INSTITUTUL POLITEHNIC "TRAIAN VUIA"
TIMISOARA
FACULTATEA DE ELECTROTEHNICA

Ing. Mureşan Ioan

ARHITECTURI MULTIMICROPROCESOR CU APLICATII IN ANALIZA IN TIMP REAL A COMPORTARII SISTEMELOR AUTOMATE

Teză de doctorat

CONDUCATOR ŞTIINŢIFIC

Prof.dr.doc. Edmond Nicolau

1986

P R E F A Ţ A

Lucrarea de faţă se înscrie în direcţiile şi orientările de perspectivă ale producţiei româneşti privind realizarea de sisteme într-o concepţie modulară şi multifuncţională prin dezvoltarea de componente tipizate, îndeosebi electronice şi microelectronice prin utilizarea celor mai perfecţionate tehnologii. Ea prezintă o continuare a activităţii depuse de autor pe parcursul a 10 ani în cadrul Catedrei de Automatică a Facultăţii de Electrotehnică de la Institutul Politehnic "Traian Vuia" Timişoara, în domeniul analizei şi sintezei sistemelor automate asistată de sisteme cu microprocesoare. In cadrul lucrării sînt abordate atît probleme teoretice privind evaluarea performanţelor arhitecturilor multimicroprocesor şi modelarea sistemelor automate cît şi aspecte practice legate de concepţia, proiectarea şi utilizarea acestora.

In perioada elaborării tezei am beneficiat de îndrumarea competentă, atentă şi exigentă a prof.dr.ing. Al. Rogojan, a prof.dr.ing. N.Budişan, a prof.dr.ing. I. Babuţia şi a prof.dr. ing. Pop Vasile. Pentru sfaturile şi observaţiile primite, pentru sprijinul profesional şi moral îmi exprim întreaga stimă şi consideraţie faţă de domniile lor împreună cu cele mai calde şi respectuase mulţumiri.

In perioada finalizării tezei am beneficiat de sprijinul deosebit de competent, de sfaturile şi observaţiile riguroase şi exigente ale conducătorului ştiinţific prof.dr.doc.ing. Edmond Nicolau, căruia pe această cale îi exprim întreaga şi deosebita mea recunoştinţă.

Pentru sprijinul îndelungat şi constant, profesional şi moral, aduc vii mulţumiri soţiei Mureşan Voichiţa şi prietenilor Rebu Nicolae şi Dragomir Toma Leonida.

Mulţumesc în mod deosebit prietenilor Geo Savii, Groza Voicu, Lungu Onuţ, Marchiş Doru, Croţu Vladimir pentru discuţiile purtate, precum şi pentru sprijinul moral pe care mi l-au acordat în perioada elaborării tezei.

Datoroz de asemenea mulţumiri tuturor colegilor din Catedra de automatică, şefului de catedră conf.dr.ing. C. Strugaru, de al căror sprijin real am beneficiat în realizarea acestei lucrări.

Mulţumesc doamnei Brindescu S. şi doamnei Magheţu L. pentru calitatea redactării acestei lucrări.

Mulţumesc în final, tuturor acelora care fără a putea fi menţionaţi, din lipsa spaţiului, au făcut posibilă apariţia acestei lucrări.

Autorul,

CUPRINS

1. INTRODUCERE

1.1. SISTEMELE MULTIMICROPROCESOR COMPONENTE DE BAZĂ A SISTEMELOR AUTOMATE MODERNE

Importanţa ridicării nivelului tehnic şi calitativ al producţiei prin accelerarea procesului de automatizare, electronizare şi robotizare este subliniată în Raportul Comitetului Central cu privire la activitatea Partidului Comunist Român în perioada dintre Congresul al XII-lea şi Congresul al XIII-lea şi activitatea de viitor a partidului în vederea înfăptuirii obiectivelor dezvoltării economico-sociale în cincinalul 1986-199o şi în perspectivă, pînă în anul 2ooo, a României, raport prezentat de tovarăşul Nicolae Ceauşescu: "Se poate afirma că la sfîrşitul secolului, industria românească se va ridica, la un nou nivel calitativ şi tehnic, produsele româneşti situîndu-se la nivelul celor mai bune produse realizate pe plan mondial. Se vor generaliza automatizarea, cibernetizarea şi robotizarea producţiei şi a altor activităţi economico-sociale."

În programul - directivă de cercetare ştiinţifică dezvoltare tehnologică şi de introducere a progresului tehnic în perioada 1986-199o şi direcţiile principale pînă în anul 2ooo a fost prevăzută automatizarea şi extinderea utilizării sistemelor automate. "... Vor fi elaborate noi tipuri de microcalculatoare, echipamente de colectare şi introducere a datelor, programe de bază pentru micro şi mini calculatoare, calculatoare de proces."

In acest context general prezenta lucrare îşi propune să investigheze utilizarea sistemelor multimicroprocesor în domeniul analizei în timp real a comportării sistemelor automate în faza de concepţie, elaborare, testare şi exploatare a acestora.

In lucrare sînt prospectate metode de analiză a performanţelor sistemelor multimicroprocesor şi se propun metode de alegere a arhitecturilor optime utilizabile în analiza în timp real a sistemelor automate. Se propun soluţii în ceea ce priveşte optimizarea acestor arhitecturi prin suportul software. Se analizează cele mai eficiente metode matematice pentru a reflecta componenta dorită a comportării sistemului automat. In final se propun soluţii pentru implementarea pe procesoare cu unităţi microprogramate a proceselor tipice analizei în timp real a comportării sistemelor automate. Performanţele obţinute, înglobînd rezultatele descrise în prezenta lucrare se plasează la nivelul sau peste nivelul performanţelor obţinute, în domeniu, pe plan mondial.

1.2. CONFIGURATII MULTIMICROPROCESOR IN ANALIZA IN TIMP REAL A COMPORTARII SISTEMELOR AUTOMATE

Configuraţiile multimicroprocesor au căpătat în ultima vreme o deosebită importanţă. In domeniul automatizărilor configuraţiile în cazul cărora sarcinile avute în vedere sînt împărţite între mai multe procesoare sînt superioare celor cu un singur calculator central puternic.

Există limite clare la amploarea şi tipul îmbunătăţirilor, în domeniul tehnicii de calcul, posibile cu tehnologia existentă. Ele se concentrează în următoarele domenii :

- Asigurarea de sisteme monochip mai complexe, adăugînd memorie şi dispozitive I/E noi pentru aplicaţii relativ simple.
- Creşterea posibilităţilor µP şi productivităţii sistemului prin asigurarea de cuvinte mai lungi, capacitate de adresare mai mare, seturi de instrucţii extinse şi mai puternice, viteze de operare mai mari cuplate cu un consum de energie mai mic, pentru aplicaţii de complexitate moderată.
- Facilitarea creşterii modulare a performanţelor sistemului prin introducerea unor linii de comandă ce permit implementarea arhitecturilor multimicroprocesor şi prin adăugarea funcţiilor soft de suport pe chip, aducerea interfeţei de bază hardware - software la un nivel cît mai apropiat de utilizator pentru aplicaţii complexe.

Ultimul punct indică o tendinţă clară a industriei producătoare de a se deplasa către SMM (sisteme multimicroprocesor) prin asigurarea suportului hardware şi software care facilitează proiectarea lor.

La nivelul sistemului performanţele pot fi ridicate prin utilizarea conceptului de execuţie concurentă. Ea necesită segmentarea procesului în sarcini şi utilizarea unui executiv de timp real pentru a gestiona, controla şi sincroniza diversele sarcini. rezultatul poate fi ori concurenţă aparentă utilizînd un singur µP în timp partajat, ori concurenţă reală utilizînd SMM.

Prima abordare, utilă numai în sisteme mici şi specializate, caută exploatarea maximă a resurselor cu un singur µP. Deoarece µP sînt fizic limitate ca posibilităţi, metoda este mai adecvată pentru minicalculatoare. A doua metodă tinde spre utilizarea completă a sistemului, nu a µP individuale.

Ideea de a utiliza mai mult de un element de prelucrare pentru a îmbunătăţi performanţele sistemului a precedat apariţia

µP, dar abia acum tehnologia permite utilizarea puterii de calcul într-o largă gamă de aplicaţii în care nu era pînă acum practică din cauza preţului prohibitiv şi dimensiunilor mari.

Avantajele care apar prin utilizarea SMM sînt: flexibilitate sporită; sensibilitate redusă la perturbaţii; timpul de execuţie a unei sarcini (task) este mult mai redus permiţînd chiar lucrul în timp real; fiabilitate sporită (prin redundanţă şi/sau imunitate la defecte); dezvoltarea modulară a sistemelor (spaţială sau funcţională); partajarea resurselor (hardware, programe, date, timp); partajarea funcţională a sarcinilor pe procesoare specializate; un raport cost/performanţe excelent.

Odată cu aceste avantaje, realizarea sistemelor multimicroprocesor ridică o serie de probleme:

- problema separării taskului ce era executat de un singur procesor în subtaskuri ce vor fi prelucrate în paralel de mai multe procesoare (defalcarea sarcinilor şi repartiţia lor pe procesoare);
- determinarea celor mai eficiente structuri de interconectare a procesoarelor;
- proiectarea unor mecanisme cît mai adecvate pentru translatarea adreselor logice în adrese fizice (memory map);
- chiar după defalcarea sarcinilor, interacţiunea între procesoare şi concurenţa pentru utilizarea resurselor continuă să apară (gestionarea resurselor şi formarea şirurilor de aşteptare);
- eliminarea interblocărilor care apar cînd un procesor aşteaptă după o resursă alocată altuia şi viceversa, nici unul din procesoare neputîndu-şi continua execuţia sarcinii până la obţinerea resursei respective;
- proiectarea unor structuri hardware şi soft care să faciliteze imunitatea la defecte a sistemelor multimicroprocesor (SMM);
- integrarea facilităţilor intrare/ieşire (I/E) şi a memoriilor de masă în SMM.

Motivele pentru care SMM nu sînt încă larg răspîndite sînt:

- calculatoarele mono-chip sau pe o singură plăcă cu performanţe suficiente ale UC (unităţii centrale), M (memoriei) şi I/E (dispozitive de intrare/ieşire) au devenit disponibile abia în ultimii ani (după 1980);
- pînă acum existau puţine tehnici verificate pentru comunicare, asigurarea priorităţilor şi a arbitrajului între calculatoare;

- software - ul corespunzător nu este încă pus la punct şi nici nu este disponibil la preţuri rezonabile.

In ceea ce priveşte aplicaţiile de timp real toate realizările de pînă acum au adoptat SMM strîns cuplate partajînd memorii şi/sau magistrale comune. Pînă la începutul anului 1985 cele mai performante magistrale oferite proiectanţilor erau: MULTIBUS-I şi VME-BUS produse de Intel respectiv MOTOROLA. Deşi conţineau funcţii care să faciliteze SMM ele şi-au originile în necesităţile anilor 197o şi erau în fond concepute pentru aplicaţii monoprocesor. Cu toate acestea ele erau în ultima vreme utilizate în special în aplicaţiile multiprocesor.

In primul trimestru al anului 1985 grupurile de lucru P896 şi P1o14 ale IEEE lansează o prognoză a arhitecturilor de magistrale pînă în anii 199o însoţită de două magistrale standard VMEBUS-32 biţi şi MULTIBUS II, a doua fiind anunţată să apară pe piaţă către jumătatea anului 1985. Fiind cea mai performantă şi sintetizînd orientările în domeniu, MULTIBUS II va fi analizată pe scurt.

Multibus II conţine 3 magistrale cu o extensie pentru încă două fig.1.1.1.

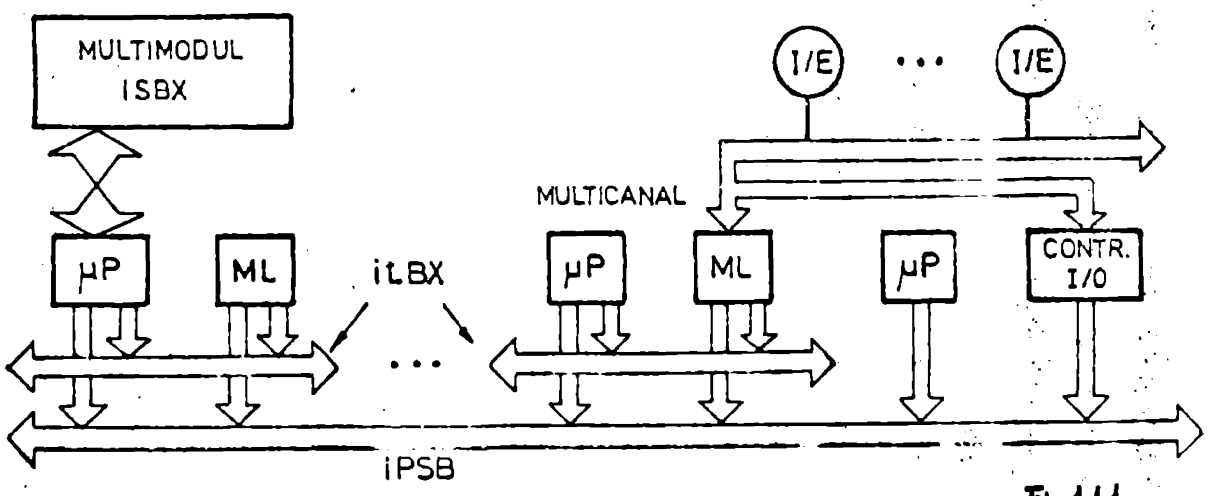


Fig 1.1.1

- iPSB (32 biţi) magistrala paralelă principală a sistemului;
- iLBX magistrala locală de extensie care are rolul de a descărca magistrala principală de tranzacţiile unitate centrală/memorie locală;
- iSSB magistrala serie a sistemului;
- magistrala multicanal cu acces direct la memorie;
- magistrala de extensie I/E pentru magistrala serie.

Evident doar magistrala paralelă este destinată aplicaţiilor de timp real.

Lucrarea de faţă îşi propune să studieze performanţele arhitecturilor SMM bazate pe noile tipuri de magistrale, propune un model arhitectural adaptat la aceste posibilităţi şi arată modul în care modelul arhitectural propus poate fi utilizat în domeniul analizei performanţelor sistemelor automate.

Pe plan naţional s-au făcut eforturi serioase în ceea ce priveşte realizarea unei familii unificate de module hardware, software şi constructive în domeniul arhitecturilor multimicroprocesor, aliniate la soluţiile tipizate pe plan mondial, în special la ICE şi CIIA - IPA.

Sistemul tipizat cunoscut sub denumirea MULTIPROM, finalizat în anul 1985 de un colectiv de cercetători de la CIEA-IPA este destinat în special aplicaţiilor din automatică: echipamente de comandă numerică; automate programabile, calculatoare de proces, echipamente de telemecanică şi transmisie, sisteme de conducere a proceselor industriale, echipamente de testare automată, sisteme de achiziţie a datelor, sisteme de dezvoltare a programelor şi reţele de calculatoare.

Magistrala familiei MULTIPROM constituie baza de dezvoltare a arhitecturilor menţionate şi este derivată din magistrala MULTIBUS I.

Lucrarea de faţă include şi studiul performanţelor arhitecturilor de tip MULTIPROM propunînd de asemenea cîteva modele hardware şi software destinate analizei în timp real a comportării sistemelor automate complexe.

1.3. DESCRIEREA GENERALĂ A LUCRĂRII

In primul capitol se prezintă cadrul general al problematicii abordate în lucrare, menţionîndu-se stadiul actual pe plan naţional şi mondial în domeniul sistemelor multimicroprocesor (SMM) cu prelucrare în timp real.

In capitolul doi sînt analizate critic principalele tipuri de arhitecturi de sisteme multimicroprocesor cu prelucrare în timp real. Se elaborează o ierarhizare a diverselor nivele de interfeţe arhitecturale structurate pe oritontală şi verticală; interfeţele orizontale delimitează nivelurile funcţionale diferite în cadrul structurii de calcul, iar cele verticale delimitează o funcţionalitate de alta în cadrul unui sistem multimicroprocesor cu prelucrare paralelă.

Se analizează apoi comparativ structura logică (relaţiile logice între elementele SMM) structura fizică (metode de transfer a informaţiilor), modul de interacţiune (tare sau slab

cuplate), modul de prelucrare al diverselor tipuri de sisteme multiprocesor. Se adoptă o clasificare a SMM bazată pe cea clasică propusă de Flynn. In final se alege ca fiind cea mai adecvată prelucrării în timp real arhitectura MIMD cu varianta SMD (multiple task, multiple data).

In capitolul trei se trec în revistă metodele cele mai moderne pentru studiul performanţelor sistemelor cu prelucrare paralelă în timp real: metode analitice (lanţurile Markov şi şiruri de aşteptare) cît şi metode de simulare (bazate pe limbajele de evaluare şi programele de simulare GPSS). Se propune modelarea acestor arhitecturi prin lanţuri Markov comasate, domeniu în care se elaborează un criteriu (o condiţie suficientă) pentru aplicarea modelelor aproximative bazate pe acest tip de lanţuri. In acest sens se propune o tehnică de alegere a stării lanţului comasat bazată pe introducerea unei relaţii de β - echivalenţă pe mulţimea S a stărilor lanţului exact. Mulţimea stărilor lanţului comasat rezultă astfel în mod firesc ca fiind mulţimea cît S/β.

In cadrul performanţelor avute în vedere se numără: încărcarea sistemului, metodele de planificare şi sincronizare.

Autorul propune un model arhitectural (MA) care prin particularizare generează diverse arhitecturi compatibile cu magistrala MULTIBUS II. Pentru acest MA se propune şi o metodă originală de sincronizare prin intermediul tehnicilor de rendez-vous, metodă adecvată tipului de aplicaţii legate de analiza în timp real a comportării MA (sistemelor automate).

In capitolul patru se particularizează MA (pxmxb) prevăzut cu p microprocesoare, m memorii comune şi b magistrale în 9 clase cu 8 cazuri distincte şi se elaborează şi studiul pe cazul general. Se propun ipotezele de modelare şi indicii de performanţă pentru toate cele 9 cazuri studiate:

1. (pxlxl) o singură magistrală globală partajată şi o singură memorie comună.
2. (pxpxl) o singură magistrală globală partajată şi p memorii comune.
3. (pxpxl) o variantă îmbunătăţită a cazului 2 în care memoriile comune au două porturi (dual port RAM).
4. (pxpxl) o alternativă a cazului 3 în care se introduc şi o magistrală proprie în afara magistralei locale şi a celei globale.

5. (pxmx2) $m < p$ o generalizare a cazului 1;
6. (pxpx2) o generalizare a cazului 2;
7. (pxpx2) o generalizare a cazului 3;
8. (pxpx2) o generalizare a cazului 4;
9. (pxmxb) p procesoare, m memorii comune și b magistrale globale $p \geq m > b$.

Pentru cele 8 cazuri particulare a MA (pxmb) se evaluează numărul mediu de procesoare active la o încărcare dată și în final se dă o clasificare a acestor arhitecturi pe baza acestui indice de performanță în ipotezele de lucru valabile pentru cerințele impuse de analiza în timp real a comportării SA. Se impun, ca cele mai performante, arhitecturile 3 și 4 respectiv 7 și 8. Se discută rezultatele pe baza soluțiilor analitice și a rezultatelor simulării în GPSS.

In capitolul cinci se definesc obiectivele, metodele și modelele utilizate pentru analiza în timp real a SA. Se generalizează utilizînd limbajul categoriilor modelele SA liniare, cu o singură intrare și o singură ieșire la clasa sistemelor liniare și biliniare multivariabile, lucru care extinde mult clasa SA ce pot fi analizate. După discuția utilității modelelor alese se propune un modul procesor cu unitate microprogramată și un modul soft pentru analiza în timp real a comportării SA, modele compatibile MULTIPROM și a căror performanțe sînt superioare celor prezentate în literatura de specialitate consultată.

Ultimul capitol sintetizează o serie de concluzii referitoare la analiza în timp real a comportării SA utilizînd SA propus și evidențiază totodată contribuțiile originale ale autorului. Sînt prezentate de asemenea o serie de domenii în care pot fi extinse aplicațiile analizei SA precum și unele direcții posibile de cercetare.

2. SISTEME MULTIMICROPROCESOR CU PRELUCRARE IN TIMP REAL

2.1. SISTEME CU MICROPROCESOR IN CONDUCEREA IN TIMP REAL A PROCESELOR INDUSTRIALE

Termenul "prelucrare în timp real" a fost utilizat în ultima vreme pentru a introduce o delimitare clară între activitățile unui calculator conectat într-o configurație de conducere automată a unui proces tehnologic și calculele efectuate de un calculator asupra unui set de date și a căror rezultate nu constituie, cel puțin în mod nemijlocit și imediat, comenzi necesare bunei desfășurări a unui proces din lumea înconjurătoare.

Interconectarea unui calculator într-o configurație de conducere automată a unui proces presupune principial trei operații: achiziția datelor sau evenimentelor din lumea înconjurătoare, prelucrarea propriu-zisă și aplicarea rezultatelor prelucrării sub formă corespunzătoare procesului condus. Termenul "prelucrare în timp real" apare astfel în mod natural datorită faptului că prelucrările sînt declanșate de evenimente externe și rezultatele depind, în afară de datele de intrare de calculele efectuate, și de valoarea variabilei timp atît ca moment, cît și ca durată.

Sistemele cu prelucrare în timp real pot fi clasificate în două mari categorii, sisteme pentru care o prelucrare se consideră corectă dacă timpul ei mediu de execuție, pe o perioadă dată, este mai mic decît o valoare maximă prestabilită; sisteme care sînt utilizate în aplicații la care dacă o prelucrare nu are loc exact în timpul fixat ea duce la perturbarea gravă a evoluției procesului condus. Sistemele din prima categorie poartă denumirea de sisteme în timp real relaxat (soft real time), iar cele din adoua categorie se numesc sisteme declanșate de evenimente externe (hard real time).

Pentru a satisface cerințele de timp real ce se impun în conducerea proceselor se caută soluții atît în ceea ce privește arhitecturile calculatoarelor, cît și în ceea ce privește sistemele de operare corespunzătoare.

Termenul de arhitectură a unui sistem de calcul poate fi utilizat în sens larg, presupunînd modul de organizare a unui calculator. De fapt, în cadrul unui calculator se pot distinge mai multe tipuri de arhitecturi, fiecare fiind definită ca o interfață între două niveluri funcționale diferite. Utilizatorii situați

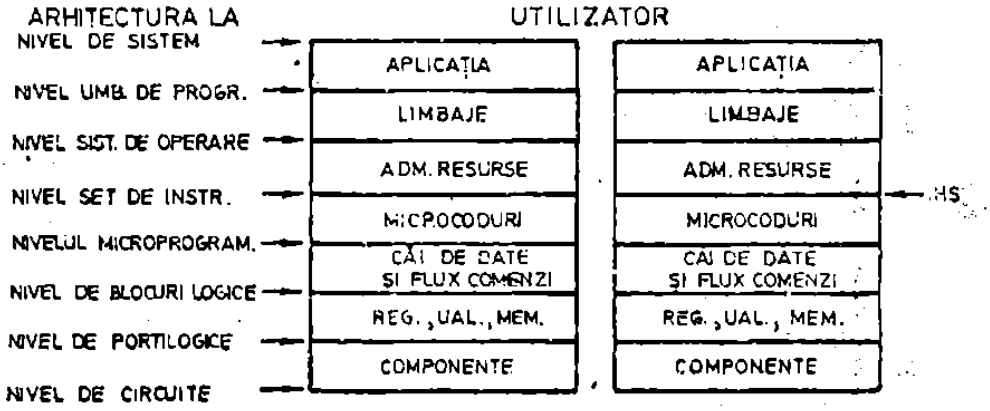


Fig.2.1 -1.

deasupra unei interfețe arhitecturale nu sînt uzual interesați de detaliile privind structura nivelurilor funcționale inferioare. El tratează exclusiv cu proprietățile interfeței lor. În figura 2.1.1-1 se prezintă ierarhia diverselor arhitecturi ce se pot distinge în cadrul unui sistem.

O asemenea structură a nivelurilor arhitecturale structură pe orizontală, poate fi completată prin subdiviziuni verticale, separînd componentele unui sistem multiprocesor (subdiviziuni corespunzînd de exemplu unității centrale, procesoarelor intrare/ieșire, procesoare aritmetice etc.). Un nivel arhitectural într-o asemenea configurație delimitează o funcționalitate sau alta, în cadrul unui sistem multiprocesor cu prelucrare paralelă, distribuită fig.2.1. -1.

Dintre arhitecturile prezentate, cea mai semnificativă este interfața hardware-software (IHS) sau arhitectura propriu-zisă a calculatorului. Sub interfața IHS totul este implementat prin hardware (inclusiv microcodurile), deasupra ei totul este prelucrat la nivel de software. Sarcina cea mai importantă a proiectantului este să decidă care funcții sînt mai bine de implementat deasupra sau sub IHS. Această interfață nu este delimitată în aceeași manieră la toate calculatoarele. Apariția mini și microcalculatoarelor a făcut posibilă introducerea unor schimbări dramatice în plasarea interfeței IHS, ridicînd-o funcțional pînă la nivelul arhitecturii sistemului (iAPX-32 INTEL).

2.2. CLASIFICAREA SISTEMELOR MULTIMICROPROCESOR (SMM)

2.2.1. STRUCTURA LOGICA A SMM

Pentru a asigura o operare armonioasă, relaţiile logice între diversele elemente ale unui SMM trebuie bine definite. In acest context, structura logică se referă la modul în care funcţiile de comandă sînt distribuite între diversele elemente ale sistemului. Cele mai evidente sînt relaţiile de subordonare (verticale) şi de cooperare (orizontale). Intr-un sistem vertical, elementele sînt structurate ierarhic implicînd relaţii sclav-stăpîn; într-un sistem orizontal, elementele sînt echivalente din punct de vedere logic, implicînd relaţii de la egal la egal.

Organizarea verticală. In forma ei cea mai simplă, organizarea verticală posedă un singur stăpîn cu mai mulţi sclavi şi are următoarele caracteristici:

- nu toate elementele sînt echivalente din punct de vedere logic;
- la orice moment dat numai un element poate acţiona ca stăpîn, mai multe elemente însă au posibilitatea de a deveni stăpîn;
- toate comunicaţiile interprocesor se fac prin stăpîn sau sînt iniţiate de el;
- hardware-ul sclavilor poate fi identic, dar ei pot fi specializaţi pe taskuri prin software.

In organizarea verticală, prelucrarea numerică este făcută de procesorul stăpîn (în general cel mai puternic), iar introducerea şi extragerea datelor (I/E) de către procesorii sclavi obţinîndu-se astfel capacităţi de trecere foarte mari. Sistemele de acest tip pot conţine mai multe decît un nivel stăpîn-sclav, formînd astfel o configuraţie tip piramidă.

Organizarea orizontală. Organizarea orizontală a sistemului necesită o coordonare mai sofisticată. Ea are următoarele caracteristici principale:

- toate elementele sînt echivalente din punct de vedere logic;
- coordonarea poate fi făcută cu sau fără ajutorul unui controlor central;
- oricare element poate comunica cu oricare alt element din sistem.

In general, sistemele orizontale sînt mai flexibile decît cele verticale şi sînt capabile de alocare dinamică a resurselor. Totuşi, ele nu sînt eficiente în aplicaţii, avînd multe taskuri foarte deosebite între ele.

Cu excepţia cazului celor mai performante dintre noile µP, sarcina mare de prelucrare necesară pentru coordonarea sistemelor orizontale contrabalansează eficienţa lor în cazul SMM. In acest caz organizarea verticală e mai adecvată.

2.2.2. STRUCTURA FIZICĂ A SMM

Structura fizică a SMM se referă la metoda de transfer a informaţiilor şi este funcţie de aranjamentul comunicaţiilor interprocesor şi de topologia de interconectare.

Comunicaţiile interprocesor. Transferurile de date între procesoare pot fi efectuate fie printr-o memorie comună, fie printr-o structură de magistrală, adesea denumită structură centralizată, respectiv distribuită. In structura cu memorie comună toate transferurile de date se fac prin memorie comună şi elementele nu au acces direct unul la celălalt. In cazul structurii cu magistrală sistem, o legătură logică stabilită pe structura de magistrală creiază o cale de comunicare între elemente; în cazul cel mai general transferurile de date sînt iniţiate şi îndeplinite într-o manieră distribuită.

In sistemele cu transferuri mari de date şi/sau instrucţii tehnicile menţionate nu sînt eficiente datorită conflictelor sporite pentru resursele comune. Problema este agravată în sistemele microprocesor datorită gîtuirii memorie-procesor şi datorită capacităţilor de I/E limitate.

Topologii de interconectare (modul în care sînt conectate elementele). Din punct de vedere fizic există foarte multe moduri de a interconecta N elemente într-un sistem, dar în stabilirea schemei de interconectare apar factori cu fiabilitatea şi dezvoltarea. O schemă fiabilă asigură o cale secundară în cazul în care legătura, calea directă, între două elemente se defectează, o schemă de interconectare facilitează adăugarea de noi elemente fără afectarea structurii existente. Cele patru scheme de interconectare mai importante sînt (fig.2.2.2-1):

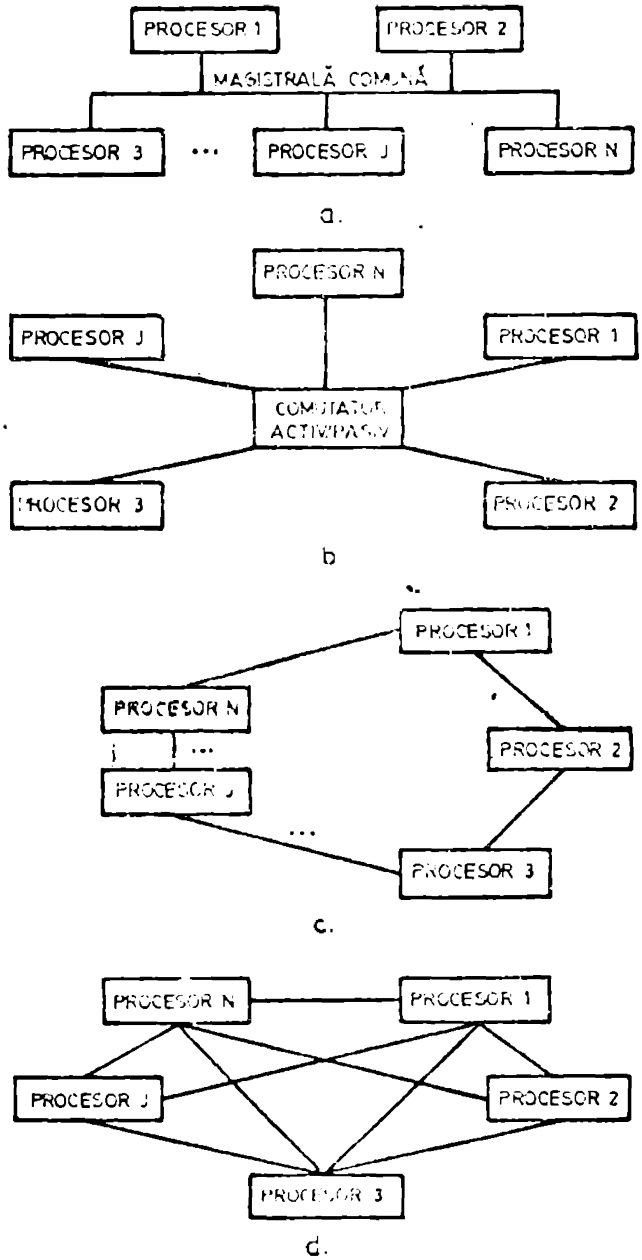


Fig 2.2.2.-1

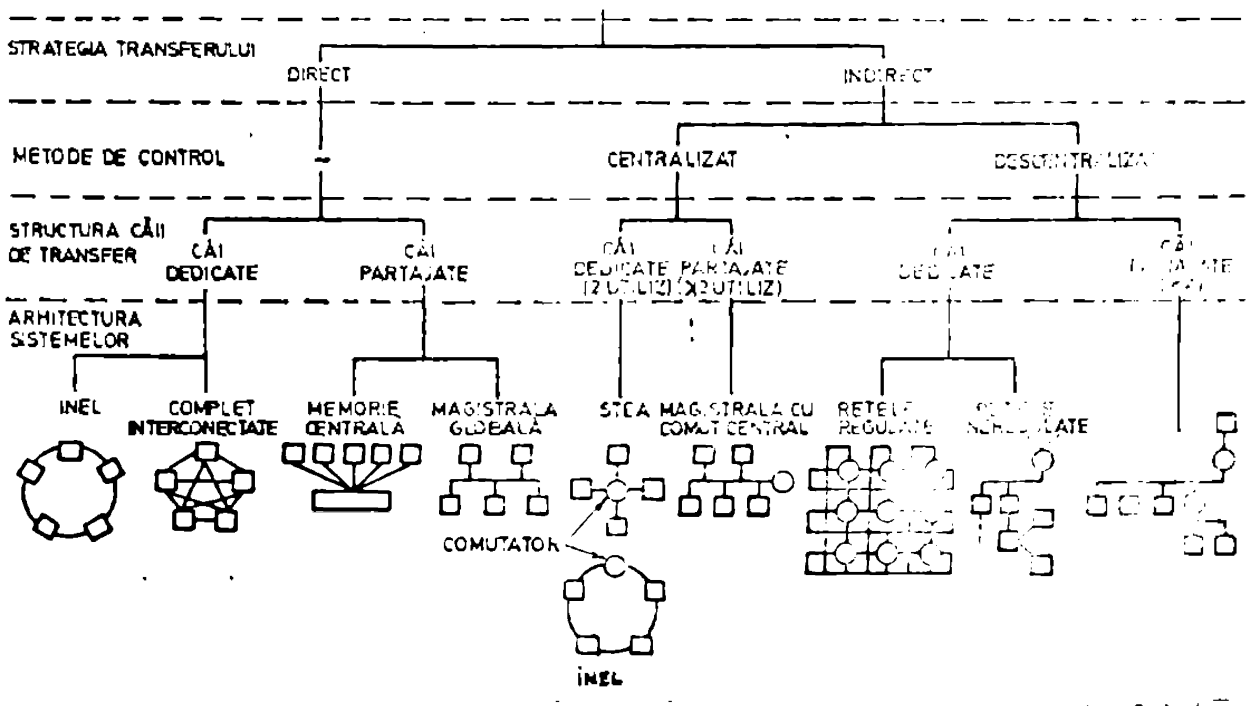


Fig. 2.3.2.1-1.

- cu magistrală comună,
- stea,
- inel,
- complet conectate.

Celelalte topologii au la bază combinaţii sau variaţii ale acestora.

Modul de transfer a datelor şi topologia de interconectare sînt elementele de bază ale sistemului de interconectare. Dar în proiectarea SMM trebuie decis şi asupra transferului direct sau indirect între elemente, controlul centralizat sau descentralizat etc.

Cele mai reprezentative clasificări ale sistemului de calculatoare sînt în funcţie de modul de interacţiune şi de modul de prelucrare.

2.2.3. MODUL DE INTERACŢIUNE

Luînd ca bază modul de interacţiune, sistemele pot fi clasificate după gradul de cuplare şi natura intercomunicaţiilor între procesoare. Cuplarea se referă la abilitatea diverselor elemente de a-şi partaja resursele, la cele două extreme situîndu-se sistemele : slab cuplate şi cele puternic cuplate.

Sisteme slab cuplate. Sistemele slab cuplate, cunoscute şi sub denumirea de reţele de calculatoare, au următoarele caracteristici:

- Reţelele conţin un număr de sisteme de calcul independente, care pot fi dispersate spaţial chiar pe zone geografice întinse.
- Diversele calculatoare din sistem sînt interconectate printr-o interfaţă de comunicare.
- Comunicaţiile intercalculator sînt supuse unui protocol rigid.
- Legăturile de comunicaţie intercalculator sînt [illegible]
- In general, reţeaua e utilizată numai pentru comunicaţii.Prelucrarea propriu-zisă se face de fapt pe site un component al reţelei.

Un utilizator din orice nod al reţelei poate utiliza facilităţile de calcul ale tuturor partenerilor.

In mod obişnuit,reţelele de calculatoare cu cerinţele lor rigide în ceea ce priveşte comunicaţiile interprocesor,nu sînt direct aplicabile în SMM.Totuşi,pe măsura creşterii numărului aplicaţiilor ce necesită calcul distribuit şi cer evoluţia tehnologiei pt,

probabil că versiuni modificate ale reţelelor de calculatoare vor fi realizabile şi cu µP.

Sisteme cuplate puternic. Cunoscute şi sub denumirea de sisteme multiprocesor propriu-zise, au următoarele caracteristici:

- Dispun de o memorie comună. O memorie primară este accesibilă tuturor procesoarelor din sistem. În plus, fiecare procesor poate avea o memorie separată de date.

- Dispun de un sistem de operare comun. Un singur sistem de operare controlează şi coordonează toate interacţiunile dintre procesoare şi procese.

- Dispun de resurse partajate. Facilităţile I/E şi alte resurse ale sistemului sînt în general partajate între procesoare. Totuşi, anumite resurse pot fi destinate unor procesoare specifice.

- Conţin procesoare de uz general simetric configurate şi cu posibilităţi de prelucrare similare.

- Redistribuirea dinamică a sarcinilor unui procesor supraîncărcat permite încărcarea egală a tuturor procesoarelor (alocare dinamică a taskurilor).

- Fiecare dintre procesoarele cooperante poate executa un număr semnificativ de calcule individual (autonomia procesoarelor).

- Sincronizarea între procesoarele ce cooperează este absolut necesară.

Limitarea majoră a sistemelor multiprocesor puternic cuplate constă în posibilitatea apariţiei conflictelor în accesul la memorie. Acest fapt tinde să limiteze superior numărul procesoarelor care pot fi efectiv guvernate de un singur sistem de operare. Cele mai multe configuraţii procesor-memorie tind să reducă cantitatea de conflicte în accesul la memoria principală. Cele mai importante structuri procesor-memorie sînt:

- cu magistrală comună; toate elementele sistemului sînt conectate la o magistrală unică;

- cu comutator; elementele sînt conectate la un modul separat, denumit comutator "crossbar", care poate asigura mai multe conexiuni simultane între perechi de elemente;

- cu memorie multiport, la care fiecare element de memorie are mai mult decît un registru de memorie (port) de acces, şi e conectat la celelalte elemente printr-o magistrală multiplă.

Sisteme distribuite cu μP. Structurile slab şi bine cuplate sînt două exemple de SMM. Alte structuri care combină cele mai bune calităţi ale fiecăreia sînt mai adecvate pentru SMM cu μP. Aceste structuri moderat cuplate sînt cunoscute sub denumirea de sisteme de microcalculatoare cu inteligenţă distribuită (DIMS).

Intr-un sistem distribuit cu microprocesoare, sarcina totală de prelucrare este împărţită în taskuri relativ independente, care pot fi apoi alocate la o varietate mare de elemente. Un asemenea sistem are următoarele caracteristici:

- Fiecare element individual conţine cîte-o UC, memorie locală RAM, ROM, şi eventual poate utiliza sau controla periferice (conţine elemente autonome).

- Ideal, fiecare element este dedicat unui task specific ceea ce determină relativa sa complexitate (procesoare dedicate pe task.

- Structura sistemului nu este în mod necesar simetrică , deoarece procesoarele sale variază în complexitate.

- Fiecare element hard şi soft este creat pentru sarcina specifică ce o îndeplineşte (optimizare software şi hardware).

- Comunicaţia interprocesoare se face mai ales prin date. Totuşi în anumite situaţii datele pot conţine comenzi sau includ răspunsuri la cereri specifice.

- In general fiecare element gestionează atît comanda I/E ,cît şi comunicaţiile sistemului. In cazul unei activităţi de comunicaţii mari, una din aceste funcţii poate fi delegată unui alt procesor. Se utilizează astfel procesoare separate pentru prelucrare şi pentru comunicaţii.

2.2.4. MODUL DE PRELUCRARE

Una din cele mai timpurii clasificări a sistemelor de calculatoare, introduse de M.I. Flynn (1966) luînd în considerare tehnicile de mărirea vitezei, e bazată pe fluxul instrucţiunilor şi datelor şi are următoarea structură:

Sisteme cu prelucrare serie:

- Sisteme cu un singur flux de instrucţii şi un singur flux de date SISD (Single Instruction Stream-Single Data Stream). Un sistem SISD e un sistem clasic (von Neumann) care execută instrucţiile secvenţial, una odată.

Sisteme cu prelucrare paralelă:

- Sisteme cu mai multe fluxuri de instrucţii şi un singur flux de date MISD (Multiple-Instruction Single Data Stream). Au existat controverse în legătură cu tipul sistemelor ce ar trebui incluse în această clasă. Un candidat a fost procesorul pipe-line. Aceste structuri sînt practic inrealizabile.

- Sisteme cu un singur flux de instrucţii şi mai multe fluxuri de date SIMD (Single-Instruction Multiple-Data Stream), care includ sistemele cu prelucrare paralelă: procesoarele matriceale şi cele asociative. Ele au în general o singură unitate centrală de control care aduce şi decodifică instrucţiile şi care apoi difuzează controlul elementelor de prelucrare (procesoare).

- Sisteme cu mai multe fluxuri de instrucţii şi mai multe fluxuri de date MIMD (Multiple-Instruction Multiple Data Stream), multiprocesoarele propriu-zise. Clasa MIMD, cea mai generală, conţine sisteme cu mai multe procesoare-fiecare cu propria unitate de comandă.

Deşi clasificarea lui Flynn este acceptată ca cea mai importantă, ea ia în considerare execuţia numai la nivelul instrucţiei şi este mult prea restrictivă. Pentru a putea include arhitecturile mai recente s-au propus în literatura de specialitate o serie de modificări şi extinderi la taxonomia lui Flynn.

În anul 1979 Bruen şi White prezintă o clasificare a lui Flynn, dar diferenţiază categoria MIMD astfel: sisteme puternic cuplate, sisteme slab cuplate, sisteme cu votare ("Voting Systems"), sisteme cu prelucrare distribuită. Aceşti autori postulează explicit că MISD este o clasă pur teoretică de calculatoare paralele fără nici o valoare practică.

Clasificările pentru calculatoarele paralele arată că categoriile SIMD şi MIMD introduse de Flynn sînt stabile şi astăzi ca noţiuni de bază pentru cele două clase de calculatoare. Calculatorul pipe-line trebuie definit ca o grupă de sine stătătoare. La categoria MISD s-a renunţat, rezultînd următoarea clasificare de bază a sistemelor paralele: SIMD, pipe-line, MIMD. O reîmpărţire mai detaliată acestor grupe principale este foarte delicată şi disputată în momentul de faţă. Din acest motiv în cele ce urmează se vor descrie clasele enumerate accentuînd mai ales diferenţele dintre ele.

In forma sa cea mai generală un sistem multiprocesor capabil să execute concurent un număr dat de taskuri, fiecare utilizînd seturi diferite de date, poate fi denumit sistem MTMD (multiple-task multiple data).

Cum modul de prelucrare şi de interacţiune sînt strîns cuplate, clasificarea bazată pe acesti factori e dată în fig. 2.2.4-1.

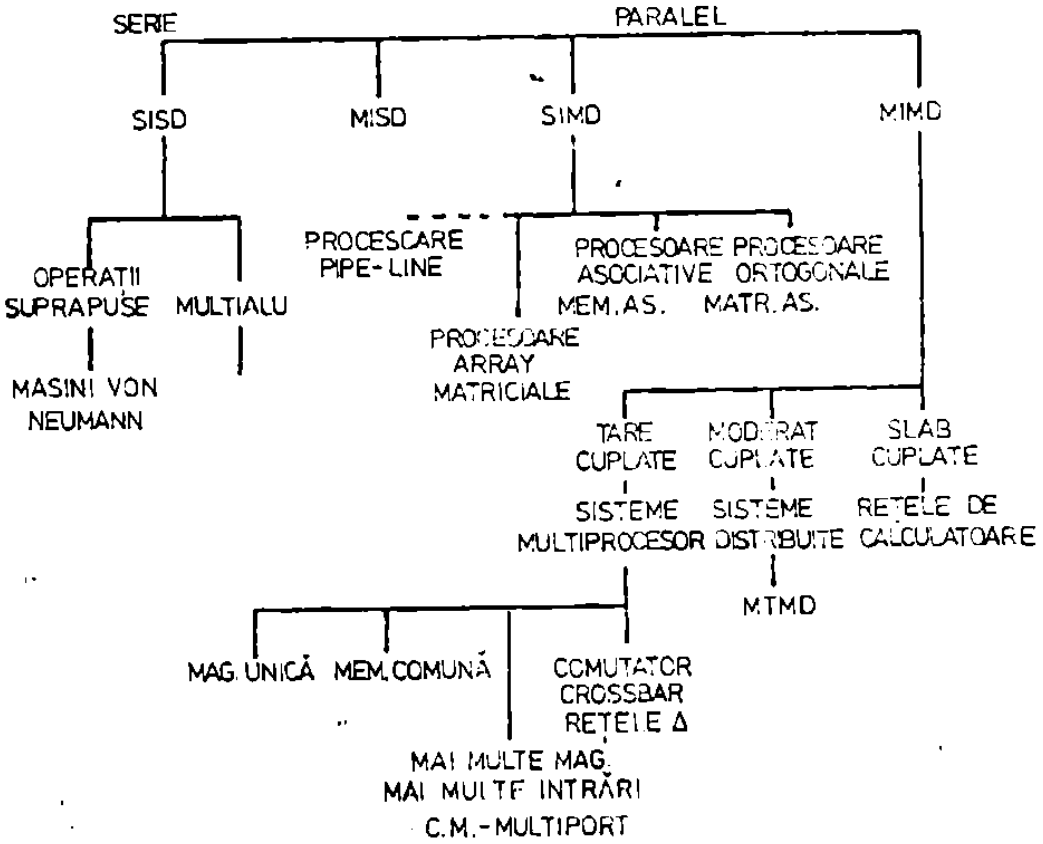


Fig.2.2.4-1.

2.2.5. DESCRIEREA PRINCIPALĂ A DIVERSELOR CLASE DE SISTEME CU PRELUCRARE PARALELĂ

O clasificare posibilă a diferitelor arhitecturi SIMD este:

- **Sisteme cu prelucrare paralelă:**
  - **Sisteme SIMD**
  - **Sisteme pipe-line**
  - **Sisteme MIMD**
- **Sisteme cu prelucrare paralelă mixtă.**

2.2.5.1. ARHITECTURA SIMD

In arhitecturile SIMD (numite şi sisteme cu procesor paralel), o singură unitate de comandă aduce şi decodifică instrucţia. Instrucţia este executată de unitatea de comandă însăşi (de exemplu salt, ramificare condiţionată) sau este difuzată către alte elemente de prelucrare (aceeaşi instrucţie este executată de un vector de procesori asupra unui vector de date), fig. 2.2.5.1-1.

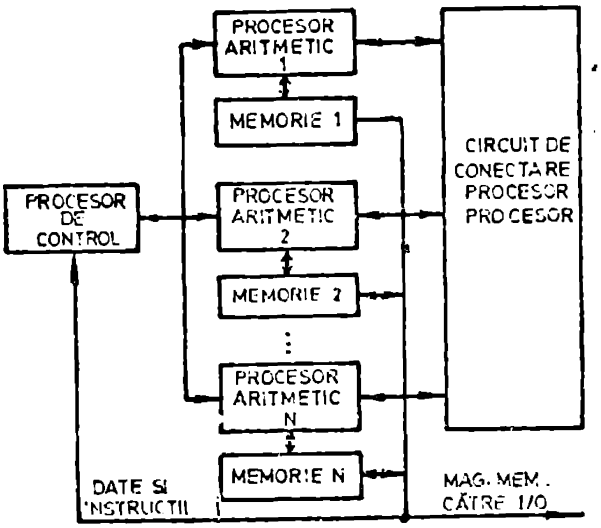


Fig. 2.2.5.1.-1.

Subclasele SIMD sînt:
- procesoare matriciale - unde instrucţiile manipulează simultan vectori de date, iar capacitatea unităţii de comandă este limitată. Sînt cele mai eficiente arhitecturi SIMD din punct de vedere al raportului cost/performanţă. Viteza de execuţie a acestui tip de organizare este foarte mare datorită paralelismului operaţiilor pe diferitele fluxuri de date. Pentru a atinge această eficienţă sînt necesare trei condiţii.

Prima este ca prelucrarea datelor să poată fi descrisă printr-o instrucţie vectorială care execută aceeaşi operaţie asupra unui vector de date. A doua condiţie este viteza foarte mare de circulaţie a datelor între procesoare. A treia condiţie este necesitatea ca datele prelucrate în paralel să fie căutate în paralel. Dacă una din aceste condiţii nu este îndeplinită, calculele vor decurge serie, blocînd anumite procesoare şi anulînd productivitatea prelucrării paralele. In acest tip de sisteme, elementele de prelucrare sînt mutual independente, fiecare dispunînd de memorie şi registre proprii, dar conlucrînd sub controlul unei singure unităţi de comandă, fig.2.2.5.1-1.

Exemple de probleme ce se pot rezolva cu mare eficienţă pe procesoarele matriceale sînt: rezolvarea ecuaţiilor diferenţiale, rezolvarea sistemelor liniare, calcul vectorial cu aplicaţii în teoria sistemelor, controlul traficului aerian, prelucrarea semnalelor radar etc.

- ansamblu de procesoare - la care unitatea de comandă este un calculator central şi elementele de prelucrare comandate, trebuind-u-i memorie proprie şi;

- **procesoare asociative care au acces şi operează asupra datelor prin conţinutul lor nu prin adresă.** Ele constituie un tip de procesoare matriciale în care elementele de prelucrare nu sînt adresate direct. Ele sînt activate când este satisfăcută o anumită relaţie (< , > , =) între conţinutul unui registru încărcat de unitatea de comandă şi datele conţinute în registrele asociative din elementul de prelucrare. Procesoarele astfel selectate de unitatea operantă primesc următoarea instrucţie din program în timp ce celelalte rămîn inactive, fig.2.2.5.1-3.

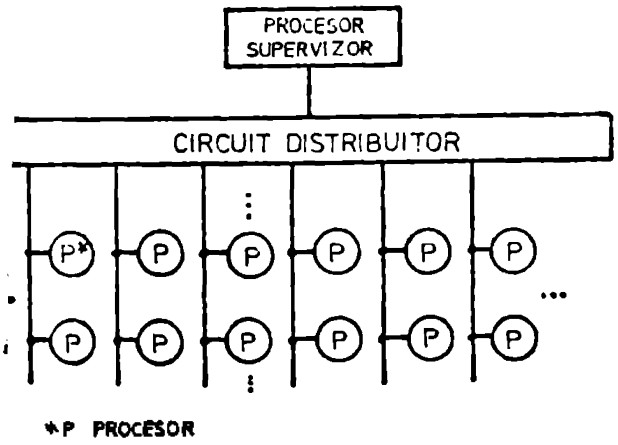


Fig.2.2.5.1-2

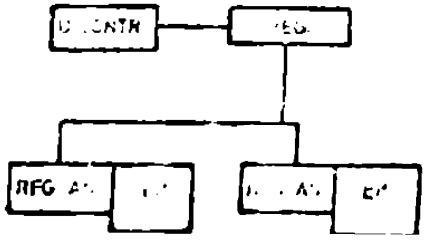
O caracteristică principală a acestor procesoare este tehnica de adresare: datele nu sînt citite de la o adresă a memoriei şi prelucrarea lor se face printr-o comparare (< , > , =) cu o valoare dată iniţial. Această comparare se poate referi la întregul cuvînt sau la părţi ale lui. Conform principiului SIMD, printr-o singură operaţie se realizează operaţii aritmetice sau logice pe un şir întreg de date.

Avantajul faţă de procesoarele matriceale constă în faptul că procedeele de căutare care necesită timpi lungi sînt eliminate, accesul la dat fiind direct.

