şi mai mici, decodarea se poate realiza chiar şi fără decodor. Semnalele de selecţie sînt alocate pur şi simplu unor biţi separaţi de



A15	A14	A13	A12	A11	A10	Dispozitiv	Adrese în reprezentare hexazecimală
0	0	0	0	0	(A10	ROM 0	0 0 0 0
0	0	0	0	1	(A10	ROM 1	0 8 0 0
0	0	0	1	0	(A10	ROM 2	1 0 0 0
0	0	0	1	1	(A10	ROM 3	1 8 0 0
0	0	1	0	0	X	PERIFERIC 0	2 0 0 0
0	0	1	0	1	X	PERIFERIC 1	2 8 0 0
0	0	1	1	0	X	PERIFERIC 2	3 0 0 0
0	0	1	1	1	X	RAM	3 8 0 0

Fig. 10-4. Decodificator de adrese pentru sisteme mici.

adresă, ca în Tabelul 10-2. Circuitele periferice cu mai mult de o intrare de selecție (ca de exemplu Perif 1 și 2 în Tabelul 10-2) pot să facă ele însele o decodiare simplă fără alte porți.

Tabelul 10-2. **Decodificarea adreselor utilizînd numai semnalele de selecție ale cipurilor (Selecția liniară)**

	Biții de adresă											
Dispozitiv	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
ROM					CS′	(A10										A0)
RAM				CS			(A9									A0)
Perif 0			CS												(A1	A0)
Perif 1	CS′	CS												(A2		A0)
Perif 2	CS	CS′													(A1	A0)

Unele microprocesoare (de exemplu cele ale firmei Intel) au instrucțiuni separate de intrare/ieșire ce produc impulsuri pe busul de control distincte de cele corespunzătoare scrierii și citirii din memorie. Aceasta permite ca adresele din ultimii 256 bytes ai spațiului de memorie să fie utilizate atît pentru dispozitivele de intrare/ieșire ca și pentru memorie.

10.4. ÎNCĂRCAREA ȘI SEPARAREA ELECTRICĂ PE BUS

Deoarece dispozitivele MOS prezintă pe bus, în principal, o sarcină capacitivă numărul de circuite care pot fi cuplate în paralel la bus este limitat de sarcina capacitivă maximă a driverelor tristate. Specificațiile de timp (întîrzieri) sînt definite de obicei pentru o sarcină de 150 pF, dar se pot tolera sarcini de pînă la 300 pF dacă se ia în considerație o întîrziere suplimentară de 45 ns (în cazul unui 8085-Intel). Deoarece capacitatea de intrare a unui dispozitiv MOS, în cazul cel mai defavorabil, are adesea numai 5 pF, se pot cupla la bus, pînă la 30 de circuite la viteza normală, sau 60, la o viteză mai redusă !

Este probabil ca pe baza altor factori să nu se recomande cuplarea unui număr atît de mare de dispozitive în paralel. Una din probleme este existența a 60 de puncte virtuale de scurtcircuit pe bus, ceea ce conduce la o depanare dificilă. În cazul sistemelor mari, ce sînt cuprinse pe o singură placă de cablaj imprimat este suficient să se separe prin circuite „tampon" numai biții cei mai puțini semnificativi de adresă, care au încărcarea maximă. Se poate adesea reduce încărcarea pe busul de date doar prin separarea porțiunii corespunzătoare memoriei ROM pe busul respectiv. Avînd în vedere că în ROM nu se pot efectua scrieri, se pot folosi separatoare unidirecționale.

În cazul sistemelor mai mari se folosește o placă, drept fund de sertar, pentru a interconecta un grup de plăci cu cablaj imprimat. De obicei în acest caz se folosesc separatoare inversoare pentru toate ieșirile și intrările unei plăci. Deși semnalele pe fundul de sertar sînt inversate, ele sînt din nou inversate înainte de a fi folosite efectiv.

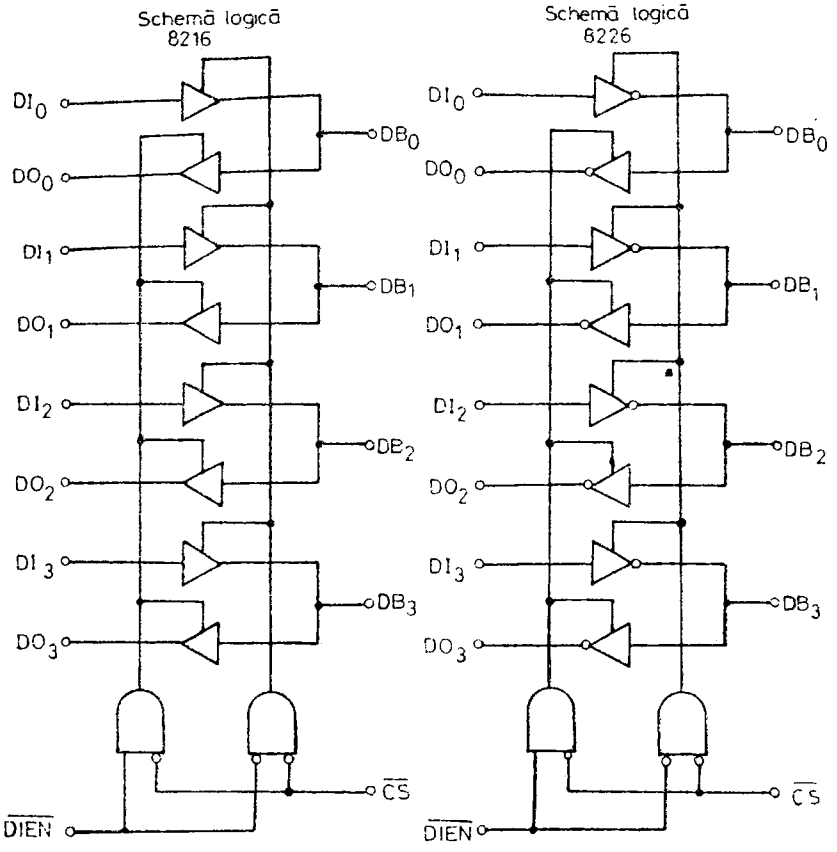


Fig. 10-5. Circuite de comandă bidirecțională pe bus.

Natura bidirecțională a busului de date face necesară utilizarea, în majoritatea cazurilor, a driverelor bidirecționale. Așa cum se indică în figura 10-5, acestor dispozitive le corespund două semnale de control. Semnalul de control al direcției de transfer (DIEN') este legat în general la un semnal de control al busului provenind de la microprocesor (de exemplu BUSEN'). Avînd în vedere că intrarea de selecție (CS') activează de fapt driverele tri-state, decodarea de adrese va trebui să furnizeze și acest semnal. Dacă un separator corespunde la mai multe dispozitive, *semnalul de selecție al separatorului va fi activat la selectarea oricărui dispozitiv din grupul respectiv.*

10.5. COMPATIBILITATEA ȘI TEMPORIZAREA PE BUS

Prin faptul că producătorii de microprocesoare oferă familii de circuite compatibile pe bus, proiectarea de sisteme care utilizează exclusiv circuite dintr-o familie devine foarte ușoară. Deși cea mai sigură soluție este să rămînem în cadrul acestor familii de componente reciproc compatibile, se întîmplă să constatăm uneori că alt producător oferă, de exemplu, un cip peri-

feric ce prezintă exact facilitățile dorite. De obicei, circuitele necesare pentru a face ca busul unui producător să arate ca al altuia sînt simple, dar specificațiile de temporizare trebuie verificate atent.

Semnalele corespunzătoare bus-ului de date și celui de adrese sînt standard, cu excepția faptului că unii producători (de exemplu INTEL, circuitul 8085) trimit biții de adresă pe bus-ul de date în timpul primei părți a ciclului de memorie. În acest caz, bus-ul separat de adrese se poate creia folosind un „registru tampon" (latch) pentru adrese, ce reține biții corespunzători cînd apar pe bus-ul de date.

Cele mai multe probleme de compatibilitate pe bus rezultă din diferențele care apar în ceea ce privește definiția și temporizarea semnalelor de control a busului. Deși, diferă în implementarea de detaliu, toate bus-urile de control ale microprocesoarelor includ următoarele semnale:

1. Un *impuls de write* — scriere *(write strobe)* care spune dispozitivului adresat să stocheze conținutul busului de date.
2. Un *impuls de read* — citire *(read strobe)* care spune dispozitivului adresat să-și pună datele pe bus.
3. Un semnal de *tact (clock)* ce este sincronizat cu semnalele procesorului.
4. Un semnal de *reset*, utilizat pentru a inițializa dispozitivele de pe bus atunci cînd se cuplează alimentarea.
5. O intrare de *ready* (*gata*) pentru a permite dispozitivelor lente să prelungească ciclul procesorului suficient de mult pentru a satisface cerințele lor de viteză.
6. O intrare de *hold (reține), halt (oprește)* sau DMA *request (cerere* de DMA) pentru a permite altor dispozitive să preia controlul bus-ului de la procesor.
7. (În mod obișnuit) O ieșire de *hold acknowledge (acceptare de hold)* pentru a permite procesorului să indice că a eliberat bus-ul.
8. O intrare de *interrupt request (cerere de întrerupere)* ce permite dispozitivelor externe să întrerupă fluxul programului pentru a rezolva o întrerupere. Sînt disponibile, uneori, mai multe intrări de întreruperi dirijate (vectored interrupt inputs).
9. O intrare de *întrerupere nemascabilă* sau *trapă* folosită pentru situații de urgență sau depanare.
10. O ieșire de *interrupt acknowledge (acceptare de întrerupere)* prin care procesorul semnalează că a acceptat o întrerupere.
11. Uneori apare și un semnal de *address latch enable (validarea latch-ului de adrese)* pentru a indica faptul că adresa este disponibilă pe bus-ul de date.

Deși producătorii realizează funcțiile listate în bus-ul lor de control, există o anumită variație în implementarea concretă a semnalelor. De exemplu, seriile MOTOROLA 6800 și MOS Technology 6500 dispun și de semnalele *read/write'* (R/W') și *valid memory address* (VMA, *adrese de memorie valide*). Impulsurile de *read* și *write* similare celor de la INTEL pot fi generate de un invertor și două porți NAND după cum urmează:

$$read = R/W' \cdot VMA \cdot ceas$$

$$write = (R'W') \cdot VMA \cdot ceas$$

Ori de cîte ori se folosesc dispozitive ce nu sînt compatibile (nu fac parte din aceeași familie) se vor compara specificațiile de temporizare ale dispozitivului periferic cu cele ale bus-ului procesorului. După cum se arată în

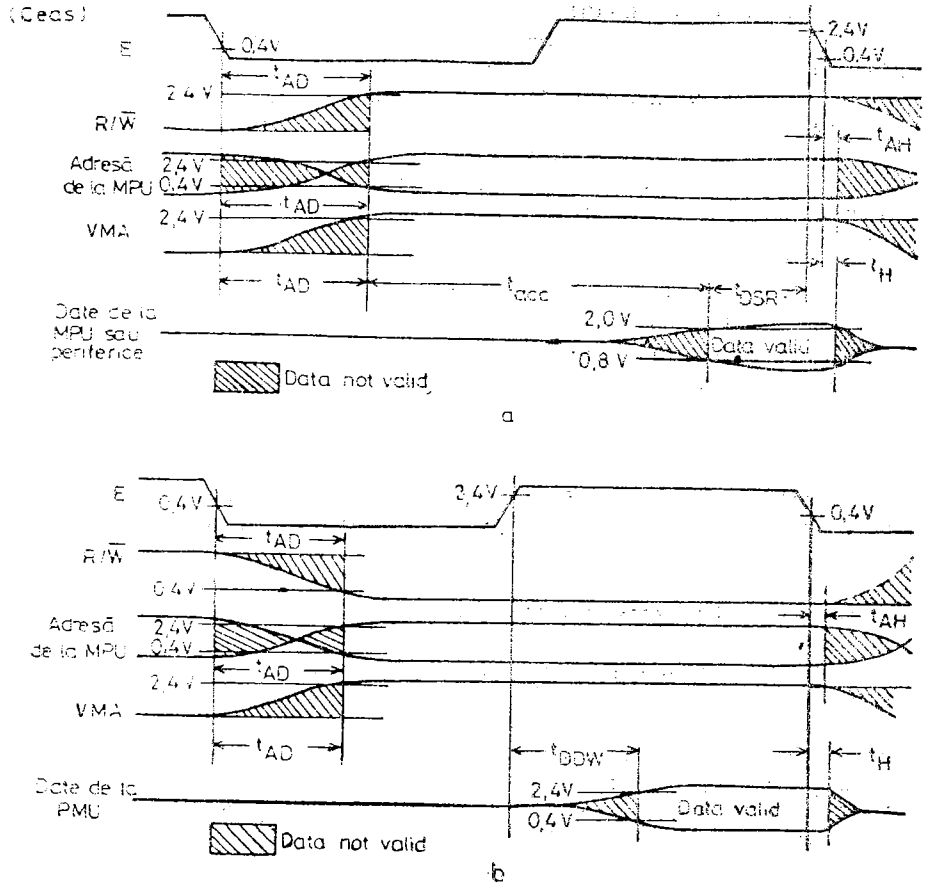


Fig. 10-6. Diagrame de timp ale ciclurilor de citire și scriere pe bus pentru microprocesorul MOTOROLA 6802. $t_{AD} < 270$ ns, $t_{ACE} < 230$ ns, $t_{DSR} > 100$ ns, $t_H > 20$ ns, $t_{AH} > 20$ ns, $t_{DDW} < 225$ ns. *(a)* Citirea datelor de la memorie sau periferice, *(b)* Scrierea datelor la memorie sau periferice.

figurile 10-6 și 10-7, aceste specificații pot fi destul de complicate. Se vor verifica datele de catalog, pentru fiecare tranziție în parte din ciclul de scriere, apoi pentru ciclul de citire și pentru ciclurile de control (dacă sînt specificate separat, ca în figura 10-7). Semnalul C/$\overline{D}$ (control/data) are valoarea 1 pentru ciclurile de control și 0 pentru ciclurile de date. În mod normal, el este conectat la una din liniile bus-ului de adrese. Deoarece 8251 are doar o adresă pentru registrul de date și una pentru registrul de control, se recomandă, de obicei, ca intrarea C/D să fie legată la bitul cel mai puțin semnificativ de adresă astfel încît cele două registre să-apară procesorului ca locații adiacente de memorie.

Produsele încadrate în „familia" unui microprocesor sînt proiectate astfel încît să fie evitate greșelile la interconectarea lor ; cu toate acestea, se recomandă prudență. Dacă, de exemplu, producătorul recomandă un singur nivel de porți în decodarea pentru „chip select", folosirea unei decodări pe mai multe nivele poate genera probleme de temporizare.

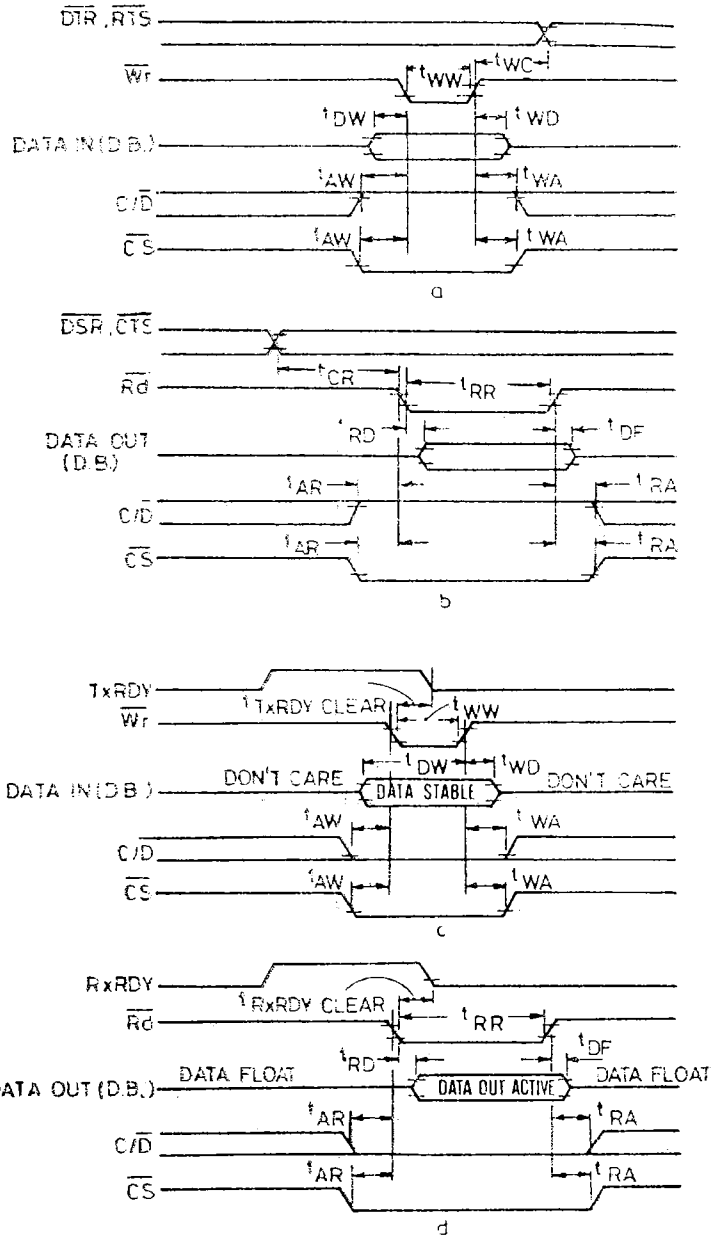


Fig. 10-7. Specificații de timp pentru interfața de comunicare (USART) tip Intel 8251A. $t_{WW} > 250$ ns, $t_{RXRDY\ CLEAR} < 150$ ns, $t_{DW} > 150$ ns, $t_{WD} > 0$ ns, $t_{HW} > 0$ ns, $t_{WA} > 0$ ns, $t_{RXRDY\ CLEAR} < 150$ ns, $t_{RR} > 150$ ns, $t_{RD} < 200$ ns, $t_{DF} = 10 \ldots 100$ ns, $t_{AR} > 0$ ns, $t_{RA} > 0$ ns, $t_{WW} > 150$ ns, $t_{WC} < 8$ ns, $t_{CR} < 20$ ns, T×RDY, R×RDY, DTR', RTS', DSR' și CTS' sînt semnale de comunicare cu exteriorul neasociate bus-ului. *(a)* Controlul scrierii (CPU → → USART). *(b)* Controlul citirii (CPU ← USART). *(c)* Ciclul de scriere a datelor (CPU ← USART). *(d)* Ciclul de citire a datelor (CPU ← USART).

10.6. ORIENTAREA PROIECTĂRII PENTRU ASIGURAREA FLEXIBILITĂȚII

Unul din avantajele cele mai mari ale proiectării centrate pe microprocesoare este flexibilitatea sa extremă. Astfel, un produs poate evolua, poate fi ușor specializat, cu condiția ca proiectarea de hard să asigure elasticitatea necesară. Deoarece pe măsură ce un sistem se rafinează și se extinde necesarul de memorie crește este bine să-l proiectăm pentru a putea controla de 2...4 ori mai multă memorie decît pare să dicteze concepția inițială. Avînd în vedere că cipurile de memorie nu trebuie să fie instalate dacă nu sînt necesare rezultă că procedînd în acest mod, costul produsului nu se majorează.

Uneori, posibilitatea de extindere a memoriei poate să fie inclusă în proiectare printr-o simplă pregătire pentru circuitele ce trebuie să apară în următoarea generație. La fiecare doi ani, (sau aproape la fiecare doi ani) capacitățile de memorie ale cipurilor se dublează. Se poate asigura o rezervă pentru circuitele din următoarea generație, prin includerea unor legături care permit să se mute intrările decodorului de selecție cip peste două locuri. În momentul în care noul cip devine disponibil, putem să mărim capacitatea de memorie cu un efort suplimentar minim. Configurația de pini a noilor circuite tinde să asigure continuitatea cu cele mai populare cipuri din vechea generație. De obicei, informațiile privind semnalele adiționale necesare (ca și alte schimbări) sînt disponibile cu mai mult de un an înainte ca noua componentă să fie produsă în serie mare.

Un alt mod de a îmbunătăți flexibilitatea este dotarea cu cîteva comutatoare de mod, ce pot să definească mai multe moduri de funcționare pentru un dispozitiv. Programul realizează lectura stării acestor comutatoare în faza de inițializare și decide care mod este impus în acel moment. În felul acesta, se pot proiecta mai multe versiuni ale aceluiași produs, identice, cu excepția pozițiilor comutatoarelor.

10.7. ORIENTAREA PROIECTĂRII PENTRU ASIGURAREA UȘURINȚEI ÎN DEPANARE ȘI A FIABILITĂȚII

Depanarea sistemelor cu microprocesoare poate fi foarte dificilă sau foarte simplă, în funcție de cîtă atenție se acordă problemelor de întreținere. În toate cazurile, cu excepția circuitelor celor mai simple (ușor „de aruncat"), microprocesorul și memoriile ROM trebuie să fie montate pe socluri. Soclul microprocesorului permite cuplarea sistemelor de testare și dezvoltare, iar soclurile destinate memoriilor ROM permit schimbări importante sau corecții minore printr-o simplă substituție a memoriilor. Dacă sistemul nu are o tastatură proprie, este necesar să prevedem un set de comutatoare în scopuri de test. De asemenea, dacă nu este dotat cu un afișaj pe tub catodic, poate fi utilă cuplarea a cel puțin cîtorva diode electro-luminescente (LED).

Cînd se aplică alimentarea, programul de restart poate începe prin aprinderea primului LED. Apoi programul poate face verificări de rutină ale RAM, $\overline{\text{ROM}}$ și controlerelor de periferice. Dacă programul afișează pe diodele electroluminescente coduri binare succesive, pe măsură ce trece prin diferite puncte

de test în procesul de autotest, starea diodelor electroluminescente va indica în ce măsură sistemul funcționează corect. Desigur, dacă sistemul are un afișaj al său propriu, programul de test poate să-l utilizeze. Menționăm faptul că funcționarea normală a sistemului începe numai după ce se trec toate auto-testele și că afișajul indică natura oricărui defect.

Testarea memoriilor fixe poate fi simplificată dacă ultimul cuvînt din spațiul memoriei ROM (oricum neutilizat, de obicei) conține un *cuvînt de verificare*. Acesta poate fi generat de o rutină simplă din sistemul de dezvoltare chiar înaintea scrierii în ROM. Se poate de exemplu, genera un caracter de verificare făcînd un SAU Exclusiv (EXOR) asupra fiecărei locații din ROM și complementînd rezultatul, ceea ce asigură existența unui număr impar de 1 asociat fiecărei poziții de bit. Prin urmare, rutina de test a ROM în sistem poate începe prin realizarea unui SAU Exclusiv asupra tuturor locațiilor din ROM și verificarea rezultatului ce trebuie să conțină numai 1. Această metodă simplă de verificare verticală a parității impare este destul de eficace ; se pot folosi însă și polinoame de verificare mai sofisticate (vezi bibliografia) care dau rezultate mai bune.

Testele de start pentru RAM pot fi, de asemenea eficace, fără a fi laborioase. Iată, de exemplu, o secvență simplă de test a RAM :

1. Scrieți în fiecare locație de RAM propria sa adresă.
2. Citiți fiecare locație și comparați-o cu adresa proprie.
3. Repetați 1 și 2, dar folosiți complementul adresei proprii.

Din păcate există o mulțime de moduri incorecte de funcționare care la pornirea sistemului nu sînt semnalizate de aprinderea primului LED. O astfel de situație poate fi, de exemplu, următoarea. Se presupune că alimentarea și ceasul sînt corecte (lucrează) și că bus-urile de date și de adrese sînt funcționale. Întreruperea de restart face ca microprocesorul să citească adresa de restart dintr-o locație particulară în ROM. Pentru ca aceasta să aibă loc, trebuie ca activarea (enable) cipului ROM să lucreze și ca biții de adresă să fie trimiși corect spre ROM. Deoarece adresa de start este trimisă de ROM spre procesor pe bus-ul de date, orice defecțiune a acestuia din urmă produce un start al procesorului de la o adresă greșită.

Deoarece există atît de multe posibilități ca procesorul să nu pornească corect, este adesea util să se introducă facilitatea ca procesorul să poată „funcționa liber" (free run). În acest mod, procesorul generează secvențial toate adresele posibile. Această secvență de numărare binară pe bus-ul de adrese face posibilă găsirea de defecte pe bus-ul de adrese sau în decodarea de selecție a cip-urilor. Orice microprocesor va funcționa liber dacă este forțat să execute în permanență aceeași instrucțiune (ce nu conține referiri la memorie ori ramificații). Acest lucru se obține prin forțarea bus-ului de date într-o stare în care instrucțiunea ce urmează să fie citită din memorie este întotdeauna aceeași. De exemplu, dacă pe bus-ul de date este forțat codul instrucțiunii CLEAR A, procesorul va șterge registrul *A* (intern) și apoi va încerca să citească următoarea instrucțiune din următoarea locație din ROM. Acest proces se repetă fără sfîrșit obținîndu-se o incrementare continuă a adresei de pe bus-ul de adrese.

În cazul unui sistem redus, fără separatoare pe bus-ul de date, funcționarea liberă poate fi impusă prin simpla introducere într-unul din soclurile pentru ROM a unui set de rezistențe de 100 Ω, legate fie la +5V, fie la masă, cu configurația necesară pentru a forța instrucțiunea CLEAR A. În siste-

mele dotate cu separatoare pe busul de date, separatoarele vor fi montate pe socluri, pentru a putea fi scoase și înlocuite cu un set similar cu rezistențe, de „funcționare liberă".

În momentul în care funcțiile de bază ale sistemului lucrează, programul de depanare poate fi oricît de sofisticat. Dacă sistemul include un afișaj, programele de auto-test pot chiar să descrie direct defectul. Dacă depanatorului îi este accesibilă o intrare de TRAP sau *întrerupere nemascabilă*, se poate include un program care afișează contorul de program și starea registrelor, ori de cîte ori se apasă un buton, chiar dacă programul este „agățat" într-o buclă. Programele de *depanare* ce sînt disponibile de obicei în ROM, la producătorii de microprocesoare, fac posibilă utilizarea oricărui terminal serie pentru testare. În felul acesta putem urmări ori schimba conținutul oricărei locații de memorie sau al oricărui registru din sistem.

10.8. PROIECTAREA PENTRU A REALIZA AUTOINIȚIALIZAREA

În sistemele ce trebuie să funcționeze continuu timp de luni sau ani fără supraveghere, problema stărilor de blocare este serioasă. Atît timp cît totul lucrează perfect nu sînt probleme, dar un singur bit afectat de zgomot poate să facă programul „să sară în aer". Dacă aceasta se întîmplă, sistemul nu-și va reveni fără o intervenție umană, cu excepția cazului în care se prevăd dotări speciale în hardware.

Cel mai sigur mod de a face un sistem să-și revină singur după o eroare este includerea unui circuit de temporizare de tip „cîine de pază". Un astfel de circuit nu este nimic mai mult decît un monostabil retrigerabil, comandat de un segment important al programului. Dacă programul «explodează» și nu mai declanșează circuitul monostabil, ieșirea acestuia acționează asupra intrării de TRAP sau întrerupere nemascabilă (NMI) a microprocesorului. Programul ce se execută în continuare trebuie să memoreze apariția defectului și apoi să transfere controlul programului de restart.

O metodă mai simplă, dar mai puțin eficace, este acționarea TRAP sau NMI (non maskable interrupt), ori de cîte ori apare o adresă nedefinită, sau cînd se fac încercări de scriere în ROM. Această metodă se bazează pe ideea că atunci cînd programul deraiază apare o dezordine generală, pe măsură ce date oarecare sînt privite ca instrucțiuni (ce sînt executate), indicatoarele de stivă sînt distruse generîndu-se o stare de haos total. Undeva, în acest proces, se pot forma adrese nedefinite sau se pot face încercări de scriere în ROM.

O altă variantă de reinițializare automată face apel la includerea unor verificări extensive de date în program. Ori de cîte ori apar date sau adrese incorecte, programul sare la locația de TRAP. Nici una din cele trei metode utilizate nu este absolut sigură, astfel că adesea ele sînt folosite împreună pentru a îmbunătăți șansele de prindere a procesorului care „calcă pe alături". Deoarece starea de blocare într-o buclă disfuncțională poate fi o indicație a unor greșeli de program sau a unor defecțiuni în stare incipientă în hardware, trebuie prevăzute unele mijloace de a înregistra apariția ei și a oricăror detalii ce fac depanarea mai simplă. Dacă este posibil, este bine să stocăm undeva o „fotografie" a stării contorului de program, a indicatorului de stivă, a registrelor și a dispozitivelor periferice.

10.9. DOCUMENTAȚIA DE HARDWARE A MICROCALCULATORULUI

Avînd în vedere natura repetitivă a conexiunilor de bus ale microprocesorului, diagramele pot fi considerabil simplificate utilizînd cîteva prescurtări. Proiectarea sistemului poate fi mult ușurată dacă se realizează un tabel al conexiunilor dintre pinii de bus, cum este Tabelul 10-3. Acesta definește toate legăturile la pinii de *bus* și *sursă de alimentare* într-un mod clar organizat. Pentru descrierea conexiunilor și a logicii se poate utiliza o diagramă logică standard. Această diagramă și tabelul de pini ai busului definesc împreună sistemul.

Deoarece numărul de ordine ai pinilor circuitelor integrate poate fi copiat direct din cataloage în tabel, efortul și probabilitatea de eroare se reduc considerabil. De asemenea, structura decodării este mai clară în această formă. Eliberînd schema logică de aglomerația produsă de conexiunile repetitive, celelalte legături logice se pot înțelege mai ușor.

Deși proiectarea este mai simplă și mai puțin complicată cu documentația separată în două formate, poate fi recomandabil să cerem desenatorului să indice numerele pinilor de pe tabela de pini a busului și pe schema logică, plasînd alături și numele (încercuite) ale semnalelor de pe bus.

Aceasta scutește depanatorul să urmărească semnalele pe un tabelul separat. Aglomerația poate fi redusă și mai mult, indicînd numai numele (încercuite) ale semnalelor, nu și conexiunile reale.

Tabelul 10-3. **Tabel al conexiunilor la BUS. Prin definirea conexiunilor repetative se simplifică reprezentarea schemelor logice**

Dispozitiv	Numărul bitului																								R/W	GND	+5V	Ø 2	IRQ'
	Bus de date								Bus de adrese																				
	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0					
MPU 6800	26	27	28	29	30	31	32	33	25	24	23	22	20	19	18	17	16	15	14	13	12	11	10	9	34	1, 21, 39	8,2		4
2-PIA 6820	26	27	28	29	30	31	32	33	24														35	36	21	1	20	25	37, 38
ACIA 6850	15	16	17	18	19	20	21	22	10															11	13	1	12	14	7
RAM 4804	5	3	8	6											13	12	16	17	1	2	15	14	10	11	7		9	18	
RAM 4804					5	3	8	6							13	12	16	17	1	2	15	14	10	11	7		9	19	
2-PROM 2716	17	16	15	14	13	11	10	9						19	22	23	1	2	3	4	5	6	7	8			12, 18	24	
Decodor 74LS138											3	2	1														4, 5, 8	16	

10.10. MEMORII DINAMICE

În majoritatea sistemelor mici cu microcalculatoare este mai practică utilizarea unor cipuri RAM *statice*. În general, bistabilele folosite în astfel de memorii pentru fiecare bit conțin patru tranzistoare și două rezistențe. Deși proiectarea inteligentă a circuitului reduce în mod serios dimensiunea celulei statice de memorie, totuși memoriile de mare capacitate sînt realizate mai economic în varianta *dinamică*.

Cipurile de memorie dinamică folosesc o celulă cu un singur tranzistor pentru fiecare bit și memorează datele ca sarcini electrice stocate în capacități de aproximativ 0,03 pF*. Aceasta permite concentrarea de patru ori mai mulți biți în cazul memoriei dinamice RAM decît în cazul celei statice, la aceeași dimensiune a cipului. De asemenea, disipația de putere este redusă considerabil în cazul memoriei dinamice, fiindcă cipurile neselectate consumă foarte puțin curent. Dezavantajul prezentat de memoriile RAM dinamice constă în necesitatea *regenerării* (împrospătării) — refresh — periodice a sarcinii stocate pe acești condensatori foarte mici pentru a o împiedica să dispară treptat.

Ori de cîte ori are loc un ciclu de scriere sau citire se regenerează de către circuitele de pe cip un întreg *rînd* de biți. Deoarece cipul este organizat ca o matrice X—Y de rînduri și coloane, *logica externă trebuie să asigure că fiecare rînd de pe cip este adresat odată la cel puțin 2 ms*. Această funcție este realizată automat prin microcod în cazul unor microprocesoare (de ex. cele produse de firma ZILOG). De asemenea ea poate fi realizată în mod inerent în unele aplicații (ca afișajele pe tub catodic), în care are loc oricum o citire periodică a blocurilor de date.

Ceea ce este important este faptul că în cursul intervalului de regenerare trebuie să se realizeze toate combinațiile binare ale ultimilor șase sau șapte biți de adrese. Cînd împrospătarea (refresh-ul) se realizează ca o funcție separată, ea poate fi realizată în *modul „salvă"* (burst mode), în care se oprește secvența normală de program și sînt regenerate 64 (sau 128) de locații consecutive. Regenerarea grupată poate fi realizată de program ca răspuns la o întrerupere cu perioada de 2 ms. Totuși, dacă există alte întreruperi ce solicită o rezolvare rapidă, această pauză poate fi intolerabilă.

Regenerarea distribuită se aplică odată unui singur rînd și nu implică mai mult de un ciclu de memorie. În general aceasta necesită circuite suplimentare deoarece ciclurile apar frecvent. De exemplu, pentru ca în 2 ms să realizăm 128 de cicluri de un rînd, fiecare ciclu de refresh trebuie să apară la $2\,000/128 = 15{,}6$ μs. S-au creat cipuri adiționale (vezi fig. 10-8) pentru a simplifica această sarcină.

Deoarece memoriile dinamice sînt optimizate pentru construirea memoriilor de mare capacitate, ele au, în general intrări de adrese multiplexate, pentru a reduce numărul de conexiuni și dimensiunea capsulei. În timpul primei părți a unui ciclu normal de scriere sau citire, jumătatea cea mai puțin semnificativă a adresei de rînd este stocată într-un latch al cipului RAM, prin acțiunea frontului anterior al *impulsului RAS* (*Row Address Strobe* — impuls de citire adrese de linie). În continuare, pe aceiași pini se aduc biții cei mai

* Deși capacitatea poate să pară nu prea mare, sarcina reprezentată de un singur bit mai totalizează încă aproximativ 3 000 000 de electroni!

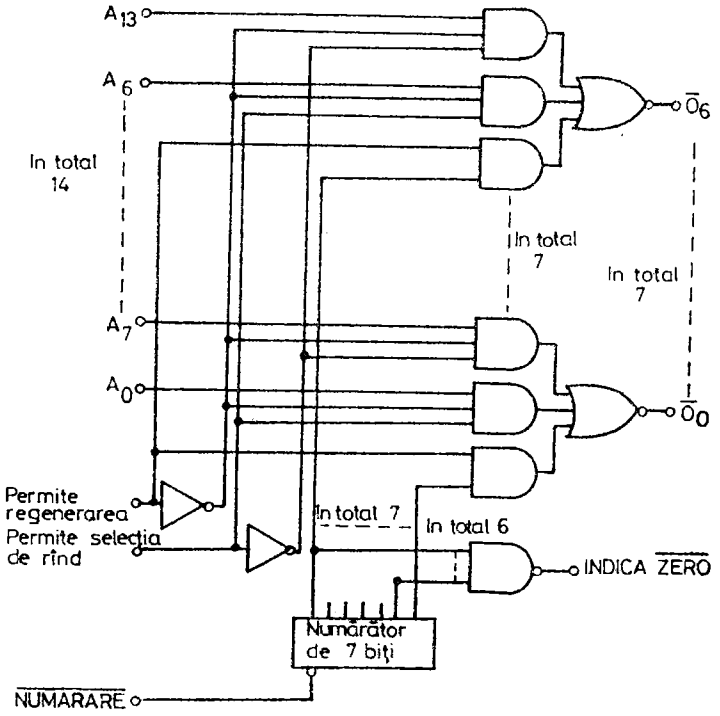


Fig. 10-8. Circuit adițional pentru memorii de tip RAM dinamic (Intel 3242).

semnificativi fiind eșantionați prin acțiunea CAS (*Column Address Strobe —impuls de adrese a coloanei*). În felul acesta, pinii de adresă au un rol dublu.

Cipul auxiliar pentru RAM din figura 10-8 include un selector cu 3 intrări care transmite adresa de linii, adresa de coloană sau conținutul unui numărător de regenerare pe liniile de adrese. Ieșirile sale pot suporta 250 pF (adică aproximativ 50 cipuri RAM). Remarcați că regenerarea poate fi realizată fără a intra într-un ciclu complet de memorie sau chiar fără selecția cipurilor RAM.

Alte tipuri de memorie dinamică asigură o stocare a datelor și mai economică, sacrificînd viteza sau simplitatea adresării. Dispozitivele CCD (charge-coupled devices — dispozitive cu cuplaj prin sarcină) și memoriile cu bule magnetice au un acces serie la date — ca un registru de deplasare lung. Așa cum vom vedea în următorul capitol, aceasta adaugă adresei o altă dimensiune : timpul.

EXERCIȚII

1. Proiectați (numai schema hardware) un terminal, bazat pe un microcalculator, cu 16 taste organizate într-o matrice $X-Y$ (vezi fig. 13-23), un afișaj de 16 digiți realizați cu elemente de afișaj cu 7 segmente cu diode luminescente (vezi fig. 13-25) și interfață serială asincronă de intrare-ieșire.

2. Ce tip de cip controler de periferic trebuie adăugat la un microcomputer realizat cu microprocesorul 8900 dacă acesta se dovedește prea lent pentru a controla transferul serial al datelor la 9600 de baud?
3. Utilizînd figura 10-2 indicați conținutul în hexazecimal al cuvîntului de control pentru:
   (*a*) Doi biți de stop, imparitate, cinci biți pe caracter, divizare cu 16 a ceasului.
   (*b*) Un bit de stop, imparitate, opt biți pe caracter, divizare cu 64 a ceasului.
4. Care sînt adresele hexazecimale ale celor cinci dispozitive din Tabelul 10-2? (Considerați biții cei mai puțini semnificativi ca specificînd adresa).
5. Adăugînd numai un decodificator cu opt ieșiri, modificați logica de decodare din figura 10-2 astfel încît să poată fi comandate încă șapte dispozitive periferice adiționale.
6. (*a*) Proiectați decodificarea adreselor pentru un microcalculator astfel încît să poată fi utilizate patru memorii ROM de 2k×8, biți, trei memorii RAM de 2k×4 biți și opt controlere de periferice. (Numărul de capsule cu circuite integrate va fi de două). Presupunem că adresa de restart este 2C (în hexazecimal) — Intel.
   (*b*) Generați o tabelă cu adresele tuturor dispozitivelor în hexazecimal. Considerați numai adresa cea mai mică atunci cînd sînt disponibile mai multe.
   (*c*) Reproiectați decodificarea pentru microprocesoarele Motorola ce au ca adresă de restart pe $FFFF_{16}$. Utilizați avantajul oferit de microprocesorul 6802 de a adresa mai rapid în pagina de bază (primii 256 octeți), folosind adresa controlerelor de periferice în această pagină deasupra adresei 128.
7. Proiectați semnalul de selecție al circuitelor de izolare a memoriei ROM (buffer) de bus-ul sistemului reprezentat în figura 10-2.
8. (*a*) Proiectați logica de conectare a opt interfețe de comunicație tip Intel 8251A (figura 10-7) la bus-ul unui microprocesor tip Motorola 6802 (figura 10-6). Verificați specificațiile temporale pentru a asigura compatibilitatea și adăugați circuite logice (acolo unde este necesar) pentru a le face compatibile.
   (*b*) Care este întîrzierea, maximă admisibilă, prin decodificatorul ce asigură semnalele de selecție a cipurilor în această combinație?
   (*c*) Care este întîrzierea maximă admisibilă prin porțile ce generează semnalele Wr' și Rd'?
   (*d*) Presupunînd o creștere a întîrzierii semnalului de 0,5 ns/pF datorată sarcinilor capacitive mai mari de 150 pF, care poate fi sarcina capacitivă totală ce ar determina atingerea limitelor de operare corectă a sistemului? Considerați că întîrzierea maximă a porților utilizate este de 32 ns.
9. Scrieți un program de test simplu pentru memoriile RAM:
   (*a*) în limbajul de asamblare al microprocesorului 8900.
   (*b*) în limbajul de asamblare al microprocesorului 8080.
10. Scrieți un program de generare a cuvîntului de paritate pentru un ROM și un program de testare a parității pentru o astfel de memorie:
   (*a*) în limbajul de asamblare al microprocesorului 8900.
   (*b*) în limbajul de asamblare al microprocesorului 8080.

BIBLIOGRAFIE

Detectarea erorilor

Vasa, Suresh, „Calculating an Error Checking Character in Software", *Computer Design*, May 1976.

Parametrii de Bus

Force, Gordon, „Microprocessor Bus Standard Could Cure Designers Woes", *Electronics*, July 20, 1978.

Interfață universală pentru periferice

Philips, Don și Goodman, „Slave Microcomputer Lightens Main Mierorpocessor Load", *Electronics*, July 7, 1977, pp. 109–112.

Smith, Lionel, *Printer Control with the UPI-41*, Intel, Santa Clara, Calif., 1977, Appendix Note # AP-27.

Circuite integrate pentru periferice

Metzger, Jerry, „Peripheral IC's for Microprocessors", *Electronics Products*, April 1978, pag. 69—72.

Cip pentru controlul periferiecelor

Weissberger, Alan, „Data-link Control Chips: Bringing Order to Data Protocols", *Electronics* June 8, 1978, pag. 104—111.

Următoarele note de aplicații de la Intel oferă informații detaliate despre unele cipuri. Ele se pot procura la prețul de $ 1 fiecare de la Intel Corporation, 3065 Bowers Ave., Santa Clara, Calif. 95051.

AP-15 8255A *Programmable Peripheral Interface Applications.*
AP-16 *Using the 8251 USART.*
AP-31 *Using the 8259 Programmable Interrupt Controller.*
AP-29 *Using the 8085 Serial I/O Lines.*
AP-25 *Simplify Your Dynamic RAM/Microprocessor Interface.*
UPI-41 *Users Manual Pub.* 9800504A ($5).
Memory Design Handbook, 1977 ($5).

Cipuri de memorie

Hnatek, Eugene, „Current Semiconductor Memories", *Computer Design*, April 1978, pag. 115—126.

Microcalculatoare cu intrare analogică

Ittner, Wiliam și Miller, Jeffrey A., „Microcomputer's On-Chip Functions Ease Users Programming Chores", *Electronics*, July 20, 1978, pag. 129—133. Vezi și *Electronics*, May 25, 1978, pag. 122.

Proiectarea hardware și software a cipurilor periferice

Motorola Semiconductor, *Microprocessor Applications Manual*, McGraw-Hill, New York, 1975.

Memorii RAM dinamice

Coker, Derrell, „16-k RAM Eases Memory Design for Mainframes Microcomputers", *Electronics*, April 28, 1977, pag. 115—120.
Brown, J. Reese Jr., „Timming Peculiarities of Multiplexed RAMs", *Computer Design*, July 1977, pag. 85—92.

11

DIMENSIUNEA TIMP

11.1. REPETAREA CIRCUITELOR ÎN TIMP

Unul din motivele pentru care cu atît de puțin hardware putem realiza atît de multe în varianta programată este faptul *că folosim aceleași circuite, la momente diferite, pentru rezolvarea unor probleme diferite.* În acest capitol vom examina alte metode de utilizare multiplă a hard-ului, prin folosirea *dimensiunii timp.*

Atunci cînd transmitem opt semnale pe opt fire distincte, avem de-a face cu *distribuirea* informației *în spațiu.* Dacă definim *opt intervale de timp* consecutive, putem să transmitem aceleași semnale, unul cîte unul, pe un *singur fir.* Aceasta înseamnă informație *distribuită în timp.* Într-un interval sincron fiecare tact va defini începutul unui nou interval de timp, iar fiecare al optulea tact este echivalent unui singur tact din cazul distribuirii în spațiu. Putem transmite cele opt semnale în diferite moduri, în funcție de felul în care alegem dimensiunile timp și spațiu. De exemplu, am putea utiliza două conductoare și patru intervale de timp sau patru conductoare și două intervale de timp. În acest fel cele două dimensiuni, timpul și spațiul, pot fi manevrate pentru a obține o situație de compromis convenabilă aplicației.

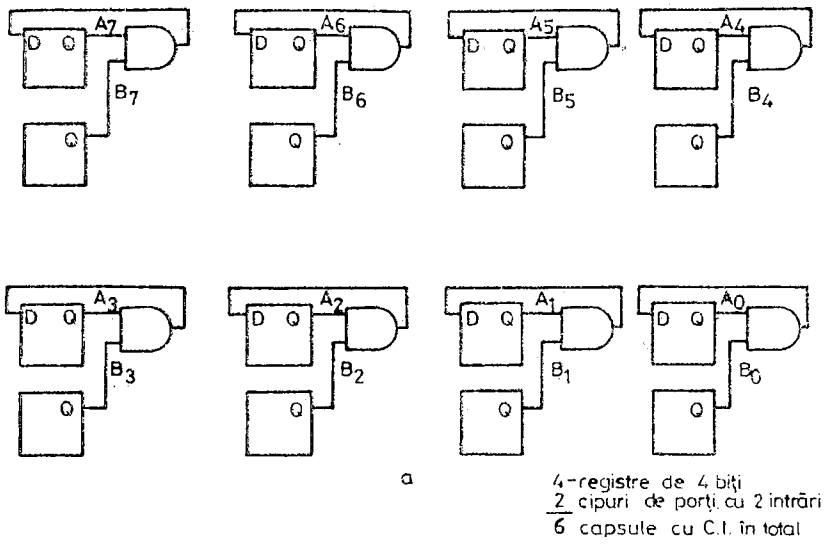
Putem utiliza dimensiunea timp și în cazul porților logice. *O poartă* AND, *de exemplu, poate fi echivalentă cu opt porți* AND *separate* dacă intrările sînt definite ca opt semnale diferite pe opt intervale de timp. Ieșirea porții va reprezenta AND-ul intrărilor din respectivul interval. De fapt, orice logică combinațională poate realiza funcția multiplicării cu n a aceluiași circuit, dacă divizăm timpul în n intervale. Avînd în vedere că adesea sînt necesare structuri logice repetitive, putem de multe ori să reducem cantitatea de logică necesară pentru a realiza o funcție prin *repetarea structurii în timp în loc de repetare în spațiu.*

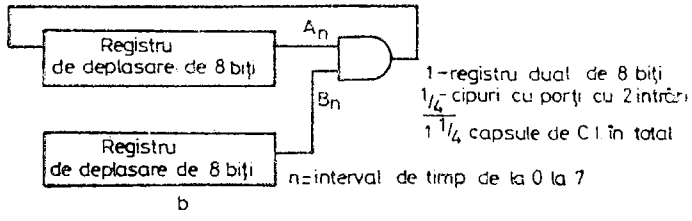
Desigur că doar un bistabil *nu poate* să realizeze ceea ce fac opt bistabili, prin simpla definire a opt intervale de timp. Totuși, un registru de deplasare de opt biți *este* echivalent cu un bistabil D lucrînd pe opt intervale de timp. La fiecare tact, intrarea registrului de deplasare este stocată într-un bistabil și va apare la ieșire din nou, opt tacte mai tîrziu.

Un registru de deplasare poate fi astfel considerat ca un bistabil D *distribuit în timp.* Ca în cazul bistabilului D, ieșirea la tactul $n + 1$ egalează intrarea din timpul tactului n. Singura diferență este faptul că între tactele n și $n + 1$ corespunzînd aceluiași interval de timp, apar alte șapte impulsuri de tact, fiecare corespunzînd unuia din celelalte șapte intervale.

11.2. FUNCȚIONAREA SERIE

Figura 11-1 prezintă trei modalități posibile de a face AND între două numere de 8 biți ce sînt stocate în registre și a depune rezultatul într-unul din registre. În figura 11-1*a* porțile se repetă în spațiu, în sensul că sînt 8 porți, identice, separate. În figura 11-1 *b* se folosește dimensiunea timp





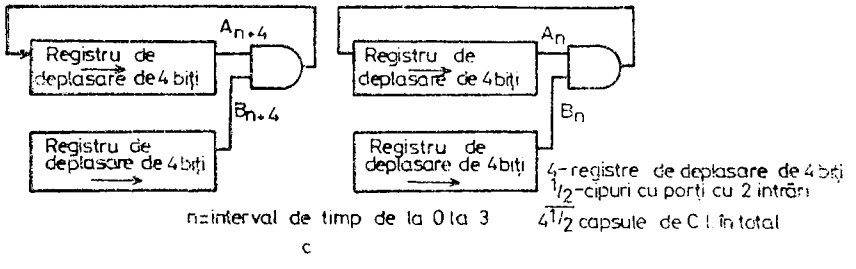


Fig. 11-1. Trei modalități de a realiza funcția AND între conținutul a două registre de 8 biți : *(a)* structură distribuită în spațiu ; *(b)* distribuire în timp (opt intervale de timp) ; *(c)* distribuire în spațiu și timp (2×4).

pentru a utiliza aceeaşi poartă în toate cele 8 intervale. Figura 11-1*c* prezintă o versiune combinată, în care se folosesc ambele dimensiuni : timp şi spaţiu. Varianta (*b*) este denumită, de obicei, *funcţionare serie*, deoarece biţii sînt manipulaţi unul cîte unul. Ea reprezintă, în mod sigur, varianta cea mai economică, deoarece sînt necesare numai o poartă şi două registre de deplasare de 8 biţi (ce sînt disponibile în aceeaşi capsulă, de exemplu 9328). Numărul total de cipuri este astfel de 1,25 în comparaţie cu 6 pentru varianta (*a*) şi 4,5 pentru varianta (*c*). Deşi varianta (*b*) este, de departe, cea mai economică, trebuie observat că sînt necesare opt tacte pentru a realiza operaţia, ceea ce înseamnă că viteza maximă este de o optime din cea a variantei paralel (vezi figura 11-1*a*). În multe cazuri, nu este necesar să efectuăm operaţia la viteza maximă, astfel încît putem efectua o economie importantă de hardware utilizînd dimensiunea timp (funcţionare serie).

După cum arată figura 11-1*c*, putem utiliza orice combinaţie a dimensiunilor timp şi spaţiu. Acest mod de lucru constituie aşa-numita *funcţionare serie-paralel*. Dacă utilizarea a opt intervale de timp face ca operaţia să dureze prea mult, varianta (*c*) reduce timpul necesar la jumătate (patru intervale), economisind totuşi hardware în comparaţie cu varianta paralelă (*a*). În felul acesta, putem realiza organizarea dimensiunilor timp şi spaţiu ale logicii repetitive pentru a utiliza la maximum viteza logicii utilizate.

Ori de cîte ori circuitele logice lucrează la mai puţin decît viteza lor maximă, se reduce eficienţa. Într-un sistem sincron, deciziile au loc la fiecare tact, ceea ce produce un nou set de stări pentru toate elementele de memorie din sistem. Cînd tactul sistemului este lent, după realizarea deciziilor logice întregul sistem intră într-o pauză, pînă la următorul tact. *Ori de cîte ori tactul are o frecvenţă mai redusă decît cea maximă, proiectantul trebuie să ia măsuri, pentru a încerca să repete logica în timp, mai curînd decît în spaţiu.* Adesea, cînd datele sînt preluate drept cuvinte sau caractere, putem organiza întregul sistem pentru a manipula datele bit cu bit (în serie). Dacă cerinţele de viteză nu permit repartizarea unui interval de timp pentru fiecare bit, se poate folosi o soluţie serie-paralel avînd numărul de intervale impus de timpul necesar. De fapt, avînd în vedere registrele de deplasare MSI disponibile, numărul de intervale de timp este de obicei 4, 8, 16 sau altă valoare standard pentru lungimea registrelor de deplasare. Economiile de cipuri pot fi mari dacă se folosesc registre de deplasare hex-tuple (de exemplu 2518 conţine şase registre de deplasare de 32 de biţi).

În cazul unei organizări serie, proiectarea este practic identică cu cea a unui sistem ce prelucrează cuvinte de un bit, cu excepţia faptului că bistabilele D sînt înlocuite de registre de deplasare de n biţi şi a tactului ce este de n ori mai rapid pentru cuvintele de n biţi. O altă excepţie se înregistrează în cazul operaţiilor de adunare, scădere şi deplasare ce pretind un transport de la bitul adiacent. Deoarece bitul adiacent a fost plasat în intervalul de timp anterior el nu mai este disponibil decît dacă a fost reţinut într-un *bistabil de transport (carry)*. Se poate astfel realiza orice „interconexiune" între intervale succesive de timp prin folosirea unui singur bistabil *D* suplimentar.

Pentru viteză mai mare, se pot folosi organizări serie-paralel. De exemplu, un sumator de 4 biţi sau o unitate logico-aritmetică (ALU) pot realiza operaţii de 16 biţi în 4 tacte. Un sistem de 16 biţi poate fi astfel construit cu o pătrime din logica combinaţională, dacă se poate accepta o reducere de 4 ori a vitezei. Sistemul este identic unui sistem cu patru biţi, cu excepţia registrelor de de-

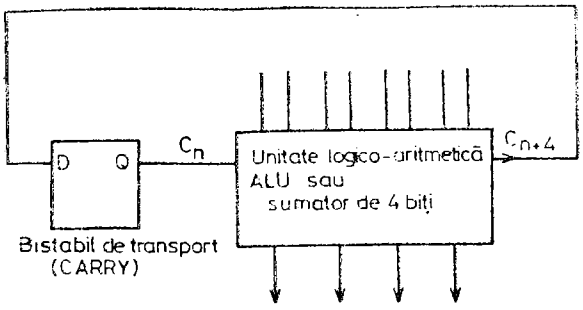


Fig. 11-2. Trecerea semnalului de CARRY de la un interval de timp la cel următor.

plasare de patru biți ce înlocuiesc bistabilii *D* și a tactului, care constă din patru tacte. Din nou, se introduce un bistabil de transport (vezi fig. 11-2) pentru a transfera transportul de la un interval de timp la următorul.

Uneori pe lîngă simplul transfer al unui semnal de la un interval la altul mai sînt necesare și alte tipuri de legături între circuitele repetate în timp. De exemplu, dorim să aflăm dacă rezultatul tuturor celor 8 operațiuni AND din figura 11-1 *a* este zero. În varianta paralelă din figura 11-1 aceasta presupune existența unei porți cu opt intrări. În varianta serie din figura 11-1 *b* putem realiza aceeași funcție cu un singur bistabil și cu o poartă cu două intrări, așa cum se indică în figura 11-3. Dacă oricare din primele șapte rezultate este 1, bistabilul va fi setat înaintea intervalului de timp 7. Ieșirea porții indică astfel, în intervalul de timp 7, dacă toare cele opt rezultate au fost zero. În mod normal, un sistem are un *numărător de intervale* care ține evidența acestora. Intervalul final (7, din figura 11-3), este echivalent tactului dintr-un sistem paralel.

Se pot implementa funcții logice mai complexe între intervalele de timp, scriind o ecuație, ca funcție de semnalele contorului de intervale și de datele serie, ce va putea fi utilizată pentru a seta un bistabil. De exemplu, circuitul din figura 11-4 va detecta un rezultat serie de 11110000. Desigur, fiecare din aceste funcții poate fi detectată în paralel, în cursul intervalului de timp 7, de către o poartă normală cu opt intrări, dacă registrul de deplasare A este de tipul intrare serie-ieșire paralelă. În felul acesta, în cadrul aceluiași sistem, putem combina funcționarea serie cu cea paralel.

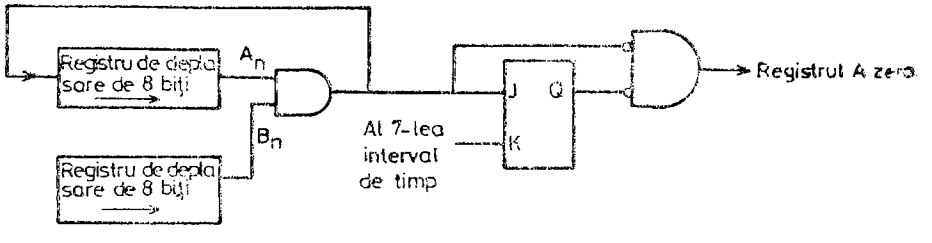


Fig. 11-3. Detector de zero, serial, realizat cu un bistabil.

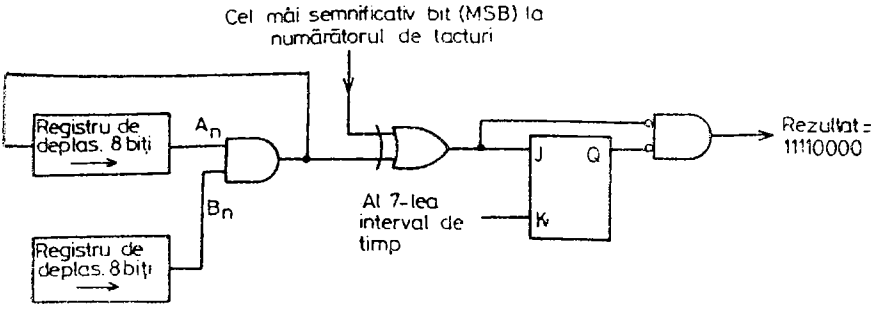


Fig. 11-4. Detectarea configurației „11110000" cu un bistabil.

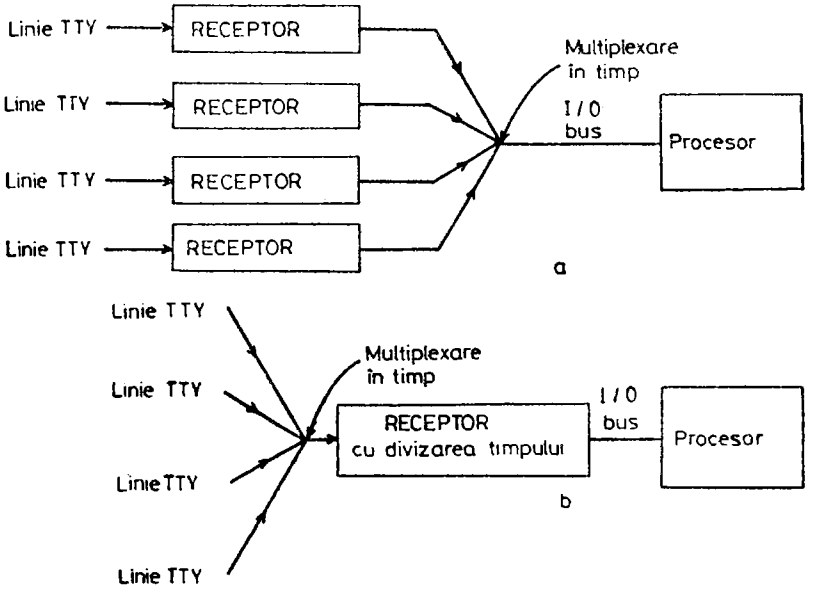


Fig. 11-5. *(a)* Abordare cu receptoare independente. *(b)* Abordare într-un sistem cu divizarea timpului.

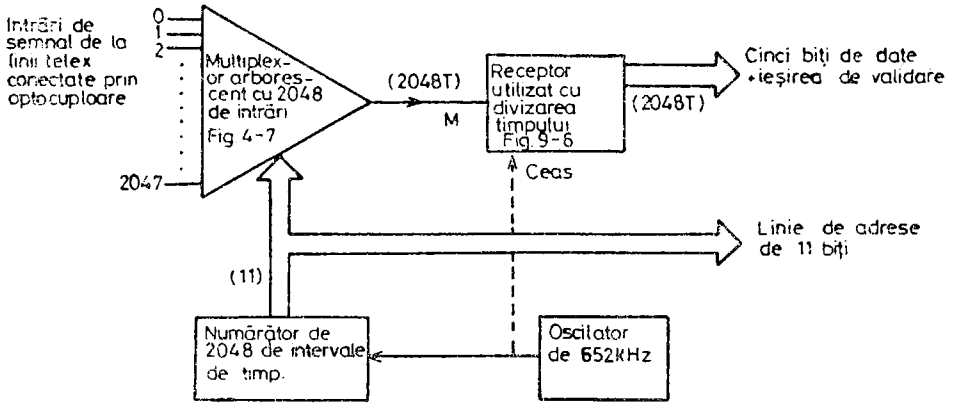


Fig. 11-6. Utilizarea circuitului cu divizarea timpului.

11.3 CIRCUITE DISTRIBUITE ÎN TIMP

Într-o largă clasă de probleme practice sînt necesare *multe repetări ale aceluiași circuit logic complet.* De exemplu, un calculator dotat cu multe tastaturi, linii de comunicație, sau alte dispozitive de intrare/ieșire, necesită, în mod normal, o logică de interfață pentru fiecare dispozitiv. Formatul datelor serie de Telex, folosit anterior ca exemplu de programare (vezi fig. 8-1*a*), constituie un exemplu perfect. Unele calculatoare implică interfațarea cu sute sau mii de astfel de linii. Deși este posibil să scriem un program pentru a recepționa semnalele de pe aceste linii bit cu bit, capacitatea procesorului este suficientă pentru a recepționa doar un număr redus din aceste semnale. Pentru a urmări multe astfel de intrări, și pentru a lăsa timp procesorului pentru alte sarcini, trebuie să utilizăm o logică separată pentru a recepționa semnalele de intrare serie și pentru a prezenta la intrarea procesorului *caractere complete.*

În figura 11-5*a* se prezintă un sistem pentru o astfel de recepție, dotat cu receptoare serie separate pentru fiecare linie. Deoarece aceste circuite sînt toate identice, *putem să repetăm circuitele în timp, în loc de spațiu* (vezi fig. 11-5*b*). În cazul variantei cu receptoare separate din figura 11-5*a*, se pot multiplexa ieșirile multor circuite pe busul de I/O al procesorului (deoarece fiecare furnizează date la alt moment, sub controlul programului). În cazul variantei cu receptor distribuit în timp, multiplexarea este îndepărtată și mai mult de procesor. Mulțimea de linii este multiplexată în una singură, folosind un arbore de multiplexare, cum este cel din figura 11-6. Acest arbore este controlat de un *contor de intervale de timp*, astfel încît fiecare linie este cuplată la intrarea receptorului în timpul unui alt interval de timp.

Ca un exemplu de repetare a subsistemelor logice în timp, vom proiecta un receptor de date serie pentru semnale Telex și apoi îl vom distribui în timp la 2048 de linii separate. Vom proiecta un receptor pentru un canal unic și apoi îl vom duplica de 2048 ori în timp utilizînd pur și simplu registre de deplasare de 2048 de biți în locul bistabililor din varianta obișnuită. Fercvența tactului va fi de 2048 mai mare decît pentru un receptor cu un un singur canal dar vom cupla, în schimb, cele 2048 de intrări la intrarea receptorului printr-un arbore de multiplexare cu 2048 de intrări, adresat de un contor de interval de 11 biți, avînd același tact.