O subgrupă a procesoarelor asociative o constituie:

- **procesoarele ortogonale.**

Caracteristic pentru acest tip de calculatoare este blocul de memorie, care poate fi adresat atît "orizontal" (cuvînt cu cuvînt pentru un calculator I/E), cît şi "vertical" (pe felii de cîte un octet pentru mai multe elemente de prelucrare). Adresarea verticală se realizează asociativ.



Microprocesoarele bipolare bit-slic sînt elemente fir pentru procesorul matricial (AP). Arhitectura bit-slice aduce un preţ mai scăzut pe funcţie decît circuitele integrate pe scară medie MSI, permit configuraţii dependente de aplicaţie şi toleranţă la defectele parţiale prin alegerea unui număr adecvat de "felii".

2.2.5.2. PROCESOARE PIPE-LINE (TIP LINIE DE ASAMBLARE)

Procesoarele pipe-line pot fi considerate ca o versiune multiplexată în timp a unui procesor matricea . Ele conţin un anumit număr de unităţi funcţionale specializate pe o anumită funcţie. Aceste unităţi sînt aranjate după tipicul unei linii de asamblare. Fiecare proces este subdivizat în sarcini ce vor fi prelucrate simultan de diversele unităţi utilizînd date diferite. Fiecare unitate acceptă noi date la intervale de timp Δt; astfel, cînd există n unităţi, execuţia întregului proces necesită n Δt unităţi de timp. Totuşi, avînd n procesoare operînd simultan, fiecare constituind unul din cele n etape prin care evoluează procesul pentru execuţia sa completă rezultă că pentru a executa k procese sînt necesare n +Δ(k-1) unităţi de timp.

Calculatoarele pipe-line au apărut datorită faptului că încărcarea în timp a componentelor unităţii centrale a calculatoarelor convenţionale este foarte redusă. Deficienţa se poate corecta prin pipe-lining, care se poate realiza la două nivele:

a) La nivel de sistem

Intr-un calculator clasic execuţia unei instrucţiuni se realizează principial prin următoarele etape:

- aducerea instrucţiei, decodificarea instrucţiei, aducerea argumentului (operandului), execuţia instrucţiei.

Prin repartizarea logicii corespunzătoare pe unităţi: unitate de aducere a instrucţiei, unitate de decodificare etc. se pot realiza astfel operaţii legate de patru instrucţii simultan.

b) La nivel de subsistem

In acest caz se introduc unităţi organizate pipe-line pentru diverse operaţii (adunare cu virgulă flotantă, înmulţire cu virgulă flotantă, funcţii logice etc.).

De exemplu, adunarea în virgulă mobilă se poate realiza astfel: etapa 1: modificarea argumentelor astfel încît exponenţii bazelor să fie identici; etapa 2: adunarea mantiselor; etapa3:

normalizarea rezultatului; etapa 4: rotunjirea rezultatului. O astfel de segmentare dă posibilitatea lucrului simultan la patru adunări.

In practică s-a impus mai ales al doilea principiu pipeline. Există cazuri în care organizarea pipe-line este statică şi cazuri în care ea este dinamică.

Viteza mare cu care se prelucrează datele într-un calculator pipe-line impune metode eficiente pentru accesul la instrucţii şi date. Ele se pot realiza pe de o parte prin introducerea instrucţiilor vectorizate (comenzi complete pentru prelucrarea unor vectori de date), pe de altă parte printr-o mărire a dimensiunii transferurilor din memorie. Aceasta din urmă se poate realiza prin memorii cu acces pe mai multe cuvinte sau prin întreţesere ("interleaving"-date care sînt conectate logic, dar sînt depuse în celule independente de memorie).

Calculatoarele pipe-line îşi ating eficienţa maximă dacă:

- apar cît mai puţine salturi în timpul execuţiei instrucţiilor;

- dependenţa între datele de prelucrat să fie cît mai mică sau să nu existe.

Dificultăţile care apar la integrarea calculatorului pipeline în categoria SIMD se înmulţesc dacă se are în vedere că diversele linii de asmblare ale unui calculator pot fi simultan în funcţie. Dacă considerăm că ele lucrează fiecare pe seturi multiple de date, atunci se poate vorbi de un sistem MIMD. Mai evident este că categoria pipe-line a fost tratată ca o categorie separată.

Concluzii privind structurile de tip MIMD. Realizarea practică a structurilor SIMD ridică mai multe probleme. Una dintre acestea este comunicaţia între procesoare, problemă care va fi tratată ulterior.

O altă problemă este cea a lipsei de adaptare între dimensiunea operandului vectorial şi a vectorului de procesor. Dacă primul are dimensiunea m şi există M elemente de prelucrare în procesorul matricial, cea mai eficientă funcţionare se obţine cînd m este un multiplu a lui M. Totuşi, dacă m este considerabil mai mare decît M, scăderea eficienţei este neglijabilă.

In execuţia unui program există şi instrucţii care nu au un caracter vectorial (cele care pregătesc operaţiile vectoriale propriu-zise). Scăderea eficienţei cauzată de ele poate fi redusă prin încercarea de a suprapune în timp (cel puţin parţial) execu-

ţia instrucţiilor secvenţiale cu cea a instrucţiilor vectoriale.

Structurile SIMD facilitează o folosire deosebit de eficientă a resurselor hardware, datorită faptului că la un moment dat toate elementele de prelucrare execută aceeaşi instrucţie. Cînd apar ramificări în program, unitatea de comandă poate urma numai una din ramificaţii, deci comandă numai procesoarele corespunzătoare, celelalte rămîn inactive.

Din cele din urmă sînt de remarcat dificultăţile implicate de proiectarea sistemelor de operare pentru sistemele SIMD.

2.2.5.4. Arhitectura MIMD

Arhitectura MIMD este cea care oferă cele mai mari posibilităţi de folosire în domeniul sistemelor de uz general.

Arhitectura MIMD nu e destinată calculelor iterative vectoriale. Acest mod de organizare (fig.2.2.5.3-1) realizează paralelismul prin lucrul concurent la sarcini (taskuri) diferite asupra unor date diferite şi apoi combinînd rezultatele execuţiei taskurilor independente. Pentru atingerea unui mare eficienţe este necesară o sincronizare riguroasă a proceselor, cît şi o potrivită alocare a taskurilor cu scopul echilibrării încărcării procesoarelor. Aceasta este o mare deosebire faţă de SIMD, unde sincronizarea se face automat la execuţia fiecărei instrucţii şi unde nu apare problema alocării taskurilor, căci toate procesoarele execută aceeaşi operaţie (dar pe date diferite).

Arhitecturile MIMD pot fi clasificate în două clase:

- sisteme distribuite,
- sisteme multiprocesor propriu-zise.

Sisteme distribuite

In sistemele distribuite mai multe procesoare îndeplinesc funcţii dedicate ca parte a unui sistem partajat. Procesoarele pot fi distribuite local (în acelaşi laborator, fabrică, vehicul) sau geografic (sisteme de comunicaţie). Taskurile şi acţiunile lor trebuie să fie complet cunoscute dinainte astfel încît funcţiile sistemului să poată fi divizate între elementele individuale de prelucrare.

Comanda proceselor, controlul [illegible], automatizările discrete sînt exemple posibile de aplicare a arhitecturilor multiprocesor distribuite. Nu sînt necesare trăsături speciale pentru sistemele utilizate, ci doar posibilitatea comunicării între sisteme

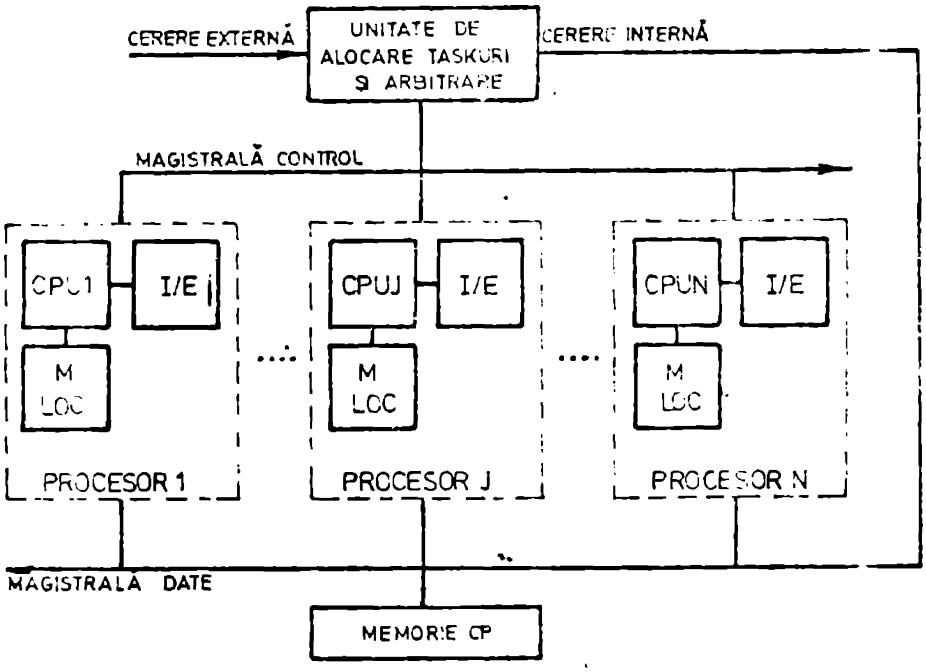


Fig.2.2.5.3-1.

Sisteme multiprocesor propriu-zise

Un sistem din această categorie utilizează un singur sistem de operare pentru a aloca dinamic sarcinile (task) primite. Sistemul de operare poate guverna fiecare EP sau există un procesor de alocare a resurselor (stăpîn) care poate aloca taskuri la EP (sclavi). EP pot fi identice și capabile să execute orice task (simetrice) sau pot fi predestinate pentru a trata funcții speciale (asimetrice).

2.2.5.4. SISTEME CU PRELUCRARE PARALELĂ MASIVĂ

Dezvoltarea rapidă a tehnologiilor VLSI a încurajat realizarea structurilor paralele masive.

Unei structuri de prelucrare paralelă masivă i se atribuie următoarele caracteristici:

- este compusă dintr-un număr foarte mare de elemente de prelucrare: 10^2 - 10^6 (posibil heterogene);

- prețul și viteza de prelucrare a structurii paralele masive depinde, în principiu, liniar de numărul elementelor de prelucrare utilizate;

- structura paralelă masivă este utilizată pentru a rezolva o unică problemă la un moment dat. Spre deosebire de aceasta

rețelele de calculatoare abordează mai multe probleme simultan, schimbînd între ele date sau rezultate parțiale la momente aleatoare de timp.

Limitările calculului paralel masiv nu sînt încă complet elucidate. Nu s-a putut încă stabili care este numărul de procesoare de la care începînd proiectarea, testarea și implementarea devin prea complexe pentru ca sistemul să rămînă eficient și rentabil. De asemenea, nu este încă stabilit din ce fază interacțiunile ce apar în calculul paralel masiv devin prea complexe pentru ca software-ul de control să rămînă în limite rezonabile.

Cercetările actuale în domeniul calculului paralel masiv se axează pe următoarele probleme:

- care sînt arhitecturile optime pentru clasele de probleme specifice și care ar fi beneficiile utilizării unei structuri paralele masive față de utilizarea unei structuri de uz general, în cazul unei probleme specifice; care sînt cele mai eficiente topologii de interconectare; care sînt cele mai eficiente metode de proiectare a algoritmilor de calcul și a programelor.

2.3. STRUCTURA ȘI FUNCȚIILE SISTEMELOR MIMD

Un sistem multiprocesor se deosebește de un set de procesoare, după cum o echipă de muncă se deosebește față de un grup de lucrători individuali. Țelul unei echipe de muncă este realizarea sarcinii globale în timpul cel mai scurt, sarcina unui lucrător este terminarea sarcinii sale individuale în timpul cel mai scurt. Din acest motiv, utilizarea lucrătorilor individuali sau procesoarelor individuale este ineficientă, mai ales dacă volumul lor de muncă este mult diferit sau dacă rezolvarea unei probleme cere rezultatul final sau parțial al altor probleme. In contrast cu aceasta, echipa sau SMM lucrează eficient pentru că membrii lor individuali comunică, cooperează și își împart sarcinile.

SMM pot fi gîndite deci ca o colecție de procesoare care comunică între ele pentru a îndeplini o funcție.

2.3.1. FUNCȚIILE SMM DE TIP MIMD

Proiectarea funcțională a SMM propriu-zise, fig.2.3.1-1, trebuie să țină cont de următoarele cerințe de bază: alocarea taskurilor (statică, dinamică); controlul resurselor sistemului; caracteristicile EP; topologiile de interconectare; interacțiuni între EP, depanare ușoară; efectul asupra performanțelor sistemului, cost, fiabilitate și flexibilitate.

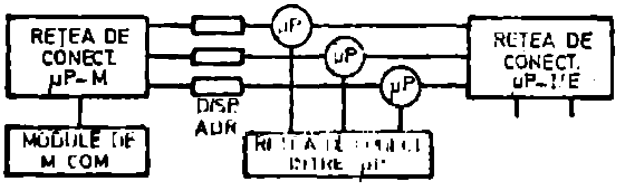


Fig.2.3.1-1.

2.3.1.1. ALOCAREA ȘI SINCRONIZAREA

Alocarea taskurilor și sincronizarea μP sînt cele mai serioase probleme în proiectarea unui SMI. Ele implică identificarea proceselor paralele, partiționarea procesului în subprocese sau taskuri, stabilirea unei scheme de priorități pentru taskuri, alocarea și gestionarea taskurilor între diversele μP, sincronizarea μP astfel ca procesul să decurgă corect, asigurarea unui mijloc de realocare dinamică a taskurilor în cazul defectărilor (degradarea lentă).

Au fost propuse trei abordări posibile pentru obținerea sistemelor de operare concurentă:

- algoritmii proiectați să utilizeze arhitectura paralelă;
- exprimarea prin codificare adecvată a paralelismului potențial;
- detectarea primară a paralelismului pe durata compilării la nivelul instrucțiilor.

2.3.1.2. CONTROLUL RESURSELOR SISTEMULUI

Procesoarele sau taskurile unui SMI își împart o mulțime de resurse în scopul îmbunătățirii performanțelor: resursele hardware (procesoare, memorie, canale, I/E registre, magistrale) și software-ul (programe, alocarea de date, zone tampon, șiruri de așteptare, variabile). Cu cît mai multe resurse sînt comune cu atît este mai mare efortul de control necesar pentru alocarea și rezolvarea conflictelor. O prea mare împărțire a sarcinii totale pe taskuri duce la structuri complexe. Aceasta duce în cele din urmă la micșorarea capacității de trecere sau la blocarea definitiv

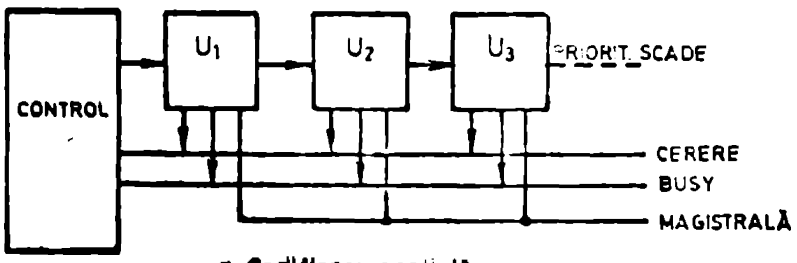
ție

în care două sau mai multe sarcini aşteaptă după resurse care urmează a fi furnizate respectiv de celelalte). Cînd apare o astfel situaţie starea de aşteptare trebuie întreruptă prin control extern şi sarcinile în cauză reiniţializate. Arbitrajul, testarea şi poziţionarea funcţioanelor şi întreruperile sînt cele mai comune metode de control hardware a resurselor.

Arbitrii

Un arbitru este o structură hardware centralizată sau distribuită, care acceptă cereri de la UP-uri (elemente de prelucrare), rezolvă conflictele şi alertează în funcţie de decizie diversele elemente. Un arbitru centralizat constă dintr-o singură unitate hardware. Intel dispune de "controlori de magistrală" ("arbitrii de magistrală"), unele firme propunînd chiar microprocesoare de alocare a resurselor. Un arbitru descentralizat este acela în care controlul logic este distribuit între toate elementele active conectate pentru a partaja o resursă. Controlul distribuit complică în mod necesar fiecare UP, dar îmbunătăţeşte fiabilitatea sistemului în cazul defecţiunilor.

Amîndouă schemele utilizează aceleaşi metode pentru arbitrare: priorităţi spaţiale (daisy chaining), codificarea fixă sau dinamică a priorităţilor, cît şi explorări repetate (polling). Alegerea metodei depinde de : simplitate, necesităţi de deservire a dispozitivului, posibilităţi de dezvoltare, susceptibilităţi de defectare, limitări impuse de liniile de comandă, cabluri de interconectare şi viteza controlorilor. Viteza arbitrului trebuie să fie mare, astfel ca timpul de acces la un dispozitiv să fie doar o fracţiune din timpul de lucru cu respectivul dispozitiv.



a. Codificare spaţială

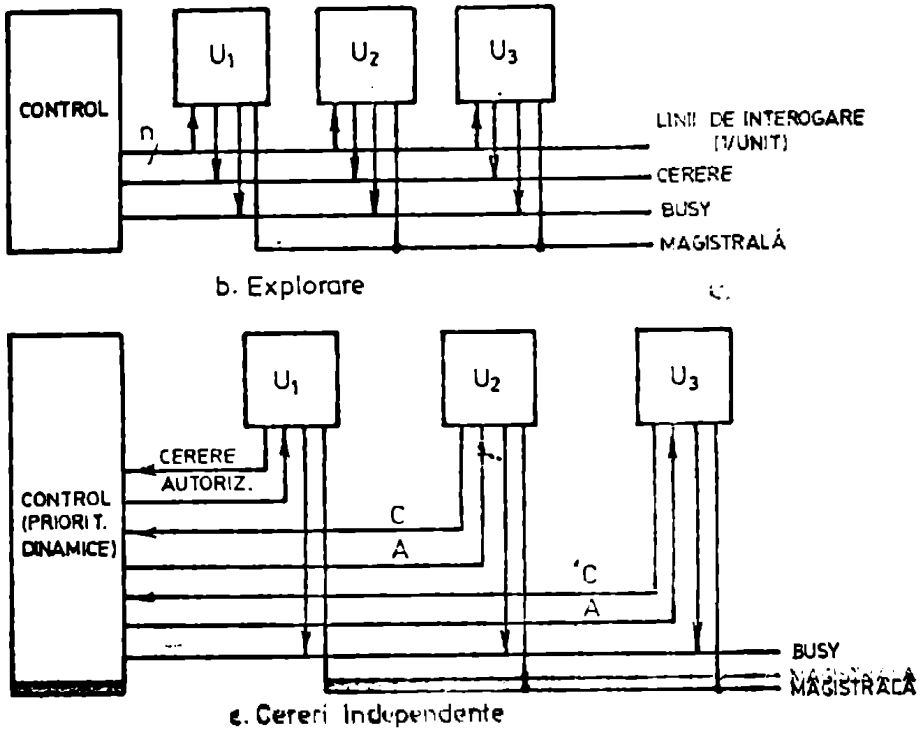


Fig.2.3.1.2-1.

Arbitrii de magistrală pot fi clasificaţi astfel :

- centrali (fig.2.3.1.2-1);

a) daisy chain (codificarea spaţială), caz în care priorităţile alocate sînt fixe, sistemul permite o bună modularitate, defectarea liniei de comandă are dovedeşte catastrofală, timpul de deservire creşte cu numărul unităţilor cuplate.

b) explorări repetate (polling); presupun un hardware complex de control mai multe linii pe magistrala de comandă, dacă controlorul este central atunci priorităţile se pot atribui dinamic, sistemul este modular, imunitatea la defectarea unei unităţi este mai mare, numărul de unităţi [illegible] modificarea modului de interogare, timpii de deservire sînt destul de lungi.

c) cereri independente (întreruperi);este potenţial cea mai rapidă metodă,este cea mai flexibilă în alocarea priorităţilor, este insensibilă la defectele modulelor individuale, necesită

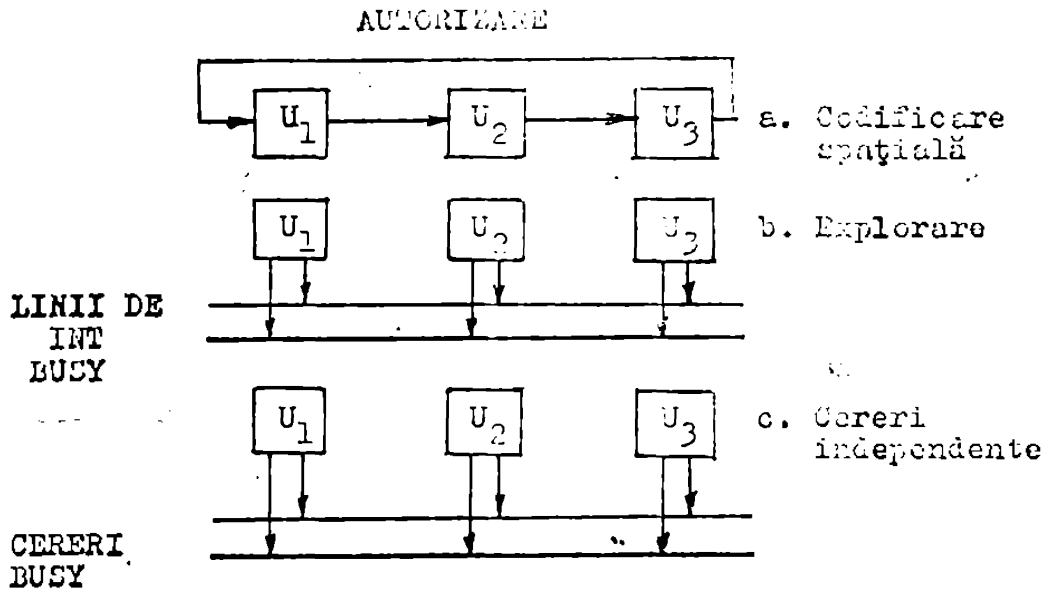


Fig.2.3.1.2-2.

multe linii de comandă şi o logică complexă; este cea mai atractivă în cazul SMM;

- descentralizaţi (Fig.2.3.1.2-2). In acest caz logica de comandă este distribuită pe diversele unităţi componente :

a) daisy chain

b) polling,

c) cereri independente.

Fanioane de stare

Conflictele asupra memoriei comune şi dispozitivelor I/E pot fi rezolvate prin metoda de testare şi poziţionare (TAS = test and Set.) Procesorul solicitant testează starea fanioanelor (care sînt pur şi simplu un indicator al ocupării resurselor). Dacă resursa este ocupată, µP trebuie să aştepte. Dacă resursa este liberă, fanionul se poziţionează pe ocupat pe timpul accesului la resursă şi apoi pe liber după terminarea accesului. Cînd apar cereri simultane pentru aceeaşi resursă trebuie să i se permită unui singur procesor să obţine controlul asupra resursei. Dacă cererile pentru o resursă comună apar asincron, operaţia TAS trebuie să fie indivizibilă (neîntreruptibilă).

Dacă se utilizează locaţii de memorie pentru fanioanele de stare, memoria trebuie să permită un ciclu complet citeşte-modifică-scrie înaintea unui acces ulterior. Procedura necesită o blocare temporară a altui acces la locaţiile memoriei cu rol de fanion. Deşi ar fi mai uşor să se blocheze accesul la un întreg modul de memorie decît la o anumită adresă, în acest caz tot restul modulului este nedisponibil restului procesoarelor. Din acest motiv, biţii de stare sînt uneori implementaţi printr-un grup de registre externe dedicate pe care se execută cicluri citeşte-modifică-scrie indivizibile.

Se presupune că fiecare fanion de stare dintr-un registru extern de stare este adresabil individual; cînd un asemenea fanion este citit de un procesor, fanionul se pune automat pe starea ocupat. Dacă fanionul indică resursă ocupată, procesorul trebuie să aştepte înainte de a primi accesul la resursă şi înscrierea fanionului nu produce nici un necaz; dacă fanionul indică resursă liberă accesul este permis, dar cererile ulterioare sînt negate pînă la ştergerea fanionului. Pentru această implementare este necesar un registru special şi de asemenea o schemă cu priorităţi pentru accesul la biţii registrului. In schimb, nu este necesar un hardware de blocare a accesului, memoria nu este perturbată şi este suficient un ciclu de magistrală pentru a determina accesul la resursă.

Intreruperi

Intreruperile pot fi utilizate pentru a semnala erorile interne ale procesorului (paritate, cod de operaţie invalid, adresă inexistentă, ...), semnale de temporizare (intervale de timp măsurate pe un ceas de timp real), activităţi ale unor dispozitive externe (sfîrşit de caracter, sfîrşit de transmisie pe canal, autorizare de dispozitiv) sau pentru sincronizarea comunicaţiilor interprocesor (prin memorie cutie poştală comună, realocarea taskurilor sau dispozitivelor). Există mai multe metode utilizate pentru a deservi întreruperile în sau.

Deservirea primelor două clase este în mod uzual alocată microprocesorului apelant. Dispozitivele externe pot fi predestinate unor anumite procesoare sau dinamic îndrumate la orice procesor liber. Alocarea în timp real poate fi realizată printr-un arbitru hardware central, un procesor specializat de viteză mare sau chiar de microprocesoarele sistemului. Alocarea se face în funcţie de: capacitatea de deservire, disponibilitate, tipul taskului şi de priorităţile stabilite prin software fiecărui procesor. Comunicaţiile interprocesor pot fi sincronizate prin trecerea unui semnal de întrerupere de la un procesor la altul, astfel încît cererile pentru fiecare pot fi cumulate pe un singur nivel de întrerupere dedicat comunicaţiilor interprocesor. Un vector de priorităţi, corespunzînd procesorului cu cea mai înaltă prioritate care solicită magistrala, poate fi plasat pe magistrala de date cînd întreruperea a fost autorizată.

Tratarea întreruperilor de la dispozitivele externe este probabil cea mai critică decizie în implementarea interfeţei dispozitive I/E - SMM. Alocarea specializată este cea mai simplă, dar limitează flexibilitatea şi reduce imunitatea la defecte. Controlorele

de întreruperi obişnuite asigură: priorităţi fixe, dinamice, regim de explorare şi mascări. Pot fi construite controloare de întreruperi, specifice aplicaţiei, iar în unele cazuri unele µP au un mecanism de întreruperi serie propriu (daisy-chaining).

Dacă se utilizează un comutator central, întreruperile trebuie dirijate la µP alocat. Altfel, deservirea nu va fi făcută la timp, cu atît mai mult cu cît comutatorul este solicitat de mai multe cereri de întrerupere. Alocarea întreruperilor printr-un procesor al sistemului necesită hardware extern pentru memorarea informaţiei despre starea întreruperii. De exemplu, dacă un procesor primeşte o întrerupere de nivel A el va repartiza următoarea întrerupere de acelaşi nivel către un alt procesor disponibil. Metoda este satisfăcătoare dacă întreruperile sînt periodice şi apar cu frecvenţă cunoscută. Nu este necesară o viteză mare a procesoarelor pentru alocarea întreruperii, deoarece gestiunea este distribuită între toate procesoarele. Degradarea parţială este posibilă dacă se realocă tratarea întreruperilor în cazul apariţiei unor defecte.

2.3.1.3. CONFIGURATII DE MEMORIE PENTRU COMUNICATII INTERPROCESOR

Tipul, mărimea şi locul ocupat de memorie au o mare influenţă asupra proprietăţilor SMM. Criteriul de alegere şi amplasare al memoriei are în vedere că mulţimea datelor ce trebuie transportate între procesoare pe magistrala sistem să fie cît mai mică. Cele mai importante decizii privesc:

- utilizarea unei memorii locale, centrale sau ambele;
- cîtă memorie locală, ce tip, cu ce mod de acces;
- modul de organizare pentru o adresare cît mai simplă.

Deciziile trebuie luate ţinînd seama de cele mai recente realizări în hardware şi software. De exemplu, memoria locală nu mai este aşa scumpă şi din ce în ce mai multe microprocesoare conţin cele ieftine au încorporată pe "chip" o memorie.

Memorie comună pentru variabile comune

In sistemele multiprocesor mai vechi mai multe unităţi centrale - UC (µP) îşi împărţeau o memorie în care erau depuse variabile locale şi globale. Memoria devenea astfel punctul critic al întregului sistem. Mai nou se memorează în memorii comune numai variabilele

colective. Memoria comună preia funcţia unei table de afişaj prin care fiecare procesor are acces la informaţia comună.Conflicte la accesul de date sînt excluse, deoarece nu sînt memorate variabile locale.

Memorie locală pentru variabile locale.

Fiecare UC (μP) trebuie să aibă o memorie pentru depunerea unor programe utilizator şi a variabilelor locale,astfel încît să aibă acces la o instrucţiune sau la o variabilă locală independent de magistrala sistem şi memoria comună.Pe lîngă faptul că operaţiile UC sînt mai rapide, memoria locală permite eliberarea magistralei sistem şi memoriei comune pentru comunicaţia între calculatoare. Cantitatea de date care este depusă în memoria comună şi care trebuie transferată pe magistrala sistem este mult redusă.

Cutii poştale (CP) locale pentru comunicare.

O memorie comună poate lua naştere dacă fiecare procesor posedă o "cutie poştală" locală care corespunde unei "table de afişaj" pentru "afişarea" variabilelor comune pentru ceilalţi parteneri. Doar UC locală poate să scrie în cutia poştală locală, celelalte o pot doar citi. Astfel se împiedică ca un procesor funcţionînd greşit să distrugă datele din cutia poştală sau să ocupe magistrala prin scrieri prelungite. Cutia poştală locală are un avantaj.Dacă ea se defectează, doar o parte a sistemului devine neutilizabilă. Dacă se defectează cutia poştală globală,cade întregul sistem.

RAM cu două porturi.

Proprietăţile sistemului se îmbunătăţesc mult dacă cutia poştală (CP) locală este implementată cu RAM cu două porturi. In acest caz UC locală poate să scrie în CP şi în acelaşi timp o altă UC poate să citească din ea. Din acest motiv, cealaltă UC nu trebuie să aştepte accesul la CP.

2.3.1.4. ALOCAREA ADRESELOR LOGICE (MEMORY-MAPPING)

Logica de alocare a memoriilor este utilizată pentru a transforma adresa logică generată de unitatea centrală într-o adresă fizică din memorie.Cînd se lucrează în regim de multiprogramare nu pot fi prevăzute din timp cererile concurente emise de diverse procesoare la o zonă de memorie.Ca atare,este uneori necesar să se mute date şi coduri pentru a face loc pentru altele.Tehnica de alocare a adreselor (memory-mapping) simplifică procedura de alocare dinamică a codurilor în memorie.Ea face posibil să se modifice adresele

logice ale locațiilor de memorie fără deplasarea fizică a informației, folosind [illegible] în mijloc nou pentru partajarea datelor între programe. Adresele logice utilizate în diverse programe pentru a indica anumite date comune sînt translatate în aceleași adrese fizice. Un efect similar poate fi obținut și prin software, utilizînd adresarea indirectă, prin indicatoare implementate ca registre ale unității centrale sau în memorie fie utilizînd adresarea indexată. Aceste facilități sînt implementate pe microprocesoarele moderne pe 16 biți Z8000, I8086, M68000.

2.3.2. CLASIFICAREA SISTEMELOR MIMD

2.3.2.1. TOPOLOGIILE DE INTERCONECTARE

Una din problemele de bază în proiectarea sistemelor multiprocesor este alegerea topologiilor de interconectare a elementelor de prelucrare, cît și alegerea caracteristicilor comunicațiilor. Pe baza diverselor criterii posibile, în literatura de specialitate au fost recomandate diverse clasificări. Un criteriul îl constituie tipul structurilor hardware, care permit transferul mesajelor de la un procesor la altul. Aceste structuri includ căile de comunicații și elementele ce controlează transferul.

Căile de comunicații sînt mijloacele fizice prin care se transferă informația între procesoare. Exemple de căi de comunicații sînt: conductoarele simple, magistralele, legăturile radio și memoriile comune, structuri care nu afectează în nici un fel informația [illegible] prin ele.

Elementele de control, denumite și comutatoare, sînt structuri care acționează asupra mesajului, modificîndu-i adresa de destinație sau calea de comunicație.

Arhitectura unui SMM rezultă în urma unui șir de decizii privind alegerea structurilor mai sus menționate. Prima decizie privește alegerea sau nu a unei structuri cu comutatoare, respectiv alegerea între transmisia directă și cea indirectă. In cazul transmisiei directe, elemente ca: tampoane, memorii etc. nu alterează informația și pot fi considerate ca simple mijloace de comunicație. In cazul transmisiei indirecte, comutatoarele modifică fie ruta, fie adresa mesajului. Rolul comutatorului poate fi preluat chiar de un procesor specializat. Un alt mod de a diferenția între transmisia directă și cea indirectă constă în a determina dacă semnalele de comandă a transmisiei sînt generate de procesoarele emițătoare și receptoare (transmisie directă) sau de structura de comutatoare (transmisie indirectă).

Dacă se decide asupra transmisiei indirecte, atunci trebuie luată o decizie asupra modului de implementare a logicii corespunzătoare: centralizată sau distribuită. In primul caz, o singură unitate de comandă gestionează toate mesajele. In al doilea caz există mai multe asemenea unităţi, fiecare prelucrînd o parte a mesajelor.

O altă decizie priveşte tipul căii de comunicaţie: partajată sau dedicată. Un mijloc de comunicaţie este partajat cînd este utilizat de mai mult de două procesoare. Un mijloc de comunicaţie poate lega două procesoare, uni sau bidirecţional, sau poate lega mai multe procesoare. Numai în cazul căilor unidirecţionale, între două elemente, nu va apare nici un conflict.

Procesoarele unui SMM sînt conectate deci într-o structură topologică prin intermediul:

- căilor (mediul de transmitere a informaţiei: magistrala, memorii);
- comutatoarelor (dispozitive fizice ce stabilesc ruta mesajelor).

Clasificarea topologiilor (fig.2.3.2.1-1, pg.12):

- nivelul 1: conectarea (directă sau indirectă bazată pe strategii de alegere a rutei, implementate prin comutatoare comandate de informaţia de comandă din mesaj);
- nivelul 2: controlul rutelor (control-o rută pentru fiecare mesaj, distribuit - algoritmii de alegere a rutei implementaţi cu comutatoare);
- nivelul 3: structura căilor (dedicate, partajate pe bază de arbitraj);
- nivelul 4: urhitectura (la acest nivel topologia devine identificabilă).

Clarificarea propriu-zisă

Strategia transferului nivel 1	Controlul rutei nivel 2	Structura căii nivel 3	Topologie nivel 4
direct		dedicată	-control distribuit al buclei -interconectare completă
		partajată	-memorie comună -control distribuit al magistralei globale
indirect	central	dedecată	-stea -buclă
		partajată	-magistrală comună controlată central
	distribuit	dedicată	-reţele regulate - reţele neregulate
		partajată	- magistrală comună.

3. ANALIZA PERFORMANȚELOR ARHITECTURILOR SEM CU MAGISTRALE GLOBALE ȘI MEMORII COMUNE MULTIPLE

3.1. MODEL ARHITECTURAL DESTINAT ANALIZEI IN TIMP REAL A PERFORMANȚELOR SISTEMELOR AUTOMATE.

O caracterizare concisă a caracteristicilor sistemelor operînd în timp real este astfel formulată de Brinch Hansen [HAN 1]:

"Aplicațiile în timp real împing tehnologia calculatoarelor și tehnicile de programare la limitele cunoscute (și uneori dincolo de ele). Unul sistem operînd în timp real i se pretinde să monitorizeze, de o manieră continuă și sigură, activități simultane supuse unor cerințe de timp critice. Consecințele unui eșec pot fi deosebit de serioase."

Din punctul de vedere al proiectanților, un sistem operînd în timp real (STR) prezintă următoarele caracteristici:

i) STR interacționează cu mediul înconjurător prin identificarea unui număr mare de evenimente care apar cu o viteză ridicată. Aceasta presupune performanțe deosebite în ceea ce privește:

- Timpul de răspuns, adică timpul scurs între recunoașterea unui eveniment și tratarea lui de către sistem;
- Viteza de circulație a datelor, adică viteza cu care datele sînt transferate înspre, în interiorul și dinspre sistem.

ii) Un STR este destinat să realizeze un răspuns la evenimente externe ce apar aleatoriu. Se poate vorbi în acest sens de o frecvență de răspuns de vîrf - determinată de durata minimă de timp care trebuie să se scurgă între apariția a două evenimente consecutive, astfel încît ele să poată fi considerate distincte. Această frecvență de vîrf trebuie să fie mult mai mare ca frecvența medie de răspuns la evenimentele externe.

iii) STR trebuie să fie determinist, în sensul că se impune să se poată evalua și demonstra corectitudinea rezultatelor obținute atît în stadiul proiectării cît și în stadiul funcționării.

iv) Un STR trebuie să fie modificabil în sensul că trebuie să permită modernizări ulterioare care să nu-i altereze caracteristicile fundamentale.

Arhitecturile propuse în cele ce urmează reprezintă structuri SMM dedicate evaluării performanţelor sistemelor automate.

Numărul de constrîngeri la care este supus un SAA este deosebit de mare. Ele provin din combinarea a două clase distincte: constrîngeri tipice sistemelor operînd în timp real şi constrîngeri tipice sistemelor dedicate evaluării performanţelor sistemelor automate.

Metoda de proiectare aleasă asigură satisfacerea acestor constrîngeri. Ea presupune că proiectarea reprezintă un proces de definire a unor modele urmată de transformări consecutive ale acestora în diferitele faze de proiectare. Cum realitatea nu poate fi niciodată plenar surprinsă, orice model implică în mod necesar ignorarea unor anumite aspecte şi reprezintă o "abstractizare" o "descriere simplificată a sistemului la nivelul dat" (LUC 1).

In acest sens proiectarea unei arhitecturi reprezintă o activitate a cărei obiectiv este trecerea de la o descriere a unui obiect la alta. Descrierea primară reprezintă o specificaţie de proiectare elaborată de către utilizator într-un limbaj specific problemei de rezolvat. Descrierea finală, la un nivel de abstractizare mai scăzut, permite sistemului respectiv utilizînd o anumită tehnologie şi permite construcţia propriu-zisă. Fiecare pas al proiectării presupune astfel definirea unui alt model, corespunzînd unui anumit nivel de abstractizare. Pe durata procesului de proiectare se au în vedere constrîngerile iniţiale care reduc consecutiv opţiunile de avut în vedere, definind astfel tot mai clar arhitectura aleasă.

Termenul de model arhitectural va reprezenta în cele ce urmează o clasă de arhitecturi care au o serie de caracteristici comune dar diferă una de alta printr-o serie de detalii. Un model arhitectural este util dacă se dovedeşte corespunzător de adaptabil pentru o clasă mare de sisteme, dar suficient de restrictiv în sensul de a-şi păstra caracteristicile de bază.

Modelul arhitectural propus trebuie să asigure satisfacerea unei clase destul de [illegible] de probleme ridicate de analiza în timp real a comportării sistemelor automate: achiziţia datelor de la un sistem de eşantionarea şi conversia analog numerică; transpunerea pe calculator a modelelor discretizate ale părţilor componente ale sistemului automat; transpunerea pe calculator a modelului discret al procesului propriu-zis; realizarea funcţiei

de măsurare indirectă (ontinuare stării) cît şi a funcţiei de reglare propriu-zisă însoţită de trimiterea comenzilor către unitatea de conversie numeric-analogică. Aceste funcţii trebuie executate în intervale bine definite de timp pentru a putea răspunde evenimentelor externe asincrone.

In STR realizate cu S.M. evenimentelor ce trebuiesc gestionate le sînt atașate un număr de procese care trebuie activate atunci cînd apar aceste evenimente. Dacă un proces este deja activ atunci este necesar să se poată trece în timp util de la un proces la altul.

Parametrii critici care afectează timpul de răspuns sînt comutarea contextului şi timpul latent. Comutarea contextului implică atît timpul cît şi suma celorlalţi factori implicaţi de tranziţia de la o funcţie la alta; timpul latent este intervalul de timp scurs pînă cînd comutarea devine efectiv posibilă. In cazul sistemelor monoprocesor valorile acestor parametrii sînt inacceptabile. Problema se poate rezolva prin exploatarea naturii independente a funcţiilor de efectuat, SMM permit exploatarea gradului de concurenţă inerent aplicaţiei date. Procesoarele SMM îndeplinesc funcţiile necesare independent una de alta dar sub controlul unui procesor în întregime dedicat activării şi sincronizării acestor funcţii. Cerinţele globale de viteză sînt atinse în acest caz prin execuţia în paralel a unui număr mare de funcţii fiecare procesor lucrînd la viteza maximă permisă de tehnologia sa.

Pe baza analizei şi clasificării sistemelor multimicroprocesor din capitolul 2 se poate concluziona că arhitecturile cu magistrală comună ar putea constitui o soluţie ideală pentru SMM cu prelucrare în timp real. Ele permit transferul direct al informaţiei între procesoare, o modularitate excelentă referitoare la procesoarele adăugate, indiferent de poziţia lor relativă. Imunitatea la defecţiuni este foarte bună în raport cu defectarea procesoarelor.

Utilitatea lor este însă pusă sub semnul întrebării din cauza următoarelor neajunsuri: numărul de conexiuni al magistralei fiind fixat nu permite o dezvoltare ulterioară, necesitînd fie înlocuirea magistralei fie dublarea ei pentru creşterea eficienţei comunicaţiilor.

Se va analiza în continuare influenţa sporirii numărului de magistrale, cît şi adăugarea de module de memorie comună asupra performanţelor SMM.

In concluzie, se propune un model arhitectural (MA) bazat pe utilizarea unui set de magistrale globale şi a unui număr de memorii comune, care pot să varieze de la aplicaţie la aplicaţie în funcţie de necesităţi.

Microprocesoarele (în număr de p) şi modulele de memorie comună (în număr de m) ale MA sînt conectate printr-o mulţime finită de b magistrale globale (care permit conectarea oricărui μP la orice modul de memorie comună). Fiecare μP mai are în folosinţă exclusivă o memorie proprie. MA este deci de tipul pxmxb, fig. 3.1-1.

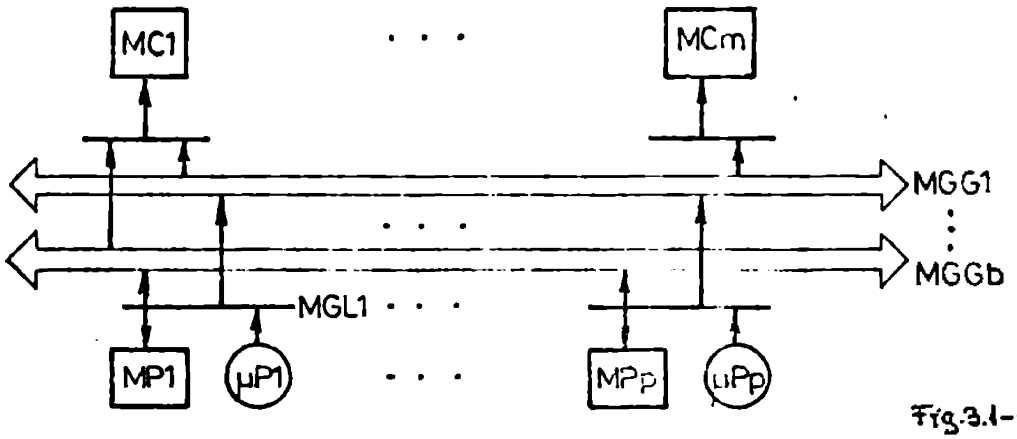


Fig.3.1-1

Schimbul de informaţii între μP se efectuiază prin tehnica de rendez-vous. El nu poate avea loc prin orice memorii ci numai pe itinerarii prestabilite.

Conflictele la resursele comune (memorii şi magistrale) pot provoca apariţia de şiruri (cozi) de aşteptare:

i) dacă b = min (m,p) atunci conflictele se vor datora exclusiv memoriilor partajate (arhitecturile "crossbar"); dacă b > min (m,p) sistemul cîştigă în redundanţă dar nu i se modifică celelalte performanţe.

ii) dacă b < min (m,p) atunci un μP poate fi forţat să aştepte eliberarea unei magistrale pentru a conlucra cu o memorie care este deja liberă.

Se vor studia în continuare cazurile:

$p \geq m > b$

$m \geq p > b$

Modelul arhitectural propus are în vedere exploatarea optimă a facilităţilor oferite de sistemul SMP MULTIPROM proiectat şi realizat la I.P. Bucureşti.

3.2. METODE ŞI MODELE UTILIZATE ÎN EVALUAREA PERFORMANŢELOR ARHITECTURILOR SMP

Studierea performanţelor arhitecturilor propuse, prealabil construirii lor, constituie o componentă foarte importantă legată de activităţile de concepţie, realizare şi întreţinere a sistemelor de calcul [PAU 1]. La nivelul activităţilor de concepţie interesul estimării performanţelor viitoare ale noilor sisteme de calcul tinde să crească în condiţiile sporirii complexităţii sistemelor de realizat şi implicit a riscului de a obţine produse insuficient adaptate cerinţelor şi performanţelor propuse. In mod asemănător alegerea unui SMP adecvat cerinţelor proprii studiului în timp real a performanţelor sistemelor automate presupune investigaţii preliminare privind comportarea sistemelor disponibile în contextul viitor de utilizare.

3.2.1. METODE DE EVALUARE A PERFORMANŢELOR

Principalele metode de evaluare a performanţelor, utilizările în diversele faze ale existenţei unui sistem de calcul sînt:

- analiza matematică pe baza valorilor medii;
- modelele analitice;
- modelele de simulare;
- măsurarea performanţelor;
- experimentarea prototipurilor.

Pentru estimarea performanţelor unui nou sistem de calcul, pentru studierea gradului de adaptabilitate la cerinţele de performanţă şi sfera de utilizare a primele trei metode.

Noţiunea de model [. . .] folosită în toate metodele de evaluare, în cadrul cărora este prezentată printr-un mijloc mai mult sau mai puţin formal sistemul de studiu (incluziv modul lor de funcţionare), este legată în mod inerent de eliminarea unor detalii funcţionale şi structurale. Modul de detaliere al unui model de sistem determină apropierea acestuia de realitate şi influenţează direct proporţional limitele modelului obţinut.

Modelele stohastice de estimare a performanţelor (uzual lanţurile Markov) bazate pe utilizarea valorilor medii ale variabilelor independente, care reprezintă parametrii siste-

mului, sînt relativ uşor de analizat, dar ele presupun un domeniu mic de variaţie a variabilelor. Dacă variaţia parametrilor este mare, atunci aceste modele ignoră situaţiile extreme de funcţionare şi subestimează necesarul de resurse pentru a menţine performanţele sistemului în limite acceptabile în cazul încărcărilor de vîrf. Utilizarea acestor modele permite obţinerea unor rezultate globale de estimare folosite în orientarea alternativelor de concepţie a sistemelor de calcul.

Modelele analitice bazate pe teoria cozilor de aşteptare permit o descriere mai precisă a sistemelor în studiu, structura acestora fiind reprezentată printr-o reţea de cozi de aşteptare. Parametrii modelului pot fi reprezentaţi prin variabile aleatoare descrise prin distribuţii de probabilitate. Creşterea nivelului de detaliu al unui model bazat pe cozi de aşteptare conduce la sporirea dificultăţii analizei matematice din punctul de vedere al reprezentării lui, cît şi al găsirii primei rezolvări problemei matematice atribuite.

Modelele de simulare pot descrie un nivel superior de detaliere, întrucît programele de simulare pot fi construit de aşa manieră încît ele să fie echivalente, din punct de vedere funcţional, cu componentele sistemului real. Deosebirea între model şi sistemul obiectiv rezidă în modul de implementare a componentelor. Parametrii de intrare ai unui model de simulare pot fi daţi sub forma unor distribuţii probabiliste. Cu ajutorul modelelor de simulare se pot obţine nu numai rezultate de ordin cantitativ privind alternativele de arhitecturi, dar şi o primă validare funcţională a algoritmilor utilizaţi.

Cei mai importanţi parametri care intervin în evaluarea performanţelor sînt: încărcarea, arhitectura sistemului şi algoritmii de planificare şi sincronizare procesaţi.

Încărcarea este definită ca un ansamblu de programe, considerate mulţimi parţial ordonate de procese [COF 1] ficcare proces care trebuie să se execute pe sistem cerînd cantităţi specifice de resurse de diferite tipuri.

Caracterizarea încărcării unui sistem poate fi făcută prin: analiza algoritmilor care materializează procesele ; efectuarea de măsurători asupra sistemului; caracterizarea printr-un ansamblu variabil de parametri a unor funcţii de distribuţie probabilistă corespunzătoare. Pentru un sistem nespecializat strict pe un domeniu de aplicaţii, nici o alternativă nu poate oferi modalităţile de reprezentare [illegible] să satisfacă o mare varietate de procese [illegible].

Un sistem de calcul multiprocesor este compus dintr-o serie de resurse pasive şi active. Arhitectura lui poate fi reprezentată, în vederea analizei printr-o metodă de simulare, printr-o reţea de cozi de aşteptare sau printr-o reţea de evaluare.

Algoritmii de planificare a proceselor într-un sistem multimicroprocesor au ca scop optimizarea utilizării resurselor în scopul obţinerii unor performanţe cerute de clasa de aplicaţii. Optimul este relativ la funcţia obiectiv aleasă şi la setul de restricţii luate în consideraţie.

Rezultatele furnizate de o metodă de evaluare a performanţelor trebuie interpretate cu precauţie, în măsura în care definiţia unui model se bazează pe un anumit număr de ipoteze simplificatoare, în funcţie de gradul de detaliere încorporat de modelul respectiv.

3.2.2. MODELAREA CONFLICTELOR CE APAR LA UTILIZAREA MAGISTRALELOR GLOBALE SI A MEMORIILOR COMUNE ÎN MODELUL ARHITECTURAL PROPUS

În literatura de specialitate [BAS], [BHA], [HOO], [SET] [WIL], [AJM 1] au fost propuse o serie de metode şi modele stohastice pentru studiul interfeţelor între procesoare în SMM. Majoritatea acestor referinţe tratează însă studiul performanţelor SMA cu o magistrală globală partajată sau de tip "crossbar".

Scăderea rapidă a costului microprocesoarelor şi a memoriilor, apariţia magistralelor standardizate (MULTIBUS, VEM) au condus către sporirea numărului arhitecturilor orientate pe utilizarea magistralelor globale şi memoriilor comune multiple. [AJM 2], [AJM 3], [GAL], [HOO], [WIL].

Diversele tehnici de modelare propuse, toate bazate pe teoria proceselor stohastice de tip Markov, răspund fiecare unui anume domeniu sau unui anumit mod de analizat performanţele SMM.

3.2.2.1 Lanţuri Markov cu timp discret, omogene.

Def. Fie : S o mulţime finită de stări ale căror elemente sînt numerotate într-un mod bine definit,

p o repartiţie de probabilităţi pe S ale cărei elemente vor fi numite probabilităţi iniţiale $p = (p(i))_{i \in S}$

$P = (p(i,j))_{i,j \in S}$ o matrice stohastică ale cărei elemente vor fi numite pro[illegible]

Prin urmare:

$$p(i)\geq 0,\ i\in S,\ \sum_{i\in S} p(i)=1 \qquad (3.2.2.1-1)$$

$$p(i,j)\geq 0\ \ i,j\in S;\ \sum_{j\in S} p(i,j)=1\ ,\ \ i\in S \qquad (3.2.2.1-2)$$

Un şir de variabile aleatoare $(X(n))_{n\geq 0}$ cu valori în S se numeşte lanţ Markov omogen finit cu spaţiul stărilor S, repartiţia iniţială de probabilitate p şi matricea de trecere P dacă

$$P(X(o)=i)=p(i),\quad i\in S \qquad (3.2.2.1-3)$$

şi

$$P(X(n+1)=i_{n+1}\,|\,X(n)=i_n,\ldots,X(o)=i_o)=P(X(n+1)=i_{n+1}\,|\,X(n)=i_n)=$$
$$=p(i_n,i_{n+1}) \qquad (3.2.2.1-4)$$

pentru $\forall i_o, i_1,\ldots i_{n+1}\in S$ şi $n\geq o$ ori de câte ori membrul întîi e definit.

In cele ce urmează matricea de trecere a unui lanţ Markov va fi păstrată fixă dar repartiţia iniţială va putea varia.

Pentru a putea evidenţia repartiţia iniţială p a lanţului probabilitatea P va fi notată P_p. In particular pentru o repartiţie iniţială concentrată în starea i , $p(j)=\delta(i,j)$, $j\in S$, δ = simbolul lui Kronecker se scrie P_i, caz în care se spune că lanţul Markov porneşte din starea i.

Fie lanţul Markov în care elementele lui S sînt definite astfel:

$$\left(\begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix},\ \ldots,\ \begin{bmatrix} a_p & b_p \\ c_p & d_p \end{bmatrix}\right) \qquad (3.2.2.1-5)$$

în care $b_i\in\{o,1,\ldots q-1\}$ reflectă starea procesorului.

Se defineşte relaţia β pe mulţimea

$A=\{a_1,a_2,\ldots,a_p\}$ ($a_i\in\mathbb{N}$ fiind numerele de ordine a procesoarelor) astfel:

$$a_i\,\beta\,a_j \iff b_i=b_j \qquad (3.2.2.1-6)$$

Pl relaţia β definită pe A x A este o relaţie de echivalenţă

$$\text{i)}\quad a_i\,\beta\,a_i \qquad \forall a_i\in A$$
$$\text{ii)}\quad a_i\,\beta\,a_j \text{ şi } a_j\,\beta\,a_k \Rightarrow a_i\,\beta\,a_k$$
$$\text{iii)}\quad a_i\,\beta\,a_j \text{ şi } a_j\,\beta\,a_i \Rightarrow a_i=a_j \qquad (3.2.2.1-7)$$

Demonstrație :

i) $b_i = b_i \quad \forall i = 1, \ldots p.$

ii) $b_i = b_j$ și $b_j = b_k \Rightarrow b_i = b_k$

iii) $b_i = b_j$ și $b_j = b_i \Rightarrow b_i = b_j$

Dacă β este o relație de echivalență pe A atunci mulțimea reprezintă o partiție a lui S :

$$S = S_1 \cup S_2 \cup \ldots \cup S_q \qquad (3.2.2.1-8)$$

a spațiului stărilor lanțului exact în mulțimi disjuncte două câte două. Pentru simplificare ele se notează

$$\hat{1}, \hat{2}, \ldots \hat{q} \qquad (3.2.2.1-9)$$

Se definește un nou șir de variabile aleatoare $(Y(n))_{n\geq}$ prin relațiile :

$$Y(n) = \hat{k} \Leftrightarrow X(n) \in S_k \qquad (3.2.2.1-10)$$

In continuare se vor examina condițiile în care $(Y(n))_{n\geq 0}$ este un lanț Markov omogen în raport cu toate probabilitățile P_p și ale cărui probabilități de trecere nu depind de p. Dacă acest lucru se întîmplă se spune că lanțul Markov $(X(n))_{n\geq 0}$ este comasabil în raport cu partiția $S = S_1 \cup S_2 \cup \ldots \cup S_q$ iar prin lanț comasat înțelegem lanțul $(Y(n))_{n\geq 0}$

Se notează (3.2.2.1-11)

$p(i,A) = \sum_{j\in A} p(i,j)$ pentru orice $A \subset S$ și $i \in S$.

Definesc mulțimea stărilor lanțului comasat astfel :

$s_c \in S/\beta$, $s_c = (\text{card } \hat{1}, \text{card } \hat{2}, \ldots, \text{card } \hat{q})$ unde card $\hat{i}$ reprezintă numărul de procesoare aflate în starea b_i.

P2. O condiție suficientă ca un lanț Markov să fie comasabil în raport cu partiția

$S = S_1 \cup S_2 \cup \ldots \cup S_q$

este ca partiția respectivă să reprezinte o mulțime cît S/β determinată de o relație de β echivalență pe S

Demonstrație

Avem

$$P_p\,(Y(n+1) = \hat{l} \mid Y(n) = \hat{k},\ Y(n-1) = \hat{i}_{n-1}, \ldots Y(o) = \hat{i}_o) =$$

$$= \frac{\sum_{i\in S_k} P_p(Y(n+1) = \hat{l},\ X(n) = i, Y(n-1) = \hat{i}_{n-1}, \ldots, Y(o) = \hat{i}_o)}{P_p\,(Y(n) = \hat{k}, Y(n-1) = \hat{i}_{n-1}, \ldots, Y(o) = \hat{i}_o)}$$

(probabilitățile $p(i, S_\ell)$, $i \in S_k$

pot fi considerate independente de i, clasa de echivalență a

procesoarelor aflate în starea ei, dată pentru o stare destinaţie cunoscută, în condiţiile în care se presupune că cele p procesoare sînt identice şi se comportă similar în condiţiile arhitecturale şi de încărcare date). In acest caz egalitatea precedentă ia forma:

$$\hat{p}(\hat{k},\hat{l}) = \frac{\sum_{i \in S_{\hat{l}}} P_p(X(n)=i,\ Y(n-1)=\hat{i}_{n-1},\ldots,Y(0)=\hat{i}_0)}{P_p(Y(n)=\hat{k},Y(n-1)=\hat{i}_{n-1},\ldots,Y(0)=\hat{i}_0)}$$

$\hat{p}(\hat{k},\hat{l})$ în virtudea proprietăţii Markov.

$p(k,l)$ este o valoare independentă de starea $i \in S_k$ a lanţului iniţial. In acest caz conform propoziţiei (5.9) [108] $(X_n)_{n\geq 0}$ este comasabil în raport cu partiţia considerată.

Criteriul enunţat afirmă în alte cuvinte că aplicaţia

$$S \xrightarrow{f} S/\beta \text{ , cu}$$

$f(i) = k$ pentru orice $i \in S_k$, $1 \leq k \leq q$

este un nou lanţ Markov ale cărui probabilităţi de trecere nu depind de repartiţia iniţială a primului lanţ.

In condiţiile P2 aplicaţia $f : S \longrightarrow S_c$, $S_c = S/\beta$ reprezintă un omomorfism față de operaţia de tranziţie a stării $s' = \delta(s)$, $f : S \longrightarrow S_c$, $f(\delta(s)) = \delta(f(s))$ pentru $s \in S$.

Funcţia f este un morfism surjectiv (prin definiţie). In aceste condiţii lanţul comasat este un model care nu poate scoate în evidenţă aspecte legate de comportarea individuală a procesoarelor. S-a văzut că o comasare valabilă se poate obţine numai prin impunerea de ipoteze de omogenitate a procesoarelor unei clase şi de influenţe uniforme între clase. Omogenitatea procesoarelor indică că toate componentele unei clase au aceeaşi structură. Uniformitatea se referă la faptul că funcţionarea fiecărui procesor al unei clase influenţează o altă clasă în acelaşi fel, şi fiecare clasă influenţează comportarea componentelor aceleiaşi clase de aceeaşi manieră.

In concluzie, se subliniază că ipotezele de omogenitate şi uniformitate asigură condiţii suficiente pentru comasabilitate cu condiţia existenţei unui omomorfism între clasa stărilor lanţului exact şi al celui comasat.

3.2.2 Procese de asteptare.

Starea sistemului este o descriere a situaţiei actuale care permite să se tragă concluzii (cu caracter probabilist) asupra comportării viitoare a sistemului. Elementele de bază ale unui proces de aşteptare sînt:

- intrarea unităţilor în sistem, aceasta este dată de legea sosirilor ; ... (sosirile)
- serviciile - determinate prin legea serviciului.

Sosirile generează două mărimi aleatoare: numărul de unităţi care intră în sistem într-o perioadă de timp dată;

- intervalul de timp dintre două serviri consecutive.

Cunoaşterea legii de probabilitate a uneia din cele două mărimi determină legea de probabilitate a celeilalte.

Serviciul determină o variabilă aleatoare:

- timpul de servire a unei unităţi.

In practică legile de probabilitate ale acestor variabile aleatoare se determină prin prelucrarea datelor statistice disponibile.

Vom utiliza următoarele notaţii: p - numărul elementelor populaţiei (procesoare) din care pot proveni unităţile în sistemul de aşteptare; n_{21} - numărul de unităţi existente în sistem (în aşteptare sau în curs de servire); n_2 - numărul unităţilor din coadă (nu sînt în curs de servire); n_{21} şi n_2 pot depinde de timp $n_{21} = n_{21}(t)$, $n_2 = n_2(t)$ fiind numărul de unităţi existente în sistem, respectiv în coadă la momentul t; S - numărul staţiilor de servire; σ - numărul staţiilor neocupate la momentul t ($\sigma = \sigma(t)$). (3.2.2.2-1)

Fie $p_k = p_k(t)$ probabilitatea să existe k unităţi în sistem la momentul t. Distribuţia variabilei aleatoare n_{21} este:

$$n_{21} : \begin{pmatrix} 0 & 1 & 2 & \dots & m \\ p_0 & p_1 & p_2 & \dots & p_m \end{pmatrix}; \quad \sum p_k = 1 \qquad (3.2.2.2-2)$$

Valoarea medie a numărului de unităţi din sistem este:

$$\bar{n}_{21} = \sum_{k=0}^{m} k\, p_k \qquad (3.2.2.2-3)$$

Distribuţia variabilei aleatoare n_2 este

$$n_2 : \begin{pmatrix} 1 & 2 & \dots & m-S & 0 \\ p_{S+1} & p_{S+2} & \dots & p_m & \sum_{k=0}^{S} p_k \end{pmatrix} \qquad (3.2.2.2-4)$$

Valoarea medie a numărului de unităţi din coadă este:

$$\bar{n}_2 = \sum_{k=1}^{n-S} k\, p_{S+k} = \sum_{k=S+1}^{m} (k-S)\, p_k \qquad (3.2.2.2-5)$$

In mod analog:

$$\sigma : \begin{pmatrix} 0 & 1 & \dots & S-1 & \sum_{k=S}^{m} \\ p_0 & p_1 & \dots & p_{S-1} & p_k \end{pmatrix}; \quad \bar{\sigma} = \sum_{k=0}^{S} p_k (S-k) \qquad (3.2.2.2-6)$$

Intre $\bar{n}_{21}$, $\bar{n}_2$, $\bar{\sigma}$ şi S avem relaţia:

$$\bar{n}_{21} = \bar{n}_2 - \bar{\sigma} - S. \qquad (3.2.2.2-7)$$

Intr-adevăr:

$$\bar{n}_{21} - \bar{\sigma} = \sum_{k>S} p_k (k-S) - \sum_{0 \le k \le S} p_k (S-k) = \sum_{k \ge 0} p_k (k-S) =$$

$$= \sum_{k \ge 0} k p_k - S \sum_{k \ge 0} p_k = \bar{n} - S.$$

Sosirile sînt caracterizate cel mai adesea de o lege Poisson. Mai precis numărul sosirilor înregistrate într-un interval de timp de lungime t este o variabilă aleatoare cu distribuţie Poisson; dacă unităţile sosesc individual (nu în grupuri) independent una de alta şi pentru intervale foarte mici probabilitatea sosirii unei unităţi este practic proporţională cu lungimea intervalului.

In condiţiile date pentru orice $t \ge 0$:

$$p_k(t) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}; \quad k = 0,1,2,\dots, \lambda > 0; (0! = 1) \qquad (3.2.2.2-8)$$

Distribuţia Poisson a sosirilor nu apare numai în cazul sosirilor în anumite sisteme de aşteptare. In general, se poate vorbi despre un flux de evenimente, care se realizează în timp şi care verifică condiţiile enunţate.

In acest caz avem de a face cu un "flux simplu" sau cu un proces Poisson staţionar. In cazul nostru evenimentele considerate sînt intrările în sistem. Se poate arăta că: dacă sosirile constituie un proces Poisson staţionar de parametru λ, atunci intervalele de timp dintre două intrări consecutive sînt variabile aleatoare independente care urmează legea exponenţială cu parametrul λ şi reciproc, dacă intervalele de timp dintre intrările consecutive sînt independente şi urmează fiecare legea exponenţială cu parametrul λ, atunci fluxul sosirilor este un flux simplu de parametru λ.

Pînă acum am vorbit numai de sosiri, fără a ţine cont şi de servicii (nici de ieşiri din sistem). Ieşirile din sistem nu pot fi studiate independent de sosiri; astfel, nici nu putem aştepta ieşiri atîta timp cît în sistem nu există unităţi (dar putem aştepta sosiri indiferent dacă în sistem există sau nu unităţi).

Modele de aşteptare cu populaţia din care provin unităţile finite.

Dacă un SIM are p procesoare identice şi S magistrale comune sau memorii comune, atunci sîntem în cazul unui model de aşteptare în care unităţile provin dintr-o populaţie cu p elemente şi acest număr nu este în general suficient de mare pentru ca aproximările obţinute luînd $p = \infty$ să fie satisfăcătoare.

Să analizăm pe scurt modelul în care se încadrează exemplul dat. O unitate intrată în sistem este un procesor care solicită o memorie. Dacă în momentul în care apare solicitarea toate memoriile sînt ocupate, procesorul trebuie să aştepte pînă ce una din ele se eliberează.

Memoriile sînt deci în acest caz staţiile de servire. Vom presupune că cererile la memorie apar independent una de alta şi că dacă un procesor se află în starea activă la momentul t, atunci probabilitatea ca el să solicite o memorie (să intre în sistem) între momentul t şi $t + h$ este $\lambda h + \epsilon(h)$.

Să tratăm mai întîi cazul $S = 1$. Dacă în sistem există k $(k < m)$ unităţi la momentul t, atunci au rămas în afara sistemului, $p - k$ şi probabilitatea să nu aibă loc nici o sosire în intervalul $(t, t + h)$ este:

$$(1 - \lambda h - \epsilon(h))^{p-k} = 1 - (p-k)\lambda h - \epsilon'(h), \qquad (3.2.2.2-9)$$

iar probabilitatea să aibă loc cel puţin o sosire în acest interval de timp este

$$\lambda_k h + \epsilon'(h) \qquad (3.2.2.2-10)$$

unde

$$\lambda_k = (p-k)\lambda, \quad 0 \leq k < m.$$

Dacă timpul de lucru cu memoria al unui procesor este exponenţial distribuit cu parametrul μ, atunci:

$$\mu_k = \mu, \quad (1 \leq k < m). \qquad (3.2.2.2-11)$$

Acum putem calcula probabilităţile p_k $(0 \leq k < m)$

$$p_k = \frac{\lambda_0 \cdot \lambda_1 \cdots \lambda_{k-1}}{\mu_1 \mu_2 \cdots \mu_k} p_0 = \frac{n(n-1) \cdots (n-k+1)\lambda^k}{\mu^k} p_0 \qquad (3.2.2.2-12)$$

și p_o se calculează pe baza egalității $\sum_{k=0}^{m} p_k = 1$. (3.2.2.2-13)

Valoarea medie a numărului de unități din sistem este:

$$\bar{n}_{21} = p - \frac{1}{\rho}(1-p_o) \quad ; \rho = \frac{\lambda}{\mu} \quad . \qquad (3.2.2.2-14)$$

Dacă avem S stații de servire ($1 < S < m$) atunci modelul de așteptare apare ca un proces de naștere și moarte cu :

$$\lambda_k = \begin{cases} (p-k)\lambda, & \text{pentru } o \leq k < p, \\ o, & \text{pentru } k > p \end{cases} \qquad (3.2.2.2-15)$$

$$\mu_k = \begin{cases} ku, & \text{pentru } 1 \leq k < S \\ Su, & S \leq k < p \end{cases} \qquad (3.2.2.2-16)$$

Se obține imediat :

$$p_k = \begin{cases} C_p^k \rho^k \cdot p_o, & \text{pentru } 1 \leq k < S, \\ \frac{k!}{S!\, S^{k-S}} C_p^k \rho^k \cdot p_o & \text{pentru } S \leq k \leq p. \end{cases} \qquad (3.2.2.2-17)$$

Notația Kendall pentru specificarea modelelor de așteptare este [JUR 1] :

$$A/B/c/k/m/Z \qquad (3.2.2.2-18)$$

unde A specifică distribuția timpului între sosiri;
B specifică distribuția timpului de servire;
C este numărul de stații de servire;
k este capacitatea sistemului;
m este capacitatea sursei;
Z este disciplina de servire a șirului.

Pentru specificarea lui A și B se utilizează următoarele notații :

GI distribuție generală cu timpii între sosiri considerați variabile aleatoare independente;
G distribuție generală pentru timpul de servire;
E_k distribuția timpilor de servire este de tip Erlang-k;
M distribuția exponențială a intervalelor între sosiri sau a timpilor de servire;
D distribuție deterministă a timpilor între sosiri sau a timpilor de servire.

3.2.2.3 Rețele de evaluare.

Rețelele de reevaluare [JUR 1] , [NICE] aduc o rezolvare a scăderilor rețelelor Petri, Ca și rețelele Petri, rețelele de evaluare (RE) se compun dintr-o mulțime de

locaţii (condiţii), care sînt conectate prin intermediul unor tranziţii.

Tranziţii elementare. Notaţii

O RE este desemnată astfel :

$$E = (L, P, R, A) \qquad (3.2.2.3\text{-}1)$$

unde L: mulţime finită nevidă de locaţii; P: este mulţimea de locaţii periferice terminale P⊆L (locaţii de intrare/ieşire ale RE); R: mulţimea locaţiilor de rezoluţie, R⊆L, fiecare avînd asociată o procedură de rezoluţie (unele locaţii de rezoluţie pot fi locaţii terminale); A este o mulţime finită nevidă de declaraţii de tranziţii, de forma:

$\{a_i\}$, $a_i = (s, t(a_i), q)$ unde s este o schemă de tranziţie, $t(a_i)$ este timpul tranziţiei iar q este o procedură de tranziţie.

Locaţiile sînt ocupate de marcaje simple (denotă doar starea locaţiei) sau de marcaje cu atribute (vectori de atribute dinamic care se modifică pe măsură ce marcajele parcurg reţeaua). Fiecare locaţie are o stare iniţială (liberă sau ocupată), stările iniţiale ale tuturor locaţiilor definind marcajul iniţial M_0 a reţelei.

Au fost definite cinci tranziţii elementare (noduri în reţea în care se decide circulaţia marcajelor prin RE şi unde are loc modificarea atributelor marcajelor). Locaţiile reprezintă condiţii care pot exista pentru o perioadă de timp şi sînt reprezentate grafic prin cerculeţe sau exagoane. O locaţie poate avea cel mult un segment dirijat înspre ea şi un segment spre exterior.

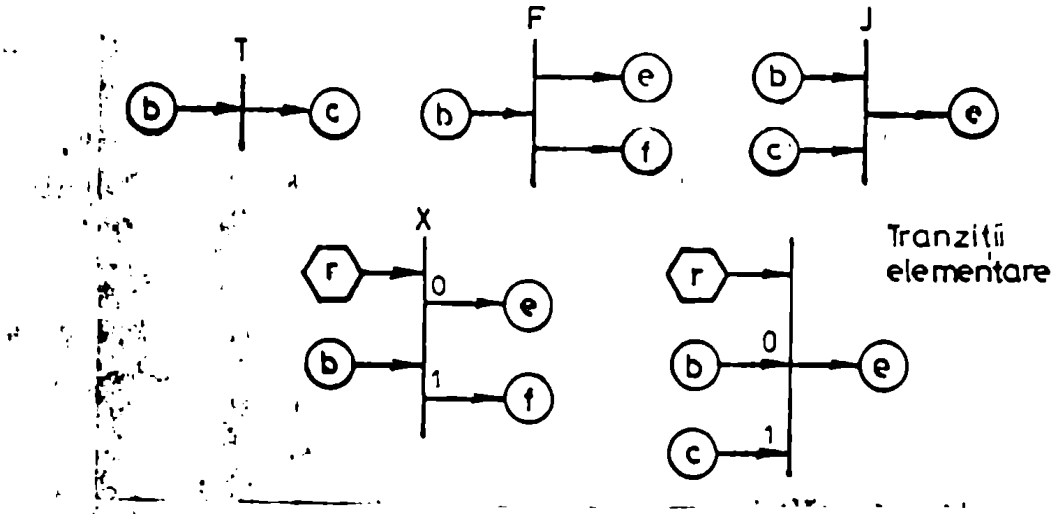


Fig. 3.2.2.3-1.

De remarcat că procedurile tranziţiilor pot referi şi altera valorile atributelor marcajelor care trec prin tranziţia asociată. Ele pot de asemenea referi şi altera variabile globale, la care au acces toate procedurile din reţea. Procedurile de rezoluţie pot referi, dar nu pot altera atribute de marcaje şi variabile globale. Ele pot de asemenea referi marcaje din alte locaţii decît cele asociate cu tranziţia asupra căreia procedura de rezoluţie acţionează. Această ultimă proprietate este dificil de indicat grafic dar uşor de implementat în prelucrarea automată a reţelelor.

Acţiunea unei tranziţii este compusă din cel mult patru faze:

1) faza autorizată: toate locaţiile au starea cerută pentru activarea tranziţiei; din acest moment, operaţia tranziţiei poate începe.

2) faza pseudo-autorizată (se aplică numai la tranziţiile X şi Y): starea locaţiei de rezoluţie este nedefinită. Toate celelalte locaţii au starea cerută pentru aprindere. Începe evaluarea procedurii de rezoluţie.

3) faza activă: acţiunea tranziţiei este în curs de desfăşurare. Starea tuturor locaţiilor asociate rămîne neschimbată.

4) faza terminată : execuţia tranziţiei s-a terminat, iar starea locaţiilor de ieşire şi a locaţiilor de intrare se modifică astfel încît să corespundă condiţiilor create de executarea tranziţiei.

Este important de remarcat că aprinderea unei tranziţii modelează o acţiune anumită a unui sistem şi că deşi faza "autorizat" implică faptul că începe un proces, iar faza "terminat" indică terminarea acelei acţiuni, modelul nu reflectă schimbările care au loc pe parcursul fazei "activ". Metamorfoza cauzată de acţiunea tranziţiei este reflectată numai în faza "terminat". Specificarea timpului petrecut în faza "activ" (modelarea timpului cerut pentru completarea acţiunii) se face prin funcţia $t(a_i)$ din declaraţia tranziţiei în definirea formală a reţelei E. Acest timp de tranziţie dă naştere timpilor de ocupare a locaţiilor de intrare ale unei tranziţii.

Faza pseudoautorizată a unei tranziţii se aplică numai la tranziţiile care au locaţii de rezoluţie terminale. Ca parte din specificarea unei reţele E se prevede o procedură de rezoluţie pentru fiecare rezoluţie posibilă. Procedura de rezoluţie este

o expresie condiţionată care este evaluată ori de câte ori tranziţia asociată este pseudoautorizată. Rezultatul final al producerii de rezoluţie este poziţionarea locaţiei de rezoluţie pe "o" sau "1", permiţînd în felul acesta tranziţiei să devină autorizată. Setul de proceduri de rezoluţie poate fi deci considerat un mecanism pentru modelarea influenţei mediului asupra acţiunii reţelei E.

RE poate fi deci formalizată astfel :(E, Mo,(ξ , Ψ)).

unde : (3.2.2.3-2)

E = (L,P,R,A);

Mo= marcarea iniţială;

ξ = mulţimea variabilelor globale ;

Ψ = mulţimea procedurilor de rezoluţie.

In continuare se specifică: locaţiile L, locaţiile periferice P, locaţiile de rezoluţie R, tranziţiile A, variabilele globale ξ , marcarea iniţială M_0, procedurile de tranziţie şi procedurile de rezoluţie.

Pentru interpretarea notaţiilor formale trebuie reţinute următoarele convenţii:

- în specificarea tranziţiilor, după precizarea tipului tranziţiei, urmează lista locaţiilor aferente, în ordinea: locaţia de rezoluţie (dacă există), locaţii de intrare, locaţii de ieşire;

- în lista de locaţii, dacă o locaţie este destinată să conţină marcaje cu atribute, ea este urmată, între paranteze drepte, de numărul de atribute al marcajului;

- referirea la marcajul dintr-o locaţie oarecare se face în felul următor: K(b), desemnînd prin aceasta marcajul din locaţia b considerat ca o entitate, deci cu toate atributele sale;

- referirea la un atribut al marcajului dintr-o anumită locaţie se face precizînd indicele atributului între paranteze;

- precizarea procedurilor de tranziţie sau de rezoluţie, mai precis a modificărilor în atributele marcajelor şi a modificărilor variabilelor globale, respectiv a poziţionării locaţiei de rezoluţie, se face astfel :

[<expresie booleană>→ <asignare> { ; <asignare>}){;

{<expresie booleană>→ <asignare> { ; asignare }}]

"Execuţia" unei asemenea proceduri constă în evaluarea, de la stînga spre dreapta, a expresiilor booleene şi la efectuarea asignărilor specificate la prima expresie booleană cu valoarea "adevărat". După întîlnirea primei expresii booleene adevărate, explorarea procedurii încetează.

Tranzitiile elementare au capacitatea necesară pentru a descrie orice sistem, indiferent de complexitatea lui, dar pentru sisteme mai complexe rețeaua care rezultă poate să fie atît de extinsă încît să împiedice evidențierea funcțiilor sistemului și deci să anuleze scopul principal al introducerii rețelelor E. Claritatea reprezentării poate fi îmbunătățită prin utilizarea macrorețelelor [JUR 1] adică a unor structuri construite din tranziții elementare, pentru care se introduce un simbol nou, care va fi apoi folosit în elaborarea rețelelor E.

O macrorețea folosită ca tranziție se va numi macrotranziție, iar una folosită ca locație se va numi macrolocație.

Cu ajutorul modelelor arhitecturilor analizate în continuare (modele realizate cu RE) se vor determina:

- timpul mediu de prelucrare realizat ;
- puterea de prelucrare ;
- datele furnizate prin rapoartele standard și speciale ale GPSS.

Marcajele cu atribute utilizate în continuare au următorul conținut:

1. Timpul cît procesorul este inactiv ;
2. Timpul cît procesorul este activ ;
3. Memoria comună ocupată.
4. Timpul cît lucrează cu memoria comună.
5. Suma timpilor cît procesorul este activ.
6. (Suma timpilor cît procesorul este activ)/(timpul total de simulare).

3.2.2.4 Strategii de simulare și modele cu evenimente discrete

Sînt bine cunoscute în teoria sistemelor modelele discrete și procedurile lor de simulare. In cazul evaluării performanțelor unei arhitecturi de calcul aceste modele ar trebui prevăzute cu un mecanism de alegere a evenimentului următor. Acest mecanism restrînge atenția algoritmului de simulare (AS) către acele componente care își pot modifica starea la următorul pas de simulare. Următorul pas logic este eliberarea AS de un regim de pași determinați fix în timp. Se obține astfel o strategie de simulare bazată pe evenimente discrete. In acest mod AS este activat pe baza unei liste de evenimente următoare, care conține timpii următori la care sînt prelucrate componentele, la o modificare internă a stării.

Esența acestui tip de simulare este ipoteza că timpii la care sînt prelucrate evenimentele sînt predictibili ca rezul-

tat al apariţiei altor evenimente. Cînd timpul de prelucrare al unei componente este determinat, componenta este planificată în lista evenimentelor următoare. O a doua ipoteză necesară este aceea că dacă timpul de prelucrare a unei componente nu poate fi determinat dinainte, ea nu-şi va modifica starea decît ca urmare a unei tranziţii a stării unei componente ce fusese deja planificată. In aceşte ipoteze AS avansează ceasul de simulare la cel mai apropiat timp de prelucrare şi efectuiază acţiunile prescrise pentru momentul corespunzător. Secvenţa de momente de acest tip nu este limitată la o succesiune de multipli întregi ai unui pas de bază ca la modelele discrete. AS neglijează evoluţia modelului între salturi.

3.2.2.4.1 Modele orientate pe evenimente

Modelul structurat are următoarele componente:

- O mulţime D de componente active şi pasive
  $D = \{\alpha_1, \alpha_2, \ldots \alpha_A, \alpha_{A+1}, \ldots \alpha_n\}$, $\{\alpha_1 \ldots \alpha_A\}$ componente act
  $\{\alpha_{A+1}, \ldots \alpha_n\}$ componente pa
- Variabile descriptive. Pentru fiecare $\alpha \in D$ activă:
  $s_\alpha \in S_\alpha$ mulţimea stărilor lui α;
  $\sigma_\alpha \in R^+$ timpul până la ieşirea din starea α;
  Pentru $\alpha \in D$ pasivă:
  $s_\alpha \in S_\alpha$ mulţimea stărilor lui α
- Parametrii. Pentru $\alpha \in D$ activ :
  Urmaşii lui α: $\{\beta_1, \beta_2, \ldots \beta_m\} \subset D$, $\alpha = \beta_i$ pentru i dat.
- O funcţie de selectare a unui component dintr-o submulţime de componente active.
- Interacţiunea componentelor. Pentru fiecare componentă activă $\alpha \in D$ se defineşte o funcţie de tranziţie locală:

$$\delta_\alpha : \{\beta_1, \ldots \beta_m\} \rightarrow \{\beta_1, \ldots, \beta_m\}$$

Dacă α este valoarea funcţiei de selectare:

$$((s'_{\beta_1}, \sigma'_{\beta_1}), \ldots, s'_{\beta_m}) = \delta_\alpha((s_{\beta_1}, \sigma_{\beta_1}), \ldots s_{\beta_n})$$

este lista variabilelor descriptive ale urmaşilor lui α după apariţia evenimentului care a condus la α.

Strategia planificării evenimentelor

Presupunem că mulţimea stărilor componentelor active este finită şi o definim ca fel:

$S_\alpha = \{0, 1, \ldots, m-1\}$, α are m stări.

Funcţia de tranziţie δ_α este compusă în m funcţii $\{\delta^i_\alpha\}_{i=1,\ldots}$ fiecare descriind activitatea componentei

dacă a pornit din una din stările i. Fiecare δ_α^i este codificată separat ca o rutină, fiecare rutină fiind planificată şi executată de AS. Formal se scrie :

$$\delta_\alpha^i (s_\alpha, \ldots) = \delta_\alpha((i, s_\alpha)\ldots)$$

adică se fixează argumentul α considerînd celelalte variabile independente.

Procedura pentru tratarea evenimentului următor utilizează variabilele $\bar{s}_{\alpha_1}, \bar{s}_{\alpha_2}, \ldots, \bar{s}_{\alpha_n}$ pentru desemnarea stării prezente a componentelor $\alpha_1, \alpha_2, \ldots, \alpha_n$ ale modelului şi un ceas pentru planificarea activităţilor. Procedura mai utilizează o listă a evenimentelor următoare de tipul:

EVENIMENTUL i a lui α, t_α

EVENIMENTUL j a lui β, t_β

ordonată după valorile crescătoare ale timpilor t_α, t_β, ... Algoritmul de simulare (AS) presupune că perechea de elemente dintr-o linie desemnează rutina denumită E-i-α care trebuie executată cînd ceasul simulării ia valoarea t_α. Dacă mai mult de un eveniment este planificat pentru execuţie atunci regulile de decizie ale funcţiei de selectare determină cîştigătorul.

3.2.2.4.2 Model orientat pe baleiere.

Să presupunem că la momentul t componenta α este descrisă de variabilele $(s_\alpha, \sigma_\alpha)$. In cazul modelului cu baleiere se poate lăsa variabila timp de numărare inversă să ia valori negative cu mentiunea că pentru valori mai mici sau egale cu zero componenta α este gata să-şi înceapă activitatea dacă condiţiile din specificaţia ei sînt îndeplinite. Astfel la momentul t pot exista mai multe componente gata să-şi înceapă activitatea, fiecare cu timpul scurs de cînd α putea fi activată.

Componentele modelului:

- O mulţime $D = \{\alpha_1, \ldots, \alpha_n\}$ de componente active şi pasive.
- Variabile descriptive:
  pentru $\forall \alpha \in D$ activă : $(s_\alpha, \sigma_\alpha)$, $s_\alpha \in S$, $\sigma_\alpha \in R$
  pentru $\forall \alpha \in D$ pasivă $s_\alpha \in S_\alpha$
- Parametrii stărilor active $\alpha \in D$:
  urmaşii lui $\alpha = \{\beta_1, \ldots, \beta_m\} \subset D$
  predecesorii lui $\alpha = \{\bar{\beta}_1, \ldots, \bar{\beta}_{\bar{m}}\} \subset D$, α putînd aparţine ambelor categorii.
- Funcţia de selectare.
- Interacţiunea componentelor

  Funcţia de tranziţie locală δ_α (α activă) au forma

$$\delta_\alpha((\lambda_{\beta_1},\sigma_{\beta_1}),\ldots,\lambda_{\beta_n},(\lambda_{\overline{\beta}_1},\sigma_{\overline{\beta}_1}),\ldots,\lambda_{\overline{\beta}_{\overline{m}}}) =$$

$$= \begin{cases} f_\alpha((\lambda_{\beta_1},\sigma_{\beta_1}),\ldots,\lambda_{\beta_m},(\lambda_{\overline{\beta}_1},\sigma_{\overline{\beta}_1}),\ldots,\lambda_{\overline{\beta}_{\overline{m}}}) \\ \qquad \text{dacă } C_\alpha((\lambda_{\overline{\beta}_1},\sigma_{\overline{\beta}_1}),\ldots,\lambda_{\overline{\beta}_{\overline{m}}}) = \text{ADEVĂRAT} \\ ((\lambda_{\beta_1},\sigma_{\beta_1}-t(s)),\ldots\lambda_{\beta_m}) \\ \qquad \text{în caz contrar} \end{cases}$$

unde:

C_α este un predicat care atașează valoarea ADEVĂRAT sau FALS (nu ambele) condiției testate.

f_α este o funcție cu același domeniu ca δ_α reprezentînd acțiunea ce o exercită α asupra urmașilor ei dacă este activată;

t(s) este timpul de simulare.

- Procedura de simulare atașată modelului cu baleiere.

Se presupune că fiecărei funcții de tranziție δ_α i se implementează o rutină corespunzînd lui f_α (R-A-α) și o rutină pentru C_α (R-C-α). Starea componentei este reprezentată de λ_α. În plus componentelor active li se asociază T_α a cărui valoare t_α este interpretată ca fiind momentul următoarei, sau a ultimei activări a lui α : $t_\alpha = t + \sigma_\alpha$.

Selectarea se face pe baza unui sistem fix de priorități atașat componentelor $\alpha_1,\ldots,\alpha_A$ active. În procedură acest lucru se implementează printr-un cursor care baleiază lista componentelor active. Fiecare componentă din listă are atașată o rutină (R-C-α) corespunzătoare.

3.2.2.4.3 Modele de simulare a interacțiunilor

În principiu aceste modele reprezintă o sinteză a proprietăților modelelor precedente.

În acest caz funcția de tranziție δ_α care era în cazul modelului cu baleiere partiționată în două: predicatul condițiilor C_α și funcția f_α este acum partajată în mai multe părți ca în cazul modelului structurat. În acest caz nu se asociază o parte a funcției cu o stare din S_α ci cu o substare corespunzînd stării programului ce-l implementează pe δ_α :

- $S_\alpha = L_\alpha \times V_\alpha$ unde:
  - V_α = mulțimea variabilelor locale ale programului,
  - L_α = mulțimea etichetelor instrucțiilor curente $= \{0,1,2,\ldots M\}$.

În acest caz C_α^e poate fi definit prin fixarea valorii în C_α astfel :

$C_\alpha^\ell((v_\alpha, \sigma_\alpha), (\ell_{\beta_2}, v_{\beta_2}, \sigma_{\beta_2}), \ldots, v_{\beta_m}) =$

$= C_\alpha((\ell, v_\alpha, \sigma_\alpha), (\ell_{\beta_2}, v_{\beta_2}, \sigma_{\beta_2}), \ldots v_{\beta_m})$

dacă α este unmaș a lui α

$C_\alpha^\ell = C_\alpha$ ... pentru $\forall \ell \in L_\alpha$

în caz contrar

Similar:

$f_\alpha^\ell((v_\alpha, \sigma_\alpha), \ldots, v_{\beta_m}) = f_\alpha((\ell, v_\alpha, \sigma_\alpha) \ldots v_{\beta_m})$

C_α și f_α au fost descompuse în mulțimile $\{C_\alpha^\ell\}$ și $\{f_\alpha^\ell\}$. Funcția δ_α poate fi definită astfel:

In starea $\ell = 0$:

Dacă C_α^0 = ADEVĂRAT aplică f_α^0 altfel nici o operație (NOP);

In starea $\ell = 1$:

Dacă C_α^0 = ADEVĂRAT aplică f_α^0 altfel NOP;

...

In starea $\ell = M$:

Dacă C_α^M = ADEVĂRAT aplică f_α^M altfel NOP.

Procedura corespunzînd modelului de interacțiune a procesoelor.

Avînd definite pentru componentele active α mulțimile $\{C_\alpha^\ell\}$ și $\{f_\alpha^\ell\}$ care definesc funcția de tranziție δ_α, sîntem în măsură să elaborăm rutinele sau "procesele" corespunzătoare.

Procesul corespunzător lui α (P-α) reprezintă o secvență de instrucțiuni divizată în M segmente cîte unul pentru fiecare stare a programului. Fiecare segment este la rîndul lui divizat în două părți: segmentul de condiții și segmentul de acțiuni astfel

P-α:

1 Condiția 1 pentru α (C-1-α)

Acțiunea 1 pentru α (A-1-α)

Fă $v_{\beta_1} := v'_{\beta_1}$

.

.

.

Fă $v_{\beta_m} := v'_{\beta_m}$

Fă $v_\alpha := v'_\alpha$

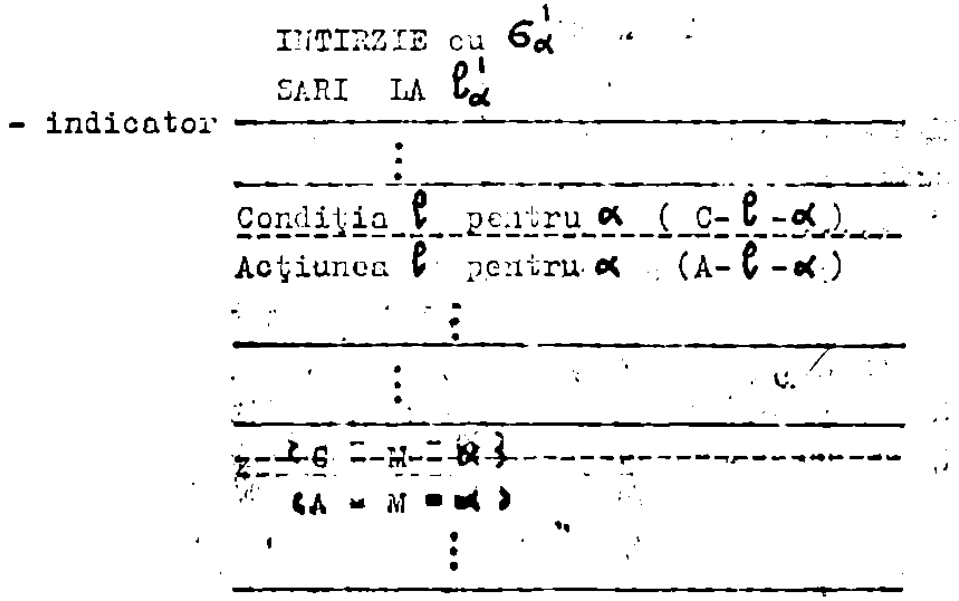
REPLANIFICA β_1 cu σ'_{β_1} LA ℓ'_{β_1}

.

.

.

REPLANIFICA β_m cu σ'_{β_m} LA ℓ'_{β_m}



Variabilele locale ale programului $v_{\alpha i}$ iau valorile concrete $\bar{v}_{\alpha i}$. Variabila de stare ℓ_α (a unei componente active) reprezintă poziţia indicatorului instrucţiei curente în procesul curent P -α dacă el este în execuţie, sau reprezintă următoarea instrucţie ce va fi executată în caz contrar. Punctele de activare desemnează primele linii ale fiecărui segment de cod din compunerea lui P-α . Astfel spre deosebire de strategiile modelelor precedente execuţia unui proces poate începe în oricare din cele m puncte de activare de exemplu în ℓ_α . După executarea acestui segment, execuţia va înceta, indicatorul va fi fixat în dreptul următorului punct de activare care poate fi

$\ell_\alpha + 1$ mai

ℓ'_α dacă apare o instrucţie de salt. Un asemenea α - indicator este asociat fiecărei rutine P-α pentru fiecare componentă activă .

Algoritmul de simulare va menţine o listă a activităţilor viitoare (LAV) cu elemente de tipul $(\alpha , \ell_\alpha , t_\alpha)$. Un asemenea triplet este interpretat de AS astfel : α va fi planificat să-şi înceapă execuţia de la eticheta ℓ_α la momentul t_α . O a doua listă , lista activărilor curente (LAC) de aceeaşi formă conţine fie componentele a căror moment de prelucrare tocmai a sosit, fie componente al căror rînd a venit mai demult dar a căror condiţii de activare nu au fost satisfăcute încă. Ca în cazul modelului cu baleiere lista are o bază, un vîrf şi un indicator de baleiere (IBL) indicînd elementul în curs de prelucrare. AS este prevăzut cu un ceas de simulare. In activitatea de baleiere se presupune că selectarea respectă o ordine strictă de priorităţi pe mulţimea componentelor active.

Procedura asociată modelului:

* INITIALIZARE :

1 CEAS = t_0 (t_0 = momentul inițial);

2 variabilelor de program li se dau valorile inițiale ($\bar{v}_{\alpha_1}, \ldots \bar{v}_{\alpha_n}$)

3. - Pentru fiecare α activă se fixează ℓ_α inițial;
   - dacă valoarea inițială a lui $\sigma_\alpha > o$ se plasează elementul (α , ℓ_α, $t_o + \sigma_\alpha$) în LAV;
   - se ordonează LAV în ordinea creșcătoare a timpilor;
   - dacă $\sigma_\alpha \leq o$ se plasează (α, ℓ_α, $t_o + \sigma_\alpha$) în LAC;
   - se ordonează LAC în ordinea priorității componentelor, prioritatea maximă în vîrf, cea minimă la bază;

* FAZA DE BALEIERE :

4. Se fixează IBL în vîrful LAC;

5. - Se deplasează IBL către bază pînă la identificarea primei componente activabile;
   - Pentru fiecare triplet (α , ℓ_α, t_α) boleiat se execută începutul lui P-α începînd de la eticheta ℓ_α;
   - Dacă executarea rutinei R-C-α dă valoarea FALS, IBL se decrementează, în caz contrar ia valoarea curentă $\bar{\alpha}$;
   - Se scoate elementul ($\bar{\alpha}$, $\ell_{\bar{\alpha}}$, $t_{\bar{\alpha}}$) din LAC.

* TRANZITIA STĂRII :

6. - Se continuă execuția P-α cu rutina R-A-$\bar{\alpha}$ de la eticheta $\ell_{\bar{\alpha}}$. Acest segment implementează funcția $f_{\bar{\alpha}}^{\ell_{\bar{\alpha}}}$ care asociază fiecărui urmaș β al lui $\bar{\alpha}$ starea (v'_β , ℓ'_β, σ'_β) , lui v_β i se asociază valoarea v'_β ;
   - Dacă β este o componentă activă și $\sigma'_\beta > o$ tripletul (β, ℓ'_β, $t + \sigma_\beta$) se trece în LAV în locul determinat de $t + \sigma'_\beta$;
   dacă $\sigma'_\beta \leq o$ tripletul se înscrează în LAC într-o poziție corespunzătoare cu prioritatea lui β (REPLANIFICA β, σ'_β la ℓ'_β, $\beta \neq \alpha$).
   - se realizează σ'_α cu instrucția INTIRZIE CU σ'_α și se trece la $\bar{\alpha}$ prin instrucția SARI LA $\ell'_{\bar{\alpha}}$.

* SFIRSITUL BALEIERII ?:

7. Dacă IBL nu a ajuns la baza LAC sari la 5.

* ACTUALIZAREA CEASULUI :

8. Avansează CEASUL până la t = timpul evenimentului viitor (TEV) din primul triplet din LAV;

X ACTUALIZAREA LAC :

9. Elimină tripletul cu timpul = TEV din LAV şi înscrează-l în LAC într-o poziţie corespunzătoare priorităţii sale.
   SFIRSIT ?

10. Dacă TEV > t_1 timpul fixat pentru terminarea simulării atunci STOP; în caz contrar sari la 5 .

GPSS (General Purpose Simulation System) este un limbaj orientat către construcţia de modele prin interconectarea de module tip. O descriere în GPSS a unui model este obţinută prin plasarea secvenţială a unor instrucţii de descriere. Fiecare din ele corespunde unui bloc din schema logică GPSS. Schema logică GPSS este asemănătoare ordinogramei unui program convenţional. Fluxul comenzilor este în acest caz cu totul deosebit în locul numărătorului de program care indică trecerea de la o instrucţie la alta, în GPSS pot exista mai multe numărătoare distincte sau "tranzacţii".

Fiecare instrucţie (bloc) de descriere reprezintă de fapt o macroinstrucţie. Cînd tranzacţia parcurge un bloc se execută codul macroinstrucţiei asociate. Uzual instrucţia presupune efectuarea anumitor teste. Dacă ele nu dau rezultate satisfăcătoare tranzacţia rămîne în bloc şi numai au loc alte acţiuni. Dacă condiţiile sînt satisfăcute se trece la partea de prelucrare a instrucţiei. Ulterior tranzacţia trece la un alt bloc specificat de instrucţia executată. Datorită faptului că o schemă logică GPSS poate fi parcursă simultan de mai multe tranzacţii se impune existenţa unui control central care să selecteze care din tranzacţiile concurente să-şi înceapă execuţia instrucţiei aferente blocului disputat. GPSS oferă facilităţi pentru introducerea şi ieşirea tranzacţiilor din schema logică : blocurile GENERATE şi TERMINATE. Pentru a introduce noţiunea de timp ataşată unei prelucrări se utilizează blocul ADVANCE care provoacă o întîrziere ce poate fi specificată înainte de trecerea tranzacţiei la blocul următor. In cazurile în care tranzacţiile se blochează la un anumit bloc ele se aşează într-o coadă FIFO.

Algoritmul de simulare descris anterior poate fi implementat în GPSS. Corespondenţa dintre procedura descrisă şi simularea prin GPSS este dată în tabelul de mai jos:

Modelul de interacţiune a proceselor:	GPSS :
Componente pasive	Entităţi de echipament
Numele proceselor $\alpha_1, \alpha_2 \ldots$	Numărul tranzacţiei 1,2,...
P-α	Program GPSS
I_α	Parametrii tranzacţiei i
Indicatorul instrucţiei curente	Blocul următor abordat de tranzacţie
LAV	FEC
LAC	CEC
Segment din procesul P-α asociat cu punctul de activare ℓ	Codul asociat blocului instrucţiei cu adresa ℓ
	... ca intrarea unei tranzacţii într-un bloc (de ex. SEIZE j implică testarea facilităţii j j=o (liberă) j=1 (ocupată))
$A - \ell - \alpha$	Activităţile blocului ℓ implicate de intrarea tranzacţiei
ÎNTÎRZIE CU σ_α	ADVANCE σ_α
SARI LA ℓ_α	TRANSFER ℓ

Prin prisma limbajului GPSS mai multe indicatoare de adresă (tranzacţiile) încearcă să execute un program începînd din diverse puncte. Pentru a corespunde modelului descris, se asociază o copie a programului sursă GPSS fiecărei tranzacţii. Instrucţiile bloc din GPSS constituie macroinstrucţii de forma segmentelor de cod propuse în model la etichetele 1,... M. Ele sînt de trei tipuri principale:

1. Nu pot nega intrarea unei tranzacţii şi nu pot provoca întîrzieri (ASSIGNi,j ; RELEASE i, LEAVEi,j; LOGIC i,j; TRANSFER Cu excepţia lui TRANSFER aceste blocuri conţin numai segmentul de tip A $-\ell-\alpha$; TRANSFER conţin numai segmentul de salt.

2. Poate nega intrarea unei tranzacţii (SEIZEi; ENTERi,j; GATEi,j; TESTi,jk). Cu excepţia lui GATE şi TEST aceste blocuri conţin segmentele de tip C $-\ell-\alpha$ şi A $-\ell-\alpha$; GATE şi TEST conţin în plus segmente de salt.

3. Nu pot nega intrarea tranzacţiei dar poate fi întîrziată (ADVANCEij). Aceste blocuri conţin numai segmentul de întîrziere.

In tabela următoare se descrie modul de prelucrare a blocurilor GPSS:

- Tipul : 1.
- Descriere: tranzacţia TRZ (.) trece prin blocul BLOC(.) în timp nul.
- Activităţi implicate: (TRZ(.), BLOC(.), -) este indicat de IBL în CEC, se execută segmentul de tip A - ℓ - α şi tripletul este înlocuit cu (TRZ(.), BLOC(. + 1), -) dacă (TRZ(.), ℓ, -) dacă apare TRANSF iar IBL al CEC este readus în vîrful CEC.
- Tipul : 2.
- Descriere : tranzacţiei TRZ(.) i se interzice intrarea în BLOC (.)
- Activităţi implicate : IBL al CEC indică (TRZ(.), BLOC(.), -), se execută segmentul tip C - ℓ - α şi se echivalează FALS. Singurul efect este că IBL este decrementat cu unu.
- Tipul : 3.
- Descriere : TRZ(.) este întîrziată la trecerea prin BLOC(.)
- Activităţi implicate : (TRZ(.), BLOC(.), -) indicat de IBL al CEC este înlăturat din CEC iar în FEC se trece (TRZ(.), BLOC (. + 1), t + σ), t fiind timpul curent de simulare.

Prelucrările decurg în două faze. In prima tripletul ataşat tranzacţiei iminente este scos din FEC şi introdus în CEC. In a doua fază CEC este baleiat pentru posibile deplasări ale tranzacţiei în program corespunzător tabelului anterior. Cînd toate tranzacţiile sînt blocate (toate segmentele de condiţii dau rezultatul FALS) se reia prima fază.

Se observă că faţă de modelul propus GPSS prezintă dezavantajul că tranzacţiile (care corespund proceselor) nu pot modifica direct parametrii altor tranzacţii. Modelul propus permite replanificarea altui proces. In GPSS nu există operaţie corespunzătoare pe care o tranzacţie să o efectueze asupra FEC.

GPSS prezintă însă următoarele avantaje faţă de modelul propus:

1. Există pentru introducerea în simulare de noi tranzacţii (GENERATE şi SPLIT) şi pentru înlăturarea de tranzacţii TERMINATE .

2. Există blocuri pentru poziţionarea IBL (BUFFER) şi pentru modificarea priorităţii unei tranzacţii (PRIORITY).

3. Ordonarea tranzacţiilor în CEC nu se face numai prin priorităţi ca şi în model. Mai multe tranzacţii pot să aibe aceeaşi prioritate. CEC este ordonat după criteriul priorităţilor, după criteriul timpului de sosire în CEC şi după numărul de ordine al tranzacţiei.

Aceste facilităţi permit adăugarea de noi componente la model cît şi de modificarea dinamică a funcţiei de selectare. Detalii privind GPSS sînt prezentate în [RAD].

3.3. ALGORITMII DE PLANIFICARE, SINCRONIZARE SI COMUNICARE INTRE PROCESE

Transpunerea pe un SMM a modelului matematic discret al unui SA sau a unei componente ale sale este o operaţie ce pretinde o viteză mare de prelucrare şi capacitatea de a efectua activităţi concurente de o manieră paralelă. Modelele de simulare a SA prezintă un grad înalt de paralelism inerent care reflectă modul de operare a sistemului simulat. SMM permit un grad înalt de paralelism pe durata execuţiei cu condiţia existenţei unui executiv care să facă posibilă gestionarea paralelă a proceselor în execuţie: comunicaţiile interprocese, activarea şi întreruperea proceselor active şi sincronizarea lor.