Proiectarea receptorului de date serie va fi modificată într-o anumită măsură deoarece *trebuie să-l proiectăm folosind numai bistabili D simpli,* fără intrări de PRESET sau CLEAR. Desigur, putem utiliza porți pentru a transforma, dacă aceasta simplifică proiectarea, bistabilul *D* în alt tip de bistabil (vezi fig. 5-2).

11.4. EXEMPLU : UN RECEPTOR DE DATE SERIE DISTRIBUIT ÎN TIMP

Figura 11-7 prezintă schema bloc a receptorului nostru de date serie. Remarcați că tehnica pe care o vom folosi este generală (mai puțin utilizarea bistabililor *D* pentru numărare) și poate fi utilizată și pentru receptoare de date serie nemultiplexate în timp cu start și stop. De fapt, același principiu poate fi folosit și pentru a scrie un program de recepție a mai multor astfel de linii prin logică programată. Se poate recepționa un cuvînt mai lung de date

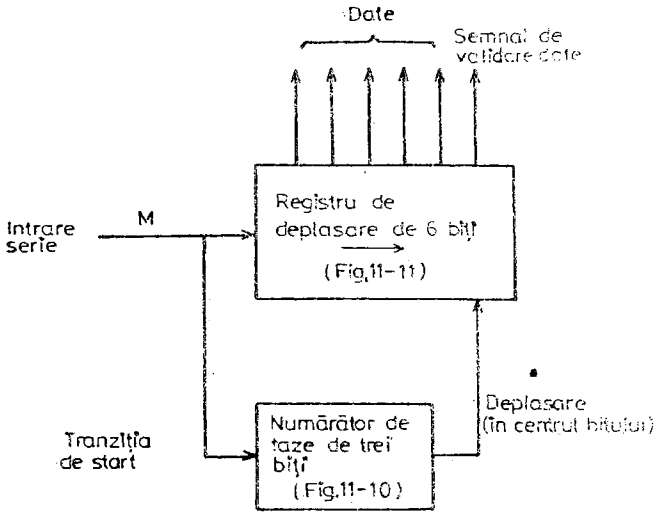


Fig. 11-7. Schema bloc a unui receptor de date serie.

prin simpla lungire a registrului de deplasare (de exemplu un registru de nouă biți pentru cuvinte de 8 biți). Deoarece logica combinațională devine nesemnificativă ca pondere atunci cînd este distribuită în timp, în cursul proiectării întregul accent cade pe minimizarea bistabililor din circuit. În loc să numărăm biții pe măsură ce sînt introduși, vom folosi sosirea bitului de *start* în etajul suplimentar al registrului de deplasare pentru a indica recepția întregului caracter.

Deoarece semnalul serie de intrare este complet asincron, iar noi dorim să eșantionăm biții de date în centrul lor, vom folosi un tact de 7 ori mai rapid decît viteza de transmisie (7×45 biți/s = 318 tacte/s) și vom utiliza *un numărător de faze* (divizor prin 7), ce este sincronizat de tranziția de start. Figura 11-8*a* reprezintă diagrama de stări a numărătorului de faze. Cînd în etajul final al registrului de deplasare există un bit de start (ST' = 1), iar linia este în repaus (M = 1), numărătorul stă în așteptare (starea WAIT). O tranziție de *start* (M = 0) avansează numărătorul în stare START PRESET (setare start), ceea ce produce resetarea întregului registru de deplasare. Pentru restul caracterului, numărătorul numără într-un ciclu de șapte stări („sărind" START PRESET), generînd un tact de deplasare de fiecare dacă cînd trece prin starea CEAS(P1). Așa cum se indică în diagrama în timp (vezi figura 11-8*b*), aceasta va avea loc în mijlocul bitului de start și al fiecărui bit de date.

Cînd toți biții au fost introduși în registrul de deplasare, bitul de start (zero) va ajunge în ultima celulă (ST) a registrului împiedicînd numărătorul P să părăsească starea AȘTEPTARE pînă cînd apare alt bit de start. Pentru a evita detectarea prematură a unui bit de start, numărătorul P este readus în starea 0 cînd a fost introdus și ultimul bit. Aceasta face schema insensibilă la apariția bitului de start pînă în mijlocul bitului de stop.

Deoarece rata tactului este impusă de un oscilator cu cuarț, punctul de eșantionare este definit pînă la $\pm 0{,}5$ din perioada acestuia. Perioada primară de tact fiind 1/7 din durata unui bit, punctul de eșantionare este definit pînă la $\pm 1/2 \times 1/7 = \pm 1/14 = \pm 7\%$ pentru toți biții.

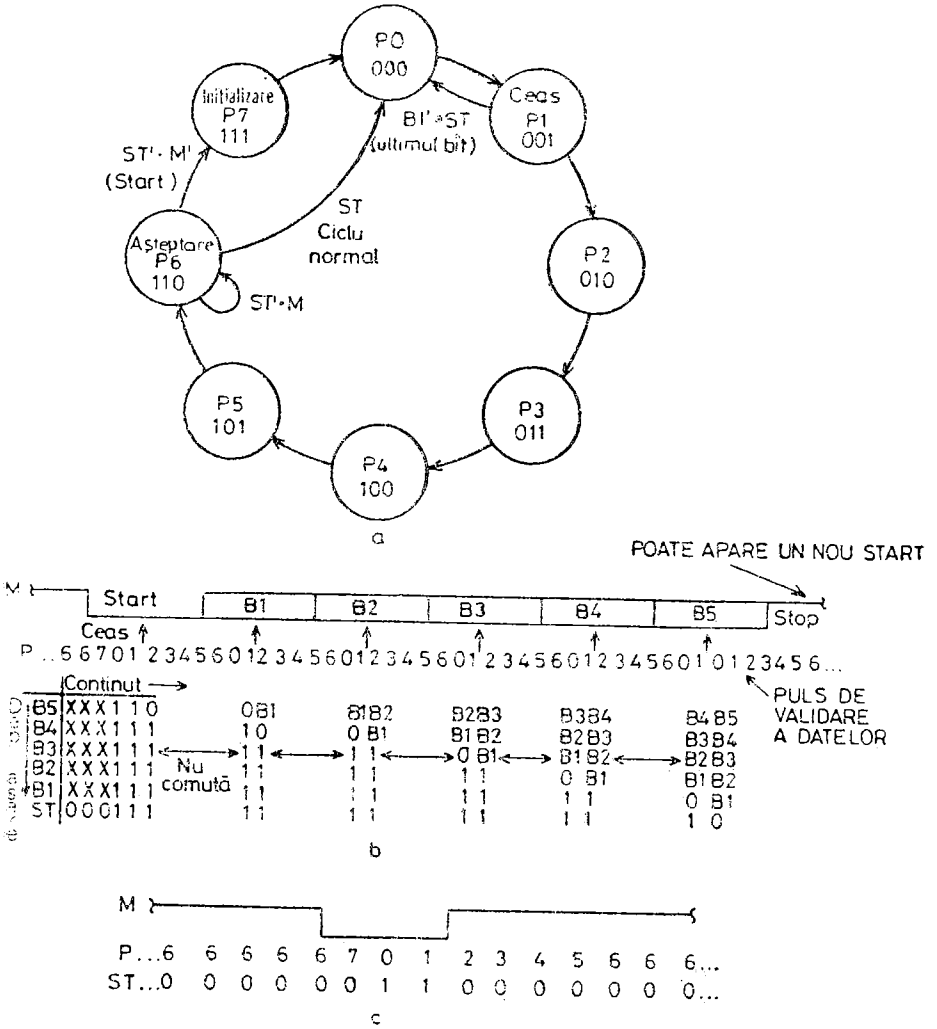


Fig. 11-8. Receptor de date serie *(a)* Diagramă de stări pentru numărătorul de faze (opt faze distincte, deci trei biți pentru codare). *(b)* Diagrama de timp pentru un caracter normal. *(c)* Zgomot pe linia neocupată (în așteptare) — diagrama de timp.

Impulsurile de zgomot de pe linia în repaus sînt rejectate de o poartă specială ZGOMOT (fig. 11-11) care readuce logica în starea normală, de repaus, dacă impulsul de start nu are o durată de cel puțin o jumătate de bit, așa cum se indică în figura 11-8*c*.

Discuția anterioară corespunde unui circuit logic convențional, dar ea rămîne de asemenea valabilă și pentru circuitul distribuit în timp, în interiorul fiecărui interval de timp. Singura diferență care apare în hardware este utilizarea unor registre de deplasare, în locul unor bistabili *D*. Deoarece frecvența tactului este de numai 318 tacte pe secundă, am putea să multiplexăm cu ușurință acest circuit în domeniul timp, pentru *a acoperi mii de linii*. De exemplu, dacă am utiliza registre de deplasare de 10 192 biți în locul bista-

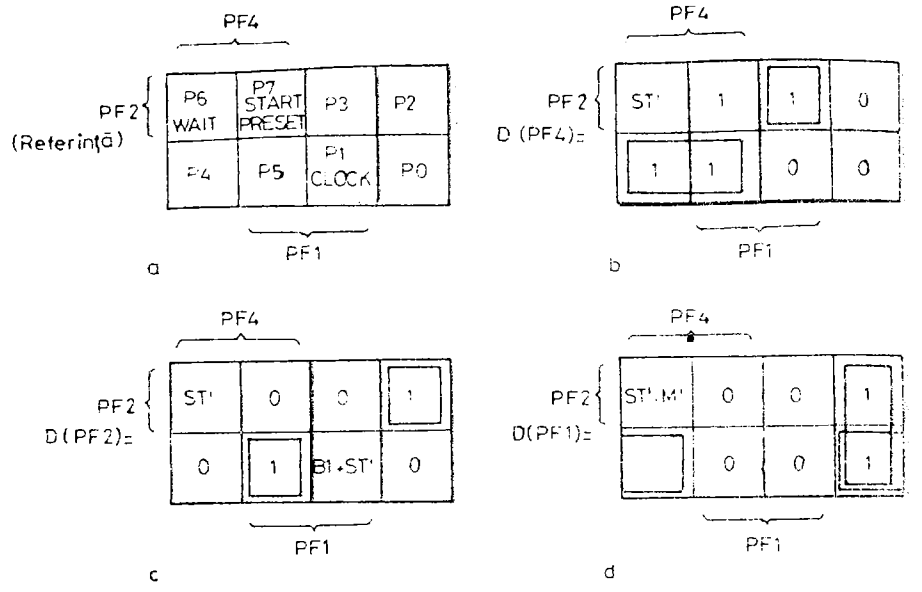


Fig. 11-9. Proiectarea numărătorului de faze pentru ceas.

bililor, frecvența tactului ar fi de numai 10 192 × 318 = 3,24 MHz. Un singur circuit ar putea astfel manipula 10 192 linii de intrare. În exemplul nostru, vom distribui circuitul în timp, între 2 048 linii de intrare. Prin urmare ne este necesar un arbore de multiplexare cu 2 048 de intrări, ca în figura 11-6. Adresa de intrare a acestui multiplexor provine dintr-un numărător binar de 11 biți ce are drept frecvență de tact 2048 × 318 = 652 kHz, ca și registrele de deplasare de 2048 de biți. Pentru una din cele 2048 linii de intrare fiecare bit al registrelor de deplasare corespunde unei stări a bistabililor.

Diagramele Veitch utilizate în proiectarea numărătorului de faze de tact sînt prezentate în figura 11-9. În poziția corespunzătoare fiecărei stări, trebuie să se scrie o expresie pentru *starea următoare* (intrarea D) a fiecăruia din cei trei bistabili. De exemplu, din starea 1 ajungem întotdeauna în starea 1, astfel încît scriem 0,0 și 1, respectiv, în colțul din dreapta jos al celor trei diagrame. Din starea 1, plecăm spre starea 0 sau starea 2, dacă bitul recepționat este sau nu ultimul bit ($B1' \cdot ST$). Deoarece PF4 și PF1 (MSB, respectiv LSB ai numărătorului) vor fi 0 în ambele cazuri, vom scrie 0 în poziția corespunzătoare stării 1 a acestor diagrame. Deoarece PF2 va fi setat numai dacă $B1' \cdot ST$ este *zero logic*, vom scrie $B1 + ST'$ în poziția corespunzătoare stării 1 a diagramei respective ($B1 + ST' = (B1' \cdot ST)'$, conform teoremei De Morgan). Vom continua să urmărim săgețile ce pleacă *din fiecare* stare a diagramei de stări din figura 11-8*a* pînă cînd completăm diagramele Veitch pentru cei trei biți (PF4, PF2, PF1) ai numărătorului.

Examinînd diagramele rezultate, observăm că funcțiile rezultate sînt destul de complicate astfel că solicită multe porți pentru implementare. Acesta este desigur, motivul pentru care utilizăm, în mod obișnuit, bistabili $J—K$ pentru acest tip de numărător. În cazul de față folosim totuși bistabili D

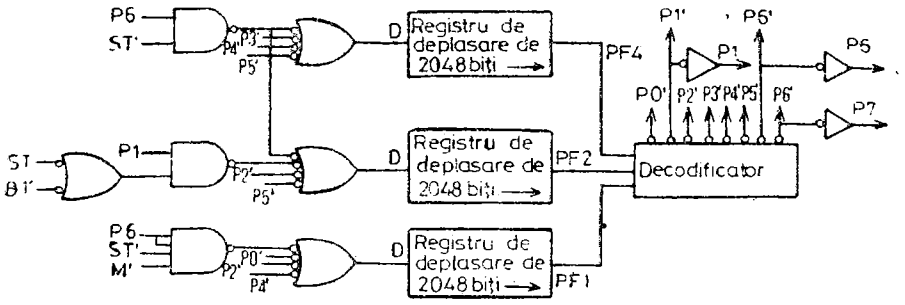


Fig. 11-10. Numărător de faze pentru ceas (divizat în timp×2048).

deoarece registrele mari de deplasare au toate practic, intrări D (deși există cîteva registre de deplasare, ca 74195, ce au intrări J și K'). Pentru a simplifica logica, vom utiliza un decodor MSI care decodează stările numărătorului. Ecuațiile noastre logice, în funcție de ieșirile decodorului $P0$—$P7$, vor fi atunci:

$$D_{(PF4)} = P6 \cdot ST' + P3 + P4 + P5 \tag{11-1}$$

$$D_{(PF2)} = P6 \cdot ST' + P1(B1 + ST') + P2 + P5 \tag{11-2}$$

$$D_{(PF1)} = P6 \cdot ST'\ M' + P0 + P2 + P4 \tag{11-3}$$

Ecuațiile sînt implementate ca în figura 11-10. De reținut că în locul unor bistabili D, am prezentat registre de deplasare de 2048 de biți. Aceasta produce echivalentul a 2048 numărătoare independente de faze de tact, fiecare activ, în intervale de timp distincte, pentru o linie diferită de intrare.

Registrul de deplasare de șase biți (vezi fig. 11-7), este proiectat, de asemenea, ca și cum ar trebui să fie implementat cu bistabili D. Deplasarea este activată numai în cursul stării P1, cînd ST este setat. Deoarece putem face deplasarea numai cînd $P1 \cdot ST = 1$, trebuie să prevedem o logică pentru a păstra starea anterioară a fiecărui registru de deplasare, cînd $P1 \cdot ST = 0$. În figura 11-11 se arată cum se poate realiza aceasta cu multiplexoare quadruple cu două intrări. Fiecare intrare de comandă a registrului de deplasare *provine din ea însăși*, dacă $P1 \cdot ST = 0$, sau din următorul etaj de la stînga, dacă $P1 \cdot ST = 1$. În felul acesta, deplasarea va fi efectuată numai dacă $P1 \cdot ST = 1$, registrul rămînînd astfel nemodificat. Intrarea de STROBE (STB) a multiplexoarelor obligă toate ieșirile lor să devină 0 în cursul stării P7 (START PRESET). Deoarece este atît de simplu să producem ștergerea conținutului unui registru, datele sînt deplasate negate. Aceasta face ca bitul (zero) de start să fie introdus ca un 1. Cînd acest 1 apare în ultimul etaj (ST) el este cuplat cu P1 pentru a genera indicația de DATA READY, semnal ce este activ (1) numai în cursul unei perioade de tact (a doua stare P1, din cursul bitului 5).

Dacă un scurt puls parazit face ca o linie în repaus să treacă în zero pentru mai puțin decît o jumătate din durata unui bit, registrul de deplasare s-ar putea seta și apoi să introducă codul lui *start*. Deoarece semnalul nu a corespuns unui bit de *start*, ST' va trebui să rămînă inactiv în timp ce numărătorul de faze va continua să numere. Cînd, în mod corect, va veni un caracter acesta va fi introdus într-o fază necorespunzătoare generînd funcționarea

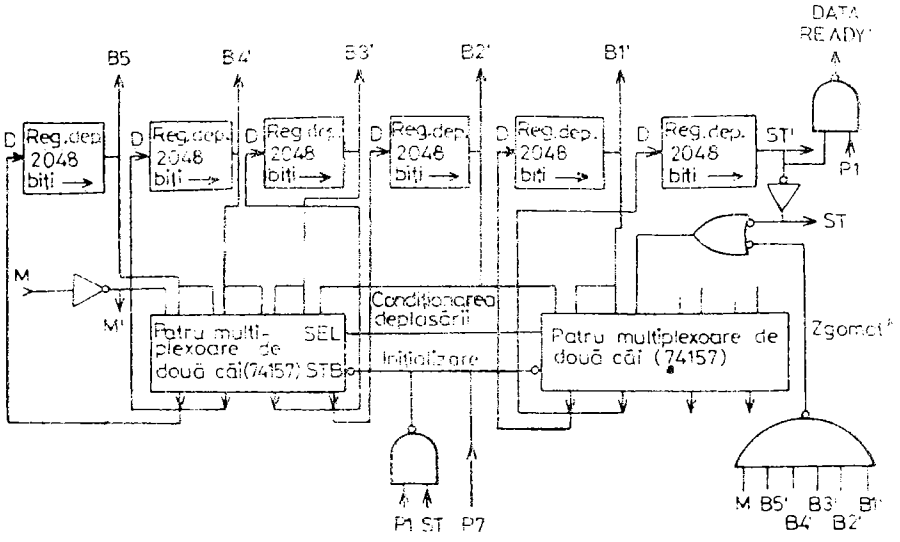


Fig. 11-11. Registru de deplasare pentru receptor de date serie.

eronată a sistemului. Pentru a preveni această situație este adăugată o poartă suplimentară pentru a genera bitul ST' dacă este detectat bitul de start activ. Aceasta blochează rapid secvența nepermițînd generarea semnalului DATA READY (date pregătite), așa cum se întîmplă în schema din figura 11-7.

Dacă semnalul DATA READY devine activ, cei cinci biți de date și cei 11 biți ce reprezintă adresa liniei (de la numărătorul intervalelor de timp) sînt încărcați într-un buffer (registru tampon) pentru a fi transferate procesorului. Deoarece uneori se întîmplă ca semnalul DATA READY să fie activ pe impulsuri de ceas succesive (dacă două linii adiacente, ca interval de timp în care sînt eșantionate, recepționează simultan un caracter), va fi necesar un buffer de date sau o conexiune de tip DMA cu memoria procesorului. Un alt mod de abordare presupune oprirea ceasului cu ocazia recepției unui caracter pînă cînd acesta a fost preluat în procesor. Pentru a fi păstrată acuratețea temporizării, ceasul va trebui să lucreze temporar la viteze mai mari, pentru a compensa opririle datorate transferurilor către procesor.

Intrarea *M* poate fi realizată ca un multiplexor structurat ca arbore cu 2048 de intrări (vezi fig. 11-6). O parte a acestui arbore este pe același modul cu logica de receptor, iar o altă parte pe modulele de interfațare cu liniile ce conțin și izolatorii optici. Dacă vom implanta un multiplexor cu 64 de intrări (precum cel din figura 4-7) pe modulul ce asigură recepția prin divizarea timpului, atunci la fiecare din cele 64 de intrări se va conecta ieșirea unui multiplexor de 32 intrări de la modulele ce conțin receptoarele de linii. În această manieră sistemul este expandabil cu module de cîte 32 de receptoare de linii ce se pot implanta numai dacă sînt necesare.

Pentru a indica într-o schemă numărul de eșantioane de timp alocate unei conexiuni sau unui circuit, vom nota într-o paranteză, numărul de eșantioane de timp urmat de *T*. Această notație este similară celei folosite într-o schemă bloc în care dacă vrem să notăm o conexiune realizată cu 32 de fire vom nota (32) alături de conexiunea în cauză. O conexiune asociată la 32 de eșantioane

de timp va fi notată cu (32T). Toate cele 64 de intrări ale multiplexorului de pe modulul de recepție vor fi notate cu (32T). Semnalul M de la ieșirea multiplexorului va fi etichetat cu (2048T), așa precum este reprezentat în figura 11-8.

Dacă vom socoti numărul de capsule de circuite integrate utilizate în exemplul de receptor prezentat în figurile 11-10 și 11-11 vom vedea că sînt suficiente *numai opt capsule*. Deoarece circuitul manipulează 2048 de linii, costul circuitelor logice raportat la o singură linie se reduce la fracțiuni de cent. Sînt necesare nouă registre pentru manipularea a cîte unui bit pe linie. Costul unui bit în cadrul unui registru de deplasare de mari dimensiuni fiind de o fracțiune de cent, rezultă că *prețul total al unei linii este de cîțiva cenți !*

Această modalitate de abordare este deci mai economică chiar decît logica programată, în care o aceeași operație poate fi executată de mai multe ori. Motivația acestei situații o găsim în faptul că în cadrul logicii programate procesul de divizare a timpului se realizează într-un caz general ce nu poate fi la fel de eficient ca o schemă logică ce rezolvă într-un caz particular problema divizării timpului. (Pentru comparație se poate consulta eficiența obținută prin program, pentru aceeași funcție, în Tabelul 8-3 din Secțiunea 8.1.1.). Prin proiectarea unei logici, ce presupune divizarea timpului, dedicată unei operații particulare care trebuie executată de mai multe ori, se poate atinge o eficiență mult mai mare.

11.5. ALTE POSIBILE ORGANIZĂRI CU MULTIPLEXARE ÎN TIMP

Există multe alte moduri de a distribui în timp circuitele. Nu este neapărat, spre exemplu, să avem un registru de deplasare pentru fiecare bit. Am fi putut folosi pentru *toate* elementele de memorie asociate la 115 receptoare de linii Telex, un singur registru de deplasare de 1 024 biți ; în figura 11-12 se prezintă o astfel de configurație. Cei nouă biți (șase în registrul de deplasare, plus trei biți ai numărătorului de faze) necesari pentru fiecare canal, sînt depuși în locații consecutive ale registrului de deplasare de 1 024 biți. Cu nouă tacte de deplasare, se poate determina o nouă stare a numărătorului și a registrului de deplasare. De asemenea acestea se vor recepționa cu un singur circuit, implementat cu structuri MSI. La cel de al zecelea impuls de tact va intra în funcție logica de recepție, incrementînd numărătorul de faze și/sau introducînd un nou bit în registrul de deplasare, ca și cum ar funcționa o singură linie de recepție.

Pe următoarele nouă impulsuri de tact se va actualiza în registru și numărător starea anterioară pentru următorul canal, în timp ce starea asociată canalului curent este introdusă în registrul de deplasare de 1 024 de biți. Deoarece 3 + 8 + 1 024 = 1 035 biți este suma celulelor de memorie ce formează calea de recirculare, numărul canalelor Telex ce se pot conecta la această structură este de 1 035/9 = 115.

Desigur, se pot folosi diverse combinații de conexiuni serie și paralel pentru stocarea biților în cazul structurilor logice ce folosesc multiplexarea în timp. Alegerea unei anumite configurații depinde de viteza impusă prin datele de proiectare, de numărul de circuite, și de tipul registrelor de deplasare disponibile. Ceea ce este de fapt important este să învățăm să utilizăm pentru conceperea unei structuri logice dimensiunea timp, cu aceeași ușurință cu care folosim dimensiunea spațiu. Sistemul din figura 11-12, spre exemplu,

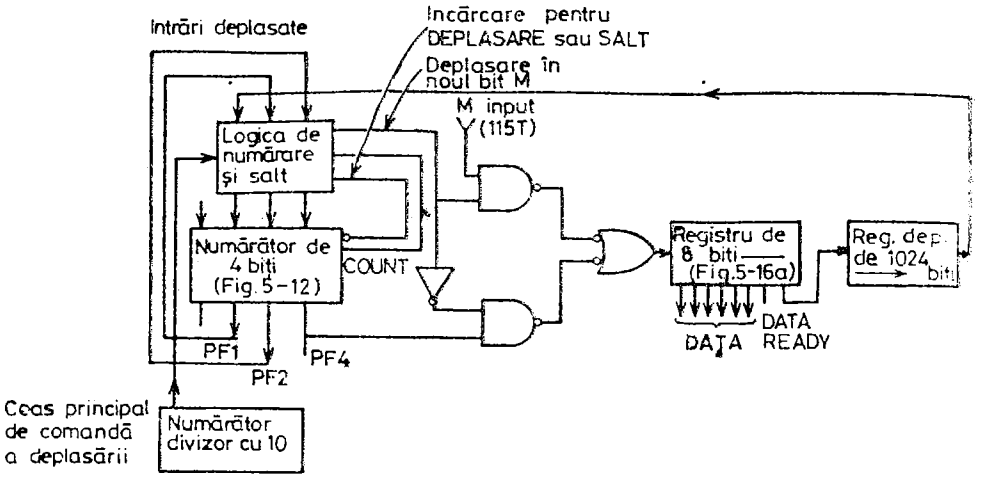


Fig. 11-12. Receptor cu divizarea timpului ce utilizează un singur registru de deplasare.

este în esență o aplicare a acestui mod de abordare. Multiplexăm în timp nouă biți pentru a realiza multiplexarea în timp a canalelor. Multiplexarea în timp este realizată pe două niveluri. utilizăm, deci, un singur registru MOS de deplasare și nouă intervale de timp, în loc de a utiliza nouă registre de deplasare „repetate în spațiu".

O altă idee ce trebuie reținută se referă la faptul că multiplexarea în timp poate fi utilizată pentru a proiecta circuite ce pot executa *sarcini diferite*. Spre exemplu, se poate proiecta un receptor serial cu facilitatea de a fi „programabil", în sensul că există posibilitatea recepționării datelor cu diferite formate (rate diferite de transfer sau număr diferit de biți pe cuvînt). Vom putea obține printr-o astfel de proiectare o eficiență suplimentară datorită unei flexibilități ce apare ca fiind similară abordării programate. O altă variantă posibilă este aceia de a proiecta o variantă ce reprezintă o combinație de circuit emițător-receptor cu un receptor-emițător al bitului de control. Vom putea utiliza deci același registru de deplasare pentru emisia sau recepția unei linii oarecare. Costul logicii combinaționale fiind neglijabil în cazul circuitelor cu multiplexare în timp, creșterea complexității acestora va rămîne fără consecințe notabile.

Registrele de deplasare cu recirculare (ex. : INTEL 2401 ce conține două registre de 1 024 biți) are încorporat un multiplexor ce permite recircularea internă a datelor stocate cu excepția intervalelor de timp în care este activată intrarea WRITE. Această facilitate este curent folosită pentru simplificarea logicii de control, intrarea WRITE fiind comutată astfel încît să fie multiplexat *accesul ceasului* la registru. Exceptînd cazul în care este necesar semnalul de inițializare P7, schema din figura 11-11 poate fi realizată cu astfel de registre nefiind necesare multiplexoare externe adiționale. Dacă se proiectează numărătoare multiplexate în timp utilizînd registre cu recirculare acestea vor număra numai cînd semnalul WRITE va fi activ.

11.6. CIRCUITE DE IEȘIRE MULTIPLEXATE ÎN TIMP

Ieșirile unor circuite multiplexate în timp pot fi transformate din secvențe de semnale scurte, separate în timp, în semnale separate în spațiu prin utilizarea unui arbore de demultiplexare similar celui din figura 4-11. Evident, adresele pentru arbore provin din numărătorul de intervale de timp. În cadrul unui interval de timp, arborele din figura 4-11 oferă însă la fiecare intrare fie prezența, fie absența unui impuls (pentru 0 și 1). Ceeace dorim însă să obținem este un nivel logic continuu a cărui stare să fie actualizată la fiecare tact. Pentru a obține aceste ieșiri continue este necesar să avem un bistabil pentru fiecare ieșire. Un *latch adresabil* (de exemplu 9334) este ca un demultiplexor cu opt ieșiri cu memorie pe fiecare ieșire. Putem, prin urmare să obținem ieșiri continue folosind latch-uri adresabile pentru *etajul final* al arborelui de demultiplexare.

Am putea deci să construim un emițător pentru transmisia datelor serie, cum sînt cele indicate în figura 11-7*a*. Un arbore de demultiplexare, repartizat parțial pe placa logică distribuită în timp și parțial pe plăcile ce conțin releele de transmisie, ar realiza transformarea timp-spațiu. Nivelul final al arborelui constă din latch-uri adresabile.

Circuitele de ieșire diferă, de asemenea, de cele de intrare, prin aceea că procesorul (sau logica de comandă) *inițiază* ieșirea. Ori de cîte ori procesorul dorește să transfere spre ieșire un caracter, el trebuie să-l depună în intervalul de timp corespunzător al registrului de deplasare de ieșire. Circuitul de ieșire asigură apoi transmisia caracterului de start și deplasarea caracterului spre exterior cu viteza adecvată. Deoarece procesorul poate transmite spre orice linie și în orice moment, el ar trebui să aștepte pentru ca să apară intervalul de timp corespunzător, dacă am folosi un registru de deplasare pentru bistabilul multiplexat în timp, cum am procedat în cazul circuitului de intrare. Această problemă poate fi ușor soluționată prin utilizarea unui RAM *dinamic, adresat de contorul de intervale de timp, similar bistabililor multiplexați în timp.* Deoarece fiecare bit poate fi adresat pe rînd, memoria va apare circuitelor multiplexate în timp drept un registru de deplasare.

După cum se arată în figura 11-13, se pot folosi multiplexoare cu două intrări, pentru a comuta intrările de date și de adrese, astfel încît procesorul să poată transmite la orice adresă, în orice moment, iar circuitul multiplexat în timp să poată continua să folosească RAM-ul drept un bistabil distribuit în timp. Multiplexorul de adrese poate, desigur, să servească toți bistabilii distribuiți în timp, dar pentru fiecare intrare de date este necesară 1/4 dintr-un multiplexor cuadruplu cu două intrări. Un circuit similar poate fi folosit pentru circuitele de *intrare* multiplexate în timp, acolo unde este necesară o citire aleatoare a stărilor intrărilor, sub controlul programului, în locul unei citiri îndată ce fiecare intrare este disponibilă (ready).

Datorită complexității lor, aceste circuite pot fi reprezentate pe schema logică a dispozitivului logic multiplexat în timp printr-o schemă simplă, de bistabil D, t_0 notă pe diagramă poate indica circuitul real (vezi fig. 11-13) și poate preciza că semnalele bistabilului D reprezintă respectivul circuit.

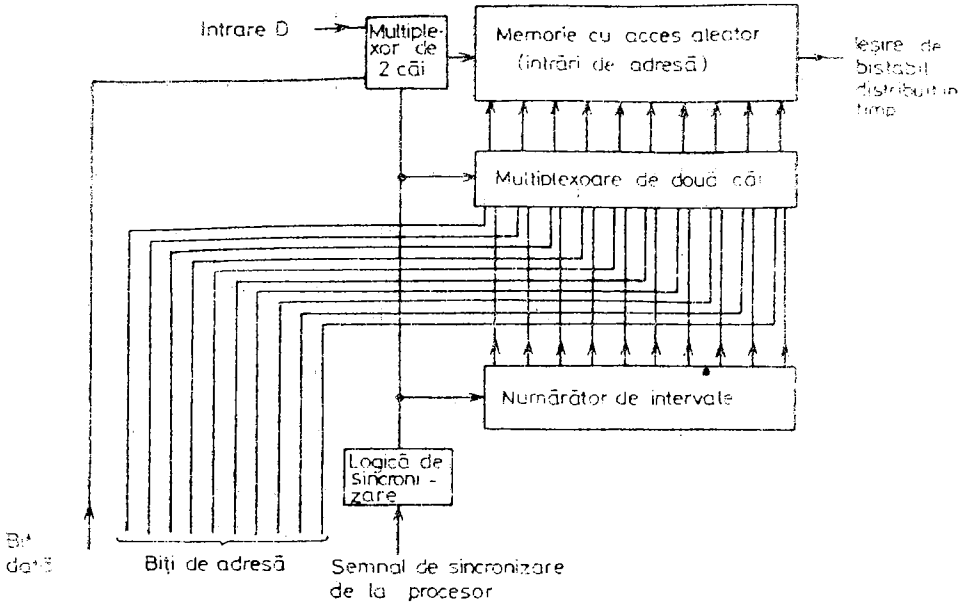


Fig. 11-13. Acces aleator, într-un sistem cu divizarea timpului, la un bistabil de tip D.

11.7. TEHNICI DE DEPANARE

Din momentul în care se înţeleg cîteva principii simple, depanarea circuitelor multiplexate în timp şi a circuitelor serie nu este mai complicată decît cea a aceloraşi circuite în formă obişnuită. De fapt, principiul de tratare a circuitului ca un circuit obişnuit poate fi folosit atît în depanare cît şi în proiectare.

Este evident că majoritatea defectelor grosiere pot fi rezolvate prin simpla urmărire a unei activităţi într-un anumit punct al circuitului. Această metodă este *ajutată*, de fapt, de multiplexarea în timp, fiindcă se poate urmări comportamentul a mii de circuite (intervale de timp), într-un singur punct.

Ca un ajutor în depanare, se poate introduce un bistabil *D* suplimentar pe unul din modulele sistemului. La intrarea de tact se cuplează semnalul de activare a intervalului de timp respectiv, iar o probă de test este cuplată la intrarea *D*. Un osciloscop conectat la ieşirea bistabilului va prezenta forme de undă *perfect analoage celor care s-ar observa pe un circuit obişnuit, în cazul unui canal unic*. Intrarea de test a bistabilului este plasată în orice punct al circuitului, la fel ca o probă de osciloscop. Bistabilul memorează valoarea semnalului din punctul respectiv, în cursul intervalului de timp dorit şi permite ca repararea circuitului să aibă loc la fel ca a unuia obişnuit (vezi fig. 11-14).

Desigur, de cel mai mare ajutor este faptul că activitatea de service este mult simplificată, datorită implementării mult mai compacte.

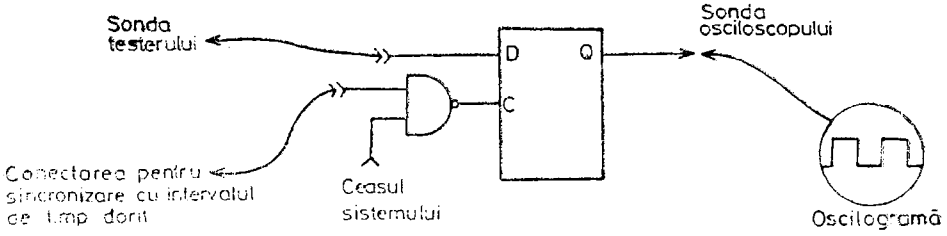


Fig. 11-14. Bistabil de test.

11.8. TRANSMISIA SERIE A DATELOR

Funcționarea serie este concepută adesea ca o modalitate de a economisi *porți logice* împărțindu-le efectiv în timp între mulți biți. Aproape la fel de important este și faptul că funcționarea serie *multiplexează în timp și firele de interconexiune*. Odată cu disponibilitatea circuitelor MSI și LSI, costul și rata de defectare a unui sistem a devenit tot mai legată de numărul *interconexiunilor* necesare. Transmițînd biții rînd pe rînd, pe o singură interconexiune se pot transmite mulți biți.

Economisirea interconexiunilor devine chiar mai importantă în cazul în care semnalele sînt transmise pe o distanță mare, deoarece costul firelor crește odată cu mărirea distanței. Prin funcționarea serie, se multiplexează nu numai cablul, ci și pinii de conector, ca și circuitele de emisie și recepție. Cînd semnalele sînt transmise la distanțe cu adevărat mari, ca în cazul liniilor telefonice, linia, circuitele de transmisie și recepție devin atît de scumpe încît se folosește, practic întotdeauna, funcționarea serie.

Pe măsură ce logica necesară pentru conversia paralel-serie devine mai ieftină, transmisia serie devine tot mai tentantă, chiar în cazul unor cabluri relativ scurte. În figura 11-15*a* se prezintă un sistem tipic de transmisie paralelă pe cablu. Pentru fiecare bit de date și pentru tact sînt necesare circuite de emisie și recepție, pini de conectori și conductoare separate.

Figura 11-15*b* prezintă un sistem serie de transmisie a acelorași date folosind un format serie cu start/stop (ca în figura 11-7*a*). În felul acesta apare problema unui compromis între costul și fiabilitatea logicii suplimentare și a emițătoarelor, receptoarelor, pinilor de conector și a firelor de conexiune. Adesea apare o *economie netă* în logica necesară în varianta serie deoarece prelucrarea serie se poate realiza integral în subsisteme concentrate la fiecare capăt, generînd o economie ce depășește costul logicii suplimentare necesare pentru a adăuga biții de start și stop și pentru a genera tactul la recepție. Cînd datele sînt în format serie, generarea sau verificarea *parității* poate fi realizată cu un singur bistabil J—K care basculează pentru fiecare 1 recepționat.
În cazul datelor în format paralel, pentru realizarea aceleiași funcții este necesar un circuit logic complex. Deoarece datele au, în mod normal, sursa sau destinația în registre de bistabili, funcționarea serie reduce numărul de capsule necesare pentru aceste registre. În loc să utilizăm registre de *patru biți*, cu intrare și ieșire paralelă la ambele extremități (vezi fig. 5-16*c*), vom folosi registre de opt biți la ambele capete (în modul intrare paralelă-ieșire serie, la extremitatea de emisie și în modul intrare serie-ieșire paralelă la extremitatea de recepție).

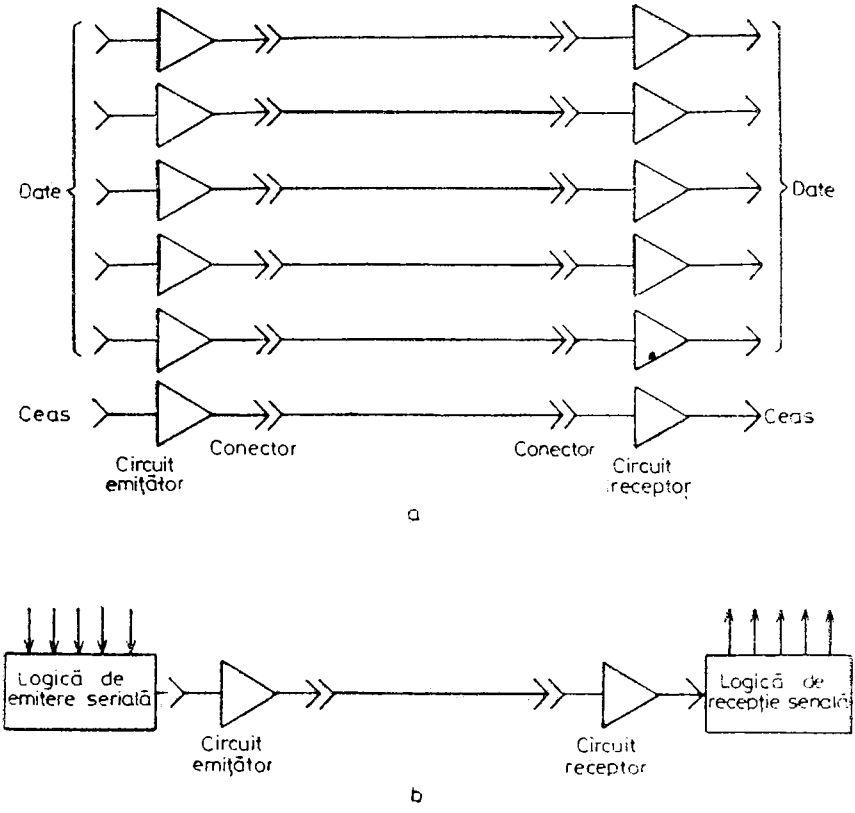


Fig. 11-15. *(a)* Transmisia în paralel a datelor. *(b)* Transmisia serială a datelor.

Folosind un oscilator cu cuarț la fiecare extremitate, se poate extinde formatul cu biți de start/stop (vezi fig. 11-7*a*) la orice număr de biți, prin simpla extindere a registrelor de deplasare de la fiecare capăt. Un oscilator cu cristal oferă o precizie de $\pm 0{,}001\%$, fără costuri suplimentare. Eroarea cumulativă pentru 30 de biți este în acest caz numai de $30 \times 0{,}001 = 0{,}03\%$. În figura 11-16 se prezintă exemplul unui format de ieșire dintr-un calculator, ce include 16 biți de date, o adresă de dispozitiv de șase biți, un bit de control, și un bit de verificare a parității. *Un singur emițător și un singur receptor îi înlocuiesc pe cei 25 ;* fiabilitatea și costul sînt astfel îmbunătățite într-un factor de 25. Economiile ce apar în logica de verificare a parității și prin folosirea unor registre de opt biți în locul unora de patru biți depășesc în mod serios costul necesar pentru logica suplimentară.

S-ar putea crede, la prima vedere, că viteza pe interfață este de 25 de ori mai redusă în cazul transmisiei serie. În realitate ea poate fi de fapt *mai rapidă*. Motivul rezidă în aceea că transferurile paralele sînt de obicei limitate în viteză, nu de viteza liniei emițător-receptor, ci de problemele de *defazare* (schew) a ceasului. Variațiile ce apar în întîrzierile driverului și receptorului, ca și variațiile de lungime ale conductoarelor din cablu (ele fiind împletite), fac necesară luarea în considerație a unei *margini de siguranță în temporizare* la fiecare front al tactului. În cazul funcționării serie, datele își produc propriul

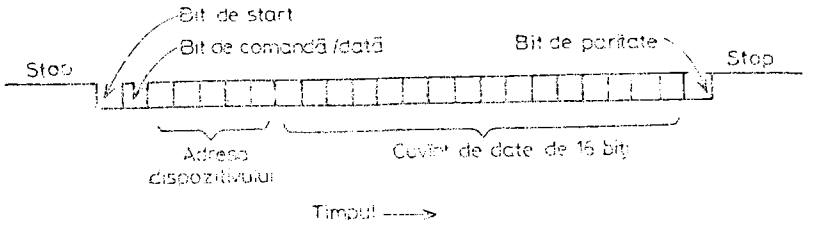


Fig. 11-16. Structura unui cuvînt emis serial.

tact, încît defazarea este un concept fără sens. Chiar la 20 MHz, circuitele de emisie și recepție sînt simple (vezi Secțiunea 12.9). Toți cei 25 de biți pot fi transmiși în 25/20 = 1,2 μs. Cerințele legate de defazare conduc la estimarea acestei valori ca un timp minim de transfer paralel pentru un cablu de orice lungime.

Dacă sînt necesare viteze mai mari, poate fi utilizat un sistem *serie/paralel*. Deoarece fiecare calc se autosincronizează, nu există probleme de defazare. Cînd toți biții de start ajung la extremitatea registrelor de deplasare, datele sînt complet transferate.

Un alt avantaj al transmisiei serie este dat de eliminarea *diafoniei*. Pe o interfață paralelă, pot fi generate semnale importante de zgomot chiar dacă cuplajul între două linii este redus, avînd în vedere faptul că diafonia este aditivă. De exemplu, dacă pe 24 din cele 25 de linii ale unui sistem paralel au loc transferuri simultane, pe cea de a 25-a linie se vor cupla aditiv 24 de semnale de diafonie. Dacă fiecare linie aduce o contribuție de zgomot de numai 0,04 V, efectul total de un *zgomot* de $24 \times 0{,}04$ V = 1 V. În felul acesta chiar în cazul unui sistem de transmisie *paralelă* cu diafonie foarte redusă, anumite configurații de date pot produce o funcționare incorectă.

11.9. DISPOZITIVE DE MEMORIE SERIE ȘI VERIFICAREA ERORILOR

Economii de hardware pot fi realizate nu numai prin transmisia serie a datelor ci și prin scrierea și citirea lor serie în dispozitivele de memorie. Cu un singur cap, în cazul unei memorii pe disc, se pot scrie sau citi zeci de mii de biți pe o singură pistă. Se poate considera că fiecare din acești biți are *o adresă localizată atît în dimensiunea timp* cît și dimensiunea spațiu.

Pentru a face posibilă scrierea și citirea unei cantități rezonabile de date la un anumit moment, datele serie sînt diviziate de obicei în *sectoare* numărate de la un *indicator* de pe disc. Fiecare sector include sute de octeți de date, ca și octeți suplimentari pentru verificarea erorilor și identificarea sectoarelor. Avînd în vedere că toți biții sînt înregistrați serie poate fi utilizat un singur amplificator de citire și scriere pentru toți biții. De asemenea verificarea erorilor, selecția de pistă, separarea și chiar transmisia spre logica de interfață a calculatorului pot fi efectuate în serie. Acționînd un numărător în sincronism cu rotația discului (cum se arată în figura 11-17), se poate adresa orice sector.

Unitățile de bandă magnetică și sistemele de comunicație folosesc un sistem aproape identic. Blocurile de date sînt denumite în aceste cazuri *înregistrări*. Înregistrărilor li se acordă numere de ordine și caractere speciale de

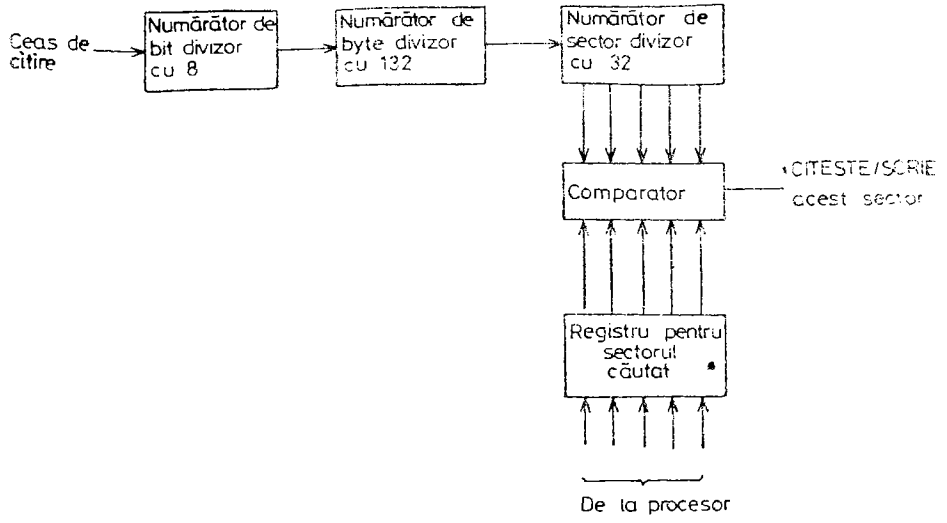


Fig. 11-17. Căutarea sectorului dorit.

identificare plasate la început. În continuare, căutarea și verificarea unei anumite înregistrări este realizată de către program. Dacă pentru o înregistrare particulară se detectează după scriere și transmisie o eroare, înregistrarea se *rescrie* sau se *retransmite*.

În cazul datelor serie, verificarea și generarea parității este foarte simplă. În figura 11-18 *a* se prezintă un bistabil *D* conectat astfel încît să basculeze de fiecare dată cînd intrarea de date este 1. Dacă acest bistabil este setat la începutul fiecărui octet, el va conține la sfîrșitul octetului indicația de paritate impară (deoarece el va fi 1 dacă a basculat de un număr par de ori). Un multiplexor cu două intrări pe ieșire face ca bitul de paritate să fie generat la sfîrșitul fiecărui octet serie. În același timp poarta OR de intrare forțează introducerea unui 1.

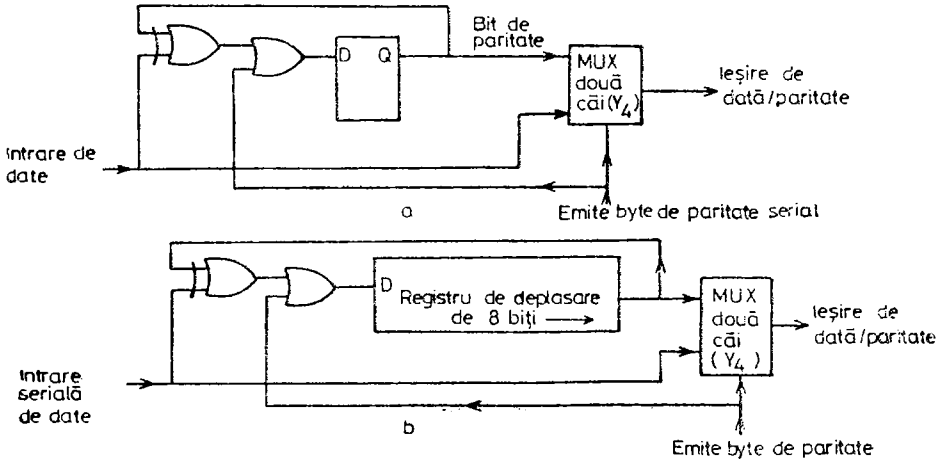


Fig. 11-18. Generarea serială a bitului de paritate : *(a)* logică pentru bit de paritate ; *(b)* logică pentru calculul parității pe byte (octet).

Pentru un mesaj lung se folosește de obicei, mai mult decît un singur bit de verificare. O tehnică obișnuită este cea a codului de *verificare pe bloc sau a octetului de paritate longitudinală*. Acesta constă dintr-un octet la sfîrșitul mesajului, compus din biți ce indică, fiecare în parte, paritatea corespunzătoare bitului respectiv din toți octeții mesajului. În cazul logicii paralele, aceasta ar pretinde un circuit, cum este cel din Fig. 11-18 a, *pentru fiecare bit*. Deoarece biții sînt serie, putem repeta circuitul în timp, în loc de spațiu, utilizînd pur și simplu un registru de deplasare de opt biți în locul unui bistabil, așa cum se arată în Fig. 11-18 *b*. Pe dureta fiecărui bit, bitul introdus în registru va fi inversat, dacă bitul corespunzător de date este 1. La sfîrșitul mesajului, octetul de paritate este scos din registrul prin multiplexorul cu două intrări. În același timp, în registru se introduc opt de 1, pentru a-l inițializa în vederea următoarei înregistrări.

EXERCIȚII

1. Proiectați un circuit care să realizeze serial negarea celor 16 biți dintr-un registru, de deplasare.
2. Proiectați un AND cu opt intrări, similar celui din figura 11-1, *a*, ce realizează funcția numai cu două intervale de timp.
3. *(a)* Proiectați un sumator serie-paralel de 32 de biți folosind un singur sumator de patru biți. Circuitul va suma conținutul registrului *A* la cel al registrului *B* și va depune rezultatul în registrul *C*. Includeți generatorul de ceas, numărătorul intervalelor de timp și bis tabilul de control al transportului (carry).
   (b) Cît va dura sumarea dacă se utilizează registrul dual de opt biți #9 328 ce are un timp maxim de propagare ed 39 ns și un timp maxim de stabilire (setup) de 16 ns? Timpul de întîrziere al sumatorului de patru biți este de 39 ns.
   (c) Care este numărul capsulelor de circuite integrate folosite?
   (d) Proiectați un sumator de 32 de biți mai rapid utilizînd două sumatoare de patru biți.
   (e) Care este timpul total de efectuare a adunării în această versiune?
   (f) Care este numărul capsulelor de circuite integrate folosite?
4. Proiectați un sumator similar celui din Exercițiul 3*a* și *b*, utilizînd trei cîmpuri de adresare de cîte doi biți pentru a desemna sursele operanzilor și destinația. Cele trei registre vor putea îndeplini oricare din funcții.
5. Proiectați un circuit ce compară conținutul a două registre de 8 biți utilizînd o singură poartă XOR. Rezultatul va fi generat în cel de al optulea interval de timp.
6. Proiectați un circuit ce transferă conținutul unui registru din opt registre de cîte opt biți în opt locații succesive ale unei memorii RAM de 1024×1 bit. O adresă de șapte biți indică locația în memoria RAM iar o adresă de trei biți indică registrul ales pentru a-i transfera conținutul în RAM.
7. Proiectați un circuit, similar celui din figura 11-4, pentru a detecta secvența 00110000
8. Proiectați un circuit ce realizează serial funcția ADD între opt locații succesive dintr-o memorie RAM de 1024×1 și un registru acumulator de 8 biți. Rezultatul se înscrie în acumulator. Un cod de șapte biți adresează octetul selectat din memoria RAM.
9. *(a)* Proiectați un circuit ce testează starea unui semnal de intrare la fiecare 29 ms și generează pe ieșire un semnal dacă starea intrării s-a schimbat în intervalul de timp anterior.
   (b) Construiți un circuit similar, bazat pe multiplexarea în timp, pentru 1 024 de intrări, ce indică modificarea uneia din intrări prin generarea adresei de selecție a acestuia.
10. Proiectați un sistem de numărătoare în cod Gray (vezi fig. 5-6 *a*) ce se activează ori de cîte ori una din cele 1 024 de intrări este activă. Fiecare intrare va fi eșantionată odată pe secundă. Folosiți trei registre de deplasare de 1 024 de biți cu *recirculare* și aceeași logică din figura 5-6 *f*.
11. Reproiectați receptorul serial de date, reprezentat în figurile 11-10 și 11-11, pentru a recepționa caractere de opt biți, la o rată de 10ms pe bit, pe 1 024 de linii. Utilizați registre cu *recirculare* cu porți AND cu două intrări conectate, în loc de multiplexoare, între secțiunile registrului. Simplificați comanda intrărilor D ale numărătorului folosind intrarea WRITE pentru a permite modificarea stării. Ecuațiile asociate intrărilor se vor simplifica deoarece apar *X*-uri în tabelul de adevăr atunci cînd WRITE = 0.

12. Proiectați logica de numărare/salt necesară receptorului serial de 115 linii reprezentat în figura 11-12.
13. Proiectați un emițător serial de date, multiplexat în timp, pe 2048 de căi, care produce formatul de start/stop, reprezentat în figura 11-8*b*, odată cu încărcarea caracterului în registrul de deplasare serie. Utilizați simbolul simplificat (2048T) pentru a indica în schema logică bistabilul de tip *D*. Acest simbol corespunde schemei din figura 11-13 în care, cu ajutorul unei memorii cu acces aleator, a fost simulat un bistabil *D* multiplexat în timp.
14. Proiectați un circuit multiplexat în timp pentru baleierea liniilor telefonice în vederea determinării stării lor (telefonul se ridică sau se pune în furcă) și a numărării pulsurilor generate de discul telefonic. Folosiți șapte bistabile multiplexate în timp, după cum urmează: unul, pentru a memora starea anterioară a liniei, două pentru a contoriza timpul de la ultima tranziție și patru pentru a număra pulsurile generate de disc. Un semnal de *validare* (data ready) va fi utilizat pentru a indica disponibilitatea codului de patru biți. Codul de patru biți al numărului va fi BCD. Cînd un telefon este activat este generat codul 1111. Linia va fi eșantionată la fiecare 20 ms. Numărul generat cu ajutorul discului este indicat de numărul de întreruperi ale curentului pe linie pentru intervale de timp de 60 ms cu pauze de cîte 40 ms între întreruperi. Dacă linia se întrerupe pentru mai mult de 30 ms înseamnă că cel cu care se vorbește a închis telefonul.
15. Proiectați un demultiplexor cu 1024 de ieșiri pentru comanda releelor. În loc de a folosi latch-uri adresabile, utilizați registre cu intrare serie și ieșire paralel. Țineți cont de faptul că releele nu răspund la semnale foarte scurte și modificați astfel rapid conținutul registrelor fără a perturba starea releelor. Desenați numărătorul de intervale de timp, un arbore pentru generarea celor 64 de semnale distincte de ceas și un grup de opt ieșiri.
16. Proiectați un sistem de transmisie pe șapte fire pentru comanda a 64 de diode luminescente (LED) ce corespund la 64 de comutatoare bipoziționale. Conexiunile comutatoarelor sînt sesizate cu ajutorul unei matrici de tipul celei din figura 4-12*b*, ce are cîte un comutator în fiecare intersecție. Această matrice este comandată prin intermediul unui numărător ce numără la 10 kHz. Cei șase biți sînt transmiși la conexiunile LED-urilor prin intermediul unei matrici de tipul celei din figura 4-12*a* ce are cîte o diodă luminescentă în fiecare intersecție.
17. Proiectați un sistem similar celui din Exercițiul 16 dar utilizînd o conexiune cu un singur fir. La fiecare extremitate se va prevedea cîte un numărător. Numărătorul de la extremitatea cu LED-uri va fi menținut în sincronism printr-un bit de start de la matricea de comutatoare. Linia va fi în repaus 50% din timp cu bitul de stop activ.
18. Proiectați un sistem de transmisie serială folosind formatul prezentat în figura 11-16. La emisie sistemul va avea 16 linii paralele de *date*, cinci linii de *adresă*, un semnal de *strob* (sincronizare) *a comenzilor* și un semnal de *strob pentru date*. La recepție ieșirile vor fi similare, dar se va adăuga o ieșire pentru *eroarea de paritate*.
19. Proiectați o memorie tampon de tip FIFO (first-in/first-out: primul intrat/primul ieșit) pentru caracterele recepționate de un receptor de date serie distribuit în timp (conform reprezentărilor din figurile 11-10 și 11-11). De cîte ori DATA READY este activat, cinci biți de date și 11 biți de adresă vor fi încărcați în două registre de 8 biți cu intrare paralel și ieșire serie. Cei 16 biți vor fi imediat deplasați cu o frecvență de 6 MHz și scriși, unul cîte unul, într-o memorie bipolară RAM de 256×1 biți. Un numărător de patru biți pentru adresa de scriere și un altul identic pentru cea de citire vor genera adresele reale pentru simularea memoriei FIFO. În paralel cu procesul de scriere se poate desfășura (pe un ceas în antifază) citirea datelor și livrarea lor paralel la intrarea unui calculator sub forma unui cuvînt de 16 biți.
20. Proiectați un sistem de verificare a erorilor pentru transmisia serială a cuvintelor de 16 biți. Cuvîntul este emis întîi inversat iar apoi neinversat. La recepție se vor compara bit cu bit cele două cuvinte transmise.

12

REALITĂȚI NEPLĂCUTE II

Zgomotul și reflexiile

12.1. INTRODUCERE

Cînd proiectăm sisteme logice, lucrăm în lumea pură a matematicilor, unde ecuațiile exprimă perfect funcțiile dorite. Cînd încercăm să transpunem ecuațiile logice în lumea reală, găsim că porțile pe care le folosim au anumite limitări și că firele de interconexiune sînt departe de a fi perfecte. În loc să transmită o replică perfectă a semnalului, un cablu de interconexiune ridică probleme legate de reflexii și zgomote, care pot face sistemul să nu funcționeze corect. Aceste funcționări incorecte sînt de tip marginal, adică se pun în evidență numai cu anumite circuite integrate și/sau numai cu anumite configurații de semnale (date). Trebuie, prin urmare, să fim foarte atenți la realizarea interconexiunilor, *neavînd nici o garanție că problemele de zgomot și reflexii se vor pune neapărat în evidență în cursul verificării.* În majoritatea cazurilor, putem evita aceste dificultăți urmărind pur și simplu cîteva *reguli de cablare.* Restul acestui capitol discută efectele fizice care fac ca aceste reguli să fie necesare.

Circuitele logice de mică viteză, cum sînt cele din familia CMOS, se apropie de idealul matematic. *Deoarece circuitele sînt lente, ele nu sînt afectate practic de zgomote și reflexii.* Chiar caracteristicile lor de curent continuu sînt practic ideale. Deoarece intrările porților nu absorb curent **fan-out-ul** (numărul de intrări ce poate fi comandat de o ieșire) este practic nelimitat. Deci majoritatea problemelor discutate în acest capitol sînt semnificative pentru circuitele CMOS numai atunci cînd se folosesc cabluri lungi; ele sînt importante în cazul circuitelor TTL chiar pentru interconexiuni normale, uzuale. În cazul familiilor logice mai rapide (de exemplu 74S sau ECL) problemele pot deveni importante chiar pentru interconexiuni cu o lungime de *cîțiva inch* (aproximativ 10 cm).

Un sistem logic secvențial (vezi fig. 8-1*a*) constă din bistabili și porți, interconectate astfel încît următoarea stare a bistabililor este o funcție de starea prezentă și de intrările sistemului. Atît timp cît schimbările de stări urmăresc funcționarea gîndită prin proiectare, sistemul lucrează normal. Zgomotul și reflexiile sînt astfel o problemă numai în măsura în care determină un bistabil să comute într-o stare incorectă. În general, dacă intrarea unui bistabil este modificată de zgomot însă pe un interval de timp inferior *timpului său de stabilire, (setup time),* el ajunge totuși în starea corectă. Prin urmare, cu cît un bistabil este mai lent, cu atît mai lung este zgomotul pe care îl poate tolera.

Durata maximă a zgomotului sau reflexiilor ce pot fi produse pe un segment de conductor este corelată cu lungimea sa. În funcție, de tipul dielectricului, *această durată variază între 9 și 12 ns/m.* Astfel, pentru fiecare tip de

logică, putem avea sisteme pînă la o anumită dimensiune fizică, fără a avea griji asupra problemelor de zgomote şi reflexii. De exemplu, un bistabil TTL (de exemplu 7474) are un timp de stabilire tipic de 15 ns. Dacă presupunem că timpul minim de stabilire este de 10 ns, putem să-l folosim fără multe probleme cu interconexiuni avînd o lungime de pînă la aproximativ 0,8 m.

Dacă, totuşi, folosim bistabili mai rapizi, Schottky TTL (de exemplu 74S74), pentru care timpul de stabilire tipic este de numai 3 ns, putem avea probleme şi cu fire de o lungime ce nu depăşeşte cu mult 15 cm ! Circuitele ECL, fiind şi mai rapide, pot apare probleme şi cu conductoare lungi de numai 7,5 cm.

12.2. CONEXIUNILE : ANTENE SAU LINII DE TRANSMISIE

Legătura dintre durata maximă a impulsurilor de zgomot şi lungimea interconexiunilor este legată de faptul că o bucată de conductor realizează o *antenă* slabă dacă nu are o lungime de cel puţin un sfert de lungime de undă. Firele de interconexiune din sistem pot fi privite ca antene de recepţie pentru zgomotul extern. *Diafonia* (cuplarea semnalelor din două sau mai multe linii de transmisiune) poate fi privită ca o transmisie radio, în care un conductor acţionează ca o antenă de emisie iar celălalt ca una receptoare. Deoarece firele constituie antene ineficiente dacă au o lungime mai mică decît un sfert de lungime de undă, se cuplează numai componentele de frecvenţă mai înaltă ale semnalului.

Un mod şi mai util de a concepe un sistem de conexiuni de mare viteză este cel al *liniei de transmisiune*. Conceptele elementare ale transmisiei pe linie ne permit să prevedem cu o precizie surprinzătoare formele de undă generate de reflexii. Aceste concepte devin nu numai utile, dar şi necesare dacă sîntem în situaţia să lucrăm cu conexiuni mai lungi decît lungimile maxime calculate anterior pentru diferite tipuri de logică (de exemplu 7,5 cm pentru ECL).