Arhitecturile derivate din MA propus se bazează din punct de vedere fizic pe una sau mai multe magistrale partajate şi pe utilizarea unor memorii comune obişnuite sau cu două porturi. Din punct de vedere logic arhitecturile se bazează pe o topologie stăpîn/sclavi, ele încorporînd un microprocesor stăpîn şi un număr de microprocesoare sclavi. Dat fiind că arhitecturile propuse fac parte din clasa SMM propriu-zise (puternic cuplate) ele implică interacţiuni complexe între µP, de bandă largă, şi solicită plasarea lor pe un spaţiu restrîns şi utilizarea memoriilor comune pentru transferul datelor şi pentru memorarea programelor.

In concluzie SMM de acest tip sisteme stăpîn/sclav cu comandă centralizată.

3.3.1. EXECUTIVUL

Executivul are rolul de a transforma hardware-ul şi software-ul de bază al SMM într-o maşină virtuală care să asigure facilităţi pentru implementarea trăsăturilor de concurenţă şi paralelism aşteptate de la sistem. Apărut în faze anterioare, pe timpul dezvoltării sistemelor cu multiprogramare, executivul a devenit o componentă esenţială în proiectarea şi realizarea sistemelor de operare pentru S.M. In asemenea sisteme comutarea sau tranziţia de la un task la altul, planificarea taskurilor şi a proceselor, gestionarea cozilor şi implementarea comunicaţiilor intertask cît şi primitivele de sincronizare sînt doar cîteva părţi componente ale unui executiv pentru SMM.

Datorită rolului său central executivul poate deveni partea slabă a SMM pe care îl deservește limitîndu-i serios posibilitățile. Pentru a le ridica performanțele rutinele executivului sînt scrise uzual în cod de asamblare (pentru a minimiza timpul cheltuit în executiv pe durata apelurilor la executiv solicitate de programele în execuție). Din motive de excluziune cele mai multe primitive trebuiesc executate cu toate întreruperile externe blocate. Aceasta limitează în mod serios răspunsul în timp real al SMM.

Avantajele utilizării unui executiv unic nedistribuit pentru SMM plasat într-un supervisor care controlează cîteva procesoare subordonate sînt:

- executivul fiind unicul gestionar al tuturor structurilor de date vitale și critice (ca de exemplu tabelele de activare a taskurilor, cozile de taskuri), problemele legate de conflictele și excluziunea mutuală în accesul la aceste structuri sînt mult reduse;

- eliminarea necesității existenței unor primitive de control al dreptului de acces la resurse;

- primitivele existente vor avea rolul de a semnala executivului că a apărut o solicitare de întrerupere, el fiind acela care traduce semnalul într-un apel de rutină pe care o planifică apoi în execuție;

Pentru implementarea ultimului aspect menționat fiecărui dispozitiv ce poate lansa o cerere de întrerupere îi este asociată o rutină de nivel inferior executată pe procesorul local.

Aceasta nu are alt rol decît de a răspunde în timp real solicitării dispozitivului I/E și de a poziționa un semafor care semnalizează executivului că a apărut o cerere de întrerupere. Executivul detectează apariția cererii de întrerupere prin verificarea semafoarelor și lansează un apel la taskul care va prelucra datele respective. După poziționarea semaforului procesorul local reia execuția taskului din locul în care a fost întrerupt fără să poarte răspunderea consecințelor poziționării semaforului, executivul fiind acela care decide cînd vor fi prelucrate datele asociate întreruperii. Procedeul garantează evoluția logică a taskului aflat în execuție (fără întreruperi) și asigură că executivul menține controlul complet al SMM.

Detalii privind proiectarea și realizarea unui executiv pentru un SMM sînt prezentate în [...1], [...2].

3.3.2. ÎNTRERUPERILE ŞI MECANISMUL DE RENDEZ-VOUS

Mecanismul întreruperii este cunoscut actualmente sub următoarea formă:

- cererea de întrerupere este lansată prin hardware;
- dacă este acceptată ea apelează o rutină software de deservire a întreruperii (aceasta este o secţiune de program care trebuie să fie executată când apare o modificare asincronă a stării unui dispozitiv o solicitare necondiţionată a deservirii sau un impas în cadrul procesorului central însuşi);
- rutina de deservire este precedată de comutarea contextului şi de alte funcţii sub controlul sistemului de operare;

în multe situaţii, mai ales în cazul sistemelor mari mecanismul de întrerupere este mascat pentru utilizator de către sistemul de operare. În cazul sistemelor mici, mai ales bazate pe microprocesoare, utilizatorul are acces la interfeţele hardware în speţă la mecanismul întreruperilor.

Întreruperile pot fi o sursă de probleme în proiectarea unui sistem. Gradul de dificultate al acestor probleme este proporţional cu nivelul arhitectural la care hardwareul asigură suport pentru sistemul de operare implementat.

Mecanismul întreruperilor nu este în sine un concept fundamental ci mai degrabă o tehnică care s-a dovedit foarte utilă în aria monoprocesoarelor operînd în regim de multiprogramare. Pentru sistemele multiprocesor operînd în mod paralel este puţin probabil că modificarea stării unui dispozitiv să cauzeze "întreruperea" vreunui proces. Mecanismul întreruperilor mai este utilizat în SMM de către perifericele I/E care gestionează taskurile de introducere şi extragere de date. Procesoarele centrale însă îşi comunică modificările de stare utilizînd alte mecanisme.

În timp ce mecanismul hardware indicînd modificarea stării rămîne esenţial, conceptele software ca "rutină de deservire a întreruperii" şi "comutare de context" (salvarea stării programului) nu mai sînt compatibile cu tendinţele moderne din domeniul prelucrării concurente şi a limbajelor de nivel înalt corespunzătoare.

Modificarea stării poate fi semnalată de un semafor cu rolul de a indica unui proces conţinînd o declaraţie de tipul "aşteaptă o întrerupere" că a avut loc un "rendez-vous", dacă procesul era în aşteptare, sau că el va avea loc când procesul ajun-

la următoarea declarație "așteaptă o întrerupere" [DER 1].
In cele mai multe cazuri modificarea semaforului și întreruperea rezultantă indică posibilitatea unui transfer de date.

Revenind asupra mecanismului întreruperilor în majoritatea textelor întîlnite se omite examinarea efectelor secundare generate de întreruperi. Întreruperile pot introduce nedeterminări grave în cazul unor programe dacă nu se introduc măsuri speciale de precauție; dacă posibilele repercursiuni, mai ales în cazul STR nu sînt judicios evaluate, utilizarea întreruperilor trebuie evitată.

Trecere în revistă principială a mecanismului întreruperilor

O cerere de întrerupere poate apărea cînd:

- un dispozitiv periferic își modifică starea (de exemplu un dispozitiv I/E, ceasul de timp real, circuitul de sesizare a căderii tensiunii, anumite butoane de comandă, etc.);
- un procesor intră într-o situație excepțională (depășire aritmetică, adresă inexistentă, oprire pe adresă, depășirea limitelor stivei, etc.).

Modificarea stării dispozitivelor I/E se poate datora
- posibilității de a transfera un cuvînt (un octet) prin instrucțiuni ale programului ;
- inițierea sau încheierea transferului unui bloc de date prin dispozitive cu acces direct la memorie;
- apariției unor situații și modificări în funcționarea periferiocului nelegate de transferul de date, dar care solicită atenția.

Cererea de întrerupere va fi deservită prin rutina corespunzătoare dacă
- este nemascabilă;
- întreruperile sînt autorizate;
- nu există cereri de întrerupere mai prioritare.

Efectul autorizării întreruperii se manifestă prin:
- suspendarea programului activ;
- memorarea informației de stare a procesorului (cît și a adresei de revenire) în stivă;
- inhibarea întreruperilor ulterioare (dacă e cazul);
- decodificarea vectorului de întreruperi pentru a obține noua stare a procesorului (inclusiv saltul la rutina de deservire a întreruperii).

Efectul instrucţiei de "revenire din întrerupere" constă în:

- recuperarea din stivă a stării procesorului pentru programul întrerupt;
- saltul la adresa de revenire;
- autorizarea întreruperilor ulterioare (dacă e cazul);

Sistemul de operare trebuie să decidă:

- din care stivă se recuperează starea procesorului dacă
- care proces este relansat în execuţie (presupunînd că au fost întrerupte mai multe procese).

Vectorul de întrerupere constă uzual din:

- adresa rutinei de deservire a întreruperilor ;
- informaţia referitoare la starea procesorului (masca de întreruperi, starea unor registre sau fanioane etc.).

Întreruperile sînt adesea utilizate în cadrul sistemelor de operare. Precauţii speciale trebuie să prevîntîmpine efectele colaterale ce pot cauza nedeterminări grave şi nedorite în programele utilizator.

In cadrul sistemelor bazate pe microprocesoare utilitatea întreruperilor apare în următoarele cazuri de bază:

i) Tratarea situaţiilor de excepţie. Aceste situaţii genorează abandonarea anormală a unui program şi sînt adesea nemăscabile: detectarea căderii tensiunii; adresarea unor locaţii inexistente; utilizarea incorectă a stivei; puncte de oprire pe adresă impuse de utilizator;

ii) Obţinerea unor cicluri izocrone;

iii) Crearea unor dispozitive de tată I/E paralel cu execuţia altor programe ;

iv) Execuţia concurentă a mai multor programe pe acelaşi procesor.

Mecanismul de "rendez-vous"

Tendinţa actuală este să se renunţe la rutinele de deservire a întreruperii în favoarea unei noţiuni indicînd "rendez-vous" hardware-software[YOU 1]. Un comentariu "aşteaptă o întrerupere" în programul structurat (pseudocod) desemnează punctul în care ar trebui să apară un rendez-vous. O întrerupere va produce continuarea procesului care era deja început dar fusese suspendat în aşteptarea unei modificări a stării unui periferic. Rutina de deservire a întreruperii (care acţionează cînd apare o modificare a stării unui periferic) reprezintă rezolvarea clasi-

a interfeţei hardware-software în cazul unei întreruperi. Abordările recente aduc o inversare faţă de rezolvarea anterioară. Procesul se autosuspendă întotdeauna la o locaţie de program specifică în aşteptarea unei modificări de stare a periferiecului, întreruperea va cauza revenirea la locaţia dinainte cunoscută din program.

Întreruperile nu vor fi utilizate decît dacă sînt absolut necesare deoarece ele implică un nivel de complexitate mult superior codului determinist secvenţial. Utilizarea tehnicii de rendez-vous are ca scop minimizarea nedeterminărilor introduse de întreruperi.

Un anumit dispozitiv poate cauza mai multe tipuri de întreruperi. In acest scop este necesar ca programul să poată anticipa următoarea întrerupere posibilă.

In cele mai multe cazuri declaraţia "aşteaptă o întrerupere" se plasează într-o buclă. Codul generat pentru "aşteaptă ..." trebuie să includă:

- fixarea părţii de adresă a vectorului de întreruperi la adresa următoarei declaraţii din program, dacă aceasta nu s-a putut face la iniţializare;

- revenirea la procesul întrerupt utilizînd o "revenire din întrerupere" sau o instrucţie similară (dacă nici un proces nu a fost întrerupt pot fi utilizate fie instrucţii "aşteaptă o întrerupere" fie o buclă în care se aşteaptă modificarea fanionului corespunzător).

Prima declaraţie "aşteaptă..." dintr-un proces poate necesita un cod aditional pentru a anula alt proces. Dacă celălalt proces nu fusese pornit trebuie să se execute în locul instrucţiei "revenire din întrerupere" o instrucţie de tipul "sari la începutul altui proces".

Situaţia 1. Bucle infinite.

Acestea sînt deseori dezutile în aplicaţiile de prelucrare în timp real. Programul (în pseudocod) ia forma:

```
începe
    iniţializarea programului,
    repetă
        "aşteaptă o întrerupere" de la anumit dispozitiv,
        colectarea unui eşantion de date;
        prelucrarea eşantionului respectiv;
    pînă la nesfîrşit;
sfîrşit.
```

Declarația "așteaptă o întrerupere" de la ceasul de timp real poate fi implementată în două moduri:

- fie printr-o instrucțiune mașină "așteaptă..." cu întreruperile inhibate;
- fie prin execuția unui alt proces care nu depinde de timp de exemplu "sari înapoi la propria adresă" cu întreruperile autorizate. In acest caz vectorul de întreruperi trebuie fixat la adresa primei instrucții urmînd declarația "așteaptă..." din cadrul buclei.

Situația 2. Blocuri de date în prelucrarea semnalelor. Programul poate avea forma

```
.
.
.
inițializare;
colectează n eșantioane;
repetă
      colectează n eșantioane
      prelucrează cele n eșantioane anterioare
până apare condiția de sfîrșit;
prelucrează cele n eșantioane anterioare
.
.
.
terminare;
.
.
.
```

Execuția codului pentru "prelucrarea celor n eșantioane prealabile" trebuie să dureze mai puțin decît execuția codului "colectează n eșantioane". Codul pentru colectarea a n eșantioane este:

```
început
      repetă
            așteaptă o întrerupere ;
            colectează un eșantion ;
            i : = i + 1
      până  i = n;
sfîrșit;
```

Durata execuției codului pentru testarea celor două condiții până și pregătirea buclei interioare trebuie să dureze suficient de puțin ca să permită apariția declarației "așteaptă..." înaintea următorului impuls de la ceasul de timp real. Dacă aces-

ta apare primul se va semnaliza o situaţie de excepţie de tipul "pierdut un impuls". Este preferabil ca programul să fie conceput astfel ca "pierderile" să nu apară. Dacă ele apar totuşi, nu e de dorit ca să producă o întrerupere de excepţie urmată de abandonarea programului. Pentru a evita terminarea eronată de acest tip în bucla (repetă) exterioară se va introduce un al treilea proces paralel pentru tratarea întreruperii de tip "pierdut impuls" :

colectează n eşantioane
    prelucrează cele n eşantioane prealabile
    prelucrează condiţia "pierdut impuls"

Situaţia 3. Puncte de intrare multiple pentru o întrerupere.

```
.
.
.
repetă
.
.
.
    aşteaptă o întrerupere de la ceasul de timp real
A:
.
.
.
 aşteaptă o întrerupere de la ceasul de timp real
B:
.
.
.
până la condiţia de sfîrşit;
.
.
.
```

In acest caz vectorul de întrerupere nu poate fi fixat numai în faza de iniţializare a programului. El trebuie fixat în mod diferit pentru fiecare din declaraţiile "aşteaptă..." din buclă. Vectorul de întreruperi trebuie fixat în cadrul codului generat pentru "aşteaptă..." înainte ca controlul programului să fie predat vreunui alt task paralel, astfel încît următoarea întrerupere de la această sursă să cauzeze saltul corect la una din cele două etichete A sau B. Dacă nu există un alt task paralel, fiecare din cele două declaraţii "aşteaptă..." trebuie implementată cu un [illegible]

Situaţia 4. Detectarea căderii tensiunii (situaţii de excepţie).

Procesul corespunzător este:

începe
    iniţializare;
    aşteaptă o întrerupere excepţională ;
    prelucrează excepţia;
sfîrşit

S-ar putea ca excepţia să nu apară niciodată astfel încît programul de detectare a căderii de tensiune să nu fie niciodată complet executat. Iniţial însă sistemul trebuie să fie corect iniţializat iar vectorul de întreruperi corect fixat pentru o întrerupere de excepţie.

3.3.3. Sincronizarea proceselor

Interacţiunea proceselor se manifestă fie prin lansarea comenzilor de la supervizor fie prin necesitatea de partajare a unor variabile comune, matrici de date, proceduri, etc. denumite la un loc variabile sistem. Ele sînt plasate în zone predestinate (memorii comune) denumite în continuare depozite sistem.

Procesele de transpunere a modelelor discrete sînt executate în paralel deci corectitudinea rezultatului complex al operaţiei depinde de viteza relativă de execuţie a diverselor procese componente. Viteza proceselor care operează asincron este puternic afectată de frecvenţa de interacţiune. Aceasta poate conduce chiar la rezultate dezastruoase dacă variabilele sistem sînt utilizate de-o manieră întîmplătoare. Pentru o cooperare reuşită procesele trebuie să-şi sincronizeze activitatea în puncte specifice de RV (rendez-vous). Acestea sînt punctele în care sînt referite variabilele sistem. Un proces de transpunere a unui model trebuie împiedecat să treacă de anumite puncte de interacţiune înainte ca RV cu procesul partener (care are de dus şi el la bun sfîrşit o sarcină pînă în acel punct) să fi avut loc.

In acest tip de SMI partajarea variabilelor sistem între procesele paralele conduce la referiri multiple a aceleiaşi variabile sistem situaţie ce conduce la o cursă critică, în sensul că planificarea a două procese este atît de critică în cît ordinea în care sînt alocate conduce la rezultate distincte şi ca atare variabila sistem va lua valori nedeterminabile. Soluţia este să se impună o ordine strictă de referire a variabilelor sistem prin sincronizarea precisă a proceselor ce le utilizează.

Sincronizarea proceselor se poate face prin introducerea unei relaţii de ordine cronologică de tip precedenţă. În cazul arhitecturilor destinate transpunerii modelelor matematice a SA, secţiunile critice apar numai legate de variabilele sistemului de resursele fizice, care este cazul în sistemele de operare clasice.

3.3.1 Definirea unei relaţii de ordine pe mulţimea evenimentelor.

Relaţia de ordine definită pe mulţimea evenimentelor este necesară pentru a garanta calcule coerente cît şi pentru a garanta corectitudinea şi integritatea rezultatelor.

Se presupune că fiecare proces de simulare S a unui SA constă dintr-o secvenţă finită de perioade de prelucrare aranjate într-o ordine predefinită. Perioadele de prelucrare sînt urmate de perioade de transfer de mesaje, perioade de interacţiune sau de rendez-vous (RV). Aceste perioade de transfer vor fi denumite evenimente de tip rendez-vous sau pe scurt r-evenimente.

Pe mulţimea evenimentelor r notată RV se poate defini o relaţie astfel:

$\rightarrow := (G, RV, RV)$ unde $G \subset RV \times RV$

Dacă r aparţine procesului S_i, se notează $r^i \in S_i$

D1. Se spune că $r_1 \rightarrow r_2$ (r_1 îl precede pe r_2) dacă:

i) r_1 şi r_2 sînt r-evenimente ale aceluiaşi proces $r_1, r_2 \in S$ şi r_1 are loc înaintea lui r_2;

ii) r_1^i este r-evenimentul de emitere a unui mesaj $r_1^i \in S_i$ şi r_2^j este r-evenimentul de recepţie al aceluiaşi mesaj $r_2^j \in S_j$, $S_i \neq S_j$.

D2. Se spune că r_1^i şi r_2^j, $r_1^i \in S_i$ şi $r_2^j \in S_j$, $S_i \neq S_j$ sînt simultane şi deci se pot executa în paralel dacă $r_1^i \rightarrow r_2^j$ şi $r_2^j \rightarrow r_1^i$.

Studiul proprietăţilor relaţiei $\rightarrow$.

Pentru a formaliza studiul relaţiei r este necesar să se introducă un atribut de timp asociat fiecărui r-eveniment.

D3 Se numeşte atribut de timp ataşat unui r-eveniment $r \in S_i$, momentul de timp la care apare acest eveniment, atributul este notat $t_i \langle r \rangle$, $t_i \langle r \rangle \in R$.

În cadrul activităţii de simulare, $t_i \langle r \rangle$ desemnează timpul de simulare la care r urmează să aibă loc. Se specifică că atributul de timp $t_i \langle r \rangle$ este specific r-evenimentului $r \in S_i$.

Po. Pentru orice r-eveniment r_1 şi r_2, $r_1^i \in S_i$ şi $r_2^i \in S_i$ fie $r_1^i \to r_2^i$ atunci $\{t_i\langle r_1^i\rangle < t_j\langle r_2^i\rangle\}$

fie $r_2^i \to r_1^i$ atunci $\{t_j\langle r_2^i\rangle < t_i\langle r_1^i\rangle\}$

Observaţie: în cazul în care apartenenţa unui r-eveniment rezultă prin specificarea atributului său de timp, se renunţă la specificarea apartenenţei prin indicele superior.

P1. Fie $r_1^i \in S_i$ şi $r_2^i \in S_i$.

Se spune că $r_1^i \to r_2^i$ dacă şi numai dacă atributul de timp a lui r_1^i este anterior atributului evenimentului r_2^i, $t_i\langle r_1\rangle < t_i\langle r_2\rangle$. Egalitatea nu poate apărea în cadrul aceluiaşi proces pentru r-evenimente diferite.

P2. Fie $r_1^i \in S_i$ şi $r_2^j \in S_j$

Se spune că $r_1^i \to r_2^j$ dacă şi numai dacă $t_i\langle r_1\rangle \leq t_j\langle r_2\rangle$

P3. Relaţia este reflexivă.

Dem 3: $\Delta_G \subseteq G$ sau $r \to r$, $r \in S$.

Conform P_1: $t\langle r\rangle \leq \langle t\ r\rangle$ q.e.d.

P4. Relaţia este tranzitivă.

Dem 4: $GoG \subset G$ sau $r_1 \to r_2$ şi $r_2 \to r_3$ implică $r_1 \to r_3$

Dat fiind că în demonstraţie nu se face apel la apartenenţa la procese a r-evenimentelor s-a renunţat la indicii superiori. Conform P1 şi P2

$$t\langle r_1\rangle \leq t\langle r_2\rangle \text{ şi } t\langle r_2\rangle \leq t\langle r_3\rangle$$

$$\Rightarrow t_1\langle r_1\rangle \leq t\langle r_3\rangle \text{ q.e.d.}$$

P5. Mulţimea RV a r-evenimentelor este o mulţime preordonată (Conform P3 şi P4).

Mulţimea RV putînd fi preordonată în moduri diverse se consideră mulţimea preordonată cu fiind perechea $(RV, \to)$.

P6. Mulţimea preordonată $(RV, \to)$ este ordonată.

Dem 6: $G \cap G^{-1} = \Delta_G$ (antisimetria)

$r_1 \to r_2$ şi $r_2 \to r_1$ implică

$t\langle r_1\rangle \leq t\langle r_2\rangle$ şi $t\langle r_2\rangle \leq t\langle r_1\rangle$, adică

$$t\langle r_1\rangle = t\langle r_2\rangle$$

deci r_1 şi r_2 coincid q.e.d.

$(RV, \to)$ fiind o mulţime ordonată, despre două r-evenimente care sînt în relaţia de precedenţă, adică $r_1 \to r_2$ sau $r_2 \to r_1$ se zice că sînt comparabile prin relaţia de precedenţă.

P7. Multimea (RV,→) este total ordonată.

Dem 7. $G \cup G^{-1}$ = RV x RV vezi Po.

P7 traduce faptul că orice două elemente din RV sînt comparabile.

P8. Mulţimea (RV,→) admite un element iniţial respectiv final:

$\forall r \in RV \quad r_0 \to r$ respectiv $r \to r_f$

Dem. 8. Conform P5 orice mulţime preordonată admite un element iniţial (final).

P9. Fie RV şi S_i (procesul i) o parte a sa.

Orice $S_i \subset$ RV admite un minorant (majorant) notat r_m^i (r_M^i).

Dem 9. $\forall r^i \in S_i \quad \exists\, r_m^i$ astfel încît $r_m^i \to r^i$ ($r^i \to r_M^i$) conform P8.

Mulţimea (RV,→) are un eveniment iniţial (final) dacă şi numai dacă are un minorant (majorant) ; evenimentul iniţial (final) este unic.

Plo. Mulţimea (RV,→) este bine ordonată (orice proces al său are un element iniţial).

Modul de sincronizare a proceselor concurente impus de regiunile critice.

Procesul de sincronizare este asigurat de o procedură specifică a executivului [BRE 1][BRE 2].

Dacă $\{S_1[X_1, t_1] \| S_2[X_1, t_2]\}$ sînt două procese de simulare concurente, X_1 fiind o variabilă sistem (care defineşte o secţiune critică comună) şi $t_1 = t_1\langle X_1\rangle$, $t_2 = t_2\langle X_1\rangle$ sînt timpii de simulare asociaţi atunci dacă

$$t_1 \leqslant t_2$$

din (P4) rezultă că

$$X_1^1 \to X_1^2$$

Astfel în mediul de simulare s-a impus o relaţie de ordine totală care implică că accesul la variabila comună X, din procesul S_1 trebuie să aibă loc înainte de a permite accesul la aceeaşi variabilă, procesului S_2. Indiferent de vitezele relative de execuţie ale lui S_1 şi S_2, S_2 trebuie să-şi suspende execuţia înainte de a intra în secţiunea sa critică, în timp ce lui S_1 i se permite să-şi execute secţiunea critică asociată. Cînd S_1 părăseşte

secţiunea sa critică executivul va înştiinţa imediat procesorul pe care este rezident S_2 că poate să-şi reia execuţia.

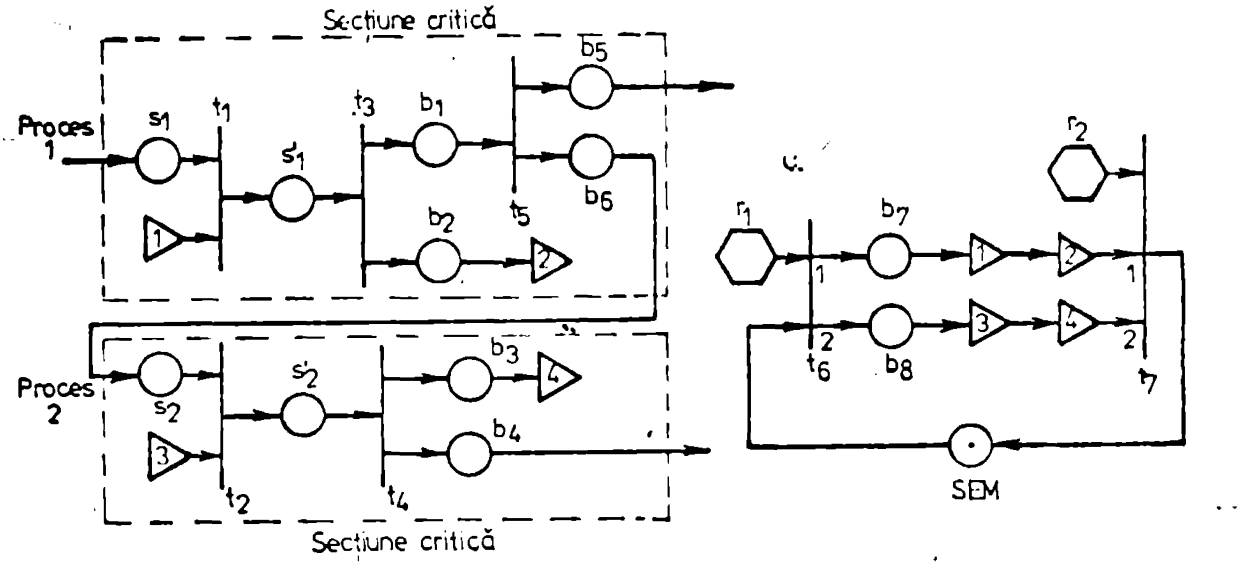


Fig 3.3.3-1

In fig. 3.3.3-1 se reprezintă cu ajutorul RE procesul de sincronizare descris.

Fie acum $\{S_1[X_1, X_2, t_1] \| S_2[X_1, X_2, t_2]\}$ două procese concurente. In fig. 3.3.3-2 se reprezintă amplasarea variabilelor sistem X_1, X_2 şi a secţiunilor critice asociate.

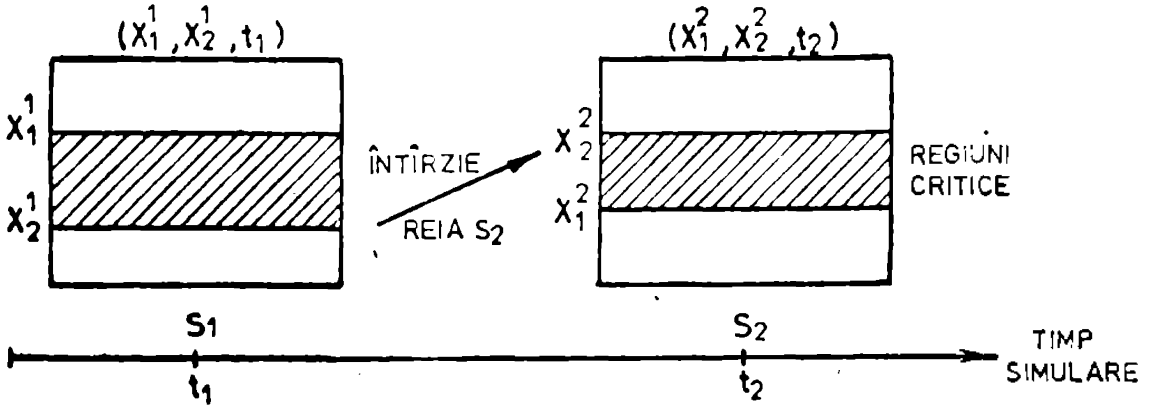


Fig. 3.3.3-2

Dacă $t_1 \leq t_2$ din (P2) şi fig. 3.3.3-2

atunci $X_1^1 \rightarrow X_2^1$

şi $X_2^2 \rightarrow X_1^2$

Din fig333-2 rezultă următoarele relații:

$$X_1^1 \rightarrow X_1^2$$

$$X_2^1 \rightarrow X_2^2$$

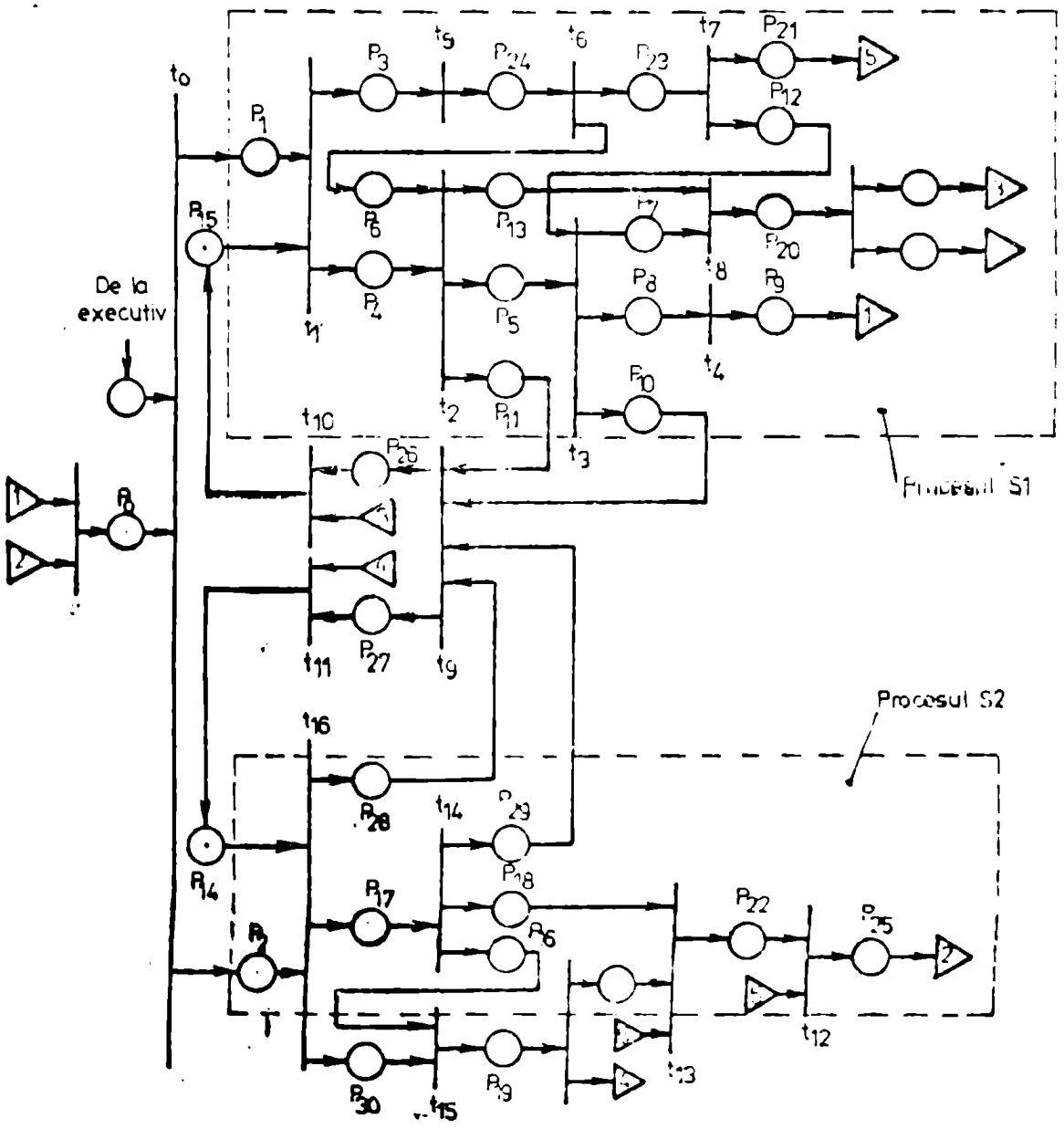
Din relațiile anterioare rezultă

$$X_1^1 \rightarrow X_2^1 \rightarrow X_2^2 \rightarrow X_1^2$$

Această relație presupune că S_2 este întîrziat de executiv la X_2 în așteptarea procesului S_1 care va executa X_1 și X_2. În momentul în care S_1 iese din regiunea sa critică executivul eliberează procesorul pe care este rezident S_2 care trece la executarea propriei regiuni critice.

În figură (3.3.3-3) se reprezintă cu ajutorul RE procesul de sincronizare descris anterior.

Cazul general a n procese concurente se tratează într-o manieră analoagă.



4. MODELAREA ARHITECTURILOR MA PROPUS (pxmxb)

4.1. IPOTEZELE DE MODELARE

i) Modulele procesor includ o UC (μP) şi o memorie proprie conectate printr-o magistrală locală (MGL).

ii) SMM mai dispune şi de alte memorii care nu sînt proprii unui procesor anume, memorii comune MC.

iii) Procesoarele şi MC sînt conectate printr-un set de magistrale globale (MGG).

v) Din punct de vedere fizic, raportat la un procesor se pot deosebi module de memorie locale şi externe. Un procesor poate conlucra cu MP disputînd MGL dar accesul la memoriile externe se face disputînd MGL şi una din MGG. (In anumite cazuri există module de memorie care nu sînt accesibile de la un anumo μP).

iv) Din punct de vedere logic, raportat la un procesor, se deosebesc memorii proprii (MP) şi memorii comune. Memoriile proprii sînt accesibile numai procesoarelor cărora le sînt locale. Memoriile comune sînt accesibile tuturor procesoarelor. Ele pot fi implementate fie utilizînd module de memorie externe tuturor procesoarelor fie utilizînd părţile din memoria locală care nu sînt proprii unui procesor.

vi) Procesoarele execută un şir continuu de instrucţii depuse în memoriile lor proprii. Aceste instrucţii sînt grupate din punct de vedere logic în taskuri sau procese care cooperează trecîndu-şi informaţii prin intermediul memoriilor comune.

vii) Datorită partajării magistralelor globale cît şi a memoriilor comune este iminent că vor apărea conflicte, provocînd trecerea unor procesoare în cozi de aşteptare pentru o resursă ocupată. Din acest motiv se impune un arbitru care să rezolve situaţiile conflictuale impunînd o anumită strategie în utilizarea resurselor.

Modelarea arhitecturii la nivelul schimburilor de informaţie presupune identificarea perioadelor de prelucrare neîntreruptă a unităţilor centrale care necesită accesul la memoriile proprii şi a perioadelor de transfer care necesită accesul la memoria comună pentru schimbul de mesaje. Perioadele de prelucrare şi perioadele de transfer corespund fiecare executării unui grup oarecare de instrucţii elementare. Execuţia proceselor care cooperează prin mesaje presupune deci execuţia repetată a ciclului: "perioadă de prelucrare - perioadă de transfer".

Se presupune de asemenea că perioadele de inactivitate datorate sincronizării proceselor sînt neglijabile, altfel spus: numărul de procese alocate procesoarelor este foarte mare faţă de numărul procesoarelor din sistem.

Stările unui procesor pot fi clasificate astfel :

4: ACTIV: procesorul execută instrucţii din memoria proprie;

3: LOCAL: procesorul lucrează cu o memorie locală;

2: TRANSFER: procesorul schimbă informaţii cu alte procesoare prin intermediul memoriilor comune;

1: ASTEPTARE: procesorul stă într-o coadă de aşteptare pentru un modul de memorie comună;

0: BLOCAT: procesorul este blocat de un alt procesor care lucrează segmentul de memorie comună a memoriei locale, utilizînd magistrala locală.

Parametrii modelului :

- durata medie a unei perioade de prelucrare $1/\lambda$;
- durata medie a unei perioade de transfer $1/\mu$;
- pentru ca mesajul produs de un proces să fie util el trebuie să fie gata într-un timp (variabilă aleatoare) de medie $1/\lambda_t$;
- timpul mediu de prelucrare, dintre două mesaje consecutive, ale unui procesor $1/\lambda_p$;
- cantitatea de lucru sau încărcarea modelului $\rho = \lambda/\mu$.
- cantitatea de lucru sau încărcarea generată de un procesor $\rho_p = \lambda_p/\mu$. (4.1-1)
- cantitatea de lucru sau încărcarea generată de un proces (task) $\rho_t = \lambda_t/\mu$.

Intre parametrii λ_t şi λ_p există o relaţie simplă dacă se presupune o încărcare globală de n procese, n fiind un multiplu întreg al numărului de procesoare p şi considerînd că fiecărui procesor îi revin exact n/p procese. Numărul de procese externe unui procesor este n-n/p.

Presupunînd o distribuţie uniformă a schimbului de mesaje probabilitatea ca procesul i să trimită un mesaj la procesul j este 1/(n-1), $i \neq j$

In acest caz

$$\lambda_p = \lambda_t (n-n/p) \frac{1}{n-1} = \lambda_t \frac{n(p-1)}{p(n-1)} \qquad (4.1-2)$$

Dacă n este foarte mare

$$\lambda_p \approx \lambda_t \qquad (4.1-3)$$

Parametrii λ_p şi λ_i sînt ambii utili în comparaţii: . λ_p determină comportarea diverselor modele odată ce numărul procesoarelor este fixat. λ_i este necesar pentru a compara performanţele fiecărei arhitecturi cînd un număr diferit de procesoare e utilizat pentru executarea uneia şi aceleiaşi cantităţi de proce

Pentru ca procesele stohastice care guvernează comportarea modelelor să satisfacă proprietatea lui Markov se mai impun următoarele ipoteze:

I. Activitatea de bază a procesoarelor se reduce la lucrul cu memoria proprie.

II. Cvasiperiodic procesoarele schimbă mesaje.

III. Durata perioadelor de transfer este o variabilă aleatoare independentă, cu distribuţie exponenţială cu media $1/\mu_j$ pentru modulul j de memorie.

IV. Cînd un procesor solicită accesul la un modul de memorie com între procesor şi memorie se stabileşte instantaneu o cale de legătură (cu condiţia ca o magistrală să fie disponibilă şi memoria liberă).

V. Dacă calea nu poate fi stabilită procesorul stă în aşteptare (în cazul procesoarelor multitask afirmaţia poate fi contrazisă, procesorul trecînd la execuţia unui alt proces, în aşteptarea eliberării resursei).

VI. La încheierea transferului memoria şi magistrala sînt eliberate instantaneu şi procesorul îşi reia activitatea de bază (stare activă). Intervalul dintre două perioade de transfer este o variabilă aleatoare independentă, cu distribuţie exponenţială de medie $1/\lambda_i$ pentru procesorul i.

VII. Un transfer de la procesorul i este îndreptat spre memoria j cu probabilitatea p_{ij}. Frecvenţa de acces de la procesorul i la memoria j se defineşte astfel: $\lambda_{ij} = \lambda_i p_{ij}$. Această ipoteză garantează existenţa lanţului Markov construit. Modelele astfel construite au un număr de stări care creşte exponenţial cu creşterea numărului de componente de SLM. In scopul reducerii numărului stărilor se introduc următoarele ipoteze suplimentare.

VIII. Se presupune că toate procesoarele au aceeaşi frecvenţă de acces la memorie λ şi că toate memoriile sînt identice, astfel încît timpul mediu de transfer este acelaşi pentru toate memoriile şi toate procesoarele: $1/\mu$.

IX. Se presupune că transferul de la orice procesor este îndreptat către oricare memorie cu aceeaşi probabilitate $1/m$, m fiind numărul de module de memorie comună.

X. Cînd o magistrală e liberă, următorul procesor este ales la întîmplare dintre capii de şiruri de aşteptare la memoriile comune.

Analitic:

$$p_{ij} = \frac{1}{m} \quad \forall i,j$$
$$\lambda_1 = \lambda_2 = \cdots = \lambda_p = \lambda$$
$$\mu_1 = \mu_2 = \cdots = \mu_m = \mu \qquad (4.1\text{-}4)$$
$$\lambda_{ij} = \frac{\lambda}{m} \quad \forall i,j$$

Ipotezele enunţate neglijează timpii necesari pentru arbitrarea şi eliberarea resurselor. Ei pot fi însă, dacă e necesar, să fie incluşi în perioada de acces $1/\mu$.

Alegerea distribuţiilor exponenţiale şi a încărcărilor simetrice s-a făcut în ideea facilitării analizei sistemului (nefiind impusă de menţinerea proprietăţilor Markov ale modelului).

4.2. INDICII DE PERFORMANŢĂ

i) Procentul de timp pe durata căruia un procesor este activ, mediat pe numărul total de procesoare sau eficienţa de prelucrare. Dat fiind ergodicitatea modelului stohastic această mărime poate fi definită ca fiind numărul mediu de procesoare active raportat la numărul total de procesoare. Cum numărul total de procesoare este o constantă dată se poate evalua direct numărul mediu de procesoare active sau pe scurt puterea de prelucrare P.

$$P = M\{\text{numărului de procesoare active}\}. \qquad (4.2\text{-}1)$$

Formulele lui Little permit calculul altor indici de performanţă plecînd de la P:

ii) $\lambda^* = P\lambda$, (4.2-2)

λ^* este frecvenţa cu care ciclează clienţii reţeaua de cozi de aşteptare.

iii) $$D = \frac{p - P}{P\lambda} \qquad (4.2\text{-}3)$$

D este întîrziere medie de deservire a unui client.

iv) $$W = D - 1/\mu = \frac{p - P(1+\beta)}{P\lambda}, \quad \beta = \frac{\lambda}{u}. \qquad (4.2\text{-}4)$$

W este timpul mediu de aşteptare în coadă.

v) $$N_q = W P \lambda = p - P(1+\beta) \qquad (4.2\text{-}5)$$

N_q este numărul mediu de procesoare din coadă.

vi) $N_g = \frac{D - W}{P\lambda} = 13$ (4.2-6)

N_g este numărul mediu de procesoare cu acces la modulele de memorie comună.

vii) $C = W + \frac{1}{\lambda} + \frac{1}{\mu} = \frac{P}{P\lambda}$ (4.2-7)

C este timpul mediu de ciclare.

Cu ajutorul acestor indici se pot construi de la caz la caz şi alte măsuri pentru evaluarea performanţelor sistemelor.

4.3.A.1. ARHITECTURA 1 : (p x 1 x 1.)

i) Există o singură memorie comună, externă tuturor procesoarelor (prin care se transmit toate mesajele) şi care este accesibilă numai prin intermediul magistralei globale;

ii) Conflictele apar la utilizarea memoriei comune, căci un singur procesor o poate utiliza la un moment dat;

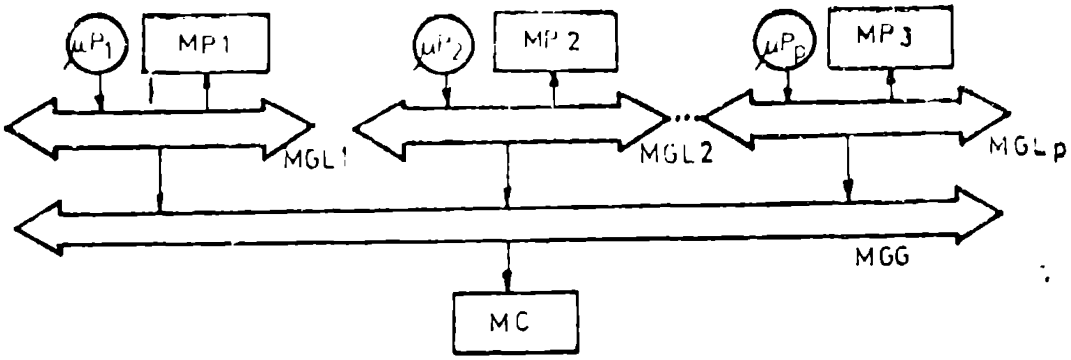


Fig. 4.3.1-1

iii) Comportarea sistemului la nivelul schimbului de informaţie prezintă următoarele caracteristici:

- procesorul execută perioadele de prelucrare după care solicită accesul la MGG;
- după solicitarea MGG procesorul trebuie să aştepte până când ea devine liberă;
- când MGG este alocată unui procesor acesta începe imediat transferul informaţiei;
- fiecare mesaj emis de un procesor este citit de alt procesor; simetria modelului implică (în medie) că numărul de mesaje emise este egal cu cel de mesaje recepţionate de acelaşi procesor. In acest caz activitatea unui procesor este întreruptă cu o frecvenţă λ care este dublul frecvenţei de generare a mesajelor λ_p:

$$\lambda = 2\lambda_p \qquad (4.3.1-1)$$

iv) Disciplina de servire din coadă este: primul intrat primul servit (FIFO).

4.3.A 1.1. MODEL CU SA

In ipotezele date sîntem în cazul unui model M/M/1 cu şir de aşteptare [MIH 2] cu număr finit de clienţi : p.

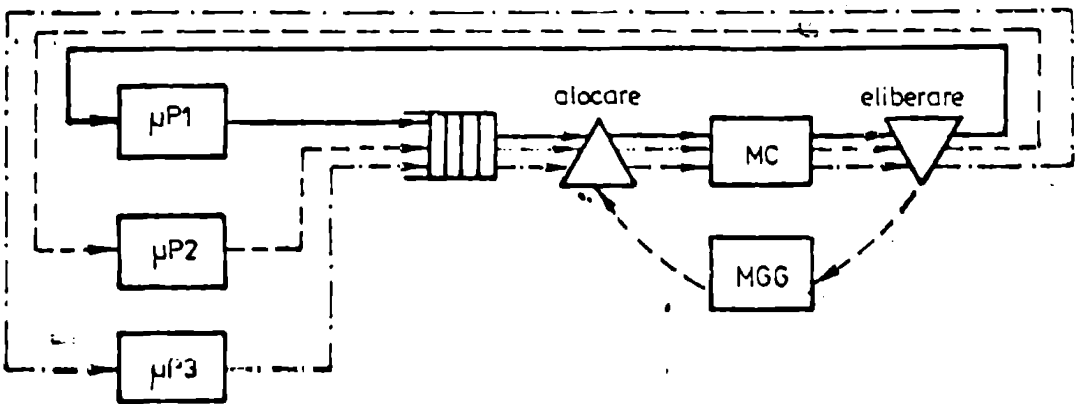


Fig. 4.3.1.-2

Se notează cu $P_j = P\{X(t) = j\}$, $t \geq o$ unde $X(t)$ reprezintă numărul de unităţi în sistem (procesoare în coadă) la momentul $t \geq o$. Analizînd modificările care pot avea loc în sistemul de aşteptare în intervalul $t + \Delta t$ cu $\Delta t > o$ se obţine:

$$\begin{cases} P_o(t+\Delta t) = P_o(t)\left[1-p\lambda\Delta t\right] + P_1(t)\mu\, t + \epsilon(\Delta t) \\ P_j(t+\Delta t) = P_j(t)\left\{1-\left[(p-j)\lambda+\mu\right]\Delta t\right\} + P_{j+1}(t)\mu\Delta t + P_{j-1}(t)\left[p-\right. \\ \qquad \left. -(j-1)\right]\lambda\Delta t + \epsilon(\Delta t) \\ \qquad\qquad\qquad\qquad j = 1,2,\ldots,p-1 \\ P_p(t+\Delta t) = P_p(t)\,(1-\mu\Delta t) + P_{p-1}(t)\lambda\Delta t + \epsilon(\Delta t) \\ \qquad\qquad \text{unde } \epsilon(\Delta t) \to o \text{ cînd } \Delta t \to o, \end{cases} \tag{4.3.1.1-1}$$

$$\begin{cases} \frac{d}{dt}P_o(t) = -\,p\lambda P_o(t) + \mu P_1(t) \\ \frac{d}{dt}P_j(t) = -\left[(p-j)\lambda+\mu\right]P_j(t) + (p-j+1)\lambda P_{j-1}(t) + \mu P_{j+1}(t) \\ \qquad\qquad\qquad\qquad j = 1,2,\ldots,p-1 \\ \frac{d}{dt}P_p(t) = -\,\mu P_p(t) + \lambda P_{p-1}(t) \end{cases} \tag{4.3.1.1-2}$$

In cazul regimului staţionar sistemul devine:

$$\begin{cases} p\lambda P_0 = \mu p_1 \\ \left[(p-j)\lambda + \mu\right]p_j = (p-j+1)\lambda_{j-1} + \mu_{j+1} \\ \qquad\qquad\qquad\qquad j = 1,2,\ldots,p-1 \\ \mu p_p = \lambda p_{p-1} \end{cases} \tag{4.3.1.1-3}$$

unde $p_j = \lim_{t\to\infty} P_j(t)$ $\quad j = 0,1,\ldots,p.$

Din ecuaţiile (4.1-3) se obţine formula de recurenţă

$$p_j = (p-j+1)\rho\, p_{j-1}, \rho = \frac{\lambda}{u} \quad . \qquad (4.3.1.1-4)$$

Soluţia unică a sistemului devine:

$$p_j = \frac{p!\rho^j}{(p-j)!} p_0 \quad j = 0,1,\ldots,p. \qquad (4.3.1.1-5)$$

Ştiind că:

$$\sum_{j=0}^{p} p_j = 1$$

avem

$$p_0 = \left[1 + \sum_{j=1}^{p} \frac{p!\rho^j}{(p-j)!}\right]^{-1} , \qquad (4.3.1.1-6)$$

Caracteristicile principale ale sistemului sînt:

- numărul mediu de procesoare în sistem (aşteptare şi deservire):

$$U = \sum_{j=0}^{p} j p_j = \sum_{j=0}^{p} j \frac{p!\rho^j}{(p-j)!}\left[1+\sum_{j=1}^{p}\frac{p!\rho^j}{(p-j)!}\right]^{-1} = \ldots =$$

$$= p - \frac{1}{\rho}(1-p_0) . \qquad (4.3.1.1-7)$$

Puterea de prelucrare va fi deci :

$$P = p - U = \frac{1}{\rho}(1-p_0) = \frac{1}{\rho}\left\{1-\left[1+\sum_{j=1}^{p}\frac{p!\rho^j}{(p-j)!}\right]^{-1}\right.$$

$$= \frac{1}{\rho}\,\frac{\sum_{j=1}^{p}\frac{p!\rho^j}{(p-j)!}}{1+\sum_{j=1}^{p}\frac{p!\rho^j}{(p-j)!}}$$

$$= \frac{1}{\rho}\,\frac{\sum_{j=0}^{p}\frac{p!\rho^j}{(p-j)!}-1}{\sum_{j=0}^{p}\frac{p!\rho^j}{(p-j)!}} \qquad (4.3.1.1-8)$$

Din formula anterioară se poate deduce formula recursivă

$$P(p) = \frac{p}{1+\varrho[p-P(p-1)]} \qquad (4.3.1.1-9)$$

unde

$$\varrho = 2\varrho_p = 2\varrho_t \frac{p-1}{p} .$$

4.3. 1.2. MODELUL CU RE.

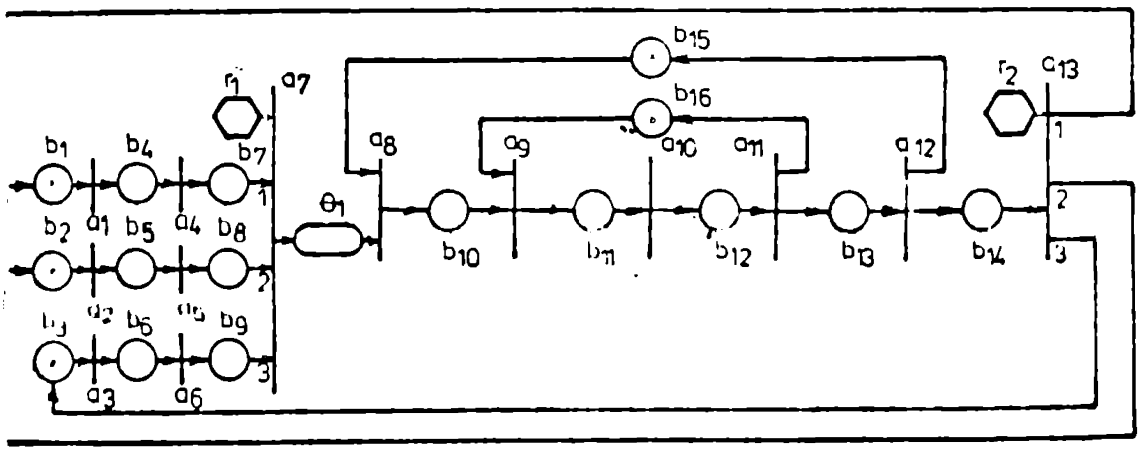


Fig.4.3.1-3

Tranzițiile din fig. 4.3.1-3 au următoarele semnificații:

a1, a2, a3 - generarea timpului cît procesorul 1,2 respectiv 3 e activ și a timpului cît lucrează cu memorie;

a4, a5, a6 - prelucrare (procesorul 1,2, respectiv 3 activ);

a7 - cerere de lucru cu memoria comună și introducerea cererii în șirul de așteptare la magistrală;

a8 - alocarea magistralei;

a9 - alocarea memoriei comune;

a10 - lucrul cu memoria;

a11 - eliberarea memoriei;

a12 - eliberarea magistralei;

a13 - readucerea procesoarelor în starea activ.

Descrierea formală a modelului este:

$$L = \{b_1[6], \ldots, b_{11}[6], b_{12}, b_{13}\}$$

$$P = \{r_1, r_2\}$$

$$R = \{r_1, r_2\}$$

$$A = \{a_1, \ldots, a_{13}\}$$

$\Sigma = \{t, \lambda, \mu\}$

$M_o(b_1) = 1$; $M_o(b_2) = 1$; $M_o(b_3) = 1$

$M_o(b_{12}) = 1$; $M_o(b_{13}) = 1$

unde t reprezintă timpul simulării.

Procedurile de tranziţie sînt:

$a_1 = (T(b_1, b_4), 0, [T \rightarrow M(b_4(1)) := 1 ;$
$M(b_4(2)) := DEXP(\lambda) ; M(b_4(4)) := DEXP(\mu)])$

$a_2 = (T(b_2, b_5), 0, [T \rightarrow M(b_5(1)) := 2;$
$M(b_5(2)) := DEXP(\lambda); M(b_5(4)) := DEXP(\mu)])$

$a_3 = (T(b_3, b_6), 0, [T \rightarrow M(b_6(1)) := 3;$
$M(b_6(2)) := DEXP(\lambda); M(b_6(4)) := DEXP(\mu)])$

$a_4 = (T(b_4, b_7), M(b_4(2)), -)$

$a_5 = (T(b_5, b_8), M(b_5(2)), -)$

$a_6 = (T(b_6, b_9), M(b_6(2)), -)$

$a_7 = (Y_3(r_1, b_7, b_8, b_9, Q_1), (0,0,0), -)$

$a_8 = (J(Q_1, b_{15}, b_{10}), 0, -)$

$a_9 = (J(b_{10}, b_{16}, b_{11}), 0, -)$

$a_{10} = (T(b_{11}, b_{12}), M(b_{11}(4)), -)$

$a_{11} = (F(b_{12}, b_{13}, b_{16}), 0, [T \rightarrow M(b_{13}(5)) := M(b_{12}(5)) +$
$+ M(b_{12}(2))])$

$a_{12} = (F(b_{13}, b_{14}, b_{15}), 0, [T \rightarrow M(b_{14}(6)) := M(b_{13}(5))/t])$

$a_{13} = (X_3(r_2, b_{14}, b_1, b_2, b_3), (0,0,0), -)$

Procedurile de rezoluţie sînt:

$r_1 : [M(b_7) = 1 \rightarrow M(r_1) := 1; M(b_8) = 1 \rightarrow M(r_1) := 2; T \rightarrow M(r_1) := 3]$

$r_2 : [M(b_{14}(1)) = 1 \rightarrow M(r_2) := 1; M(b_{14}(1)) = 2 \rightarrow M(r_2) := 2;$
$T \rightarrow M(r_2) := 3]$

unde DEXP semnifică generarea de numere aleatoare după o distribuţie exponenţială de parametru specificat.

4.3.1.3. SCHEMA LOGICA A PROGRAMULUI GPSS

In această schemă logică facilitatea 1 reprezintă MGC,

facilitatea 2 reprezintă MC, iar şirul de aşteptare 1 se formează la ocuparea MCG.

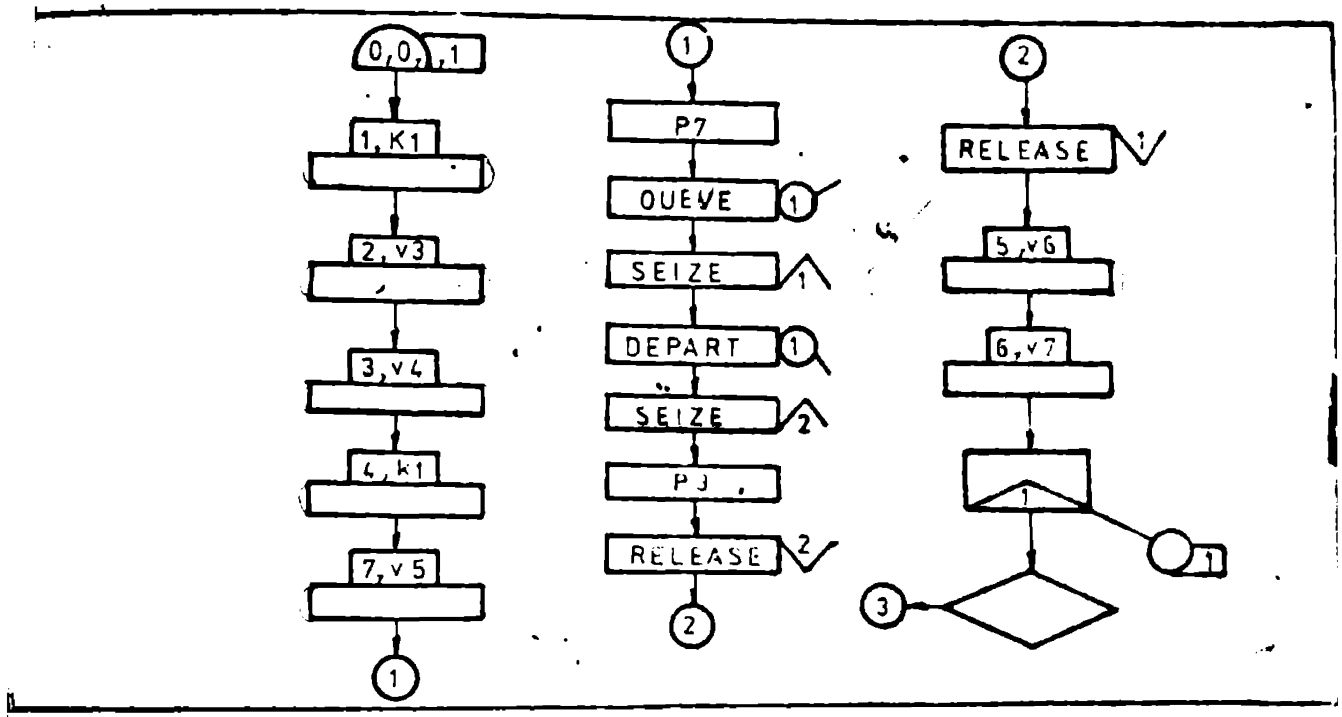


Fig. 4.3.1-4

4.3.A.2. ARHITECTURA 2: (pxpxl)

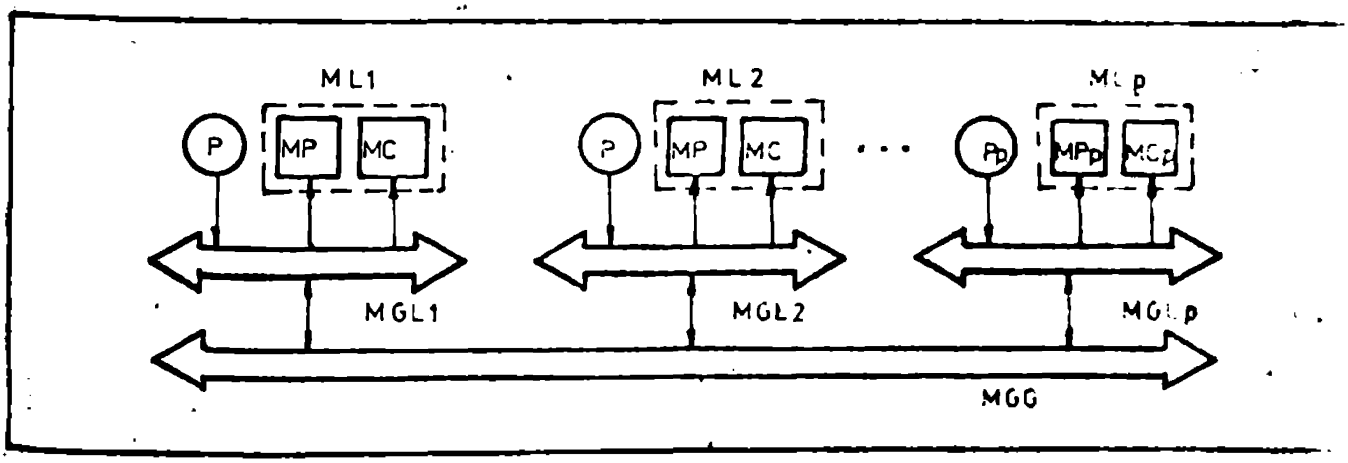


Fig. 4.3.2-1

i) Memoriile locale sînt divizate din punct de vedere logic în memorii proprii şi memorii comune;

ii) Fiecare procesor este conectat cu memoria sa locală printr-o magistrală locală;

iii) Un procesor utilizează pentru accesul la o memorie comună externă: magistrala locală proprie, magistrala globală şi

magistrala locală a procesorului destinație.

iv) Conflictele pot apărea pentru accesul la oricare din magistrale. Strategia aleasă pentru a evita blocările este aceea că orice procesor care a cîștigat controlul MGG poate întrerupe orice procesor și are prioritate la utilizarea oricărei resurse. Procesoarele active întrerupte trec în starea blocată; cele din coadă nu își modifică starea dar eliberează magistrala locală.

v) Cozile sînt deservite după disciplina FIFO;

vi) In cazul unui transfer de la procesorul i către procesorul j (i ≠ j); procesorul i solicită MGG; dacă ea devine disponibilă atunci el preia și controlul MGL_j și transferul are loc. Preluarea mesajului de către procesorul j în MP_j are loc utilizînd numai MGL_j dar această activitate nu poate fi privită ca făcînd parte dintr-o perioadă de prelucrare.

Simetria modelului permite utilizarea ipotezei că numărul de mesaje primit și cel trimis de un procesor sînt în medie egale, adică

$$\frac{1}{\lambda} = \frac{1}{\lambda_p} + \frac{1}{\mu} \quad \text{sau} \quad \lambda = \frac{\lambda_p \cdot u}{\lambda_p + u} \qquad (4.3.2\text{-}1)$$

Utilizarea teoriei șirurilor de așteptare în acest caz este foarte dificilă datorită problemelor ridicate de modelarea fenomenelor de blocare datorate procesoarelor care utilizează memoria locală a altor procesoare.

4.3.2.1. MODELAREA CU LANȚ MARKOV

Starea sistemului este definită printr-o mulțime ordonată de p matrici patrate de ordinul 2

$$\left(\begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix}, \ldots, \begin{bmatrix} a_p & b_p \\ c_p & d_p \end{bmatrix}\right) \qquad (4.3.2.1\text{-}1)$$

în care a_i reprezintă numărul de ordine al procesorului;

b_i reprezintă starea procesorului;

c_i reprezintă numărul de ordine al memoriei adresate de procesorul a_i;

d_i reprezintă poziția procesorului în coadă.

Starea b_i poate lua următoarele valori:

4. activ
3. adresarea unei memorii locale
2. adresarea unei memorii externe
1. în coadă
0. blocat

Această definiţie a stării conduce la un număr foarte mare de stări pentru cazul (p x p x 1):

$$1 + p\,(C^{0}_{p-2} + C^{1}_{p-2} + \ldots + C^{p-2}_{p-2}) \qquad (4.3.2.1\text{-}2)$$

unde 1 : corespunde stării în care toate procesoarele sînt în starea 3 sau 4,

p : este numărul de stări în care este ocupată :C

$C^{0}_{p-2} + C^{1}_{p-2} + \ldots + C^{p-2}_{p-2}$ corespunde numărului de stări cu procesoare în coadă.

Conform teoriei lanţurilor Markov comasate se pot obţine lanţuri cu un număr mult mai mic de stări dar care furnizează o descriere mult mai puţin detaliată.

Definiţia stării în lanţul comasat este:

$$(n_{34},\ n_{12},\ n_{o}) \qquad (4.3.2.1\text{-}3)$$

unde:

n_{34} este numărul procesoarelor care lucrează cu magistrala locală;

n_{12} numărul procesoarelor care adresează o memorie externă sau stau în coada de aşteptare;

n_{o} numărul procesoarelor blocate.

In acest caz numărul stărilor este 2p-1. Diagrama de tranziţie a stărilor pentru lanţul comasat este prezentată în fig.: 4.3.2-2.

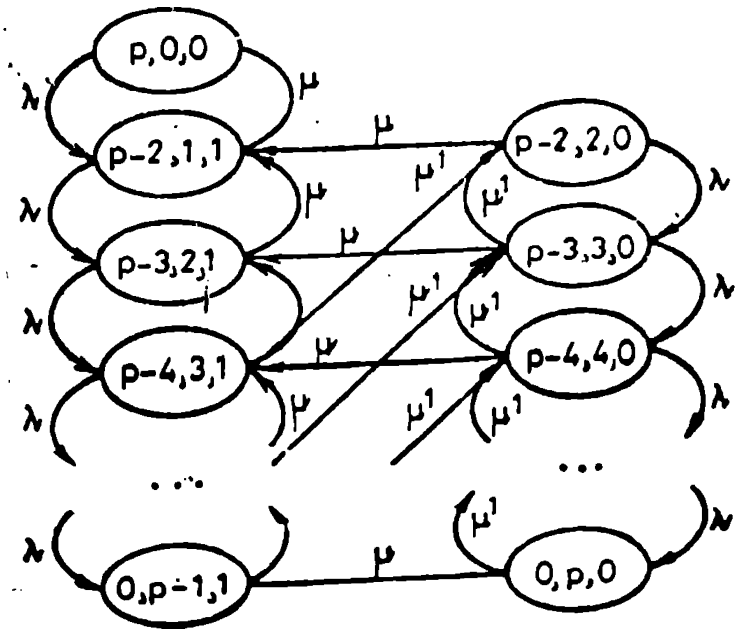


Fig.4.3.2-2

Tranziţiile λ_c corespund generării unei cereri de acces de la un procesor activ. Tranziţiile μ_c corespund încheierii unei perioade de transfer.

Procesoarele care solicită utilizarea MGG cînd aceasta este ocupată se aşează într-o coadă de aşteptare. La eliberarea MGG un alt procesor trece la utilizarea ei. Incepe un nou transfer, şi procesorul destinaţie se blochează cu condiţia să nu fi fost în coada de aşteptare. Depinzînd de starea procesorului destinaţie, sistemul trece într-o stare cu un procesor blocat (tranziţii μ_c) sau fără procesoare blocate (tranziţii μ_c').

Frecvenţa cererilor de acces la MGG este proporţională cu numărul procesoarelor active n_{34}. Frecvenţa de încheiere a transferurilor este μ, dacă $0<n_{12}\leq 2$ căci în momentul în care se încheie un transfer în coadă a mai rămas cel mult un procesor (care va bloca un procesor activ cînd îşi începe transferul). Dacă $n_{12}>2$ apar două posibilităţi:

- tranziţiile μ_c' : la sfîrşitul perioadei de transfer un procesor preia MGG şi ocupă MGL a unui procesor din cele $n_{12}-2$ procesoare rămase în coadă cu probabilitatea $(n_{12}-2)/(p-1)$

- tranziţiile μ_c : cînd după preluarea MGG se ocupă MGL a unui procesor activ cu probabilitatea $1-(n_{12}-2)/(p-1)$.

In rezumat:

$$\lambda_c = n_{34} \qquad (4.3.2.1\text{-}4)$$

$$\mu_c = \begin{cases} \mu & 0<n_{12}\leq 2 \\ \left(1-\dfrac{n_{12}-2}{p-1}\right)\mu & n_{12}>2 \end{cases} \qquad (4.3.2.1\text{-}5)$$

$$\mu_c' = \frac{n_{12}-2}{p-1}\mu \qquad n_{12}>2 \qquad (4.3.2.1\text{-}6)$$

$$\text{unde} \begin{cases} u_c + u_c' = u \\ n_0 + n_{12} + n_{34} = p \end{cases}$$

Probabilităţile de echilibru a stărilor se deduc revolvînd sistemul de ecuaţii liniare (4.3.2.1-3) corespunzător.

Dacă S este spaţiul stărilor lanţului Markov [...], s o stare iar $\pi(s)$ este probabilitatea de echilibru atunci puterea de prelucrare este dată de

$$P = (1-\varepsilon)\sum_{s\in S} n_{34}(s)\pi(s) \qquad (4.3.2.1\text{-}7)$$

Factorul $1-\varrho$ ține cont de timpul de citire a mesajelor, inclus în perioadele de prelucrare:

$$\frac{\frac{1}{\lambda} - \frac{1}{\mu}}{\frac{1}{\lambda}} = 1-\varrho \qquad (4.3.2.1-8)$$

Cazul $p = 2$

$S_0 = (2,0,0); S_1 = (0,1,1); S_2 = (0,2,0)$

$p_{01} = 2$; $p_{10} = \mu$; $p_{21} = \mu$ deci :

$$\underline{P} = \begin{bmatrix} 1-2 & 2 & 0 \\ u & 1-u & 0 \\ 0 & u & 1-u \end{bmatrix}$$

și deci :

$$[\pi_0 \ \pi_1 \ \pi_2] \ \underline{P} = [\pi_0 \ \pi_1 \ \pi_2] \qquad (4.3.2.1-9)$$

de unde

$$\begin{cases} \pi_0 (1-2\lambda) + \pi_1 \mu = \pi_0 \\ \pi_0 2\lambda + \pi_1 (1-\mu) + \pi_2 \mu = \pi_1 \\ \pi_2 (1-\mu) = \pi_2 \end{cases}$$

ținînd cont că $\pi_0 + \pi_1 + \pi_2 = 1$

rezultă $\pi_0 = \frac{1}{2\varrho + 1}$, $\pi_1 = \frac{2\lambda}{\mu} \ \frac{1}{2\varrho+1}$, $\pi_2 = 0$

și deci $P = \frac{2(1-\varrho)}{2\varrho + 1}$ (4.3.2.1-10)

Pentru $n = 3$ puterea de prelucrare devine:

$$P = \frac{3(1-\varrho^2)}{3\varrho^2 + 3\varrho + 1} \qquad (4.3.2.1-11)$$

unde $\varrho = \frac{\varrho_p}{1+\varrho_p} = \frac{\varrho_t}{\varrho_t + \frac{p}{p-1}}$

4.3. .2.2. MODELUL CU M

Tranzițiile din fig. 4.3.2-3 au următoarea semnificație:

a_1, a_2, a_3 - trecerea procesorului 1, 2 respectiv 3 în starea activ;

a_4, a_5, a_6 - generarea timpului cît procesorul 1,2 respectiv 3 va rămîne în starea activ, a timpului cît va fi blocat, generarea cererii de memorie;

a_7, a_8, a_9 - prelucrare și perioadă de blocare în care alt procesor îi transmite un mesaj; generarea timpului cît va lucra cu memoria cerută;

a_{10}, a_{11}, a_{12} - selectarea traseului marcajului în funcţie de memoria cerută, în vederea alocării MGG, dacă este cazul (traseul 1 pentru MC 1, 2 pentru MC 2, respectiv 3 pentru MC 3);

a_{13}, a_{14}, a_{15} - selectarea cererii de memorie cea mai prioritară (se dă prioritate cererii venită de la procesorul care a abţinut MGG);

a_{16}, a_{17}, a_{18} - lucrul cu memoria cerută (MC 1, MC 2 respectiv MC 3); calculul valorii atributului 6;

a_{19}, a_{20}, a_{21} - selectarea traseului marcajului în funcţie de procesorul de la care a pornit cererea de memorie (în vederea eliberării MGG, dacă este cazul);

a_{22} - selectarea cererii de memorie care va putea primi MGG;

a_{23} - alocarea MGG;

a_{24} - selectarea traseului marcajului corespunzător cererii de memorie care a obţinut şi MGG;

a_{25} - selectarea traseului marcajului, corespunzător cererii de memorie, prin care se va elibera MGG;

a_{26} - eliberarea MGG;

a_{27} - selectarea traseului marcajului în funcţie de procesorul care a lucrat cu memoria.

Descrierea formală a modelului este:

$L = \{ b_1[7], \ldots, b_{40}[7], b_{41} \}$

$P = \{ r_1, \ldots, r_{16} \}$

$R = \{ r_1, \ldots, r_{16} \}$

$A = \{ a_1, \ldots, a_{27} \}$

$\xi = \{ t, \lambda, \mu \}$

$M_o(b_1) = 1$; $M_o(b_2) = 1$; $M_o(m_3) = 1$; $M_o(b_{41}) = 1$

In plus faţă de atributele prezentate, a fost introdus parametrul 7 în care se memorează timpul cît procesorul este activ, adunat cu timpul cît el este blocat.

Procedurile de tranziţie sînt:

$a_1 = (Y(r_1, b_{25}, b_{38}, b_1), (o,o), -)$

$a_2 = (Y(r_2, b_{27}, b_{39}, b_2), (o,o), -)$

$a_3 = (Y(r_3, b_{29}, b_{40}, b_3), (o,o), -)$

$a_4 = (T(b_1, b_4), o, [T \rightarrow M(b_4(1)) := 1;\ M(b_4(2)) := DEXP(\lambda);$
$M(b_4(3)) := DEXP(\mu);\ M(b_4(4)) := FM;\ M(b_4(7)) := M(b_4(2)) + M(b_4(3))])$

a_5 = (T(b_2,b_5), o,[T→M (b_5(1)): = 2; M(b_5(2)):=DEXP(λ);
M(b_5(3)): = DEXP (μ);M(b_5(4)):= FM; M(b_5(7): =
M(b_5(2)) + M (b_5(3))])

a_6 = (T(b_3,b_6), o, [T→M(b_6(1)):=3; M(b_6(2)):= DEXP (λ);
M(b_6(3)): = DEXP (μ); M(b_6(4)) : = FM;M(b_6(7)):=
M(b_6(2)) + M(b_6(3))])

a_7 = (T(b_4,b_7),M(b_4(7)), [T→M(b_7(3)):= DEXP (μ)])

a_8 = (T(b_5,b_8),M(b_5(7)), [T→M(b_8(3)):=DEXP (μ)])

a_9 = (T(b_6,b_9),M(b_6(7)),[T→M(b_9(3)):=DEXP(μ)])

a_{10}= (X(r_4,b_7,b_{11},b_{12}),(o,o),-)

a_{11}= (X(r_5,b_8,b_{14},b_{15}),(o,o),-)

a_{12}= (X(r_6,b_9,b_{17},b_{1o}),(o,o),-)

a_{13}= (Y(r_7,b_{1o},b_{11},b_{19}),(o,o), -)

a_{14}= (Y(r_8,b_{13},b_{14},b_{2o}),(o,o), -)

a_{15}= (Y(r_9,b_{16},b_{17},b_{21}),(o,o), -)

a_{16}= (T(b_{19},b_{22}),M(b_{19}(3)),[T→M(b_{22}(5)): =M(b_{19}(5)) +
+ M(b_{19}(2)); M(b_{22}(6)): = M(b_{19}(5))/t])

a_{17}= (T(b_{20},b_{23}),M(b_{24}(3)),[T→M(b_{23}(5)):=M(b_{24}(5)) +
+ M(b_{20}(2)):M(b_{23}(6)):=M(b_{20}(5))/t)])

a_{18}= (T(b_{21},b_{24})?M(b_{21}(3)),[T→M(b_{24}(5)):=M(b_{21}(5)) +
+ M(b_{21}(2)); M(b_{24}(6)):=M(b_{21}(5))/t)])

a_{19}= (X(r_{1o},b_{22},$b_{25}$$b_{26}$),(o,o), -)

a_{2o}= (X(r_{11},b_{23},b_{27},b_{28}),(o,o), -)

a_{21}= X(r_{12},b_{24},b_{29},b_{3o}),(o,o), -)

a_{22}= (Y$_3$(r_{13},b_{12},b_{15},b_{18},b_{31}),(o,o,o), -)

a_{23}= (J (b_{31},b_{41},b_{32}),o, -)

a_{24}= (X$_3$(r_{15},b_{23},b_{33},b_{34},b_{35}),(o,o,o), -)

a_{25}= (Y$_3$(r_{14},b_{26},b_{28},b_{3o},b_{36}),(o,o,o), -)

a_{26}= (F(b_{36},b_{41},b_{37}),o, -)

a_{27}= (X$_3$(r_{16},b_{37},b_{38},b_{39},b_{4o}),(o,o,o), -)

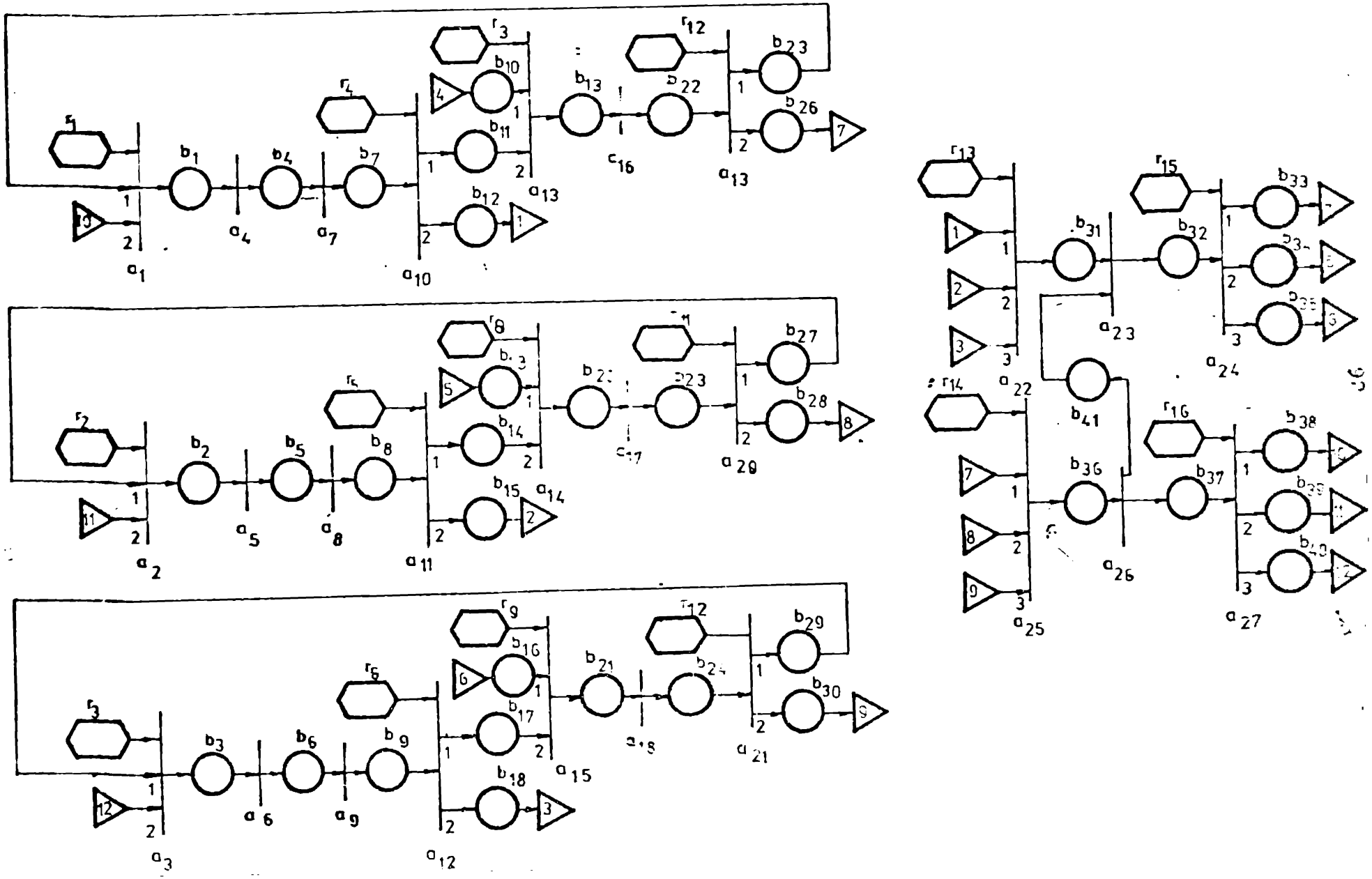


Fig 4.3.2-3

unde $FM(x) = \begin{cases} 1, \text{ dacă } x \in [0, 0.33) \\ 2, \text{ dacă } x \in [0.33, 0.66) \\ 3, \text{ dacă } x \in [0.66, 1.) \end{cases}$

x fiind o variabilă aleatoare care ia valori în intervalul $[0,1]$ conform unei distribuţii normale.

Procedurile de rezoluţie sînt:

r_1: $[T \to M(r_1) := 1]$

r_2: $[T \to M(r_2) := 1]$

r_3: $[T \to M(r_3) := 1]$

r_4: $[(M(b_7(4)) = 1) \to M(r_4) := 1; T \to M(r_4) := 2]$

r_5: $[(M(b_8(4)) = 2) \to M(r_5) := 1; T \to M(r_5) := 2]$

r_6: $[(M(b_9(4)) = 3) \to M(r_6) := 1; T \to M(r_6) := 2]$

r_7: $[(M(b_{10}) = 1) \to M(r_7) := 1; T \to M(r_7) := 2]$

r_8: $[(M(b_{13}) = 1) \to M(r_7) := 1; T \to M(r_7) := 2]$

r_9: $[(M(b_{16}) = 1) \to M(r_9) := 1; T \to M(r_8) := 2]$

r_{10}: $[(M(b_{22}(4)) = 1) \to M(r_{10}) := 1; T \to M(r_{10}) := 2]$

r_{11}: $[(M(b_{23}(4)) = 2) \to M(r_{11}) := 1; T \to M(r_{11}) := 2]$

r_{12}: $[(M(b_{24}(4)) = 3) \to M(r_{12}) := 1; T \to M(r_{12}) := 2]$

r_{13}: $[T \to M(r_{13}) := 1]$

r_{14}: $[T \to M(r_{13}) := 1]$

r_{15}: $[(M(b_{32}(4)) = 1) \to M(r_{15}) := 1; (M(b_{32}(4)) = 2) \to$
$M(r_{15}) := 2; T \to M(r_{15}) := 3]$

r_{16}: $[(M(b_{37}(1)) = 1) \to M(r_{16}) := 1; (M(b_{37}(1)) = 2) \to$
$M(r_{16}) := 2; T \to M(r_{16}) := 3]$

4.3. .2.3. MODELUL CU SA

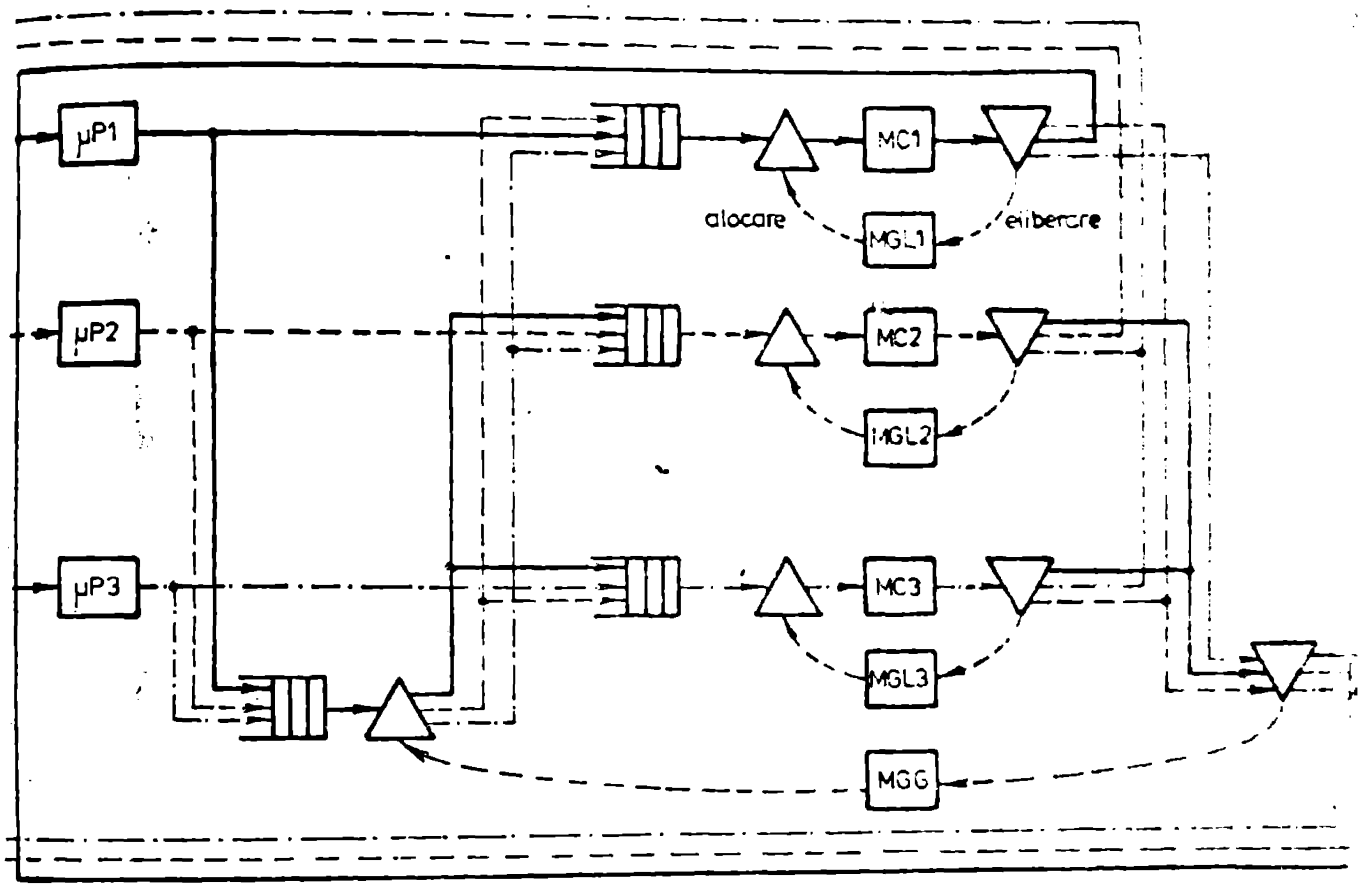


Fig.4.3.2.-4

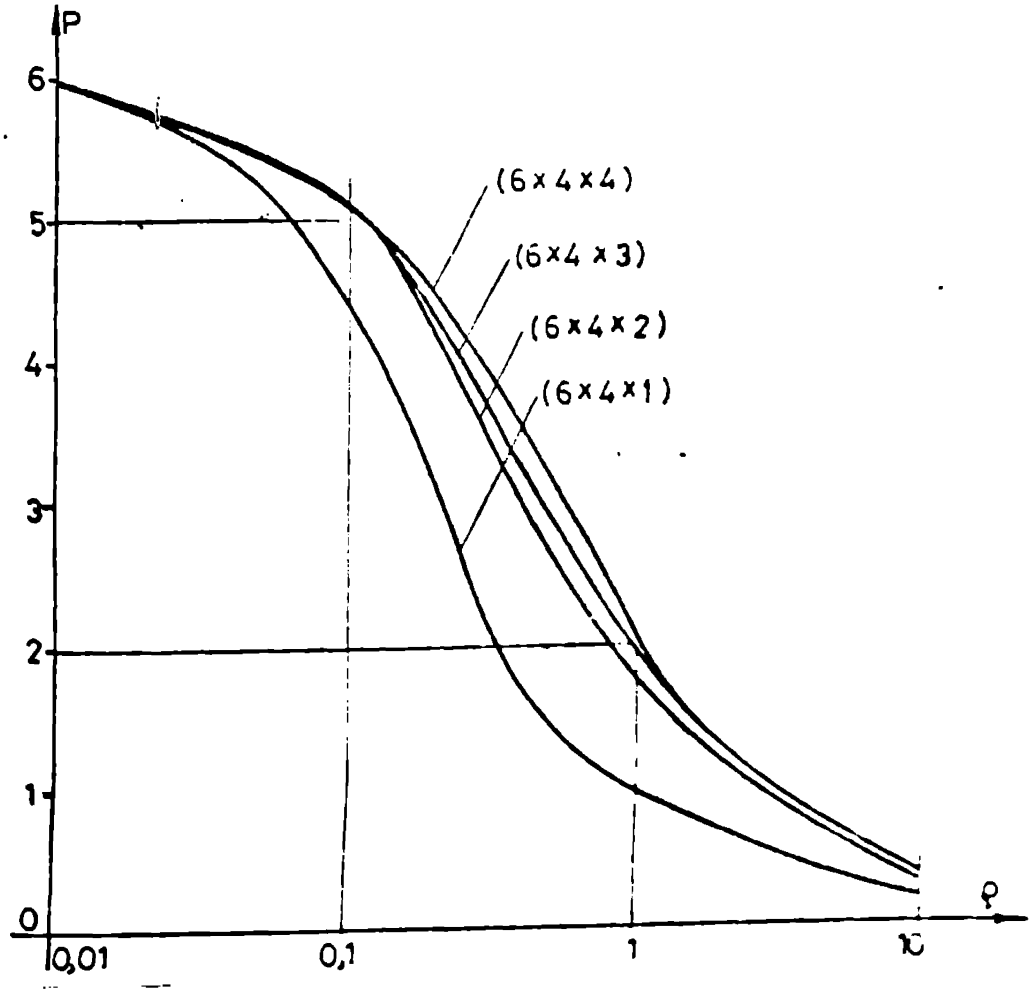


Fig.4.4

4.3.2.4. SCHEMA LOGICĂ A PROGRAMULUI GPSS

In această schemă logică facilitatea 1 reprezintă MC1, facilitatea 2 reprezintă MC2, facilitatea 3 reprezintă MC3, facilitatea 4 reprezintă MCC. Şirul de aşteptare 1 se formează la MCC. Funcţia 2 reprezintă memoria cerută (analoagă funcţiei din modelul cu RE).

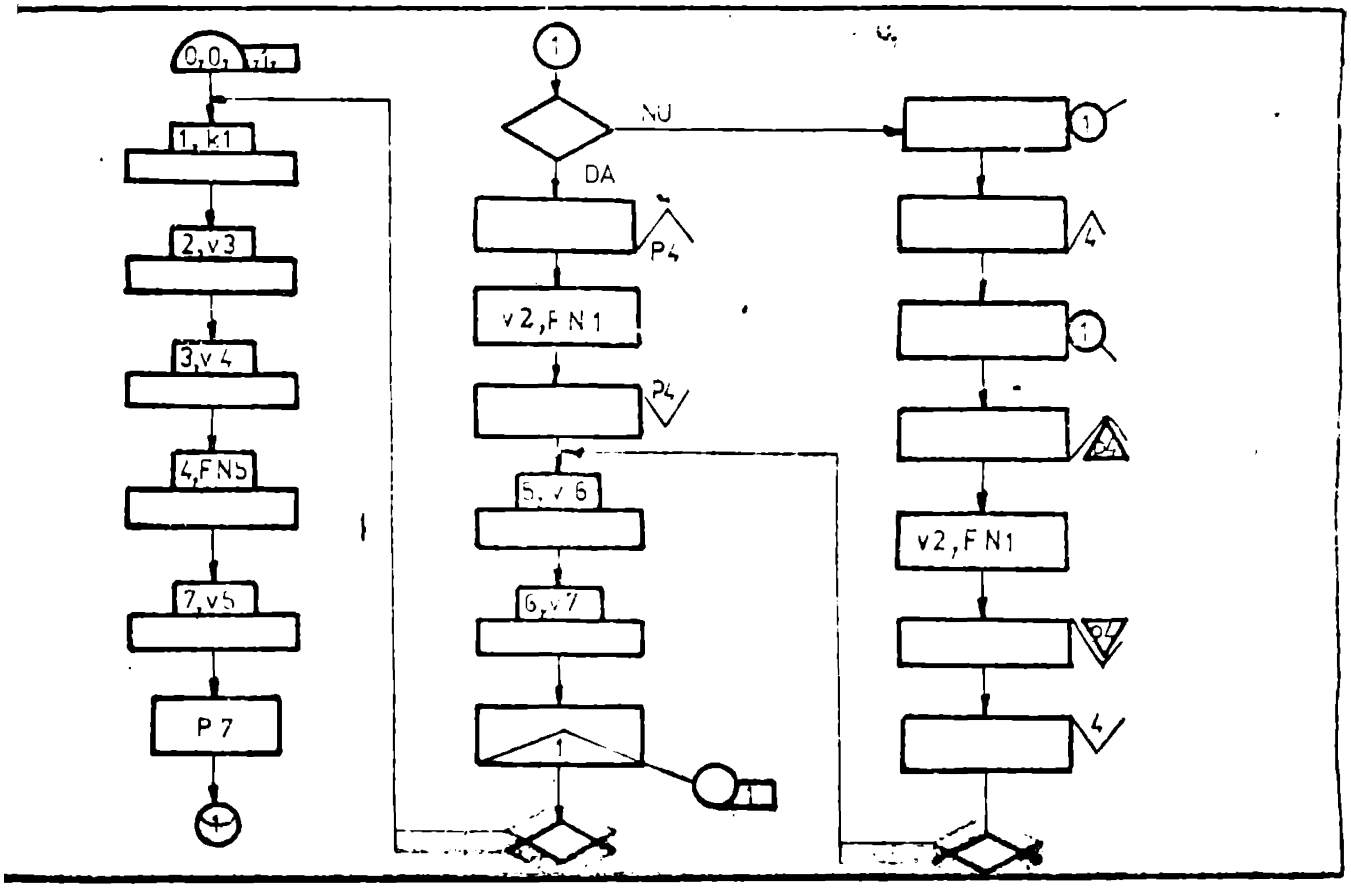


Fig 4.3.2-6

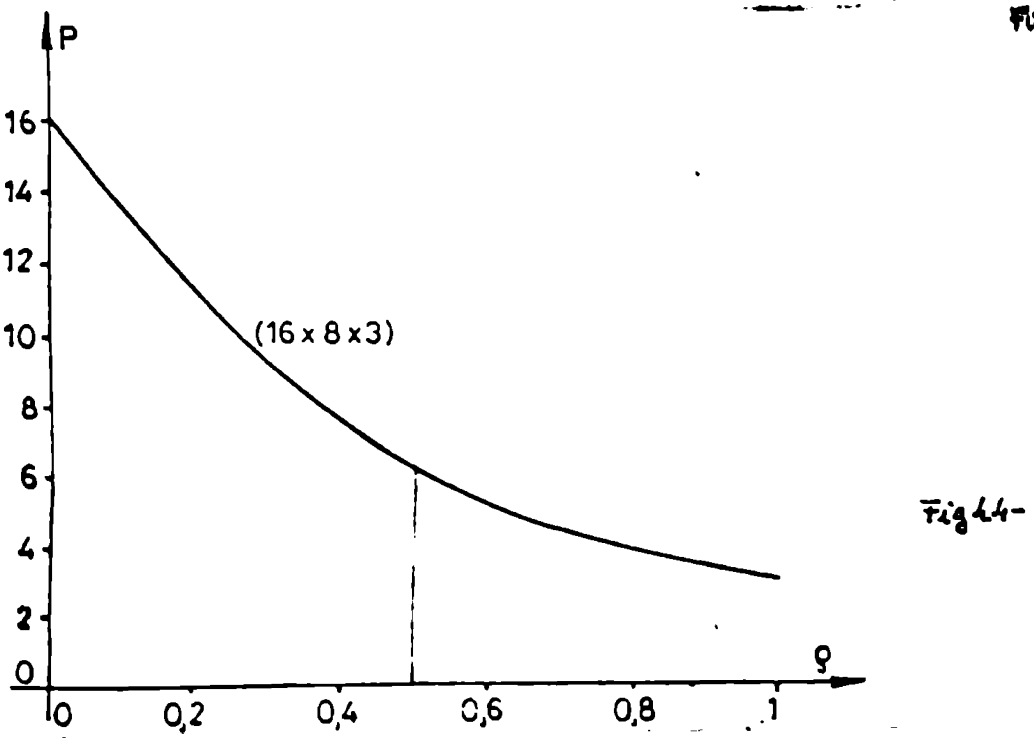


Fig 4.4-

4.3. .3. ARHITECTURA 3 (pxpxl)

Această arhitectură reprezintă o variantă îmbunătăţită a arhitecturii 2.

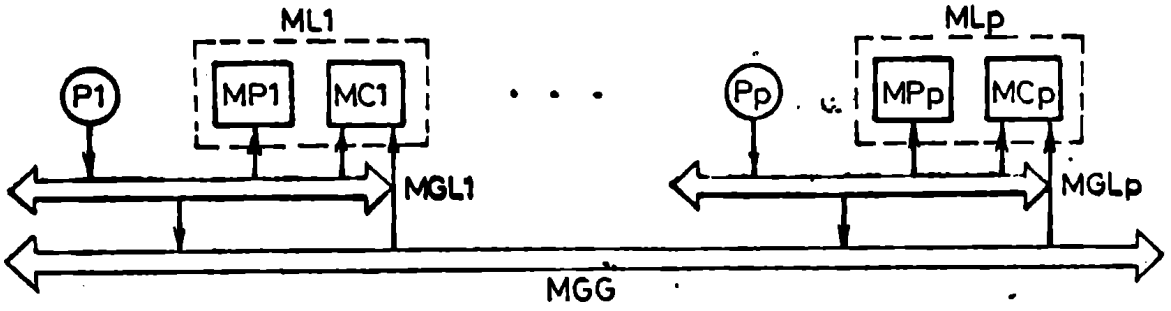


Fig.4.3.3-1

i) Partea de memorie comună a memoriei locale este o memorie cu două registre de acces (Dual Port RAM) MCX2, MC devin astfel accesibile direct prin MGG. Nu mai apar conflicte la MGL, nici la MC care poate fi simultan adresată de doi utilizatori;

ii) Conflictele se datoresc doar partajării MGG

iii) Procesoarele îşi adresează MC locală numai prin intermediul MGL

iv) Transferul mesajelor este similar cu cel descris la arhitectura 2. In cazul transferului unui mesaj prin MCX2, procesorul corespunzător nu mai este blocat.

v) Disciplina de deservire din coadă este FIFO.

vi) Perioada de prelucrare include şi de această dată trecerea mesajului din MCX2 în MP astfel încît

$$\lambda = \frac{\lambda_p \mu}{\lambda_p + \mu} \qquad (4.3.3-1)$$

4.3. .3.1. MODEL CU SA

Deoarece în acest caz există o singură sursă de conflict, arhitectura 3 poate fi simulată printr-un model de aşteptare M/M/1/ /p cu staţie centrală de deservire .