Cu toate că realizarea fizică a sistemelor de interconectare utilizate pentru circuitele logice nu seamănă deloc cu imaginea pe care o avem asupra unei linii de transmisiune, valorile *impedanţei* şi ale *vitezei de propagare* sînt destul de uniforme (chiar şi pentru conexiunile de tip wire-wrap realizate aleator). Viteza de propagare este determinată în mare măsură, de tipul de izolaţie. Aproape toate izolaţiile standard de conductoare dau o întîrziere de 4,5 ns/m, iar cablajele din steclotextolit dau o întîrziere de 5,4 ns/m. Încărcarea capacitivă produsă de porţi, de-a lungul liniei poate mări puţin întîrzierea, astfel încît valoarea de 6 ns/m este acoperitoare chiar în cazul cel mai defavorabil. În condiţii reale, este foarte greu să realizăm o impedanţă a liniei de transmisiune sub 30 Ω sau peste 600 Ω. De exemplu, pentru a realiza o impedanţă de 600 Ω, trebuie să suspendăm, la aproximativ 6 m, faţă de masă un conductor izolat nr. 18. Dacă lipim pe un plan de cupru pus la masă un conductor nr. 30, impedanţa este încă de aproximativ 70 Ω. Acelaşi conductor la aproximativ 2,5 mm deasupra masei produce o impedanţă de aproximativ 200 Ω. Putem, în felul acesta, presupune că firele legate prin wire-wrap au o impedanţă ce variază între 100 şi 200 Ω, sau 150Ω±33%. Aceasta este suficient de precis pentru tipul de analiză pe care intenţionăm să o facem, mai ales dacă nu dorim să intrăm prea mult în detalii. Folosind conductoare pereche împletite (twisted pair), putem controla impedanţa în limite de ±10%. În funcţie de con-

ductoarele folosite, în acest caz impedanța va fi de aproximativ 100 Ω. Traseele conductoare din cablajele imprimate, plasate deasupra unui „plan de masă" pot fi proiectate pentru o valoare de aproximativ 50 Ω, dacă se doresc performanțe foarte înalte.

12.3. REFLEXIILE ȘI OSCILAȚIILE

Impedanța caracteristică (Z_0) a unui fir este raportul tensiune/curent prin fir, pentru semnale de înaltă frecvență. Dacă terminația este o rezistență egală cu impedanța caracteristică a conductorului (vezi fig. 12-1*a*), nu vor apare probleme de propagarea semnalelor de-a lungul firului. Dacă totuși apar alte impedanțe la capătul liniei, ceva trebuie să se întîmple. Rezultatul este producerea unei *reflexii* astfel încît să se îndeplinească toate condițiile pe frontieră.

Figura 12-1*b* prezintă exemplul unei linii *terminate serie*. În acest caz, pe linie apare inițial numai jumătate din treapta aplicată, ea divizîndu-se în mod egal între rezistorul serie și impedanța liniei. Pe linie se deplasează astfel o treaptă de tensiune $E/2$ și o treaptă de curent $E/2Z_0$. Condiția de gol de la extremitatea din dreapta impune anularea curentului în punctul respectiv la orice moment de timp. Curentul se va anula în momentul în care treapta ajunge la extremitatea din dreapta dacă se produce o reflexie, constînd dintr-o treaptă de curent $-E/2Z_0$. Acest semnal reflectat produce o tensiune $E/2Z_0 \times Z_0 = E/2$, care se adaugă la treapta $E/2$ originală pentru a genera o tensiune totală de ieșire de $E/2 + E/2 = E$. Acest tip de problemă liniară poate fi rezolvată ușor urmărind relația *coeficientului de reflexie*.

$$\text{coeficientul de reflexie} = \Gamma_R = \frac{Z_R - Z_0}{Z_R + Z_0}. \qquad (12\text{-}1)$$

Semnalul reflectat are astfel valoarea semnalului direct, multiplicată prin coeficientul de reflexie. Coeficientul de reflexie la capătul deschis din figura 12-1*b* este:

$$\Gamma_R = \frac{\infty - Z_0}{\infty + Z_0} = 1. \qquad (12\text{-}2)$$

La capătul de emisie $Z_r = Z_0$, astfel că

$$\Gamma_R = \frac{Z_0 - Z_0}{Z_0 + Z_0} = 0 \qquad (12\text{-}3)$$

deci nu există reflexie la extremitatea de emisie. Dacă există reflexii la ambele capete, semnalul original va continua să parcurgă linia înainte și înapoi, între cele două extremități, cu o perioadă egală cu timpul de propagare pe linie (9—12 ns/m) așa cum se arată în figura 12-1*c*. De fiecare dată cînd treapta se „ciocnește" cu discontinuitatea de la capătul liniei, se calculează o nouă amplitudine, prin multiplicarea treptei ce sosește prin $1 + \Gamma_R$ (treapta ce sosește plus reflexia produsă).

În secțiunea următoare se dezvoltă o metodă mult mai generală de determinare a reflexiilor. Totuși, este interesant de urmărit coeficientul de reflexie produs de diferite terminații. Un scurtcircuit produce un coeficient de reflexie de —1 (pentru tensiune rezultantă 0). Toate terminațiile avînd o impedanță mai mică decît cea caracteristică produc coeficienți de reflexie

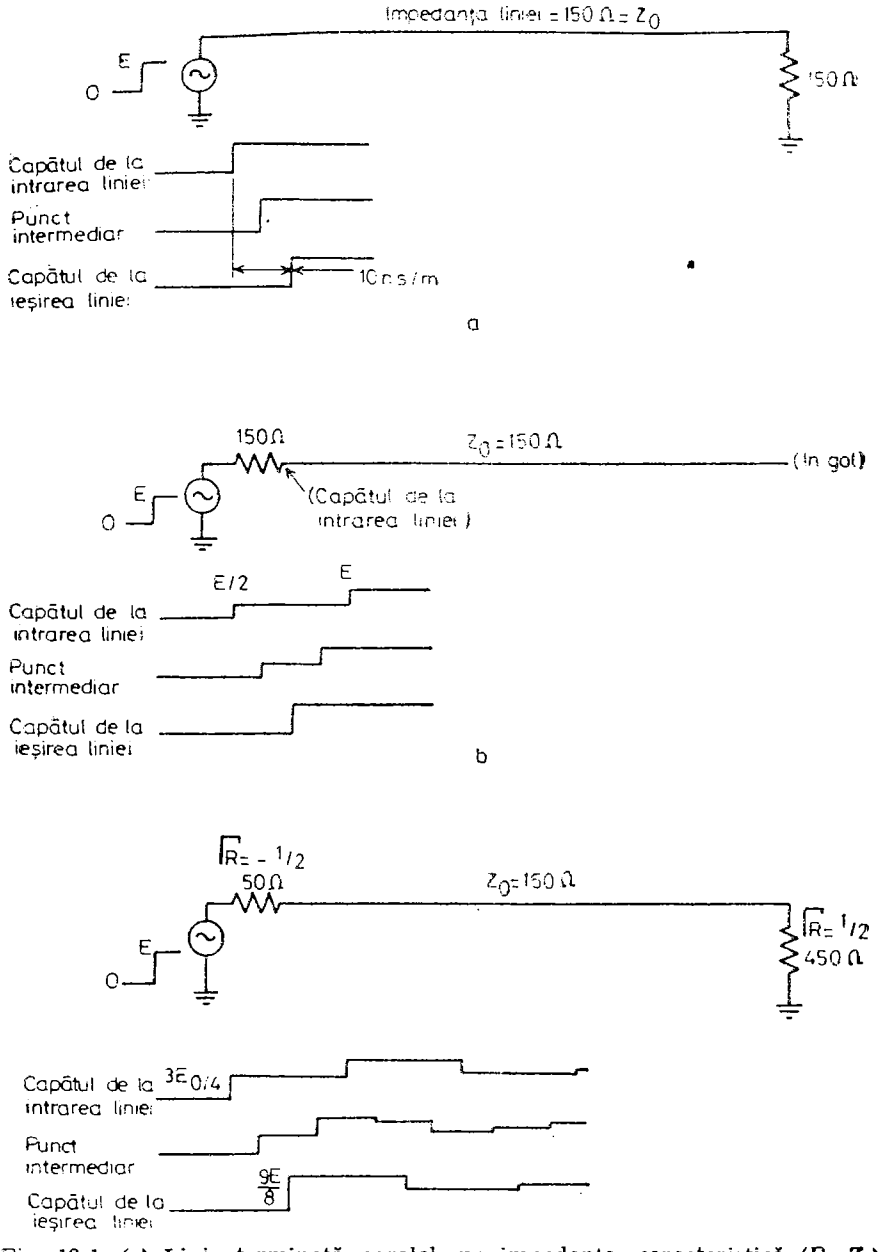


Fig. 12-1. *(a)* Linie terminată paralel pe impedanța caracteristică ($R=Z_0$); *(b)* Linie terminată serie pe impedanța caracteristică ($R=Z_0$); *(c)* Linie incorect conectată ($R_1=Z_0/3$, $R_2=3Z_0$).

negativi (reducînd tensiunea de ieşire) şi toate terminaţiile cu o impedanţă mai mare decît cea caracteristică produc reflexii pozitive (mărind tensiunea de ieşire), ce se apropie asimptotic de un coeficient de reflexie de +1 (tensiune dublă) pentru o impedanţă infinită. O terminaţie de trei ori mai mare decît impedanţa caracteristică produce un coeficient de reflexie de 50% (supracreştere de 50%).

Deşi ecuaţia 12-1 poate fi folosită şi în cazul terminaţiilor complexe (incluzînd capacităţile şi inductanţele), o abordare mai intuitivă generează, de asemenea, rezultate bune. De exemplu, o terminaţie capacitivă absoarbe curentul :

$$I = C\frac{dv}{dt} \tag{12-4}$$

în timpul frontului crescător al semnalului incident. Acest curent poate fi gîndit ca un curent reflectat ce va produce un impuls reflectat de tensiune cu valoarea $-IZ_0$, suprapus peste reflexiile produse de terminaţia rezistivă. O terminaţie capacitivă poate fi concepută ca producînd un puls reflectat pe fronturile semnalului incident, care se opune semnalului respectiv şi îi întîrzie prin urmare variaţia. Sarcinile capacitive plasate de-a lungul liniei (de exemplu, intrările de porţi) produc de asemenea, impulsuri reflectate ce se propagă pe linie *în ambele direcţii*. Dacă astfel de sarcini sînt distribuite relativ uniform de-a lungul liniei, astfel încît timpul de propagare între ele este *mai mare decît* timpul de creştere al semnalului, reflexiile nu vor fi aditive.

Segmentele de linie fără terminaţie, prin care se cuplează de exemplu intrarea unei porţi, cum este cel din figura 12-2, crează în semnal discontinuităţi cu *durata egală cu întîrzierea dus-întors produsă pe derivaţia respectivă.* În acest interval, impedanţa rezultantă este $Z_0/2$, deoarece linia şi segmentul sînt comandate în paralel. Se produce astfel o reflexie cu

$$\Gamma_R = \frac{Z_0/2 - Z_0}{Z_0/2 + Z_0} = \frac{-Z_0/2}{3Z_0/2} = -\frac{1}{3} \tag{12-5}$$

până cînd reflexia de la capătul segmentului readuce semnalul la normal. (Se consideră că intrarea porţii nu încarcă, practic, derivaţia de linie şi semnalul $(2/3)E$ se reflectă la capătul derivaţiei, devenind $(4/3)E$, iar apoi este redus la E prin coeficientul de reflexie $-1/3$ ce apare prin comanda liniei principale

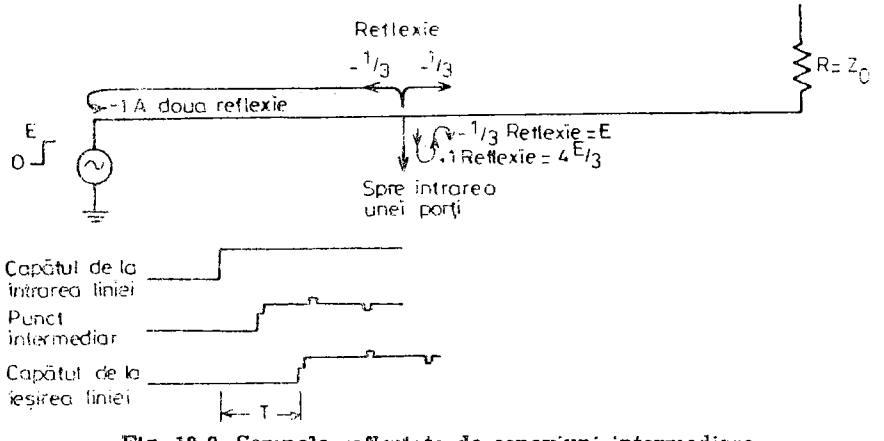


Fig. 12-2. Semnale reflectate de conexiuni intermediare.

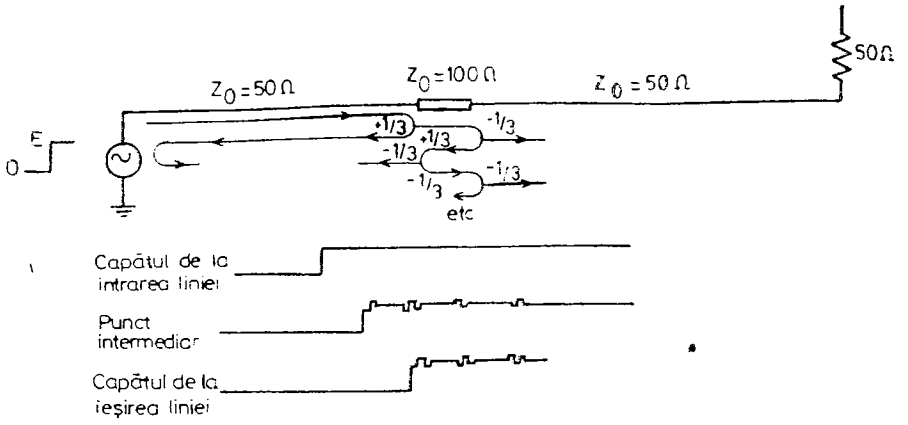


Fig. 12-3. Semnale reflectate datorită unui conector nepotrivit.

în ambele direcţii). Şi în acest caz, efectele reflexiilor nu vor fi aditive dacă derivaţiile sînt suficient de distanţate, de-a lungul liniei, încît reflexiile să nu se suprapună. Este posibil totuşi ca reflexiile să se suprapună dacă derivaţiile sînt echidistante (de exemplu, se poate suprapune *reflexia la capătul sursei* a reflexiei pe prima derivaţie cu reflexia la cea de-a doua derivaţie). Dacă *durata* reflexiilor este mai mică decît valoarea minimă a timpului de stabilire a logicii utilizate, efectul lor poate fi, de obicei, ignorat. Discontinuităţile în semnal produse de impedanţa conectorului pot fi, *în mod obişnuit, ignorate,* deoarece durata reflexiilor pe care le produc (vezi fig. 12-3) este egală cu timpul în care semnalul parcurge dus-întors conectorul. De fapt, dacă durata reflexiei va fi mai redusă decît timpul de creştere al semnalului, efectul nu va mai fi vizibil ca o „crestătură". Va fi reflectată doar *panta* semnalului, cu un efect mult mai puţin observabil. Chestiunea importantă care trebuie observată este faptul că intrarea bistabilului *integrează efectul reflexiilor,* dacă perioada lor este suficient de scurtă.

12.4. SOLUŢIA GRAFICĂ A PROBLEMELOR DE REFLEXIE

Deşi conceptul de coeficient de reflexie este utilizabil în cazul terminaţiilor liniare, majoritatea circuitelor logice au caracteristici curent/tensiune *neliniare.* Putem totuşi analiza problemele de reflexii, folosind o metodă grafică simplă. Prin această metodă, putem găsi tensiunile ce satisfac ecuaţia $Z_0 = E/I$ impusă de către linia de transmisie şi, în acelaşi timp, satisfac relaţia E/I specifică terminaţiilor. Vom putea astfel să citim direct pe diagramă tensiunile existente la fiecare moment de transmisie.

Figura 12-4 prezintă o soluţie grafică a problemei terminaţiei rezistive din figura 12-1*c* cu $E = 1$ V. Desenăm pur şi simplu un *plan al impedanţelor* cu tensiunea pe axa verticală şi curentul pe axa orizontală. Pentru simplificare, *vom scala axele tensiunii şi curentului astfel încît orice dreaptă dusă la 45° să reprezinte valori de tensiune şi curent într-un raport egal cu impedanţa*

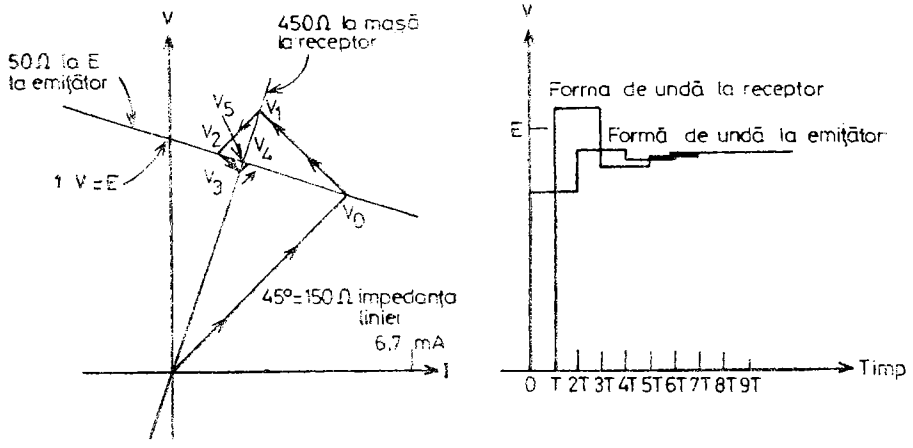


Fig. 12-4. Soluția grafică a configurației din figura 12-1c, pentru $E=1$V; *(a)* planul impedanță; *(b)* forme de undă.

caracteristică ($Z_0 = E/I$) a conexiunii. Deoarece în figura 12-1*c*, impedanța este de 150 Ω, 1 V pe axa Y reprezintă o aceeași deplasare ca $1/150 = 6{,}7$ mA pe axa X.

Vom construi acum „drepte de sarcină" pentru circuitele aflate la fiecare capăt al firului de conexiune, imediat după tranziția analizată. Deoarece la ambele extremități se află impedanțe liniare, putem construi dreptele de sarcină prin cîte două puncte tensiune/curent. Capătul depărtat va avea un curent nul la o tensiune nulă și $1/450 = 2{,}2$ mA la 1 V. Capătul sursei va avea curentul nul la 1 V iar pentru 6,7 mA (cu o cădere de potențial de $IR = 0{,}067 \times 50 = 0{,}33$ V) o tensiune de 0,67 V. Unind aceste puncte, obținem dreptele de sarcină ce indică toate combinațiile posibile de tensiune și curent la ambele extremități.

Deoarece orice linie desenată la 45° poate reprezenta valori posibile de tensiune și curent pe linia de transmisiune, putem acum afla în mod grafic tensiunile. Tensiunea inițială la capătul dinspre sursă o aflăm construind o linie la 45° ce trece prin origine (tensiune și curent zero) pînă cînd intersectează dreapta de sarcină a extremității de emisie. V_0 este astfel tensiunea inițială la capătul dinspre sursă, satisfăcînd relațiile E/I ale liniei de transmisie și ale rezistenței de 50 Ω în punctul de +1 V. Putem acum afla tensiunea la capătul depărtat (cînd semnalul ajunge acolo), ducînd o dreaptă la 45° din V_0, pînă cînd intersectează dreapta de sarcină de la capătul depărtat, la V_1. Din acest punct ducem o altă dreaptă la 45° pînă cînd intersectează din nou dreapta de sarcină — corespunzătoare capătului dinspre sursă la V_2, care este astfel tensiunea la capătul dinspre sursă cînd reflexia se întoarce. O altă linie la 45°, ce intersectează dreapta de sarcină corespunzătoare capătului depărtat, ne dă tensiunea de la capătul depărtat (V_3) după trei perioade de propagare ($3T$), cînd reflexia ajunge din nou la capătul respectiv. Fiecare tensiune găsită poate fi transferată imediat pe formele de undă (vezi fig. 12-4*b*). Putem să căutăm oricît de multe noi valori, dar la un anumit punct devine evident că ele se apropie de o valoare finală.

Este interesant de remarcat că dacă am fi mediat salturile formei de undă, s-ar fi obținut o exponențială $V = \exp(-t/RC)$, unde R este rezistența paralel distribuită, iar C este capacitatea totală a liniei. Cînd lucrăm cu sem-

nale lente (ca în cazul CMOS), putem uneori uita reflexiile și liniile de transmisie, gîndindu-ne doar la *încărcarea capacității conexiunilor*. Faptul că cele două modalități de raționament dau aceleași rezultate ne asigură că rezultatele sînt bune. Presupunerea că rezistența și capacitatea sînt concentrate duce la ignorarea unor detalii neimportante în cazul circuitelor lente.

Forța abordării grafice este pusă în evidență în cazul în care operăm cu linii de sarcină *neliniare*. Figura 12-5 prezintă liniile de sarcină ale diferitelor intrări și ieșiri de CI. Să observăm faptul că deoarece sensul pozitiv al curen-

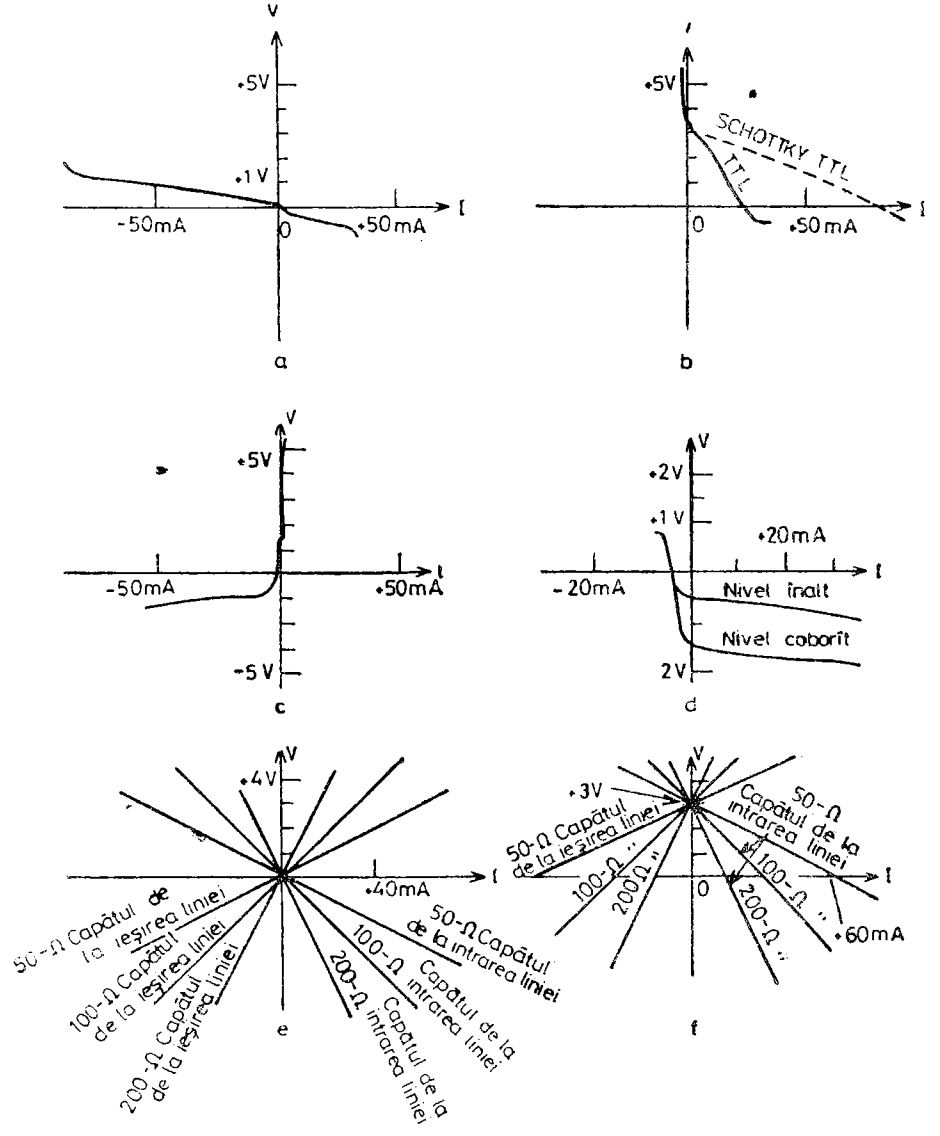


Fig. 12-5. Exemple de caracteristici de sarcină (Z_0=100 Ω corespunde la 45°): *(a)* TTL cu ieșirea la nivel coborît (0 logic); *(b)* TTL cu ieșire la nivel înalt (1 logic); *(c)* intrarea porții TTL; *(d)* ECL cu ieșirea la nivel coborît și înalt; *(e)* rezistențe la masă; *(f)* rezistențe la +3V.

tului este *spre* linie (intră) la extremitatea sursei și *dinspre* linie (iese) la extremitatea sarcinii, aceeași valoare de rezistență va fi prezentată diferit, în funcție de capătul la care ne situăm (vezi fig. 12-5*e* și *f*). Procedura de determinare a formelor de undă generate de reflexii este și în cazul terminațiilor neliniare, exact aceeași ca în cazul rezistențelor simple din figura 12-4.

În condiții reale, proiectantul dorește să realizeze un sistem de transmisie a semnalului care să funcționeze corect. Unul din punctele tari ale soluției grafice constă în faptul că *este ușor de văzut ce efect vor avea unele modificări ale terminației, ca introducerea de diode sau rezistențe de limitare, asupra formelor de undă rezultate.* Proiectantul poate astfel să „jongleze" cu liniile de sarcină pînă obține un rezultat satisfăcător.

Figura 12-6 ilustrează un exemplu practic : cînd s-a introdus familia TTL porțile nu aveau pe intrări diode de limitare. Așa cum se arată în figura 12-6*a*, aceasta înseamnă existența unor oscilații, după o tranziție negativă, care pot să traverseze pragul logic și să producă o funcționare improprie. Pentru a soluționa această problemă în 1970 a fost prevăzută o diodă de limitare spre masă, în toate circuitele TTL. Așa cum se arată în figura 12-6*b*, această diodă reduce suficient de mult oscilațiile pentru a preveni funcționarea logică improprie. Este ușor de prevăzut, doar prin urmărirea figurii 12-6*a*, care va fi efectul introducerii diodei.

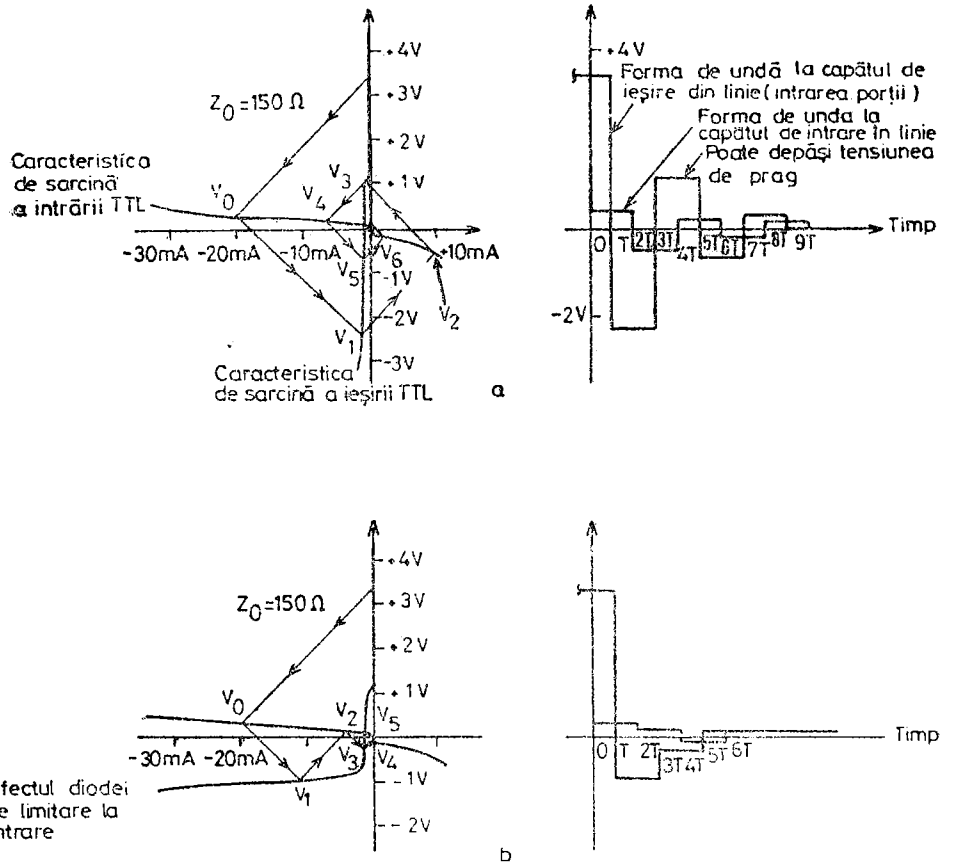


Fig. 12-6. *(a)* Tranziția în 0 a unui circuit TTL fără diodă de limitare pe intrare. *(b)* Tranziția în 0 a unui circuit TTL cu diodă de limitare pe intrare.

12.5. EFECTUL REFLEXIILOR ASUPRA FUNCȚIONĂRII SISTEMULUI

Reflexii de forma celor din figura 12-6*a* pot să creeze probleme *dacă apar pe semnalul de tact chiar și atunci cînd este utilizat un tact foarte lent.* Dacă oscilațiile pe acest semnal au o perioadă destul de mare, o singură tranziție de tact va arăta de fapt ca două sau mai multe tacte. Este posibil deci ca, în unele cazuri extreme, *sistemul să parcurgă mai multe cicluri de tact* (parazite). Deoarece lățimea minimă a tactului, necesară pentru a produce basculare bistabililor este de același ordin de mărime cu timpul de stabilire al familiei logice în cauză, trebuie să urmărim cu grijă liniile de tact cu lungimi mai mari decît cele menționate în Secțiunea 12-1, la *orice* frecvență de tact.

Modificarea semnalelor logice este mult mai puțin critică dacă frecvența de tact nu se apropie de limitele familiei logice în cauză. Dacă nivelele logice se stabilizează la valorile lor finale cu mult înaintea următorului tact, nu vor apare probleme legate de reflexii sau chiar de diafonie. Dacă totuși frecvența de tact se apropie de limită, în determinarea frecvenței maxime de tact trebuie să se ia în considerare întîrzierea pe conexiuni. Deoarece în această situație, deciziile privind bistabilii se iau foarte aproape de modificările semnalelor logice, oscilațiile produse de reflexii pot să producă o funcționare nesigură. Dacă, de exemplu, o intrare de bistabil se prezintă ca forma de undă din capătul depărtat (figura 12-6*a*) și tranziția de tact apare la $5T$, cel mai mic zgomot va produce o funcționare incorectă.

Marginea de zgomot este considerată în mod normal a fi diferența dintre nivelul de c.c. garantat al familiei logice și tensiunea sa de prag de intrare, din cazul cel mai defavorabil. În cazul TTL nivelul 1 de *ieșire* este garantat a fi mai mare de 2,4 V, iar *pragul* de 1 este garantat a fi sub 2 V. Marginea de zgomot în c.c. în cazul cel mai defavorabil este deci de 2,4—2 = 0,4 V. După cum se arată în figura 12-7, o poartă plasată la capătul dinspre generator al unei linii de transmisie de 100 Ω nu va vedea un nivel 1 pînă la $2T$ (2 × × 5 ns/m=10 ns/m de conductor). De asemenea o poartă plasată la capătul depărtat va vedea numai 3,2 V/2,75 V = 86% din tensiunea finală, pînă la $6T$ (15 ns/m). În felul acesta, în cazul cel mai defavorabil, înainte de $3T$, tensiunea văzută de o poartă la capătul depărtat este de 2,4 V × 86% =

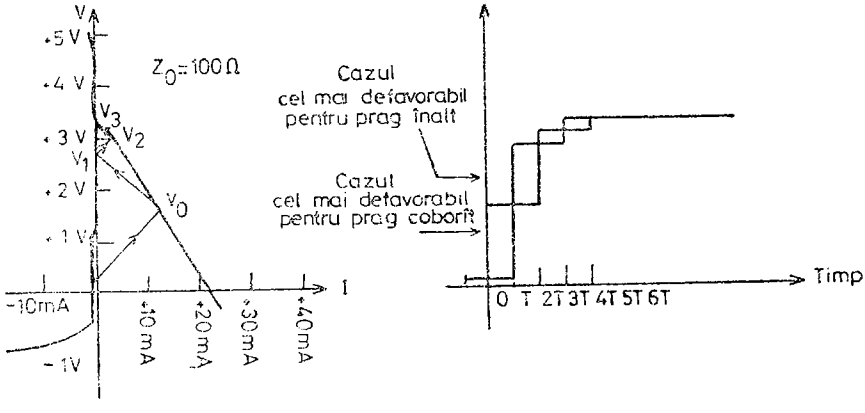


Fig. 12-7. Tranziția în 1 a unei ieșiri TTL.

= 2,06 V, ceea ce conduce la o *margine de zgomot* **dinamică** de numai 0,06 V. Prin urmare, pentru o funcționare sigură și în cazul cel mai defavorabil, cu logica TTL obișnuită, trebuie să presupunem o întîrziere ($3T$) de 15 ns/m de cablu. Putem să presupunem că bascularea are loc pe prima tranziție, și deci o întîrziere pe fire de 5 ns/m, dacă folosim circuite cu impedanță mai mică de ieșire în starea 1. Seria TTL Schottky (74S) are o impedanță de ieșire în 1, ce este apropiată de jumătatea celei corespunzătoare din TTL (vezi fig. 12-5*b*). Se poate deci conta pe ea pentru a produce un nivel 1 adecvat încă din prima trecere.

12.6. ZGOMOTUL PE MASĂ, PE CONEXIUNILE DE ALIMENTARE ȘI ZGOMOTUL PRODUS DE INJECȚIA DE CURENT

Un lucru interesant pe care îl observăm din figurile 12-1, 12-6 și 12-7 este acela că puțin contează ceea ce se află la capătul depărtat al unei bucăți de conductor pînă cînd trece întîrzierea de propagare dus-întors ($2T$). *Indiferent ce are la capătul depărtat segmentul de conductor se prezintă ca o rezistență egală cu impedanța caracteristică, pînă cînd a trecut suficient timp pentru ca semnalul să parcurgă dus-întors linia.* Dacă se folosesc fire obișnuite pentru conexiunile de sursă și masă, aceasta poate să conducă la probleme serioase în cazul circuitelor logice rapide — chiar dacă frecvența tactului este redusă.

În figura 12-8 se evidențiază modul în care zgomotul de masă poate afecta funcționarea bistabililor. Pentru simplitate, se presupune că firul de pe ieșirea bistabilului este terminat pe Z_0, pentru a elimina reflexiile și

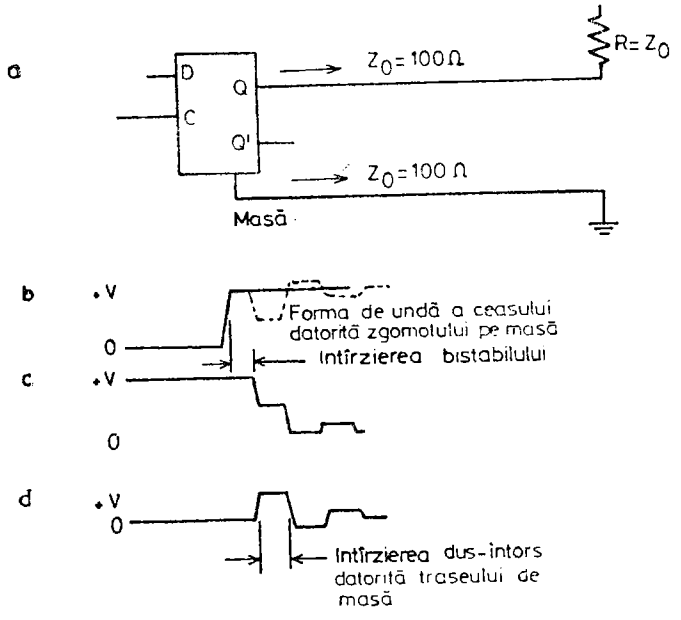


Fig. 12-8. Efectul unui traseu de masă de lungime mare : *(a)* Circuit cu probleme date de traseul de masă ; *(b)* Ceasul ; *(c)* Ieșirea Q a bistabilului ; *(d)* Semnalul pe traseul de masă.

pentru ca bistabilul să nu funcționeze incorect. Atunci cînd un tact impune bistabilului să se reseteze, semnalul de ieșire se *divizează inițial în mod egal* între firul de masă și firul de pe ieșirea Q, deoarece amîndouă arată ca aceeași impedanță $Z_0 = 100\ \Omega$. Aceasta se întîmplă deoarece orice curent de la ieșirea Q provine din firul de masă. După ce a trecut intervalul de propagare dus-întors, tensiune pe masă revine la 0V (datorită reflexiei cu -1). Între timp, tensiunea pe intrarea de tact a bistabilului arată ca jumătate din nivelul său real, datorită tensiunii $+V/2$ de pe terminalul de masă al bistabilului (vezi fig. 12-8 *b*). Dacă durata de întîrziere dus-întors este suficient de mare, bistabilul poate să fie comandat, în realitate de două ori (două tacte).

În cazul circuitelor TTL acest efect este chiar mai accentuat, deoarece în timpul comutării circuitul ca atare produce o absorbție puternică de la sursa de curent *atît* tranzistorul „de sus" *cît și* cel „de jos" din etajul de ieșire al CI fiind deschise. Acest salt de curent produce o tensiune IZ_0, care se adaugă la cea prezentată în figura 12-8 *d*. În traseul de sursă de (+) se produce un puls similar spre minus. Dacă durata (dată de întîrzierea dus-întors) și amplitudinea acestui puls sînt excesive, se pot pierde, datorită întreruperii alimentării, anumite stări ale bistabililor.

Deși logica CMOS are pe alimentare un salt similar, nu există probleme, în cazul lungimilor de cablu obișnuite, deoarece circuitele sînt mult mai lente. Dacă circuitele logice sînt lente, impulsurile de zgomot, cum sînt cele indicate punctat în figura 12-8 *b*, nu cauzează funcționări eronate deoarece pur și simplu bistabilii nu pot să răspundă atît de repede. Un metru de traseu de cablaj imprimat de exemplu, produce un impuls de zgomot cu o durată de numai 2×6 ns $= 12$ ns. Deși fără efect în cazul CMOS, el poate produce probleme în cazul circuitelor mai rapide, ca 74 *S* sau ECL.

O modalitate foarte eficace de reducere a zgomotului pe masă este reducerea impedanței conductoarelor de masă transformîndu-le într-un *plan de masă*. Un plan de masă este, de exemplu, o folie continuă de cupru, constituind de obicei o față a cablajului imprimat. Impedanța legăturii de masă devine atît de scăzută încît semnalul apare aproape în întregime pe traseul de semnal. Se utilizează adesea un *cablaj multistrat*, ca în figura 12-9. Acest tip de cablaj furnizează o masă excelentă, dar este destul de scump, deoarece se realizează prin lipirea unui cablaj dublu-placat cu unul mono-placat. O metodă și mai bună — dar și mai scumpă — constă în laminarea unui strat izolator între două cablaje *dublu-placat*, producînd un total de patru straturi. Unul din planele interne este folosit ca *plan de masă* (ground plane) iar celălalt ca *plan de alimentare* (power plane). În felul acesta, se poate reduce, în mare măsură, zgomotul atît pe masă cît și pe sursă.

Cu o proiectare îngrijită, putem evita utilizarea cablajelor multistrat (ce sînt de 3-5 ori mai scumpe) prin folosirea pentru interconexiuni, a cablajelor imprimate dublu strat. În minicomputerele moderne se utilizează plăci mari, aproximativ 40×40 cm^2, dublu strat, pentru interconectarea circuitelor TTL sau TTL Schottky. Un plan de masă destul de eficace este realizat

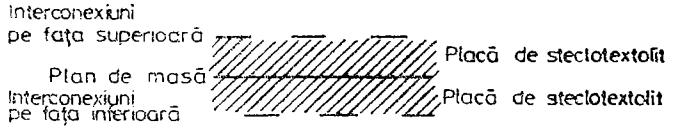


Fig. 12.-9. Secțiune transversală printr-o placă cu cablaj multistrat.

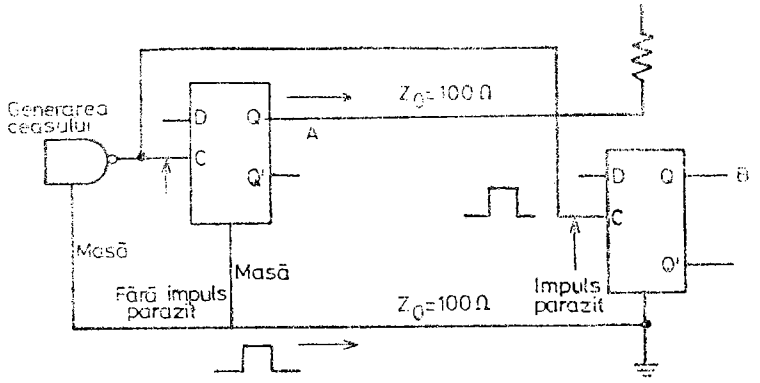


Fig. 12-10. Efectul cablajului asupra zgomotului pe firul de masă.

printr-o rețea de conexiuni de masă și alimentare ce merg la fiecare circuit. Se distribuie capacități de șuntare pe înaltă frecvență plasate între sursă și masă, pe întreaga placă — cu o medie uzuală de un condensator de 0,01 μF la fiecare 25 capsule de circuite integrate. Rezultatul este că în locul unei singure căi de masă, ca în figura 12-8 *a*, se formează *un mare număr de căi paralele în rețeaua de conexiuni de sursă și masă*. Căile de masă paralele sînt cu atît mai eficace cu cît două circuite sînt mai depărtate. Conexiunile de masă și sursă sînt realizate cît mai late (în limita spațiului existent) pentru a reduce și mai mult impedanța. Printr-o topologie îngrijită, se poate reduce în mod considerabil costul plăcii de circuit. Aceste economii pot fi depășite de costul verificărilor suplimentare și al reproiectării topologiei dacă volumul producției este mic. Deoarece problemele generate de zgomote se pun în evidență în situații limită, nu este suficientă funcționarea corectă a prototipului. De asemenea, se vor verifica, cu un osciloscop de înaltă frecvență toate punctele de masă și sursă, în cursul funcționării în regim nominal — de preferat în timpul rulării unor programe de diagnoză.

Zgomotul pe masă poate fi sau nu o problemă, în funcție de topologia concretă a cablajului. De exemplu, în figura 12-10 se prezintă circuitul din figura 12-8 *a*, avînd masa driver-ului de tact conectată *direct* la masa bistabilului. Aici nu vor apare probleme, deoarece impulsul de zgomot pe masă apare și pe semnalul de tact, astfel încît nu există un zgomot *net*. Va exista însă o problemă de zgomot, în cazul bistabilului *B*, deoarece impulsul de zgomot generat de bistabilul *A* apare pe semnalul de tact, dar nu și pe masa lui *B*.

Un alt tip de problemă de zgomot, datorată *injecției de curent**, este ilustrată în figura 12-11. O intrare Schottky TTL ia pînă la 2 mA pentru a fi ținută în 0. Să presupunem că intrările *A* și *B* sînt ambele 0, dar că *B* ia inițial partea cea mai mare din cei 2 mA. Cînd intrarea B trece în 1, intrarea *A* va primi o treaptă de curent de 2 mA. Dacă impedanța conductoarelor este de 150 Ω, aceasta va produce un semnal de tensiune $E = IZ = 0{,}002 \times \times 150 = 0{,}3$V, care va dura cît întîrzierea dus-întors la ieșirea porții driver. Deoarece 0,3V este mai puțin decît marginea de zgomot TTL, nu va produce probleme. Totuși, dacă se injectează simultan mai mulți curenți similari, în aceeași linie, pot să apară probleme. De exemplu, dacă toate *intrările D*

* current dumping (n.t.).

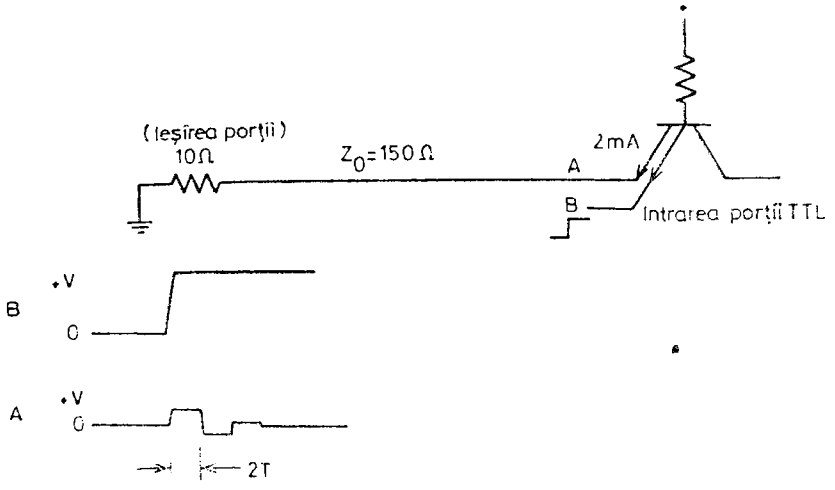


Fig. 12-11. Zgomotul datorat injecţiei de curent.

ale unui registru constituit din bistabili *D* separaţi, se schimbă în acelaşi timp, poate să apară o injecţie de curent pe intrarea de tact a fiecărui bistabil. Curentul *total* injectat va fi *suma* curenţilor corespunzători ai bistabililor individuali. Dacă întîrzierea de propagare în dublu sens la driverul de tact este suficientă, bistabilii pot să „vadă" ca tact impulsul de zgomot, chiar dacă ieşirea driverului rămîne în 0.

12.7. DIAFONIA ÎN INTERCONEXIUNILE LOGICE

O altă problemă ce solicită o atenţie deosebită în cazul logicii rapide este cuplajul electromagnetic dintre traseele de semnal, sau *diafonia*. Cînd folosim circuite logice rapide, diafonia *spre liniile de tact* este o problemă la *orice* frecvenţă a tactului, la fel ca şi reflexiile. Dacă nu ne apropiem de frecvenţa de tact maximă admisibilă, *putem ignora diafonia pe traseele de semnal logic* ; aceasta datorită faptului că diafonia este produsă numai de *tranziţiile* de semnal. Într-un sistem sincron, toate tranziţiile de semnal au loc imediat după tact. Dacă semnalele logice au timp să se stabilizeze după tact, ele vor atinge valoarea finală corectă înaintea următorului tact, în ciuda unor nivele temporar incorecte la ieşire, cauzate de diafonie. O excepţie o constituie diafonia *cu revenire la nivelele logice anterioare*, ce apare în cazul sistemelor ECL. Acest tip de diafonie poate cauza oscilaţii datorită formării unor bucle de reacţie pozitivă. În cazul TTL, acest fenomen este improbabil deoarece în regiunea de tranziţie în care există cîştig direct se rămîne foarte puţin timp.

O modalitate simplă de reducere a diafoniei este utilizarea unui cablaj imprimat multistrat, cu un plan de masă, ca în figura 12-9. Planul de masă reduce impedanţa conductoarelor individuale şi intercepteză parţial cîmpurile electrostatice (capacitive) şi electromagnetice (inductive) ce leagă cele două conductoare. Tensiunea de diafonie produsă pe o linie pasivă este, în principal, rezultatul *impedanţei mutuale* de cuplaj cu o linie generatoare de zgomot, ce produce o tensiune pe impedanţa caracteristică (către masă) a liniei.

Prin urmare, dorim o impedanță cît mai redusă spre masă și o impedanță mutuală cît mai mare spre linia generatoare de zgomot. Cînd distanța la planul de masă devine comparabilă cu distanța dintre linii, diafonia se reduce rapid.

Deși o placă multistrat dotată cu plan de masă reduce în mod sigur problemele legate de diafonie, varianta cu cablaj dublu-strat, mult mai puțin costisitoare este adesea acceptabilă, dacă topologia cablajului este îngrijit proiectată. Motivul este acela că pe lîngă toate traseele de masă și sursă, ce acționează ca un plan de masă, *conductoarele de semnal ajută la formarea unui astfel de plan.* Pentru a realiza o diafonie neglijabilă, conductorul de semnal de pe o față a cablajului intersectează de obicei alte trasee, de pe cealaltă față, la unghiuri drepte. Totuși, datorită faptului că la un tact oarecare, majoritatea semnalelor nu se modifică, rezultă că majoritatea conexiunilor se află în c.a., la masă și lucrează ca un plan de masă destul de bun. Singurele probleme reale apar atunci cînd două trasee „merg în paralel" pe distanțe mai mari. În acest caz, putem adesea intercala un conductor de masă între cele două fire (trasee). Problemele cu adevărat dificile, cum sînt cele ale distribuției tactului, pot fi adesea rezolvate prin închiderea conductorului de tact între conductoare de masă, sau chiar lipind pe placă fire împletite în perechi.

În producția de serie mică, planul de masă (posibil, și cel de alimentare) se justifică, probabil, avînd în vedere costul său, ce este mai redus decît cel corespunzător rezolvării problemelor suplimentare de topologie ce apar fără planul de masă. Oricum, verificarea nu trebuie considerată încheiată, pînă cînd formele de undă nu au fost urmărite, pe un osciloscop rapid. Dacă ne apropiem de frecvența limită de tact a circuitelor integrate, formele de undă vor fi verificate și în sensul asigurării unei margini dinamice de zgomot corespunzătoare.

12.8. DIAFONIA ȘI ZGOMOTUL ÎN CABLURI

Deși este posibil ca în cazul interconexiunilor logice să realizăm proiectarea astfel încît două trasee să nu se urmărească cîtuși de puțin, acest fapt nu este practic realizabil în cazul interconexiunilor lungi, realizate prin *cablu.* Ori de cîte ori lungimea cablului este suficient de mare pentru ca întîrzierea semnalului să fie comparabilă cu timpul de creștere, conceptul simplu de divizare a semnalului între impedanța mutuală și impedanța către masă, nu mai este util.

Așa cum se prezintă în figura 12-12, inductanța mutuală (L_{12}) și capacitatea mutuală (C_{12}), ce sînt distribuite de-a lungul cablului, produc diafonie pe măsură ce semnalul se deplasează pe linie. Zgomotul capacitiv se propagă din locul în care este produs, în ambele direcții cu aceeași polaritate ca și semnalul excitator, în timp ce diafonia inductivă se propagă în sens invers cu aceeași polaritate, iar *în sens direct cu polaritate opusă.* Efectul rezultant constă în *sumarea* efectelor capacitiv și inductiv *în cazul diafoniei la extremitatea de la generator* și *scăderea* lor *în cazul celei de la capătul de la receptor.* Dacă întîrzierea dus-întors pe linie este egală cu timpul de creștere al semnalului, diafonia de la capătul de la generator va avea un maxim. Deoarece diafonia de la capătul receptor este suma a două semnale de polaritate opusă, ea va fi mai redusă (putînd fi zero sau de polaritate inversă).

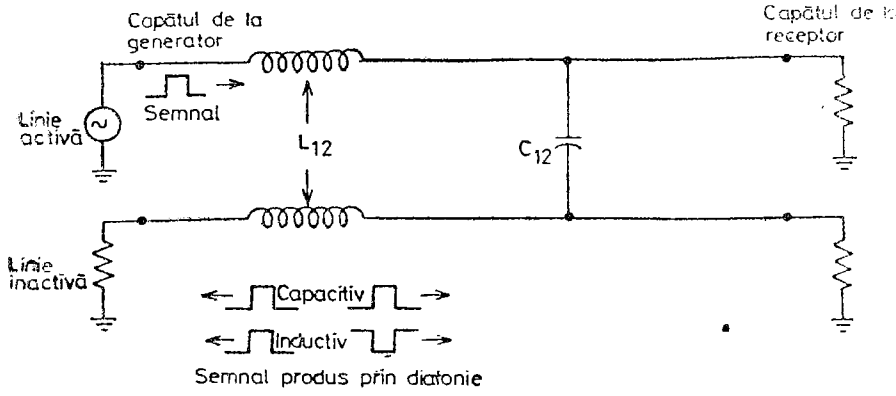


Fig. 12-12. Diafonia între traseele lungi.

Folosind un cablu coaxial sau un cablu plat cu dielectric (vezi fig. 12-13), semnalele pot fi trimise pînă la aproximativ 10 m folosind ca emițătoare și receptoare porți obișnuite. În cazul circuitelor ECL, este necesară drept terminație o rezistență; circuitele TTL Schottky pot fi însă folosite fără terminații. Cînd generăm semnal pe un cablu coaxial de 50 Ω cu un circuit TTL, va trebui, totuși, să utilizăm un circuit de comandă special, de curent mare (de ex. 74S140).

Cînd cablurile sînt duse la distanțe mai mari, problemele legate de zgomotul *extern* pot deveni semnificative. Pe măsură ce distanța crește, obținerea unei mase comune de bună calitate devine tot mai dificilă. Firele lungi de masă tind să acționeze ca antene și să capteze posturi de radio, zgomotul de rețea, zgomotul produs de instalațiile de aprindere a automobilelor sau de periile motoarelor electrice, ș.a.m.d. Dacă două echipamente distincte sînt cuplate la rețeaua de alimentare în puncte diferite, cablul de masă nu trebuie să fie cuplat la masa de semnal la ambele extremități. Dacă ambele extremități ale acestui conductor sînt legate la masa liniei de alimentare, așa cum se arată în figura 12-14, curenții de zgomot produși de motoare sau alte echipamente pe linia de alimentare se vor scurge și prin cablul de masă pentru logică, ce formează un traseu paralel. Acest curent poate să producă o tensiune de zgomot semnificativă. Dacă *o extremitate sau alta* nu este conectată la masa liniei de alimentare, calea paralelă este eliminată. În măsura în care masa de semnal este izolată, *carcasa* de metal poate fi legată la masa de protecție.

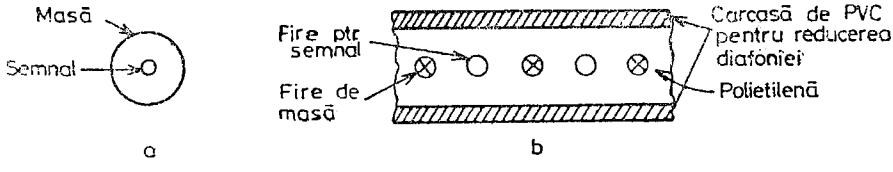


Fig. 12-13. Cabluri de interconectare: *(a)* cablu coaxial; *(b)* cablu plat, dublu dielectric.

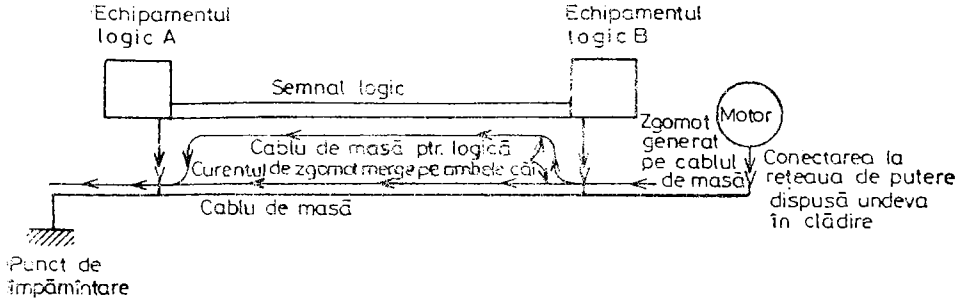


Fig. 12-14. Buclă de masă datorată rețelei de c.a. de alimentare.

12.9. TRANSMISIA DIFERENȚIALĂ A SEMNALELOR

Cînd se permite traseului de masă al semnalului să floteze la un capăt, zgomotul ce se produce pe conductorul de masă al unui astfel de cablu (în raport cu pămîntul) este fără importanță în măsura *în care un zgomot identic se induce în toate traseele de semnal.* Deoarece în timp ce masa de semnal este conectată la restul logicii din sistem, fiecare linie de semnal ajunge numai la o intrare de poartă, apare o sarcină *dezechilibrată* ce tinde să producă, din cauza zgomotului, o recepție defectuoasă a semnalului. Cu cît frecvența este mai ridicată cu atît dezechilibrul este mai mare, astfel încît adesea în sistemele logice de mare viteză se folosește o transmisie a semnalelor *pe mod diferențial.*

În cadrul acestei transmisii, fiecărei linii de semnal i se asociază o masă separată. Împletind masa asociată cu firul de semnal, zgomotul inductiv și diafonia sînt aproape identice, pentru traseele de masă și de semnal. Folosind un *receptor de linie cu intrare diferențială, semnalele de mod comun* sînt rejectate și se detectează numai *diferența* dintre tensiunile de masă și de semnal. Deoarece ambele intrări ale receptorului diferențial au aceeași impedanță, încărcarea celor două conductoare este identică. Deși putem lega, cu o oarecare reducere, a echilibrării, masele de la mai multe perechi împletite, la un singur punct de la driver, *conectarea maselor la ambele extremități conduce la dispariția avantajelor funcționării diferențiale.* Dacă masele sînt comune la ambele capete, curenții de masă se vor scurge în paralel prin toate conductoarele de masă. Diafonia către liniile de semnal va varia totuși în funcție de apropierea perechilor din cablu. Dezechilibrul ce rezultă generează un semnal diferențial, producînd o recepție incorectă a semnalelor.

Figura 12-15 prezintă un sistem diferențial de transmitere a semnalelor. Emițătorul diferențial injectează curent la o ieșire sau alta pentru a transmite un 1 sau un 0. Receptorul diferențial răspunde numai la *diferen-*

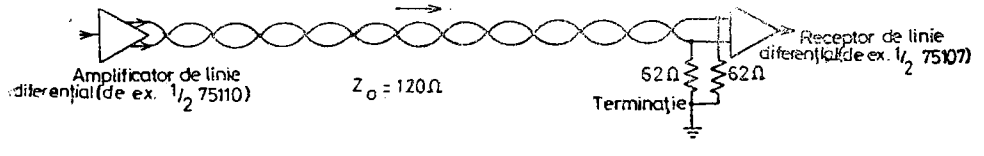


Fig. 12-15. Sistem de transmisie diferențială

țele de potențial între cele două intrări. Toate liniile unui cablu formează perechi diferențiale, separate. În cazul familiilor logice de foarte mare viteză, funcționarea diferențială este necesară pentru toate conexiunile, cu excepția celor foarte scurte. Regulile de conexiune utilizate în cazul ECL recomandă o transmisie pe cablu coaxial sau diferențială, pentru lungimi mai mari decît *25 cm*.

Plasînd terminații la ambele capete, se poate realiza un sistem bidirecțional, cu un receptor și un transmițător la fiecare capăt. De fapt, putem realiza un sistem *partiționat*, avînd emițătoarele și receptoarele împrăștiate de-a lungul liniei, astfel încît să servim mai multe dispozitive distincte cu un singur cablu. Masele logice ale diferitelor sisteme trebuie să fie legate împreună printr-un conductor gros *separat*. Tensiunile de zgomot dintre diferitele mase nu vor produce dificultăți, *în măsura în care se încadrează în gama tensiunilor de intrare de mod comun a receptorului diferențial de linie* (± 3V în cazul circuitului 75107).

Dacă cablul este foarte lung sau sistemul este plasat într-un mediu foarte zgomotos (de exemplu, lîngă o antenă radar), zgomotul de mod comun poate atinge ușor valori prea mari pentru intrarea receptorului diferențial. Deși se pot utiliza divizoare de tensiune, plasate pe intrarea receptorului diferențial, pentru a mări gama de mod comun, în punctul respectiv, pentru liniile într-adevăr lungi se va utiliza *cuplajul prin transformator*. Transformatoarele de impulsuri ecranate, de mici dimensiuni sînt ieftine și *pot rejecta zgomotul de mod comun de mii de volți*. Singura problemă este că prin tarnsformator trec doar componentele de înaltă frecvență, astfel încît semnalele transmise trebuie codate astfel încît să nu pretindă un răspuns în curent continuu. Izolatoarele optice oferă o rejecție similară a zgomotului de mod comun și asigură, în același timp, un răspuns pînă în curent continuu. Deși răspunsul de înaltă frecvență al izolatoarelor optice este în prezent destul de limitat, ele vor fi în viitor foarte utile în transmisia de date.

Transmisia de date cu cuplaj prin transformator, pentru distanțe lungi a fost dezvoltată în mare măsură de către industria telefoanelor pentru transmisia vocii prin modulația de impulsuri în cod (PCM). De obicei, se folosește cablu telefonic obișnuit, izolat cu clorură de polivinil (PVC) sau cu hîrtie, în perechi împletite, pentru a transmite semnale digitale, cu o frecvență de 6,3 MHz, la distanțe de pînă la 1 300 m. Folosind cabluri de calitate mai bună, în perechi împletite, distanța dintre receptoarele digitale poate să crească pînă la 5 km. În cazul cablurilor coaxiale, se pot transmite semnale digitale cu frecvența de 60 MHz, cu o distanță între receptoare de 1 700 m. Formatul de transmisie utilizat în aceste sisteme telefonice este indicat în figura 12-16 *a*. Pentru a indica un 1, se transmite un impuls, în timp ce absența impulsului semnifică 0. Pentru a preveni formarea unei componente de curent continuu, se transmit impulsuri cu polaritatea (+) sau (—), în mod alternativ. Sistemul se numește cu „alternarea polarității unității" (AMI — alternate mark inversion).

Deși sistemul telefonic utilizează un factor de umplere de 50%, se poate realiza un sistem simplu prin utilizarea unor semnale cu un factor de umplere de 100%, așa cum se arată în figura 12-16 *b*. Deoarece nivelul mediu al semnalului este întotdeauna zero, transformatorul utilizat trebuie să aibă un răspuns de joasă frecvență suficient doar pentru o urmărire corectă fără o „cădere" semnificativă a palierului, pe durata unui bit. Pentru transmisiile de mare viteză, poate fi astfel utilizat un transformator miniatură cu numai cîteva spire, cu miez „în *H*" sau toroidal din ferită. Folosind o înfășurare cu

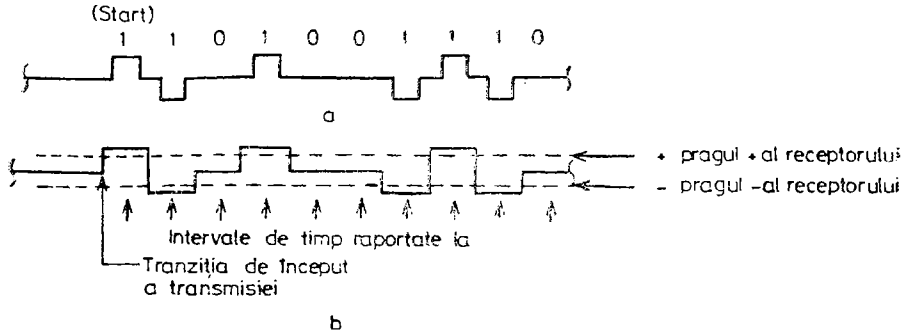


Fig. 12-16. Format de transmisie AMI : *(a)* factor de umplere 50% ; *(b)* factor de umplere 100%.

priză mediană pe o parte a transformatorului, circuitele de driver și de prag nu trebuie să fie bipolare. Comandînd o secțiune a înfășurării cu priză mediană produce un puls (+) de ieșire, în timp ce pulsul (—) se obține prin comanda celeilalte secțiuni.