Expresia puterii de prelucrare obţinută în cazul arhitecturii 1 rămîne şi în acest caz valabilă dar ea trebuie corectată

u factorul $(1-\rho)$ care ţine cont de timpul necesar pentru a ransfera un mesaj în cadrul ML, Se obţine

$$P = (1-\rho)\ \frac{1}{\rho}\ \frac{\sum_{j=0}^{p} \frac{p!\,\rho^j}{(p-j)!} - 1}{\sum_{j=0}^{p} \frac{p!\,\rho^j}{(p-j)!}} \qquad (4.3.3.1-1)$$

Formula de recurenţă ia forma :

$$P(p) = \frac{p(1-\rho)}{1+\rho\,(p-\frac{P(p-1)}{1-\rho})} \qquad (4.3.3.1-2)$$

unde:

$$\rho = \frac{\rho_p}{\rho_{p+1}} = \frac{\rho_t}{\rho_t + \frac{p}{p-1}} \qquad (4.3.3.1-3)$$

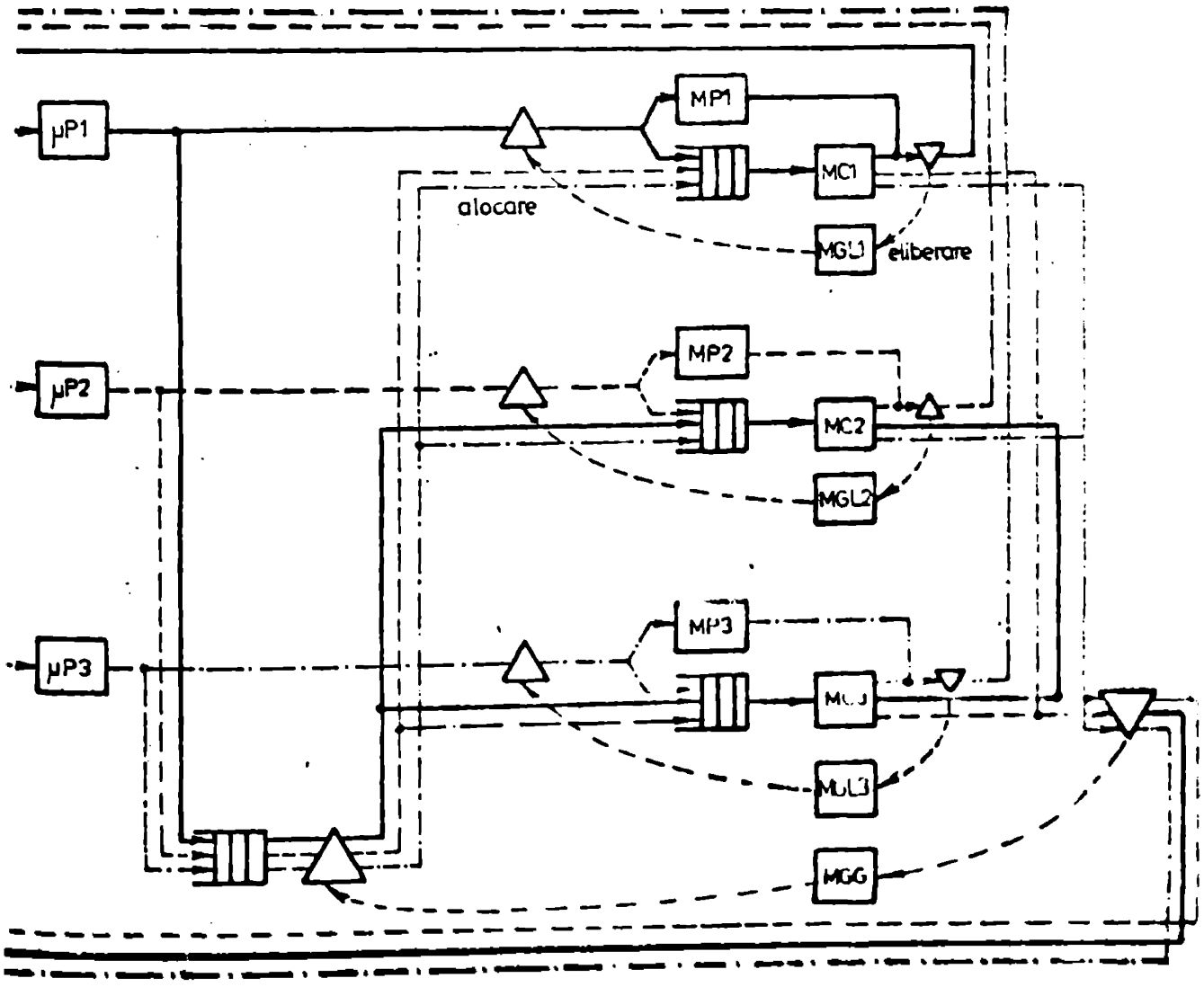


Fig.4.3.3-2

4.3.4.3.2. MODELUL CU RE

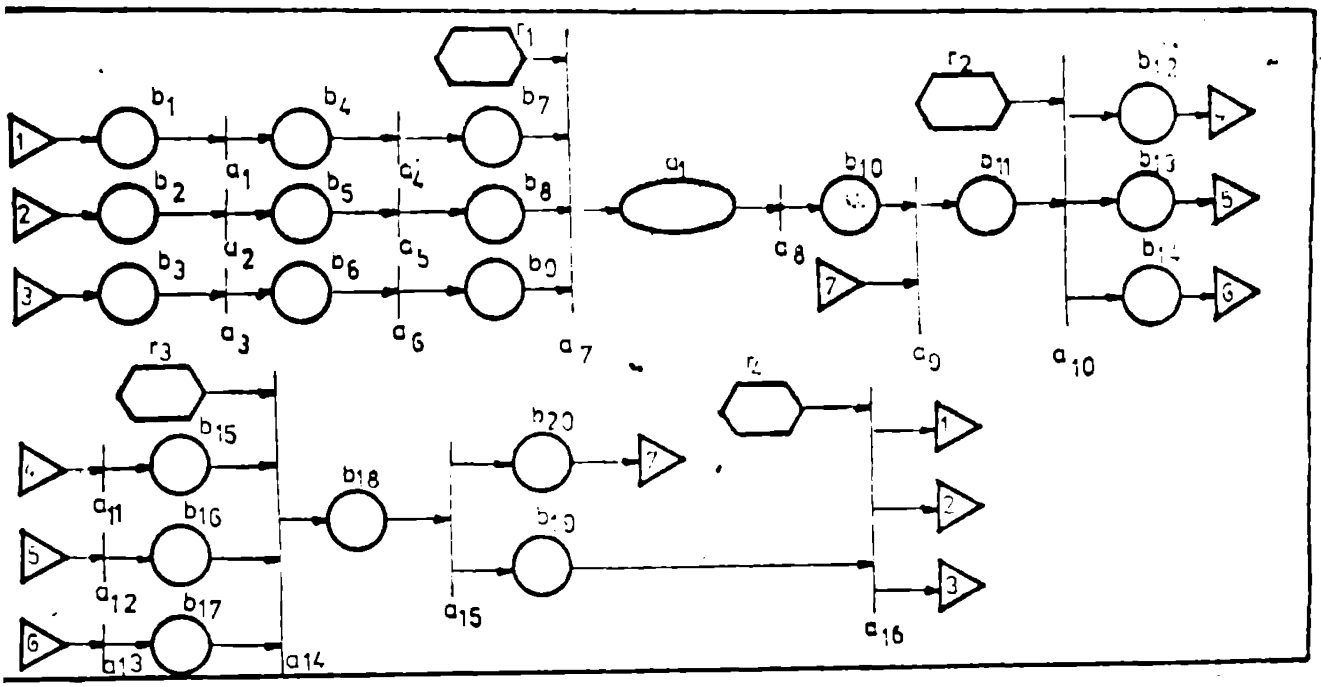


Fig.4.3.3-3

Tranziţiile din fig. 4.3.3.3-3 au următoarea semnificaţie:

a_1, a_2, a_3 - generarea timpului cît procesorul 1,2, respectiv 3 este activ, a timpului cît este blocat şi a memoriei cerute;

a_4, a_5, a_6 - procesorul 1,2, respectiv 3 este în starea activ;

a_7 - ataşarea cererilor de lucru cu o memorie comună la şirul de aşteptare pentru ocuparea MCG;

a_8 - detaşarea unei cereri din şirul de aşteptare; generarea timpului cît va lucra cu memoria;

a_9 - alocarea MCG;

a_{10} - selectarea traseului marcajului în funcţie de memoria cerută;

a_{11}, a_{12}, a_{13} - lucrul cu memoria MC1, MC2 respectiv MC3;

a_{14} - selectarea marcajului în funcţie de memoria cu care s-a lucrat;

a_{15} - eliberarea MCG;

a_{16} - selectarea traseului marcajului în funcţie de procesorul care a emis cererea de lucru cu memoria respectivă, pentru a iniţia un nou ciclu.

Descrierea formală a modelului este:

$L = \{b_1[7], \ldots, b_{19}[7], b_{20}\}$

$P = \{r_1, r_2, r_3, r_4\}$

$R = \{r_1, r_2, r_3, r_4\}$

$A = \{a_1, \ldots, a_{16}\}$

$\xi = \{t, \lambda, \mu\}$

$M_o(b_1) = 1;\ M_o(b_2) = 1;\ M_o(b_3) = 1;\ M_o(b_{20}) = 1$

Procedurile de tranziție sînt:

$a_1 = (T(b_1, b_4),\ o, [T \rightarrow M(b_4(1)) := 1;\ M(b_4(2)) := DEXP(\lambda);$
$M(b_4(3)) := DEXP(\mu);\ M(b_4(4)) := MM1;\ M(b_4(7)) := M(b_4(2)) + M(b_4(3))])$

$a_2 = (T(b_2, b_5),\ o, [T \rightarrow M(b_5(1)) := 2;\ M(b_5(2)) := DEXP(\lambda);$
$M(b_5(3)) := DEXP(u);\ M(b_5(4)) := MM2;\ M(b_5(7)) := M(b_5(2)) + M(b_5(3))])$

$a_3 = (T(b_3, b_6), o, [T \rightarrow M(b_6(1)) := 3;\ M(b_6(2)) := DEXP(\lambda);$
$M(b_6(3)) := DEXP(\mu);\ M(b_6(4)) := MM3;\ M(b_6(7)) := M(b_6(2)) + M(b_6(3))])$

$a_4 = (T(b_4, b_7), M(b_4(7)), -)$

$a_5 = (T(b_5, b_8), M(b_5(7)), -)$

$a_6 = (T(b_6, b_9), M(b_6(7)), -)$

$a_7 = (Y_3(r_1, b_7, b_8, b_9, Q_1);\ (o, o, o), -)$

$a_8 = (T(Q_1, b_{10}),\ o,\ [T \rightarrow M(b_{10}(3)) := DEXP(\mu)])$

$a_9 = (\ldots(b_{10}, b_{20}, b_{11}),\ o, -)$

$a_{10} = (X_3(r_2, b_{11}, b_{12}, b_{13}, b_{14}), (o, o, o), -)$

$a_{11} = (T(b_{12}, b_{15}),\ M(b_{12}(3)), -)$

$a_{12} = (T(b_{13}, b_{16}),\ M(b_{13}(3)), -)$

$a_{13} = (T(b_{14}, b_{17}),\ M(b_{14}(3)), -)$

$a_{14} = (Y_3(r_3, b_{15}, b_{16}, b_{17}, b_{18}), (o, o, o), -)$

$a_{15} = (F(b_{18}, b_{19}, b_{20}), o, [T \rightarrow M(b_{19}(5)) := M(b_{18}(5)) + M(b_{18}(2));$
$M(b_{18}(6)) := M(b_{18}(5))(t)$

$a_{16} = (X_3(r_4, b_{19}, b_1, b_2, b_3), (o, o, o), -)$

4.3.3.3. SCHEMA LOGICA A PROGRAMULUI GPSS

In acest program facilitatea 1 reprezintă MC1, facilitatea 2 reprezintă MC2, facilitatea 3 reprezintă MC3, facilitatea 4 reprezintă MCG. Sirul de aşteptare 1 se formează la MCG. Funcţiile 2, 3 şi 4 reprezintă memoria cerută:

$$FM1(x) = \begin{cases} 2, & \text{dacă } x \in [0,0.5) \\ 3, & \text{dacă } x \in [0,5,1] \end{cases}$$

$$FM2(x) = \begin{cases} 1, & \text{dacă } x \in [0,0.5) \\ 3, & \text{dacă } x \in [0.5,1] \end{cases}$$

$$FM3(x) = \begin{cases} 1, & \text{dacă } x \in [0,0.5) \\ 2, & \text{dacă } x \in [0.5,1] \end{cases}$$

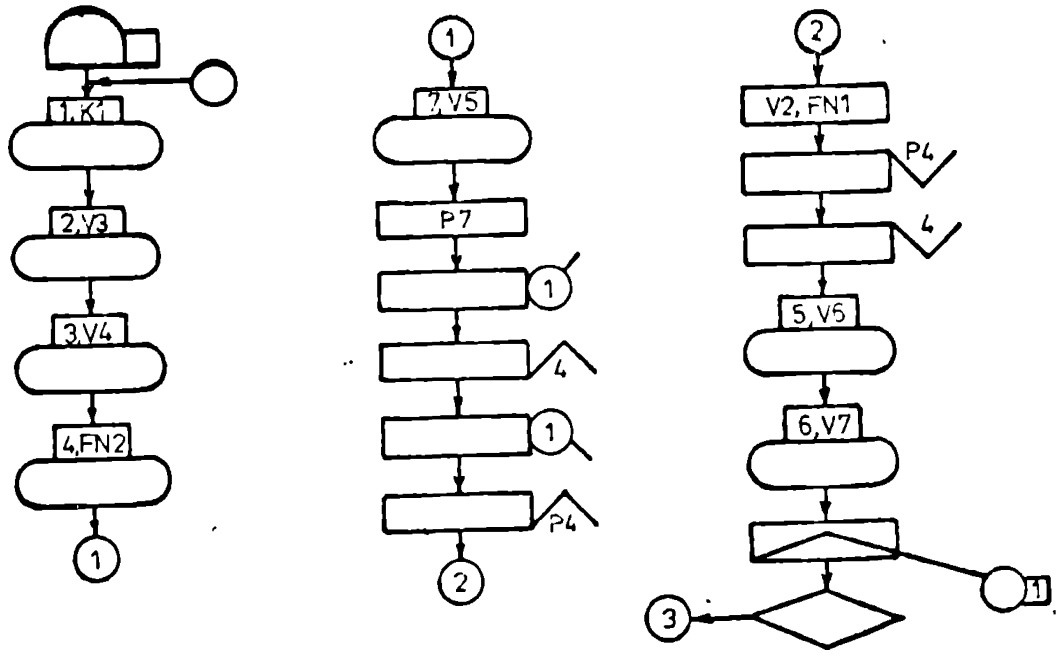


Fig.4.3.3-4

4.3.4.4. ARHITECTURA 4. (pxpxl)

Dacă nu este disponibilă o MCX2 se poate utiliza o alternativă a arhitecturii 3.

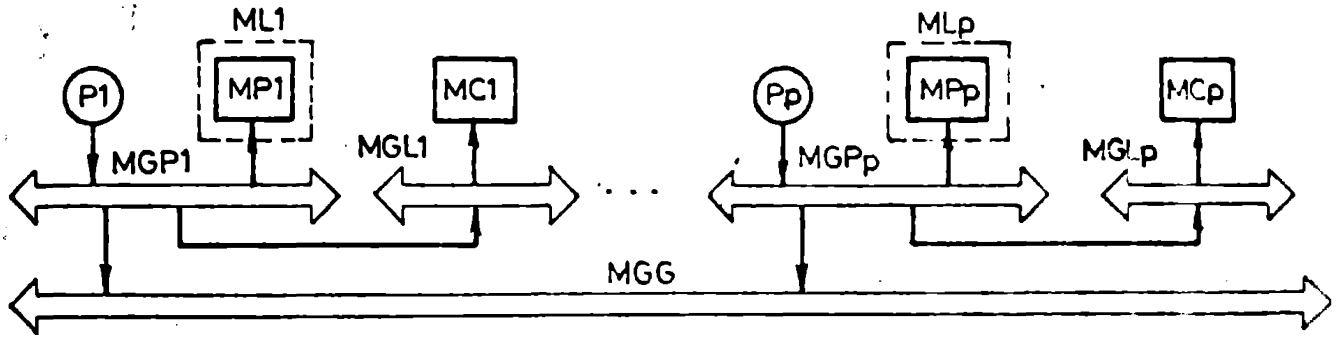


Fig. 4.3.4 -1

i) Un procesor lucrează cu MP numai prin intermediul MGP. Accesul la modulul de MC local are loc prin magistrala proprie MGP şi prin MGL corespunzătoare. Pentru accesul la o MC externă procesorul utilizează MGP, MG şi MGL a destinaţiei.

ii) Conflictele apar în utilizarea MGG şi a MC.

iii) Ca în cazul arhitecturii 2 se dă prioritate procesorului ce are controlul MGG.

iv) Ca şi în cazul arhitecturii 1 activitatea procesoarelor fiind simetrică, un procesor generează şi primeşte mesaje astfel că un procesor este întrerupt cu o frecvenţă ( ) dublă faţă de frecvenţa de generare a mesajelor.

$$\lambda = p\lambda_p \qquad (4.4-1)$$

v) Disciplina de deservire a cozilor este FIFO.

4.3.A 4.1. MODELAREA CU LANT MARKOV

Cazul (2x2x2)

In acest caz starea este definită ( ) astfel:

$$\left(\begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix}, \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix}\right) \qquad (4.3.4.1-1)$$

Diagrama de tranziţie a stărilor este următoarea:

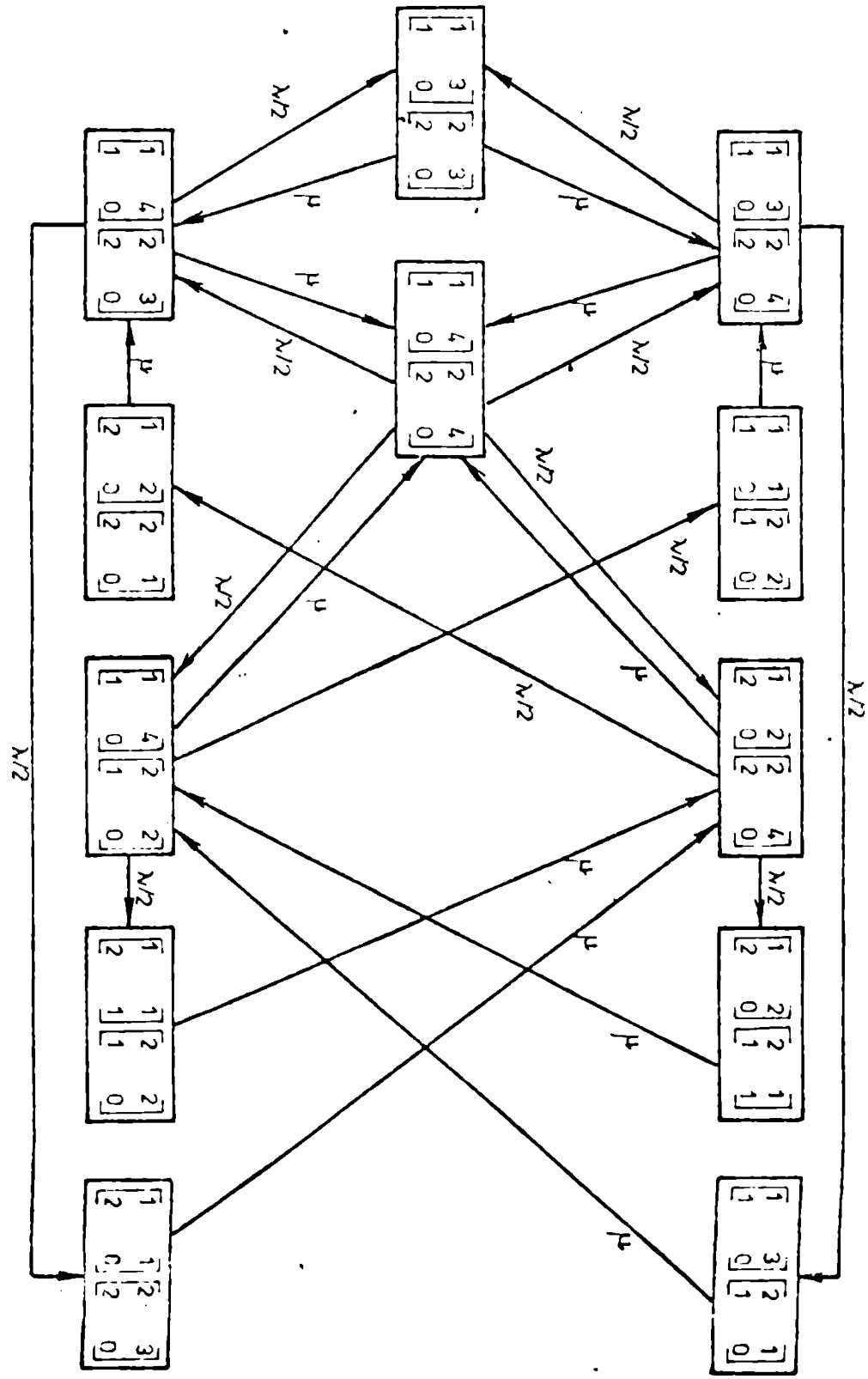


Fig. 4.3.4

Simetria sistemului poate fi utilizată pentru obţinerea lanţului Markov comasat. Structura acestuia e similară cu a lanţului exact cu singura deosebire că procesoarele sînt ordonate conform stării lor (sînt grupate: procesoarele active împreună, procesoarele din coadă împreună,...) fără a ţine cont de numărul de ordine al procesorului ci numai de cel al memoriei. Poziţia unei matrice a stării nu mai corespunde numărului de ordine al procesorului. Poziţia numărului de ordine al memoriei indică poziţia din sistemul ordonat (stare) de procesorul asociat.

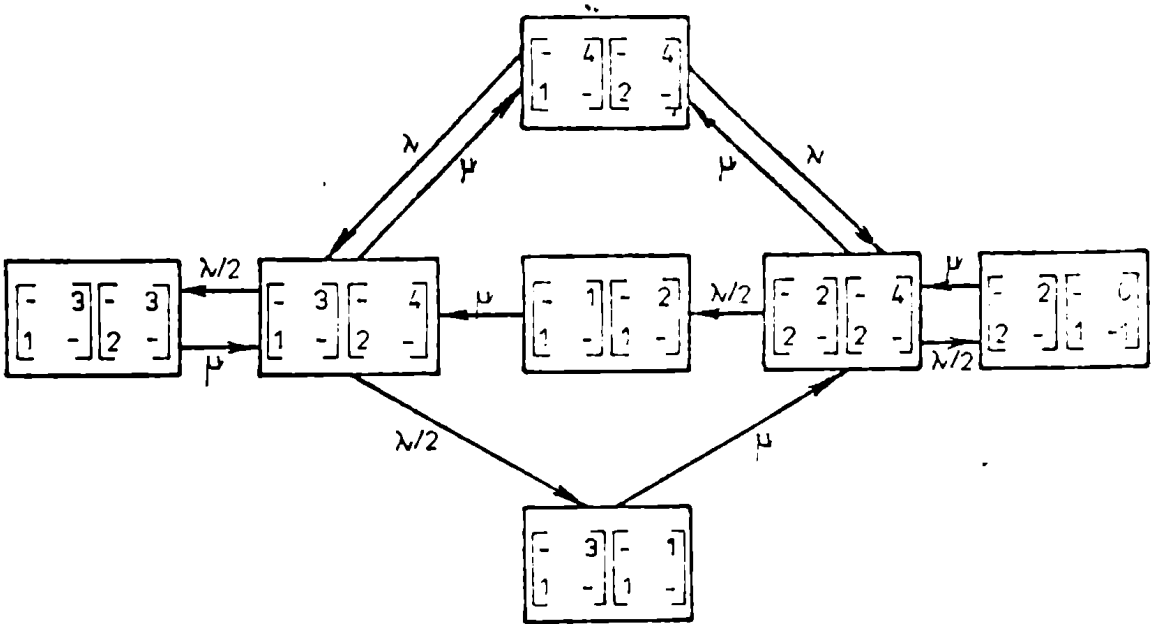


Fig.4.3.4-3

După calculul probabilităţilor de tranziţie valoarea puterii de prelucrare rezultă

$$P = \frac{6\,(\rho + 1)}{7\rho^2 + 8\rho + 4} \qquad (4.3.4.1\text{-}2)$$

unde $\rho = 2\rho_p = 2\rho_i \frac{p-1}{p}$ (4.3.4.1-3)

4.3. 4.2. MODEL CU SA

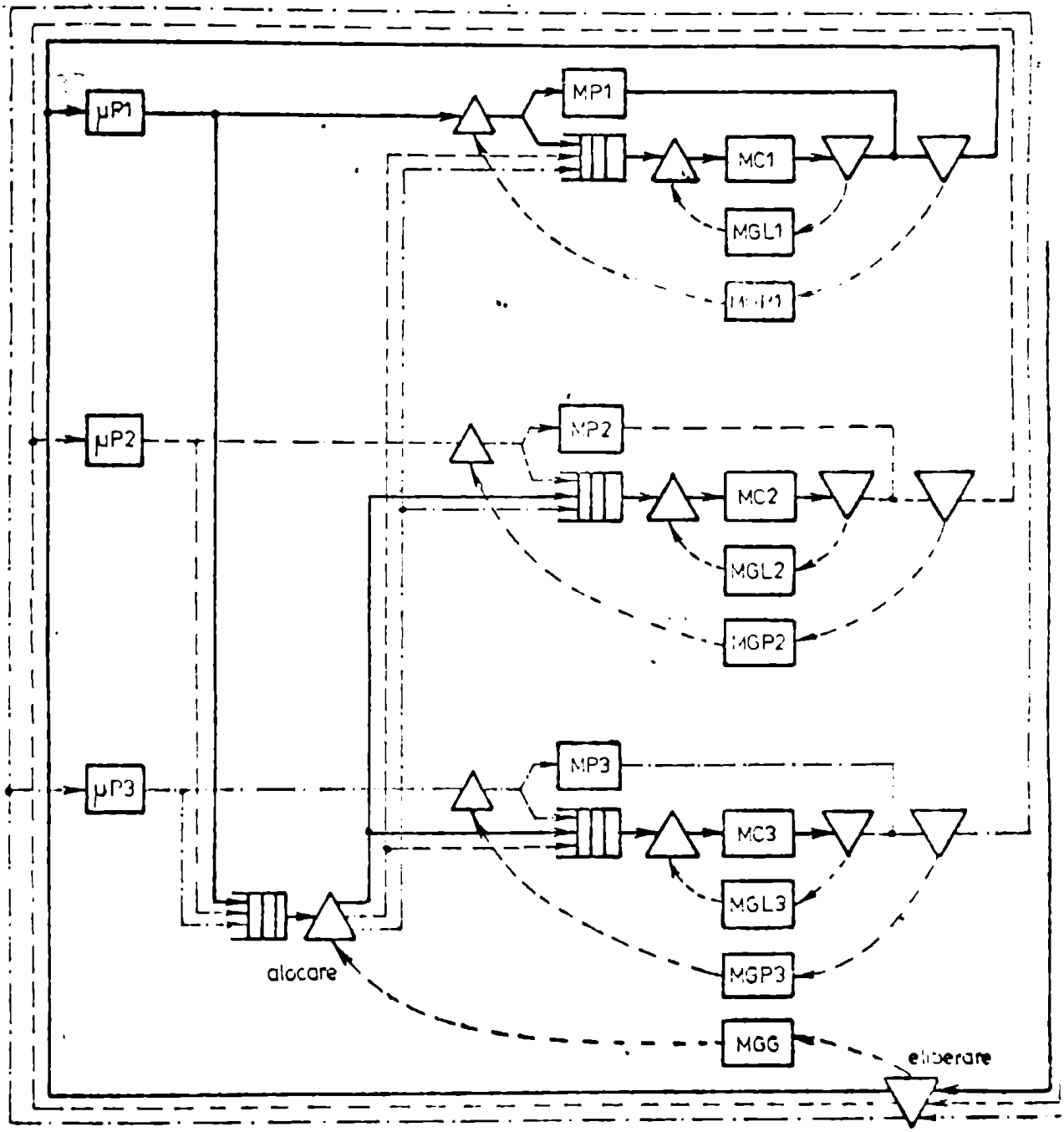


Fig.4.3.4-4

4.3.A4.3. MODELUL CU IE

Tranziţiile din Fig.4.3.4-5 au următoarea semnificaţie:

a_1, a_2, a_3 - generează timpul cît procesorul este activ, cît va lucra cu memoria, memoria cerută;

a_4, a_5, a_6 - prelucrare;

a_7, a_8, a_9 - selectează cererea cea mai prioritară de la procesо-

rul care a primit MGG) şi o ataşează în şirul la MGL;

a_{10}, a_{11}, a_{12} - detaşează o cerere din şir;

a_{13}, a_{14}, a_{15} - alocare MGL;

a_{16}, a_{17}, a_{18} - selectarea traseului în funcţie de necesitatea alocării MGG;

a_{19}, a_{20}, a_{21} - lucrul cu MG1, MG2 respectiv MG3;

a_{22}, a_{23}, a_{24} - eliberare MGL;

a_{25}, a_{26}, a_{27} - selectarea traseului marcajului în funcţie de necesitatea eliberării MGG;

a_{28} - selectarea marcajului care va primi MGG;

a_{29} - alocare MGG;

a_{30} - selectarea traseului marcajului în funcţie de memoria cerută;

a_{31} - selectarea marcajului care va elibera MGG;

a_{32} - eliberare MGG;

a_{33} - selectarea traseului în funcţie de MGL care trebuie eliberată.

Descrierea formală a modelului este:

$$L = \{ b_1\,[8], \ldots, b_{43}[8], b_{44}, \ldots, b_{47} \}$$

$$P = R = \{ r_1, \ldots, r_{10} \}$$

$$A = \{ a_1, \ldots, a_{33} \}$$

$$\xi = \{ t, \lambda, \mu \}$$

$$M_o(b_1) = 1;\ M_o(b_2) = 1;\ M_o(b_3) = 1;$$

$$M_o(b_{44}) = 1;\ M_o(b_{45}) = 1;\ M_o(b_{46}) = 1;\ M_o(b_{47}) = 1$$

unde atributul 8 al marcajelor va fi 1 dacă MGG este alocată şi 0 în caz contrar.

Procedurile de tranziţie sînt:

a_1 = (T(b_{28}, b_1), 0, [T→M (b_1(1)): = 1; M(b_1(2)): =

= DEXP (λ); M(b_1(3)): = DEXP (μ); M(b_1(4)): = FM; M(b_1(7)): =

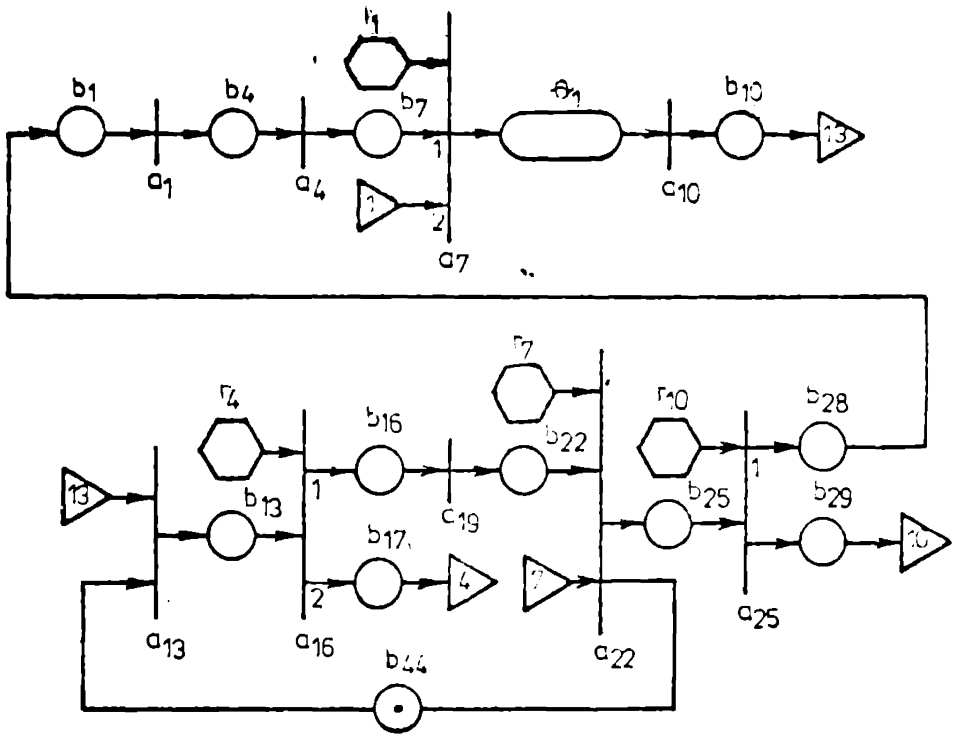
= M(b_1(2))/2]}

a_2 şi a_3 analog cu a_1

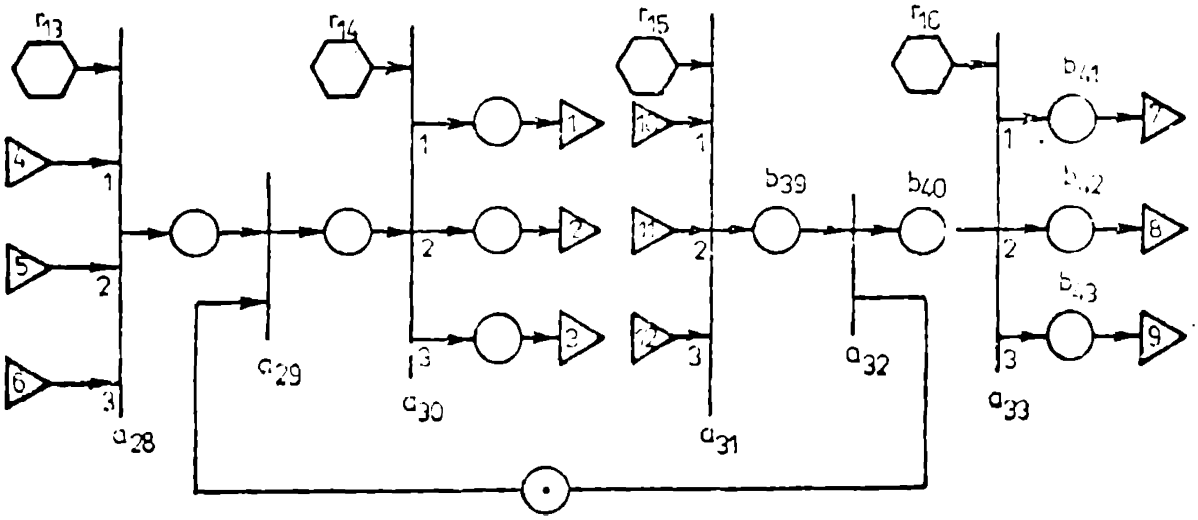
a_4 = (T(b_4, b_7), M(b_4(7)), -)

a_5 şi a_6 analog cu a_4

Fig.4.3.4-5



Analog pentru procesorul 2 și 3



a_7 = $(Y(r_1, b_7, b_{35}, Q_1), o, -)$

a_8 şi a_9 analog cu a_7

a_{10} = $(T(Q_1, b_{10}), o, -)$

a_{11} şi a_{12} analog cu a_{10}

a_{13} = $(J(b_{10}, b_{42}, b_{13}), o, -)$

a_{14}şi a_{15} analog cu a_{13}

a_{16} = $(X(r_4, b_{13}, b_{16}, b_{17}), (o,o), -)$

a_{17}şi a_{18} analog cu a_{16}

a_{19} = $(T(b_{16}, b_{22}), M(b_{16}(7)), [T \rightarrow M(b_{22}(5)) := M(b_{22}(5)) - M(b_{22}(6)) + M(b_{22}(2)); M(b_{22}(6)) := M(b_{22}(5))/t])$

a_{20} şi a_{21} analog cu a_{19}

a_{22} = $(Y(r_7, b_{22}, b_{41}, b_{25}), (o,o,) -)$

a_{23} şi a_{24} analog cu a_{22}

a_{25} = $(X(r_{10}, b_{25}, b_{28}, b_{29}), (o,o), -)$

a_{26} şi a_{27} analog cu a_{25}

a_{28} = $(Y_3(r_{19}, b_{17}, b_{19}, b_{21}, b_{34}), (o,o,o) -)$

a_{29} = $(J(b_{34}, b_{47}, b_{35}), o, [T \rightarrow M(b_{35}(8)) := 1])$

a_{30} = $(X_3(r_{14}, b_{34}, b_{35}, b_{36}, b_{37}), (o,o,o), -)$

a_{31} = $(Y_3(r_{15}, b_{29}, b_{31}, b_{33}, b_{38}), (o,o,o), -)$

a_{32} = $(F(b_{39}, b_{40}, b_{47}), o, [T \rightarrow M(b_{40}(8)) := o])$

a_{33} = $(X_3(r_{16}, b_{40}, b_{41}, b_{42}, b_{43}), (o,o,o), -)$

unde FM este analogă cu cea din 3.5.2.2.

Procedurile de tranziţie sînt:

r_1 : $[T \rightarrow M(r_1) := 1]$

analog r_2 şi r_3.

r_4 : $[(M(b_{13}(1)) = M(b_{13}(4))) \vee (M(b_{13}(8)) = 1) \rightarrow M(r_4) := 1 ;$
$T \rightarrow M(r_4) := 2]$

analog r_5 şi r_6

r_7 : $[T \rightarrow M(r_7) := 1]$

analog r_8 şi r_9

r_{10} : [(M(b_{25}(1))= M (b_{25}(4))→M(r_{10}): = 1; T→M(r_{10}): = 2]

analog r_{11} şi r_{12}

r_{13} : [T→M(r_{13}): = 1]

r_{14} : [(M(b_{35}(4)) = 1)→M(r_{14}): = 1; (M(b_{35}(4)) = 2)→M(r_{14}):=2;
T → M(r_{14}): = 3]

r_{15} : [T→M(r_{15}): = 1]

r_{16} : [(M(b_{40}(1)=1)→M(r_{16}): =1;(M(b_{40}(1)=2)→M(r_{16}): = 2 ;
T→M(r_{16}): =3]

4.3.A.4.4. SCHEMA LOGICA A PROGRAMULUI DE SIMULARE

In acest program facilităţile au următoarea semnificaţie: 1-MC1; 2 - MC2; 3 - MC3 ; 4 - MGL1; 5 - MGL2; 6 - MGL3 ; 7 - MGG. Sirul de aşteptare 1 se formează la MGL1, 2 la MGL2, 3 la MGL3 şi 4 la MGG. Funcţia FN2 reprezintă memoria cerută de oricare procesor şi este analoagă cu FN din modelul cu RE. Variabila 5 calculează p2 + p3, iar variabila 7, p4 + 3.

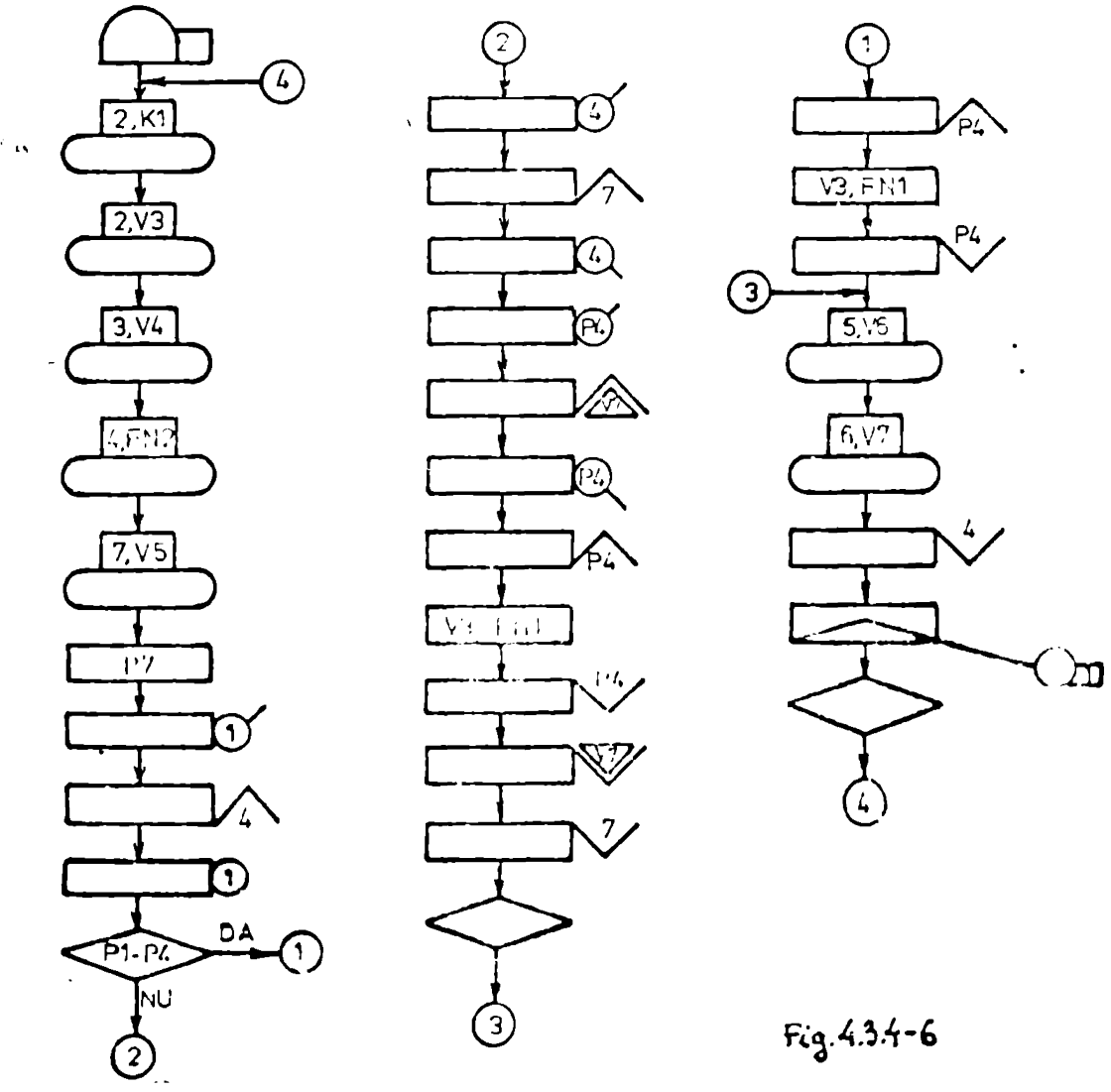


Fig.4.3.4-6

4.3.A.5. ARHITECTURA 5 : (pxmx2), m>1, m ≤ p

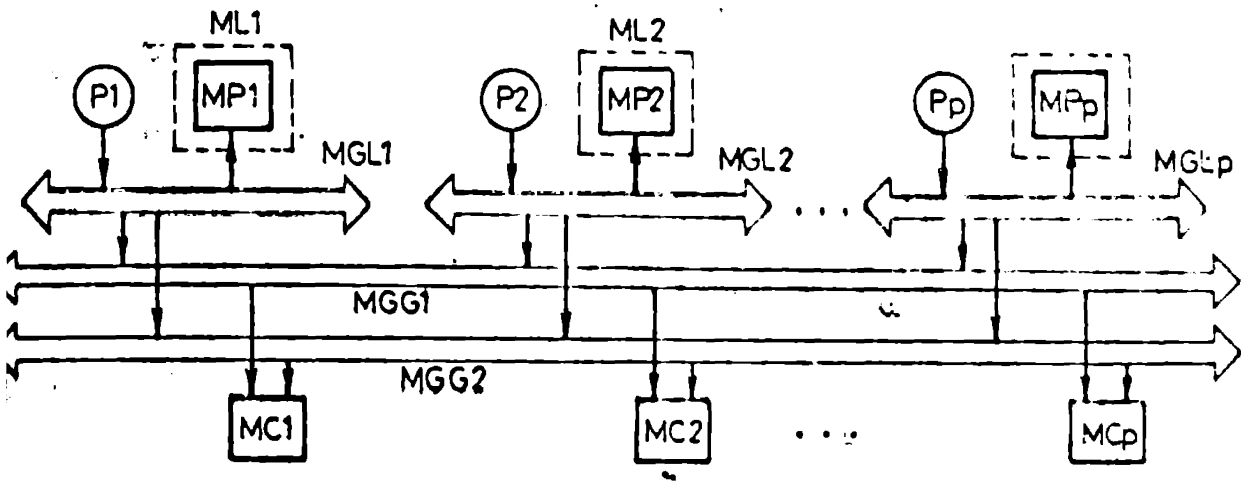


Fig.4.3.5-1

i) MC sînt externe tuturor procesoarelor.

ii) Accesul la MC presupune ca ML şi una din cele două MG să fie libere.

iii) m>1 altfel una din MG ar fi inutilă

dacă p = m = 2 SMM este de tip "crossbar"

iv) Datorită faptului că MC trebuie adresate pentru citirea şi scrierea mesajelor legătura dintre timpul mediu de prelucrare şi timpul mediu dintre două mesaje la acelaşi procesor este: $\lambda = 2\lambda_p$. (4.3.5-1)

v) Disciplina de deservirea cozilor este FIFO.

4.3.A.5.1. MODEL SA

Acest caz va fi tratat şi în cazul mai general pxmxb cu b magistrale utilizînd modele de tip lanţ Markov. Utilizînd însă chiar lanţurile Markov comasate conduce la un număr foarte mare de stări. Din acest motiv s-a recurs în acest caz la modelarea cu sisteme de cozi de aşteptare de tip M/M/2/P/P în care MG sînt singurele surse potenţiale de conflict.

Urmînd modelul lui Palm M/M/S şi în aceleaşi condiţii la modelul M/M/1 se pot deduce următoarele rezultate:

Spunem că sistemul de aşteptare se află în starea S_j la momentul t dacă j procesoare nu sînt în stare activă. Evident dacă j < b (numărul magistralelor) nu se formează şir de aşteptare şi există b-j MGG libere; dacă j> b apar şiruri de aşteptare (în ordinea PIPS - FIFO).

In intervalul $(t, t + \Delta t)$ pot avea loc următoarele tranziţii.

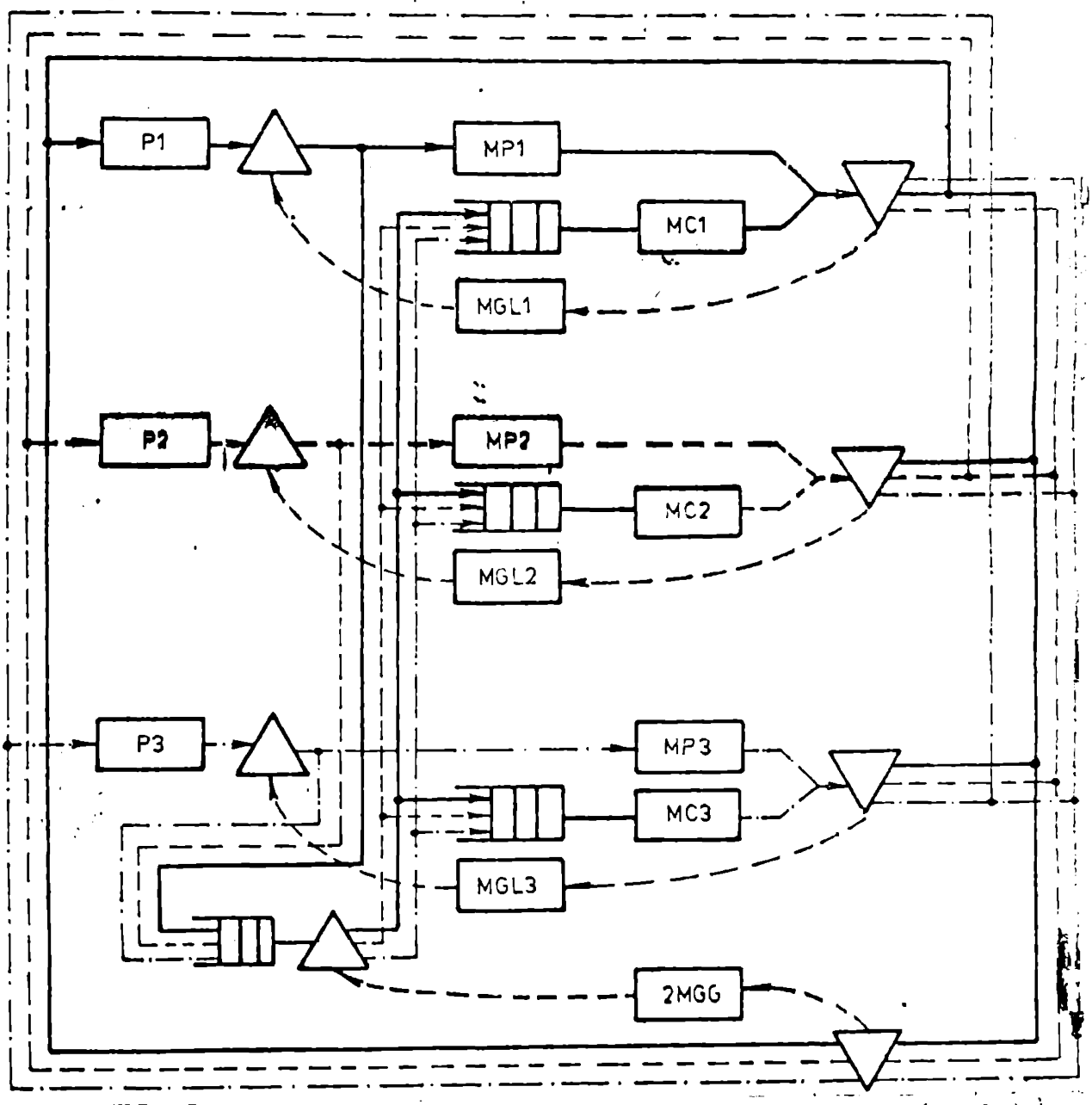


Fig. 43.5-2

i) $S_j \to S_i$ cu probabilitatea:

$(1-\mu j)\,\Delta t\,[1-(p-j)\lambda\Delta t]$, dacă $j = 0,1,\ldots, b-1$

sau

$(1-\mu b)\,\Delta t\,[1-(p-j)\lambda\Delta t]$ dacă $j = b, b+1, \ldots, p$

ii) $S_{j+1} \to S_j$ cu probabilitatea:

$(j+1)\mu\Delta t$ dacă $j=0,1,\ldots, b-1$

$b\mu\Delta t$ dacă $j=b, b+1, \ldots p$

iii) $S_{j-1} \to S_j$ cu probabilitatea

$(p-j+1)\lambda\Delta t$

Aşadar:

$$P_o(t+\Delta t) = \mu \Delta t\, P_1(t) + (1-p\lambda\Delta t)P_o(t) + \varepsilon(\Delta t)$$

$$\varepsilon(\Delta t) \to o \text{ cînd } \Delta t \to o$$

$$P_j(t+\Delta t) = \{1-[uj + (p-j)\lambda]\Delta t\}P_j(t) + (j+1)\mu \Delta t\, P_{j+1}(t) +$$

$$+ (p-j+1)\lambda \Delta t\, P_{j-1}(t) + \varepsilon(\Delta t),\ j = 1,2,\ldots,b-1$$

$$P_j(t+\Delta t) = \{1-[b\mu+(p-j)\lambda]\Delta t\}P_j(t) + b\mu\Delta t\, P_{j+1}(t) +$$

$$+ (p-j+1)\lambda\Delta t\, P_{j-1}(t) + \varepsilon(\Delta t),\ j=b,b+1,\ldots,p$$

Deci, împărţind cu Δt ambii membrii ai ecuaţiilor de mai sus şi trecînd la limită se obţine sistemul de ecuaţii diferenţiale:

$$\frac{d}{dt} P_o(t) = \mu P_1(t) - p\lambda P_o(t)$$

$$\frac{d}{dt} P_j(t) = -[\mu j+(p-j)\lambda] P_j(t)+(j+1)\mu P_{j+1}(t) + (p-j+1)\lambda P_{j-1}(t)$$

$$j=1,2,\ldots,b-1$$

$$\frac{d}{dt} P_j(t) = -[b\mu+(p-j)\lambda] P_j(t)+b\mu P_{j+1}(t)+(b-j+1)\lambda P_{j-1}(t)$$

$$j=b,b+1,\ldots p$$

In cazul echilibrului statistic:

$$\lim_{t\to\infty} P_j(t) = p_j \qquad \text{şi} \qquad \frac{d}{dt} P_j(t) = o$$

sistemul devine:

$$p\lambda p_o = \mu p_1$$

$$(j+1)\mu p_{j+1} = [bu_j + (p-j)\lambda]\cdot p_j-(p-j+1)\lambda p_{j-1}$$

$$j =1,2,..b-1.$$

$$b\mu\, p_{j+1} = [b\mu + (p-j)\lambda]\, p_j-(p-j+1)\lambda p_{j-1}$$

$$j = b,b+1,\ldots p.$$

Soluţiile acestui sistem sînt:

$$p_j = \begin{cases} C_p^j \rho^j p_o & \text{dacă } j = o,1,\ldots b-1 \\ \dfrac{p(p-1)\ldots(p-j+1)}{b^{j-p+1}(b-1)!}\rho^j p_o & \text{dacă } j=b,b+1\ldots p. \end{cases}$$

Cum:

$$\sum_{j=o}^{p} p_j = 1 \ldots$$

se obţine:

$$p_o = \left[\sum_{j=o}^{b-1} C_p^j \rho^j + \sum_{j=b}^{p} \rho^j \frac{p!}{b!\,(p-j)!\,b^{j-p}} \right]^{-1}$$

sau schimbînd indicii de sumare:

$$p_o = \left[\sum_{j=o}^{b-1} C_p^j \rho^j + \frac{p!}{(p-b)!\,b!} \rho^b \sum_{k=o}^{p-b} \frac{(p-b)!}{(p-b-k)!} \left(\frac{\rho}{b}\right)^k \right]^{-1}$$

Notînd:

$$A(\rho,\ b;p-b) = \sum_{k=o}^{p-b} \frac{(p-b)!}{(p-b-k)!} \left(\frac{\rho}{b}\right)^k$$

şi cum legea lui Poisson este dată de:

$$\tilde{p}(i,\alpha) = e^{-\alpha} \frac{\alpha^i}{i!}, \quad i \in N$$

în care luăm $i = p - b$ şi $\alpha = b\rho^{-1}$

Atunci:

$$\tilde{p}_o^{-1} = \sum_{j=o}^{b-1} C_p^j \rho^j + C_p^b \rho^b \frac{1-\tilde{p}(i+1,\alpha)}{\tilde{p}(i,\alpha)}$$

sau:

$$p_o^{-1} = \frac{1}{e^{-\alpha}\frac{\alpha^p}{p!}} \left[\sum_{k=o}^{b-1} \left(\frac{b}{\rho}\right)^{-k} \frac{b^k}{k!(p-k)!} e^{-\alpha}\alpha^p + \right.$$

$$\left. + \frac{\rho^b}{b!(p-b)!} e^{-\alpha}\alpha^p \frac{1-\tilde{p}(i+1,\alpha)}{e^{-\alpha}\frac{\alpha^{p-b}}{(p-b)!}} \right] =$$

$$= \frac{1}{p(p,\alpha)} \left[\sum_{k=o}^{b-1} \frac{b^k}{k!} \tilde{p}(N-k,\alpha) + \frac{b^b}{b!}(1-\tilde{p}(i+1,\alpha)) \right]$$

notînd:

$$B(p,b,\alpha) = \sum_{k=o}^{b-1} \frac{b^k}{k!} \tilde{p}(p-k,\alpha) + \frac{b^b}{b!}(1-\tilde{p}(i+1,\alpha))$$

$$p_o = \frac{\tilde{p}(p,\alpha)}{B(p,b,\alpha)}$$

şi

$$p^j = \begin{cases} \frac{b^j}{j!} \cdot \frac{\tilde{p}(p-j,\alpha)}{B(p,b,\alpha)} & \text{dacă } j = o,1\ldots\ b-1 \\ \frac{b^b}{b!} \cdot \frac{\tilde{p}(p-j,\alpha)}{B(j,b,\alpha)} & \text{dacă } j = b,b+1,\ldots,\ p. \end{cases}$$

caz în care puterea de prelucrare P devine

$$P = \frac{\mu b}{\lambda} \cdot \frac{B(p-1,b,\alpha)}{B(p,b,\alpha)}$$

In cazul în care b = 2

$$P = \frac{2\mu}{\lambda} \cdot \frac{B(p-1,2,\alpha)}{B(p,2,\alpha)}$$

4.3.3.5.2. MODELUL CU RE

Tranziţiile din fig. 4.3.5-3 au următoarea semnificaţie:

a_1, a_2, a_3 - generează timpul cît procesorul este activ, memoria cerută şi timpul cît va lucra cu memoria;

a_4, a_5, a_6 - prelucrare;

a_7 - selectarea marcajului care va intra în şirul de aşteptare pentru ocuparea unei MCC;

a_8 - cerere de magistrală ş

a_9 - alocarea unei magistrale;

a_{10} - selectarea traseului marcajului în funcţie de memoria cerută;

a_{11}, a_{12}, a_{13} - intrarea în şirul de aşteptare la MC1, MC2 respectiv MC3;

a_{14}, a_{15}, a_{16} - ieşirea din şirul de aşteptare respectiv;
a_{17}, a_{18}, a_{19} - alocarea MC1, MC2 respectiv MC3;
a_{20}, a_{21}, a_{22} - lucrul cu memoria MC1, MC2, respectiv MC3;
a_{23}, a_{24}, a_{25} - eliberarea MC1, MC2 respectiv MC3;
a_{26} - selectarea marcajului care va elibera o MSG;
a_{27} - selectarea traseului marcajului în funcţie de procesorul care a lucrat cu memoria.