Începînd fiecare secvență de date (sincronizată de un cristal de cuarț) cu un „puls de start", se poate genera digital o secvență de tacte pentru a transfera ieșirile, trecute printr-un OR, ale circuitelor de prag (+) și (—) la intrarea unui registru de deplasare. Un sistem simplu, realizat cu TTL, poate transmite date serie pînă la 20 MHz (descris în Ref. 1). O singură pereche de transformatoare și de circuite emițător/receptor poate astfel transmite serie mulți biți. Plasînd terminații la fiecare capăt al liniei și asigurîndu-ne că emițătoarele se află în starea de mare impedanță atunci cînd nu sînt activate, putem realiza ușor un sistem bidirecțional divizat în timp.

Dacă datele se transmit în mod continuu într-o direcție, putem realiza un sistem și mai simplu, de modulație în fază, cum este cel prezentat în figura 12-17. În acest sistem, semnalul de la emisie este pur și simplu obținut prin XOR între semnalul de date și un tact cu factor de umplere de 5 %. Acest tip de semnal este autosincronizat, prezentînd o tranziție pe fiecare bit. Declanșînd un monostabil, cu o durată egală cu 3/4 din cea a bitului, tranzițiile intermediare pot fi ignorate. Decodarea implică pur și simplu urmărirea direcției tranziției de tact (sau a polarității semnalului, înaintea tranziției de tact). Spre deosebire de sistemul AMI, acesta nu se poate porni și opri. Așa cum se arată în figura 12-17 *a*, la apariția semnalului se produce o componentă de c.c. ce produce funcționarea nesigură a circuitului de prag, pînă cînd „căderea" transformatorului îndepărtează componenta de c.c. Un avantaj al acestui sistem, în cazul utilizării sale pentru semnale continue, este acela că pragul este fixat la 0V, astfel încît semnalul poate avea *orice*

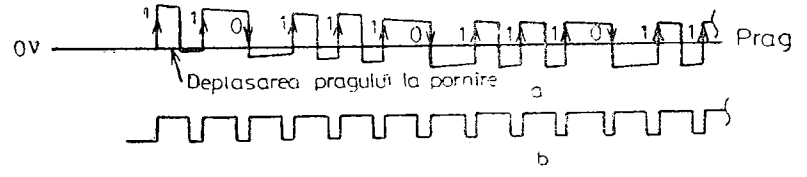


Fig. 12-17. Modulație în fază utilizînd un transformator : *(a)* semnalul ; *(b)* ceasul perioada=3/4 bit.

amplitudine. Alte sisteme, ca AMI, de exemplu, pretind că pragul să fie fixat la jumătate din amplitudine pentru ca să se obțină optimumul de performanță, în prezența zgomotului.

EXERCIȚII

1. (*a*) Calculați coeficienții de reflexie și primele trei nivele de tensiune pentru circuitul din figura 12-1*c*, dar cu $Z_0 = 100\ \Omega$.
   (*b*) Folosind procedeul grafic deteminați forma de undă până la 5T.
2. Desenați linia de sarcină până la +3V pentru o diodă la capătul îndepărtat al liniei.
3. Redesenați toate dreptele de sarcină din figura 12-5 considerînd că 45° corespunde la $Z_0 = 50\ \Omega$.
4. Desenați forma de undă în punctul median al liniei din figura 12-6*b*. Faceți reprezentarea pe o scară a timpului în nanosecunde (în loc de T), pentru un cablu de 1 m cu o întîrziere de 5ns/m.
5. Găsiți formele de undă, la capătul de la generator și de la receptor, prin metoda grafică pentru tranziția pozitivă a unui semnal la ieșirea unei porți TTL Schottky pe o linie cu $Z_0 = 100\ \Omega$.
6. Găsiți prin metoda grafică formele de undă la capătul de la generator și de la receptor al unei linii atacate de un circuit ECL. Impedanța caracteristică a liniei este de 100 Ω și este terminată cu o rezistență de 100 Ω conectată la −2 V.
   (*a*) Pentru tranziția pozitivă a semnalului.
   (*b*) Pentru tranziția negativă a semnalului.
7. Găsiți prin metoda grafică formele de undă la capătul de la generator și de la receptor și în punctul median pentru o linie cu impedanța caracteristică de 100 Ω, neterminată, atacată de o poartă ECL. Poarta ECL determină un curent de 160 μA către minus.
   (*a*) Pentru tranziția pozitivă a semnalului.
   (*b*) Pentru tranziția negativă a semnalului.
8. Găsiți prin metoda grafică formele de undă la capătul de la generator în punctul median și la capătul de la receptor al unei linii lungi cu $Z_0 = 100\ \Omega$ atacată de o poartă ECL. Linia este fără terminație iar la intrare are o rezistență de 50 Ω conectată în *serie* cu generatorul și o rezistență de 180 Ω la +5 V la ieșirea porții.
9. Găsiți prin metoda grafică formele de undă la capătul de la generator și de la receptor al unei linii, cu $Z_0 = 100\ \Omega$, atacată de o poartă TTL cu colectorul în gol și cu un rezistor de 180 conectat la 5 V.
   (*a*) La capătul îndepărtat.
   (*b*) La capătul apropiat.
10. Desenați forma de undă *reală* a ceasului din figura 12-8 dacă impedanța liniei de masă este de 50 Ω.
11. Dacă bistabilul din figura 12-8 poate comuta cu un impuls de ceas avînd durata de 3ns, cît de lungă poate fi conexiunea la masă realizată pe placa de circuit imprimat?
12. Redesenați figura 12-10 în așa fel încît ambii bistabili să funcționeze corect (păstrați poziția relativă a porții și a bistabilelor).
13. (*a*) Dacă un registru de 16 biți, construit cu bistabile de tip *D*, este comandat pe o linie de ceas cu $Z_0 = 150\ \Omega$, cît de mare trebuie să fie curentul injectat de intrarea fiecărui bistabil pe linia de ceas, pentru a apare o funcționare incorectă a registrului? Considerați că marginea de zgomot este de 1,5 V.
   (*b*) Cît de lungă trebuie să fie linia de ceas pentru a apare disfuncționalități, presupunînd că bistabilii pot fi comandați cu impulsuri de ceas cu durata de 5ns?
   (*c*) Sugerați o soluție pentru rezolvarea problemei.
14. Redesenați figura 12-15 astfel încît să poată fi tolerat un nivel de zgomot pe modul comun de 6 V.
15. Desenați un sistem de interconexiuni similar celui din figura 12-15 cu un emițător și un receptor la fiecare extremitate și în punctul median.
16. Proiectați un receptor serial de date, ce funcționează la 20 MHz, de tipul celui din figura 12-16*b*. Este disponibil un ceas cu cristal de cuarț de 80 MHz. Formatul mesajului este cel prezentat în figura 12-16.
17. Proiectați logica de generare a semnalului din figura 12-16*b* cu ajutorul unui transformator 1 : 1 cu priză mediană conectat la ieșirea a două porți ECL. O linie de 100 Ω trebuie să fie atacată cu impulsuri de 1 V.
18. Desenați schema circuitelor logice ce generează și detectează semnalele reprezentate în figura 12-17 *a*.

REFERINȚA

1. T. Blakeslee, „Transformer Coupled Transceiver Speeds Two-Way Data Transmission", *Electronics*, March 1, 1973, pag. 94—96.

BIBLIOGRAFIE

Balph, Tom, „Use ECL 10 000 Layout Rules, "*Electronic Design*, August 17, 1972, pag. 72—76.

Blakeslee, T., „Transformer Coupled Transceiver Speeds Two-Way Data Transmission", *Electronics*, March 1, 1973, pag. 94—96.

Blood, William, Jr., *MECL System Design Handbook*, Motorola Semiconductor Products, Inc., Mesa, Ariz,, 1972.

Davis, J. H., „T2 : A 6.3 Mb/s Digital Repeatered Line", *Proceedings of IEEE International Conference on Communications*, 69CP369-COM, 1969, pag. 34—9 to 34—16.

De Falco, John, „Reflections and Crosstalk in Logic Circuit Interconnections", *IEEE Spectrum*, July 1970.

Dreher, Thad, „Cabling Fast Pulses ? Don't Trip on the Steps", *The Electric Engineer*, August 1969, pag. 71—75 (transmisie prin cablu coaxial).

Epand, Donald and Liddane, Ken, „Selecting Capacitors Properly", *Electronic Design* 13, June 21 1977, pag. 66—71.

Fairchild Staff, *The TTL Applications Handbook*, Fairchild Semiconductor, Mountain View, Calif., 1973, Capitolul 14, „Transmission Line Interface Elements".

Gray, Harry J., *Digital Computer Engineering*, Prentice-Hall, Englewood Cliffs, N. J., 1963, Capitolul 9, „Pulse and DC Power Transmission Systems".

Marshall, Joseph, „Flat Cable Aids Transfer of Data," *Electronics* July 5, 1973, pag. 89—92.

Sear, Brian E., „Packaging for High Frequency," *The Electronic Engineer*", July 1972, pag. 20—23.

Texas Instruments Staff, *Designing with TTL Integrated Circuits*, McGraw-Hill, New York, 1971, pag. 25—45 and 83—105.

Texas Instruments Staff, *Series 54S/74S Schottky TTL Applications*, Buletin CA-176, Texas Instruments, Dallas, Tex., 1973.

Torrero, Edward, „Focus on Fast Logic", *Electronic Design*, June 8, 1972, pag. 50—57 (o trecere în revistă a circuitelor ECL și Schottky TTL).

Trompeter, Ed, „Cleaning Up Signals with Coax", *Electronic Products*, July 16, 1973, pag. 67—69 (o bună discuție a zgomotului pe masă pentru cablurile coaxiale).

13

DISPOZITIVE DE INTRARE-IEȘIRE

13.1. INTRODUCERE

Ca urmare a evoluției spectaculoase a tehnologiei circuitelor integrate, atît din punctul de vedere al performanțelor cît și din cel al costului, *prețul de cost și rata de defectare în sistemele digitale sînt practic determinate de dispozitivele mecanice de intrare-ieșire.* Deoarece un sistem complet este interfațat la intrare și ieșire cu o serie de dispozitive *mecanice* (afirmația este făcută în sens larg, deoarece apar și sisteme termice, optice etc.) nu există nici o cale de a evita aceste tipuri de dispozitive în proiectarea și realizarea sistemelor de care ne ocupăm. Eficiența dată de utilizarea componentelor electronice integrate face ca acestea să se miște din ce în ce mai mult către „exteriorul" sistemelor, înlocuind pe cît posibil componentele mecanice. *Structurile mecanice se simplifică pe măsura creșterii complexității structurilor electronice.*

În acest capitol nu se va face o trecere completă în revistă a dispozitivelor de intrare-ieșire. Scopul său este de a prezenta cîteva tehnici, din categoria celor mai eficiente, ce pot constitui exemple semnificative de simplificare a structurilor mecanice ca rezultat al interacțiunii cu sistemele electronice. Cu toate că majoritatea exemplelor sînt din categoria echipamentelor periferice standard folosite în domeniul calculatoarelor numerice, principiile pe care ele le ilustrează sînt valabile și în cazul altor tipuri de aplicații. Ne vom concentra atenția asupra a trei tehnici de bază care s-au dovedit a fi viabile din punctul de vedere al simplificării dispozitivelor de intrare-ieșire, (1) reacția negativă, (2) operarea incrementală, (3) partajarea în timp.

13.2. SISTEM AUTOMAT DE REGLARE A POZIȚIEI

Reacția negativă (vezi fig. 13-1) face posibilă obținerea unei precizii mari și/sau a unui răspuns rapid din partea unor componente mecanice de performanțe relativ modeste. Prin sesizarea *poziției instantanee* este generat un *semnal de eroare* care este folosit la comanda unui motor electric cu performanțe modeste. În primul moment motorul va fi acționat foarte puternic, obținîndu-se astfel un răspuns rapid. Pe măsură ce motorul se apropie de poziția finală semnalul de eroare devine din ce în ce mai mic, tinzînd către zero. Dacă motorul depășește, datorită comenzii inițiale puternice, poziția dorită, apare un semnal de eroare cu polaritatea inversă, care comandă mișcarea motorului înapoi către poziția corectă. Deoarece senzorul detectează de fapt *poziția dispozitivului care trebuie poziționat* și nu cea a axului motorului, *toleranțele sistemului de transmisie nu determină nici o eroare.* Frecările, inerția și

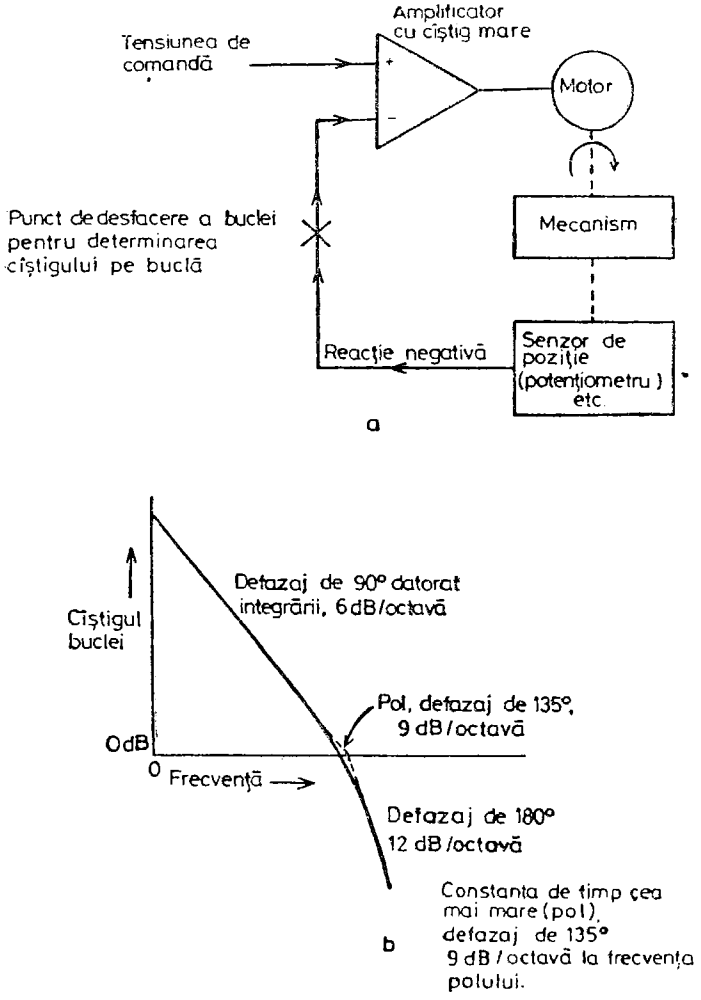


Fig. 13-1. Sistem automat de reglare a poziției.

alte forțe exterioare afectează în mod neglijabil funcționarea sistemului, deoarece semnalul de eroare va comanda motorul exact cu atît cu cît este necesar ca să se obțină deplasarea care trebuie. Cu cît cîștigul amplificatorului este mai mare cu atît funcționarea sistemului este mai apropiată de ideal deoarece un cîștig mare înseamnă că se poate obține un semnal de comandă mai puternic, pornindu-se de la un semnal de eroare mai mic. Creșterea valorii cîștigului amplificatorului este totuși limitată, deoarece sistemul poate intra în *oscilație*. Acest fenomen va apare datorită faptului că motorul rotindu-se spre poziția corectă poate să o depășească. Încercînd să readucem motorul spre poziția corectă, aceasta va fi iarăși ușor depășită în celălalt sens, și așa mai departe. Sistemul este stabil dacă se realizează condiția ca valoarea *cîștigului pe buclă* să fie mai *mică* decît *unitatea, la frecvența la care defazajul pe buclă este de 180°* (deoarece la această frecvență reacția negativă se transformă într-o reacție pozitivă). *Cîștigul pe buclă* se calculează mergînd în jurul buclei (vezi fig. 13-1) și înmulțind între ele cîștigul amplificatorului,

al motorului, al mecanismului, al senzorului şi al reţelei de sumare. De asemenea se pun în evidenţă şi o serie de constante de timp electrice şi mecanice, denumite şi *poli*. Fiecare constantă de timp (pol) determină o *frîngere a caracteristicii de frecvenţă* la o frecvenţă egală cu 1/constanta de timp. Peste această frecvenţă se obţine un defazaj de aproximativ 90° iar cîştigul începe să scadă, proporţional cu frecvenţa, cu 6 dB/octavă.

Problema stabilităţii este de regulă studiată utilizînd caracteristicile Bode. Deoarece defazajul introdus este corelat cu panta de scădere a caracteristicii cîştig-frecvenţă, stabilitatea va presupune ca în punctul în care cîştigul pe buclă este unitar (zero dB) panta caracteristicii cîştig-frecvenţă să fie mai mică de 12 dB/octavă (defazaj de 180°).

Cîştigul pe buclă al sistemului de reglaj din figura 13-1 scade iniţial cu 6dB/octavă (defazaj de 90°) datorită faptului că poziţia este proporţională cu *integrala* turaţiei motorului, deci şi cu integrala tensiunii de la ieşirea amplificatorului. Datorită acestei scăderi iniţiale cu 6dB/octavă a caracteristicii cîştig pe buclă-frecvenţă, instabilitatea sistemului poate fi dată de o scădere suplimentară de 6 dB/octavă (în total 12 dB/octavă) corespunzătoare celei mai mari constante de timp a părţii mecanice a sistemului.

13.3. SISTEM DE REGLARE AUTOMATĂ A VITEZEI

În figura 13-2 este reprezentat un alt tip de sistem de reglare automată care permite reglarea vitezei. În acest caz este sesizată şi folosită pentru control prin reacţie *viteza* mecanismului. Tensiunea de comandă de la intrare trebuie să specifice de această dată nu poziţia ci viteza. Caracteristica de frecvenţă reprezentată în figura 13-3 arată faptul că instabilitatea într-un astfel de sistem nu poate să apară decît după *cea de a doua constantă de timp* (al doilea pol) deoarece reglajul în sistem nu mai este de tip integral, precum cel din exemplul anterior. Deoarece *cîştigul cu bucla închisă* (vezi fig. 13-3, *b*) începe să scadă odată cu scăderea sub unitate a cîştigului pe buclă, răspunsul în frecvenţă al sistemului se poate extinde mult peste valoarea corespunzătoare celei mai mari constante de timp a părţii mecanice. Va fi deci posibil să comandăm de exemplu, deplasarea unei hîrtii în 17 ms cu un motor ce are o constantă de timp mecanică de 50 ms. În realitate, banda de frecvenţă şi cîştigul pe buclă pentru ambele tipuri de sisteme automate de reglaj a poziţiei şi vitezei pot fi considerabil mărite prin adăugarea în bucla de reacţie unor reţele RC de corecţie (cu efect derivator sau integrator). Aceste reţele adaugă poli şi zerouri suplimentare care pot modifica în mod convenabil caracteristica cîştig pe buclă-frecvenţă.

Un alt avantaj al sistemului de reglare automată a vitezei îl constituie posibilitatea de a controla acceleraţia motorului prin intermediul unor forme de undă convenabile pentru tensiunea de comandă a vitezei sub forma unor rampe lente (vezi fig. 13-4). Deoarece poziţia este dată de integrala vitezei, sistemul de reglare automată a vitezei poate fi folosit pentru *modificarea* cu precizie a *poziţiei*, proporţional cu durata semnalului de comandă a vitezei. Dacă se mai foloseşte şi o buclă de poziţie (de exemplu, cu o celulă fotoelectrică) pentru stoparea comenzii vitezei, se poate obţine o poziţionare precisă, *absolută*, utilizînd sisteme de reglare automată a vitezei.

Informaţia de viteză necesară pentru generarea semnalului de reacţie, se poate obţine fie cu ajutorul unui tahometru de curent continuu (care este de fapt un mic generator de curent electric) fie poate fi obţinută elec-

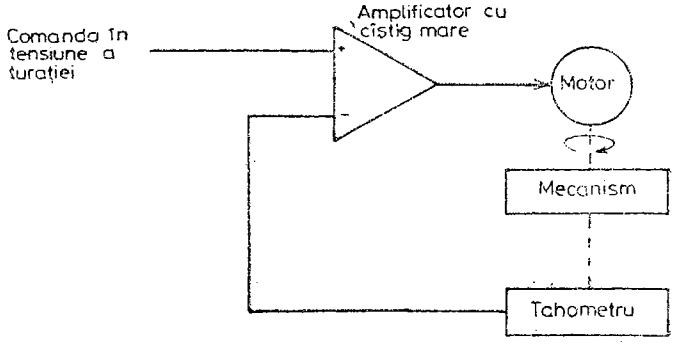


Fig. 13-2. Sistem automat de reglare a vitezei.

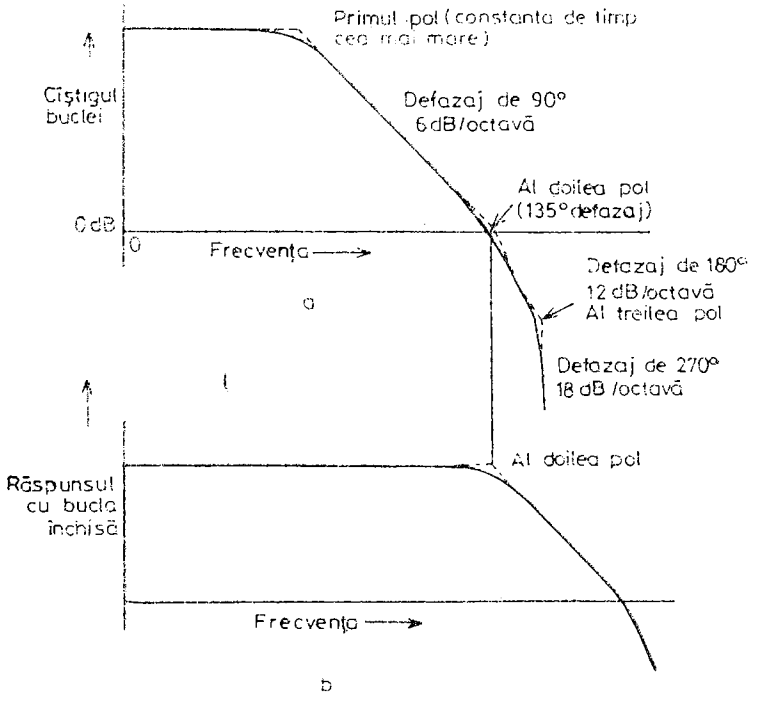


Fig. 13-3. *(a)* Cîștigul buclei. *(b)* Răspunsul cu bucla închisă.

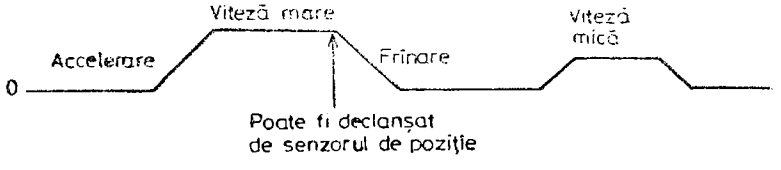


Fig. 13-4. Forma de undă a tensiunii de comandă a vitezei.

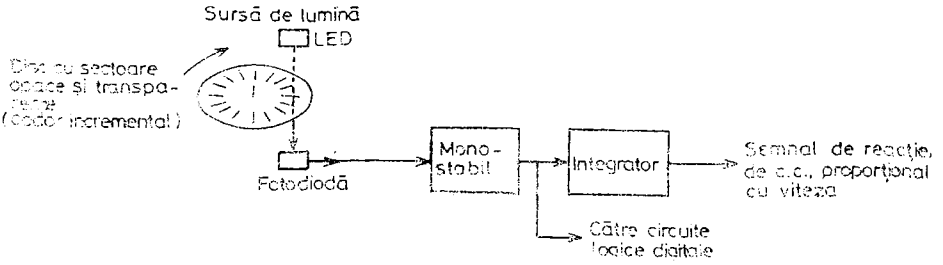


Fig. 13-5. Generarea semnalului de tahometru utilizînd un codor incremental.

tronic dintr-un tren de impulsuri ca în figura 13-5. Un circuit monostabil (de exemplu 74121) produce un impuls de durată și amplitudine bine definite pentru fiecare impuls generat de fotocelulă. Cu cît impulsurile date de fotocelulă sînt mai dese, cu atît este mai mare factorul de umplere al succesiunii de impulsuri de la ieșirea monostabilului, deci cu atît este mai mare componenta continuă la ieșirea integratorului. Liniaritatea și acuratețea acestui tip de circuit sînt excelente (în plus se elimină și paraziții dați de periile generatoarelor de c.c.).

Dacă nu se cer performanțe deosebite se poate folosi ca semnal tahometric *tensiunea contraelectromotoare a motorului*. Motorul cu magnet permanent generează o tensiune direct proporțională cu viteza de rotație. Această tensiune diferă față de cea de la terminalele motorului prin căderea de tensiune pe rezistența armăturii. Măsurînd curentul prin motor se poate determina căderea de tensiune pe această rezistență. Deci semnalul proporțional cu viteza de rotație se obține scăzînd din tensiunea de la bornele motorului căderea de tensiune pe rezistența armăturii (care este dată de produsul dintre curent și rezistența armăturii). Circuitul din figura 13-6 comandă motorul la turația impusă de tensiunea de intrare, independent de sarcină. Uneori și motoarele de curent alternativ sînt reglate prin compararea tensiunii contraelectromotoare cu tensiunea de comandă la începutul fiecărei semialternanțe. Dacă este sesizată o scădere a vitezei, în respectiva semialternanță motorul se alimentează folosind tiristoare sau triacuri.

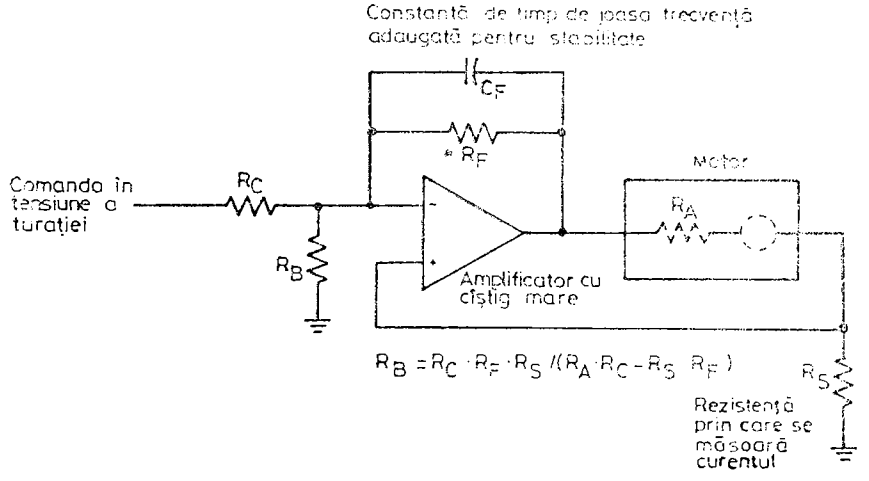


Fig. 13-6. Utilizarea motorului ca tahometru.

13.4. CONTROLUL NUMERIC

Cu toate că sistemele cu reacție discutate anterior lucrează prin sumări de mărimi analogice, aceleași principii pot fi aplicate unor sisteme digitale. Realizarea analogică a unor operații cu ajutorul *amplificatoarelor operaționale* este justificată de faptul că ele sînt niște circuite integrate foarte ieftine, rapide și precise. Scăderea tensiunii de reacție din cea de comandă necesită (vezi figura 13-1 și 13-3) în mod normal numai două rezistențe conectate convenabil la intrarea unui amplificator operațional.

În cazurile în care nu este necesar un răspuns foarte rapid, reacția negativă poate fi introdusă prin intermediul unui sistem digital. Spre exemplu, impreciziile care au o evoluție lentă în timp pot fi măsurate și corectate prin operații digitale asupra parametrilor prelucrați. Dacă dorim să obținem o precizie foarte bună în măsurarea unor tensiuni, spre exemplu, utilizînd circuite analogice ieftine, vom măsura periodic deriva sistemului cu semnal nul pe intrare. *Rezultatul obținut pentru măsurarea tensiunii cu zero pe intrare este scăzut din măsurătorile curente efectuate.* Dacă se fac periodic măsurători de calibrare poate fi calculat și un *factor de scală* pe baza căruia să se realizeze corecții suplimentare. Toate aceste calcule numerice pot fi realizate cu un sistem cu *logică programată.* Tot sistemul de măsurare reprezintă un compromis direct între cît mai multe instrucțiuni în program și un hardware exterior cît mai simplu. Se poate merge mai departe și obține compensarea prin program a *neliniarităților traductorului,* utilizînd pentru caracteristicile de transfer fie expresii analitice, fie tabele.

Și precizia sistemelor mecanice poate fi mult sporită prin corecții programabile. În multe cazuri un simplu microcontact sau o fotocelulă pot oferi o referință de precizie în operația de poziționare a unui obiect. Prin program se poate realiza corecția tuturor ieșirilor sistemului, pornind de la aceste referințe. Se pot astfel compensa efecte de uzură sau de dilatare termică, în mod automat, la intervale de cîteva secunde.

Folosind procedee de recalibrare și liniarizare digitală, pot fi utilizate în aplicații de precizie traductoare cu parametri modești. Intrările digitale pot fi obținute direct prin construirea unui oscilator ai cărui parametri să varieze în funcție de mărimea măsurată (de exemplu presiune, temperatură, poziție). Starea numărătorului comandat de acest oscilator poate fi folosită la intervale de timp regulate pentru a determina prin calcul parametrul dorit. Acest tip de semnal conține valoarea măsurată tradusă în *frecvența impulsurilor.* Preprocesarea acestui tip de semnal este foarte simplă dacă se folosesc circuite integrate specializate (spre exemplu trei multiplicatoare de șase biți care permit multiplicarea numerelor de 18 biți) pentru realizarea multiplicării. În esență, numărarea reprezintă un proces de integrare prin acumulare de impulsuri. Citirea periodică a stării numărătorului reprezintă o divizare ce oferă drept rezultat o *valoare medie.*

În multe aplicații se pot obține cu o precizie foarte bună semnale codate în frecvență. Spre exemplu, un simplu *oscilator controlat în tensiune (voltage-controlled oscilator — VCO)* poate genera o frecvență ce depinde liniar de tensiunea aplicată cu o precizie de 0,05%.

Pentru aplicațiile uzuale cristalele de cuarț sînt tăiate în așa fel încît frecvența de oscilație să depindă cît mai puțin de temperatură. Se pot obține,

însă, cristale de cuarț astfel tăiate încît frecvența de oscilație să depindă liniar de temperatură cu precizie de 0,1% între 0°C și 100°C. Cu oscilatoare construite utilizînd astfel de cuarțuri pot fi detectate diferențe de temperatură de 0,0001°C în domeniul —80°C...+250°C.

13.5. FUNCȚIONAREA INCREMENTALĂ

Un alt procedeu modern pentru controlul precis al deplasărilor mecanice este constituit de *funcționarea incrementală*. Orice mișcare poate fi obținută printr-o *succesiune de deplasări foarte mici*. Deoarece numărul de pași este o mărime digitală, interfața de control se pretează la o abordare ce presupune circuite integrate digitale. Utilizarea unui *codor incremental* (vezi fig. 13-5) într-un sistem automat de reglaj al vitezei permite obținerea unui *sistem de reglaj incremental al mișcării*. Tensiunea de comandă a vitezei determină accelerarea treptată a mecanismului, menținîndu-l apoi la o viteză constantă pînă ce de la codorul incremental rezultă un număr dat de impulsuri (de multe ori numai unul singur). În acest punct începe procesul de *frînare* care determină o oprire lină.

Pentru obținerea unei bucle de poziționare *absolute* poate fi utilizat un codor incremental ce acționează asupra unui *numărător digital reversibil* (vezi fig. 13-7). Fiecare stare a numărătorului va reprezenta în acest fel o poziție mecanică discretă. O comandă digitală de trecere într-o nouă poziție determină un semnal de *viteză* în direcția necesară (determinată printr-o comparație digitală) pînă ce starea numărătorului corespunde cu poziția dorită.

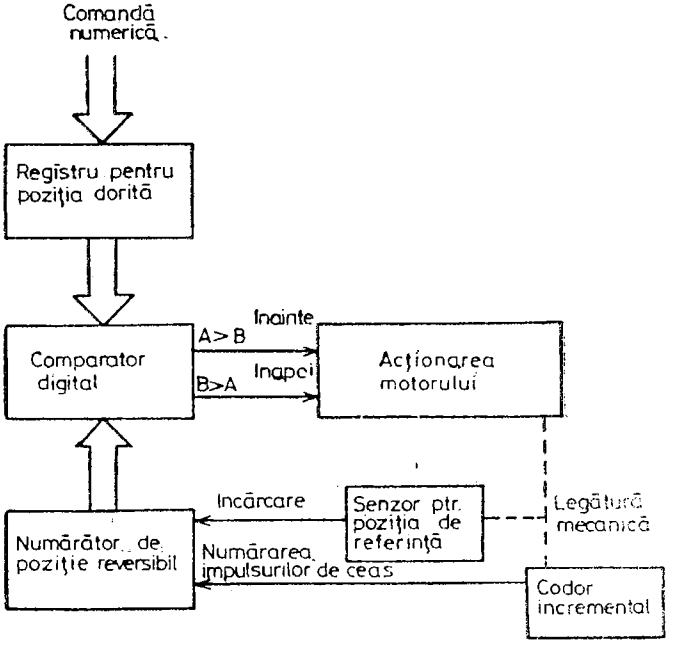


Fig. 13-7. Sistem de poziționare numeric ce utilizează un sistem automat de reglaj al vitezei.

Atunci cînd sistemul este inițializat, la conectarea sursei, numărătorul trebuie poziționat cu valoarea *absolută* corectă. Acest fapt este realizat, uzual, printr-o *secvență de inițializare* ce găsește referința. Un microcomutator sau un indicator optic și un senzor montat pe mecanism indică atingerea poziției de referință, determinînd și poziționarea numărătorului în starea corespunzătoare. Ca și în cazul *autoinițializării* (vezi Secțiunea 6.6) numărătorul va fi, de asemenea, setat în această stare dacă pe parcursul operării normale este detectată poziția de referință.

Un sistem similar se poate construi folosind, în locul unui dispozitiv de reglare a vitezei, un *motor pas cu pas*. Motorul pas cu pas prezintă marele avantaj de fiabilitate determinat de *absența periilor colectoare*. De asemenea, motorul pas cu pas se deplasează *exact cu un increment* pentru fiecare tranziție a semnalului de comandă, astfel încît nu mai este necesară introducerea unei reacții negative de control. În figura 13-8 este reprezentat un sistem tipic ce folosește un motor pas cu pas. Menționăm că deși nu mai este necesar un codor incremental pentru bucla de reacție, este în continuare necesar un senzor pentru poziția de referință în vederea realizării calibrării absolute. Motorul pas cu pas este reprezentat ca fiind conectat la numărătorul de poziție deoarece semnalul de comandă necesar este digital, de forma celui generat de un

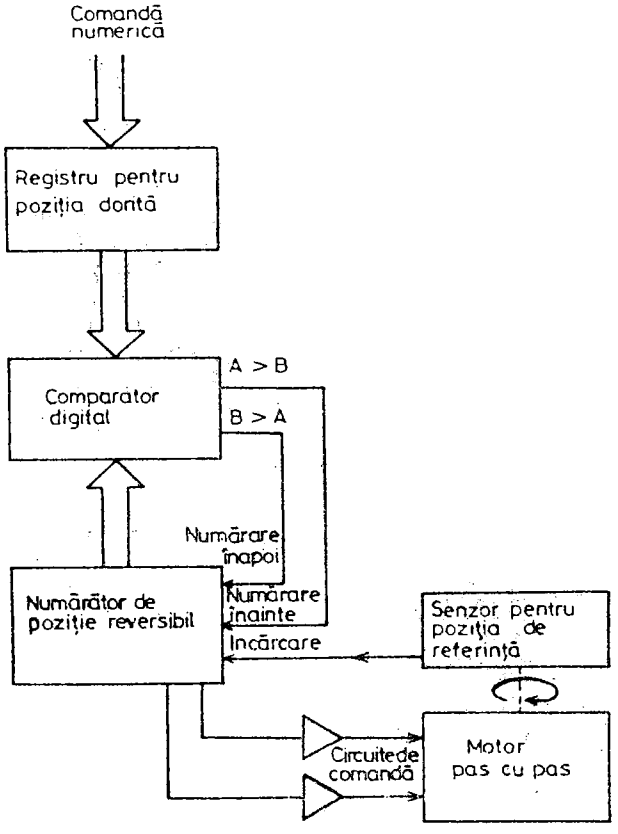


Fig. 13-8. Sistem de poziționare numeric ce utilizează un motor pas cu pas.

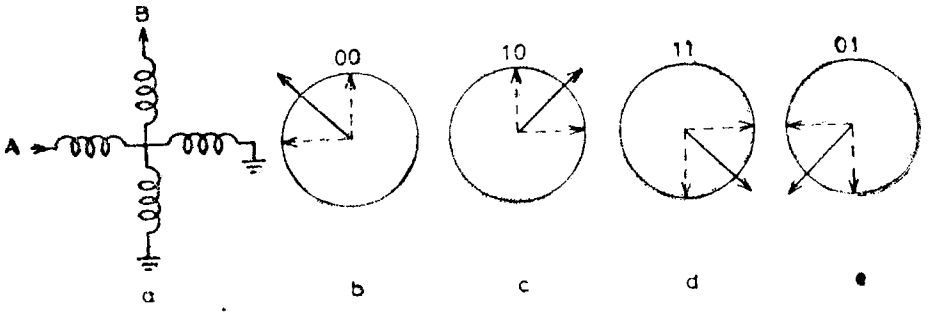


Fig. 13-9. Comanda unui motor pas cu pas de la un numărător Moebius : *(a)* săgeata indică direcţia forţei pentru „1" ; *(b)* primul pas ; *(c)* al doilea pas ; *(d)* al treilea pas ; *(e)* al patrulea pas.

numărător *Moebius* cu patru stări (vezi figura 5-7, *e*). În consecinţă există posibilitatea generării semnalului de comandă utilizînd direct cei mai puţin semnificativi doi biţi ai numărătorului de poziţie.

Rotirea vectorului forţă corespunzător secvenţei de numărare Moebius este reprezentată în figura 13-9. Parcurgerea secvenţei în ordine inversă determină rotirea în sens contrar. Circuitele de comandă acţionează asupra înfăşurărilor motorului cu polarităţile (+) şi (—) pentru 1 şi 0 aplicaţi la intrare. Există motoare cu înfăşurări bifilare (cu priză mediană) ce au două înfăşurări pentru fiecare fază. Acest fapt permite utilizarea a patru circuite integrate de comandă, cu o singură polaritate, la comanda cîte unuia din cele două capete opuse ale celor două înfăşurări, asociate fiecărei faze. În situaţiile curente un pas este de 1/100 sau 1/200 dintr-o rotaţie completă, deoarece motorul este construit cu 100 sau cu 200 de poli. În consecinţă poziţionarea poate fi foarte precisă ; în plus viteza poate atinge 1 000 paşi pe secundă (600 rotaţii pe minut). Se poate obţine o deplasare liniară de foarte mare precizie montînd pe axul motorului pas cu pas un şurub fără sfîrşit (vezi figura 13-10). Deoarece eroarea *nu* este cumulativă se poate realiza o deplasare de mii de paşi cu o eroare dată de deplasarea cu unul singur (1/10 000 = 0,001% precizie). Mărimea unui increment fiind cuantificată digital, folosind memorii digitale şi circuite logice adecvate se pot realiza interpolări convenabile, se pot trasa curbe ş.a.m.d. De fapt, numărătorul de poziţie, registrul, şi comparatorul reprezentate în figurile 13-7 şi 13-8 pot fi simulate într-un sistem cu *logică programată* (pot fi o parte a unui *program* de microprocesor).

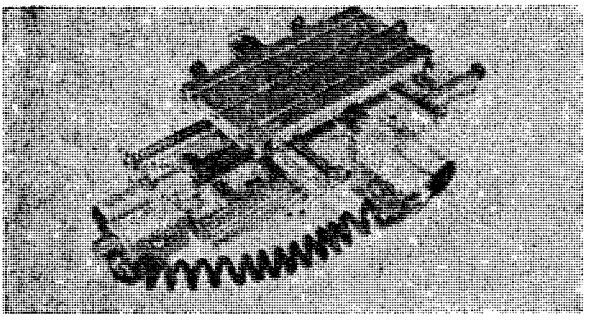


Fig. 13-10. Poziţionare liniară cu şurub fără sfîrşit.

Codificatoarele incrementale simple generează aceleaşi impulsuri pentru fiecare sens de deplasare. Din această cauză se poate întîmpla ca direcţia curentă de deplasare să nu fie corelată cu polaritatea semnalului de comandă. Rezolvarea acestei probleme implică utilizarea unui *codificator incremental* cu *două ieşiri* generate de doi senzori aşezaţi astfel încît să dea

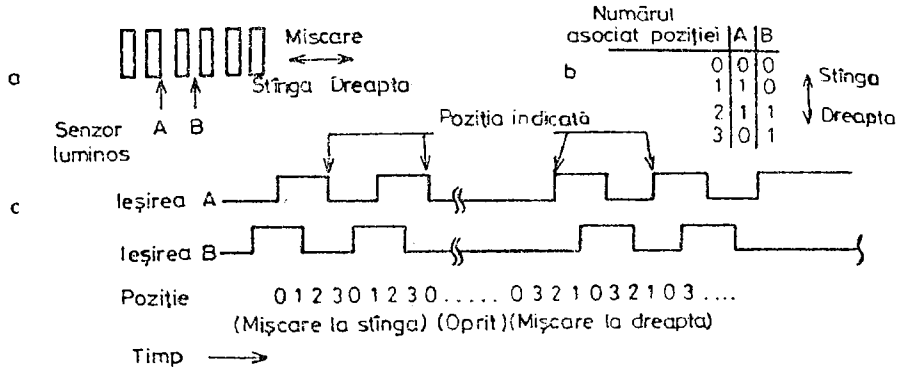


Fig. 13-11. Codor incremental cu sesizarea direcţiei de mişcare *(a)* pistă de codare; *(b)* stările la ieşirea senzorului; *(c)* ieşirile pentru cazul în care se schimbă sensul mişcării.

semnale ce sînt defazate cu 90° (vezi fig. 13-11 *a*). Dacă pista marcată se deplasează la dreapta, ieşirile *A* şi *B* generează succesiunea directă a unui numărător de tip Moebius, după cum este reprezentat în figura 13-11 *b*. Deplasarea în sens invers determină generarea secvenţei inversate. În consecinţă, se poate determina prin *logică* direcţia de mişcare. În figura 13-11 *c* sînt prezentate formele de undă asociate mişcării la stînga, staţionării şi deplasării către dreapta. Un principiu similar este folosit pentru determinarea rotaţiei în sensul acelor de ceasornic sau invers, cu ajutorul codorului incremental rotativ.

După ce am trecut în revistă cîteva din principiile de bază utilizate în echipamentele de intrare-ieşire (I/O) moderne, vom prezenta cîteva exemple de utilizare.

13.6. UNITĂŢI DE BANDĂ MAGNETICĂ DE 1600 BIŢI/INCH

Unităţile de bandă magnetică constituie un exemplu foarte bun de *partajare a timpului („time sharing")* deoarece milioane sau miliarde de biţi pot fi citiţi folosind un singur cap cu nouă piste şi circuite adiţionale de scriere-citire. Acest lucru duce la obţinerea unui preţ foarte scăzut, raportat la bit, de stocare a informaţiei în detrimentul timpului de acces (uneori sînt necesare minute pentru a găsi o anumită informaţie înregistrată).

La început unităţile de bandă foloseau role presoare (comandate de electromagneţi) pentru a presa banda pe un *cabestan* (un ax de oţel cu diametrul de aproximativ 6 mm) care se rotea fără oprire, cu viteză constantă. Pentru cele două sensuri de deplasare ale benzii erau necesare două cabestane şi două role presoare, cîte unul, respectiv, una pentru fiecare sens. Atunci cînd rola presoare presează banda pe cabestan banda este accelerată rapid pînă la viteza cabestanului. Presarea benzii şi accelerarea sa necontrolată determinau de multe ori deteriorarea benzii.

Unităţile de bandă magnetică moderne, de 1 600 bit pe inch (BPI), folosesc un *singur cabestan* pentru a conduce banda. Acest cabestan, cu un

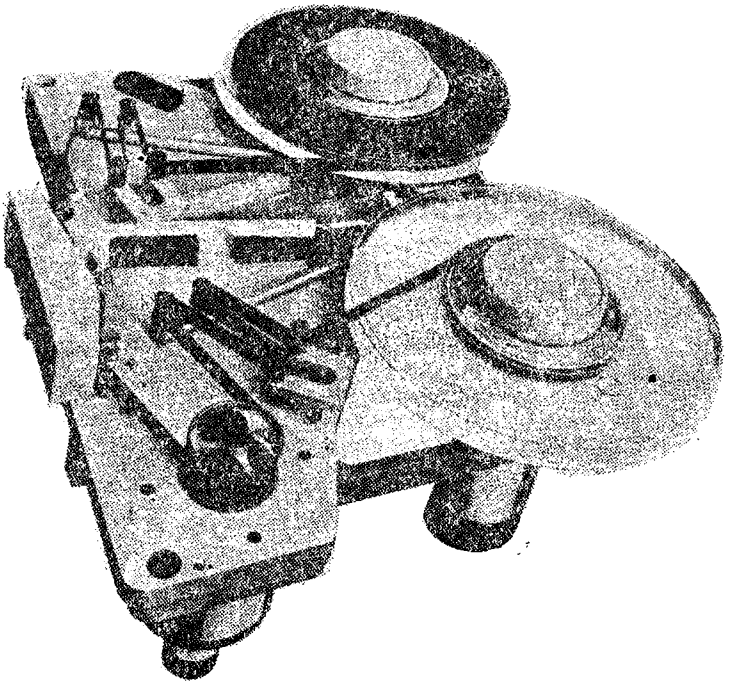


Fig. 13-12. Unitate de bandă cu un singur cabestan (Precision Instruments).

diametru de aproximativ 5 cm, se află pe axul unui servomotor cu inductanță scăzută (vezi fig. 13-12).

Pe axul servomotorului se află un tahometru care dă semnalul de reacție negativă pentru *sistemul de reglare automată a vitezei.* Banda poate fi astfel accelerată treptat pentru obținerea valorii dorite a vitezei, utilizînd o mărime analogică de comandă (vezi fig. 13-4). Printr-o tensiune negativă de comandă se obține deplasarea treptată a benzii în sens invers, folosind același cabestan.

Cele două role cu bandă sînt montate pe axele a două motoare de c.c. Fiecare motor de antrenare a rolei primește un semnal de reacție negativă de la un senzor cu fotocelulă pentru a menține o buclă a benzii, de dimensiune convenabilă, între rolă și cabestan. În acest fel în momentul în care banda începe să se deplaseze, se accelerează numai o porțiune mică a benzii și nu rola întreagă. Cînd bucla benzii devine prea largă pe partea pe care se înfășoară fotocelula va trimite semnalul de comandă spre motorul rolei care se înfășoară pentru realizarea corecției. În cealaltă parte, bucla se strînge comandînd motorul rolei care se desfășoară să aducă bandă. Există deci trei sisteme de reglare automată separate, care lucrează toate ca rezultat al comenzii vitezei motorului ce antrenează cabestanul. La unele unități de bandă, senzorii buclei benzii pornesc sau opresc motoarele pentru comanda rolelor. Acest tip de reglaj automat este numit adesea „reglaj bang-bang".

La unitățile de bandă cu performanțe ridicate se folosește uneori pentru decodarea datelor, un al patrulea sistem de control. Acesta nu presupune o parte mecanică distinctă, ocupîndu-se în mod electronic de variațiile vitezei benzii, care scapă sistemului de control al cabestanului.

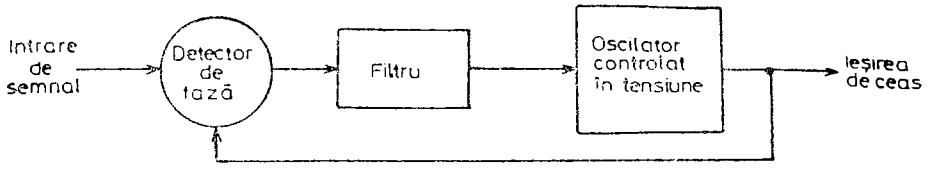


Fig. 13-13. Buclă cu calare de fază.

Cînd datele sînt citite de pe bandă este generat un semnal de ceas (intern), similar cu cel din figura 12-17 *a*. Dacă pentru decodarea acestui semnal am folosi un circuit cu temporizare fixă, abaterile de viteză ar duce la creşterea numărului de erori. Din acest motiv se utilizează o *buclă cu calare de fază (phase-locked loop — PLL)* pentru a genera un semnal de ceas în fază cu *media* semnalului de intrare. O buclă cu calare de fază (PLL) este un sistem de reglaj automat care foloseşte reacţia negativă pentru a controla, nu o poziţie sau o viteză, ci *faza unui semnal*. Un detector de fază (vezi fig. 13-13) compară faza semnalului de la intrare cu cea a semnalului de la ieşirea *oscilatorului controlat în tensiune (voltage-controlled oscillator — VCO)*. Tensiunea continuă de comandă a VCO este proporţională cu media erorii de fază a celor două semnale, astfel că, frecvenţa creşte atunci cînd faza semnalului de la oscilator este în urma fazei semnalului de intrare. Astfel, semnalul de ceas obţinut urmează îndeaproape faza semnalului recepţionat. Pentru implementarea unui astfel de sistem sînt disponibile circuite integrate care îndeplinesc această funcţie prin realizarea unei bucle cu calare de fază (de exemplu NE562).

Deoarece pentru sincronizarea (calarea) oscilatorului cu semnalul citit este necesară o anumită durată de timp, un *preambul* de 40 de 0 şi un 1 precedă sau urmează (în cazul citirii inverse) fiecare înregistrare.

O altă problemă mecanică din tradiţionalele unităţi de bandă şi care este rezolvată electronic în unităţile de bandă moderne (1600-BPI) este *alunecarea ceasului (skew)*. Tipurile mai vechi utilizau codarea **fără întoarcere la zero (NRZ)**, în care toate pistele foloseau acelaşi semnal de ceas. Ca urmare, aceste benzi aveau o lungime mare pentru a elimina variaţia în timp între semnalele citite de pe o pistă faţă de alta. Deşi toate pistele erau scrise sincron, variaţiile relative în timp se datorau întinderii inegale, capului şi variaţiilor de ghidare. Cu toate că unităţile de bandă anterioare încercau să sincronizeze toate pistele în acelaşi timp (sau cu mici corecţii fixe), unităţile de bandă 1600-BPI utilizează pe fiecare canal un semnal de *autosincronizare*.

Acest lucru face posibilă utilizarea unui „buffer" digital cu lungime variabilă pe fiecare canal. Ori de cîte ori toate cele nouă canale au un nou bit validat („ready"), la ieşire este transmis un caracter. Totuşi, fiecare canal poate transmite unul sau doi biţi înaintea celorlalţi, dacă este necesar.

13.7. UNITĂŢI DE CASETĂ MAGNETICĂ

Unităţile de casetă magnetică încearcă să ne furnizeze o unitate de bandă magnetică cu un cost minim, care se poate cumpăra la preţul unei unităţi de bandă de hîrtie.

Prin citirea şi scrierea *unei piste* o dată (adesea comutabil) se obţine un grad şi mai mare de partajare a timpului (time sharing), odată cu accentuarea dezavantajului de viteză.

Mediul de stocare utilizat este o casetă mică de bandă, care, la început a fost proiectată pentru a fi folosită la viteza de 4,7 cm/s*, pentru domeniul audio. Multe încercări au fost făcute în anii 1971—1972 pentru a obține o casetă fiabilă. Cabestanul minuscul și rola presoare folosite la viteza de 4,7 cm/s, erau nepotrivite pentru viteze mai mari și porniri și opriri brusce.

Casetele mai noi nu mai utilizează cabestanul ci, mai simplu, se antrenează direct de la axul motoarelor. Această soluție este simplă și fiabilă dar întrucît diametrul rolei variază de la 2 cm cînd este goală, la 5 cm cînd este plină, viteza benzii variază într-un raport de 2,5 de la un capăt la altul al benzii. Se vor lua în discuție două modalități de abordare electronică a acestei probleme.

Primul mod de abordare, lasă să varieze viteza benzii și utilizează un format de înregistrare, care permite autosincronizarea fiind insensibil la o variație de patru ori a vitezei. Figura 13-14 prezintă formatul de înregistrare, care constă dintr-o tranziție pozitivă a semnalului de ceas la începutul intervalului asociat fiecărui bit, și o tranziție după prima treime a intervalului dacă se transmite 1, sau după a doua treime a intervalului dacă se transmite 0. Decodarea este realizată digital, cu un numărător reversibil, de șase biți.

La începutul intervalului de bit (tranziția pozitivă) numărătorul începe să numere înainte; în momentul tranziției negative, numărătorul începe să numere înapoi. Dacă numărătorul ajunge la zero înaintea următoarei tranziții pozitive, este detectat 1 (numărătorul începe să numere invers *înainte* de jumătatea intervalului de bit) ; în situația contrară, este detectat 0.

Deși acest procedeu citește datele cu acuratețe, în ciuda variației vitezei, este ineficient din două motive. În primul rînd, densitatea „biților" variază cu viteza benzii, astfel că se obține o densitate mare pentru viteză mică și o densitate mică pentru viteză mare. În al doilea rînd, procedeul de codare din figura 13-14, înregistrează un bit în domeniul a 3 tranziții, față de două pentru codarea în fază reprezentată în figura 12-17, *a*.

Prin menținerea unei viteze constante și utilizarea codării în fază, poate fi înmagazinată pe bandă o cantitate de date de peste *două ori* mai mare. Viteza constantă poate fi menținută, în ciuda variației diametrului rolei, prin *reacție negativă*. O metodă de obținere a unui semnal de viteză a benzii constă în preînregistrarea pe bandă a unei *piste speciale de sincronizare (clock track)*.

De pe această pistă este generată o tensiune de tahometru, care este folosită ca semnal de reacție negativă. Această tehnică este totuși ineficientă, întrucît pista de sincronizare, amplificatorul de citire ș.a.m.d., constituie capacități de stocare irosite. În plus, sînt necesare casete speciale cu pistă de sincronizare înregistrată.

Viteza benzii poate fi menținut cu o abatere de $\pm 5\%$ fără pista de sincronizare, prin folosirea motoarelor, care comandă cele două role cu bandă, ca tahometre. Tensiunea generată de motorul *tras* de bandă („*drag motor*" —

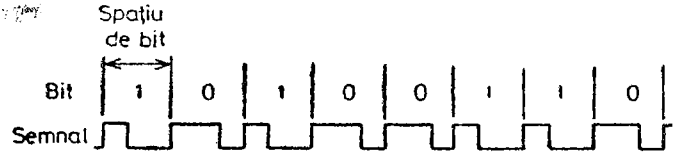


Fig. 13-14. Codarea în durată a impulsurilor.

* 1 7/8 inch/s (n.t.).

motorul antrenat) este proporțională cu viteza sa de rotație S_2. Tensiunea proporțională cu viteza motorului de *antrenare* (S_1) poate fi obținută scăzînd din tensiunea de la borne, căderea de tensiune pe rezistența armăturii (măsurată prin intermediul unei rezistențe, senzor de curent, ca în figura 13.6). Vitezele celor două axe motoare pot fi exprimate în funcție de viteza benzii și de diametrele rolelor, astfel:

$$V = \frac{S_1 D_1}{2} = \frac{S_2 D_2}{2} \tag{13-1}$$

Diametrele rolelor sînt legate de diametrul butucului d, lungimea și grosimea benzii, L și respectiv t, prin relația

$$\frac{\pi}{4}(D_1^2 - d^2) + \frac{\pi}{4}(D_2^2 - d^2) = Lt \tag{13-2}$$

sau

$$D_1^2 + D_2^2 = \frac{4Lt}{\pi} + 2d^2 = \frac{K^2}{4} \tag{13-3}$$

unde K este o constantă pentru un diametru al butucului, pentru o lungime și o grosime a benzii date.

Din relațiile 13-1 și 13-3 rezultă viteza:

$$V = \frac{KS_1S_2}{(S_1^2 + S_2^2)^{1/2}} \tag{13-4}$$

Această funcție poate fi aproximată simplu cu un generator de funcții, cu diode, prin două segmente de linie dreaptă. Mecanismul de reglare al vitezei obținut astfel (vezi fig. 13-15), controlează viteza benzii cu abatere de ±5% la 50 cm/s și atinge viteza stabilizată în 35 ms. Acesta este un exemplu foarte bun al modului în care se pot utiliza electronic diferite semnale ce par a nu fi potrivite la un preț de cost mult mai redus decît cel care ar rezulta în cazul utilizării mai multor componente electromecanice. Putem spune, într-un mod mai simplu, că ne adaptăm la ceea ce este accesibil — într-un proces similar cu proiectarea logică adaptată la aparent, nepotrivitele componente ieftine.

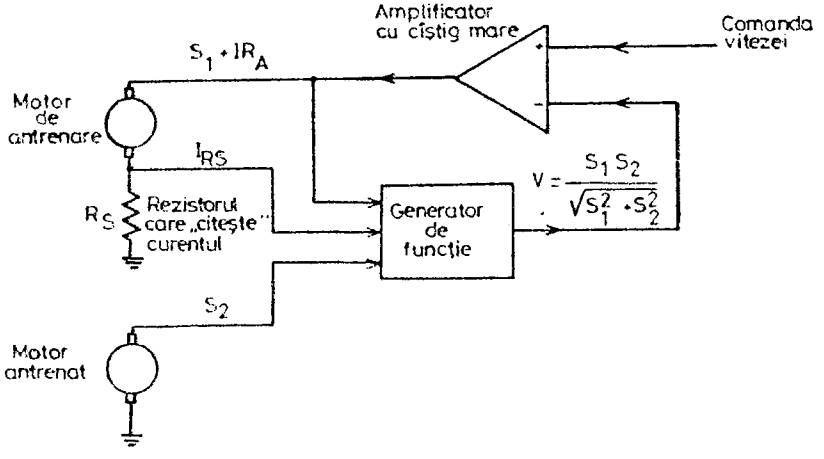


Fig. 13-15. Sistem de reglare automată a vitezei de antrenare a benzii magnetice.

13.8. UNITĂȚI DE DISC FLEXIBIL

Alt dispozitiv periferic folosit atît pentru memorie auxiliară cît și ca un dispozitiv de intrare-ieșire este *unitatea de disc*. Unitățile de disc mari (de exemplu IBM3331) au pachete de discuri amovibile, pe care se pot stoca, *miliarde* de biți la un preț de cost mai mic decît 1/200 cenți pe bit.

Unitatea de disc flexibil (vezi fig. 13-16) este un dispozitiv cu un preț de cost minim, ce poate stoca 3 milioane de biți pe o bucată de material magnetic de dimensiunea unui disc de patefon de 45 ture/min. Discul este ținut întotdeauna pentru protecție, într-o anvelopă de plastic împreună cu care este introdus și în mecanismul de antrenare. *Un singur cap* este deplasat radial pe suprafața discului de un *motor pas-cu-pas*, pentru a citi oricare din cele 77 piste de date. O fotocelulă indică poziția capului pe pista zero, astfel că la conectarea sursei de alimentare, prin secvența de inițializare se face deplasarea capului către pista zero, pînă cînd fotocelula indică poziția corectă a capului. Prin sistemul de comandă se generează numărul corect de semnale pentru deplasarea pas-cu-pas de pe o pistă pe oricare alta, folosind în acest scop o implementare hardware (sau software) ca aceea din figura 13-8.

Discul flexibil tip „Dynastore" elimină necesitatea unei precizii ridicate a dispozitivului de poziționare a capului, prin folosirea pentru poziționare a *semnalelor de reacție negativă de pe pistele înregistrate*.

Discul este preînregistrat de firmă cu marcaje de sector pentru fiecare pistă, ca în figura 13-17 ; fiecare pistă este împărțită în 32 sectoare. Fiecare sector începe cu o informație de poziționare preînregistrată de firmă (SERVO 1 și SERVO 2) și adresa de pistă și sector. Toate datele sînt înregistrate în spațiul *dintre* marcajele de sector.

Informația preînregistrată SERVO 1 și SERVO 2 este înregistrată după un algoritm tablă de șah după cum se arată în figura 13-17. (Zonele haşurate au semnal înregistrat, în timp ce zonele albe nu au informație înregistrată). Dacă, capul este centrat pe pistă, amplitudinea semnalului în timpul zonei SERVO 1 și 2 va fi egală deoarece fiecare este citit de o jumătate din cap. În situația în care capul nu este centrat pe mijlocul pistei ci este deplasat

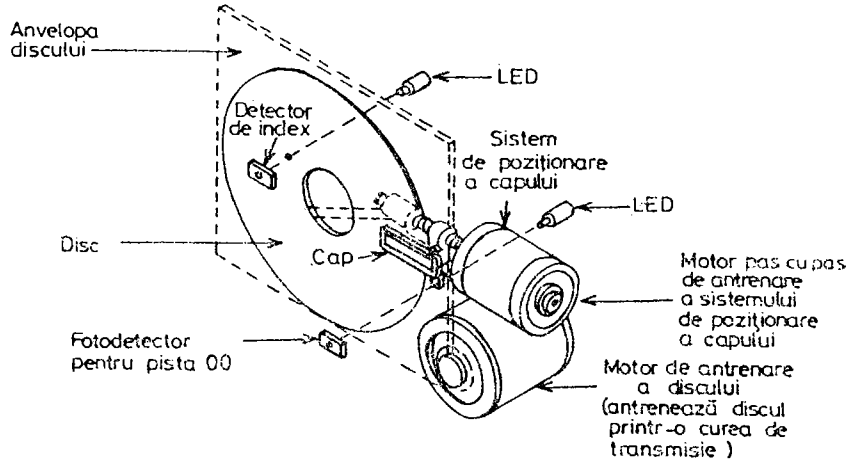


Fig. 13-16. Sistem de antrenare standard pentru disc flexibil.

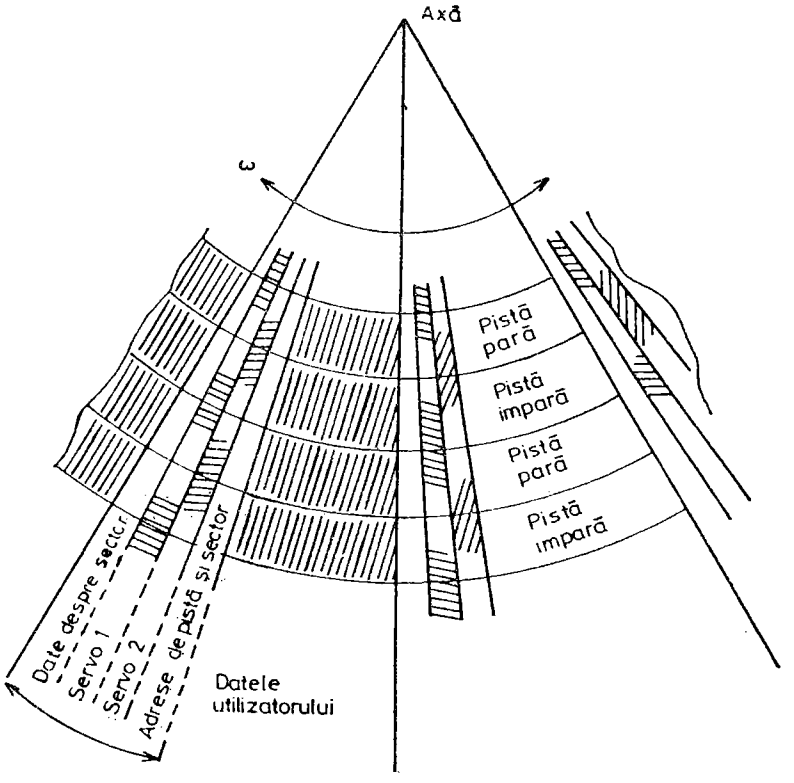


Fig. 13-17. Formatul Dynastore pentru disc flexibil.

spre axul discului, semnalul va fi mai mare în timpul zonei SERVO 2 decît în zona SERVO 1, întrucît mai mult din zona înregistrată a lui SERVO 2 va fi sub cap. Dacă, capul va fi descentrat, spre periferia discului, semnalul SERVO 1 va fi mai mare. Astfel se poate genera un semnal de reacție pentru controlul poziției capului, prin citirea celor două semnale SERVO 1 și SERVO 2 cu ajutorul unor circuite de eșantionare și memorare („sample-hold"). Semnalul de eroare a poziției este dat de diferența dintre cele două semnale memorate :

$$\text{EROAREA POZIȚIEI} = \text{SERVO 1} - \text{SERVO 2}. \qquad (13\text{-}5)$$

Semnalul de eroare este egal cu zero atunci cînd capul va fi centrat pe pistă și pozitiv cînd capul este descentrat spre axul discului. Acest semnal este utilizat pentru reglarea poziției *finale* a capului.

Cînd capul este *deplasat* către o nouă pistă, acest mecanism de reglare a poziției este dezactivat pînă cînd capul este la o distanță egală cu jumătate din lățimea pistei față de poziția corectă, cea corespunzătoare valorii zero a semnalului EROAREA POZIȚIEI. Înainte de acest punct, capul este antrenat printr-o comandă *digitală*, la o viteză controlată către pista corectă. Prin numărarea inversărilor de polaritate a semnalului EROAREA POZIȚIEI

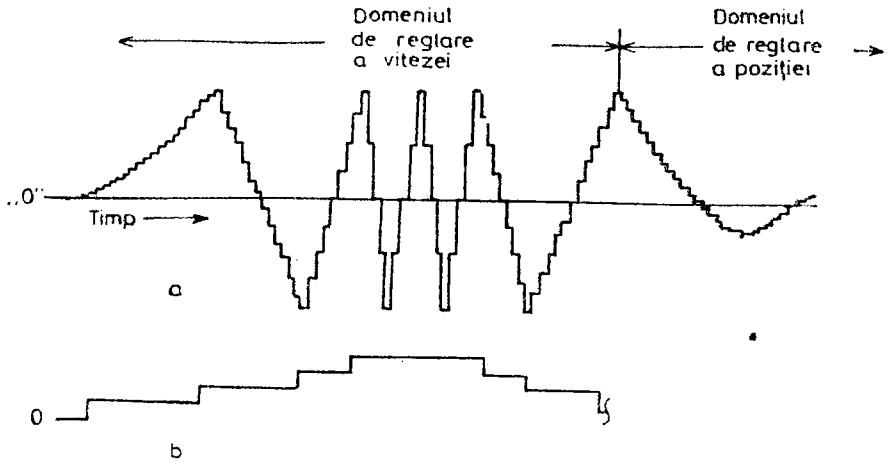


Fig. 13-18. Deplasarea capului peste nouă piste la discul flexibil Dynastore; *(a)* semnalul de eroare pentru poziție; *(b)* tensiunea de comandă a vitezei.

(vezi fig. 13-18), logica digitală comandă un *numărător de pistă reversibil* (track counter). Semnalul de reacție pentru controlul vitezei se obține din *frecvența* acestor inversări de polaritate. Tensiunea care *comandă* viteza este generată de un numărător reversibil de doi biți, astfel încît capul este accelerat și decelerat treptat pe parcursul primelor și ultimelor două piste ale deplasării. Cînd semnalul EROAREA POZIȚIEI se apropie de zero, pentru o anumită pistă, sistemul de reglare automată a poziției este activat pentru poziționarea definitivă a capului.

Este astfel posibil să se obțină electronic o centrare foarte precisă a capului pe pistă fără menținerea strînsă a unor anumite toleranțe. Un simplu motor de c.c., un cablaj și sistemul de transmisie pot fi deci utilizate pentru poziționarea cu acuratețea obținută *electronic* de la semnalul normal de citire. Un sistem similar de poziționare a capului este folosit pentru discul IBM3331 (6 miliarde de biți). În acest sistem, una din cele 24 de suprafețe ale discului este înregistrată complet cu informația pentru reglarea poziției sub formă de „tablă de șah", astfel că informația pentru reglare este disponibilă *continuu*. Datorită preciziei îmbunătățite de poziționare a capului pot fi puse pe fiecare suprafață a discului de *două ori mai multe piste*, față de cîte erau posibile în cazul unităților de disc anterioare (de exemplu 2311), care utilizau semnalul de reacție de la sistemul de comandă a poziției capului.

Atît unitatea de disc IBM3331 cît și unitatea de disc Dynastore, folosesc încă o altă tehnică interesantă pentru realizarea *compromisului complexitate electronică față de implementarea mecanică*. Ambele unități utilizează tehnica de *modulare în frecvență modificată* (MFM) pentru înregistrarea datelor, ceea ce face posibil să se stocheze de două ori mai mulți biți pe fiecare pistă. Complexitatea electronicii adiționale este compensată pe deplin întrucît fiecare suprafață a discului este echivalentul a două suprafețe anterioare. Figura 13-19 prezintă pentru comparare principiul modulației de frecvență modificată (MFM) și tehnica de modulație în fază simplă. Ambele scheme

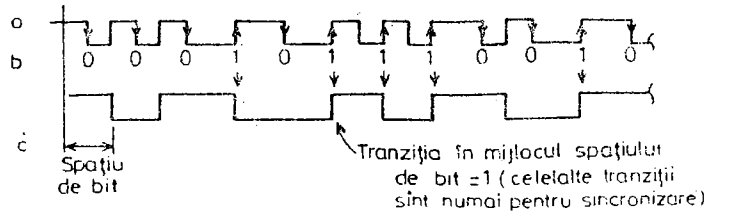


Fig. 13-19. *(a)* Modulație în fază simplă ; *(b)* datele ; *(c)* modulație în frecvență modificată.

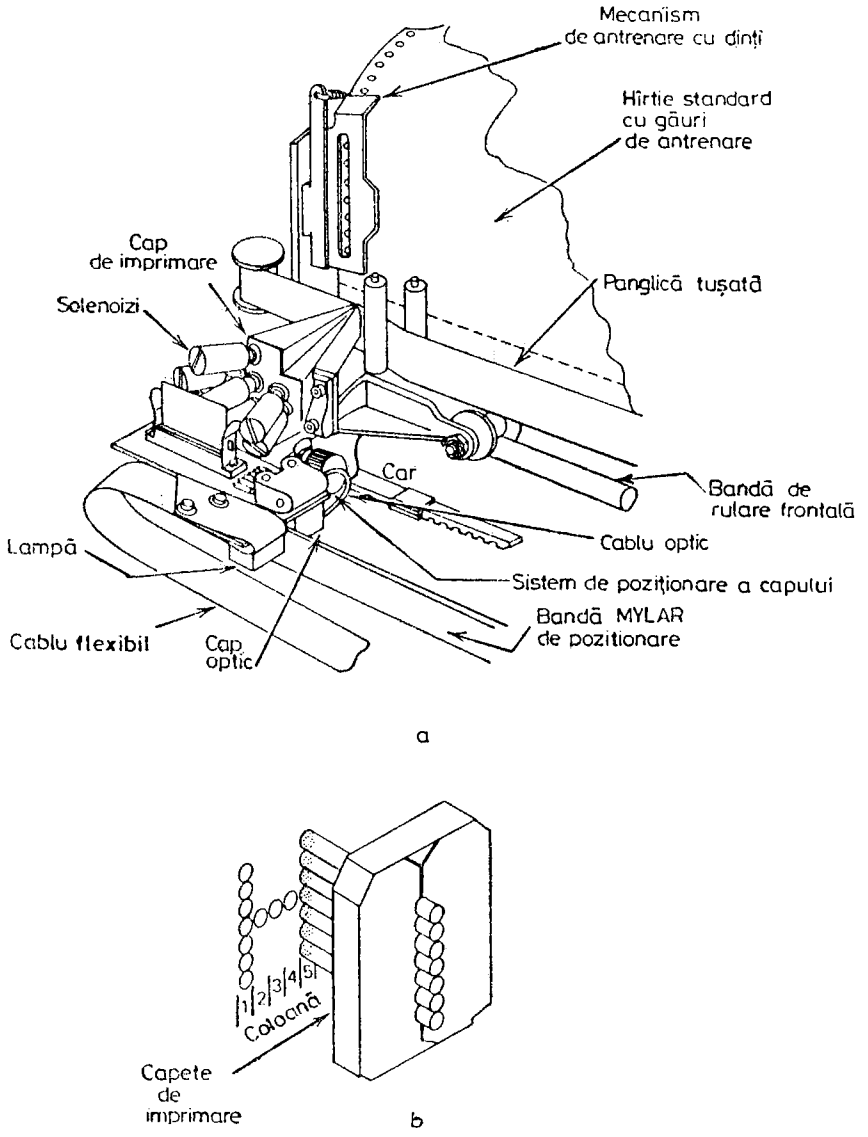


Fig. 13-20. Imprimantă Centronics : *(a)* structura carului ; *(b)* detaliu al capului de imprimare.

sînt autosincronizate, dar în timp ce pentru modularea simplă în fază semnalul de ceas este generat de un circuit monostabil (ca în fig. 12-17) pentru tehnica MFM este necesar un circuit de decodare mult mai complex. Explicația pentru ineficiența modulației în fază simple este dată de faptul că pentru *fiecare* bit este generată *o tranziție de sincronizare*, astfel că pentru orice bit trebuie generate una sau două tranziții. Pentru MFM nu pot apărea tranziții la *o distanță mai mică de intervalul unui bit*. Fiecare interval asociat unui bit are o tranziție în mijloc pentru a indica transmiterea unui 1, și nici o tranziție pentru 0. La marginile intervalelor de bit sînt adăugate tranziții suplimentare între perechi consecutive de zerouri. Astfel, tranzițiile sînt la 1, 1,5 sau 2 intervale de bit. Semnalul de ceas este obținut printr-o buclă cu calare de fază, PLL (vezi fig. 13-13), care generează un semnal de ceas continuu, calat în fază cu tranzițiile primite. Decodarea fiecărui bit constă în a vedea dacă tranziția este detectată aproape de mijlocul intervalului corespunzător bitului.

13.9. IMPRIMANTE CU MATRICI DE PUNCTE

Un exemplu foarte bun de utilizare a reacției negative și a tehnicilor de partajare a timpului îl constituie imprimanta cu matrici de puncte, Centronics. În locul utilizării unui electromagnet separat pentru fiecare din cele 132 caractere de-a lungul liniei, se folosesc șapte electromagneți de viteză mare care sînt deplasați de-a lungul liniei de un car, așa cum arată în figura 13-20. Electromagneții comandă mișcarea unor mici „ciocănele" metalice care sînt montate împreună într-o coloană verticală ce reprezintă înălțimea unei litere. În timp ce carul transportor se deplasează peste fiecare literă, cei șapte electromagneți comandă ciocănelele care marchează sau nu fiecare din cele cinci poziții care formează reprezentarea unui caracter sub formă de matrice în puncte. Așa cum se arată în figura 13-21, prin această metodă poate fi tipărit un număr total de 64 caractere.

Un mecanism simplu deplasează carul peste hîrtie. Distanța între coloane, pe orizontală, este menținută cu exactitate prin sesizarea optică a poziției *curente* a capului de tipărire. O fotocelulă pe capul de tipărire sesizează lumina transmisă printr-o bandă de Mylar imprimată cu fîșii alternante transparente și întunecate. Caracterele pot fi tipărite astfel perfect, chiar dacă capul are o viteză mai mare la începutul liniei decît la sfîrșitul ei.

Ca în orice sistem care utilizează partajarea timpului apar probleme legate de viteză. Cele șapte ciocănele trebuie să puncteze de $5 \times 132 = 600$ ori pentru fiecare linie, în loc de o singură dată ca în cazul unei singure bobine pe caracter.

Întrucît sînt numai șapte electromagneți întreg efortul care se cere este de a-i face rapizi. În cazul imprimantei Centronics, electromagneților le este necesar un timp de numai 1 ms pentru a tipări un punct și a reveni, astfel că pot fi tipărite 165 caractere pe secundă. Pentru o viteză de tipărire mai mare, un alt model de imprimantă utilizează două capete de tipărire, care se deplasează împreună, fiecare tipărind o jumătate de linie. Astfel viteza de tipărire se dublează, obținîndu-se 330 caractere/secundă sau 125 de linii complete pe minut. Întîrzierea dată de întoarcerea carului este eliminată prin *tipărirea alternativă a liniilor în sens invers*. Logica de comandă inversează literele în acest caz într-un mod simplu, utilizînd numărătoare reversibile pentru a număra coloanele și poziționarea caracterelor.

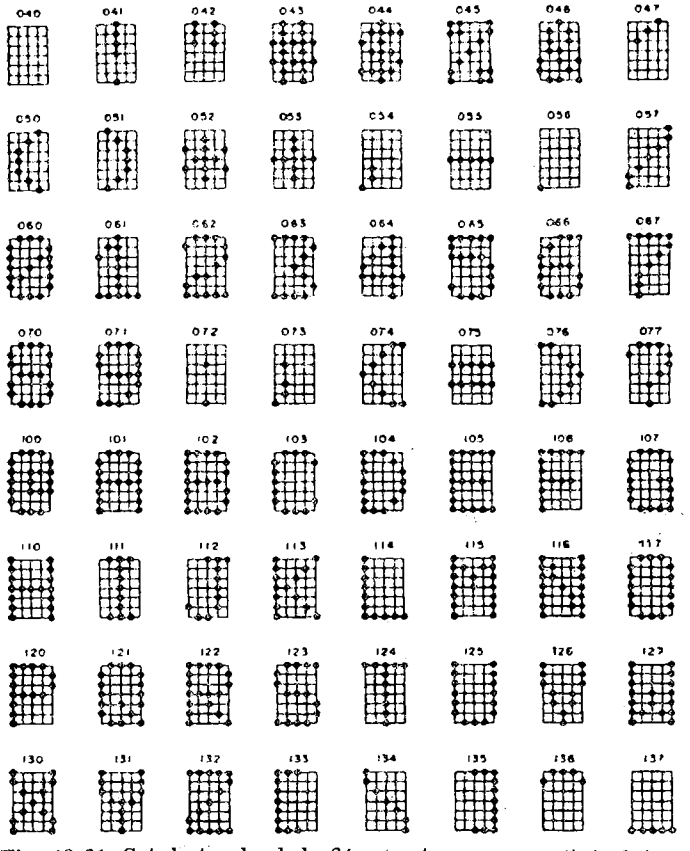


Fig. 13-21. Setul standard de 64 caractere reprezentate într-o matrice de 5×7 puncte.

Întrucît forma caracterelor este memorată într-un ROM, un set special de caractere poate fi uşor obţinut, prin schimbarea conţinutului memoriei ROM. Caracterele aldine sînt tipărite ignorînd fiecare alt impuls de la banda de codare a poziţiei carului. Este foarte important faptul că putem genera electronic aproape orice semnal, ceea ce duce la simplificarea părţii *mecanice* a sistemului. Transformarea codurilor caracterelor într-o secvenţă de semnale de comandă la cei şapte electromagneţi costă mult mai puţin decît un mecanism mult mai complex.

13.10. SISTEME DE AFIŞAJ ŞI TASTATURI

Monitorul TV este un exemplu de aplicaţie performantă a tehnicii de partajare a timpului în care, un desen conţinînd aproximativ 250 000 de elemente este produs prin modularea unui *singur* fascicul de electroni. Generarea electronică a unui semnal video pentru afişarea unor caractere în matrice de 5 × 7 puncte cere citirea conţinutului unui ROM într-o ordine diferită. Întrucît un monitor TV constituie un exemplu de componentă ieftină, sistemele de afişaj pe tub catodic (CRT display) au devenit o interfaţă standard de ieşire, acolo unde nu este necesară obţinerea unui text imprimat, tipărit *(hard-copy)*.

Un sistem de afișaj pe tub catodic este atît de flexibil încît a devenit ineficientă folosirea altor dispozitive de afișare într-un sistem care cuprinde un astfel de sistem de afișaj. De exemplu, lămpile indicatoare pot fi înlocuite prin programarea unui anumit mesaj care să apară într-o anumită parte a ecranului. Întregul panou de control poate fi în esență, reprezentat ca un format de display. În felul acesta modificările unui sistem, diferite moduri de operare pot fi realizate fără schimbări hardware.

Un *terminal* CRT (vezi fig. 13-22), combină o tastatură ca de mașină de scris, cu un afișaj pe tub catodic (prin programare se poate obține ca o tastatură standard să îndeplinească funcțiile particulare ale *oricărui* panou de control). Funcții speciale se pot programa simplu ca secvențe de taste. De exemplu, toate funcțiile care în mod obișnuit se utilizează la consola de comandă a unui calculator, pot fi obținute la un terminal. Fiecare buton special este programat ca o secvență pe tastatură. Spre exemplu, apăsînd tasta „L", se poate obține același lucru ca în cazul tastei „LOAD" de pe o consolă. O consolă specială prezintă avantajul de a fi întrucîtva auto-explicativă, dar adesea este practic să se afișeze pe ecran, toate *tastele semnificative* care pot fi apăsate la un moment dat. În acest fel sistemul este autoexplicativ, și numai funcțiile pertinente sînt afișate în orice moment.

Unele terminale cu afișaj pe tub catodic cuprind un microprocesor programat pentru realizarea unei game întregi de funcții logice în *cadrul terminalului*. Astfel de terminale sînt numite *terminale inteligente*. Acestea, în mod obișnuit permit ca textul să fie pregătit și modificat prin ștergere, inserare, deplasare etc., direct pe ecran. Este posibil, de exemplu, să se citească un fișier (de exemplu numele și adresa unui beneficiar), să se opereze modificări asupra lui, apoi să se trimită înapoi către calculatorul central. Cum microprocesoarele devin din ce în ce mai puternice, terminalele inteligente cu afișaj pe tub ca-

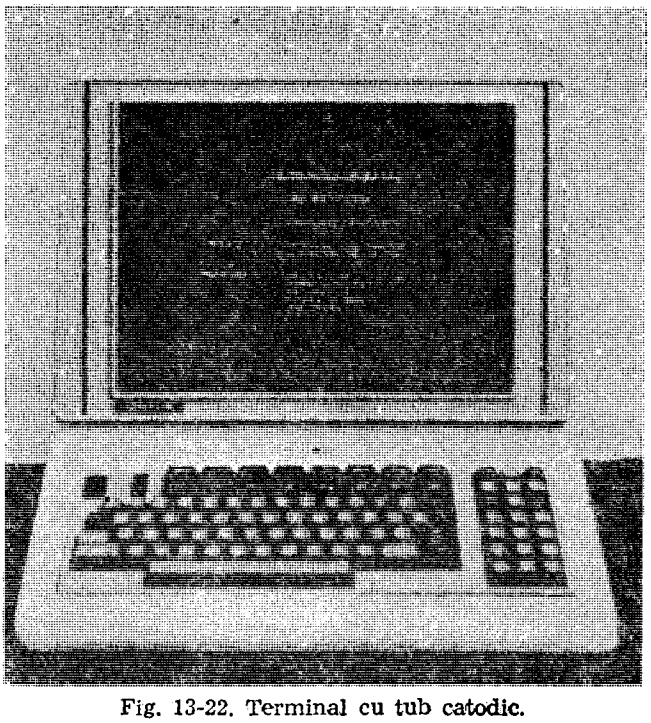


Fig. 13-22. Terminal cu tub catodic.

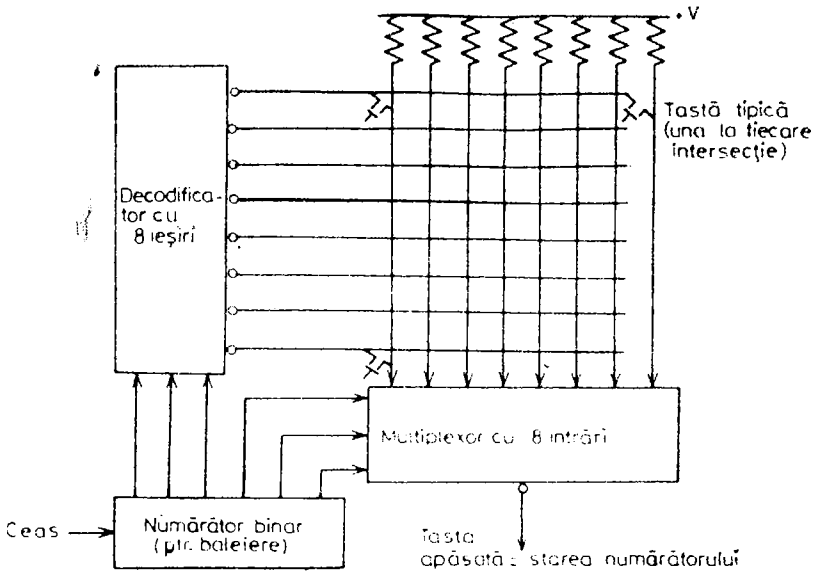


Fig. 13-23. Sistem de baleiere a tastaturii.

todic vor fi din ce în ce mai aproape de un întreg sistem de calcul. Aceste sisteme sînt disponibile cu interfețe pentru memorii pe disc, pentru casetă și imprimantă.

Tastaturile cu caracter special și sistemele de afișaj luminescent mai sînt încă folosite pentru anumite aplicații. Pentru interfațarea tastaturilor cu intrările de date seriale sau paralele se utilizează circuite LSI, realizate pe un singur cip. Toate aceste circuite baleiază tastele pe rînd, una cîte una, așa cum se vede în figura 13-23. Contactele tastelor formează o matrice X—Y. Pentru fiecare din cele 64 de stări ale numărătorului este adresată o anumită tastă, care se găsește la intersecția dintre ieșirea decodificatorului și intrarea multiplexorului, selectate în momentul respectiv. Cînd o anumită tastă este adresată de numărător, ieșirea multiplexorului va fi în 1 dacă *tasta respectivă* este apăsată. Întrucît ieșirea are asociat un interval de timp pentru fiecare tastă, se poate utiliza o logică cu partajarea timpului pentru eliminarea zgomotelor de contact și pentru înlăturarea efectului de apăsare a unei taste înainte ca tasta apăsată anterior să fie eliberată.

Dacă *trei sau mai multe* contacte sînt deschise simultan în matricea X—Y, rezultatul va fi incorect, în afară de cazul cînd se plasează o diodă în serie cu fiecare comutator. Figura 13-24 arată un exemplu de cale de semnalizare falsă cînd contactele X_1—Y_1, X_2—Y_1 și X_2—Y_2 sînt închise. Situația apare ca și cum contactul X_1—Y_2 ar fi închis, deoarece intrarea Y_2 a multiplexorului va fi trasă jos de X_1 pe calea falsă permisă de închiderea celorlalte trei comutatoare. Prin punerea unei diode în serie cu fiecare contact, calea de semnalizare falsă este blocată prin dioda de la comutatorul X_2—Y_1.

Deoarece componentele LED conduc curentul numai într-o direcție, ele pot fi comandate într-o matrice X—Y fără a avea problemele date de calea falsă de semnalizare. În plus, diodele LED sînt *mai strălucitoare*, la același curent mediu, dacă sînt comandate în pulsuri și nu continuu. Este, de asemenea, posibil să se comande la un moment dat, numai o linie din matricea X—Y a diodelor, obținîndu-se o eșalonare în timp a comenzilor. De exemplu,

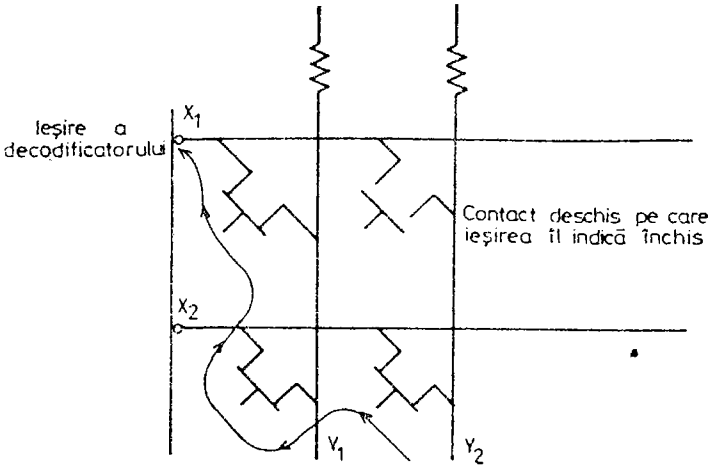


Fig. 13-24. Calea de semnalizare falsă într-o matrice de contacte.

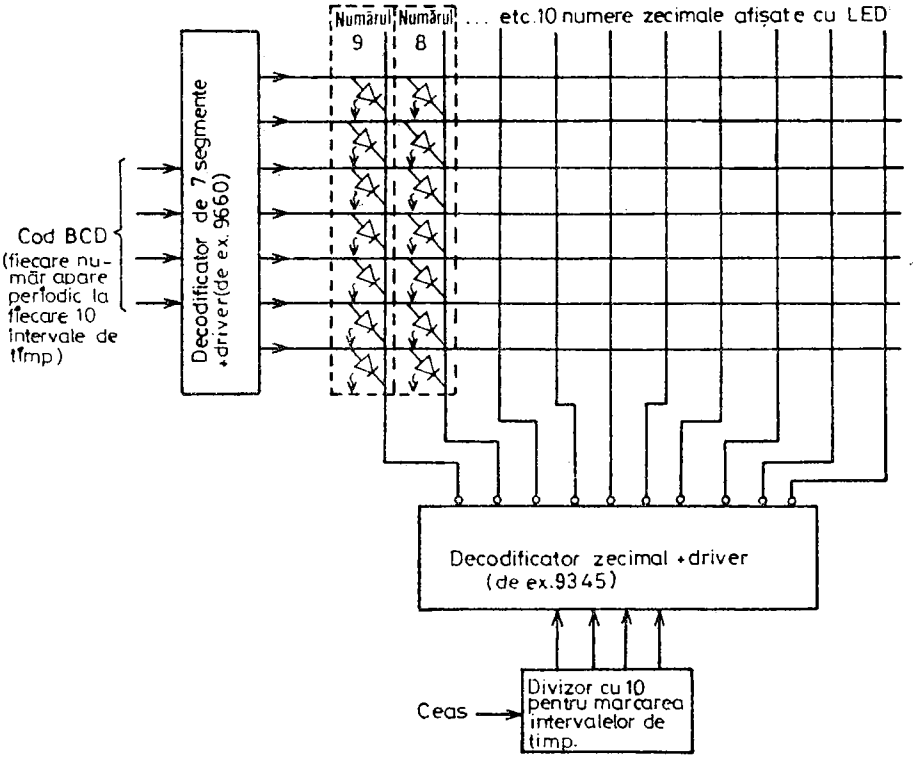


Fig. 13-25. Sistem de afişare cu LED-uri pentru 10 cifre zecimale realizat prin multiplexare în timp.

o matrice de 80 de LED-uri poate fi comandată de opt generatoare pozitive de curent și de un decodor cu 10 ieșiri pentru a selecta pe rînd liniile. Astfel acest sistem de afișare conține 10 LED-uri cu șapte segmente, multiplexînd în timp același generator de curent și logica asociată acestuia.

Elementele de afișare cu 7 segmente pot utiliza un singur decodor pentru a comanda mai multe cifre, așa cum se poate vedea în figura 13-25. Este de remarcat faptul că această *partajare a timpului* în cazul oricărui dispozitiv de intrare-ieșire, *se poate extinde în logica de control atît de mult cît ne convine.*

Întrucît fiecare cifră ocupă un eșantion de timp separat, orice operație logică sau matematică poate fi făcută pe cifră, multiplexînd în timp circuitele logice și lăsînd pe seama *efectului integrator al ochiului* perceperea globală a rezultatului.

13.11. REPREZENTAREA DIGITALĂ A SEMNALELOR ANALOGICE

Tensiunile analogice pot fi reprezentate într-un sistem digital în același fel ca și numerele. De exemplu, un semnal în gama de tensiune de $\pm 2{,}048$ V poate fi reprezentat, cu o rezoluție de 1 mV, printr-un număr binar de 12 biți. Numărul binar reprezintă polaritatea și numărul de milivolți. Procesul de transformare a tensiunilor analogice în numere digitale se numește *conversie analog-digital* (A/D). Transformarea inversă, dintr-un număr digital, reprezentînd un semnal de tensiune, într-un semnal analogic, se numește *conversie digital-analog* (D/A).

Formele de undă ale semnalelor continue pot fi reprezentate prin eșantionarea periodică a semnalului. Astfel, reprezentarea digitală a semnalului este o *secvență de numere binare.* Semnalul poate fi reconstituit prin convertirea secvenței de numere binare, într-o secvență de nivele de tensiune urmată de o filtrare, pentru a netezi descontinuitățile semnalului astfel obținut (vezi fig. 13-26). Frecvența de eșantionare trebuie să fie cel puțin *de două ori frecvența componentei celei mai înalte din spectrul semnalului.*

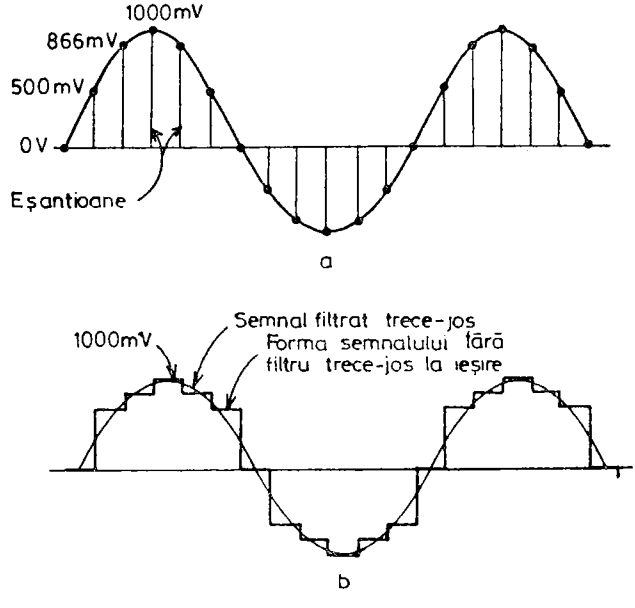


Fig. 13-26. Eșantionarea și reproducerea unui semnal continuu : *(a)* semnalul logic de intrare ; *(b)* semnalul reconstruit.

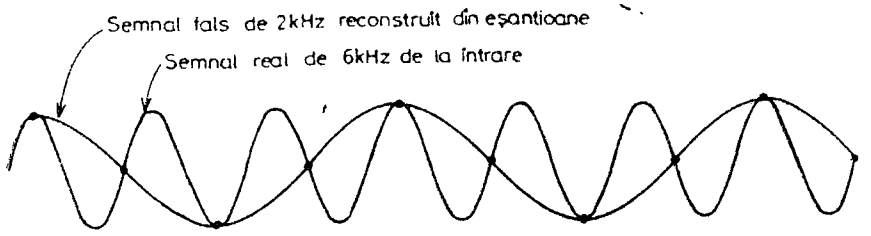


Fig. 13-27. Refacerea incorectă a unui semnal cu frecvenţa mai mare decît 1/2 din frecvenţa de eşantionare.

Dacă în semnalul de intrare sînt prezente componente de frecvenţă mai mare, vor rezulta distorsiuni în momentul reconstituirii semnalului analogic, deoarece aceste componente vor fi convertite în „semnale diferenţă" de frecvenţe mai mici. De exemplu în figura 13-27 se poate vedea cum semnalul iniţial de 6 kHz, eşantionat la 8 kHz, va fi reconstituit într-un semnal de $6-8/2 = 2$ *kHz*. Pentru a înlătura distorsiunile de acest fel, se va utiliza un *filtru trece-jos* pentru a elimina, înainte de eşantionare, componentele semnalului de frecvenţă mai mare decît jumătate din frecvenţa de eşantionare.

Cu un filtru similar şi la ieşire, semnalul reconstituit şi filtrat poate fi o reproducere perfectă a semnalului de intrare, în gama de frecvenţă reprodusă. Desigur, rezoluţia limitată a reprezentării digitale va duce la *zgomotul de cuantizare* iar deficienţele filtrului vor duce la apariţia unor frecvenţe nedorite. Totuşi, din punct de vedere teoretic, procesul de eşantionare este capabil de o reproducere perfectă a semnalului.

Un sistem general D/A include un filtru trece-jos şi un convertor A/D la intrare precum şi un convertor D/A şi un filtru trece-jos la ieşire, aşa cum se arată în figura 13-28. Uneori, filtrele trece-jos nu sînt necesare, dacă răspunsul în mod normal lent al traductoarelor elimină componentele de frecvenţă mai mare decît jumătate din frecvenţa de eşantionare.

Digitizarea semnalelor analogice este avantajoasă din mai multe considerente. Cel mai important motiv este probabil *precizia*. În cazul folosirii tehnicilor analogice, costul creşte rapid pentru o precizie mai mare de aproximativ 1% pentru fiecare operaţie. Întrucît erorile tind să se acumuleze, operaţiile complexe scapă foarte uşor de sub control. În schimb, în cazul prelucrării digitale, precizia se dublează de fiecare dată, cînd lungimea cuvîntului creşte cu un bit. De exemplu, cu un cuvînt de 16 biţi se poate obţine o precizie de 0,0015% utilizînd circuite digitale obişnuite, produse pe scară largă. Datele reprezentate numeric pot fi *memorate* şi extrase perfect de pe suporturi magnetice. Tehnicile de detecţie şi corecţie a erorilor permit obţinerea unor rezultate perfecte cu toate imperfecţiunile mediului de înregistrare. Semnalele pot fi *transmise* la distanţe mari şi regenerate cu circuite digitale simple obţinîndu-se performanţe apropiate de ideal.

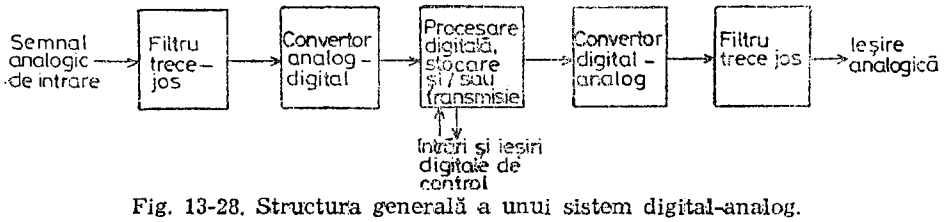


Fig. 13-28. Structura generală a unui sistem digital-analog.

Dacă problemele practice tind să limiteze performanţele care se pot obţine cu filtrele analogice, prin filtrarea digitală se elimină practic orice limitare. Astfel, cu un filtru digital (unde fiecare eşantion de ieşire reprezintă suma ponderată a 128 eşantioane anterioare de intrare) se poate obţine o atenuare de aproximativ 100 dB în afara benzii de trecere (vezi Ref. 2)

Un alt mare avantaj al prelucrării numerice îl constituie *flexibilitatea oferită de programarea digitală*. De exemplu, un sistem de control al unui proces industrial poate fi modificat pentru a îmbunătăţi procesul, printr-o modalitate simplă — încărcarea de noi programe. Programele pot fi scrise astfel încît să optimizeze procesul de control automat prin automodificare, în funcţie de rezultatele obţinute la un moment dat. Orice funcţie neliniară poate fi generată uşor prin „tabele de corespondenţă". Datele adunate în timpul experienţelor spaţiale, în prospectările petroliere, spre exemplu, pot fi prelucrate mai tîrziu într-un sistem de calcul, dacă sînt stocate digital pe o bandă magnetică. În cazul sistemelor de control numeric a operaţiilor mecanice, este posibilă definirea digitală a funcţiilor maşinii folosind un „limbaj" special, prin care se determină automat succesiuni de operaţii mecanice.

Un alt avantaj este un rezultat al vitezei care se poate obţine în cazul funcţionării digitale. Această caracteristică face posibilă *partajarea timpului* unui circuit digital între un mare număr de semnale analogice convertite digital. Astfel, semnalele telefonice pot fi reprezentate numeric prin 8 000 de eşantioane de 7 biţi pe secundă sau 56 000 biţi pe secundă. Întrucît circuitele digitale obişnuite pot lucra la o frecvenţă de sute de ori mai mare, un singur multiplexor digital poate comuta *sute* de conversaţii telefonice, fiecare avînd alocat un *interval de timp*. Memoriile digitale RAM pot fi utilizate pentru a comuta intervalele de timp, astfel că, şapte memorii RAM rapide de 1 024 biţi pot să realizeze funcţia unei matrici de comutare cu 1 024 × 1 024 puncte.

13.12. CONVERSIA ANALOG-DIGITAL

În acest capitol vom prezenta cîteva tehnici de conversie. Convertoarele digital-analog sînt disponibile sub forma circuitelor integrate monolitice (de exemplu MC1406). În principiu, în cazul convertoarelor D/A, fiecare intrare digitală comuta un generator de curent, cu o pondere numerică corespunzătoare. Aceşti curenţi sînt însumaţi şi convertiţi într-o tensiune proporţională la ieşire.

Conversia analog-digital este însă o problemă mai dificilă. Cea mai întîlnită tehnică de conversie analog-digital de viteză mare utilizează principiul reprezentat în figura 13-29. *Aproximaţiile succesive* ale valorii tensiunii de intrare sînt realizate de un convertor D/A a cărui ieşire este comparată cu semnalul de intrare printr-un circuit *comparator*. Cînd valoarea semnalului de intrare este mai mare decît aceea a ieşirii convertorului D/A, ieşirea comparatorului este activă. Începînd cu cel mai semnificativ bit, în poziţia fiecărui bit este încercat 1, iar *dacă semnalul de intrare este mai mare decît semnalul rezultat la ieşirea convertorului D/A, este păstrat 1*. După fiecare comparaţie codul numeric, rămas în latch-ul adresabil, reprezintă tensiunea de intrare. (Bistabilul D este necesar deoarece latch-ul nu este un dispozitiv care comută comandat de frontul ceasului).

Ca un exemplu de conversie A/D prin aproximaţie succesivă, să presupunem că definim un semnal în gama ± 127 mV printr-un număr de 8 biţi. Cel mai semnificativ bit va reprezenta semnul iar ceilalţi vor reprezenta res-

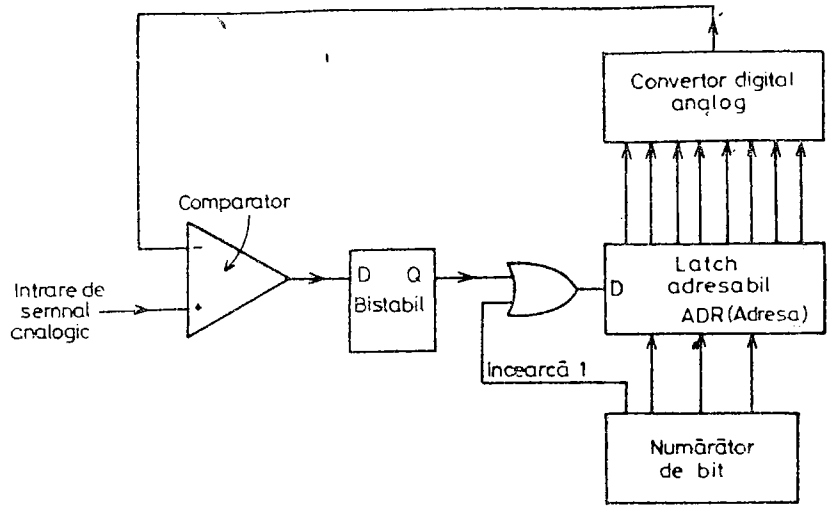


Fig. 13-29. Conversia analog-digital prin aproximare succesivă.

pectiv 64, 32, 18, 8, 4,2, 1 mV. De exemplu, unui semnal de intrare de 82 mV îi va corespunde prin secvența de aproximație succesivă arătată în Tabelul 13-1.

Astfel, această tehnică găsește codul binar corespunzător, în principiu, prin divizarea gamei de posibilități rămase la jumătate, încercînd pînă la obținerea preciziei dorite.

Tabelul 13-1. **Exemplu de aproximare succesivă : intrarea = 82 mV**

Bit = 1	Conținutul latch-ului	Ieșirea D/A (mV)	<82 mV ? (condiție realizată, stochază 1)
7 (semn)	10000000	0	1
6 (MSB)	11000000	+64	1
5	11100000	+96	0
4	11010000	+80	1
3	11011000	+88	0
2	11010100	+84	0
1	11010010	+82	1
0	11010011	+83	0
REZULTAT	11010010		

Desigur dacă este posibil, aproximarea succesivă poate fi *programată.* În acest caz, pe lîngă procesorul programat, sînt necesare un convertor D/A la ieșire și un comparator de intrare. Folosind un circuit de *eșantionare-memorare (sample-and-hold)* același convertor D/A poate fi folosit pentru compararea mai multor ieșiri cu intrările. Circuitul de eșantionare-memorare înmagazinează o tensiune analogică pe un capacitor în timpul unui scurt impuls de *eșantionare.* Între impulsurile de eșantionare este disponibilă o tensiune continuă care este egală cu tensiunea de intrare la sfîrșitul perioadei de eșantionare (tipic — cîteva microsecunde). La un singur convertor D/A pot fi astfel conectate mai multe circuite de eșantionare-memorare, tensiunile de ieșire obținîndu-se pe rînd.

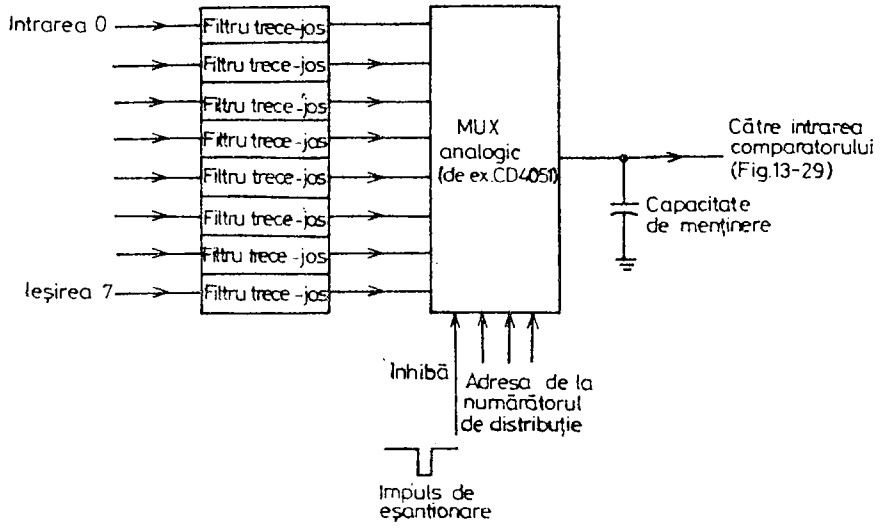


Fig. 13-30. Multiplexarea intrărilor analogice.

Circuitele de eşantionare-memorare sînt utilizate de asemenea în cazul semnalelor de intrare cu variaţie rapidă pentru a preveni schimbarea acestora în timpul procesului de conversie A/D. O schimbare a semnalului de intrare (corespunzătoare propagării unui transport) în timpul conversiei A/D cu aproximare succesivă, poate duce la un rezultat eronat. În cazul mai multor intrări analogice convertorul D/A, comparatorul şi circuitul de eşantionare-memorare lucrează adesea, prin *partajarea timpului* între mai multe intrări. Pentru a conecta, la un moment dat, una din intrările analogice la circuitul de eşantionare în timpul impulsului de eşantionare (vezi fig. 13-30), este folosit un *multiplexor analogic* (de exemplu multiplexorul CMOS, CD4051). După ce este realizată conversia A/D este eşantionat următorul canal de intrare. De remarcat, că pentru *fiecare* semnal de intrare este necesar un filtru trece-jos separat. Deoarece multiplexoarele analogice CMOS se comportă, cînd sînt deschise, ca nişte rezistenţe de valoare mică, ele pot fi utilizate şi ca demultiplexoare. Astfel, un singur convertor D/A poate comanda opt capacităţi de memorare pentru opt canale de ieşire. Pentru fiecare canal de ieşire sînt necesare un amplificator cu impedanţă mare de intrare şi un filtru trece-jos.

Dacă viteza de conversie nu este importantă se pot utiliza convertoare A/D mai simple. De exemplu voltmetrele digitale convertesc numeric tensiunile pentru afişare vizuală sau tipărire, astfel că semnalul trebuie convertit numai de 60 ori/s. În figura 13-31 este arătată tehnica de conversie prin integrare cu dublă pantă, utilizată în aceste voltmetre. Un capacitor de integrare se în-

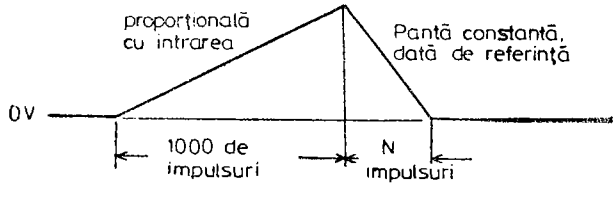


Fig. 13-31. Convertor A/D cu dublă pantă.

carcă cu o viteză constantă proporțională cu semnalul de intrare, pe durata a 1 000 de impulsuri de ceas. Condensatorul este descărcat apoi pînă la valoarea inițială a tensiunii la borne de un curent constant de referință. Numărul de impulsuri care corespund de descărcării indică direct tensiunea de intrare (de exemplu 1 000 de impulsuri arată că panta de intrare este egală cu cea de referință, 500 impulsuri indică o valoare egală cu jumătate din panta de referință etc.). Prin metoda de integrare cu dublă pantă se poate obține o precizie ridicată, folosind circuite neperformante, întrucît frecvența ceasului, valoarea capacității, tensiunea de prag și neliniaritățile nu afectează, din principiu, precizia măsurării. Desigur, dezavantajul acestui tip de conversie este viteza redusă. Folosind această tehnicăaufost realizate voltmetre monolitice integrate (LSI DVM), care au o rezoluție de 0,005%.

Conversia D/A de viteză mică este de asemenea, foarte simplă. Controlînd *factorul de umplere* al unui semnal digital care apoi este filtrat, se poate obține, cu precizie, tensiunea analogică corespunzătoare. Spre exemplu, un semnal logic cu amplitudinea de 5 V cu factorul de umplere 1/100, are o valoare medie de 5/100 = 50 mV. Printr-o filtrare bună se obține la ieșire o tensiune continuă de 50 mV. Pot fi folosite numărătoare digitale simple pentru a obține impulsuri modulate în durată, care apoi sînt integrate, obținîndu-se un nivel continuu. Sursele de tensiune de alimentare în comutație obțin tensiunea stabilizată de ieșire cu o foarte mică disipație, prin modularea în durată a impulsurilor.

Cînd se lucrează cu o gamă largă de amplitudini ale semnalului, codarea digitală *liniară* a semnalului poate fi ineficientă. Astfel, în sistemele telefonice cu codificare de tip PCM trebuie prelucrată o gamă largă de nivele ale vocii. Pentru a menține factorul de distorsiuni la un nivel foarte mic, semnalul vocal ar trebui reprezentat prin cel puțin 64 nivele de cuantizare. Totuși, amplitudinile semnalului telefonic pot varia într-o gamă de peste 128 : 1 ; pentru o codare liniară ar fi necesare 64 × 128 = 8 000 nivele sau 13 biți. Acest lucru ar fi foarte ineficient, cel mai mare nivel de semnal fiind cuantizat prea fin, în 8 000 de nivele.

Soluția constă în utilizarea treptelor de codare *neliniară*, care cresc pe măsura creșterii amplitudinii semnalului. În acest scop se utilizează un cod de 8 biți din care 3 biți indică una din cele 8 *trepte* (1, 2, 4, 8, 16, 32, 64, 128) iar ceilalți 5 biți sînt folosiți pentru polaritate și 16 nivele. Astfel, fiecare cod binar reprezintă un nivel bine determinat, *dar distanța între nivelele succesive devine mai mare cu cît semnalul este mai departe de 0V*. Dacă cel mai mic par este de 1 mV, atunci putem ajunge pînă la ±16 mV cu pași de 1 mV, următorii 16 pași vor fi de 2 mV fiecare, apoi 16 pași de cîte 4 mV, 16 pași de 8 mV, 16 pași de 16 mV, 16 pași de 32 mV, 16 pași de 64 mV și 16 pași de cîte 128 mV fiecare. Astfel gama de tensiuni rezultată, cu pasul minim de 1 mV, va fi :

$$\pm 16(1 + 2 + 4 + 8 + 16 + 32 + 64 + 128) =$$

$$= \pm\ 16 \times 255 = \pm\ 4\,080 \text{ mV} \qquad (13\text{-}6)$$

Un convertor D/A pentru un astfel de cod comprimat se poate obține dintr-un convertor D/A de 5 biți și un convertor D/A *multiplicator* de 8 biți, fiecare bit fiind comandat de ieșirea unui *decodor* digital. Conversia analog-digital poate fi realizată prin aproximare succesivă (vezi fig. 13-29) folosind un convertor D/A neliniar. Dacă operațiile matematice trebuie realizate pe acest tip de codare neliniară, conversia la un cod liniar de 13 biți se poate face ușor, așa cum se arată în tabelul 13-2.

Tabelul 13-2. **Reprezentarea liniară cu 13 biţi a codului PCM de opt biţi**

Cod de 8 biţi	Valoarea binară echivalentă cu 13 biţi	Dimensiunea pasului (mV)
S 0 0 0 W X Y Z	S 0 0 0 0 0 0 0 1 W X Y Z	1
S 0 0 1 W X Y Z	S 0 0 0 0 0 0 1 W X Y Z 0	2
S 0 1 0 W X Y Z	S 0 0 0 0 0 1 W X Y Z 0 1	4
S 0 1 1 W X Y Z	S 0 0 0 0 1 W X Y Z 0 1 1	8
S 1 0 0 W X Y Z	S 0 0 0 1 W X Y Z 0 1 1 1	16
S 1 0 1 W X Y Z	S 0 0 1 W X Y Z 0 1 1 1 1	32
S 1 1 0 W X Y Z	S 0 1 W X Y Z 0 1 1 1 1 1	64
S 1 1 1 W X Y Z	S 1 W X Y Z 0 1 1 1 1 1 1	128

S = semn

13.13. AMPLIFICATOARE OPERAŢIONALE

Ori de cîte ori tehnicile digitale se folosesc pentru prelucrarea semnalelor analogice, sîntem puşi în faţa unor decizii fundamentale: unde trebuie făcută separarea între reprezentarea digitală şi cea analogică a semnalelor. Deşi tehnicile digitale prezintă multe avantaje, *tehnicile analogice sînt adesea mai simple şi mai economice*. Proiectantul de sisteme digitale trebuie să ştie ceea ce poate fi făcut uşor cu circuite analogice.

Adesea este bine să nu convertim imediat toate semnalele analogice în semnale numerice. De multe ori tehnicile analogice pot fi utilizate, pentru a prelucra semnalele mai economic, înainte de a fi transformate digital. Calea care trebuie urmată întotdeauna este folosirea tehnicii care se potriveşte cel mai bine scopului. Mecanismul de poziţionare din figura 13-1 este un bun exemplu. Deşi este posibilă convertirea digitală a semnalului analogic dat de senzorul de poziţie, scăderea lui din semnalul digital de comandă şi transformarea rezultatului într-o tensiune analogică de comandă a motorului, se realizează *mult mai simplu* printr-o scădere analogică, la intrarea amplificatorului.

Un alt exemplu este tensiunea de comandă pentru reglarea vitezei din figura 13-4. Putem obţine rampa de tensiune prin numărare digitală înainte şi înapoi, convertind apoi rezultatul digital într-un semnal analogic.

Totuşi, este mult mai simplu să se genereze două nivele analogice, folosind două rezistenţe diferite conectate la ieşirea unei porţi digitale, obţinînd pantele cu un simplu circuit analogic de integrare.

Numai pentru un număr foarte mic de circuite integrate analogice care s-au dovedit utile într-o gamă largă de aplicaţii, s-a dezvoltat o piaţă suficient de mare pentru a putea fi considerate componente ieftine. Excepţie fac *amplificatoarele operaţionale* care au devenit componente liniare standard şi sînt disponibile, patru pe o capsulă, pentru mai puţin de 1$.

Majoritatea prelucrărilor analogice sînt realizate prin diverse conexiuni ale amplificatoarelor operaţionale. Aşa cum se vede în figura 13-32 *a*, un amplificator operaţional ideal are un cîştig *diferenţial* de tensiune (A_v) infinit. Chiar şi circuitele reale ieftine au cîştigul în tensiune tipic 200 000 şi minim 20 000.

Întrucît amplificatorul operaţional de bază are amplificarea în tensiune necontrolată, nu poate fi folosit pentru prelucrări analogice, fără adăugarea *reacţiei negative*. Circuitul din figura 13-32 *b*, *inversează* semnalul de intrare şi îl amplifică, avînd cîştigul controlat precis de raportul celor două rezistenţe. Acest lucru se poate demonstra dacă se observă că orice tensiune de intrare

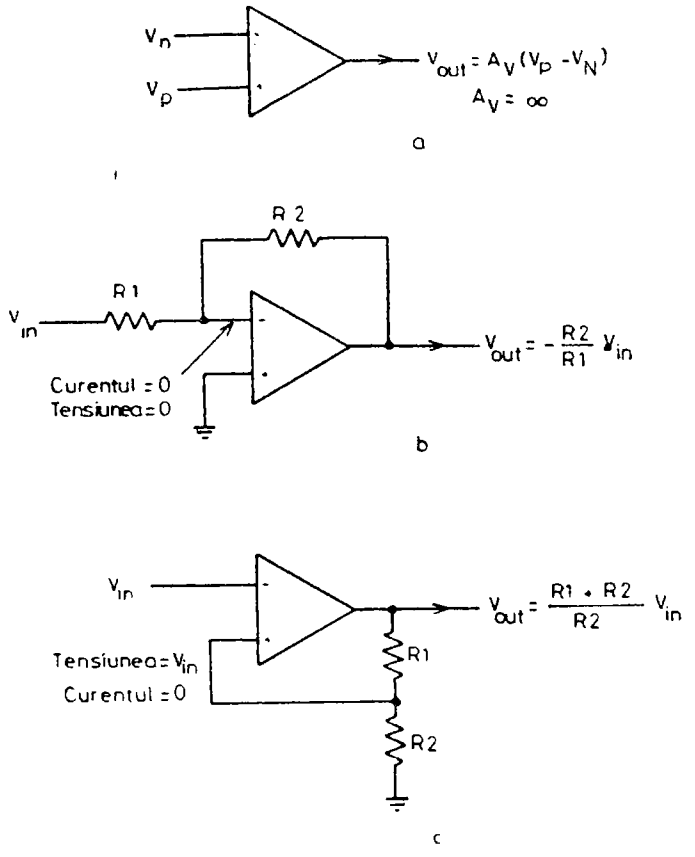


Fig. 13-32. Amplificatoare operaționale : *(a)* amplificatorul ; *(b)* amplificator inversor cu cîștigul controlat de circuitul de reacție ; *(c)* amplificator neinversor cu cîștigul controlat de circuitul de reacție.

dă naștere unui curent înspre intrarea inversoare (—) a amplificatorului, de valoare $I_{in} = V_{in}/R_1$. Tensiunea de la ieșirea amplificatorului este negativă, fiind dată de curentul care trece prin rezistența R_2, egal dar de sens opus cu cel de la intrare :

$$V_{out} = -I_{in} \cdot R_2. \tag{13-7}$$

Substituind, se obține :

$$V_{out} = -\frac{V_{in}}{R_1} \cdot R_2 \tag{13-8}$$

Rezultă deci :

$$V_{out} = -\frac{R_2}{R_1} \cdot V_{in} \tag{13-9}$$

Astfel, ieșirea amplificatorului operațional furnizează suficientă tensiune pentru a menține tensiunea de 0V la intrarea sa inversoare. Impedanța de intrare este tocmai R_1, deoarece intrarea inversoare a amplificatorului este la același potențial ca masa.

Un *amplificator neinversor* (vezi fig. 13-32 *c*) se obține prin divizarea tensiunii de la ieșire cu două rezistențe și conectarea mărimii rezultate pe in-

trarea inversoare (—) a amplificatorului. De asemenea, amplificatorul va produce o tensiune de ieşire, astfel ca diferenţa de tensiune între intrarea (+) şi intrarea (—) să fie zero :

$$\text{potenţialul intrării} + = \text{potenţialul intrării} (-). \tag{13-10}$$

Înlocuind, obţinem :

$$V_{in} = V_{out} \cdot \frac{R_2}{R_1 + R_2} \tag{13-11}$$

$$V_{out} = V_{in} \cdot \frac{R_1 + R_2}{R_2} \tag{13-12}$$

Impedanţa de intrare, în acest caz, va fi, în principiu, infinită.

Fără a utiliza rezistenţe scumpe, de mare precizie, precizia valorii amplificării în tensiune va fi limitată la $\pm 1\%$ (pentru rezistenţe de 1%), ceea ce este echivalent cu o reprezentare numerică de 7 biţi. Dacă această precizie este satisfăcătoare, putem realiza multe operaţii complexe asupra semnalelor analogice cu ajutorul unui hardware foarte redus. Figura 13-33 ilustrează cîteva exemple.

În figura 13-33 *a* se arată cum se pot însuma mai multe semnale la intrarea amplificatorului, întrucît curenţii din rezistenţele de pe intrare se adună, suma lor trecînd prin rezistenţa de reacţie. De asemenea, valorile diferite ale rezistenţelor conducînd la un factor de scală diferit pentru curentul de intrare produs de o tensiune dată, fiecare tensiune de intrare poate fi *ponderată*. Rezultatul este echivalent cu realizarea unei *multiplicări* (printr-un factor de ponderare) pentru fiecare tensiune de intrare, după care se *adună* rezultatele. Pentru realizarea *digitală* a acestor operaţii este necesar un hardware mult mai complex, pe cînd, tehnica analogică cere doar *cîteva rezistenţe*.

Majoritatea mecanismelor de poziţionare utilizează un astfel de circuit, pentru a însuma *semnalul de comandă* cu *semnalul de reacţie* de la traductorul de viteză sau poziţie. Deoarece factorii de scală sînt determinaţi de valorile rezistenţelor, pentru ajustarea variaţiilor valorii nominale a tensiunii de la ieşirea tahometrului (sau altui traductor) se folosesc rezistenţe variabile (semireglabile). Tensiunea de comandă nu trebuie să fie întotdeauna analogică, ea poate fi furnizată direct de la un circuit digital. Prin conectarea unui semnal logic printr-o rezistenţă în *punctul de sumare în curent* (vezi fig. 13-33), cînd semnalul va fi în starea 0 logic (0V) nu se va produce nici un curent. Curentul determinat de valoarea rezistenţei se va produce atunci cînd semnalul logic va fi în starea 1. Comenzile de reglare a unei viteze mari sau mici necesită rezistenţe de valori mai mici, respectiv mai mari, în punctul de sumare. Multe convertoare D/A lucrează în acelaşi fel, folosind o combinaţie binară de valori ale rezistenţelor (de exemplu, 1 000, 2 000, 4 000, 8 000, 16 000 Ω, etc.).

Figura 13-33 *b* arată cum se poate obţine *o integrare* cu ajutorul unui amplificator operaţional.

Deoarece curentul prin condensator este

$$I = C \frac{dv}{dt} \tag{13-13}$$

viteza de variaţie (panta) tensiunii de ieşire va fi aceea cerută pentru a produce un curent de reacţie care va egala curentul de intrare. Astfel se va obţine o pantă constantă, proporţională cu tensiunea de intrare. Deoarece tensiunea la intrarea inversoare (punctul de sumare) este şi în acest caz zero, se pot folosi

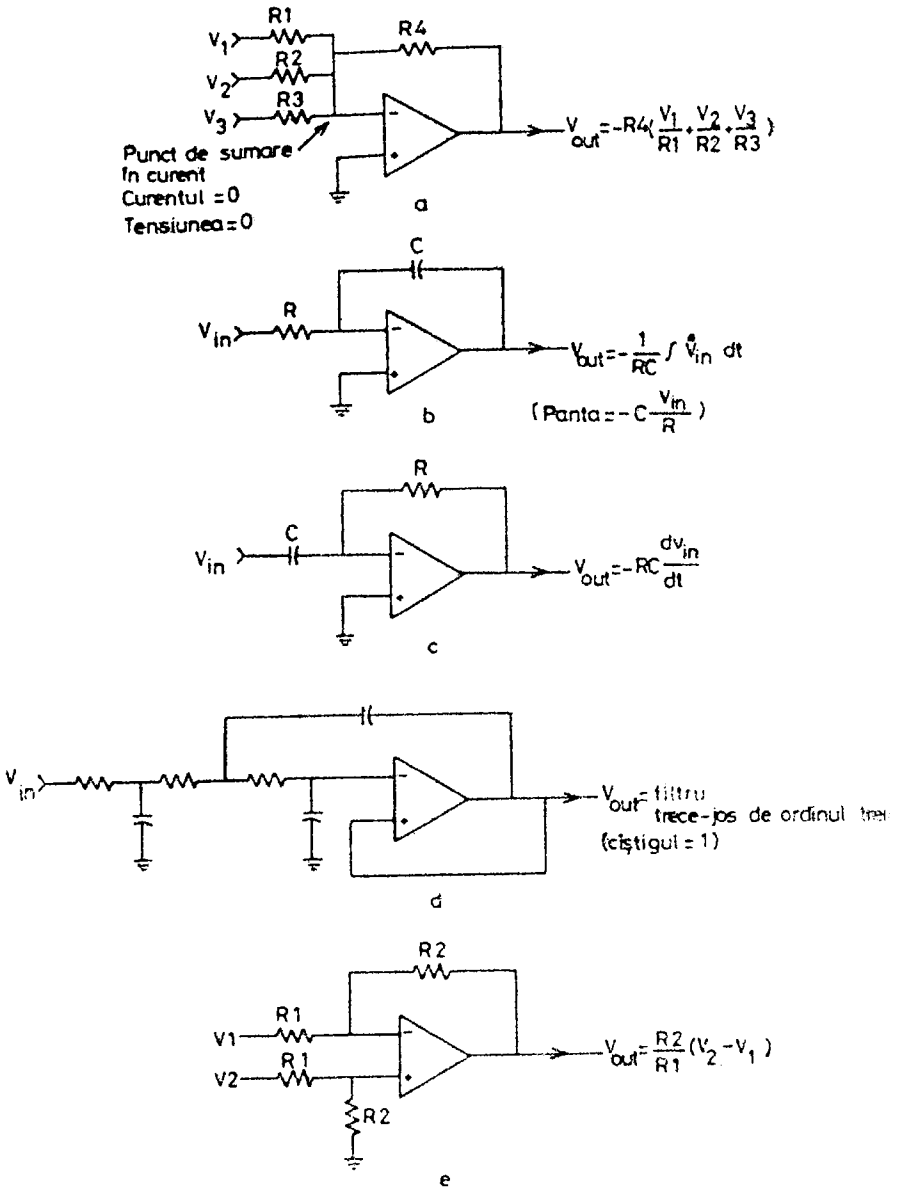


Fig. 13-33. Alte circuite cu amplificatoare operaţionale : *(a)* sumator ponderat ; *(b)* integrator ; *(c)* derivator ; *(d)* filtru activ de tip trece-jos de ordinul trei ; *(e)* amplificator diferenţial.

mai multe intrări, ca în figura 13-33*a*, iar panta semnalului rezultat la ieşire va fi egală cu *suma ponderată* a intrărilor. Pentru a realiza acest lucru digital, nu este nevoie numai de multiplicări şi adunări pentru obţinerea sumei ponderate, ci şi de acumularea sumei rezultatelor. Conversia A/D prin integrare cu dublă pantă arătată în figura 13-31 este de obicei realizată cu un amplificator operaţional integrator. Proprietatea de eliminare a erorilor caracteris-

tică acestei tehnici demonstrează că se poate obține o precizie bună folosind în locul componentelor de precizie anumite configurații „inteligente", care permit anularea erorilor.