59

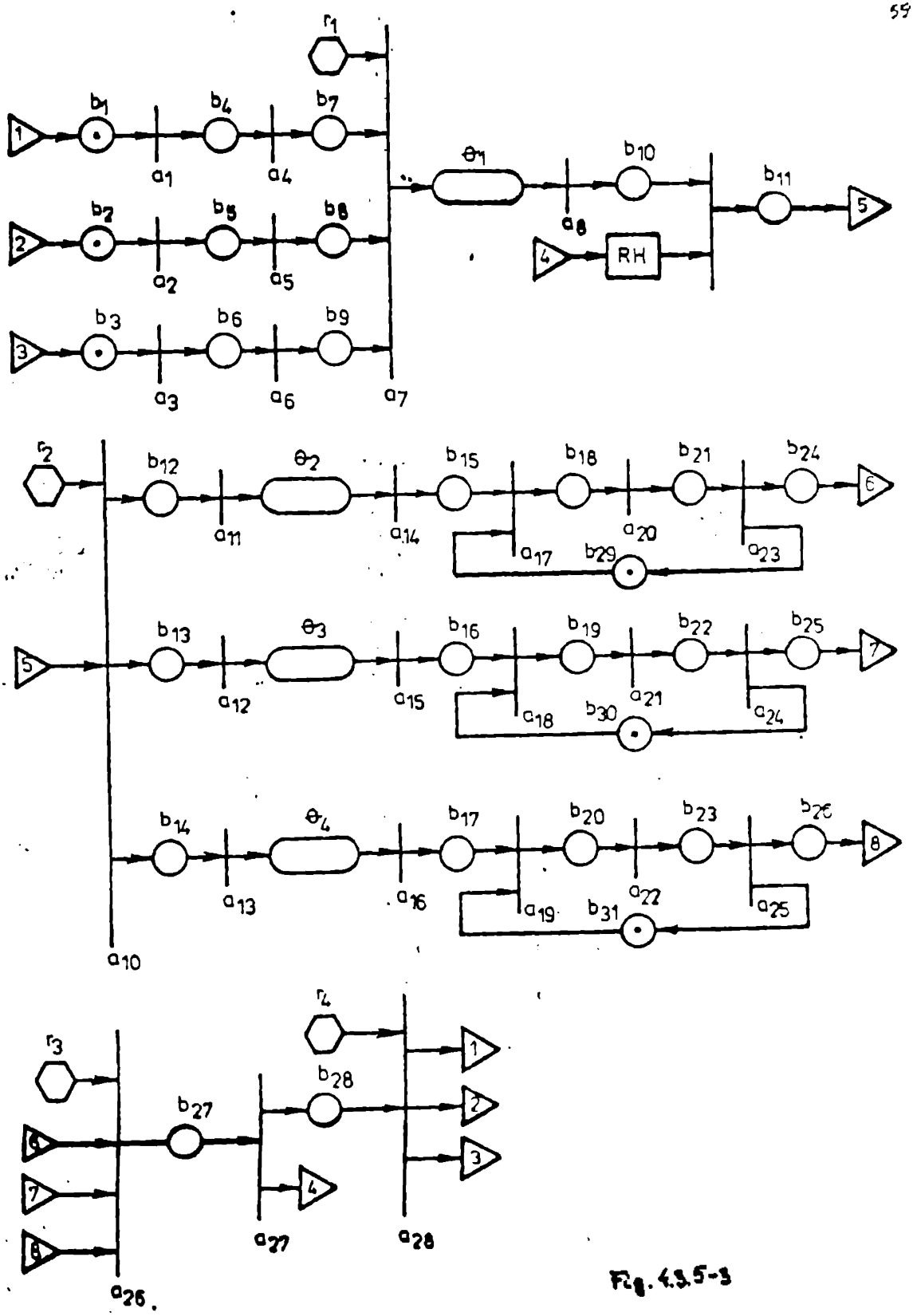


Fig. 4.3.5-3

Macroreţeaua RH reprezintă alocatorul pentru cele 2 magistrale globale.

Descrierea formală a modelului este:

$L = \{ b_1 [7], \ldots, b_{28}[7], b_{29}, b_{30}, b_{31} \}$

$P = R = \{r_1, r_2, r_3, r_4\}$

$A = \{a_1, a_2, \ldots, a_{28}\}$

$\xi = \{t, \lambda, \mu\}$

$M_o(b_1) = 1;\ M_o(b_2) = 1;\ M_o(b_3) = 1;$

$M_o(b_{29}) = 1;\ M_o(b_{30}) = 1;\ M_o(b_{31}) = 1.$

Procedurile de tranziţie sînt:

$a_1 = (T(b_1, b_4), o, [T \rightarrow M(b_4(1)) := 1;\ M(b_4(2)) := DEXP(\lambda);$
$M(b_4(3)) := DEXP(\mu);\ M(b_4(4)) := FM;\ M(b_4(7)) := M(b_4(2))/2])$

$a_2 = (T(b_2, b_5), o, [T \rightarrow M(b_5(1)) := 2;\ M(b_5(2)) := DEXP(\lambda);$
$M(b_5(3)) := DEXP(\mu);\ M(b_5(4)) := FM;\ M(b_5(7)) := M(b_5(2))/2])$

$a_3 = (T(b_3, b_6), o, [T \rightarrow M(b_6(1)) := 3;\ M(b_6(2)) := DEXP(\lambda);$
$M(b_6(3)) := DEXP(\mu);\ M(b_6(4)) := FM;\ M(b_6(7)) := M(b_6(2))/2])$

$a_4 = (T(b_4, b_7),\ M(b_4(7)),\ -)$

$a_5 = (T(b_5, b_8),\ M(b_5(7)),\ -)$

$a_6 = (T(b_6, b_9),\ M(b_6(7)),\ -)$

$a_7 = (Y_3(r_1, b_7, b_8, b_9, Q_1), (o, o, o),\ -)$

$a_8 = (T(Q_1, b_{10}), o,\ -)$

$a_9 = (I(b_{10}, OH, b_{11}), o,\ -)$

$a_{10} = (X_3(r_2, b_{11}, b_{12}, b_{13}, b_{14}), (o, o, o),\ -)$

$a_{11} = (T(b_{12}, Q_2), o,\ -)$

$a_{12} = (T(b_{13}, Q_3), o,\ -)$

$a_{13} = (T(b_{14}, Q_4), o,\ -)$

$a_{14} = (T(Q_2, b_{15}), o,\ -)$

$a_{15} = (T(Q_3, b_{16}), o\ -)$

$a_{16} = (T(Q_4, b_{17}), o\ -)$

a_{17}= (J (b_{15},b_{29},b_{18}),o, -)

a_{18}= (J (b_{16},b_{3o},b_{19}), o, -)

a_{19}= (J (b_{17},b_{31},b_{2o}), o, -)

a_{2o}= (T (b_{18},b_{21}), M(b_{18}(3)), -)

a_{21}= (T (b_{19},b_{22}), M(b_{19}(3)), -)

a_{22}= (T (b_{2o},b_{23}), M(b_{2o}(3)), -)

a_{23}= (T (b_{21},b_{24}), o, -)

a_{24}= (T (b_{22},b_{25}), o,-)

a_{25}= (T (b_{23},b_{26}), o,-)

a_{26}= (Y_3(r_3,b_{24},b_{25},b_{26},b_{27}),(o,o,o), -)

a_{27}= (F (b_{27},b_{28},RH),o, [T→M(b_{28}(5)):=M(b_{27}(5)) +
+ M(b_{27}(2)); M(b_{28}(6)): = M(b_{28}(5))/t])

a_{28}= (X_3(r_4,b_{28},b_1,b_2,b_3),(o,o,o), -)

Procedurile de rezoluţie sînt :

r_1: [T→M(r_1): = 1]

r_2: [(M(b_{11}(4)) = 1)→M(r_2):=1; (M(b_{11}(4))=2)→M(r_2:=2;
T→M (r_2): = 3]

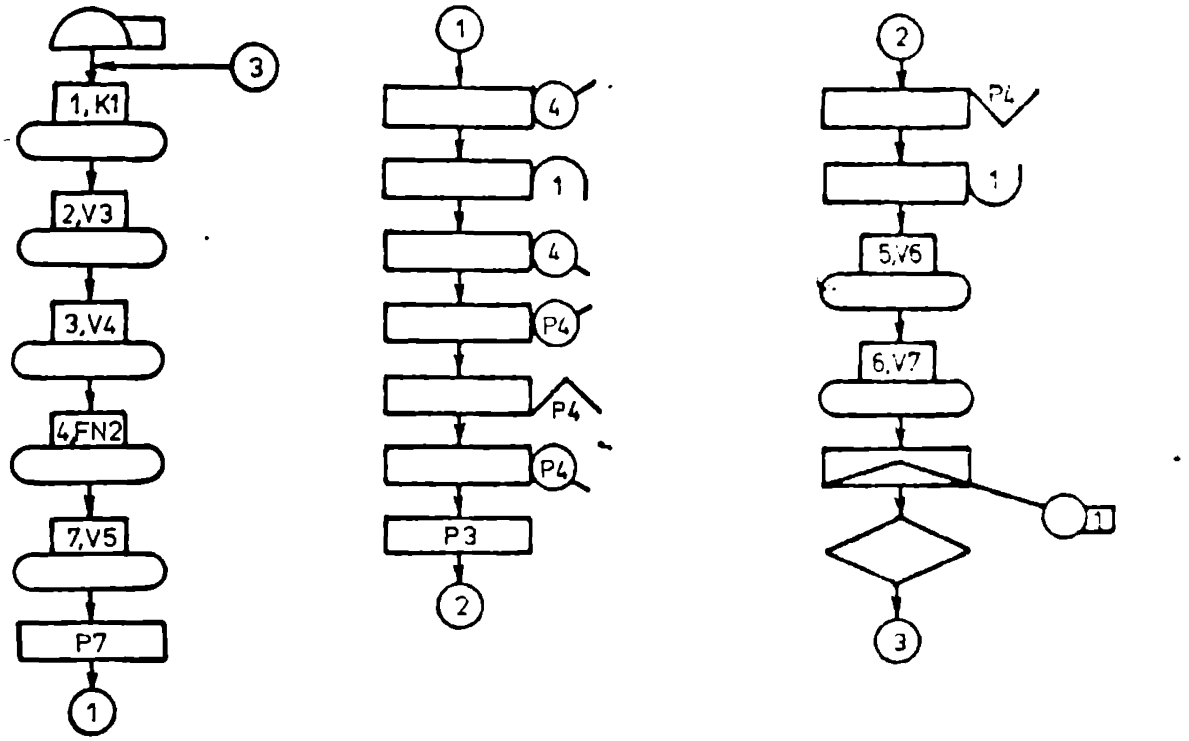
r_3: [T→M(r_3): = 1]

r_4: [(M(b_{19}(1)) = 1)→M(r_4): = 1; (M(b_{19}(1)) =
= 2)→M(r_4): = 2; T→M (r_4) : = 3]

4.3.A.5.3. SCHEMA LOGICA A PROGRAMULUI GPSS

In acest program facilitatea 1 reprezintă MC1, facilitatea 2 MC2, facilitatea 3 MC3. Storage-ul 1 reprezintă cele două MGG. Sirurile de aşteptare 1, 2, 3, 4 se formează la MC1, MC2, MC3 respectiv MGG. Funcţia 2 este analoagă cu cea din 4.3.2.4. Variabila 8 reprezintă MGL necesară, în funcţie de memoria ocupată.

Fig. 4.3.6-4



4.3.4.6. ARHITECTURA 6 (pxpx2)

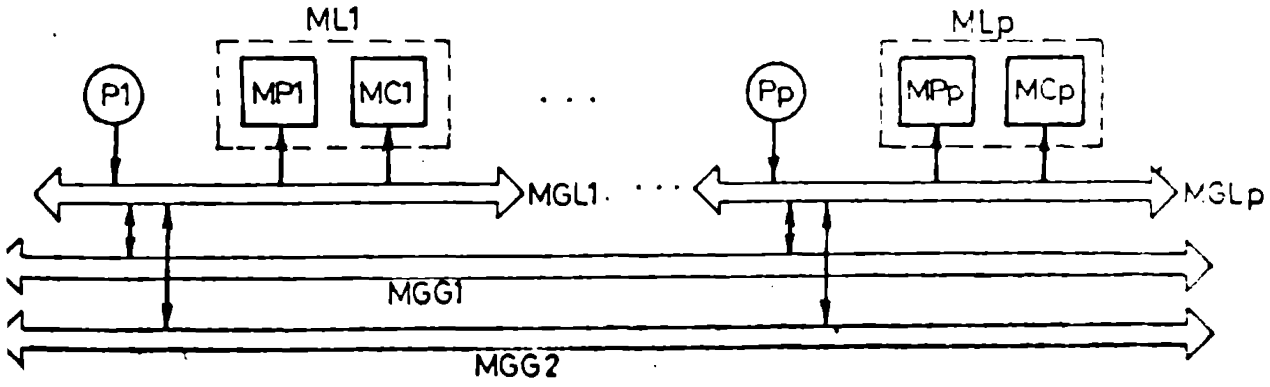


Fig. 4.3.6-1

i) Memoria comună este distribuită în modulele locale ale fiecărui procesor.

ii) M L a fiecărui procesor este alcătuită din două părți: memorie proprie și memorie comună.

iii) Memoriile comune externe pot fi adresate utilizînd MGL a sursei și a destinației cît și una din MGG.

iv) Procesoarele care au cîștigat accesul la una din MGG pot întrerupe procesoarele ce lucrează cu MGL, Procesoarele întrerupte trec în starea blocată.

v) Disciplina de deservire a cozii este FIFO.

vi) Ca şi în cazul magistralei unice

$$1/\lambda = 1/\lambda_p + 1/\mu$$

sau

$$\lambda = \lambda_p \mu/(\lambda_p + \mu)$$

4.3. .6.1. MODELARE CU LANTURI MARKOV

Pentru starea sistemului se utilizează aceeaşi definiţie

$$\left(\begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix}, \ldots, \begin{bmatrix} a_p & b_p \\ c_p & d_p \end{bmatrix}\right)$$

cu semnificaţiile menţionate anterior.

Cazul 6.1.

Un prim model aproximativ se obţine redefinind starea astfel:

$S = (n_{34}, n_2, n_o, n_1)$

Se observă că $p - n_{34} - n_2 - n_o - n_1$ procesoare a căror MGL e liberă stau în coadă pentru resurse ocupate.

$S_o = (p,o,oo)$
$S_1 = (p-2,1,1.o)$
$S_2 = (p-3,1,1,o)$
$S_3 = (p-2,1,o,1)$
$S_4 = (p-3,1,o,1)$
$S_5 = (p-4,2,2,o)$
$S_6 = (p-5,2,2,o)$
.
.
.
$S_{p+1} = (o,2,2,o)$

$S_{p+2} = (p-4,1,1,o)$
$S_{p+3} = (p-5,1,1,o)$
$S_{2p-1} = (p-4,2,1,1)$
$S_{2p} = (p-5,2,1,1)$

$S_{2p-2} = (o,1,1,o)$
$S_{3p-5} = (o,2,1,1)$

$S_{3p-4} = (p-4,1,o1)$
$S_{3p-3} = (p-5,1,o1)$
$S_{4p-7} = (p-4,2,o,2)$
$S_{4p-6} = (p-5,2,o,2)$

$S_{4p-8} = (o,1,o,1)$
$S_{5p-11} = (o,2,o,2)$

Sînt în total deci 5p - 1o stări.

Diagrama de tranziţie a stărilor pentru modelul 6.1. este.

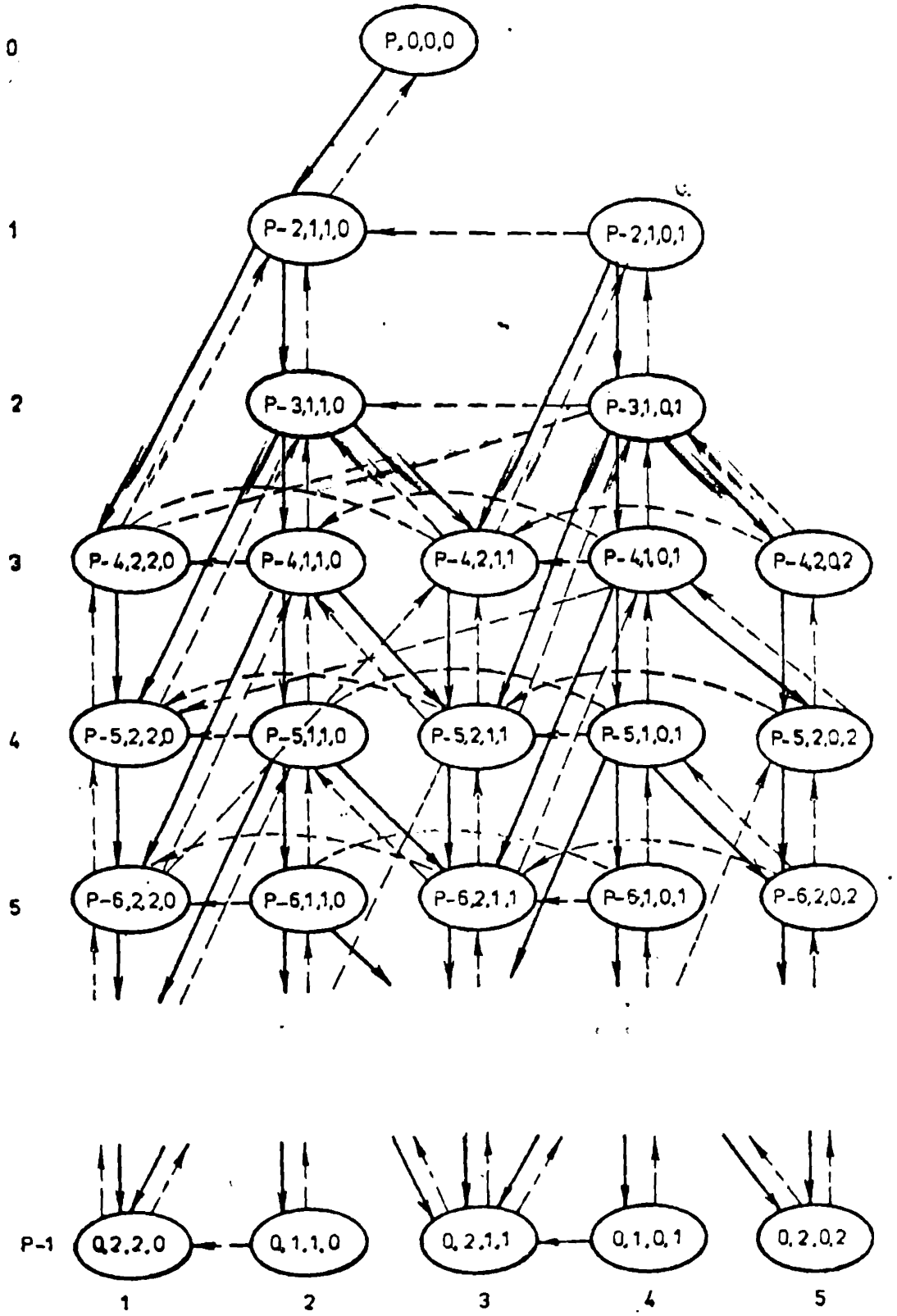


Fig.4.3.6-2

Cazul 6.1. (4x4x2)

Diagrama de tranziţie a stărilor

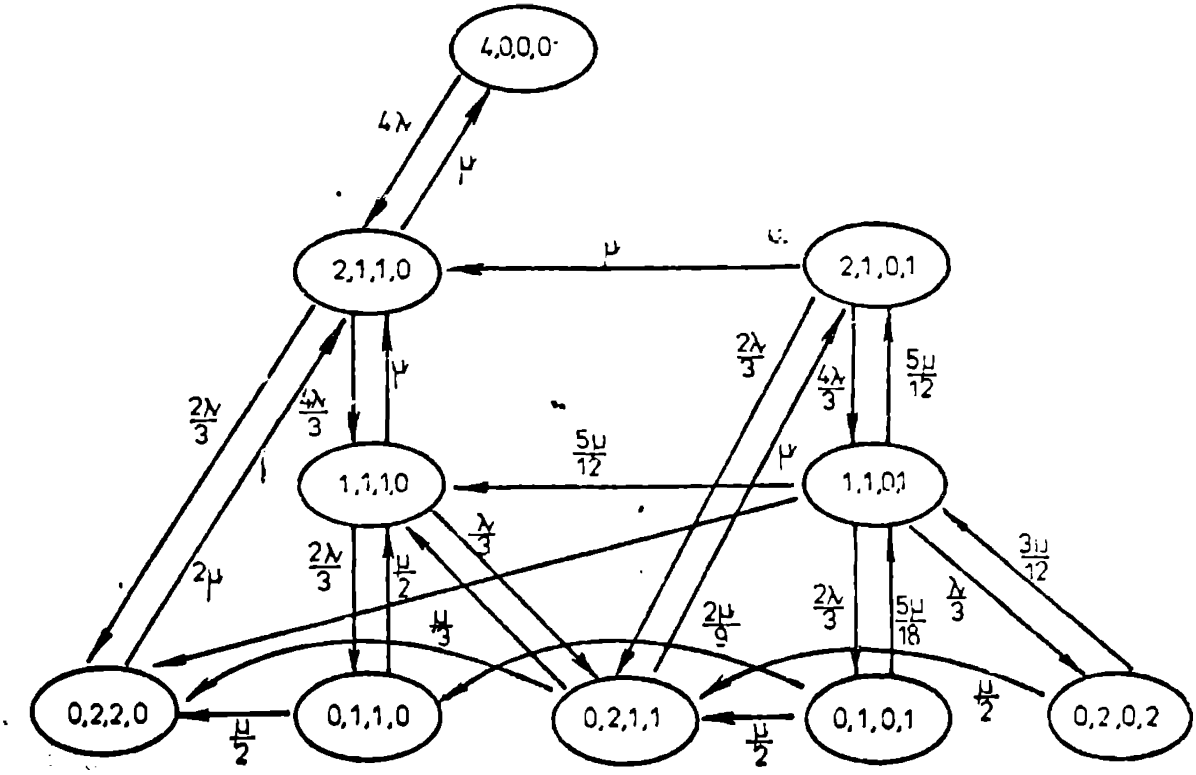


Fig.4.3.6-3

Cazul 6.2.

In multe cazuri nivelul de detaliere al modelului 6.1 (cu diagrama de tranziţie corespunzătoare şi cu calculele laborioase legate de calculul probabilităţilor de tranziţie corespunzătoare) este prea ridicat.

Un model mai aproximativ se poate construi definind starea lanţului Markov astfel:

(n_{34}, n_2, n_{01}) unde $n_{01} = n_0 + n_1$

In acest caz

$s_0 = (p,0,0)$

$s_1 = (p-2,1,1)$

$s_2 = (p-3,1,1)$

$s_3 = (p-4,1,1)$ $\qquad$ $s_p = (p-4,2,2)$

$s_4 = (p-5,1,1)$ $\qquad$ $s_{p+1} = (p-5,2,2)$

$\vdots$ $\qquad$ $\vdots$

$s_{p-1} = (0,1,1)$ $\qquad$ $s_{2p-4} = (0,2,2)$

Lanţul rezultant are deci $2p-3$ stări.

Diagrama de tranziție a stărilor pentru cazul 6.3.

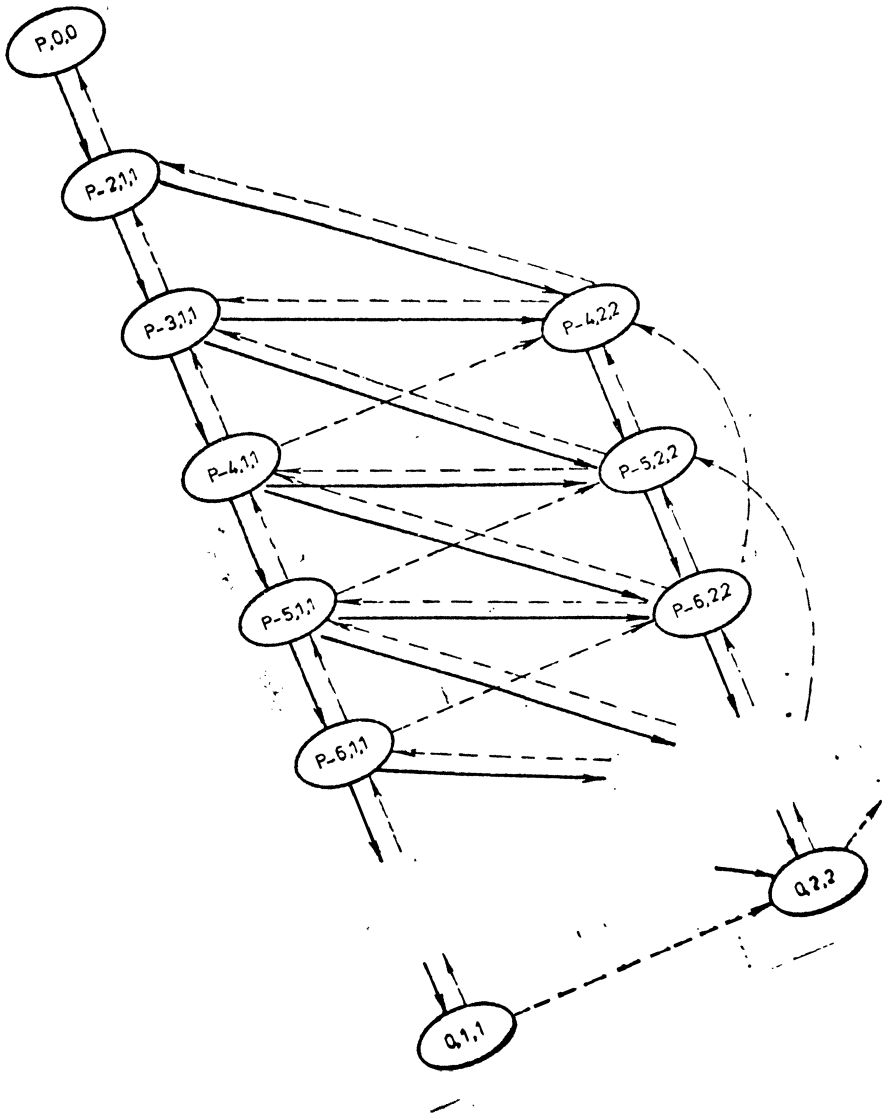


Fig.4.3.6-1

Graful 6.5: (Markov)
Diagrama de tranziţie a stărilor.

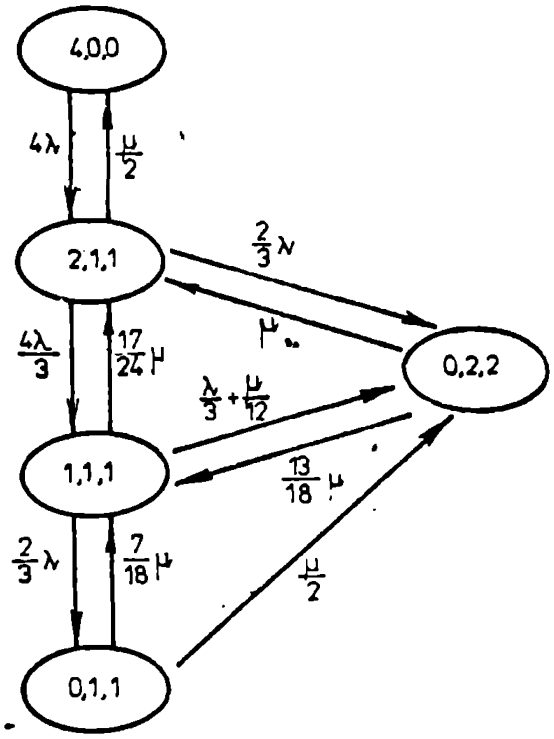


Fig. 4.3.6-5

4.3.6.2 MODELUL CU RE

Tranziţiile din fig. 4.3.6-5 au următoarea semnificaţie:

a_1, a_2, a_3 - selectarea mesajului care va iniţia un nou ciclu;
a_4, a_5, a_6 - generarea timpului cît procesorul este activ, a memoriei cerute şi a timpului de lucru cu memoria ;
a_7, a_8, a_9 - prelucrare;
a_{10}, a_{11}, a_{12} - selectarea cererii mai prioritare;
a_{13}, a_{14}, a_{15} - ataşarea cererii la şirul de aşteptare la MGL1, MGL2 respectiv MGL3;
a_{16}, a_{17}, a_{18} - detaşarea unei cereri din şir;
a_{19}, a_{20}, a_{21} - alocarea MGL1, MGL2 respectiv MGL3;
a_{22}, a_{23}, a_{24} - selectarea traseului mesajului în funcţie de memoria cerută;
a_{25}, a_{26}, a_{27} - lucrul cu MC1, MC2 respectiv MC3;
a_{28}, a_{29}, a_{30} - eliberarea MGL1, MGL2 respectiv MGL3;
a_{31}, a_{32}, a_{33} - selectarea traseului mesajului în funcţie de procesorul care a utilizat MC1, MC2 respectiv MC3;
a_{34} - selectarea mesajului în vederea ataşării la şirul de aşteptare la MGC;

a_{35} - etagarea cererii în şir ;

a_{36} - detagarea unei cereri din şir;

a_{37} - alocarea unei MGG;

a_{38} - selectarea traseului marcajului în funcţie de memoria cerută;

a_{39} - selectarea marcajului care va elibera o MGG;

a_{40} - eliberarea unei MGG;

a_{41} - selectarea traseului marcajului în funcţie de procesorul care a emis cererea ;

Descrierea formală a modelului este:

$L = \{b_1[8], \ldots, b_{47}[8], b_{48}, b_{49}, b_{50}\}$

$P = R = \{r_1, r_2, r_{16}\}$

$A = \{a_1, \ldots, a_{41}\}$

$\xi = \{t, \lambda, \mu\}$

$M_o(b_{45}) = 1$; $M_o(b_{46}) = 1$; $M_o(b_{47}) = 1$

Alocatorul de resurse RM are capacitatea 2.

Procedurile de tranziţie sînt:

$a_1 = (Y(r_1, b_{31}, b_{45}, b_1), (o, o), -)$

Analog a_2 şi a_3

$a_4 = (T(b_4, b_1), o, [T \rightarrow M(b_4(1)) := 1; M(b_4(2)) := DEXP(\lambda); M(b_4(3)) := DEXP(\mu); M(b_4(4)) := FM; M(b_4(7)) := M(b_4(2)) + M(b_4(3))])$

Analog a_5 şi a_6

$a_7 = (T(b_4, b_7), M(b_4(7)), -)$

Analog a_8 şi a_9

$a_{10} = (Y(r_4, b_7, b_{37}, b_{10}), (o, o), [\rightarrow M(b_{10}(3)) := DEXP(\mu)])$

Analog a_{11} şi a_{12}

$a_{13} = (T(b_{10}, Q_1), o, -)$

Analog a_{14} şi a_{15}

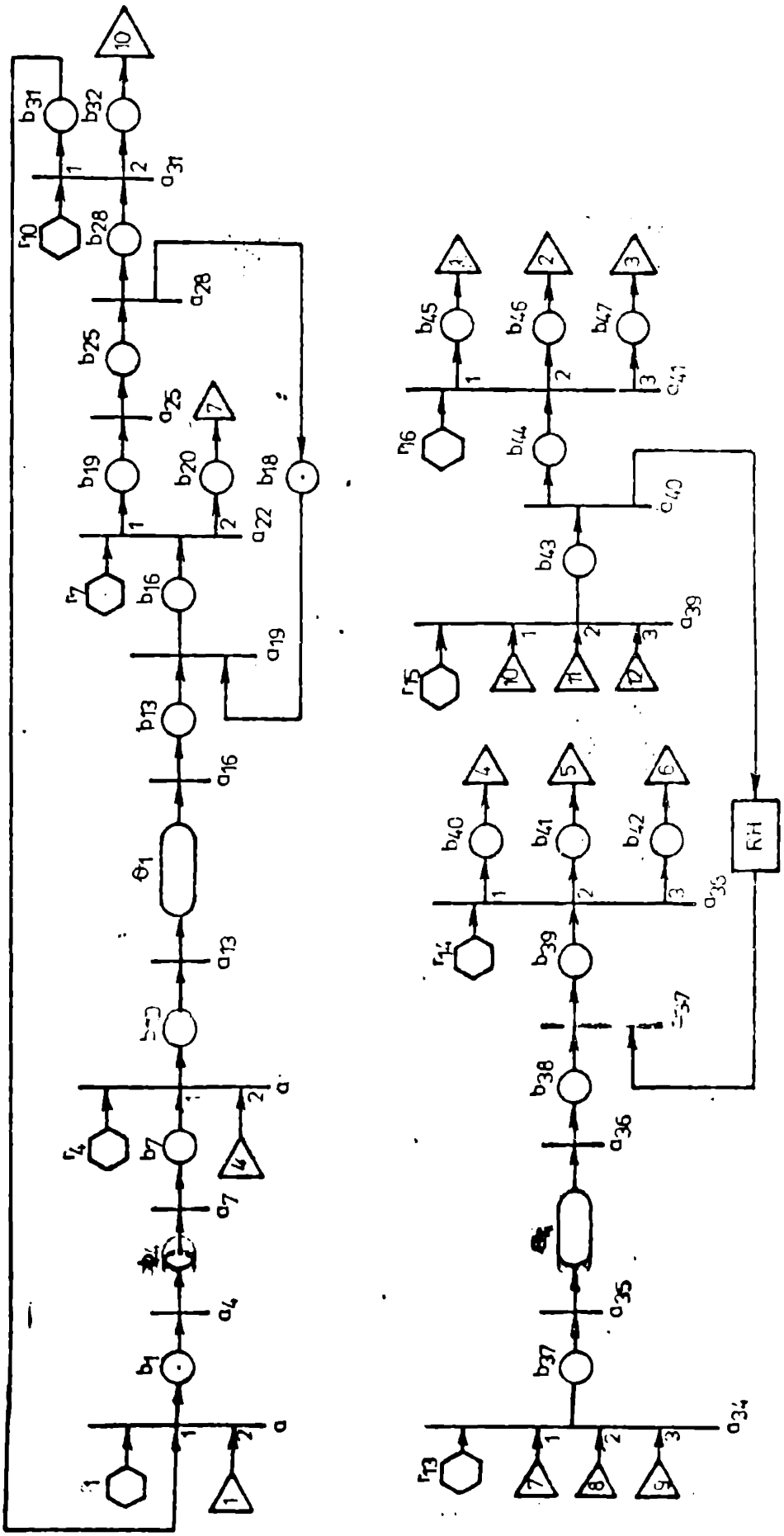
$a_{16} = (T(Q_1, b_{13}), o, -)$

Analog a_{17} şi a_{18}

$a_{19} = (T(b_{13}, b_{48}, b_{16}), o, -)$

Analog a_{20} şi a_{21}

Fig. 4.3.6-6



$a_{22} = (X(r_7, b_{16}, b_{19}, b_{20}), (o,o), -)$

Analog a_{23} şi a_{24}

$a_{25} = (T(b_{19}, b_{25}), M(b_{19}(3)), -)$

Analog a_{26} şi a_{27}

$a_{28} = (F(b_{25}, b_{28}, b_{48}), o, [T \rightarrow M(b_{28}(5)) := M(b_{25}(5)) + M(b_{25}(2));$
$M(b_{28}(6)) := M(b_{28}(5))/t])$

Analog a_{29} şi a_{30}

$a_{31} = (X(r_{10}, b_{28}, b_{31}, b_{32}), (o,o), -)$

Analog a_{32} şi a_{33}

$a_{34} = (Y_3(r_{13}, b_{20}, b_{22}, b_{24}, b_{37}), (o,o,o), -)$

$a_{35} = (T(b_{37}, Q_4), o, -)$

$a_{36} = (T(Q_4, b_{38}), o, -)$

$a_{37} = (J(b_{38}, RH, b_{39}), o, [T \rightarrow M(b_{39}(8)) := 1])$

$a_{38} = (X_3(r_{14}, b_{39}, b_{40}, b_{41}, b_{42}), (o,o,o), -)$

$a_{39} = (Y_3(r_{15}, b_{32}, b_{34}, b_{36}, b_{43}), (o,o,o), -)$

$a_{40} = (F(b_{43}, b_{44}, RH), o, [T \rightarrow M(b_{44}(8)) := o])$

$a_{41} = (X_3(r_{16}, b_{44}, b_{45}, b_{46}, b_{47}), (o,o,o), -)$

Procedurile de rezoluţie sînt :

$r_1 : [T \rightarrow M(r_1) := 1]$

Analog r_2, r_3.

$r_4 = [T \rightarrow M(r_4) := 2]$

Analog r_5, r_6.

$r_7 : [(M(b_{16}(4)) = M(b_{16}(1))) \rightarrow (M(b_{16}(8)) = 1)) \rightarrow M(r_7) := 1 ;$
$T \rightarrow M(r_7) := 2]$

Analog r_8, r_9.

$r_{10}: [(M(b_{28}(4)) = M(b_{28}(1)) \rightarrow M(r_{10}) := 1; \quad T \rightarrow M(r_{10}) := 2]$

Analog r_{11}, r_{12}.

$r_{13}: [T \rightarrow M(r_{13}) := 1]$

$r_{14}: [(M(b_{39}(4)) = 1) \rightarrow M(r_{14}) := 1; (M(b_{39}(4)) = 2) \rightarrow M(r_{14}) := 2;$
$T \rightarrow M(r_{14}) := 3]$

$r_{15}: [T \rightarrow M(r_{15}) := 1]$

r_{16}: [$(M(b_{44}(1))=1) \rightarrow M(r_{16}):=1; (M(b_{44}(1))=2) \rightarrow M(r_{16}) := 2$;
$T \rightarrow M(r_{16}) := 3$]

4.3.6.3 SCHEMA LOGICA A PROGRAMULUI GPSS

Semnificația facilităților utilizate este următoarea: facilitatea 1 reprezintă MC1, 2-MC2, 3-MC3, 4-MGL1, 5-MGL2, 6-MGL3. Storaje-ul 1 reprezintă cele două MGG. Sirurile de așteptare 1,2,3,4 se formează la MGL1, MGL2, MGL3, respectiv MGG. Funcția 2 reprezintă memoria cerută. Variabila 8 reprezintă P4+3 (MGL necesară).

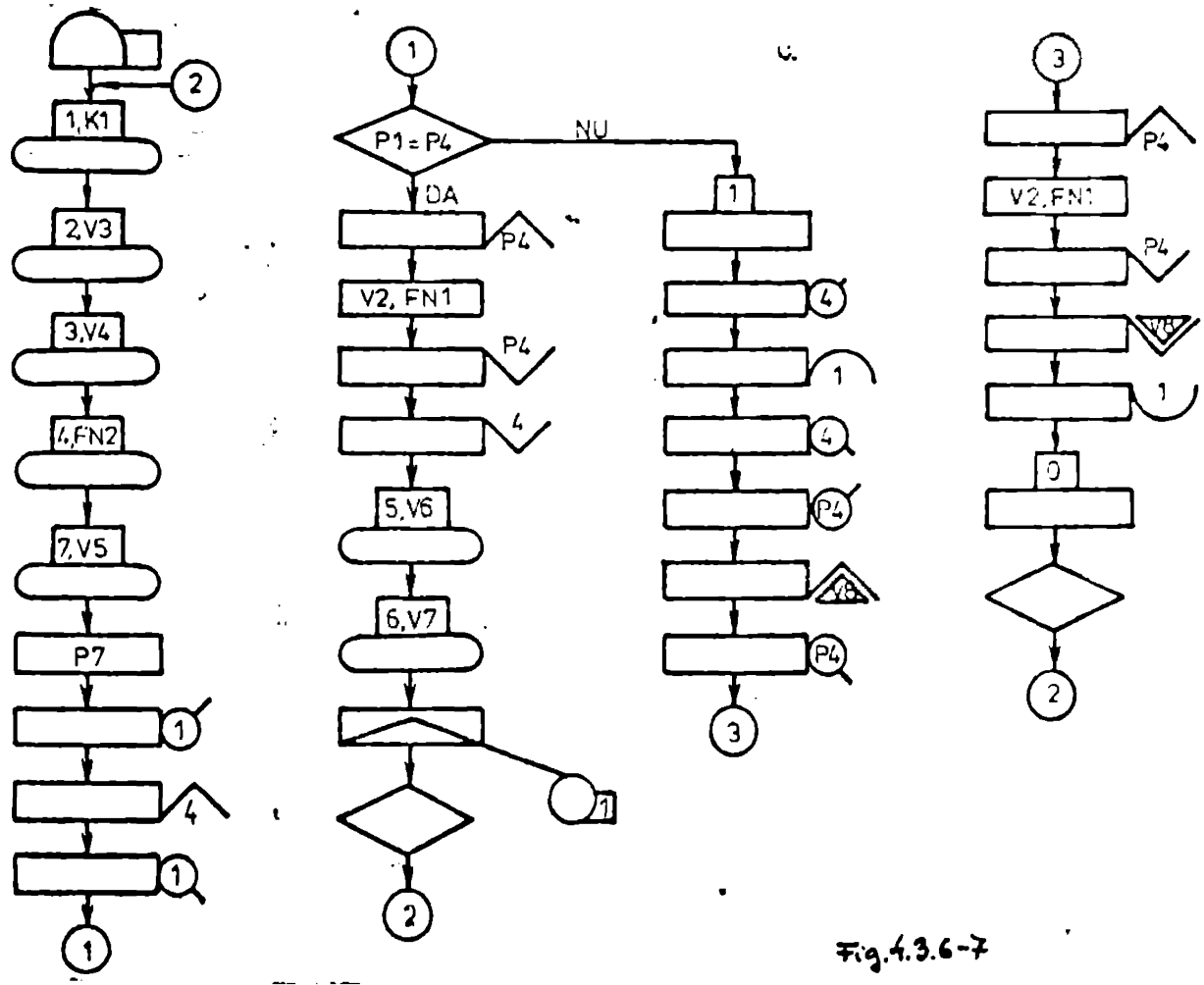


Fig.4.3.6-7

4.3.7 ARHITECTURA 7. (pxpx2)

Această arhitectură constituie o variantă mai performantă a arhitecturii 6.

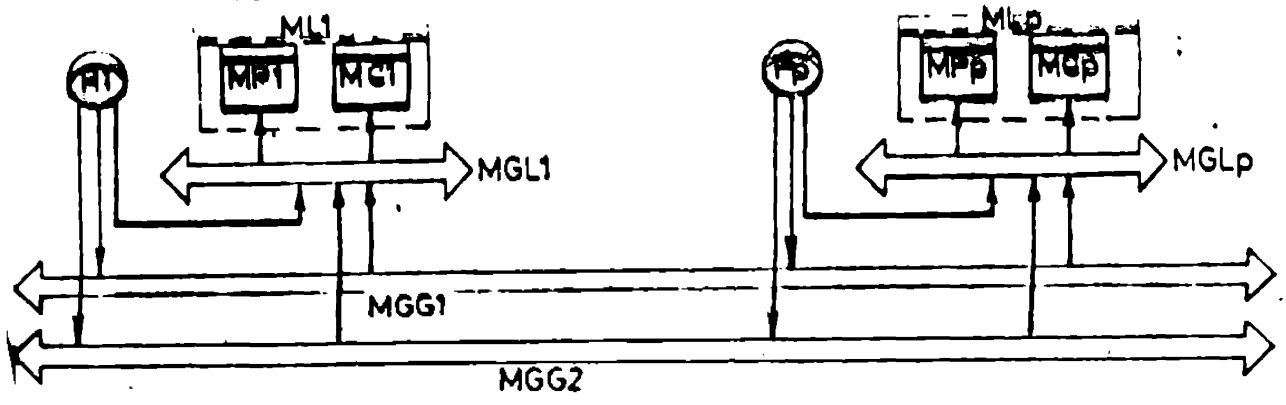


Fig 4.3.4.7-1

i) Se evită ca un procesor (de exemplu procesorul 1) să intre într-o stare în care este împiedicat să lucreze cu o MC externă, de către un procesor extern care utilizează MGI_1.

ii) Se consideră că cererile venind de la MCG pot întrerupe orice procesor (care trece în starea blocat).

iii) [illegible] ca în cazul arhitecturii 6.

iv) Strategia de deservire este FIFO

v) $\frac{1}{\lambda} = \frac{1}{\lambda_p} + \frac{1}{\mu}$ sau $\lambda = \frac{\lambda_p \mu}{\lambda_{p+\mu}}$ (4.3.7-1)

Lanţul Markov exact care modelează comportarea sistemului are starea definită ca în cazul arhitecturii 6. Dispare automat situaţia ca un procesor să treacă în coada de aşteptare pentru o resursă externă fiindcă magistrala să locală este ocupată.

4.3.7.1 MODELUL APROXIMATIV 7.1 (pxpx2)

Acest model este construit ca în cazul 5.1

Starea va fi deci descrisă de tripletul

(n_{34}, n_2, n_o)

$p - (n_{34} + n_2 + n_o)$ procesoare vor sta în coadă pentru resurse ocupate.

Stările lanţului Markov corespunzător vor fi:

$s_o = (p, o, o)$

$s_1 = (p-2, 1,1)$

$s_2 = (p-2, 2,o)$

$s_3 = (p-3, 1,1)$

$s_4 = (p-3, 2,1)$

$s_5 = (p-3, 2,o)$

$s_6 = (p-4, 2,2)$

$s_7 = (p-5, 2,2)$

$\vdots$

$s_{p+2} = (o, 2,2)$

$s_{p+3} = (p-4, 1, 1)$

$s_p = (p-5, 1, 1)$

$\vdots$

$s_{2p-1} = (o, 1, 1)$

$s_{2p} = (p-4, 2, 1)$

$s_{2p+1} = (p-5, 2, 1)$

$\vdots$

$s_{3p-4} = (o,2,1)$

$s_{3p-3} = (p-4, 2\ o)$

$s_{3p-4} = (p-5, 2\ o)$

$\vdots$

$s_{4p-7} = (o,2\ o)$

Lanțul conține deci 4 p-6 stări.

Diagrama de tranziție a stărilor.

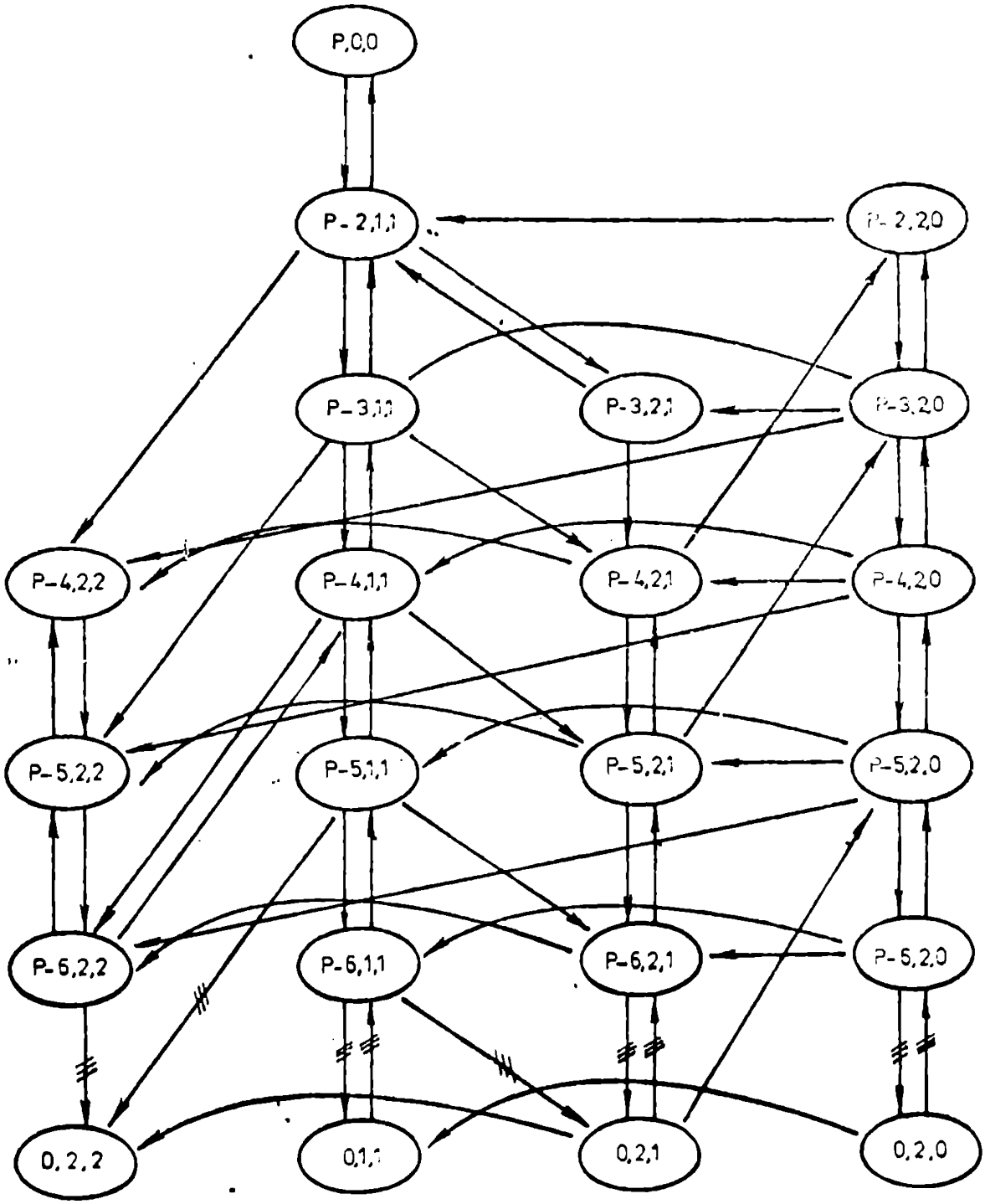


Fig. 4.3.4-2

4.3.7.2 CAZUL 7 (4=4+0)

Modelul aproximativ.