Figura 13-33 *c* arată cum se poate obține un circuit de *diferențiere* folosind la intrare o capacitate. Întrucît curentul de intrare este proporțional cu panta semnalului, tensiunea de ieșire cerută pentru a face curentul prin R egal cu curentul de intrare va fi proporțională cu panta semnalului de intrare. Diferențierea digitală impune o memorare a valorii eșantionului de semnal anterior și o scădere.

În figura 13-33 *d* se arată cum un amplificator operațional se folosește pentru obținerea unui filtru trece-jos de ordinul trei, fără inductanțe. Prin conectarea în cascadă a două astfel de circuite, se obține un filtru de ordinul șase. Prin folosirea condensatoarelor în locul rezistențelor și invers, se obține un filtru trece-sus. Deși filtrarea poate fi realizată digital, aceasta este destul de complicată. De exemplu, pentru a obține aceleași performanțe cu acelea ale unui filtru analogic de ordinul șase, trebuie memorate și *multiplicate* cu un factor de ponderare, aproximativ nouă eșantioane digitale anterioare și apoi *sumate* pentru a obține fiecare eșantion de ieșire filtrat. Filtrele analogice sînt mai economice decît cele digitale, cu excepția cazurilor în care este necesară o tăiere foarte netă realizată cu o bună precizie.

În figura 13-33 *e* este arătată schema unui amplificator diferențial. Tensiunea de zgomot pe mod comun este anulată deoarece apare în mod egal pe cele două intrări ale amplificatorului. Dacă rapoartele de rezistențe nu sînt precis egale, atunci la ieșire apar și semnale de mod-comun.

EXERCIȚII

Se consideră un servomecanism de reglare a vitezei ce are o constantă de timp a componentelor mecanice de 50 ms iar constantele de timp electrice sînt de 2 ms și 1 ms. Care poate fi cîștigul maxim pe buclă în curent continuu dacă nu se utilizează rețele de compensare ?

2. În cazul Exercițiului 1, ce trebuie făcut pentru a se putea obține un cîștig pe buclă de 100 în curent continuu ?
Care este efectul creșterii cîștigului pe buclă în curent continuu asupra preciziei poziționării în cazul unei sarcini constante ce încarcă mecanismul.

4. Cît de rapid apreciați că este răspunsul sistemului din Exercițiul 1 în atingerea vitezei finale (aproximativ) ?
Reușește servomecanismul să împiedice variațiile de viteză datorate unor fluctuații ale sarcinii ce au o frecvență mai mare decît frecvența la care cîștigul pe buclă este unitar ?

6. Concepeți un generator numeric de secvență, similar celui prezentat în figura 13-9, pentru un motor pas cu pas cu trei înfășurări.
Proiectați un circuit logic sincron, pentru a fi utilizat cu un codificator incremental în sistemul reprezentat de figura 13-11, ce produce impulsul CHANGE avînd durata egală cu perioada impulsului de ceas, ori de cîte ori se modifică ieșirea codificatorului. Circuitul va genera, de asemenea, și impulsul RIGHT odată cu impulsul CHANGE produs de o deplasare către dreapta.

8. Proiectați un detector de fază (vezi fig. 13-13), folosind un circuit digital, pentru a produce impulsuri cu factorul de umplere mai mare sau mai mic de 50% în funcție de faza relativă a semnalului CLOCK OUTPUT ce are factorul de umplere de 50% și de semnalul logic SIGNAL INPUT ce are factorul de umplere de asemenea de 50%.

9. Proiectați codificatorul și decodificatorul logic pentru semnalul codificat în durată reprezentat în figura 13.-14.

10. Desenați schema bloc pentru servomecanismul de poziționare a capului din figurile 13-17 și 13-18. Includeți și două circuite de eșantionare și memorare (sample-and-hold), toată logica necesară generării tensiunii de comandă a vitezei și a comutării pe sistem automat de reglare a poziției.

11. Proiectați logica de generare a unui semnal cu modulație în frecvență modificată (vezi fig. 13-19).
12. Proiectați logica de decodificare a semnalului cu modulație în frecvență modificată. Presupuneți că un „detector de vîrf" generează un semnal de ieșire ori de cîte ori are loc o tranziție în *oricare* din direcții. Presupuneți că prima tranziție, după o pauză de cel puțin trei intervale de timp, este întotdeauna o tranziție în 1.
13. Proiectați circuitul similar celui din exercițiul 12 cu facilitatea de a identifica tranzițiile în 1 fără a ține cont de intervalul dintre pulsuri. Contați pe faptul că succesiunea 1, 0, 1 nu va putea fi niciodată decodificată dacă tranzițiile ceasului sînt interpretate ca treceri în 1.
14. Comparați performanțele modulației în frecvență simple și modificate (vezi fig. 13-19) prin prisma modului în care suportă variațiile de viteză cu o perioadă mai mică decît cîțiva biți.
15. Desenați schema bloc a structurii logice necesare generării semnalelelor de comandă pe coloană pentru imprimanta reprezentată în figura 13-20. Indicați capacitatea memoriei ROM utilizate.
16. Desenați schema bloc a sistemului logic ce generează semnalul video și semnalele de sincronizare pe orizontală și verticală pentru un sistem de afișaj pe tub catodic (CRT), ce folosește matrici de caractere de 5×7, reprezentate în figura 13-2t. Presupuneți că un număr de 16×32 de caractere sînt stocate într-o memorie RAM, iar ecranul are 256 de linii ce trebuie baleiate de 60 de ori pe secundă fără baleiaj întrețesut.
17. Proiectați o tastatură de tipul aceleia din figura 13-23 la care sînt eliminate semnalizările parazite datorate contactelor imperfecte realizate de comutatoare prin blocarea baleierii matricii de comutatoare timp de 20 ms sau pînă cînd tasta este eliberată, indiferent care tastă a fost apăsată. Generați la ieșire un impuls de strob, cu lungimea unei perioade de ceas, pentru fiecare tastă apăsată.
18. Presupunînd că figura 13-24 reprezintă o porțiune dintr-o matrice de 8×8, ca alte taste vor apare ca fiind apăsate ca urmare a căi false de semnalizare ce apar.
19. Care va fi efectul utilizării matricii din figura 13-25, la aprinderea unor becuri ?
20. Care va fi procentul de distorsiuni în semnalul reconstruit dacă componentelor cu frecvență mai mare decît jumătatea frecvenței de eșantilonare le corespunde jumătate din energia întregului semnal ?
21. Desenați forma de undă a zgomotului pentru ieșirea nefiltrată din figura 13-26 *b*. (Menționăm că zgomotul este diferența dintre semnalul de ieșire obținut și cel corect).
22. Construiți un tabel similar tabelului 13-1 pentru o tensiune de intrare de 5 mV.
23. Combinați schemele din figurile 13-29 și 13-30 și adăugați detaliile necesare ce permit sistemului astfel obținut să eșantioneze ciclic continuu opt intrări analogice, generînd un semnal de sincronizare de cîte ori NUMĂRĂTORUL DE INTERVALE și un latch adresabil conțin numărul intrării și valoarea semnalului de pe această intrare.
24. (*a*) Proiectați un convertor D/A de opt biți folosind numărătoare digitale pentru a produce o secvență cu un factor de umplere proporțional cu numărul, prin integrarea căreia să se obțină o ieșire analogică ce variază între 0 V și 5 V
(*b*) Ce valoare trebuie să aibă constanta de timp a integratorului astfel încît ondulația maximă să fie jumătate din rezoluția convertorului ?
(*c*) În cît timp variază semnalul de ieșire de la valoarea minimă la cea maximă, dacă la intrare se aplică un semnal treaptă.
25. (*a*) Proiectați un amplificator inversor cu o impedanță de intrare de 2 000 Ω și un cîștig de 50.
(*b*) Care este *cîștigul pe buclă* dacă cîștigul amplificatorului în buclă deschisă este de 10 000 ?
26. Proiectați un amplificator neinversor cu cîștigul unitar fără a folosi nici o rezistență.
27. Amplificatoarele operaționale din figurile 13-32 și 13-33 au toate produsul amplificare-bandă de 1 MHz și sînt compensate intern fiind necondiționat stabile și pentru valori unitare ale cîștigului.
(*a*) La ce frecvență trebuie să înceapă să scadă cîștigul cu bucla deschisă dacă cîștigul în curent continuu al amplificatorului este de 100 000 ?
(*b*) În cazul amplificatorului din Problema 25, cît de mare poate fi această frecvență fără apariția riscului instabilității ?
28. Proiectați un convertor D/A de șase biți folosind circuitul din figura 13-33 *a* cu o rețea de rezistențe ponderată binar.
29. Ce valoare trebuie să aibă rezistența plasată în serie cu intrarea neinversoare în cazul amplificatorului din Problema 25 pentru a se elimina efectul *curenților de polarizare* ai intrărilor presupuși egali ?

30. (*a*) Proiectați un convertor A/D cu integrator cu dublă pantă (figura 13-31) folosind comutatorul analogic cuadruplu CD4066 realizat în tehnologie CMOS pentru a selecta curenții de intrare. (Acest dispozitiv se comportă ca un „releu" analogic — comandat de un semnal logic — ce realizează un contact cu o „rezistență de contact" de aproximativ 100 Ω. Presupuneți disponibilă o sursă standard de precizie de 3,6 V. Semnalul la intrare care trebuie convertit variază de la 0 V la 5 V.
(*b*) Proiectați un convertor, realizat pe același principiu, pentru a converti, prin baleiere circulară, semnalele disponibile la șapte intrări, folosind circuitul CD 4051 (multiplexor analogic de opt căi) pentru a selecta intrările și referința de curent. Un *semnal de sincronizare* va indica momentul în care numărătorul conține starea corespunzătoare mărimii analogice de pe o intrare.

REFERINȚE

1. T. K. Hemingway, *Electronic Designers Handbook*, Business Publications, London, 1965.
2. Lawrence Rabiner et al., „An Approach to the Approximation Problem for Nonrecursive Digital Filters", *IEEE Transactions on Audio and Electroacustics*, June 1970, pag. 83—106.

BIBLIOGRAFIE

Control prin reacție

Fitzgerald, Tom, „Poles and Zeroes in Peripherals", *Computer Design*, December 1968, pag. 36—43.

Hemingway, T. K., *Electronic Designers Handbook*, Business Publications London, (Include grafice care prezintă răspunsul a 11 rețele corectoare de fază).

Van Allen, Leland, „D. C. Motor Control Circuit Cancels Armature Resistance", *Electronics*, April 12, 1973, pag. 104, (conține o descripție mai detaliată a circuitului din figura 13-6).

Dispozitive periferice

Barton, David, „Why Multiplex LED's" ? *IEEE Spectrum*, November 1972, pag. 30—32.

Flores, Ivan, *Peripheral Devices*, Prentice-Hall, Englewood Cliffs, N. J., 1973.

Kaye, David, „Pulse Width and Phase Enconding Compete to be Cassette Standard," *Electronic Design*, November 9, 1972, pag. 32.

Ohm, William, „Reel-to-Reel Drive Design for a Cassette Recorder", *Computer Design*, August 1973, pag. 67—70 (descrie sistemele automate de reglaj a vitezei care utilizează motoare).

Peatman, John, *The Design of Digital Systems*, McGraw-Hill, New York, 1972, Capitolul 7.

Riley, Wallace, „The Technology Gap Starts to Close for Peripherals", *Electronics*, July 3, 1972, pag. 59—74.

Sidhu, Pawitter S., „Group-Coded Recording Realiability Doubles Diskette Capacity", *Computer Design*, December 1976, pag. 84—88.

Amplificatoare operaționale

Burr Brown Staff, *Operational Amplifiers: Design and Applications*, McGraw-Hill, New York, 1971.

National Semiconductor Staff, *Linear Applications*, National Semiconductor, Santa Clara, Calif., 1973.

Smith, John I., *Modern Operational Circuit Design*, Wiley, New York, 1971.

Brokaw, Paul, „An IC Amplifier Users Guide to Decoupling and Grounding", *Electronic, Products*, December 1977, pag. 45—53.

Analog/Digital

Mayo, J. S., „Experimental 244 MC/s PCM Terminals," *Bell System Technical Journal*, November 1965.

Strong, Norman, „LSI Converts an Old Technique into Low-Cost-A-D Conversion", *Electronics*, September 11, 1972, pag. 102—105.

Filtre

Al-Nasser, Farouk, „Tables Shorten Design Time for Active Filters", *Electronics*, October 23, 1972, pag. 113—118.

Brokaw, Paul, „Simplify 3 Pole Active Filter Design", *EDN*, December 15, 1970, pag. 23—28.

Rabiner, Lawrence, et al., „An Approach to the Approximation Problem for Nonrecursive Digital Filters, „*IEEE Transactions on Audio and Electroacustics*, June 1970, pag. 83—106.

Tufts, Donald W. and D. W. Rorabacher, „Designing Simple, Effective Digital Filters", *IEEE Transactions on Audio and Electroacustics*, June 1970, pag. 142—158.

14

UTILIZAREA STATISTICII în proiectarea digitală

14.1. FIABILITATEA SISTEMELOR

Metodele de prezicere — prin utilizarea statisticii — a fiabilității sistemelor digitale au evoluat în anii din urmă pînă la nivelul la care echipamentul și componentele pot fi livrate garantîndu-se valoarea **mediei timpului de bună funcționare (MTBF)***. Ca urmare defectarea unui echipament se poate prezice matematic, cu o bună aproximație. Pentru inginerul proiectant este important să cunoască cel puțin bazele matematice ale fiabilității, deoarece fiabilitatea constituie întotdeauna un obiectiv important al proiectării. Costul determinat de repararea defecțiunilor a crescut cel puțin tot atît de repede pe cît a scăzut costul CI, astfel că fiabilitatea poate să fie cel mai important obiectiv al proiectării — chiar și în cazul dispozitivelor ieftine din bunurile de larg consum.

Spre deosebire de componentele mecanice, componentele electronice moderne nu se uzează practic niciodată. Defectările pot fi prezise numai pe baze statistice. De fapt pentru sistemele nou proiectate rata de defectare este determinată în mod obișnuit de *defectările timpurii*, care pot fi eliminate în sistemele realizate ulterior prin schimbarea procedurilor de fabricație, pe baza acumulării de experiență. Cauzele tipice care determină defectările inițiale sînt date de erorile de proiectare, de erorile de fabricație, de factorii adverși din timpul transportului și manipulării, de erorile de testare și depanare, de procedurile improprii de operare și întreținere și de deficiențele componentelor. După cum rezultă și din figura 14-1, rata de defectare a unui sistem se îmbunătățește odată cu fabricarea fiecărui nou exemplar al sistemului.

Tot din figura 14-1 rezultă că inițial toate sistemele au o rată de defectare mai mare. De multe ori nivelul de putere, de temperatură sau de vibrație este mărită** în mod intenționat în timpul unei *perioade inițiale de îmbătrînire (rodaj)**** pentru a se favoriza defectarea componentelor slabe. Această tehnică, și de fapt toate prezicerile statistice ale defectărilor, se bazează pe *presupunerea că orice defectare este un rezultat al unei solicitări care depășește capacitatea de rezistență a unei componente oarecari*. Dacă solicitarea variază aleator în timp (ca în fig. 14-2) timpul *mediu* necesar pentru ca solicitarea respectivă să determine o defectare (*MTBF*) va depinde de solicitarea aplicată

---

* Prescurtarea *MTBF* reprezintă inițialele cuvintelor englezești „*Mean Time Between Failure*" (n.t.).

** Acest tip de încercări este cunoscut sub numele de *încercări accelerate* (n.t.).

*** Această perioadă este denumită în mod curent în literatura de limbă engleză „*burn-in*" (n.t.).

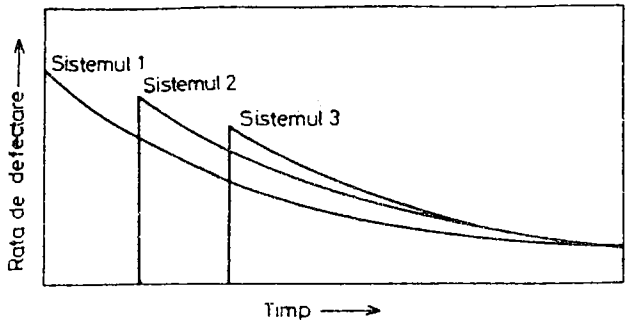


Fig. 14-1. Îmbunătățirea fiabilității ca urmare a creșterii experienței fabricantului.

elementului respectiv. Dacă, de exemplu, solicitarea este o vibrație iar defectarea este dată de întreruperea unei legături electrice în CI, testele în care se aplică în mod intenționat șocuri vor putea pune în evidență această problemă într-un timp foarte scurt. Dacă solicitarea este o tensiune sau o temperatură, testarea în condiții de tensiune și/sau temperatură crescute va face ca problema să apară mai devreme.

Deși modelul de defectare din figura 14-2 nu poate fi luat ad litteram rezultatele matematice care se obțin prin utilizarea sa par să corespundă destul de bine cu rezultatele experimentale reale. De exemplu, experimentările au pus în evidență următoarele relații aproximative care există între solicitarea aplicată și rata de defectare ($1/MTBF$):

1. Rata de defectare pentru rezistoarele cu strat de carbon se dublează pentru fiecare creștere cu 20% a puterii sau pentru fiecare creștere cu 38°C a temperaturii ambiante.

2. Rata de defectare a capacitoarelor cu tantal, cu electrolit solid, se dublează pentru fiecare creștere cu 15% a tensiunii aplicate.

3. Rata de defectare a tranzistoarelor de putere se dublează pentru fiecare creștere cu 10°C a temperaturii joncțiunii și pentru fiecare creștere de 7,5 V a tensiunii aplicate.

Prin urmare proiectantul poate să realizeze o proiectare care să conțină din start elementele care asigură o fiabilitate îmbunătățită prin utilizarea tuturor componentelor mult sub valorile lor maxime garantate de producător. Deoarece fiabilitatea tuturor componentelor este afectată de temperatură, ea poate fi îmbunătățită în mod semnificativ prin răcire forțată cu aer. În mod ideal sistemul trebuie să poată funcționa fără ventilatoare astfel ca rata de defectare a ventilatoarelor să nu îi mai afecteze siguranța în funcționare.

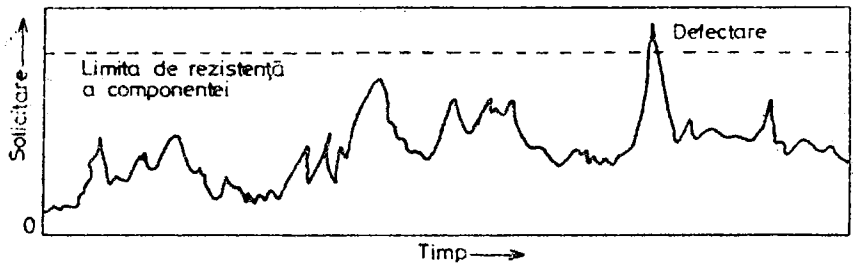


Fig. 14-2. Modelul teoretic de defectare a unei componente.

14.2. CALCULUL MEDIEI TIMPULUI DE BUNĂ FUNCȚIONARE

Pentru a calcula *MTBF* pentru un sistem dat este necesar să cunoaștem rata de defectare ($1/MTBF$) a fiecărei componente a sistemului. Presupunînd că defectarea oricărei componente determină defectarea sistemului, *rata de defectare a sistemului va fi egală cu suma ratelor de defectare ale componentelor sale:*

$$\lambda_{sistem} = \lambda_1 + \lambda_2 + \lambda_3 + \ldots + \lambda_n \qquad (14\text{-}1)$$

unde s-a notat

$$\lambda = 1/MTBF. \qquad (14\text{-}2)$$

Rata de defectare se exprimă în mod obișnuit în *procente pe 1 000 de ore* sau în număr de defectări la un milion de ore. Lucrul cel mai greu de determinat în calculul ratei de defectare a unui sistem este rata reală de defectare a elementelor utilizate; de exemplu rata de defectare a CI poate varia de la 0,00019% /1 000 h pentru circuitele de mare fiabilitate la 0,01% /1 000 h pentru circuitele standard încapsulate în plastic. Circuitele fabricate prin procese noi pot avea o rată de defectare mai ridicată datorită unor probleme necunoscute legate de proces.

Din cauza valorii foarte mici a ratei de defectare a componentelor electronice moderne ea este dificil de stabilit cu suficientă acuratețe. În conformitate cu figura 14-3, rezultă că estimarea pe baza unei testări care conduce numai la cîteva defectări poate să ducă la rezultate incorecte. De exemplu, dacă după 1 000 de ore de funcționare din 100 de circuite integrate s-au defectat numai două, valoarea estimată a ratei de defectare este de $100 \times 1\,000/2 = 50\,000$ h. Referindu-ne la figura 14-3, constatăm însă că pentru un nivel de încredere de 60% valoarea reală a ratei de defectare se poate plasa undeva în intervalul $2{,}5 \times 50\,000 = 125\,000$ h și $0{,}7 \times 50\,000 = 35\,000$ h. Dacă

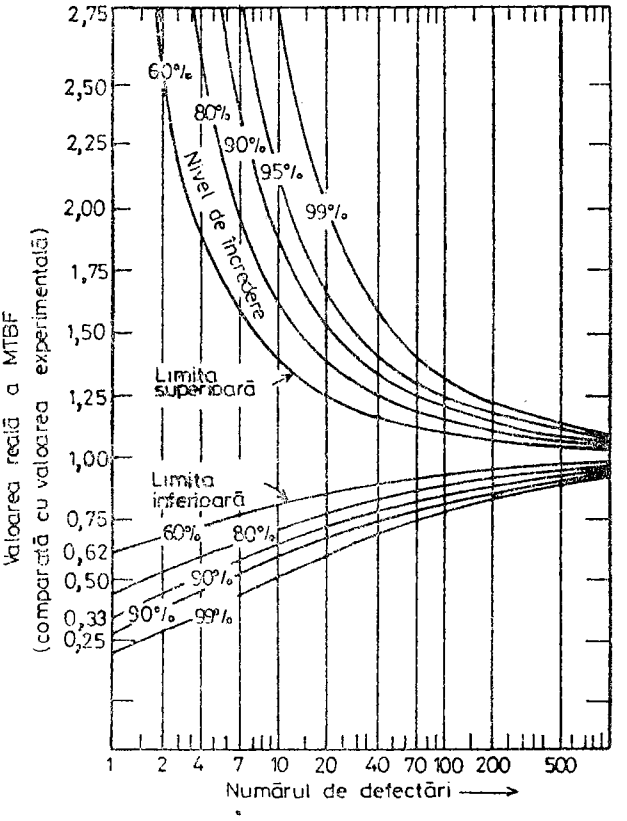


Fig. 14-3. Acuratețea estimării MTBF în funcție de numărul de defectări observate.

aşteptăm pînă se adună mai multe defecte, putem ajunge să avem o idee mult mai exactă asupra valorii reale a ratei de defectare. Totuşi chiar după acumularea a 10 defecte, pentru un nivel de încredere de 90%, intervalul în care se plasează valoarea reală a ratei de defectare este încă de +180%... —38% din valoarea estimată.

În cazul CI de mare fiabilitate este complet nepractic să testăm un număr suficient de circuite pentru un timp suficient de lung astfel încît să putem obţine cele 10 sau 20 de defectări necesare pentru o determinare rezonabil de precisă a ratei de defectare. De exemplu, dacă circuitele au o rată de defectare de 0,001%/1 000 h, vor fi necesare 10^9 circuit-oră pentru a rezulta 10 defectări. Chiar dacă s-ar testa simultan 1 000 de circuite pentru a rezulta 10 defectări va fi nevoie de o testare de 10^6 ore sau *114 ani*.

Pentru a se putea obţine date despre fiabilitate a devenit necesar să se utilizeze *testarea accelerată*. Prin testarea la temperaturi mai mari, toate reacţiile activate termic care pot determina defectări îşi măresc viteza; rata de defectare creşte cu un factor care poate fi calculat. Rata de defectare la temperatura normală se calculează din rata de defectare determinată la temperaturi ridicate prin aplicarea unui factor de corecţie. De exemplu testarea la 125°C „accelerează" rata de defectare cu un factor de 100 faţă de situaţia de la 55°C. Cei 114 ani de testare necesari în exemplul anterior se produc în aceste condiţii la aproximativ 1 an. Deşi majoritatea problemelor legate de proces întîlnite pînă acum în fabricarea CI au fost de tipul acelora care sînt accelerate de temperatură, datele experimentale obţinute prin testări accelerate sînt însoţite totuşi de un uşor risc. Pe măsură ce timpul trece, precizia datelor de fiabilitate de care dispunem se îmbunătăţeşte progresiv.

Tabele cu ratele de defectare pentru diverse componente sînt furnizate de diferite surse. Probabil că sursa cea mai detaliată este îndrumarul MIL-HDBK-217B care include informaţii care permit să se ţină seama de efectele de temperatură, de condiţiile de mediu ambiant şi de calitatea dispozitivului. În Tabelul 14-1 se dau valorile tipice ale ratelor de defectare pentru

Tabelul 14-1. **Ratele de defectare ale componentelor**

Componenta	Rata de defectare (%/1000 h)
Circuit integrat SSI, MSI şi LSI	0,01
Circuit integrat de mare siguranţă în funcţionare	0,001
Tranzistor (de semnal mic)	0,005
Tranzistor de putere	0,05
Diodă (de semnal mic)	0,002
Diodă de putere	0,02
Rezistor (cu strat de carbon)	0,001
Rezistor de putere (bobinate)	0,05
Rezistor variabil (cermet)	0,2
Capacitor (polistiren)	0,01
Capacitor (Mylar)	0,05
Capacitor cu tantal (cu electrolit solid)	0,02
Capacitor electrolitic (cu aluminiu)	0,2
Cristal din cuarţ	0,05
Contact de conector	0,005
Contact de conector realizat prin cablaj imprimat	0,01
Contact de întrerupător (putere mică)	0,02
Releu (încapsulat ermetic, putere mică)	0,05
Ventilator (tip Rotron)	2,0
Transformator (de putere)	0,5
Bobină	0,03
Conexiune cu sîrmă înfăşurată (wire-wrap)	0,00001
Conexiune realizată prin lipire în baie	0,0001

componente de uz general, de calitate ridicată care funcționează în condiții de mediu de laborator cu o încărcare mai mică de 50% față de valoarea nominală a curentului, puterii etc.

Deși în primele ediții ale MIL-HDBK-217B se arăta că rata de defectare a CI crește odată cu creșterea complexității, versiunea cea mai recentă a acestui îndrumar omite factorul determinat de complexitatea circuitului. Cu toate că posibilitatea de defectare datorată utilizării incorecte a CI scade la circuitele LSI, procentul de defectări datorate numai cipului crește. Pentru circuitele SSI, diode, tranzistoare *un procent substanțial din defectările în exploatare este determinat de utilizarea incorectă și de erorile de proiectare.* Întrebuințarea circuitelor LSI — care este mult mai puțin afectată de aceste erori — reduce în mod major această sursă de defectări. Testări extensive asupra unor registre de deplasare de 1 024 biți și a unor memorii RAM au arătat că rata de defectare atinge un nivel de 0,01%. Obiectivul de proiectare, explicitat în capitolele precedente, de a reduce la minimum numărul de capsule prin utilizarea celui mai ridicat nivel de integrare posibil, va duce deci și la obținerea unui nivel maxim al fiabilității.

În unele aplicații, în care este necesară o fiabilitate foarte ridicată se dublează o mare parte din hardware (sau chiar în totalitate). De exemplu, se utilizează adeseori două unități centrale și mijloace de testare automată care realizează comutarea de la una la alta în cazul unei defecțiuni. Deoarece în acest caz sistemul se va defecta numai dacă cel de al doilea procesor se defectează *în timpul reparației* primului procesor, se vede că — teoretic — se poate atinge o fiabilitate aproape perfectă. Dacă presupunem că *timpul mediu pentru repararea procesorului* (*MTTR*)* este de o oră și că ambele procesoare au *MTBF* de 1 000 h, rezultă că fiecare procesor nu va fi disponibil 0,1% din timp. Probabilitatea ca ambele procesoare să fie defecte simultan este deci de $0{,}1\% \times 0{,}1\% = 10^{-6}$, ceea ce este echivalent cu *MTBF* pentru *ambele* procesoare de 10^6 1 ore sau *114 ani.*

14.3. FUNCȚIA DE FIABILITATE

În cazul în care un număr mare de evenimente se produc cu o rată medie constantă, *m*, probabilitatea de apariție a *n* evenimente (P_n) este dată de o *distribuție Poisson :*

$$P_n = \frac{m^n e^{-m}}{n!}. \qquad (14\text{-}3)$$

Dacă presupunem că vîrfurile — care determină defectări — ale nivelului de solicitare din Figura 14-2 au o distribuție Poisson, numărul mediu de defectări într-un interval de timp dat, *T*, va fi $m = T/MTBF$. Probabilitatea ca în intervalul *T să nu existe defectări* ($n = 0$) se obține din expresia 14-3 :

$$P_0 = \frac{(T/MTBF)0\ e^{-T/MTBF}}{0!},$$

$$P_0 = e^{-T/MTBF}.$$

* Prescurtarea *MTTR* reprezintă inițialele cuvintelor englezești „*Mean Time To Repair*" (n.t.).

Această mărime este denumită funcția de fiabilitate, R :

$$R = e^{-T/MTBF} \tag{14-4}$$

Aplicarea acestei relații în cazurile concrete poate duce la obținerea unor rezultate oarecum surprinzătoare. De exemplu, deși *MTBF* este media timpului de bună funcționare, probabilitatea ca într-un interval de timp *egal cu MTBF* să nu apară defectări va fi de numai

$$R = e^{-1} = 0{,}367. \tag{14-5}$$

Probabilitatea de a nu avea defectări într-un interval de timp egal cu jumătate din *MTBF* este

$$R = e^{-1/2} = 0{,}607. \tag{14-6}$$

Expresia 14-4 se poate rescrie:

$$T/MTBF = -\ln R. \tag{14-7}$$

Utilizînd această relație, găsim că obținerea succesului (funcționare fără defectări) cu o probabilitate de 90% impune limitarea duratei de utilizare a sistemului la 10,5% din *MTBF* [—ln (0,9) = 0,105]. Natura exponențială a funcției de fiabilitate face ca atingerea, valorii foarte ridicate a fiabilității cerute de exemplu de misiunile spațiale să fie într-adevăr foarte dificilă. De exemplu, pentru o fiabilitate de 99%, valoarea *MTBF* trebuie să fie de 100 de ori mai mare ca durata misiunii (—ln 0,99 = 0,01) !

Dacă succesul unei misiuni depinde și de funcționarea fără defectări a încă unui alt sistem, fiabilitatea care se obține este dată de produsul fiabilităților fiecărui sistem. Dacă funcționarea globală depinde de două sisteme, fiecare avînd o fiabilitate de 90%, fiabilitatea totală va fi de $0{,}9 \times 0{,}9 = 81\%$. Uneori îmbunătățirea fiabilității se obține prin duplicarea sistemelor. În acest caz fiabilitatea unui sistem este dată de

$$R_{sistem} = R_1 + R_2 - (R_1 R_2). \tag{14-8}$$

De observat că această relație presupune că repararea *nu* este posibilă în timpul misiunii.

Abordarea statistică se poate utiliza și pentru reducerea costului determinat de controlul de recepție al elementelor sistemului. Este o naivitate să presupunem că un fabricant va livra întotdeauna produse care satisfac toate specificațiile numai pentru că în contract se specifică acest lucru. O metodă de a menține onestitatea furnizorului este aceea de a testa *toate* produsele pe care ni le-a livrat. Deoarece această metodă conduce la o creștere foarte mare a efortului, în mod obișnuit se testează numai eșantioane *prelevate statistic*. Distribuția Poisson se poate utiliza pentru elaborarea unor planuri de control care specifică numărul minim de piese care trebuie testate pentru a realiza cu un risc rezonabil valoarea dorită a nivelului de calitate acceptabil (*AQL*)*.

Atunci cînd se lucrează cu un furnizor nou, testările trebuie să fie făcute cu mai multă grijă, în comparație cu cazul produselor unui furnizor pentru care fiabilitatea produselor este deja verificată și cunoscută. Desigur că o testare de 100% ne oferă certitudinea că procentul de defecte determinat experimental nu depășește *procentul de defecte tolerat* pentru lotul respectiv de produse. De obicei se testează însă numai o parte a unui lot mare, obținîndu-se

* *AQL* reprezintă inițialele cuvintelor englezești „*Acceptable Quality Level*" (n.t.).

chiar în condițiile unei astfel de testări parțiale un nivel de semnificație rezonabil, C. Pentru un număr de defectări, n, nivelul de semnificație — în conformitate cu relația 14-3 — este dat de $1 - P_n$. Deoarece în relația 14-3 mărimea m este dată de raportul dintre procentul de defecte (PD) și nivelul de calitate acceptabil (AQL) se obține

$$1 - C_n = P_n = \frac{(PD/AQL)^n \times e^{-PD/AQL}}{n!} . \qquad (14\text{-}9)$$

14.4. TESTAREA ȘI ÎMBĂTRÎNIREA COMPONENTELOR

Creșterea costului manoperei și scăderea costului componentelor ne-au adus în situația în care costul aferent punerii în funcțiune și asigurării de service în primul an poate depăși costul CI (vezi Tabelul 2-1). O soluție pentru această problemă constă în utilizarea unor componente care au fost supuse unui program de teste speciale de fiabilitate și îmbătrînire. După cum se poate vedea din Figura 14-4 aceste măsuri pot reduce cu un ordin de mărime procentul de CI necorespunzătoare care sînt asamblate într-un echipament, iar rata de defectare în primul an cu un factor de 4.

În mod obișnuit CI de uz general*, ieftine, sînt furnizate cu un *nivel de calitate acceptabil* (AQL) de 1%. Aceasta înseamnă că pînă la 1% din piesele primite pot fi defecte. Dacă pe o placă de cablaj imprimat sînt asamblate împreună 100 de astfel de circuite, *probabilitatea ca cel puțin unul din CI de pe placă să nu fie bun este de 63% !* Dacă procentul de CI necorespunzătoare poate fi redus la 0,1% probabilitatea ca unul din circuitele de pe placă să nu fie bun scade la 10%. Un program special de testare și îmbătrînire poate reduce

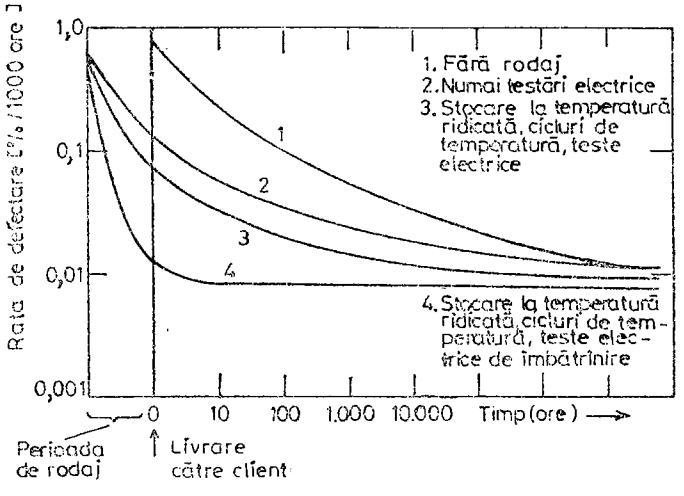


Fig. 14-4. Îmbunătățirea ratei defectărilor inițiale a CI de mare fiabilitate (Reliability Inc., Houston, Texas). Datele se bazează pe informațiile de la șase clienți, pentru un milion de circuite.

* *Commercial grade* în literatura de limbă engleză (n.t.).

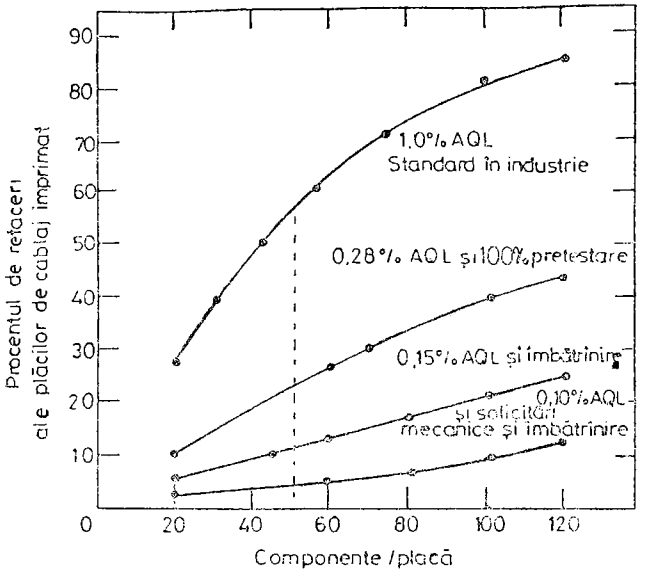


Fig. 14-5. Dependența numărului de refaceri în funcție de mărimea plăcii de cablaj imprimat și de calitatea CI.

costul de refacere cu un factor de 6. În figura 14-5 se prezintă dependența procentului de plăci care trebuie refăcute în funcție de mărimea plăcii și de valoarea nivelului de calitate acceptabil.

Majoritatea fabricanților oferă CI speciale, de mare fiabilitate suplimentul de preț fiind mic. Economia pe care o realizăm în ceea ce privește verificarea funcționării și lucrările de refacere acoperă de multe ori acest supliment de preț. Creșterea fiabilității și costul de service mai mic în primul an oferă și o economie suplimentară.

Pentru a obține nivelul de semnificație dorit se poate calcula raportul PD/AQL pentru fiecare valoare a lui n și se elaborează un *plan de control* în concordanță cu valoarea dorită a AQL. De exemplu, pentru un nivel de semnificație de 90% și un AQL de 1% se obține planul de control din Tabelul 14-2. Dacă au fost testate 231 de piese, fără a găsi una defectă, se poate accepta cu o probabilitate de 90% că întregul lot, indiferent de mărimea sa, realizează specificația de AQL de 1%. Dacă se găsește o piesă defectă înainte de cea de-a 231 piesă testată, testarea continuă pînă la cea de-a 390 piesă dacă nu se mai găsește o altă piesă defectă. Lotul este acceptat dacă numărul de piese defecte nu este mai mare decît numărul de acceptare indicat în tabel, corespunzător cantității de piese prelevate din lot.

Tabelul 14-2. **Planul de control (AQL 1%, nivel de semnificație 90%)**

Numărul de piese defecte	0	1	2	3	4	5	6	7	etc.
Cantitatea de piese prelevată din lot	231	390	533	668	798	927	1 054	1 178	

Pentru a testa la un *AQL* de 0,01% cantitatea de piese care trebuie prelevată din lot va fi de 10 ori mai mare decît aceea precizată în Tabelul 14-2, astfel că pentru a accepta lotul va fi necesar să fie testate cel puțin 2 310 piese. Dacă se acceptă o mărire a riscului se obține micșorarea numărului de piese care trebuie prelevate, dar deoarece 0,1% înseamnă o piesă defectă la 1 000 de piese, vom atinge pentru numărul minim de piese prelevate cifra de 1 000.

14.5. UTILIZAREA PARTAJATĂ A INSTALAȚIILOR

Distribuția Poisson este utilă proiectanților de sisteme digitale nu numai pentru calculele de fiabilitate. Această distribuție poate fi utilizată și pentru a economisi „hardware" în cazul în care un sistem trebuie să servească un număr mare de „clienți" independenți, relativ inactivi. O centrală telefonică constituie un exemplu clasic pentru acest tip de problemă care, de altfel, se regăsește în multe sisteme care lucrează cu divizare în timp în care un procesor central este cuplat la un număr mare de *terminale* utilizate pentru colectarea informației, introducere de date etc. Deși din punct de vedere teoretic toate terminalele pot fi utilizate simultan, utilizarea lor medie este însă mult mai mică. Prin folosirea statisticii, instalațiile sistemului pot fi utilizate de un număr mare de clienți, supraîncărcarea sistemului fiind însă menținută la un nivel acceptabil.

De exemplu, într-o centrală telefonică, numărul de căi de transmisie (căi vocale) între două orașe este astfel calculat încît în timpul perioadei de vîrf să rezulte o probabilitate de *blocare* (nici o cale nu este disponibilă) de aproximativ 1/100, sau mai mică. Dacă nu s-ar accepta posibilitatea de blocare, pentru fiecare telefon ar fi necesară o cale de transmisie corespunzînd posibilității teoretice ca toate persoanele dintr-un oraș să cheme simultan toate persoanele din celălalt oraș. Costul unui astfel de sistem este în afara oricărei discuții, deoarece ar fi necesare mii de căi de transmisie în locul uneia singure.

Într-un sistem de calculatoare cu divizare în timp există o problemă similară. Dacă, de exemplu, un sistem de verificare a creditului unui client trebuie să manipuleze cereri de la 100 de terminale răspîndite printr-un mare magazin am putea să asigurăm la intrare 100 de memorii tampon pentru a acumula 100 de cereri simultane, caracterizate de 10 cifre (40 de biți). În acest caz procesorul central trebuie să fie capabil să prelucreze în mod simultan la viteza maximă, cererile de la toate canalele.

O abordare mult mai rezonabilă constă în a prevedea memorii tampon și capacități de prelucrare corespunzătoare unei valori a ratei de apariție a cererilor care se va produce numai din cînd în cînd. Atunci cînd este depășită această valoare a ratei de apariție a cererilor se dă un semnal de ocupat sau eventual nu se răspunde, situație acceptabilă cu condiția ca să se producă rareori. Problema este deci de a se determina capacitatea care trebuie asigurată pentru o valoare dată a probabilității de a obține un semnal de ocupat (sau de a nu obține răspuns).

Și pentru acest tip de probleme — la fel ca și în cazul problemelor de fiabilitate — se poate aplica expresia 14-3. Dacă m este numărul *mediu* de terminale ocupate, atunci P_n va fi probabilitatea ca n terminale să fie ocupate la orice moment de timp:

$$P_n = \frac{m^n e^{-m}}{n!}. \qquad (14\text{-}3)$$

Dacă asigurăm facilități de prelucrare pentru un număr de k terminale simultan, *sistemul va fi ocupat ori de câte ori* $n > k$. Probabilitatea de a fi ocupat este deci:

$$P = \sum_{n=k+1}^{\infty} P_n = \sum_{n=k+1}^{\infty} \frac{m^n e^{-m}}{n!}, \qquad (14\text{-}10)$$

expresie care este denumită *sumarea exponențială Poisson*. Deoarece probabilitatea scade repede odată cu creșterea lui n, este de obicei suficient să se calculeze numai P_{k+1} și eventual P_{k+2}. Se poate obține o soluție exactă prin calcularea probabilității de a *nu* fi ocupat și scăderea valorii obținute din unitate:

$$P = 1 - \sum_{n=0}^{k} \frac{m^n e^{-m}}{n!}. \qquad (14\text{-}11)$$

În figura 14-6 se prezintă dependența dintre numărul mediu de circuite necesar și numărul de facilități de prelucrare pentru k solicitări simultane pentru diferite valori ale probabilității de a fi ocupat. Să presupunem, de exemplu, că pentru sistemul de verificare a creditului menționat anterior pot

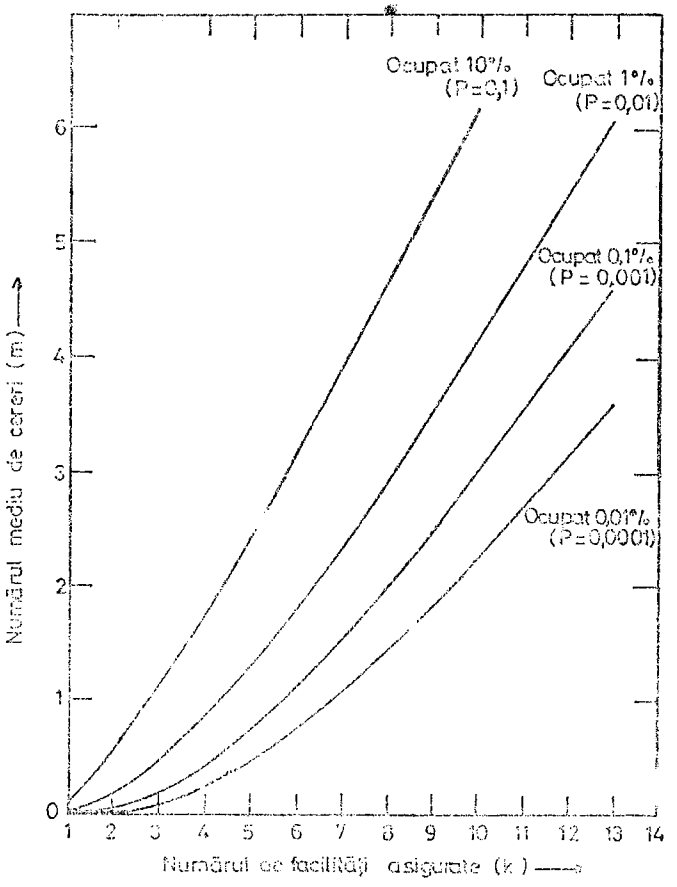


Fig. 14-6. Probabilitatea de depășire a capacității (ocupat).

exista în medie două cereri simultane de la un număr de 100 de terminale. Dacă asigurăm facilități (memorii tampon și capacitate de prelucrare în procesor) pentru a manipula șapte cereri simultane, din figura 14-6 rezultă că semnalul de ocupat va apare numai pentru 0,1% din timp, sau odată la 1 000 de chemări. Sistemul va avea deci o capacitate de 7/100 deci, 1/14 din capacitatea care ar corespunde acoperirii totale a situației *cele mai* defavorabile, asigurînd totuși un serviciu excelent.

Curbarea liniilor din figura 14-6 descrie ineficiența abordării statistice în cazul în care numărul de facilități de prelucrare este mic. Situația este analogă cu aceea care apare la determinarea cu precizie a ratei de defectare pornind de la un număr mic de defectări. În figura 14-7 se indică dependența eficienței abordării statistice în funcție de numărul de facilități de prelucrare pentru diverse valori ale probabilității de a fi ocupat. Pentru numere foarte mari toate curbele tind spre 100%, ceea ce înseamnă de fapt că fluctuațiile aleatoare sînt mediate atît de bine încît sarcina apare ca și cum ar fi continuă. De exemplu, o eficiență de 10% înseamnă că fiecare facilitate este utilizată în medie numai 10% din timp. De observat aspectul curbelor din figura 14-7 : pentru fiecare valoare a probabilității de a fi ocupat există „un punct

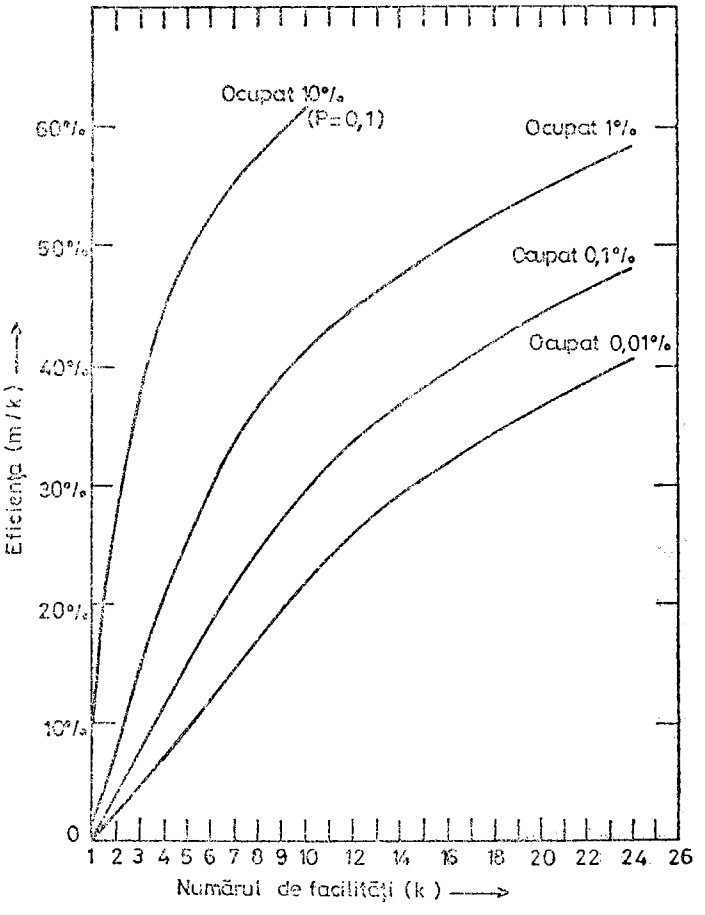


Fig. 14-7. Eficiența utilizării unor facilități de prelucrare distribuite.

de frîngere" sub care eficiența scade rapid odată cu reducerea numărului de facilități care se asigură.

În industria de telefoane s-au realizat tabele ample (vezi Bibliografia) care dau *probabilitatea de blocare (gradul de servire)* în funcție de numărul de căi de transmisie și de sarcină, exprimată în *Erlang* (numărul mediu de chemări). Rezultatele care se obțin diferă ușor în funcție de faptul că chemările nerezolvate sînt *memorate* (distribuție Poisson), *eliminate* (distribuția Erlang B) sau *amînate* (distribuția Erlang C).

Toate expresiile precedente (și fig. 14-6) oferă o bună precizie numai *dacă numărul de clienți este mult mai mare decît numărul de facilități care sînt asigurate.* Dacă, de exemplu, vom încerca să aplicăm distribuția Poisson unei situații caracterizată de existența a doi clienți care sînt fiecare activi 50% din timp ($m = 1$), din figura 14-6 va rezulta că în cazul existenței a patru facilități va exista o blocare de aproximativ 1%. Deoarece în mod evident două facilități pot servi integral doi clienți, este clar că aplicarea distribuției Poisson nu este potrivită. *Deoarece distribuțiile Poisson și Erlang presupun surse infinite, ele nu sînt potrivite pentru estimarea blocărilor în multe sisteme mici.*

Ori de cîte ori numărul de clienți *nu* este mult mai mare decît numărul de facilități, trebuie utilizată distribuția *binominală.* În acest caz probabilitatea de blocare este dată de

$$P = 1 - \sum_{i=0}^{i=k} \frac{n!}{i!\,(n-i)!}\, p^i\,(1-p)^{n-i} \qquad (14\text{-}12)$$

unde p este probabilitatea ca fiecare client să fie activ,

n — numărul total de clienți,

k — numărul de facilități asigurate.

Cu toate că această expresie nu poate fi redusă la tabele simple, valoarea sa nu este dificil de calculat. Mărimea $n\,!/i\,!\,(n-i)\,!$ se poate determina din tabelele de „coeficienți binominali" (unele calculatoare de buzunar pot determina direct valoarea întregii funcții).

EXERCIȚII

1. Care este valoarea *MTBF* a unui sistem care este compus din următoarele piese: 100 de CI (standard), 20 de rezistoare, 15 capacitoare cu mylar și două capacitoare cu tantal?
2. Care este efectul asupra valorii *MTBF* a sistemului din Exercițiul 1 dacă i se adaugă un ventilator, care reduce temperatura joncțiunilor CI de la 70°C la 55°C și temperatura ambiantă de la 50°C la 30°C?
3. Să presupunem că un sistem a funcționat 3 ani, timp în care a avut 8 căderi.
   (a) Care este valoarea *MTBF* determinată experimental pe baza acestor date?
   (b) Care este valoarea minimă a *MTBF* la un nivel de încredere de 60%?
   (c) Care este valoarea minimă a *MTBF* la un nivel de încredere de 80%?
4. Presupunînd că la un nivel de încredere de 60%, un sistem are *MTBF* de 300 de ore, care este timpul care trece pînă la a treia defectare?
5. Care este valoarea *MTBF* a unui sistem format din două sisteme identice, fiecare având *MTBF* de 400 de ore și *MTTR* de 2 ore?
6. Care este probabilitatea ca un circuit *LSI* care controlează injecția de combustibil într-un motor de automobil să nu se defecteze în timpul vieții automobilului? Se va presupune că viața unui automobil este de 10 ani și că *MTBF* a circuitului integrat este de 200 000 ore.
7. Care este probabilitatea ca să nu apară defectări ale CI în timpul vieții automobilului din Exercițiul 6, dacă mai există în plus un CI stabilizator de tensiune cu *MTBF* de 300.000 ore și un sistem de aprindere cu CI cu *MTBF* de 150.000 ore.

8. Presupunînd că repararea și refacerea unei plăci de cablaj imprimat cu 50 de CI, se realizează cu un cost mediu de $ 100, cît de mult putem plăti în plus pe CI ca să trecem de la un *AQL* de 1% la unul de 0,15%?
9. (*a*) Un sistem care conține 5 000 de CI este livrat după 10 ore de funcționare. Care este numărul aproximativ de defectări în primul an de exploatare care se înlătură prin utilizarea de CI îmbătrînite în locul acelora care sînt testate numai electric? Pentru rezolvare se va utiliza o aproximație liniară a curbelor din figura 14-4.
(*b*) Care este suplimentul de preț care poate fi plătit per CI, pentru a compensa cheltuielile de service, știind că o operație de service costă $ 300?
10. Să se realizeze un tabel, similar cu Tabelul 14-2, pentru un nivel de semnificație de 60%, un *AQL* de 1% și $n = 0, 1, 2, 3, 4$
11. (*a*) Cît de multe elemente trebuie să fie măsurate fără a găsi nici un defect la un *AQL* de 0,15% cu un nivel de semnificație de 90%?
(*b*) Dar pentru un nivel de semnificația de 60%?
12. De la mai multe surse se primesc cuvinte cu o viteză medie de 10 cuvinte pe secundă. Aceste cuvinte sînt stocate într-un registru tampon care este golit la fiecare 0,5 secunde. Cît de mare trebuie să fie capacitatea registrului tampon astfel ca probabilitatea de a o depăși să fie mai mică de 0,01%?
13. Cîte circuite de transmisie (trunchiuri) sînt necesare pe o anumită direcție dintr-un sistem telefonic cu 10 000 de linii, fiecare fiind activă 5% din timp, cu 5% din chemări mergînd pe direcția respectivă. Se va presupune că se acceptă o blocare de 0,1%.
14. La închiderea bursei, un sistem de prelucrare a informațiilor despre cursul acțiunilor primește într-o secundă un număr mediu de 5 cereri. Dacă discul pe care se face memorarea poate manipula 11 cereri pe secundă, care este probabilitatea ca în acest interval de timp să se ignore o cerere ca urmare a supraîncărcării?

BIBLIOGRAFIE

Fiabilitate

Bell Telephone Labs, *Physical Design of Electronic Systems, Vol. IV, Design Process*, Prentice Hall, Englewood Cliffs, New Jersey, 1972. (Cap. 3 „Statistics", Cap. 7, „Reliability; Cap. 8 „Reliability of Electronic Parts").

Brauger, Joseph, „A Logical Approach to Testing IC's, *Electronic Products*, August 1969, pp. 140—149. (Avantajele de cost oferite de testare și rodaj).

Grant and Leavenworth, *Statistical Quantity Control*, McGraw-Hill, New York, 1972.

Green and Bourne, *Reliability Technology*, Wiley, New York, 1972. (Conține tabele complete de statistică și fiabilitate a componentelor).

ITT Staff, *Reference Data for Radio Engineers*, Howard Sams, Indianopolis, 1972, Chapter 39, „Probability and Statistics"; Chapter 40, „Reliability and Life Testing".

MIL-HDBK-217B, *Reliability Stress and Failure Rate Data for Electronic Equipment.*

MIL-STD-414, *Sampling Procedures and Tables for Inspection by Variables for Percent Defective.*

MIL-STD-883, *Test Methods and Procedures for Microelectronis. RADC Reliability Handbook*, Rome Air Development Center (Def. Cen. ASTIA Doc. AD 821640), September 1967.

Reliability Analyiss Center, *Digital Failure Rate Data*, Rome Air Development Center, Griffiss AFB, New York 13441, order # MDR-8 (1978), $ 50,00 phone (315) 330-4151.

Taylor, Gordon M., „Forced Air Cooling in High Density Systems", *Electronics*, January 24, 1974, pp. 87—89. (Se examinează aspecte ale fiabilității unui sistem).

Vyenielo, Martin, „Component quality assurance: Which plan is better?" *Electronics*, September 30, 1976, pp. 97—99.

„Mathematics of Realibility" *Electrencs*, Novembgr 30, 1962, pp 54—75.

„Special Issue on Reliability of Semiconductor Devices, „*Proceedings of IEEE*, February 1974.

Utilizarea multiplă a instalațiilor

A.T.&T. *Switching Systems*, American Telephone and Telegraph Co., New York 1961 (Conține grafice pentru așteptări și blocări).

ITT Staff, *Reference Data for Radio Engineers*, Howard W. Sams, Indianapolis, 1972, Chapter 32, „Traffic Concepts".

Panice, Joseph A., *Queing Theory*, Prentice-Hall, New York, 1969.

15

CONSECINȚELE SOCIALE ale ingineriei

15.1. INTRODUCERE

Deși acest capitol pare să fie în afara subiectului acestei cărți el este, într-un sens, cel mai important. Dacă cititorul va uita cîteva din ideile expuse în capitolele anterioare lucrul cel mai rău care se poate întîmpla va fi defectarea cîtorva CI. Ignorarea capitolului de față va avea însă consecințe foarte serioase.

În mod tradițional inginerul a fost îndrumat să fie credincios formulelor și proiectelor sale și să lase pe alții să filozofeze. Sistemul său de valori era limitat la realizarea unui „proiect bun" în sens tehnic. În acest sistem de valori un pistol automat bine proiectat oferă pentru inginer o sursă de satisfacție în aceiași măsură în care o oferă și un dispozitiv care ajută un orb să citească o carte. Tot în acest sistem de valori decizia de a realiza rețeaua electrică la suprafață — montînd conductoarele pe stîlpi — sau subteran va fi strict determinată numai de considerente de cost relativ al instalării.

Ne găsim astăzi într-un punct în care se simte în mod presant necesitatea unui „nou" inginer, care trebuie să utilizeze tehnologia pentru a îmbunătăți *calitatea* vieții fiecăruia. Circuitele integrate ne oferă orizontul unor posibilități fantastice ; totuși fără o schimbare de esență a conștiinței profesionale a inginerului, ele vor reuși doar să ne complice viața cu din ce în ce mai multe fleacuri* lipsite de utilitate.

15.2. „FRUMOASELE ZILE DE ALTADATA"

În epoca „frumoaselor zile de altădată" majoritatea oamenilor lucrau de la răsăritul pînă la apusul soarelui numai pentru a supraviețui. Pentru a se încălzi, în locul sistemului de azi controlat de un termostat automat, era necesar să fie doborîți copacii, să fie tăiați în bușteni și apoi în bucăți mai mici cu care era alimentat focul toată iarna. Pentru a găti o mîncare, mai întîi se făcea focul în sobă. Înainte de a face o baie apa se scotea din fîntînă, după care era încălzită. Pentru a păstra alimentele la rece în timpul verii se utilizau blocuri de gheață care se tăiau în timpul iernii și se păstrau în ghețării. Astăzi cu banii pe care îi cîștigăm în cîteva minute de muncă plătim căldura, apa caldă și frigiderul pentru o zi.

Se pare că tehnologia ne-a eliberat din sclavie deoarece a devenit posibil să facem foarte multe cu foarte puțină muncă. Minierul de astăzi apasă pe un buton și pornește o mașină care scoate într-o oră tot atîta cărbune cît 50

* În original „gadgets" (n.t.).

de mineri care lucrează cu tîrnăcopul într-o zi. Un tractor modern ară un cîmp într-o oră — efort pentru care erau necesare zile de muncă. Un funcționar al zilelor noastre în loc să stea toată ziua aplecat peste birou adunînd cifre apasă pe un buton la un calculator și realizează aceeași muncă în cîteva minute.

Dacă lucrurile se rezolvă astăzi atît de ușor ne punem desigur problema efectiv interesantă de ce denumim acele zile de muncă ce-ți frîngea șalele „frumoasele zile de altădată" ? Realitatea este că tehnologia constituie o binecuvîntare cu două fețe : prin reclamă și publicitate am creat „necesități" despre care oamenii nu și-ar fi imaginat niciodată că pot exista. Pentru a satisface aceste „necesități" trebuie să muncim mult mai din greu decît ar fi necesar.

Noi am folosit tehnologia în mod eronat făcînd ca majoritatea orașelor noastre să constituie o lume — de coșmar — a aerului poluat, a circulației, a zgomotului și a giganticelor reclame luminoase în mișcare. Noi am utilizat noua noastră forță pentru a transforma munții și dealurile în terase presărate cu întinderi de case identice, păduri de stîlpi de rețele electrice și de antene TV. Chiar și atunci cînd mergem la iarbă verde sîntem asaltați de cîrîitul aparatelor de radio cu tranzistoare, de bubuitul motoarelor de motociclete, totul arătînd ca un loc de parcare. Sistemele noastre de calculatoare au devenit inamici care ne contestă personalitatea, care ne invadează intimitatea, care ne trimit ca răspuns la plîngerile noastre pătimașe formulare irelevante.

15.3. O COMPETIȚIE LIPSITĂ DE SENS

Tehnologia satisface atît de bine necesitățile noastre reale sau imaginare, încît ajunge să ofere soluții chiar și pentru problemele pe care ea însăși le creează și — mai mult — soluții la problemele create de soluții. De exemplu, pentru a elimina efortul pe care îl cere cositul putem să cumpărăm o mașină de tuns iarba. Atunci cînd sănătatea noastră ajunge să sufere din cauza lipsei de exercițiu fizic cumpărăm un aparat de antrenament mecanic pentru a *crea efortul* eliminat de utilizarea mașinii de tuns iarba. În cele din urmă realizăm același efort dar în loc să lucrăm în aer liber avînd mulțumirea unei realizări, stăm în casă cu ochii pe ceas trăgînd de manetele aparatului de antrenament.

Mașina de tuns iarba și aparatul de antrenament constituie deci un set de aparate care corespund unul altuia în sensul că se anulează reciproc, cu excepția unui singur punct : nu numai că trebuie să muncim pentru a le plăti pe fiecare, dar trebuie să le și menținem în stare de funcționare fiind deranjați de zgomotul și mirosurile pe care le generează mașina de tuns iarba. Putem ajunge la un succes absolut realizînd o combinație formată dintr-o mașină de tuns iarba prevăzută cu un scaun și un umbrar — astfel că nici nu mai e nevoie să mergem, fiind totodată feriți și de razele soarelui — și un aparat de antrenament mecanic care prin mișcarea pedalelor antrenează un generator electric care debitează pe o rezistență de sarcină variabilă, conectată la un microcomputer care calculează numărul de calorii consumate. Mai mult, putem să cumpărăm chiar o lampă care să înlocuiască soarele, ascuns de umbrarul de pe secerătoare !

Din păcate astfel de combinații nu se vînd de obicei în perechi căci în acest caz nimeni nu le-ar mai cumpăra, deoarece s-ar vedea ironia situației : fiecare parte se vinde separat fără ca fiecare fabricant să-și considere produsul său ca o parte a unei capcane subtile.

Această situație constituie o problemă reală, procesul respectiv fiind determinat de modul în care gîndim și nu de cine știe ce conspirație. Modul în care fiecare din noi face ceea ce are de făcut este absurd : inginerul proiectează produsele, comerciantul se ocupă cu vînzarea lor iar conducerea optimizează profitul. Comerciantul nu-și face nici o grijă dacă vinde țigări care favorizează producerea cancerului pulmonar sau medicamente ineficace ; el își face doar meseria. La fel și inginerul — își face doar meseria, încercînd probabil să facă o treabă bună — dar fără a se gîndi la *efectele pe care le exercită produsul asupra societății.*

Dacă vrem să întrerupem această înlănțuire distructivă, fiecare inginer trebuie să-și asume responsabilitatea acțiunii sale și a efectului ei asupra societății. Atunci cînd un pacient îi cere doctorului un medicament care este dăunător este absolut natural ca doctorul să-l refuze, deoarece ar putea vedea cu proprii săi ochi efectele nocive pe care le-ar avea medicamentul dacă ar face altfel. În profesiunea de inginer o astfel de punere față în față a inginerului cu clienții săi nu se realizează. *Clientul inginerului este întreaga societate* și nu un individ. Ca urmare, se cere din partea inginerului o mult mai mare grijă în ceea ce privește rezultatele finale ale activității sale. Într-adevăr, inginerul poate produce heroină cu carul, fără ca niciodată să-l vadă la față pe drogatul care o consumă.

15.4. INGINERUL CA VÎNZĂTOR DE ILUZII

Revoluția CI ne-a adus în situația de a avea o putere extraordinară. Dintr-o dată putem să realizăm lucruri foarte complicate cheltuind foarte puțini bani. Lista lucrurilor care sînt de făcut este fantastică ; ordinea în care se realizează va fi determinată în parte și de ingineri. Deja, mult din ceea ce s-a făcut nu înseamnă altceva decît niște năstrușnicii complicate care fac viața mai dificilă, în loc să-i amelioreze calitatea.

Una dintre primele aplicații „de bunuri de larg consum" a circuitelor LSI este de exemplu comanda digitală a temperaturii unei mașini de gătit, prin apăsarea succesivă a tastelor unei claviaturi corespunzătoare valorii dorite a temperaturii. Valoarea introdusă este verificată automat pentru a stabili dacă este rezonabilă ; în caz contrar apare un mesaj de eroare care arată că valoarea respectivă este în afara gamei permise. Un afișaj digital indică valoarea care s-a fixat pentru temperatură și timp. Timpii la care se programează pornirea și oprirea cuptorului se introduc tot digital, de la claviatură. Această nouă lume a electronicii a produs deci o jucărie pentru adulți, care de fapt nu oferă un avantaj real față de abordarea mecanică simplă utilizată anterior.

În loc ca puterea CI să fie folosită pentru a ajuta un orb să vadă sau să ne scutească de oboseala unor preocupări rutiniere ea a generat fleacuri care îi silesc pe vicioșii lor utilizatori să lucreze 80 de ore pe săptămînă pentru a-și menține situația cu care s-au obișnuit. Motivul simplu care explică faptul că astăzi omul pare să nu aibă niciodată destui bani, chiar dacă el poate cîștiga suficient pentru satisfacerea necesităților fundamentale în numai două ore de muncă pe zi, este acela că el muncește pentru a plăti

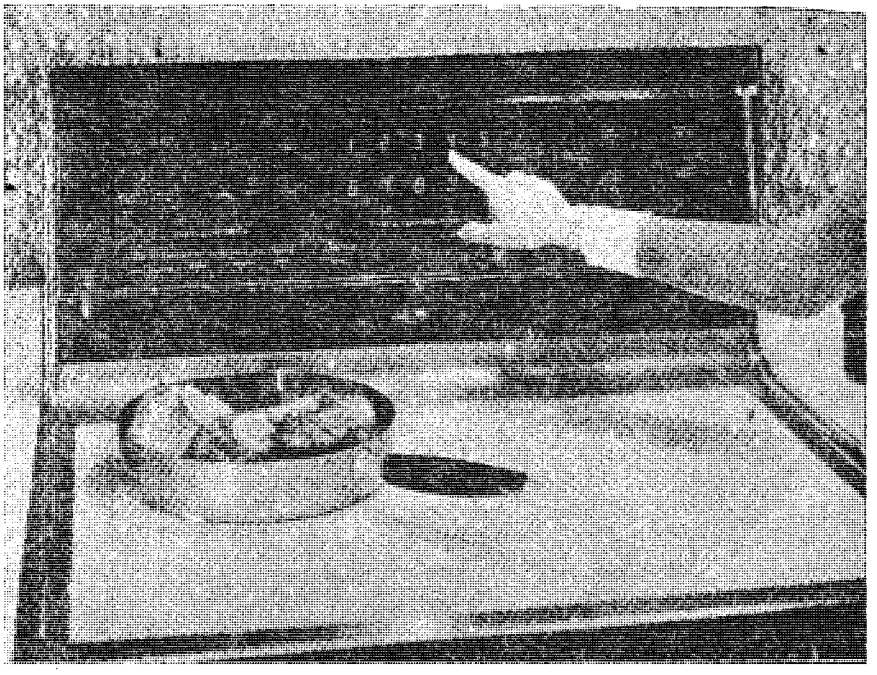


Fig. 15-1. Comanda senzorială a unei mașini de gătit (Frigidaire).

astfel de fleacuri. La fel ca în cazul drogurilor, nevoia de drog nu este niciodată satisfăcută, căci pofta vine mîncînd. Inginerul cu dragostea și atracția sa față de tot felul de lucruri năstrușnice este creatorul — gata să acționeze în orice moment — a unor astfel de capcane.

Gordon Moore* a extrapolat ritmul actual de creștere al funcțiilor de circuit realizate de CI (porți, biți, etc.) produse anual și a găsit că, *în 1985 se va produce un sfert de milion de funcții de circuit pentru fiecare om de pe pămînt.* Aceste noi circuite pot să constituie o binecuvîntare sau un blestem în funcție de scopul pentru care vor fi utilizate. Deoarece aproape orice treabă *se poate face* automat există tendința de a automatiza lucruri care pot fi făcute tot atît de ușor și manual.

Un exemplu perfect de automatizare pentru propria-ți plăcere este dat de dispozitivul Heatkit GR-2001 pentru preprogramarea televizorului. Acest dispozitiv utilizează un procesor bipolar de 60 de cipuri care permite utilizatorului să-și programeze pe două zile înainte programul pe canalele pe care dorește să le urmărească. Deoarece comanda la distanță a eliminat deja necesitatea de a se scula de pe scaun și de a schimba canalul, acest dispozitiv pare a fi realizat pentru acei oameni a căror degete obosesc atunci cînd apasă butoanele!

Pe măsură ce revoluția CI evoluează devine din ce în ce mai important ca inginerii să treacă dincolo de etapa care corespunde întrebării „se poate face ?", problema importantă într-o lume în care practic se poate face totul, fiind de fapt „are utilitate" ?.

* Gordon Moore este vicepreședinte al firmei INTEL (n.t.).

Este interesant de observat că în timp ce am investit atîta efort în dezvoltarea unor circuite LSI care să pornească și să oprească mașina de gătit, aproape 100% din convorbirile noastre telefonice sînt realizate prin comutări cu *relee*. La fel, chiar telefonul însuși este în esență neschimbat de la începutul secolului. Același microfon cu cărbune utilizat de Alexander Graham Bell în 1877 este folosit la un loc cu același tip de disc utilizat de Strowger în 1895. Singura schimbare o constituie carcasa lucioasă realizată prin injectarea unei mase plastice. Cine poate ști oare ce scădere a prețului ar fi posibilă dacă tehnologia de CI ar fi aplicată sistemului nostru telefonic !

15.5. NOUL TIP DE INGINER

Dacă vrem ca tehnologia să elibereze omul și nu să-l facă un sclav este nevoie de un nou tip de inginer care să fie mai mult decît un tehnolog foarte specializat. Inginerul trebuie să fie conștient de consecințele sociale ale activității sale și trebuie să ia în considerare atît partea estetică cît și partea economică a unei probleme. Dacă sîntem atît de bogați încît să avem mașini de gătit digitale, în mod sigur vom mai putea cheltui în plus 10% pentru a face ca un sistem de calcul să răspundă la *numele* oamenilor și nu la niște numere sau să-i înghesuie într-o categorie aleasă, de exemplu, din alte 100 de categorii și nu din patru. Sistemele de calcul pot fi dezumanizante sau nu, în funcție de modul în care sînt programate. Aceste sisteme fie ne pot trimite formulare improprii care ne fac să ne simțim asemenea unor numere, fie — cu prețul unui mic efort suplimentar — pot purta grija corespondenței de *rutină* și — chiar în plus — pot prezenta *oamenilor* pentru soluționare problemele interesante.

Un exemplu perfect al rezultatului aplicării numai a unor rațiuni economice unei probleme îl constituie decizia de a construi rețeaua de distribuție a energiei electrice la suprafață. O analiză bazată pe rațiuni economice pure arată că este mai ieftin să se întindă rețeaua de fire deasupra pămîntului. Care este însă în acest caz costul urîțeniei rețelei de fire și a victimelor datorită electrocutărilor și zdrobirii mașinilor de stîlpii care susțin rețeaua ? De altfel, în anii din urmă în multe comunități s-au luat hotărîri care interzic plasarea utilităților deasupra pămîntului. Odată ce compania de electricitate a fost forțată să dezvolte tehnici pentru instalarea subterană a cablurilor, s-a dezvoltat și echipamentul care a făcut ca instalarea subterană să constituie alternativa *mai ieftină*.

Sistemele digitale de la începuturi au încercat adeseori să potrivească omul cu mașina, și nu invers. Acum, datorită faptului că complexitatea logicii digitale este atît de ieftină, putem începe să realizăm dispozitive care să facă exact ceea ce vrem noi să facă. De exemplu, pe măsură ce costul memoriei scade, costul suplimentar determinat de utilizarea numelui cuiva în locul unui număr devine nesemnificativ. Mașinile care își făceau munca asemenea unui funcționar insuportabil și rigid, pot fi acum programate să facă munca de rutină asemenea unui splendid și eficient funcționar care are și o memorie perfectă.

Piesele fabricate de mașini au însemnat întotdeauna o standardizare plictisitoare ; acum putem să începem să facem mașini care pot fi programate să realizeze o varietate infinită de piese care oferă aceleași avantaje relative la cost, ca și acelea care mai înainte se puteau obține numai dacă toate piesele ar fi fost identice. Caracterul personal al mobilei, al finisării etc. din frumoasele zile de altă dată, atunci cînd orice era făcut cu mîna, poate fi realizat acum cu ajutorul mașinilor programabile.

În timpurile de început ale automatizării exista teama că ea ar putea duce la un puternic șomaj. Trecerea timpului a arătat însă că situația este cu totul alta. Noile industrii create prin automatizare au asigurat tot atîtea noi locuri de muncă cîte au dispărut. Deși, cu certitudine, lucrăm mai puține ore pentru a produce aceleași bunuri, munca preluată de mașini constituie partea rutinieră, partea care îți frînge șalele, partea care te plictisește. Partea care rămîne omului este partea interesantă, creativă. Succesul încercării de a face tehnologia slujitorul și nu stăpînul nostru depinde de gradul de conștiință al noului tip de ingineri.

BIBLIOGRAFIE

Balabanian, Norman, „Technology and Values" (letter), *IEEE Spectrum*, February 1974, pp. 26—27.

Blake, Peter, *God's Own Junkyard*, Holt, Rinchart & Winston, New York, 1964.

Braden, William, *The Age of Aquarius*, Quadrangle, Chicago, 1970.

Christiansen, Don, „The New Professionalism", *IEEE Spectrum*, June 1972, p. 17.

Florman Samuel C., *The Existential Pleasures of Engineering*, Marin's Press, New York, 1976.

Fromm, Eric, *The Revolution of Hope*, Harper & Row, New York, 1968.

Grosch, Herb, „Problems and Priorities", *Datamation*, March 1972, p. 48.

IEEE, Committee on Social Implications of Technology Newsletter (a free newsletter to members of IEEE).

Kemeny, *Man and the Computer*, Scribner's, New York 1972.

Lindgren, Nilo, „Semiconductors in the 80' s", *IEEE Spectrum*, October 1977, pp. 42—47.

Toffler, Alvin, *Future Shock*, Bantain Books, New York, 1970.

Ullrich, Manfred and Hegendorter, Max, „TV receiver puts two pictures on screen at same time". *Electronics*, September 1, 1977, p. 102.

Weizenbaum, Joseph, *Computer Power and Human Reason : From Judgement to Calculation*, W. H. Freeman, San Francisco, Calif., 1977.

„Bipolar processor in TV set leads to preprogrammed channel selection", *Electronic Design* 3, February 1, 1977.

„Electric range holds cooking programs", Electronics, March 4, 1976, p. 33.

POSTFAȚA

Evoluția calculatoarelor sau, mai general, a sistemelor digitale, constituie un rezultat al revoluției tehnologice din domeniul componentelor, în special al circuitelor integrate.

O ilustrare cantitativă a evoluției calculatorului poate fi dată, de exemplu, de considerarea factorului de merit* definit de (viteză de comutație × fiabilitate)/(consum de putere × greutate × preț). **De la etapa corespunzătoare calculatorului *ENIAC* (1947) la etapa corespunzătoare unui microcalculator de 16 biți (1985) acest factor de merit a crescut de 10^{18} ori!** Este clar că simpla lectură a acestui număr are darul de a te pune foarte serios pe gînduri!

Unul din punctele de discontinuitate, de evoluție prin salt, l-a constituit apariția microprocesorului (1971) care a determinat amorsarea unui proces cu reacție pozitivă — economic, științific, social, politic — al cărui caracter puternic regenerativ se menține și în prezent.

Caracterizarea, des utilizată, a actualei trepte de dezvoltare prin două circuite integrate digitale standard industrial (memoria dinamică de și microprocesorul de 32 biți) este fără îndoială corectă dar, totodată, și incompletă deoarece „uită" alte sute de circuite „de plan secund", descrise în cataloage al căror volum a început să se măsoare în „kilo-pagini".

Unul din meritele esențiale ale acestei cărți îl constituie faptul că autorul *consideră microprocesorul* (microcalculatorul) — deși extrem de important prin el însuși — *doar unul dintre multiplele mijloace pe care trebuie să le stăpînească cei ce lucrează în domeniul sistemelor digitale.* Această idee constituie un element fundamental în structura de rezistență a cărții, *autorul concentrîndu-se numai asupra conceptelor de bază* și nu asupra detaliilor ce caracterizează funcționarea unui microprocesor particular.

Structurarea întregii cărți se bazează pe considerarea proiectării logice la nivelul *sistemului.* Acest fapt constituie alt merit esențial al cărții din cel puțin două motive.

Un prim motiv este dat de *răspîndirea abordării numerice,* care pătrunde în domenii noi (automobile, monitorizarea activităților casnice, jucării etc.) și își consolidează pozițiile în domenii deja abordate (achiziție și prelucrare de date, transmisie de date, electronică medicală, radio și televiziune, inteligență artificială, robotică etc.); *elaborarea unor soluții eficiente pentru orice problemă generată de abordarea numerică pleacă în mod obligatoriu de la nivelul sistemului.* În plus, nenumăratele circuite noi specifice prelucrării digitale a semnalelor, își găsesc astfel locul fără dificultate în sistemele digitale alături de microprocesor sau microcalculatorul dedicat.

Un al doilea motiv — cu un puternic ecou pentru proiectantul de sisteme — este dat de faptul că atît *circuitele LSI și MSI* — cît și *componentele mecanice și electrice sînt componente de sistem* a căror combinare trebuie

* V. Zieren ș.a. *Silicon micro-traducers,* European Electronics, Issue Five (1981), p. 10

realizată *în mod optim pentru a se obține un sistem eficient din punctul de vedere al raportului performanță/cost.* Datorită revoluției în industria de componente costul logicii constituie o parte nesemnificativă din costul întregului sistem, astfel că devine obligatorie considerarea sistemului ca un tot unitar.

În acest context cartea se ocupă în esență de modul în care trebuie utilizate circuitele LSI și MSI standard în proiectarea logică.

Punctul de plecare îl constituie cerința de a da pentru sistem o soluție eficientă din punct de vedere economic, sau altfel spus, cum să facem ca unui caiet de sarcini dat să-i corespundă produsul cel mai ieftin posibil. Prin fixarea acestui punct de plecare autorul arată că știe bine că problema esențială nu este să produci (se poate produce orice și oricum) ci, de fapt, să reușești să vinzi ceea ce produci.

Tehnica de proiectare expusă de autor se axează pe utilizarea în primul rînd a **componentelor ieftine** (în original: **bargain components)** standardizate, electronice prin natura lor, disponibile în cantități mari, din mai multe surse și la prețuri mici pentru a satisface majoritatea cerințelor sistemului, iar apoi pentru a completa ce mai este nevoie, a circuitelor integrate SSI. Criteriul de eficiență utilizat constă în *reducerea la minimum a numărului de capsule de circuite integrate* și nu a numărului de circuite basculante și porți în conformitate cu abordarea tradițională.

Materia cărții se poate împărți în patru părți mari care sînt discutate în continuare.

● **Impactul circuitelor LSI asupra abordării sistemelor digitale.** În această parte autorul abordează o serie de probleme fundamentale din punctul de vedere al tehnicii de proiectare descrisă în carte (capitolele 1 și 2) cît și o serie de probleme conexe sistemelor digitale (capitolele 14 și 15).

● **Fundamente. Circuite logice MSI.** Această parte, Capitolul 3 (Logica combinațională I : proiectarea logică tradițională), Capitolul 4 (Logică combinațională II : proiectarea cu circuite MSI și LSI) și Capitolul 5 (Logică secvențială : proiectare), tratează problemele tradiționale ale circuitelor logice combinaționale și secvențiale, constituind o trecere în revistă a chestiunilor fundamentale care constituie baza oricărei activități de proiectare de sisteme logice, punîndu-se accentul pe utilizarea — acolo unde este posibil — a circuitelor MSI și LSI. Tehnicile de bază ale proiectării logice tradiționale sînt prezentate deoarece ele sînt necesare atît pentru „a umple interstițiile" dintre circuitele MSI și LSI cît și pentru sistemele foarte mici.

● **Logică programată. Circuite logice LSI.** Această parte, compusă din Capitolul 7 (Logica programată I : microcalculatoare), Capitolul 8 (Logică programată II : programare asistată de calculator), Capitolul 9 (Logică programată III : sisteme de dezvoltare), și Capitolul 10 (Logică programată IV : proiectarea hardware a microcalculatoarelor), acoperă conceptul de programare, limbajele de programare, sistemele de dezvoltare pentru microcalculatoare și tehnicile de proiectare pentru hardware de microcalculator.

● **Implementarea sistemelor digitale de la schema funcțională la sistemul care merge.** În capitolul 11 (Dimensiunea timp) se discută metodele de utilizare multiplă a circuitelor pentru prelucrarea serie a biților sau pentru funcționarea cu divizarea timpului, utilizînd distribuirea în timp a procesării.

Capitolul 13 (Dispozitive de intrare-ieșire) prezintă cîteva tehnici (de exemplu reacția negativă, funcționarea incrementală) care sînt utilizate în dispozitivele de intrare-ieșire. În calitate de exemplu se descriu cîteva periferice de calculator.

Capitolele 6 (Realităti neplăcute I : probleme de propagare şi blocare) şi 12 (Realităti neplăcute II : zgomotul şi reflexiile) iau în considerare realităţile neplăcute ale lumii reale cum ar fi de exemplu, efectele date de propagare, blocarea în stări necontrolabile, zgomotul, reflexiile şi diafonia.

Impresia globală pe care o lasă această carte este deosebit de favorabilă. Cîţiva din factorii care contribuie la crearea acestei impresii sînt :

— utilizarea unui limbaj accesibil, care face cartea utilă unui cerc foarte larg de cititori ;

— urmărind cuprinsul cărţii se constată că asimilarea sa oferă un nivel de start mai mult decît mulţumitor pentru un proiectant de sisteme digitale ;

— ancorarea în realitatea existenţei circuitelor LSI ; în acest sens este interesant de subliniat faptul că însuşi autorul recomandă completarea lecturii cărţii cu consultarea unuia, sau a mai multor cataloage de circuite LSI ;

— urmărirea cu stăruinţă a înţelegerii fenomenului sau a elementelor determinante în alegerea unei soluţii ;

— considerarea unor exemple (discutate în detaliu, utilizînu-se diverse implementări) cu rolul de cărăuşi de idei.

Viabilitatea acestei cărţi este probată şi de faptul că evoluţiile în domeniu, ulterioare, scrierii sale, se aşază în mod natural peste edificiul creat de această carte, fără să dărîme nimic. Cartea se constituie într-un exemplu rar de rezistenţă la eroziunea timpului într-un domeniu cu o dezvoltare exponentială.

I. Impactul circuitelor LSI asupra abordării sistemelor digitale

În capitolul 1 (Ideea : să adaptăm ceea ce avem de făcut la componentele ieftine) se expun mai întîi motivele care au făcut ca abordarea digitală să constituie calea urmată pentru soluţionarea unei varietăţi extreme de probleme care în esenţa lor nu sînt digitale sau cel puţin electrice. Explicaţia o găsim în procesul de fabricaţie a circuitelor integrate, proces deosebit de eficient cu condiţia însă de a lucra în serii mari. Obţinerea unei eficienţe ridicate în utilizare impune cu necesitate standardizarea, astfel încît diferite sisteme digitale să poată fi realizate cu un număr minim de tipuri de circuite integrate.

Autorul introduce noţiunea de componentă ieftină definită drept o componentă standardizată, care se produce în cantităţi mari de către diverşi producători la costuri mici.

Desigur că prin utilizarea componentelor standardizate vom fi întotdeauna confruntaţi cu neadaptarea componentei la aplicaţie. Totuşi această neadaptare este mai mult decît compensată de economia imensă pe care o realizăm prin simplul fapt al utilizării componentelor standardizate.

Rezultă în consecinţă principiul general de proiectare: *trebuie să adaptăm ceea ce avem de făcut la componentele ieftine existente.*

În încheierea capitolului se prezintă cîteva exemple de componente ieftine, pentru a da culoare acestui concept şi se discută strategia de alegere a unei componente ţinîndu-se cont de evoluţia tipică a preţului unui circuit integrat.

Dezvoltările anilor '80, la cîţiva ani după scrierea cărţii, aduc în centrul atenţiei un nou circuit LSI ieftin : **aria de porţi.** Proiectarea cu ariile de porţi implică, în bună parte, utilizarea tehnicilor tradiţionale în configuraţii ne-

standard. Astfel utilizarea microprocesorului pierde caracterul de soluție ce tindea să fie unică, oferind proiectantului o alternativă competitivă.

În capitolul 2 (Obiectivele proiectării unui sistem digital) se afirmă încă de la început că obiectivele tradiționale ale proiectării logice nu mai sînt valabile datorită faptului că în prezent în prețul de cost al unui sistem logic costul circuitelor integrate este neglijabil.

Analiza structurii costului unui sistem digital realizat în tehnologia modulară actuală arată că acesta este practic proporțional cu numărul de capsule de circuite integrate. Această observație oferă în mod direct *criteriul de eficiență* utilizat de autor: reducerea la minimum a costului sistemului se realizează prin *reducerea la minimum a numărului de capsule de circuite integreate.*

În partea finală a capitolului se descrie structura generală a unui sistem digital urmărindu-se implementarea ideii că proiectarea oricărui sistem digital începe cu necesitate de la nivelul de complexitate corespunzător sistemului.

Aceste două capitole stabilesc și ilustrează prin exemple ideile de proiectare utilizate ulterior în toată cartea.

Capitolul 14 (Utilizarea statisticii în proiectarea digitală) discută dintr-un punct de vedere strict practic utilizarea metodelor statistice pentru caracterizarea fiabilității sistemelor și pentru utilizarea partajată a sistemelor.

Capitolul 15 (Consecințele sociale ale ingineriei) reflectă cîteva din problemele pe care le poate genera folosirea lipsită de control a rezultatelor eforturilor de producție, cercetare și dezvoltare, militînd pentru implicarea socială, conștientă, a inginerului.

II. Fundamente. Circuite logice MSI

În cartea sa — tipărită în deceniul opt — deceniul în care au apărut și s-au impus microprocesorul, memoriile semiconductoare și o serie întreagă de alte circuite de mare complexitate — Thomas Blakeslee simte nevoia să se justifice pentru tratarea tehnicilor elementare de proiectare a circuitelor logice. La ce i-ar mai fi utile procedeele de implementare cu porți simple și cu bistabile inginerului proiectant de sisteme, pentru că acestuia i se adresează cartea în discuție, atunci cînd are la dispoziție *ROM*-uri, *PLA*-uri, registrele cele mai diverse sau alte circuite dedicate? Autorul găsește o justificare: oricît ar fi de bine gîndite structurile *MSI* și LSI ele nu se pot interconecta întotdeauna direct, în absența unor circuite de adaptare realizate, de regulă, cu porți simple sau bistabile în configurații ce se optimizează cu procedee clasice.

A mai trecut aproximativ un deceniu și, poate spre surprinderea multora, există astăzi motive în plus pentru tratarea atentă a procedeelor clasice, elementare, de implementare a sistemelor digitale.

Ce s-a petrecut între timp? Iginerul pentru care scria în deceniul opt Thomas Blakeslee avea drept scop major în proiectare *minimizarea numărului capsulelor de circuite integrate*, complexitatea circuitelor dintr-o capsulă contînd într-un plan secund. În anii '80 inginerul proiectant capătă acces la structura integrată pe siliciu prin intermediul facilităților de **„custom-design"** (realizarea circuitelor integrate la cererea clientului) sau prin utilizarea din ce în ce mai intensivă **a ariilor de porți (gate-arrays).** Nu se mai pune în acest caz problema reducerii numărului de capsule ci, în primul rînd, aceea a reducerii complexității structurii, pentru a putea fi efectiv realizată într-un context tehnologic dat.

Atunci cînd trecem de la proiectarea *cu* cipuri la proiectarea *pe* cipuri este absurd să folosim pentru un circuit logic combinațional un ROM, vom utiliza întotdeauna o structură de tip PLA. De asemenea oricît de spectaculoase se dovedesc tehnicile de implementare cu multiplexoare și demultiplexoare, pe siliciu ele vor ceda locul, în majoritatea cazurilor, unor rețele de porți elementare structurate sub formă de PLA. Redevin astfel interesante tehnicile de minimizare și transformare algebrică, pe care le ignoram atunci cînd proiectam cu MUX, DMUX, ROM sau alte structuri programabile. Un PLA este un ROM optimizat prin eliminarea produselor elementare neutilizate, deci este normală impunerea acestui circuit ca tipic pentru logica asociată unei structuri digitale implementate pe o arie de porți sau într-un circuit dedicat. Este adevărat că există situații particulare (impuse uneori și de tehnologia utilizată) cînd o structură oarecare de porți se dovedește cea mai utilă ; în nici un caz însă nu se mai pune problema implementării direct pe siliciu a unui circuit combinațional sub formă de ROM.

Proiectarea unui circuit logic combinațional nu presupune numai minimizări sau transformări ale funcțiilor logice ce-l descriu ci și **adecvarea la criterii ce depășesc formularea la nivelul algebrei logice.** Acest proces de adecvare este probabil imposibil de formalizat fiind, în consecință, dificil de expus într-un tratat. Dacă la aspectele formale expuse în carte nu ar fi lucruri esențiale de adăugat (exceptînd completarea tabelului 3-1 cu identitate $A + A' \cdot B = A + B$, deosebit de utilă în minimizări algebrice) pare deosebit de utilă exemplificarea procesului de adecvare, realizabil pe baza unor criterii neformalizate. O vom face, într-un prim exemplu, prin *proiectarea structurii unui sumator complet de un bit.*

Exemplul 1. Schema bloc a sumatorului complet pentru cuvinte de un bit este dată în figura *P*-1, unde A și B sînt intrările asociate celor două numere, de cîte un bit fiecare, C este intrarea de transport (carry) de la ordinul binar inferior, S este ieșirea la care apare bitul ce reprezintă suma iar C_+ ieșirea ce indică depășirea către ordinul binar superior.

Ecuațiile logice ce descriu funcțiile celor două ieșiri se pot scrie și direct. Astfel

$$S = A \oplus B \oplus C$$

deoarece S este suma modulo doi (SAU EXCLUSIV) între A și B ce se sumează tot modulo doi cu C, iar

$$C_+ = AB + AC + BC$$

întrucît se obține depășire de fiecare dată cînd cel puțin doi biți la intrare sînt activi.

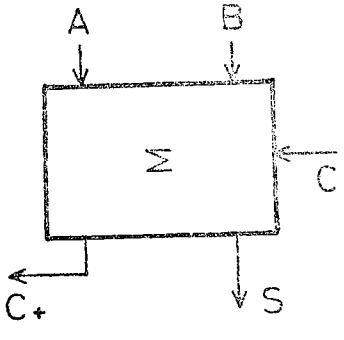


Fig. *P*-1. Sumator complet de un bit.

Prin aceasta partea pur formală, ce-l implică numai pe logician, este epuizată. Din acest moment intră în acțiune inginerul.

(*a*) O primă problemă de inginerie este aceea de a încerca formularea celor două ecuații logice, S și C_+, astfel încît să se *maximizeze*, pe cît posibil, *circuitele utilizate în comun* la implementare. Pentru aceasta este foarte di-

ficil de formulat un procedeu universal valabil dar este important să credem că uneori putem avea succes în căutarea unor forme convenabile de exprimare a ecuațiilor logice. Să încercăm, deci !

Într-o primă etapă vom scrie :

$$C_+ = A(B + C)_{\cdot} + BC$$

regretînd apariția în paranteză a formei $B + C$ în loc de $B \oplus C$ ce se regăsește și în formularea sumei S. Dar, ultimul produs din C_+ ia în considerație existența simultană a variabilelor B și C cu valoarea logică adevărată, situație exclusă prin $B \oplus C$. Nu am putea atunci spera ca formula

$$A(B + C) + BC = A(B \oplus C) + BC$$

să fie o identitate ?
Într-adevăr :

$$A(B \oplus C) + BC = A(bC' + B'C) + BC = ABC' + AB'C + BC$$

și conform principiului semiabsorbției ($A + A'B = A + B$) putem scrie

$$ABC' + AB'C + BC = AB + AC + BC.$$

Rezultă că avem de implementat sistemul de ecuații logice

$$S = A \oplus B \oplus C$$

$$C_+ = A(B \oplus C) + BC$$

în care am reușit să realizăm ce am urmărit : $B \oplus C$ se regăsește în ambele formule.

Complicînd într-o manieră, din păcate, nealgoritmică, forma minimală algebric a unei ecuații am obținut o minimizare a ansamblului.

Rezultă schema din figura P-2, unde un circuit XOR este folosit în comun de cele două funcții logice.

Pornind de la un regret și continuînd cu o speranță am parcurs o primă etapă prin care *inginerul depășește ceea ce-i poate oferi logicianul.*

b) Sînt rarisime aplicațiile care presupun un singur sumator complet de un bit. De regulă, utilizînd sisteme de tipul celui din figura *P*-1, se construiesc sumatoare de mai mulți biți (8, 16, 32 sau mai multi biți) prin conectări de tipul celei din figura *P*-3. Se observă, în acest caz, că pentru finalizarea

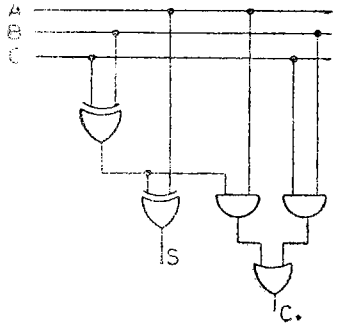


Fig. *P*-2. O primă implementare cu porți a sumatorului complet de un bit.

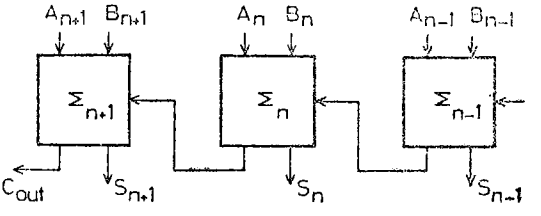


Fig. *P*-3. Interconectarea sumatoarelor complete de un bit.

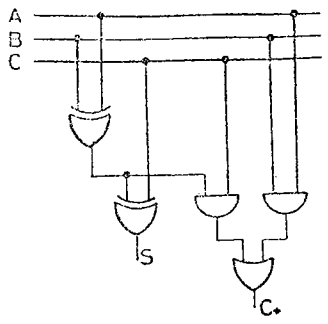


Fig. P-4. O a doua implementare a sumatorului complet de un bit. A fost minimizat timpul de propagare a semnalului C.

procesului de sumare este necesar ca semnalul de transport să se propage printr-un număr de sumatoare care foarte rar este mai mic de opt ! Critică este, deci, propagarea semnalului C prin schema logică din figura P-2. Analizînd schema rezultă că semnalul C se propagă printr-un XOR, un AND şi un OR, calea cea mai lungă ce apare în această configuraţie. Nu am putea oare evita această situaţie, oferind semnalului de depăşire o cale de propagare mai directă, printr-un număr mai redus de porţi logice ?

Pornind de la prima formulare a ecuaţiei C_+ să încercăm un alt mod de aplicare a legii distributivităţii, obţinînd

$$C_+ = C(A + B) + AB$$

şi în consecinţă sistemul

$$S = A \oplus B \oplus C$$
$$C_+ = C(A \oplus B) + AB,$$

care se implementează conform reprezentării din figura P-4. De această dată C se propagă la ieşirea C_+ numai printr-un AND şi un OR. *Obţinem scurtarea timpului de propagare numai prin reformularea unei ecuaţii.* Circuitul a rămas identic dar un timp de propagare critic s-a înjumătăţit.

Cum să facem ca şi în alte situaţii să obţinem un efect similar ? Nu putem da o reţetă, dar este bine să credem că uneori acest lucru este posibil.

(*c*) Într-un sumator de n biţi semnalul C se propagă printr-un număr de $2n$ porţi neinversoare. În acest caz, dacă se lucrează la viteză mare, se poate petrece un efect deosebit de neplăcut, care se materializează în posibila dispariţie a semnalului după ce a parcurs un număr suficient de mare de porţi neinversoare. Să explicăm. Aproape toate tehnologiile presupun realizarea porţilor elementare astfel încît timpul de propagare al tranziţiei pozitive este *diferit* de cel al tranziţiei negative. În figura P-5 este reprezentat efectul trecerii unui impuls de mică durată prin trei nivele logice neinversoare realizate cu porţi la care s-a presupus că tranziţia negativă se produce cu o întîrziere mai mică decît cea pozitivă. Remarcăm tendinţa netă de micşorare a duratei impulsului iniţial. După un număr oarecare de porţi el poate deveni atît de scurt încît să nu mai poată declanşa un nou circuit. Ce se întîmplă însă în cazul utilizării unor circuite inversoare ? Figura P-6 va reprezenta

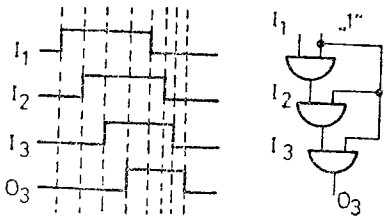


Fig. P-5. Propagarea deformată a unui impuls scurt prin trei porţi neinversoare.

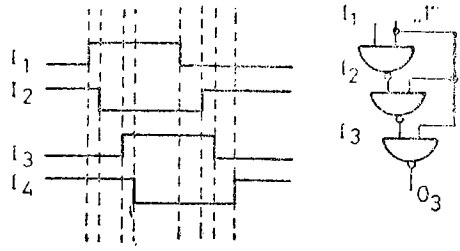


Fig. P-6. Propagarea minimal deformată a unui impuls scurt prin trei porţi inversoare.

efectul propagării unui impuls scurt prin trei nivele inversoare. Efectul cumulativ constatat în figura *P*-5 nu se mai petrece. După un număr par de porți impulsul tinde să se refacă iar după un număr impar de circuite abaterea previzibilă este cea introdusă de un singur nivel.

Vom elimina nivele neinversoare pe calea care corespunde propagării transportului într-un sumator complet aplicînd teorema lui de Morgan ecuației C_+ :

$$C_+ = ((C(A \oplus B))'(AB)')'$$

Se va obține, asociat unei formulări algebrice mai stufoase, structura din figura *P*-7 ce poate fi considerată, pentru exercițiul nostru, o soluție finală acceptabilă.

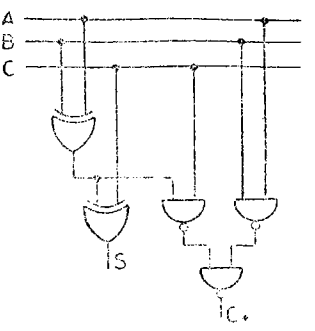


Fig. *P*-7. Soluția finală a unui sumator complet de un bit.

Exemplul anterior ilustrează, credem, suficient de elocvent, ce cale mai are de străbătut inginerul proiectant după ce a epuizat mijloacele de minimizare puse la dispoziție de logician.

Am punctat cu această ocazie și unele aspecte ce nu sînt suficient abordate în carte.

Un merit deosebit al cărții lui T. Blakeslee este accentuarea importanței gîndirii sistemice, în sensul că o bună abordare la nivel de schemă bloc asigură în mare măsură succesul unui proiect. Ar fi interesant de menționat și faptul că proiectarea finală, de detaliu, este de asemenea influențată de „dimensiunea cărămizilor" cu care operăm. Nu întotdeauna este recomandabil ca odată definite schemele bloc să se treacă direct la implementări ce presupun nivelul cel mai coborît, al porților elementare și al bistabililor. De multe ori utilizarea unor *blocuri cu* **funcții** *ceva mai complexe* ne oferă soluții net avantajoase și aceasta nu numai din perspectiva utilizării unui număr redus de capsule de circuite integrate, ci strict din punctul de vedere al complexității structurale globale.

În această carte se subliniază faptul că utilizarea unui numărător MSI, spre exemplu, este mai avantajoasă decît utilizarea a trei bistabili JK, dar acest avantaj se manifestă numai prin minimizarea numărului de capsule cu circuite integrate, structura efectivă este mult mai complexă. *O astfel de abordare este inacceptabilă atunci cînd proiectăm direct pe siliciu.* Deci, utilizarea unor funcții mai complexe, în accepțiunea noastră, nu înseamnă folosirea unor circuite MSI sau LSI standard, ci o abordare conceptuală la un nivel la care uneori se pot obține implementări mai eficiente. Iarăși, nu ne vom putea explica decît pe un exemplu, acest mod de abordare nefiind ușor algoritmabil deoarece ține de acea parte a ingineriei prin care aceasta se manifestă ca o *artă*.

Exemplul 2. Vom considera automatul descris de graful din figura *P*-8 ce generează, declanșat de comanda START și controlat de variabila aleatoare VA, elementele limbajului

$$L(G) = \{a^n\ b^m\ c^p / m,\ n,\ p \geqslant 1\}.$$

Variabilele de ieșire (terminalele gramaticii G) sînt codificate după cum urmează : $a = 00$, $b = 01$, $c = 10$ iar $e = 11$.

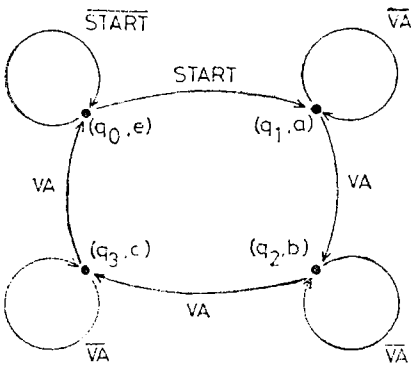


Fig. *P*-8. Graful automatului generator al secvențelor de tipul $a^n\ b^m\ c^p$ cu $m,\ n,\ p \geqslant 1$.

(*a*) Pentru simplificarea circuitului combinațional asociat automatului vom codifica stările $q_0, \ldots, q_3$ prin codurile asociate ieșirilor corespunzătoare.

(*b*) Se observă (*cum ?!*) faptul că pentru START, cînd $Q_1Q_0 = 11$, și VA, în restul stărilor, automatul *comută prin numărare*. Prin Q_1 și Q_0 s-au notat cei doi biți asociați stării. Rezultă :

— cei doi bistabili pentru memorarea stării automatului vor fi conectați în configurație de *numărător condițional* ;

— *condiția de numărare* va fi *multiplexată*, alegîndu-se, cînd Q_1 și Q_0 *coincid ambele cu valoarea 1 logic*, semnalul START, iar în rest semnalul VA.

Deci automatul va fi format dintr-un *numărător*, un *multiplexor* de două căi și un circuit ce stabilește *coincidența* lui Q_1 și Q_0 pe valoarea de 1 logic. Utilizăm în descrierea automatului **funcțiile** de *numărare*, *multiplexare* și *coincidență*, nu pe cele de *bistabil* și *poartă logică*. Vom desena structura logică a automatului ca în figura *P*-9, *direct* pe baza observațiilor anterioare.

(*c*) Se pune întrebarea dacă prin aplicarea procedeelor formale, expuse și în această carte, se poate deduce configurația din figura *P*-9. Poate una mai simplă ! ? Să încercăm !

Păstrînd codificările deja făcute diagrama de tranziție a automatului este aceea din figura *P*-10. Din aceasta se deduc diagramele Veitch ale intrărilor bistabililor asociați automatului (figura *P*-11), de unde rezultă :

$$J_1 = Q_0 VA_0' = (Q + VA')',$$

$$K_1 = Q(START) = (Q_0' + (START')',$$

$$J_0 = VA,$$

$$K_0 = Q_1(START) + Q_1 VA = ((Q_1(START))'(Q_1'VA)')'.$$

Spre surprinderea noastră circuitul rezultat este *mai complex* decît cel obținut anterior (vezi figura *P*-12). Nu este de neglijat nici faptul că topologia primului este mai simplă.

Deci, aplicînd procedee elementare, lucrînd cu *cărămizile cele mai mici* nu am reușit, chiar dacă am aplicat strict tehnicile de minimizare cele mai puternice, să obținem o structură atît de simplă ca cea desenată *direct* pe baza *observării* unor **funcții adecvate.**

Spre bucuria noastră *ingineria poate fi practicată și neformal, nealgoritmic*. Nu riscăm întotdeauna prea mult adoptînd soluții ad-hoc, inspirate de observații validate cel mult de experiența anterioară, sau încurajate de lipsa oricărei experiențe. Este totuși recomandabilă, de fiecare dată, și compararea cu soluții deduse formal care nu întotdeauna sînt mai complexe decît cele intuite direct. Lăsăm cititorului plăcerea de a pune în evidență unele avantaje pe care le totuși, soluția formală din exemplul dat.

Proiectarea unui sistem digital se face în majoritatea cazurilor alternînd etape ce decurg strict formal cu etape în care opțiunile se bazează pe criterii

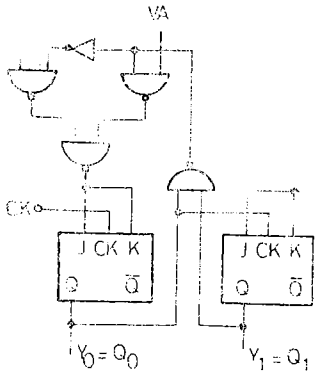


Fig. *P*-9. Schema automatului generator.

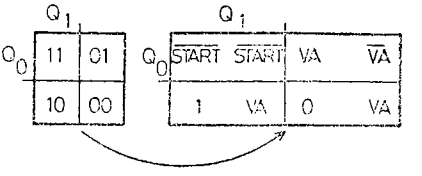


Fig. *P*-10. Diagrama de tranziție automatului generator.

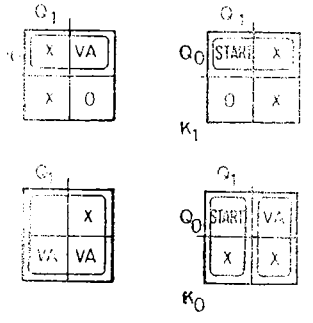


Fig. *P*-11. Detalierea diagrade tranziție pentru implementarea cu bistabile de tip *J-K*.

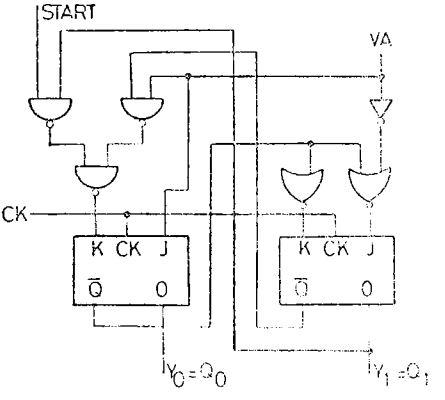


Fig. *P*-12. Soluția formală obținută pentru automatul generator.

greu sau imposibil de formalizat. Nu de puține ori adoptăm soluții în concordanță cu piesele de care dispunem în magazia de componente sau condiționate de tehnologiile la care avem acces. Astfel, de multe ori prima etapă în proiectare o constituie inspectarea conținutului magaziei de piese.

Soluțiile formale ne oferă întotdeauna *certitudinea* rezolvării problemei, ceea ce este foarte important. Varianta obținută formal, dacă ne mulțumește, poate fi reținută ; în caz contrar, ne putem aventura și în căutarea altora. Uneori, o parte dintre noi, vor avea noroc.

Cartea lui Thomas R. Blakeslee chiar dacă nu trece în revistă toate modurile de implementare are esențialul merit de a oferi cititorului și deschiderea către soluțiile neformalizate, inginerești. Citind această carte avem sentimentul reconfortant că ingineria rămîne în continuare o activitate ce mai presupune și ingeniozitate.

Capitolul 5 al cărții tratează despre circuite secvențiale la nivelul automatelor elementare. Sînt făcute cu această ocazie anumite opțiuni ce nu sînt suficient argumentate. În secțiunea 5.6 este amintită posibilitatea utilizării

registrelor în realizarea automatelor ca fiind o opțiune impusă de generarea unor secvențe mai complicate decît cele ce se pot obține folosind numărătoarele. De asemenea, în secțiunea 5.9 este justificată opțiunea pentru bistabilele de tip J-K pentru avantajele ce le prezintă față de numărătoare. Considerăm că imaginea ce și-o formulează cititorul privitor la opțiunile inițiale în declanșarea procesului de proiectare este prea limitativă, dacă nu chiar confuză. Din acest motiv simțim nevoia unei succinte prezentări a procedeelor de implementare a automatelor finite elementare.

Automate finite elementare sînt cele mai simple circuite secvențiale cu autonomie în evoluție. Formal sînt definite prin

$$A = (X, Y, Q, \lambda, \delta),$$

unde: X este mulțimea simbolurilor (configurațiilor binare) aplicate la intrare, Y este mulțimea configurațiilor de ieșire iar Q este mulțimea stărilor automatului; λ este funcția de tranziție a stării, de forma

$$\lambda : X \times Q \to Q,$$

iar δ este funcția de tranziție a ieșirii, definită prin

$$\delta : X \times Q \to Y.$$

(*a*) Automatul de tip *Mealy* este cel anterior definit. O altă definiție posibilă este cea dată de *Moore*, unde

$$A' = (X, Y, Q, \lambda, \eta)$$

primele patru elemente ale cvintuplului au semnificația anterioară iar funcția de tranziție a ieșirii η este de forma

$$\eta : Q \to Y$$

În aplicațiile concrete ambele forme sînt semnificative.

(*b*) În funcție de răspunsul automatului la ieșire față de modificarea intrării se mai pot face distincții. Automatele Mealy și Moore pot fi de tip *imediat* sau *cu întîrziere*. Astfel:

- $y(t) = \delta(x(t), q(t))$ pentru *Mealy imediat*,
- $y(t) = \delta(x(t-1), q(t-1))$ pentru *Mealy* cu întîrziere,
- $y(t) = \eta(q(t)) = \eta(\lambda(x(t-1), q(t-1)))$ pentru *Moore imediat*,
- $y(t) = \eta(q(t-1) = \eta(\lambda(x(t-2), q(t-2)))$ pentru *Moore cu întîrziere*.

Ieșirea automatului Mealy imediat corespunde intrării din ciclul (tactul) curent. Cea a automatelor Mealy cu întîrziere și Moore imediat reacționează la valoarea intrării din tactul anterior. Automatul Moore cu întîrziere răspunde la variația intrării cu o întîrziere de două tacturi. Aceste diferențieri pot fi mai ușor urmărite pe schemele de principiu reprezentate în figura *P*-13.

Nici una din cele patru variante nu este neglijabilă la nivelul gîndirii inițiale, de sistem, prin care se declanșează orice proces de proiectare.

(*c*) In cazul general registrele R din figura *P*-13 sînt formate din bistabile de tip D. Practica arată și teoria poate parțial justifica, faptul că înlocuirea bistabililor de tip D cu cei de tip J-K simplifică, în imensa majoritate a cazurilor, complexitatea circuitului logic combinațional asociat. Succint afirmația se poate justifica prin aceea că bucla suplimentară pe care o presupune bistabilul J-K față de cel de tip D preia o parte din efectul pe care-l are CLC-ul

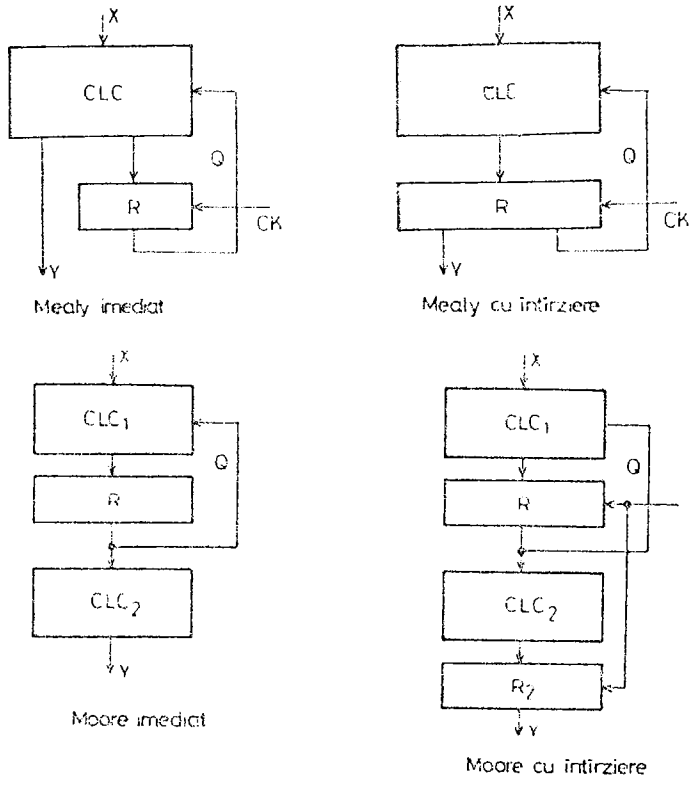


Fig. *P*-13. Tipuri de automate finite elementare.

din bucla automatului. Mai nuanţat, am putea spune că bistabilul J-K prin autonomia conferită de cea de a doua buclă de reacţie, permite o comandă mai puţin restrictiv condiţionată. Cunoscîndu-se starea la momentul t se poate comanda cea de la $t + 1$ prin configuraţii de tipul JK = 0X, JK = X1 ş.a. Apariţia X-urilor în definirea funcţiilor combinaţionale pe buclă permite o minimizare mai puternică. Să exemplificăm !

Exemplul 3. Vom implementa automatul generator descris în **Exemplul 2** utilizînd pe buclă un registru cu bistabili de tip D (un veritabil registru) de doi biţi. Pornind de la diagrama de tranziţie din figura *P*-10 se definesc diagramele asociate intrărilor celor două bistabile D (figura *P*-14), din care rezultă ecuaţiile :

$$D_1 = Q_1Q_0' + Q_1(START)' + Q_1'Q_0VA,$$
$$D_0 = Q_1Q_0(START) + Q_1'Q_0VA' + Q_0'VA.$$

Nu mai este cazul să desenăm noua variantă a automatului pentru a ne da seama că ne aflăm în faţa unei soluţii *net mai complexe* cu toate că implementăm două funcţii (D_1 şi D_0) faţă de patru (J_1, K_1, J_0 şi K_0).

(*d*) „Registrul" de JK-uri poate fi realizat conectînd în paralel mai multe astfel de bistabile. Nu este însă singura soluţie. Se poate concepe şi o conectare în serie (cascadă) a bistabilelor de tip JK astfel încît să formeze un *numărător presetabil*. Notăm ca deosebit de important faptul că *presetarea numără-*

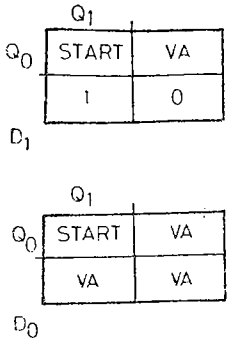


Fig. *P*-14. Diagramele asociate intrărilor celor două bistabile *D* din implementarea cu registru a automatului generator.

torului, folosit pentru implementarea unui automat elementar, *trebuie să fie comandată cu frontul impulsului şi nu cu palierul acestuia*. Din această perspectivă utilizarea, spre exemplu, a numărătorului 74193 nu este admisă deoarece funcţia de presetare acţionează direct asupra latch-urilor *slave* ale celulelor de numărare. După cum ştim pe bucla unui automat este obligatorie utilizarea unor bistabili acţionaţi pe frontul impulsului de tact.

Soluţia cu numărătoare se poate dovedi deosebit de eficientă în cazuri particulare în care o mare parte din stări pot fi astfel codificate încît tranziţiile să presupună incrementarea, fapt subliniat şi în carte.

Autonomia conferită de comutarea prin numărare permite definirea unor funcţii logice foarte simple pentru intrările de presetare, deoarece tranziţiile ce presupun incrementare implică *X*-uri în tabelele ce definesc aceste funcţii. Exemplul din secţiunea 5.11 este elocvent în acest sens.

Deci, în concluzie, referitor la automatele finite se recomandă folosirea registrelor de bistabile de tip *D* atunci cînd circuitul combinaţional asociat buclei este implementat prin procedee ce presupun structuri de tip ROM, DCD sau MUX, deci în cazurile în care nu se pune problema minimizării algebrice. Intotdeauna cînd se va pune problema minimizării logicii combinaţionale (PLA, porţi elementare), va fi preferată folosirea bistabilelor tip JK. Cuplarea acestora se poate face, în cazul general, în paralel, formînd „registre" de JK-uri. În cazul în care configuraţia particulară a grafului de tranziţie o permite, se va alege şi conectarea în serie a bistabililor JK sub forma numărătoarelor presetabile sincron. Am fi preferat detalii suplimentare privind procedee de alocare a stărilor, ce au fost prea sumar expuse în secţiunea 5.8. Recomandăm, pentru o abordare mai nuanţată, lucrarea lui *C. R. Clare*: *Designing logic systems using algorithmic states machines*, Mc Graw Hill, New York, 1972.

Există lucruri care nemarcate suficient devin generatoare de confuzii grave. Greu se poate găsi, în acest sens, o explicaţie lipsei distincţiei dintre comutarea condiţionată de frontul impulsului de ceas şi cea condiţionată de palierul impulsului de ceas.

T. Blakeslee nu marchează distincţia, pe care o considerăm esenţială, între un *latch* şi un *bistabil master-slave*. Ambele tipuri de circuite sînt considerate ca elemente de memorare. Dacă s-ar fi avut în vedere numai funcţia de memorare existenţa latch-urilor ar fi fost suficientă. O memorie de tip RAM este o colecţie de latch-uri organizată într-o matrice cît mai ordonată în care un element comută, memorează o nouă configuraţie binară, condiţionat de *palierul* unui impuls de comandă. Bistabilii de tip master-slave, cei ce sînt luaţi în consideraţie în majoritatea exemplelor date în lucrarea analizată, comută comandate de *frontul* unui impuls. Acest fapt este permis de o structurare mai avansată a circuitului şi este implicat de aplicaţiile concrete. Configuraţia reprezentată în figura 3-1 presupune o memorie realizată *obligatoriu* cu bistabili master-slave ce comută pe front dacă se are în vedere contextul tehnologic implicat de tratarea din carte. Nespecificarea acestui lucru,

măcar într-o secțiune ulterioară, generează confuzii esențiale în mintea cititorului neavizat. Memoria din figura 3-1 este obligatoriu un registru, nu poate fi concepută ca o colecție de latch-uri precum un RAM, dacă dorim o tranziție controlată a stărilor. În caz contrar, pe durata palierului activ se închide o buclă necontrolată prin circuitul logic combinațional, starea ulterioară a automatului (circuitului secvential) fiind determinată și de durata impulsului de comandă, ceea ce este lipsit de sens conform definiției anterior date pentru un astfel de circuit.

Prezentarea, măcar sumară, a configurației de tip master-slave s-ar fi impus cu necesitate. Cititorul poate compensa această lacună consultînd orice manual de circuite integrate digitale. Lectura acestei cărți se dovedește de mare utilitate, în primul rînd, pentru un cititor deja introdus în domeniul tratat, deoarece T. Blakeslee ilustrează în primul rînd un *mod de gîndire, un mod de abordare cu șanse de succes a problematicii aflate sub incidența soluțiilor cu circuite* MSI *sau* LSI. Nu de puține ori experiența concretă, de inginer, a autorului răzbate prin opțiunile și orientările date. Observăm acest lucru cu plăcere deoarece ne dă certitudinea că ne aflăm în fața unei cărți ce nu a fost scrisă numai după alte cărți sau articole. Ne aflăm mai puțin în fața unei sinteze teoretice cît în fața unei sinteze inginerești ce pornește de la o pregătire teoretică și o experiență practică. Cele trei capitole discutate sînt un fundament indispensabil pentru abordarea sistemelor digitale.

Deși tratarea este în primul rînd *coerentă*, neaspirînd, din considerente inginerești, la *rigoare*, este important și faptul că, și prin celelalte capitole ale cărții, ea se *închide* frumos, într-un mod puternic marcat de personalitatea autorului. Ne aflăm încă o dată în fața unui exemplu care ne arată că rigoarea depersonalizează iar coerența este un mod de potențare a personalității.

III. Logica programată

Un al treilea grup de capitole al cărții are următorul subtitlu : „Design techniques for the microcomputer age" (Tehnici de proiectare în era microcalculatoarelor). Autorul evidențiază astfel, la sfîrșitul deceniului opt, rolul central pe care-l deține microprocesorul în abordarea sistemelor digitale, marcînd o eră despre care astăzi încă putem spune că este departe de a apune chiar dacă pentru a supraviețui, microprocesorul a fost forțat la evoluții arhitecturale remarcabile. Pentru a se menține, microprocesorul tinde să devină continuu altceva și poate peste cîțiva ani vom fi puși în situația de a ne întreba : folosind microprocesorul de ultimul tip mai sîntem sau nu în era microprocesoarelor ? *Pînă a ajunge însă în situația de a putea pune astfel de întrebări trebuie să ne acomodăm cu tehnicile, devenite aproape tradiționale, de utilizare a microprocesoarelor din era lor de glorie.* Principiile de utilizare ale acestor sisteme se pot aprofunda foarte bine și pe exemplarele primelor generații care fac obiectul a patru capitole din carte.

Microprocesorul, ca **dispozitiv**, se împlinește în structura de **microcalculator.** Ce este însă un microcalculator ? Cu riscul de a face distincții pedante vom deosebi un microcalculator de un calculator folosind reprezentările din figurile *P*-15 și *P*-16. Microprocesorul înglobează controlul și dispozitivele efectorii (registre plus ALU), lăsînd în exterior numai sisteme cu o structură foarte uniformă : memoriile. Ceea ce este *extensibil* într-un sistem de calcul —

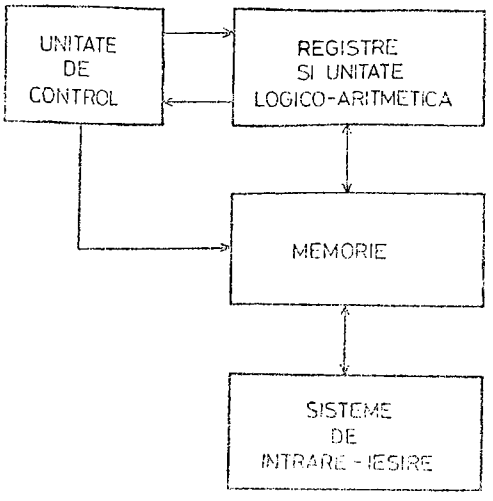


Fig. P-15. Structură de calculator.

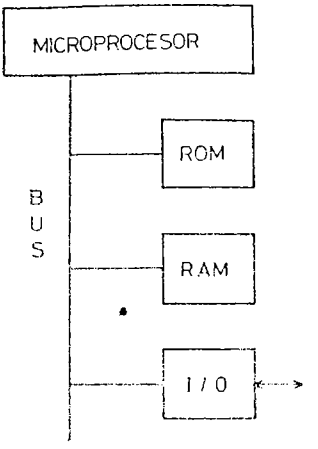


Fig. P-16. Structură de microcalculator.

partea de memorie, a fost net separat de ceea ce este *fix* — microprocesorul. O altă distincție, nu absolută, dar curentă, este dată de faptul că microprocesorul controlează direct și echipamentele de intrare-ieșire. De asemenea specific în aplicațiile curente, drept consecință și a faptului anterior menționat, este apariția distinctă a unei zone de memorie fixă (ROM). Deoarece inițializarea sistemului trebuie făcută tot sub controlul microprocesorului, este obligatorie și prezența unui program de inițializare (rezident în memoria fixă) la alimentarea microcalculatorului.

Dacă structura de calculator are întotdeauna asociată funcția de calculator, în cazul microcalculatoarelor funcția asociată structurii poate fi și alta. S-au încetățenit două clase de funcțiuni : cea clasică, de **echipament de calcul** și o a doua pentru care se utilizează termenul de **logică programată.** În textul la care ne referim, microcalculatoarele sînt abordate în această ultimă accepțiune.

Un calculator acceptă la intrare, de regulă, șiruri de simboluri alfanumerice, pe cînd un sistem cu logică programată acceptă șiruri de simboluri binare cu semnificație independentă. La intrarea unui calculator apar *coduri* iar la intrarea unui sistem cu logică programată se aplică *biți independenți.* Similar se petrec lucrurile la ieșire. Evoluția unui calculator este numai declanșată de un șir de simboluri aplicate intrării dar se *desfășoară puternic controlat de programele interne* ; pe intervale de timp foarte mari evoluția intrării este ignorată. Spre deosebire de un sistem de calcul, unul ce rezolvă o problemă de logică programată *evoluează puternic condiționat și de semnalele de intrare.* Modificarea semnalelor binare aplicate pe intrare este cît mai fidel urmărită de evoluția stării microcalculatorului aflat sub controlul programului intern. Ieșirea este dată atît de continua evoluție a intrărilor cît și de modul în care acestea sînt interpretate de programul rulat. Ieșirile unui sistem cu logică programată sînt *continuu semnificative* spre deosebire de cele ale unei structuri cu funcție de calculator în care ieșirile sînt semnificative numai în intervale de timp limitate.

Un sistem de calcul poate lucra uneori în timp real, pe cînd un sistem cu logică programată este prin definiție unul ce funcționează în timp real.

Avantajul principal al sistemelor cu logică programată este dat de maxima lor flexibilitate. Dezavantajul cel mai important este dat de timpul de răspuns foarte mare în comparație cu acela al sistemelor combinaționale sau secvențiale clasice. Aceste caracteristici limită constituie și criteriile cele mai importante ce se iau în considerație atunci cînd se optează pentru rezolvarea unei probleme folosind logica programată.

Utilizarea logicii programate presupune o structură fizică universală ce se actualizează pentru diverse funcții particulare, printr-o structură informațională : *programul.*

Conceperea, punerea la punct și întreținerea programelor presupun instrumente dedicate.

În această carte după un capitol ce definește conceptul de microcalculator și specifică modul său de utilizare într-un sistem cu logică programată, urmează alte două capitole ce ne familiarizează cu instrumentele destinate actualizării funcționale prin programare. Unul se ocupă cu instrumentele software utilizate, cum ar fi asambloarele, macroasambloarele, compilatoarele, sistemele de operare și mașinile virtuale, iar altul definește și prezintă facilitățile oferite de sistemele de dezvoltare, care, spre deosebire de primele unelte, sînt niște sisteme fizice programabile, puternic orientate către aplicații de logică programată. Ultimul capitol din secțiunea dedicată logicii programate se ocupă de structura fizică a unui microcalculator și de criteriile de proiectare ce se impun pentru astfel de sisteme.

Așa cum este normal, *structura fizică trece în acest caz pe un plan secund,* datorită tipizării avansate impuse de utilizarea microprocesoarelor într-un ansamblu în care se mai adaugă numai configurații simple, aproape standardizate, cum ar fi memoriile și circuitele de interfațare. Majoritatea efortului de proiectare și de punere la punct este concentrat în *definirea algoritmilor și în programare.* Acest lucru se reflectă și în prețul proiectării, care se poate minimiza în primul rînd prin utilizarea unor instrumente software adecvate și a unor sisteme de dezvoltare performante care să reducă drastic efortul de programare. Cei ce debutează în utilizarea logicii programate au, uneori, tendința de a minimaliza rolul instrumentelor de dezvoltare, adoptînd strategii de implementare ce presupun o investiție cît mai mică în programe și sisteme de dezvoltare. Chiar dacă în realizarea unui proiect se asigură un start mai rapid, finalizarea va fi cu certitudine întîrziată de alegerea unor căi ce nu adoptă instrumente de dezvoltare performante. O astfel de atitudine se manifestă mai des la inginerii orientați spre hardware, care, confruntați cu această nouă metodă de abordare, nu apreciază just rolul instrumentelor ce vor facilita finalizarea software-ului presupus de logica programată. Deci atenție la stil !

În economia lucrării un spațiu foarte redus este acordat *microprogramării.* Este adevărat că această tehnică este specifică în primul rînd tehnologiilor de realizare a calculatoarelor, de care lucrarea de față nu se ocupă. Nu poate fi justificat însă modul în care este încadrată în lucrare prezentarea acestei modalități de implementare a unui sistem logic. Microprogramarea este prezentată ca un caz particular de programare într-o variantă mai eficientă și pe o structură dedicată de tip calculator. Imaginea pe care o obține cititorul referitor la microprogramare și sistemele microprogramate este în cel mai bun

caz trunchiată, dacă nu chiar confuză. Cîteva pagini de text și două scheme, prea complexe pentru a oferi exemple sugestive, nu pot acoperi un domeniu atît de vast ca acela al microprogramării. Carența menționată este parțial compensată de o bună bibliografie referitoare la acest domeniu.

Logica programată constituie o etapă tranzitorie în evoluția tehnicilor de implementare a sistemelor digitale. Într-o primă etapă permite o *integrare relativ puternică* pentru o *imensă diversitate de funcții* utilizînd structuri ce tind către *configurații fizice uniforme*. Din start apar totuși deficiențe de neevitat. Cea mai importantă este adîncirea procesului de *secvențare* a procesării. Cu un circuit logic combinațional funcția *ȘI* de trei variabile se realiza în absența oricărei secvențări, într-un interval de timp nesemnificativ. Utilizînd logica programată aceeași funcție presupune o amplă secvență ce se materializează într-un program care activează o structură fizică de mii de ori mai complexă decît o poartă cu trei intrări.

Deceniul opt a impus microprocesorul care se află în plină glorie și în deceniul nouă în care au apărut, însă, și alternative tentante la logica programată. Este vorba de tehnologiile ce permit **custom design** (proiectarea la cererea clientului) sau **semi-custom design.** Avantajul folosirii unei structuri fizice, unice, actualizabilă funcțional prin programare începe să nu mai fie esențial atunci cînd industria pune la dispoziția utilizatorului facilități prin care-și poate realiza, într-o structură dedicată, în limite suficient de largi, orice funcție.

De multă vreme *paralelismul* este o tentație nedisimulată de cei ce gîndesc în domeniul structurilor digitale. În astfel de condiții stimularea *secvențialului*, și prin logica programată, este din ce în ce mai greu de justificat. Secvențialul, prin forma sa cea mai exacerbată, programul, își atinge limite ce sperăm că nu vor fi depășite. Logica programată apare astfel ca un ultim pas pe un drum ce trebuia parcurs pînă la capăt pentru a-și epuiza semnificația și sensul. Nu este suficient ca limitele să fie întrevăzute, ele trebuie să fie atinse ; acesta pare a fi unul din resorturile ascunse ale logicii programate. Parafrazînd un gînd al lui E. Canetti am putea spune că : *„Nu se poate scăpa dintr-o dată de un* **dispozitiv** ajuns *periculos. Trebuie mai întîi să te chinuiești îndelung folosindu-l anapoda."*

Pentru o mare varietate de aplicații, în condiții în care nu există acces la tehnologiile cele mai avansate, tehnica de implementare pe care o comentăm se dovedește încă deosebit de utilă.

Microprocesoarele, îmbogățite cu facilități arhitecturale din ce în ce mai sofisticate, vor tinde către utilizarea exclusivă în sistemele de calcul, nemaifiind tentate de accesul deosebit de facil pe care-l au la semnalele de intrare-ieșire (figura *P*-15), să-și aservească funcționarea la evoluția unor semnale cu o încărcătură semantică minimală, așa cum se întîmplă în sistemele cu logică programată. Deja microprocesoarele ultimelor generații sînt atît de diferite arhitectural de bunicul lor, microprocesorul 8080 — pentru a folosi o expresie a lui T. Blakeslee — încît ne îndoim că mai pot fi astfel numite.

Microprocesoarele de 32 de biți lucrează în configurații de sistem atît de sofisticate încît cu dificultate mai pot fi încadrate într-una din cele două figuri prin care am definit distincția dintre un calculator și un microcalculator. Un astfel de dispozitiv greu, credem, că va putea fi utilizat într-un sistem cu logică programată.

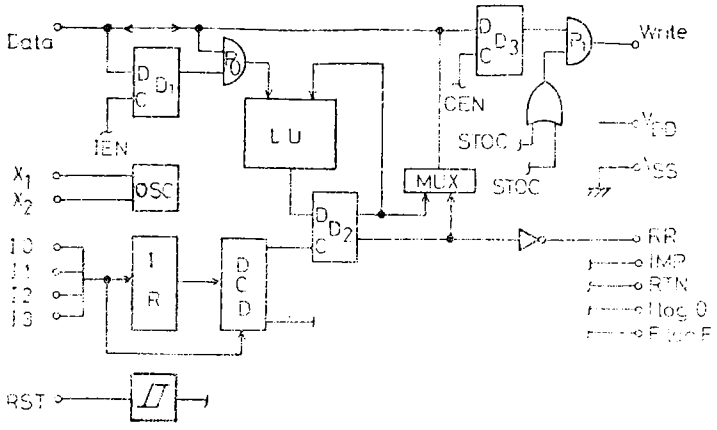


Fig. *P*-17. Structura internă a *Unității de Control Industrial MC* 14 500 *(Industrial Control Unit).*

Pentru logica programată se impun însă structuri mai simple care justifică utilizarea lor datorită unei relații mai judicioase între complexitatea structurală și amploarea programelor. Un exemplu interesant este cipul **Industrial Control Unit** — UNITATE DE CONTROL INDUSTRIAL (MC 14500 — MOTOROLA, produs la *IPRS — BĂNEASA* sub indicativul βP14500 și la *MICROELECTRONICA* cu indicativul MMC4500) ce se poate constitui într-un **microprocesor de un bit.** O astfel de structură se adaptează mai firesc procesării pe bit decît un microprocesor de 8, 16 sau 32 de biți. Într-un microprocesor standard apare dificultatea operării cu bitul într-o structură dedicată operării asupra cuvîntului de 8 sau mai multi biți. Avantajul utilizării structurii de tip MC14500 pentru a realiza aplicații de logică programată constă tocmai în orientarea acesteia către operarea pe bit.

În figura *P*-17 este reprezentată schema internă a acestui circuit iar în anexă (vezi pagina 358) se poate consulta o descriere detaliată a funcționării.

Pentru accesul la mai mulți biți pe intrarea de date este adusă, în aplicații curente, ieșirea unui multiplexor ce permite selectarea bitului dorit. De asemenea pentru a se distribui semnale pe mai multe căi este folosit un demultiplexor cu ieșirile memorate (latch adresabil).

Astfel, structura efectorie de principiu, curent realizată cu acest circuit, se prezintă ca în figura *P*-18, pentru un sistem cu logică programată

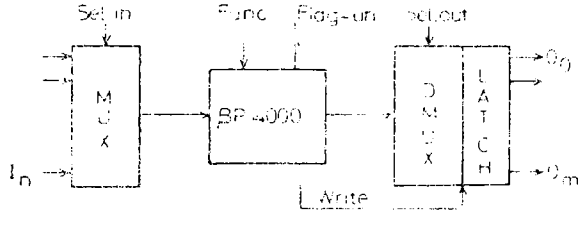


Fig. *P*-18. Interconectarea cu exteriorul a circuitului *MC* 14 500.

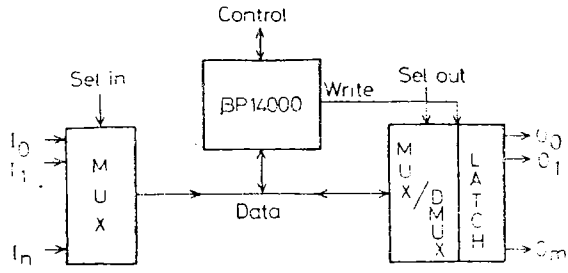


Fig. *P*-19. Variantă de interconectare cu *BUS* intern.

cu $n + 1$ intrări și $m + 1$ ieșiri. Operarea pe bit este *firească* într-o astfel de configurație iar numărul intrărilor și al ieșirilor este independent de microprocesorul MC 14500. O parte din latch-urile de ieșire pot fi utilizate ca memorii temporare deoarece, pentru date, în microprocesor nu există decît un registru de un bit. În acest caz devine interesantă configurația din figura *P*-19 în care DMUX-ul a fost înlocuit cu un MUX—DMUX ce permite și citirea conținutului latch-urilor de la ieșire, care sînt folosite în consecință și ca locații de memorie. Astfel MC 14500 devine componenta centrală a unei familii de circuite ce mai conține un multiplexor 8 : 1 cu ieșire tristate (MC 14512) și un multiplexor/demultiplexor 1 : 8 cu memorie (MC 14599).

Pentru a exemplifica modul de utilizare a familiei descrise vom arăta cum se realizează funcția logică dată în figura *P*-20. Secvența de comenzi executată va fi următoarea (pentru mnemonicele utilizate se va consulta anexa) :

LD	*A*	/*RR*	← *A*/	
AND	*B*	/*RR*	← *AB*/	
STO	*TEMP*	/Latchul	*TEMP*	← *AB*/
LD	*C*	/*RR*	← *C*/	
ANDC	*D*	/*RR*	← *CD'*/	
OR	*TEMP*	/*RR*	← *AB* + *CD'*/	
AND	*E*	/*RR*	← (*AB* + *CD'*)*E*	
ANDC	*F*	/*RR*	← (*AB* + *CD'*)*EF*/	
STO	*LOAD*	/Latchul	*LOAD*	← rezultatul/.

Se poate observa că majoritatea operațiilor efectuate în microprocesor au fost strict legate de funcția logică executată.

Comparînd acest program cu unul scris pentru un microprocesor standard vom observa absența „balastului" datorat structurii neadecvate a unui microcalculator uzual. În aplicația prezentată *accesul direct la bit* este cel ce a conferit simplitate și limpezime programului.

Controlul structurii din figura *P*-19 poate fi foarte simplu făcut cu un CROM elementar și prezentăm în figura *P*-21 o variantă. Sistemul funcționează cu același ceas ca și microprocesorul, comenzile fiind date de ieșirile

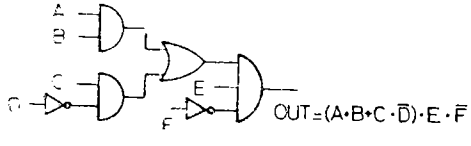


Fig. *P*-20. Exemplu de funcție logică.

de tip „flag" din circuitul de bază. Numărătorul utilizat permite încărcarea comandată de frontul impulsului de ceas. Registrul R se încarcă numai în cazul saltului la subrutină. Patru din ieșirile ROM-ului comandă direct funcția de executat în ciclul curent, iar celelalte opt au dublă semnificație: constituie adresă de salt (în cazul cînd se efectuează o astfel de operație sub comanda flag-urilor microprocesorului) sau pot fi folosiți pentru intrările de selecție ale circuitelor ce asigură intrarea și ieșirea din sistem. Deoarece funcția de salt nu presupune și operarea pe calea de date, nu vom avea niciodată nevoie simultan de adresă și de coduri de selecție a circuitelor de intrare-ieșire.

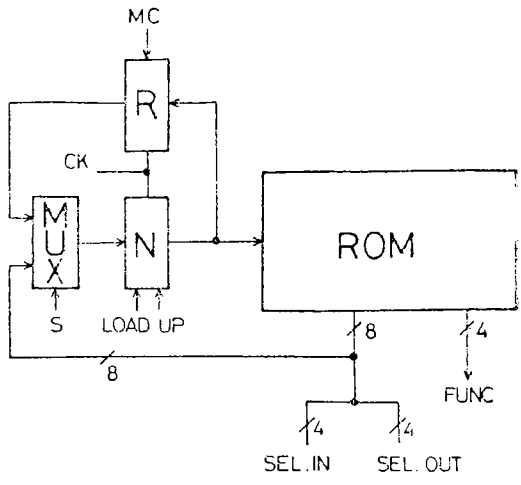


Fig. *P*-21. Controler pentru unitatea de control industrial *MC* 14 500.

Configurația de control prezentată ne permite să lucrăm la viteză maximă cu microprocesorul de un bit datorită faptului că se pot concepe programe cu un număr minim de pași. Prețul plătit este o memorie fixă cu cuvinte de 12 biți. Dacă ne permitem să reducem viteza de operare vom putea micșora și ROM-ul pînă la unul cu un cuvînt de patru biți. În acest caz vor apare pași suplimentari pentru selecția intrărilor și a ieșirilor sau pentru generarea adresei de salt. Vom opta pentru o soluție sau alta în funcție de performanțele de viteză cerute în aplicația concretă.

Putem deci formula, fără riscuri prea mari, concluzia că pentru logica programată arhitecturile orientate pe bit se pot dovedi cele mai performante; structuri de tipul celei descrise anterior sînt gîndite din start pentru astfel de aplicații, pe cînd microprocesoarele sînt adaptate, în configurații nu întotdeauna cele mai firești, să suporte aplicații de logică programată.

O soluție intermediară pentru logica programată o oferă *microcalculatoarele „one-chip"*, care vin cu avantajul unei structuri foarte compacte. Un astfel de circuit are structura din figura *P*-16, dar fiind implementat pe o singură pastilă de siliciu apar o serie de restricții cantitative. Astfel, în realizările curente, memoria *RAM* are în jur de 64 ÷ 256 de locații, memoria *ROM* este de 1 ÷ 4 kcuvinte iar interfețele sînt cel mult în număr de trei. Memoria ROM poate fi de tip EPROM sau REPROM pentru realizarea prototipurilor.

Structura fizică deosebit de compactă compensează deficiențele arhitecturale menționate anterior propulsînd acest tip de circuit în domeniul aplicațiilor de logică programată, și nu numai în acest domeniu.

O variantă curentă de microcalculator one-chip este reprezentată în figura *P*-22. Cele trei porturi de intrare-ieșire (I/OA, I/OB, I/OC) împreună cu semnalele de sincronizare (SA, SB, SC) pot fi conectate în exterior în cele mai diferite moduri, obținîndu-se un sistem cu un număr oarecare de intrări

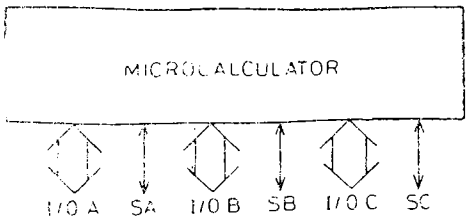


Fig. P-22. Microcalculator one-chip.

ce determină evoluția unui alt grup de ieșiri, în funcție de programul stocat în memoria internă de tip ROM. Memoria RAM de care dispune sistemul va fi utilizată exclusiv pentru stocări temporare de date ; ea nu va conține programe, în aplicațiile de logică programată.

Atunci cînd cu *un singur cip* putem să rezolvăm clase imense de probleme în tehnica logicii programate, ne permitem să nu mai facem obiecții teoretice subtile referitor la neadecvarea dintre arhitectura utilizată și funcția realizată. Atitudinea *pragmatică* este cea mai firească într-un asemenea caz, cu condiția ca să avem acces la tehnologia ce permite o astfel de abordare.

Logica programată cu tot nefirescul ei, ce se manifestă chiar în termeni ca o discrepanță între scop (logică) și modul de realizare (program), va mai rămîne o bună bucată de vreme o tehnică de neînlocuit într-o gamă foarte largă de aplicații. Sistemele tehnice pe care dorim să le automatizăm sînt, în imensa majoritate a cazurilor, suficient de lente pentru ca secvențarea excesivă, presupusă de logica programată, să nu constituie un motiv de limitare a performanțelor.

ANEXĂ

UNITATE DE CONTROL INDUSTRIAL *MC14500* — *microprocesor de un bit*

Schema de principiu a circuitului MC14500 a fost reprezentată în figura *P*-17 din text. Pentru a specifica modul de lucru al circuitului explicităm pe scurt funcțiile blocurilor interne, după cum urmează :

- IR — registru de instrucțiuni — format din patru bistabili master-slave de tip *D* ; la fiecare impuls de ceas se încarcă cu codul aplicat pe intrările I0 ... I3 ;
- DCD — este decodificatorul de instrucțiuni care în funcție de conținutul registrului IR uneori de intrările I0 ... I3 și în funcție de ceas, generează semnale interne de comandă (cum ar fi : IEN, STO ș.a.) sau semnale la bornele cipului (ex. : JMP, Flag O ș.a.) ;
- LU — unitatea logică — ce aplică bitului de la intrare și celui din D_2 funcția logică indicată în IR.
- D_1 — este un bistabil de tip *D* ce condiționează prin poarta P_6 accesul datelor din exterior la LU ;
- D_2 — este registrul pentru rezultat (RR) în care se înscrie ieșirea unității LU, dacă a fost efectuată o funcție logică sau una de intrare ;
- D_3 — bistabil ce condiționează generarea prin P_1 a semnalului *write* atunci cînd se emite conținutul lui D_2 pe calea RR ;
- MUX permite accesul în exterior, pe calea Data, a valorii adevărate sau complementate din RR.

Semnalele JMP, RTN, Flag O și Flag F sînt folosite pentru a comanda controlerul (o variantă a acestuia este prezentată în figura P-21). Semnalul *write* este utilizat pentru scrierea în latch-urile de ieșire. Intrarea RST permite resetarea întregului circuit, iar X1 și X2 sînt folosite pentru timing. După cum se observă circuitul are o capsulă cu 16 conexiuni.

Listăm în continuare cele 16 instrucțiuni prin mnemonică, cod și mod de acțiune :

- NOPO — 0000 — activează ieșirea *Flag* 0
- LD — 0001 — încarcă registrul RR : $Data \rightarrow RR$
- LDC — 0010 — încarcă complementul : $\overline{Data} \rightarrow RR$
- AND — 0011 — funcția AND : $RR\ Data \rightarrow RR$
- ANDC — 0100 — funcția AND cu complementul : $RR\ \overline{Data} \rightarrow RR$
- OR — 0101 — funcția OR : $RR + Data \rightarrow RR$
- ORC — 0110 — funcția OR cu complementul : $RR + \overline{Data} \rightarrow RR$
- XNOR — 0111 — SAU EXCLUSIV. Dacă $RR = Data$, $1 \rightarrow RR$
- STO — 1000 — stocare : $RR \rightarrow Data$, activ *write*
- STOC — 1001 — stocare complement : $\overline{RR} \rightarrow Data$, activ *write*
- IEN — 1010 — încarcă D_1 : $Data \rightarrow$ registrul D_1
- OEN — 1011 — încarcă D_3 : $Data \rightarrow$ registrul D_3
- JMP — 1100 — salt. Activează ieșirea *JMP*
- RTN — 1101 — reintoarcere din subrutină. Activează *RTN*
- SKZ — 1110 — sare următoarea instrucțiune dacă $RR = 0$
- NOPF — 1111 — activează ieșirea *Flag F*.

IV. Dimensiunea timp — dimensiunea spațiu

Funcțiile unui sistem digital se organizează în coordonate structurale pe două axe principale : una a *timpului* și cea de a doua a *spațiului*. O funcție este realizată printr-o *secvență* ce activează o anumită *structură* fizică. Procesarea este distribuită în timp, în spațiul oferit de resursele sistemului digital. Dacă o funcție poate fi distribuită atunci acest lucru se poate petrece în două modalități limită ; una presupune distribuirea în timp a unei resurse unice, iar cealaltă distribuirea în spațiul oferit de o structură multiplicată identic.

Capitolul *„The time dimension"* din cartea lui T. R. Blakeslee a fost gîndit de autor astfel încît să ne putem face o imagine despre dezvoltările posibile pe axa timpului în condiții în care se impune minimizarea expansiunii în spațiu. În acest sens, capitolul face o bună propagandă ideii de a minimiza structura fizică a unui sistem în detrimentul performanțelor legate de timpul de execuție. De multe ori timpul de execuție maxim admis de o operație este mult mai mare decît cel pe care-l reclamă pentru execuție un circuit standard. Atunci cînd dispunem de timp este normal să-l folosim pentru a cîștiga spațiu prin reducerea structurii fizice. Nu aceasta este însă situația în toate aplicațiile. De multe ori timpul este o mărime critică ce poate fi satisfăcută numai plătind cu spațiu. Există deci și o expandare pe coordonata spațiului de care lucrarea în discuție nu se ocupă. Tehnologiile curente în anii '70 nu permiteau accesul comod al configurații spațiale extinse.

În a doua jumătate a deceniului nouă, *prelucrarea distribuită (în spațiu) tinde să devină o tehnică uzuală*. Din acest motiv, fără a intra în detalii, simțim nevoia unor comentarii referitoare la aceste sisteme.

Există două categorii de sisteme pentru prelucrarea distribuită. O primă categorie presupune multiplicarea unor structuri de prelucrare controlate informațional, tipic reprezentată prin configurații *multiprocesor*. Cea de a doua categorie, dezvoltată mai recent extinde spațial, în rețele cu cele mai diferite forme, elemente de prelucrare care nu presupun un control informațional local. În această categorie intră *sistemele sistolice* de care ne vom ocupa în continuare sumar, ca o alternativă opțională la sistemele distribuite în timp.

Sistemele sistolice au fost introduse, la sfîrșitul deceniului trecut, de H. T. Kung și C. E. Leiserson de la Universitatea Carnegie-Mellon (H. T. Kung: *Let's design algorithms for VLSI sistems*, Proceedings of the Caltech Conference on Very Large Scale Integrations, Ianuary 1979; H. T. Kung, C. E. Leiserson: *Systolic arrays (for VLSI)*, în Sparse Matrix Proceedings 1978, din 1979, o versiune ulterioară fiind publicată și în C. A. Mead, L. A. Conway: *Introduction to VLSI Systems*, Addison—Wesley, Reading Mass, 1980).

Ideea care stă la baza introducerii sistemelor sistolice este prezentată în figura *P*-23, unde, unei configurații clasice (*a*), îi este opusă una (*b*) ce presupune mai multe elemente de prelucrare, *P*, conectate în cascadă (**pipeline**). Dacă un ciclu complet de citire-prelucrare-scriere se poate efectua, spre exemplu, în 200 ns, configurația clasică va realiza 5 milioane de operații pe secundă iar cea sistolică 25 milioane de operații pe secundă, *cu condiția ca algoritmul de prelucrare să permită o prelucrare sistolică*. Nu orice problemă își poate găsi o abordare sistolică. Așa cum nu orice funcție poate fi distribuită în timp, tot așa nu este garantată oricînd distribuția în spațiu. Sîntem în etapa în care sistemele sistolice nu garantează soluții universale. Această categorie de sisteme este, și poate va rămîne pentru totdeauna, orientată pe funcțiuni particulare pe care le execută foarte rapid. Din acest motiv astfel de structuri sînt integrate, pentru început măcar, în sisteme de uz general pentru care *realizează performant funcții critice*.

În figura *P*-24 este prezentat modul de conectare a unui sistem sistolic într-o mașină universală clasică. Sistemul sistolic, atașat unui calculator standard, poate realiza, spre exemplu, funcția de transformator Fourier la o viteză cu cîteva ordine de mărime mai mare decît ar reuși-o sistemul în absența lui. Dacă această funcție este importantă în aplicațiile cărora sistemul este dedicat, prezența rețelei sistolice este justificată.

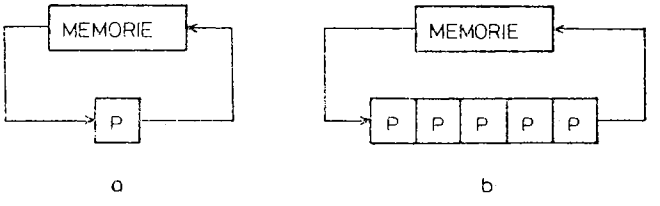


Fig. *P*-23. Ideea de sistem sistolic.

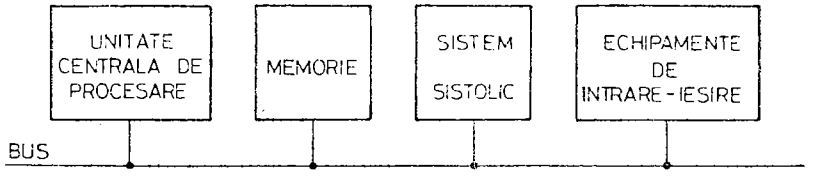


Fig. *P*-24. Integrarea unui sistem sistolic într-o configurație clasică de calculator.

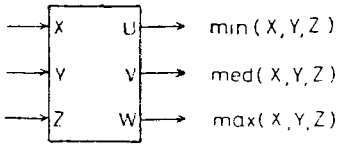


Fig. *P*-25. Sortator elementar.

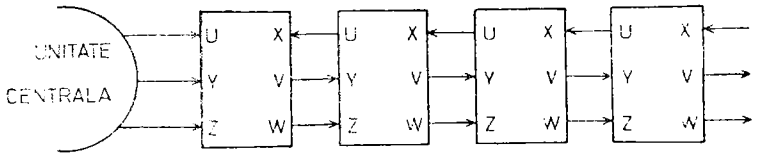


Fig. *P*-26. Realizarea cozii prioritare.

Să exemplificăm ideea prin realizarea hardware a unei *cozi prioritare* **(priority queue)** în variantă sistolică (după C. E. Leiserson : *Systolic priority queues*, în Proceedings of Caltech Conference on VLSI, California Institute of Technology, Pasadena, Ianuary 1979). Elementul de prelucrare elementar este sortatorul (tree-sorter) reprezentat în figura *P*-25. El este constituit din elemente de memorie și circuite logice combinaționale, asigurînd stocarea mărimilor de intrare/ieșire și compararea lor pentru ordonarea impusă. Interconectarea sortatoarelor se realizează conform reprezentării din figura *P*-26, astfel încît spre ieșire (U) să migreze valorile minime iar în profunzimea structurii cele medii și maxime. Se asigură astfel la extragere ordonarea simbolurilor numerice înserate într-o ordine oarecare.

Pentru funcția *INSERT* se aplică la intrarea Y a primului sortator numărul ce se dorește inserat iar la intrarea Z numărul $-\infty$ (cel mai mic număr presupus de reprezentarea din sistem). După un ciclu, $-\infty$ apare la ieșirea U iar numărul se propagă înspre interiorul sistemului căutînd să ocupe poziția ce i se cuvine în șirul deja prezent în sistem. Se va propaga, în ciclurile următoare, cu atît mai adînc cu cît este mai mare.

În cazul funcției *EXTRACT MIN* se aplică la intrările Y și Z simbolul ∞ (cel mai mare număr permis la reprezentarea din sistem) ; drept urmare simbolul numeric de la intrarea X a primului sortator va fi transferat la ieșirea U. Ținînd cont de modul cum a fost realizată inserția acest simbol este de valoare minimă relativ la toate simbolurile din șir. Inițializarea sistemului presupune introducerea numărului ∞ în toate locațiile.

Sisteme sistolice au fost definite pentru un număr foarte mare de funcții ș printre acestea citățm : filtrare digitală, convoluție, transformare Fourier, multiplicări de matrici, recunoașteri de limbaje, baze de date relaționale (o bună bibliografie este disponibilă în H. T. Kung : *Why Systolic Architectures ?*, în Computer, Ianuary, 1982).

V. Tehnologii VLSI și abordarea funcțională

Deceniul care a urmat ultimei ediții a acestei cărți, cel de al noulea, a fost marcat de apariția și impunerea tehnologiilor *VLSI* **(Very Large Scale Integration).** Acest eveniment a avut drept consecință trecerea *abordării*

sistemice într-un plan secund. În locul acesteia se impune *gîndirea funcțională*. Dacă tehnologiile *LSI* erau perfect servite de gîndirea sistemică, *posibilitățile tehnologiilor VLSI pot fi puse în valoare numai pornind de la o abordare funcțională*, care nu neglijează sistemicul pe care se sprijină în continuare ca pe un procedeu valabil, dar numai într-o a doua instanță.

În dezvoltarea unei tehnologii există un joc, cu reguli aproape clare, între *structură* și *funcție*. Tehnologia *oferă* structuri și trebuie să *satisfacă* funcțiuni. Startul este dat de o sumă de cerințe funcționale în virtutea cărora sînt adoptate structuri preliminare. Pot urma însă etape în care evoluția este dictată de considerente preponderent structurale, statuîndu-se un cadru funcțional rigidizat. Tendința de a *lărgi* cadrul funcțional nu mai este direct sprijinită de tehnologie, care este „obsedată" de *perfecționările* structurale.

Tehnologia tinde să fixeze *interfețe* la limita dintre abordarea structurală și cea funcțională, interfețe care permit evoluția nestingherită a tehnologiei. Un cadru cu o dinamică funcțională accentuată nu este propice perfecționărilor tehnologice. Nu se poate răspunde performant la *două* întrebări deodată : *ce* și *cum* să facem ?

În domeniul structurilor digitale *interfața* a fost, și este încă în mare măsură, dominată de conceptul de *mașină universală*. Este vorba de *structura de calculator* care poate fi actualizată funcțional prin programare. (Structura de calculator, cum menționam și anterior, nu este întotdeauna destinată funcției de calculator.)

Componentele esențiale ale mașinii universale menționate sînt *procesorul* și *memoria*. Cu modificări neesențiale aceste două structuri sînt definite de aproximativ o jumătate de secol. Cadrul funcțional oferit de *mașina Turing* și *calculatorul de tip von Neumann* a fost conservat decenii de-a rîndul pentru ca implementările concrete să poată fi perfecționate și, în același timp, să se poată dezvolta și generaliza aplicații în domenii cît mai largi.

Investițiile tehnologice cele mai ample au sprijinit dezvoltarea a două componente : microprocesorul și memoria. Progresele au fost atît de mari încît au generat o serie întreagă de incompatibilități. Nu este cazul să dezvoltăm în acest text o argumentare amplă, vom menționa numai faptul că la sfîrșitul deceniului nouă dispunem de posibilități tehnologice ce depășesc capacitatea noastră de a le folosi la adevărata lor valoare datorită insuficiențelor în abordarea funcțională. Evoluția explozivă sub presiunea cerințelor *pur structurale* ne pune la dispoziție mijloace cu care nu știm prea bine ce să facem în primul rînd pentru că putem să facem foarte multe. Sîntem profund dezorientați în plan funcțional.

O soluție ar putea fi să vedem ce s-a realizat deasupra interfeței, deci în domeniul programării mașinii universale, să preluăm în hardware funcțiuni software. O astfel de abordare se dovedește însă simplistă, în primul rînd datorită faptului că funcțiunile software s-au dezvoltat restricționate de mașina universală, deci ele nu constituie exact ceea ce ne trebuie. Multe tehnici software s-au dezvoltat pentru a compensa insuficiențele conceptului înghețat de calculator. Toate acestea constituie acum un *balast teoretic* și *funcțional*.

Nu putem concepe funcțiuni noi rămînînd sub influenta vechii interfețe. Cu toate că se practică curent, migrarea către hardware a funcțiilor din software, așa cum au fost gîndite pentru mașina universală, nu este decît un paleativ.

Soluția trebuie să fie mai radicală. Trebuie regîndite modelele inițiale și realizată o *interfață mai flexibilă*. Ne vom putea sprijini, în acest sens, pe conceptul de *arhitectură*. Vechea interfață era și ea o arhitectură (setul de instrucțiuni al unui calculator, spre exemplu) dar era una fixă, imuabilă la nivelul căreia nu se petrecea nimic esențial. Acum motorul evoluției trebuie să acționeze la acest palier, cel al arhitecturii.

Vom adopta pentru arhitectură definiția conform căreia reprezintă un *ansamblu de funcțiuni definit la o interfață** între două sisteme, între un sistem și om, societate sau natură. Abordarea arhitecturală nu impune restricții structurale, fapt pentru care gîndirea sistemică își păstrează domeniul ei, în spatele celei arhitecturale căreia îi urmează, cu criterii și tehnici specifice.

Dacă nu impune restricții structurale, totuși gîndirea arhitecturală presupune, pentru a se putea desfășura, libertăți structurale maxime. Tehnologia este dispusă să adopte abordări arhitecturale (funcționale) numai atunci cînd este suficient consolidată în privința posibilităților structurale.

Apariția tehnologiilor VLSI marchează momentul în care electronica este capabilă să suporte o abordare funcțională, lăsîndu-se dominată de gîndirea arhitecturală care pornește *de la ceea ce trebuie să facă electronica și nu de la ceea ce poate să facă electronica*. Trebuința se definește funcțional iar putința este limitată structural. Gîndirea funcțională marchează eliberarea, pentru un interval de timp imprevizibil, de sub imperiul evoluției structurale.

Acest proces, apărut în anii '80, va deveni dominant în ultimul deceniu al mileniului. Este semnificativ în acest sens că depășirea, în curs, a generației a patra de calculatoare, se face, în primul rînd, nu printr-o evoluție structurală (o nouă tehnologie) ci printr-o nouă manieră de a gîndi interfața dintre calculator și utilizator. La interfața dintre om și calculator se preconizează arhitecturi (ansambluri de funcțiuni) mai „prietenoase", în sensul că vor fi orientate pentru satisfacerea cerințelor, chiar subiective, ale utilizatorului și nu către adaptarea la limitările structurale ale mașinii.

Abordarea funcțională presupune nu o interfață rigidă, precum cea structurală, ci multiple interfețe dinamic poziționate și definibile. Inginerul care va aborda domeniul electronicii trebuie să posede cunoștințe care să-i permită accesul la mult mai multe paliere conceptuale. Se întrevede posibilitatea, poate chiar necesitatea, ca de la nivelul dispozitivului care comută (tranzistorul *MOS*, să zicem) pînă la limbajele de nivel înalt, să dispunem de ingineri care să poată realiza o abordarea competentă. Între comutatorul electronic elementar și o funcție complexă trecînd prin paliere structurale și de limbaje se poate stabili un continuu nefragmentat de interfețe rigide. Gîndirea sistemică ne va mai sprijini la anumite paliere, dar va fi total insuficientă.

---

* Mihai Drăgănescu : *Arhitectură și structură în sistemele deschise și introdeschise*, ICCI, București, 1978.

Sub presiunea gîndirii sistemice, structurale, electronica a evoluat către punctul în care abordarea sistemică se dovedește limitativă. Cartea lui T. Blackeslee este una din cărțile care marchează granița dintre era tehnologiilor structurale și a celor funcționale.

Nu putem prevedea în această etapă toate consecințele gîndirii funcționale. Este totuși interesant să marcăm, conștienți de simplitatea exemplului, diferența dintre implementările reprezentate în figurile *P*-9 și *P*-12. În primul caz abordarea a fost de tip funcțional în cel de al doilea a fost bazată pe structurarea cu cărămizi elementare (porți și bistabili). De asemenea, configurația din figura *P*-24 reprezintă o soluție *incipient funcțională* prin sistemele sistolice dedicate funcțional. Parcurgem o perioadă de tranziție în care tendințele funcționale se conturează, la început, timid, în configurațiile clasice, dovedindu-și eficiența în soluții strict particulare.

Așa cum la implementările cu porți elementare nu s-a renunțat în epoca microprocesoarelor, tot așa gîndirea funcțională nu anulează abordarea sistemică, ce se instituie ca un palier superior, posibil datorită spectaculoasei evoluții tehnologice.

Gh. M. Ștefan

februarie 1988

M. C. Bodea

GLOSAR

Absoarbe curent (Sink current) — Caracteristica unui dispozitiv de a accepta curent de la o sarcină externă. Un tranzistor saturat absoarbe curent de la sarcină prin colectorul său.

Analog — Atribut ce indică o mărime care variază continuu.

ASCII — vezi USASCII.

Asincron — Nesincronizat cu semnalul de ceas al unui sistem.

Bait — vezi Byte.

Baud — Este unitatea de măsură a vitezei de transmisie serie reprezentînd 1 bit/s.

BCD — Zecimal binar codat. Un cod în care fiecărei cifre zecimale îi corespund patru biţi cu ponderile 8, 4, 2, 1. Evident se reprezintă numai numerele de la 0 la 9.

Benchmark — test standard pentru evaluarea performanţelor unui echipament de calcul.

Binar — În sistemul binar se utilizează două cifre 0 şi 1. Orice număr se reprezintă în acest sistem ca o sumă de puteri ale lui 2, la fel cum în sistemul zecimal se reprezintă ca o sumă de puteri ale lui 10 (vezi fig. G-1).

Biquinar — Un cod în care fiecărei cifre zecimale îi corespund patru biţi cu ponderile 5, 4, 2, 1.

Bistabil — (flip-flop) Circuit digital utilizat pentru stocarea unui bit de informaţie.

Bit — Constituie un acronim pentru binary digit (număr binar). Pentru un bit sînt posibile doar două valori: 0 şi 1. Este cantitatea de informaţie cea mai mică posibil.

Bit Slice — Un circuit integrat folosit pentru realizarea procesoarelor microprogramate conţinînd, în mod uzual, logică şi registre pentru patru biţi. Are intrări şi ieşiri de transport care permit să se interconecteze orice număr de unităţi *bit slice* obţinîndu-se astfel cuvinte cu un număr mare de biţi.

Buffer — (1) Un amplificator utilizat pentru a extinde gama de variaţie a mărimilor de la ieşire (2) Un sistem digital de stocare intermediară utilizat pentru a reţine temporar datele.

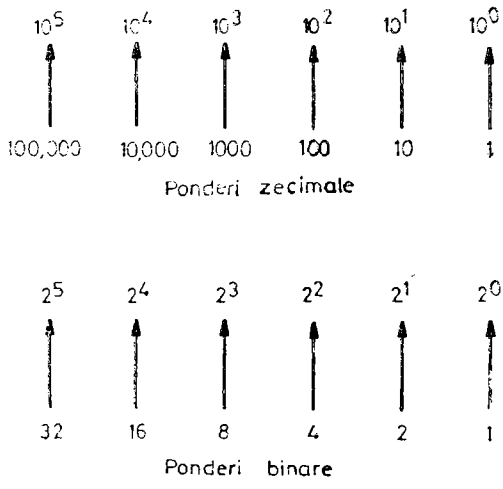


Fig. G-1.

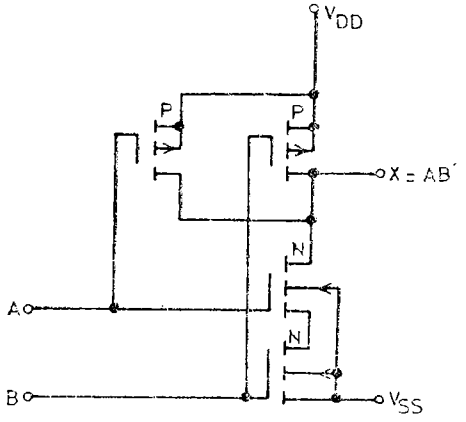


Fig. G-2. Poartă NAND, CMOS (1/4 CD4011).

BUS — Un grup de conexiuni prin care se transferă semnale. Conectarea la BUS a diferitelor dispozitive se face paralel.

Byte — Cod binar de 8 biți

CRT — Tub catodic (cathode ray tube).

Ceas — Un semnal care determină realizarea unei decizii logice, una pentru fiecare tranziție a ceasului.

Cip — Bucata de siliciu, în care se realizează structura circuitului integrat.

CI — Circuit integrat. Un circuit realizat în totalitate într-un cip.

Clear — În mod obișnuit semnifică o aducere la zero (reset) nesincronă cu ceasul.

Clock — vezi Ceas.

CMOS — MOS complementar (Complementary Metal Oxide Semiconductor). Este o familie logică bazată pe utilizarea de tranzistoare MOS cu canal *N* și canal *P*. Se caracterizează printr-o viteză de lucru convenabilă (65 ns maximum), o excelentă rejecție a zgomotului, un consum mic de putere și fan-out foarte mare. Se mai întîlnesc și denumirile COS MOS sau McMOS (vezi fig. G-2).

Cod Gray — Este un cod binar caracterizat prin aceea că între două valori succesive se schimbă numai un bit.

Coduri Hamming — Coduri binare care fac posibilă detectarea și corectarea erorilor.

Comparator diferențial — (Differential comparator) Un circuit care compară două intrări analogice și produce la ieșire 1 sau 0 logic, în funcție de relația mai mare — mai mic existentă între cele două semnale analogice.

CPU — Unitatea centrală (Central processing unit).

Cross compiler — compilator ce generează cod pentru altă mașină.

Declarație (Statement) — Un rînd din programul sursă.

Die — vezi Cip.

Digital — reprezentat prin valori discrete (opusul lui analog). În mod uzual poate semnifica și binar.

DIP — Capsulă cu terminalele așezate pe două rînduri în linie (dual-in-line package). Constituie capsula standard. Capsulele pot fi din plastic sau ceramică cu 14, 16, 24 sau mai multe terminale (vezi fig. G-3).

Dinamic — Memoriile și registrele de deplasare dinamice trebuie reîmprospătate cel puțin la fiecare milisecundă (aproximativ), altfel datele se pierd. ROM-urile dinamice oferă semnal la ieșire numai imediat după semnalul de ceas.

Disassembler — Un program care convertește codul generat de un asamblor în limbajul de asamblare.

DMA — Acces direct la memorie (Direct Memory Acces). Se utilizează pentru a citi sau scrie date direct în memoria calculatorului.

DTL — logică cu diode și tranzistoare. Este prima familie logică de succes. În momentul de față este înlocuită de TTL.

ECL — (Emitter-coupled logic). În prezent este cea mai rapidă familie logică. MECL III are un timp de propagare pe poartă de numai 1 ns. Standardul industrial — seria ECL 10000 — are un timp de propagare de maximum 3 ns. Nivelele logice sînt — 0,9 V și 1,5 V. Sînt necesare rezistoare de terminare chiar și pentru conexiunile cele mai scurte. Se poate realiza OR — virtual prin conectarea în paralel a ieșirilor (vezi fig. G-4).

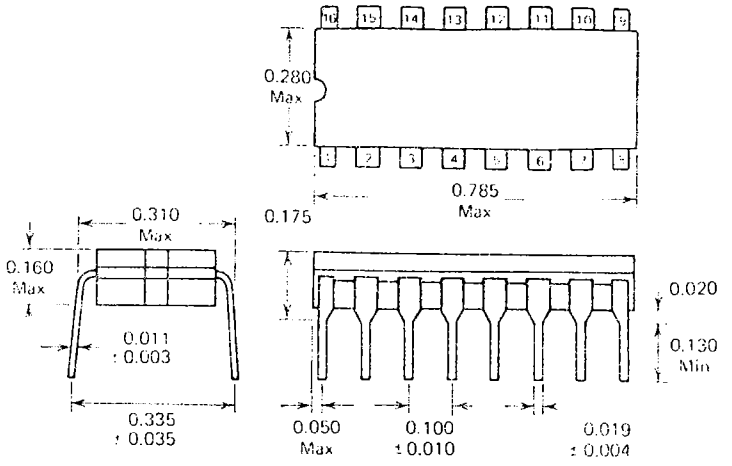


Fig. G-3. Capsulă cu 16 terminale aşezate în linie, pe ambele părţi (Dual in line package — DIP).

EOF — (End of file) — sfîrşit de fişier.

EPROM — Un PROM care poate fi şters şi reutilizat indefinit. În general EPROM-urile sînt şterse prin iluminare în ultraviolet (din acest motiv capsula are o fereastră din cuarţ).

Flip-flop — vezi bistabil.

Fan-out — Reprezintă numărul de ieşiri conectate la o poartă sau numărul maxim care poate fi conectat respectînd valoarea nominală a curentului de ieşire.

FPLA (field programmable logic array). Este un PLA care poate fi programat de utilizator.

Gate — vezi Poartă.

Half Duplex — vezi Semiduplex.

Hamming — vezi Coduri Hamming.

Hexazecimal (Hexadecimal) — Sistem de numărare cu baza 16. Fiecare cifră reprezintă o putere a lui 16 „Cifrele" sînt 0, 1, 2, ..., 8, 9, *A*, *B*, *C*, *D*, *E*, *F*. Fiecărui grup de patru biţi a unui număr binar îi corespunde o „cifră" hexazecimală ; de exemplu 10011111 = 9*F* (vezi tabelul G-1).

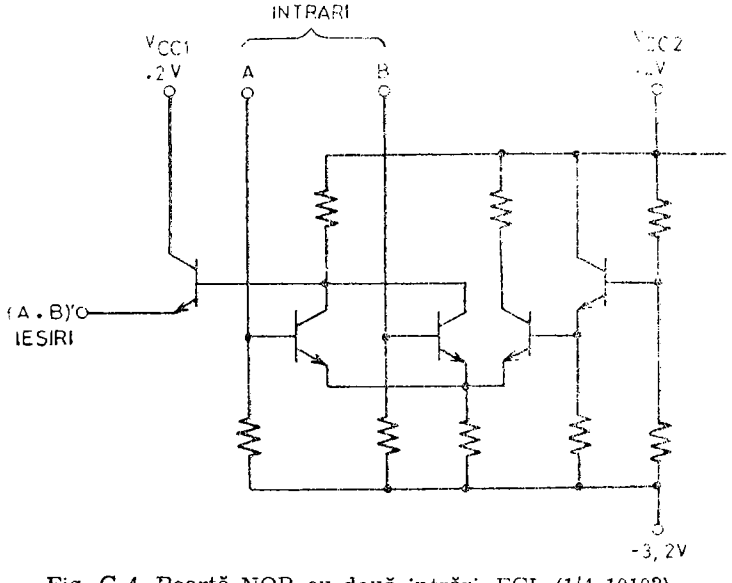


Fig. G-4. Poartă NOR cu două intrări, ECL (1/4 10102).

Tabelul G-1. **Tabel de conversie hexazecimal-zecimal**

→	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
↑																
0X	000	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015
1X	016	017	018	019	020	021	022	023	024	025	026	027	028	029	030	031
2X	032	033	034	035	036	037	038	039	040	041	042	043	014	045	046	047
3X	048	049	050	051	052	053	054	055	056	057	058	059	060	061	062	063
4X	064	065	066	067	068	069	070	071	072	073	074	075	076	077	078	079
5X	080	081	082	083	084	085	086	087	088	089	090	091	092	093	094	095
6X	096	097	098	099	100	101	102	103	104	105	106	107	108	109	110	111
7X	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8X	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9X	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
AX	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
BX	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
CX	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
DX	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
EX	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
FX	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Inversor — un dispozitiv logic care complementează o variabilă logică.

IC (integrated circuit) — vezi CI.

I/O (input-output) — intrare-ieşire. O interfaţă a microprocesorului cu lumea exterioară lui.

Izolare dielectrică (dielectric isolation) — O metodă de izolare a componentelor în CI care oferă o densitate mare de împachetare şi o viteză bună.

LED (light-emitting diode). Diodă semiconductoare care în polarizare directă emite lumină. Valoarea tipică a tensiunii este de peste 1 V iar a curentului de cîţiva mA. Există diode care emit în roşu, galben, verde şi infraroşu.

Logică majoritară (Majority logic) — Este o funcţie logică combinaţională care este „adevărată" atunci cînd mai mult de jumătate din intrări sînt „adevărate".

LSI (Large Scale Integration) — integrare pe scară largă. Un circuit LSI este acel circuit a cărui complexitate este de peste 100 de porţi logice.

Margine de zgomot (Noise margin) — Tensiunea de zgomot necesară ca circuitele logice să funcţioneze incorect. Este dată de diferenţa între nivelul L de la ieşire şi nivelul H de la intrare.

Izolator optic (Optical isolator) — Un dispozitiv semiconductor care constă dintr-o diodă electroluminiscentă şi un fototranzistor. Deoarece cele două dispozitive sînt separate electric între intrare şi ieşire se pot tolera diferenţe de sute de volţi.

Logică cu prag (Threshold Logic) — Logică combinaţională în care ieşirea este „adevărată" dacă un anumit număr de intrări sînt „adevărate".

Mască (Mask) — O placă fotografică din sticlă, a cărei imagine este transferată pe plachetă pentru a defini, în procesul de fabricaţie a CI, de exemplu, ariile pe care urmează să se facă o difuzie.

MOS (Metal-oxide semiconductor) — Circuite care utilizează tranzistoarele MOS. La aceste tranzistoare curentul trece printr-un canal de tip *N* sau *P*, intensitatea sa fiind controlată de cîmpul, electric determinat de tensiunea aplicată pe grilă (poartă).

MODEM (*Mo*dulator/*Dem*odulator) — Se utilizează pentru transmiterea de date pe liniile telefonice.

MSI (Medium-Scale-Integration) — Integrare pe scară medie. Un circuit MSI conţine între 12 şi 100 de porţi (peste 100 de porţi este un circuit LSI).

MTBF (Mean time between failures) — Media timpului de bună funcţionare. Numărul mediu de ore între două defectări aleatoare, după o perioadă iniţială de rodaj.

Multistart (Multilayer) — Plăci cu cablaj imprimat cu mai multe straturi realizate prin lipirea unor plăci dublă faţă ; găurile sînt metalizate.

Nixie, tub (Nixie tube) — Tub de descărcare în gaze utilizat pentru afişare.

NMOS — MOS cu canal *N* (este mai rapid ca PMOS).

NRZ (Nonreturn to Zero) — Reprezentare a unui semnal prin 1 şi 0 fără tranziţiile suplimentare corespunzătoare ceasului.

Numărător zecimal (Decade counter) — numărător care numără modulo 10.

Octal — Sistem de numărare cu baza 8. Ficcare cifră reprezintă o putere a lui 8. Cifrele sînt 0, 1, 2, ..., 7. Fiecărui grup de trei biţi a unui număr binar îi corespunde o cifră în octal ; de exemplu 011111 = 37 (octal).

Octet — vezi Byte

PCM (Pulse code modulation) — Se transmit periodic eşantioane ale semnalului analogic codate binar.

PLA (Programmable logic array) — Un ROM cu decodificatorul parţial realizat.

Placă cu cablaj imprimat (PC board — printed circuit board) — Cablajul se găseşte transpus pe o sită prin intermediul căreia se acoperă cu un lac protector suprafaţa unui placat cu cupru. Zonele neacoperite se corodează, pe placatul de cupru rămînind astfel cablajul dorit. Pentru plăcile dublă-faţă găurile sînt metalizate.

PMOS — MOS cu canal *P* (mai lent ca NMOS).

Poartă (Gate) — un bloc logic digital a cărui ieşiri binare depind de intrările binare după o regulă logică dată.

Port — Un loc prin care pot trece semnale de intrare sau ieşire. Uneori fiecare adresă de intrare/ieşire într-un microcalculator este referită ca „portul numărul...".

Program obiect — Este programul generat de un asamblor sau compilator pornind de la programul sursă.

Program Status Word (PSW) — Un cuvînt care conţine diferitele stări ale maşinii care trebuie salvate atunci cînd se cere o întrerupere. Include validarea întreruperii şi coduri de condiţie.

Proiectare logică (Logic design) — Proiectarea cu circuite digitale.

PROM (Programmable read-only memory) — Memorie de tip „citeşte numai" programabilă. Este un ROM care poate fi programat de utilizator.

Poartă din siliciu (Silicon gate) — MOS la care poarta este realizată din siliciu policristalin şi nu din metal ; circuitele sînt mai dense şi mai rapide.

Program sursă (Source Program) — Un program scris într-un limbaj care trebuie să fie „tradus" de calculator pentru a produce programul obiect.

Proiectare de sistem (System design) — Proiectarea de principiu a unui sistem, lăsînd la o parte detaliile.

Plachetă (Wafer) — Un disc subţire din siliciu în care sînt produse în mod simultan un mare număr de CI. La finele procesului de fabricaţie plachetele sînt separate în cipuri prin tăiere sau zgîriere.

RAM (Random access memory) — memorie cu acces aleator. Este o memorie în care datele pot fi scrise sau citite la orice locaţie indicată de o adresă binară de intrare. Impulsul de scriere face ca datele de intrare să fie scrise la adresa indicată ; astfel conţinutul adresei indicate apare la ieşirea de date.

Registru de deplasare (Shift Register) — Un lanţ de elemente de stocare în care conţinutul elementelor de stocare se deplasează cu o poziţie pentru fiecare tranziţie a ceasului.

ROM (Read-only memory) — Memorie de tip „citeşte numai". Pentru fiecare adresă binară de intrare la ieşire apare o structură fixă de date, definită prin procesul de fabricaţie.

RTL — Familie logică depăşită, bazată pe rezistoare şi tranzistoare.

Schottky TTL — O versiune îmbunătăţită a circuitelor TTL în care se evită saturaţia tranzistoarelor prin conectarea între colector şi bază a unei diode Schottky. Timpul de propagare pentru familia 74S este de maximum 7ns faţă de 15ns pentru TTL. Puterea disipată este însă de aproximativ două ori mai mare ca la TTL. Schottky-TTL de putere mică (Low-power) seria 74LS are timpul de propagare de maximum 28 ns, cerînd numai cu 20% mai multă putere ca TTL obişnuit.

Semiduplex (Half duplex) — Un circuit care este capabil să transmită sau să recepţioneze date, dar nu simultan.

Sense Amplifier — Un amplificator utilizat pentru citirea tensiunii de ieşire mici oferită de memoriile cu miezuri de ferită. Este asemănător unui **Comparator diferenţial** cu o tensiune de ofset (de prag) incorporată.

Sertar (Rack) — O cutie din tablă de oţel utilizată pentru asamblarea diverselor echipamente electronice prevăzută pe fiecare parte cu traverse standard, gata găurite, pentru montarea de panouri, subsertare, etc., toate avînd înălţimea unui multiplu de 1 3/4 ţoli.

Setup time — timpul, înainte de tranziţia ceasului, necesar ca datele de intrare la un bistabil să se stabilizeze pentru o funcţionare corectă.

SOS (Silicon-on-sapphire) — O tehnologie MOS mai rapidă caracterizată prin aceea că siliciul este crescut pe o plachetă de safir numai în zonele în care este necesar. Ca urmare izolarea componentelor se realizează cu aer.

SSI (Small-scale integration) — Integrare pe scară mică. Circuite integrate care conţin mai puţin de 12 porţi logice.

Static — O memorie statică sau un registru de deplasare static nu trebuie permanent acţionate de semnalul de ceas pentru a-şi păstra datele, aşa cum este necesar la cele dinamice.

Subsertar — ansamblu metalic care acceptă mai multe plăci cu cablaj imprimat. Mai multe subsertare se pot monta într-un sertar.

Sincron — sincronizat de un ceas comun sistemului.

Tact — vezi Ceas

Testarea parităţii (Parity check) — La biţii unui cuvînt se adaugă bitul de paritate astfel că numărul de 1 din cuvînt (inclusiv bitul de paritate) este par (paritate pară) sau impar (paritate impară). Paritatea impară are avantajul de a indica eroare pentru un cuvînt compus numai din 0. O eroare dublă nu este detectată.

TDM (Time division multiplex) — multiplexare în timp prin acelaşi canal de transmisie. Se trimit diferite semnale în diferite intervale de timp.

Timp de propagare (Propagation delay) — Este intervalul de timp între variaţia semnalului de la intrare sau a ceasului şi variaţia semnalului de la ieşire.

Timp real (Real time) — Arată că datele sînt prelucrate îndată ce apar, producînd imediat un semnal la ieşire.

Triger Schmitt — Un circuit basculant utilizat pentru a produce o tranziţie netă, plecînd de la un semnal de intrare lent variabil. O reacţie pozitivă modifică valoarea pragului îndată ce a avut loc basculareа. Dacă semnalul scade uşor datorită zgomotului suprapus — sub valoarea iniţială a pragului la ieşire nu se mai întîmplă nimic ; se obţine o imunizare la zgomot.

Tristate — O ieşire logică care poate fi inactivă, la nivel coborît sau ridicat. Acest fapt permite conectarea mai multor ieşiri la un singur bus, deoarece numai una este activă la un moment dat.

Traductor — Un dispozitiv care converteşte energia dintr-o formă în alta.

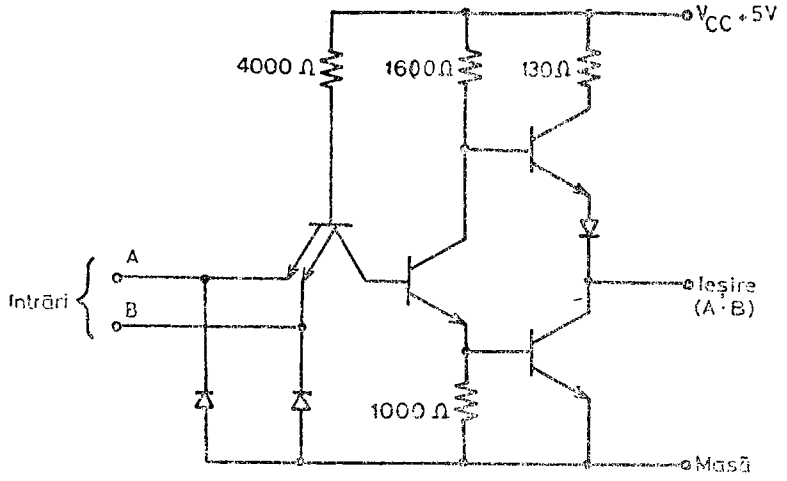


Fig. G-5. Poartă NAND cu două intrări, TTL (1/4) 7 400).

TTL (T^2L) — Tranzistor — tranzistor logic. O familie logică foarte răspîndită care utilizează tranzistorul multiemitor. Porțile au un timp de propagare de maximum 15 ns, se alimentează la +5V și consumă aproximatic 2 mA/poartă. Nivelele logice sînt în mod uzual +0,2 V și +3,4 V iar valoarea nominală a pragului de la intrare este de 1,2 V (vezi fig. G-5).

USASCII (United States of America Standard Code for Information Interchange) — Un cod de 8 biți utilizat pentru comunicații. De obicei este denumit ASCII (vezi Tabelul G-2).

Tabelul G-2. **Codul ASCII**

Biții secvenței transmise 4,3,2,1	Biții secvenței transmise 8[a],7,6,5							
	0000	0001	0010	0100		0101	0110	0111
0000	NUL	DLE	SP	0		P		p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	„	2	B	R	b	r
0011	ETX	DC3		3	C	S	c	s
0100	EOT	DC4	S	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN		6	F	V	f	v
0111	BEL	ETB	,	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	()	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K		k	
1100	FF	FS	,		L		l	ı
1101	CR	GS	—	=	M		m	
1110	SO	RS	.		N		n	
1111	SI	US	/	?	O	—	o	DEL

a Bitul 8 este folosit pentru paritate. Poate fi 1 sau 0 în funcție de tipul de paritate folosit.

Wire wrap — O metodă foarte fiabilă de realizare a conexiunilor fără lipire : sîrma de conexiuni este înfășurată strîns în jurul unui terminal aurit cu secțiune pătrată. Concentrarea efortului la colțurile terminalului produce un mare număr de microsuduri. Mașini automate lucrînd după un program pot interconecta prin wire-wrap panouri mari fără intervenția omului.

INDEX ȘI DICȚIONAR BILINGV