Diagrama de tranziţie a stărilor

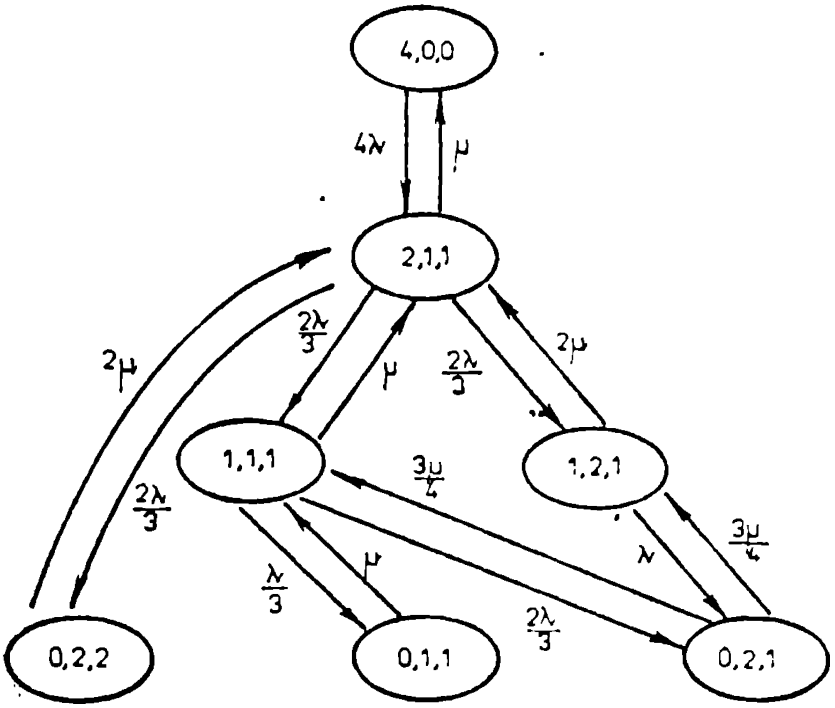


Fig. 4.3.7-3

4.3.7.3 MODELUL CU RE

Modelul cu RE din acest caz trebuie să conţină pe lîngă ceea ce se vede în fig. 4.3.7-4 încă două porţiuni identice pentru p_2 şi p_3, precum şi porţiunea de alocare şi eliberare a unei MGG aşa cum apare în modelul precedent.

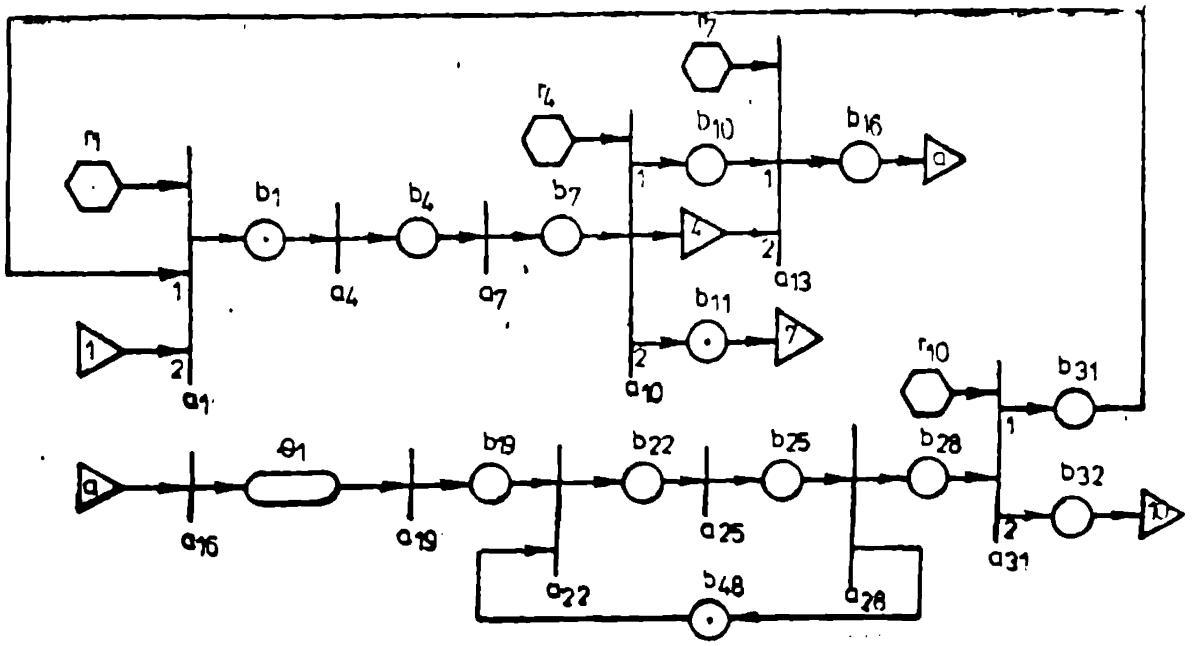


Fig. 4.3.7-4

Tranzitiile din fig. 4.37-4 au următoarea semnificație:

a_1, ... ,a_9 analog cu cazul precedent;

a_{10},a_{11},a_{12} analog cu a_{22}, a_{23}, a_{24} din cazul precedent;

a_{13},a_{14},a_{15} analog cu a_{10}, a_{11}, a_{12} din cazul precedent;

a_{16}, ...,a_{24}analog cu a_{13}, ... , a_{21} din cazul precedent;

a_{25}, ...,a_{33}analog cu cazul precedent.

Descrierea formală a modelului este identică cu cea din cazul precedent.

Procedurile de tranziție sînt analoage cu cele din cazul precedent conform specificațiilor de mai sus.

Procedurile de tranziție sînt:

r_1 : [T → M(r_1) : = 1]

analog r_2, r_3

r_4 : [(M (b_7(4)) = 1)→M(r_4): = 1; T→M(r_4): = 2]

analog r_5, r_6.

r_7 : [T→M (r_7) : = 2]

analog r_8, r_9.

r_{10} : [(M (b_{28}(1)) = M(b_{28}(4)) → M(r_{10}) : = 1; T→M(r_{10}) : = 2]

analog r_{11}, r_{12}.

4.3.7.4 Schema logică a programului GPSS

Facilitățile din acest program au următoarea semnificație: facilitatea 1 reprezintă MC1, 2 - MC2, 3 - MC3, 4 - MGL1, 5 - MGL2, 6 - MGL3. Sorage-ul 1 reprezintă cele două MGC. Funcția 2 reprezintă memoria cerută. Variabila 8 calculată cu p 4 + 3, reprezintă [illegible] MGL necesară.

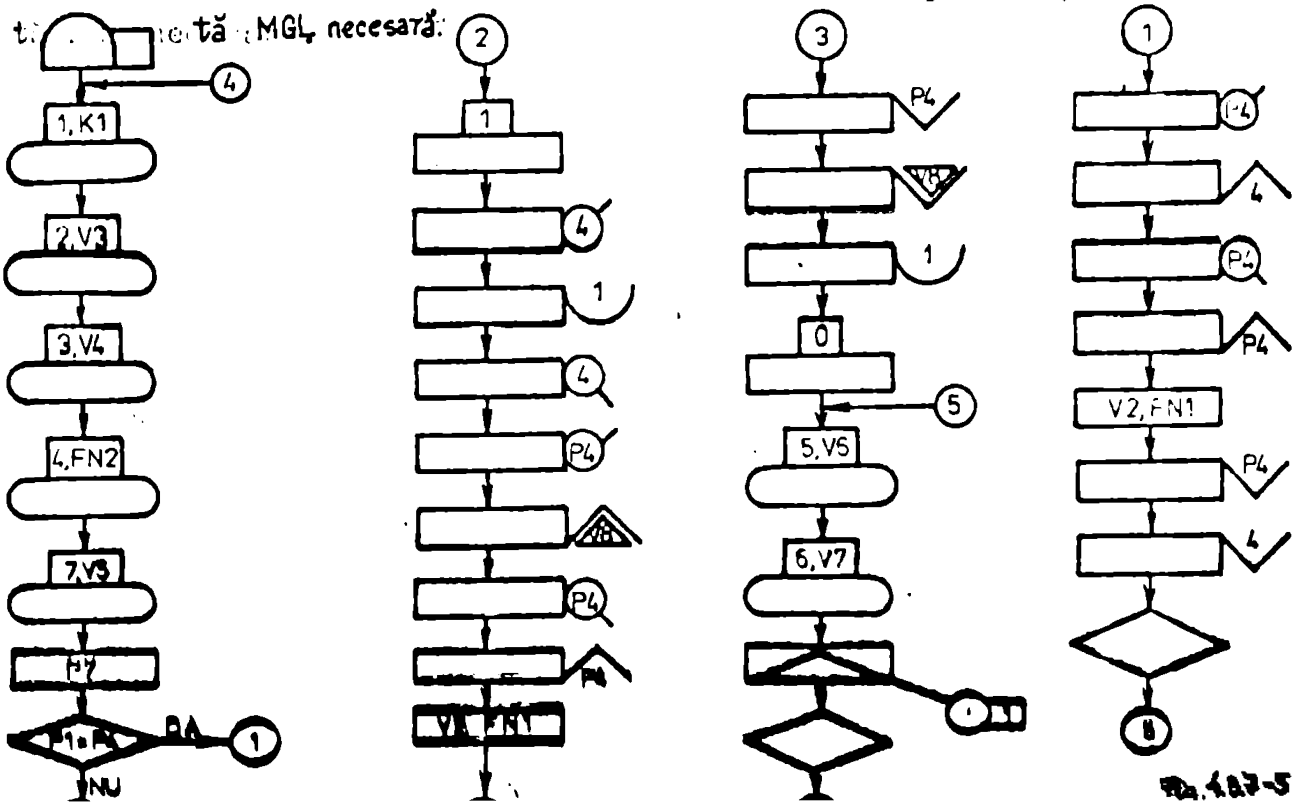


Fig.4.37-5

4.3.8 ARHITECTURA 6 (pxpx2)

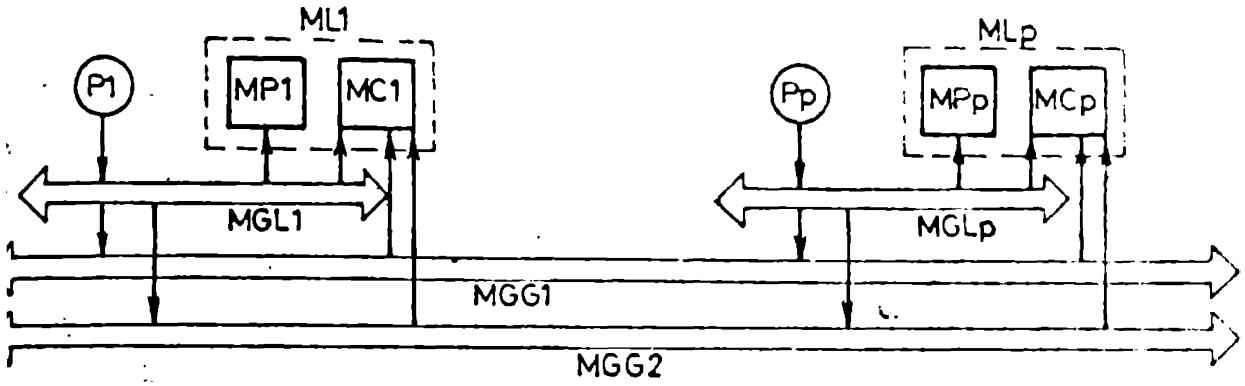


Fig 4.3.8-1

i) Se presupune în acest caz că se folosesc memorii cu mai multe porturi care permit mai multe accese simultane. În acest caz ele sînt accesibile direct de la MGG.

ii) Singurele surse de conflict între procesoare le constituie cele 2 MGG.

iii) Mesajele sînt schimbate între procesoare ca în cazul arhitecturii 6.

iv) $$\frac{1}{\lambda} = \frac{1}{\lambda_p} + \frac{1}{\mu} \qquad (4.3.8-1)$$

4.3.8.1 Model cu SA

Această arhitectură poate fi studiată utilizînd ca în cazul arhitecturii 5 un sistem de cozi de așteptare de tipul M/M/2/P/P

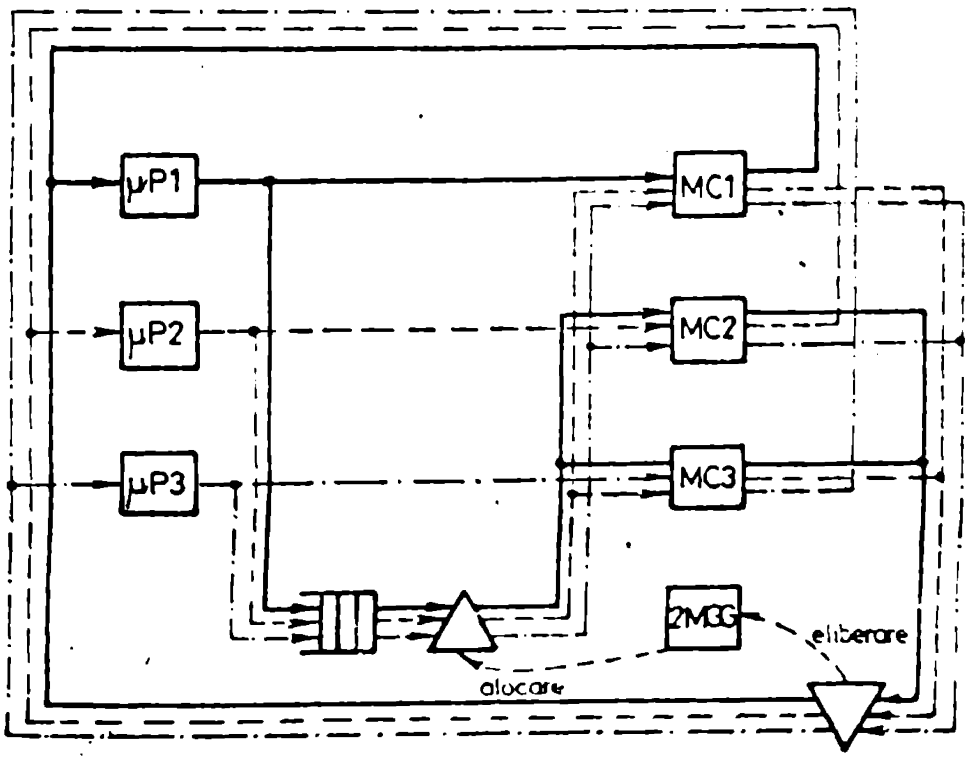


Fig. 4.3.8-2

4.3.8.2 MODELUL CU RE

Porţiunea de model din fig. 4.3.8-3 trebuie completată ca şi în cazul precedent pentru a obţine modelul complet al arhitecturii 8.

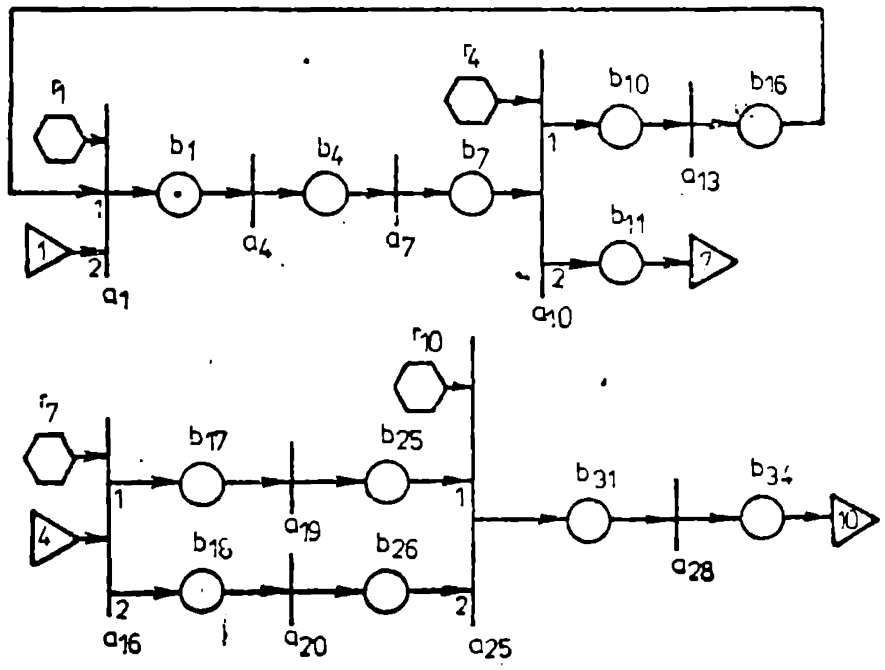


Fig. 4.3.8-3

Tranziţiile din fig. 4.3.8.3 au următoarea semnificaţie:

$a_1, \ldots, a_9$ - analog cu cazul 6;

a_{10}, a_{11}, a_{12} - analog cu a_{22}, a_{23}, a_{24} din cazul 6;

a_{13}, a_{14}, a_{15} - lucrul cu MC1;

a_{16} - selectarea traseului în funcţie de memoria cerută;

a_{19}, a_{21}, a_{23} - lucrul cu MC2;

a_{20}, a_{22}, a_{24} - lucrul cu MC3;

a_{25}, a_{26}, a_{27} - selectarea marcajului care va elibera o MC;

a_{28}, a_{29}, a_{30} - calculul atributelor 5 şi 6.

Descrierea formală a modelului este:

$$L = \{ b_1 [7], \ldots, b_{47} [7] \}$$

$$P = L = \{r_1, \ldots, r_{16}\}$$

$$A = \{a_1, \ldots, a_{30}, a_{34}, \ldots, a_{41}\}$$

$$\xi = \{t, \lambda, \mu\}$$

$M_o(b_1) = 1$; $M_o(b_2) = 1$; $M_o(b_3) = 1$.

Procedurile de tranziţie sînt :

$a_1, \ldots, a_6$ sînt identice cu cele din cazul arhitecturii 6.

$a_7 = (T(b_4, b_7), M(b_4(7)), [T \rightarrow M(b_7(3)) := DINP(\mu)])$

analog a_1, a_9.

$a_{10} = (X(r_4, b_7, b_{10}, b_{11}), (o,c), -)$

analog a_{11}, a_{12}.

$a_{13} = (T(b_{10}, b_{16}), M(b_{10}(3)); [T \rightarrow M(b_{16}(5)) := M(b_{16}(5)) + M(b_{16}(2)); M(b_{16}(6)) := M(b_{16}(5))/t])$

analog a_{14}, a_{15}.

$a_{16} = (X(r_7, b_{40}, b_{17}, b_{18}), (o,o), -)$

analog a_{17}, a_{18}.

$a_{19} = (T(b_{17}, b_{25}), M(b_{17}(3)), -)$

analog a_{21}, a_{23}.

$a_{20} = (T(b_{18}, b_{26}), M(b_{18}(3)), -)$

analog a_{22}, a_{24}.

$a_{25} = (Y(r_{10}, b_{25}, b_{26}, b_{21}), (o,o), -)$

analog a_{26}, a_{27}.

$a_{28} = (T(b_{31}, b_{34}), o, [T \rightarrow M(b_{34}(5)) := M(b_{31}(5)) + M(b_{31}(2)); M(b_{34}(6)) := M(b_{34}(5))/t])$

analog a_{29}, a_{30}.

Procedurile de rezoluţie sînt :

$r_1, \ldots, r_6$ - analog ca în cazul arhitecturii 6.

$r_7 = [(M(b_{20}(4)) = 2) \rightarrow M(r_7) := 1; T \rightarrow M(r_7) := 2]$

analog r_8, r_9.

$r_{10} = [T \rightarrow M(r_{10}) := 1]$

analog r_{11}, r_{12}.

4.3.8.3 SCHEMA LOGICĂ A PROGRAMULUI GPSS

In acest program memoriile 1, 2, 3, 4 reprezintă MC1, MC2, MC3 (cu capacitatea 3 - fiind memorii cu 3 căi de acces)

respectiv celo două MGC. Sirul de aşteptare 1 se formează la MGC.

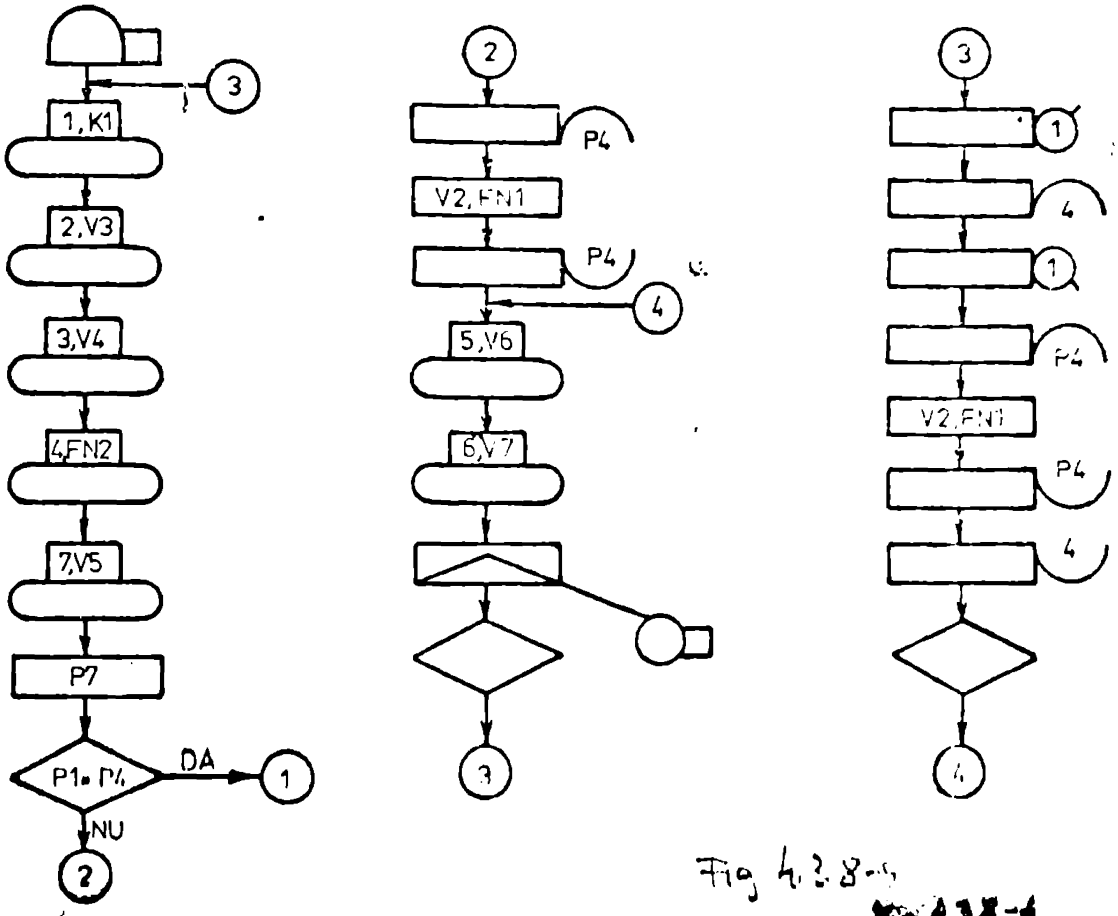


Fig 4.3.8-5

Fig.4.38-4

4.3.9 ARHITECTURA 3 (paralel) p

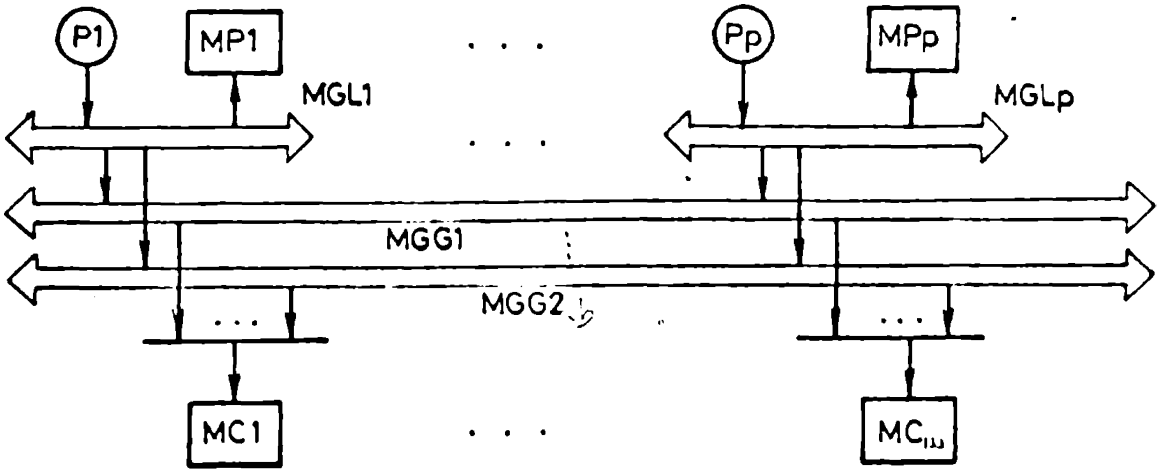


Fig 4.3.9-1 Fig.4.39-1

i) Procesoarele şi MC sînt conectate printr-un set de b MGC.

ii) Fiecare MGC poate conecta orice procesor la orice MC.

iii) Fiecare procesor are acces exclusiv la o MC.

iv) Schimbul de mesaje are loc printr-o cutie MC în care procesorul expeditor scrie şi din care procesorul destinatar obţine cînd condiţiile de rutare vor fi satisfăcute.

v) Conflicte pot apărea atît datorită MGC partajate cît şi a MC partajate.

Model cu SA

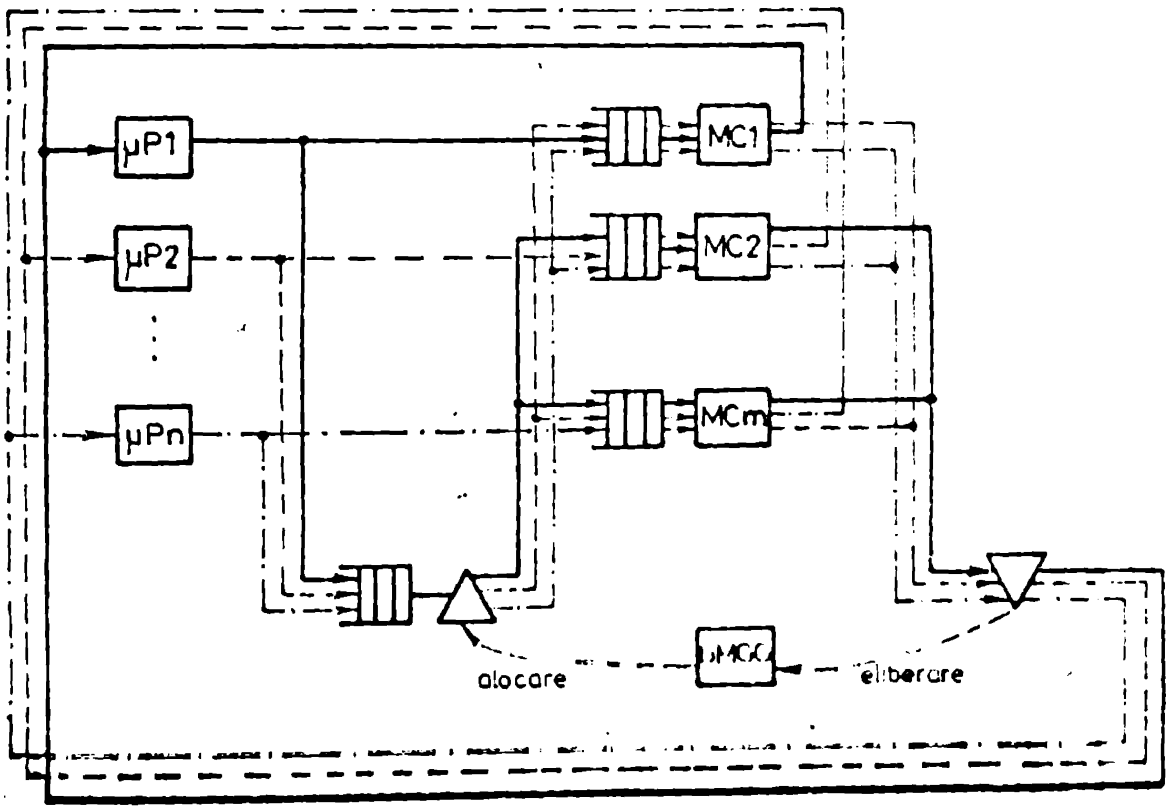


Fig 4.39-2

4.39.1 MODELUL APROXIMATIV 9.1 (pxmxb) $p \geq m > b$.

Starea sistemului este reprezentată de perechea

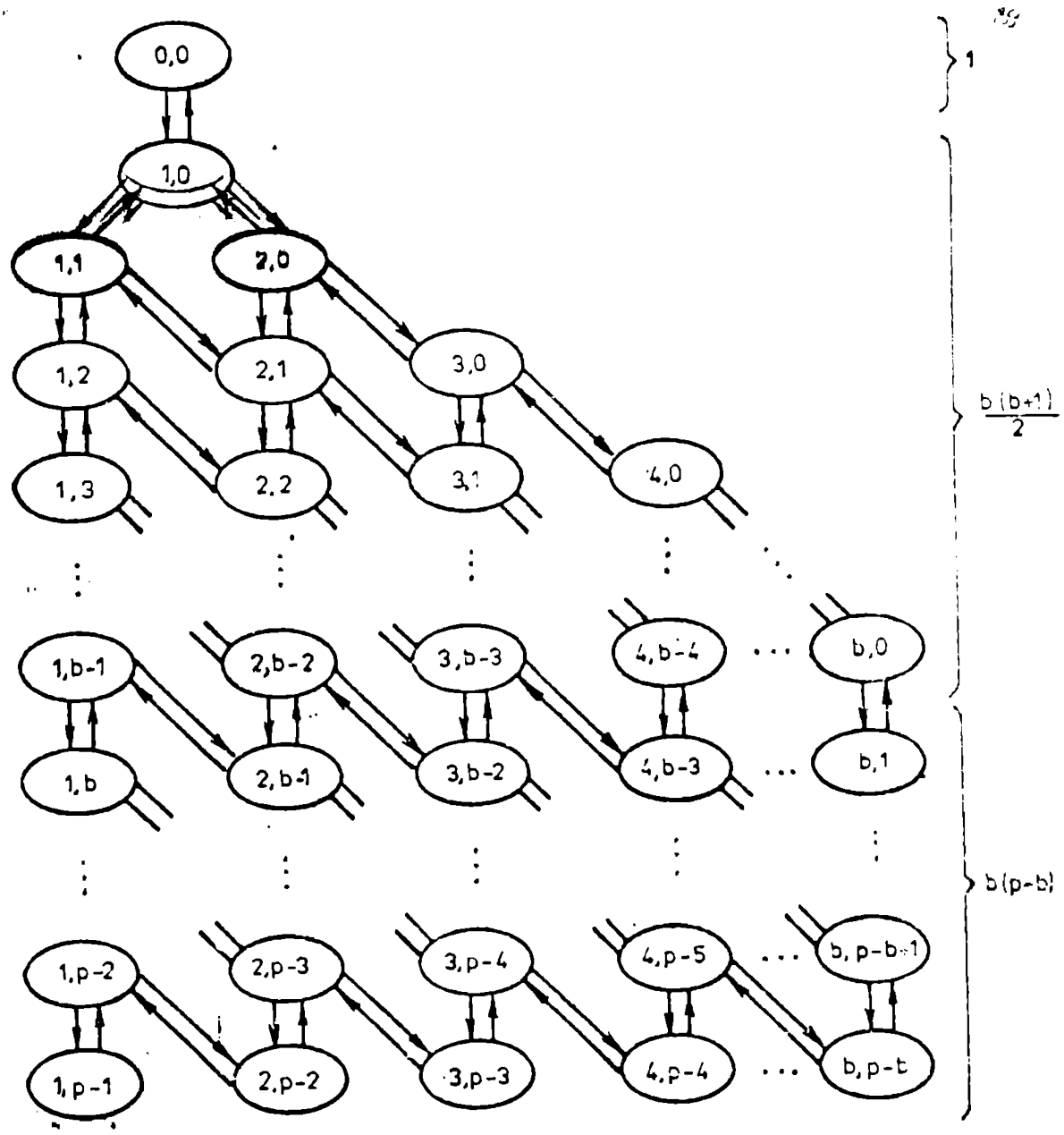
(n_2, n_1)

unde n_2 = numărul procesoarelor care lucrează o memorie externă comună

n_1 = numărul procesoarelor în coadă pentru o resursă

evident $p = n_{34} + n_2 n_1$

Modelul rezultat va fi un lanț Markov comasat având următoarea diagramă de tranziție a stărilor



Numărul total de stări este:

$$N = 1 + \frac{b(b+1)}{2} + b(p-b) = 1 + b\left[p + \frac{1}{2}(1-b)\right]$$

Fig. 4.3.9-3

Frecvenţele de tranziţie sînt evaluate presupunînd că ficcare procesor activ poate solicita orice memorie cu aceiaşi probabilitate. In plus se presupune că fiecare procesor din coadă poate solicita orice MC, curent ocupată, cu aceiaşi probabilitate (cînd o magistrală devine liberă).

Presupunînd că sîntem în starea (i; j), pot apărea tranziţii în cel mult patru stări vecine:

(i+1,j), (i-1,j), (i,j+1), (i,j-1)

tranziţii notate respectiv

$i \to i+1$, $i \to i-1$, $j \to j+1$, $j \to j-1$

Frecvenţele de apariţie a acestor tranziţii sînt:

$$F(i \to i+1) = (p-i-j)\lambda \frac{m-i}{m} \qquad \begin{matrix} 0 \leq i < b \\ p-1-j > 0 \end{matrix}$$

$$F(i \to i-1) = \begin{cases} i\mu \left[\frac{i-1}{i}\right]^j & i < b \\ b\mu \left[\frac{b-1}{m}\right]^j & i = b \end{cases}$$

$$F(j \to j+1) = \begin{cases} (p-i-j)\frac{i}{m}\lambda & i<b,\ p-i-j>0 \\ (p-b-j)\lambda & i=b,\ p-b-j>0 \end{cases}$$

$$F(j \to j-1) = i\mu - F(i \to i-1)$$

4.3.9.2 MODELUL 9.2. (pxmxb) $p \geq m > b$

Starea lanţului Markov este definită ca în cazul modelului 9.1.

Starea lanţului Markov comasat poate fi definită pornind de la un lanţ Markov mai precis cu starea:

$(n_2, pr1, pr2, \ldots prb, pr\ b+1, pr\ b+2, \ldots, pr\ m)$

unde

pr1, pr2,...,prb sînt numerele de ordine a procesoarelor care lucrează cu o MC aranjate în ordine descrescătoare;

pr b+1, pr b+2,... , pr m sînt numerele de ordine a procesoarelor în coadă pentru o MC inaccesibilă din lipsa unei MC, aranjate în ordine descrescătoare.

In acest caz metoda de evaluare a frecvenţelor de tranziţie presupune că se ştie numărul de stări comasate într-o macrostare. Se adună toate frecvenţele de tranziţie din stările comasate către fiecare stare vecină şi se defineşte frecvenţa de tranziţie ca fiind raportul dintre suma frecvenţelor şi numărul de stări comasate.

Dacă se comasează stările $\{s_i\}$, i = 1,. . .,L în macrostarea X_1 stările $\{t_j\}$, j=1, . . . M în macrostarea X_2, şi dacă frecvenţele de tranziţie din s_i în t_j ale lanţului iniţial,sînt $r_{si,tj}$ atunci frecvenţa de tranziţie din X_1 în X_2 este dată de

$$F_{x1,x2} = \frac{1}{L} \sum_{i=1}^{L} \sum_{j=1}^{M} r_{si,tj}$$

In cazul în care stările comasate au probabilităţi staţionare legale nu se comite nici o eroare. In caz contrat noile frecvenţe sînt aproximative.

Această metodă este dificil de aplicat în cazul general pxmxb deoarece numărul de stări ale lanţului iniţial este din ce în ce mai mare.

Notăm cu ℓ suma dintre numărul de procesoare,lucrînd cu o MC,şi numărul de procesoare din coadă. Se observă că ℓ este în acelaşi timp diferenţa dintre numărul total de procesoare şi numărul procesoarelor active.

$\ell = o$ apare o singură stare ;

$\ell = 1$ apare o singură stare ;

$\ell = 2$ apar două stări (una cu un procesor lucrînd cu MC [illegible]),

$\ell \geq 3$ $n_0 = 1$ apare o stare,

Se ştie că numărul partiţiilor neordonate a unei mulţimi de n elemente în k părţi, k,n∈Z este:

$$c_k(n) = c_k(n-k) + c_{k-1}(n-k) + \ldots + c_1(n-k) + c_o(n-k)$$

unde $c_k(n) = o \quad n<k, k<o$

$c_o(n) = o \quad n>o$

$c_k(k) = 1 \quad k \geq o$

Numărul stărilor cu $n_2 = i$, $i<b$, $\ell \geq 3$ este $p_1(\ell)$ Din acestea $P_{i-k}(\ell-i)$ au k cozi, la memorie, vide.

Numărul stărilor dintr-un SMM (pxmxb) cu $\ell = k+b$, $\ell > b+1$, $n_m = b$, este:

$$\sum_{j=o}^{k} [c_b(j+b) c_{m-b}(k-j+m-b)] \quad k \leq p-b$$

Din acestea $c_b(\ell)$ stări nu au nici un procesor în coadă pentru MC,în scopul de a lucra cu o MC.

iar $c_{b-j}(\ell-b)$ este numărul stărilor fără coadă la MC şi fără coadă la $j<b$ din memorii.

Numărul stărilor în care există procesoare în coadă - pentru MCG este:

$$\sum_{j=0}^{k-1} \left[c_b(j+b) c_{m-b}(k-j+m-b) \right] \quad , \; k \leq p-b$$

Numărul stărilor ce au procesoare în coadă pentru o MCG şi cu cel puţin o coadă vidă la ne 8eupate este:

$$\sum_{j=0}^{k=1} c_{b-1}(j+b-1) \qquad k \leq p-b$$

Frecvenţele de tranziţie din cazul 9.1 rămîn neschimbate cu o singură excepţie

$$F(i \to i-1) = \begin{cases} \dfrac{\sum_{n=0}^{i} n \mu \, c_i - n(0j)}{c_i \; (i+j)} & i < b \\ \dfrac{\sum_{n=0}^{b} \cdot n \mu \, c_{b-n}(i+j-b)}{\sum_{n=0}^{\ell-b} c_b \; (n+b) c_{m-b}(\ell-n+m-2b)} & i=b \end{cases}$$

4.3.9.3 MODELUL 9.3 (pxmxb) $p \geq m > b$

Modelele precedente deşi foarte generale conduc la calcule analitice deosebit de laborioase. Din acest motiv am studiat şi cel mai simplu model în care starea este un număr care reprezintă numărul procesoarelor active. Nu se mai ţine o evidenţă a stării cozilor interne. Frecvenţele de tranziţie au fost calculate utilizînd tehnica de mediere propusă în cadrul modelului 9.2.

Diagrama de tranziţie a stărilo:

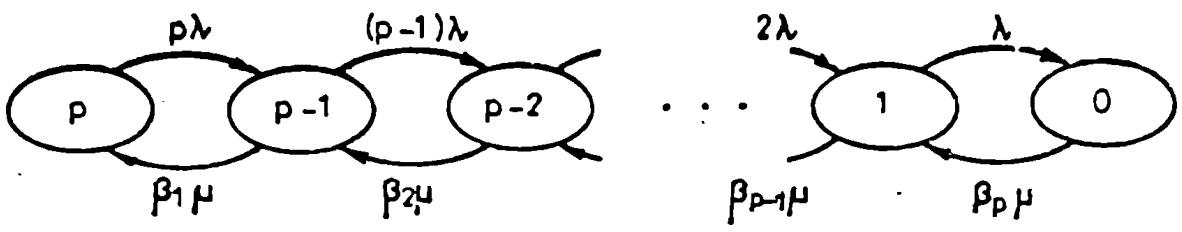


Fig. 4.3.9-4

Dacă notăm cu $\tilde{\pi}(i)$ probabilitatea staționară a stării i atunci:

$$\tilde{\pi}(i) = \left[\frac{\lambda}{\mu}\right]^{p-i} \cdot \frac{p!}{i!} \prod_{k=1}^{p-i} \beta_k^{-1} \tilde{\pi}(p)$$

$$\tilde{\pi}(p) = \left\{ 1 + \sum_{j=o}^{p-1} \left[\left(\frac{\lambda}{\mu}\right)^{p-j} \cdot \frac{p!}{j!} \prod_{k=1}^{p-j} \beta_k^{-1} \right] \right\}^{-1}$$

unde

$$\beta_i = \frac{\sum_{j=1}^{b-1} j p_j(i) + b \sum_{j=o}^{i-b} \left[p_b(j+b) p_{m-b}(i-2b-j+m) \right]}{\sum_{j=1}^{b-i} p_j(i) + \sum_{j=o}^{i-b} \left[p_b(j+b) p_{m-b}(i-2b-j+m) \right]} \qquad i \geq 1$$

β_i este raportul dintre suma frecvențelor de tranziție din toate stările cu p-i procesoare active în stările cu p-i+1 procesoare active și numărul stărilor cu p-i procesoare active.

Puterea de prelucrare va fi:

$$P = \sum_{i=1}^{p} \tilde{\pi}_i = \sum_{i=1}^{p} \frac{\left(\left[\frac{\lambda}{u}\right]^{p-i} \frac{p!}{(i-1)!} \prod_{k=1}^{p-i} \beta_k^{-1} \right)}{1 + \sum_{j=o}^{p-i} \left[\left(\frac{\lambda}{u}\right)^{p-j} \frac{p!}{j!} \prod_{k=1}^{p-j} \beta_k^{-1} \right]}$$

4.4. ANALIZA COMPARATIVĂ A REZULTATELOR SIMULĂRII

Rezultatele prezentate în continuare sînt obținute pe baza modelelor analitice și a celor de simulare GPSS. Dat fiind complexitatea mare a calculelor analitice este dificil ca ele să fie utilizate pentru modele mai complexe. Simularea oferă la acest nivel o alternativă viabilă pentru estimarea performanțelor SM propuse.

Din multitudinea parametrilor standard oferiți de protocoalele GPSS s-au utilizat numai aceia care au permis calculul numărului mediu de procesoare active la o încărcare dată.

Estimările obținute, deși trebuie privite cu o anumită prudență datorată faptului că ele neglijează o mare parte a pierderilor de performanță datorate problemelor de sincronizare între procese și/sau procesoare. Pe de altă parte s-a menținut ipoteza unor încărcări între o și 1 deci relativ mici.

In ceea ce privește prețul de cost s-a presupus pur și simplu că el este o funcție liniară în care valoarea inițială depinde de costul magistralelor și a altor prețuri independente de numărul procesoarelor; panta depinzînd exclusiv de numărul de procesoare. Arhitecturile care depășesc ca număr lo - 2o de procesoare nu mai satisfac ipotezele de modelare ale intrînd în clasa SMM cu prelucrare masivă pentru care atît posibilitățile cît și prețul sînt mai mari.

Rezultatele numerice obținute, în ciuda ipotezelor simplificatoare, sînt în acord cu performanțele așteptate intuitiv de proiectant. Ele permit o judecată onestă asupra diverselor arhitecturi pentru diverse încărcări, și număr total de procesoare și magistrale.

Toate rezultatele tabelate și graficele sînt obținute cu ajutorul simulării GPSS sau cu ajutorul modelelor analitice (acolo unde a fost posibil și rezonabil).

i) Cazul b = 1, p = 2 ilustrat în figurile
cît și în tabelele ilustrează

Arhitectura 3 (A3) este superioară arhitecturii 4 (A4) care la rîndul ei e superioară arhitecturilor A1 și A2. Arhitecturile 1 și 2 se comportă astfel: pentru încărcări ușoare A1 e superioară, A2 (surprinzător căci A1 generează în medie mai multe apelări ale MGG). Rezultatul se datorează faptului că la încărcări mici, întîrzierile introduse de coadă sînt și ele mici, conflictele putînd fi deci neglijate. In cazul A2 fiecare acces la o memorie externă blochează un procesor, a cărui probabilitate de a fi activ pe magistrala sa locală este foarte mare în cazul încărcărilor mici. Performanțele celor două arhitecturi sînt similare pentru $S_p \approx 0,5$. Pentru încărcări mai mari A2 devine mai avantajoasă. In scopul comparării se vor utiliza însă mai ales încărcările mici, căci un SMM bine proiectat lucrează în această zonă a caracteristicilor sale. Descompunerea sarcinilor procesoarelor în procese, alocarea lor corectă și sincronizarea judicioasă au ca scop ultim tocmai reducerea încărcării SMM.

Pentru încărcări deosebit de mici performanțele arhitecturilor sînt aproape identice și A1 nu pare a fi o alegere rea dat fiind simplitatea ei. Pentru sisteme cu încărcări mari utilizarea unor memorii locale comune separate de memoriile proprii se justifică numai în cazul memoriilor cu două porturi.

Rezultatele indicate de fig.2,3,4 pentru p=2, 5, 1o confirmă aprecierile anterioare. La creșterea numărului de procesoare performanțele arhitecturilor A2, A3 și A4 devin similare

şi tind să devină identice. A3 nu mai este net avantajoasă faţă de A2 şi A4, lucru care pentru încărcări mari se explică intuitiv prin efectul de gîtuire datorat MGG.

A4 pare să se situieze constant ca performanţe între A2 şi A3 independent de numărul de procesoare, lucru foarte important căci ea este un bun candidat pentru SIM foarte mari. Încrucişarea curbelor A1 şi A2 are loc la încărcări foarte mici.

Odată cu creşterea numărului comunicaţiilor interprocesor A1 devine semnificativ mai puţin performanţă ca celelalte.

ii) Cazul b = 2, figurile şi tabelele

O primă concluzie ce se desprinde este că în cazul A5 este că creşterea numărului de procesoare peste 12 duce la o creştere îndoielnică a performanţelor. A5 nu poate constitui o soluţie acceptabilă pentru SIM mari, în care aşa cum s-a presupus procesoarele interacţionează prin schimbări de mesaje. Transferul unui singur mesaj necesită două apelări ale memoriilor comune dublînd aproape traficul pe magistrale faţă de A6, A7, A8.

O comparaţie a performanţelor celor patru arhitecturi pentru o încărcare fixă arată că A8 e superioară lui A7 care la rîndul ei e superioară lui A6. In cazul SIM mari după atingerea saturaţiei diferenţele între ele tind să devină mai mici fiind factorul critic al sistemului. In cazul a 12 procesoare diferenţele sînt deja foarte mici.

O comparaţie a puterii de prelucrare în cazul b = 1 şi b = 2 arată că la saturaţie puterea de prelucrare a SIM cu două MGG este dublă faţă de a SIM cu o MGG. Rezultatul este foarte intuitiv căci la saturaţie factorul critic devine reţeaua de interconectare.

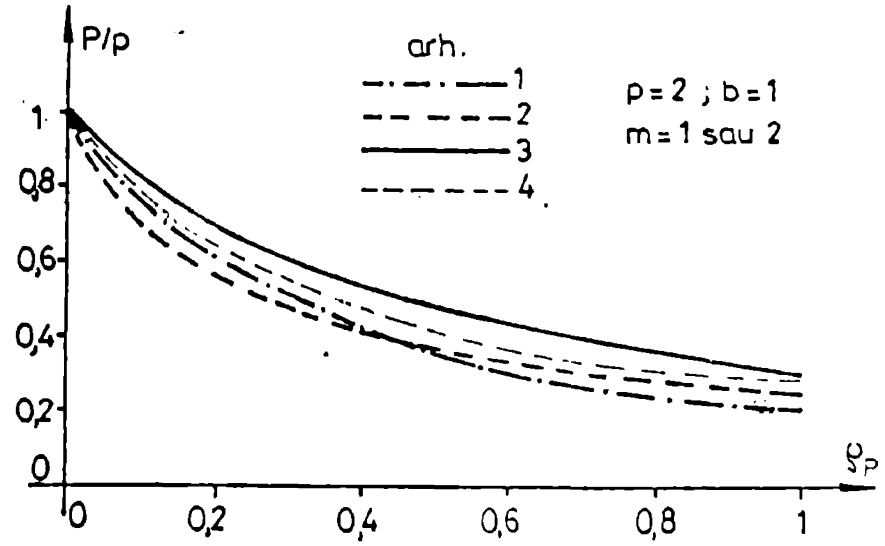
In final se poate remarca că diferenţa între A7 şi A8 e mult mai mare decît între A2 şi A3. Utilizarea memoriilor cu porturi multiple prezintă se pare interes numai pentru SIM de mare performanţă.

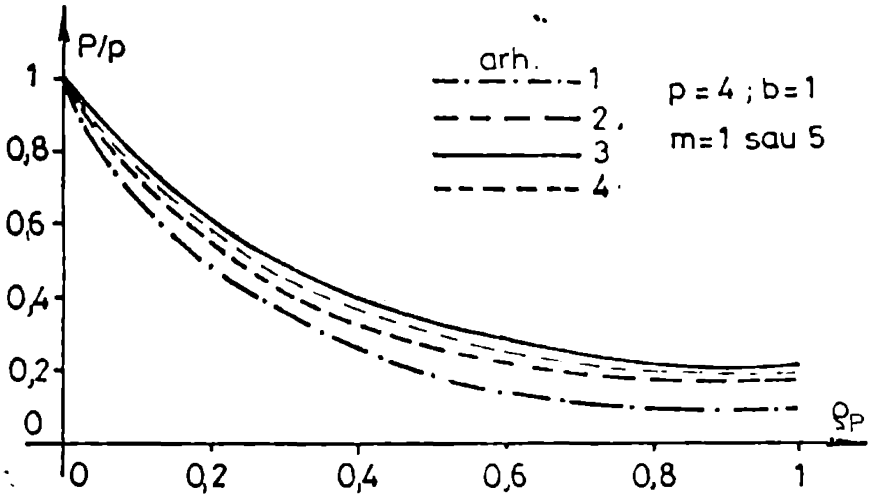
Complexitatea modelelor analitice este deja foarte mare în cazul a două magistrale fapt care a condus la utilizarea de modele analitice aproximative şi numai pentru SIM de mici dimensiuni.

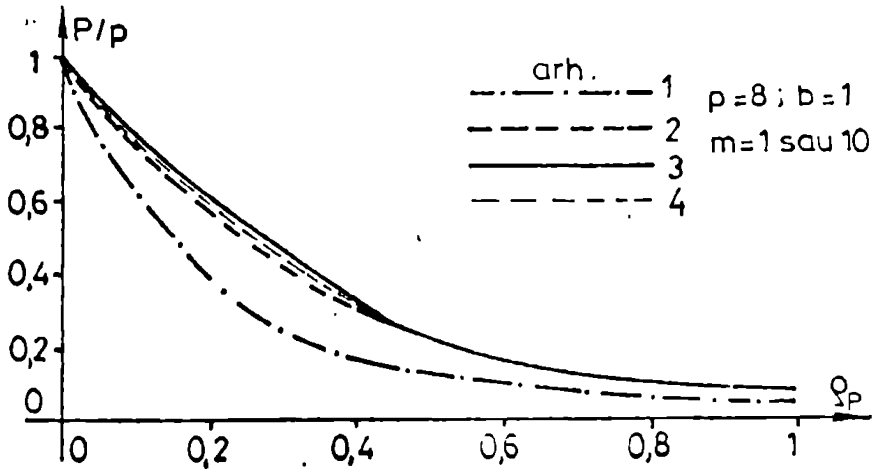
Cazul MA cu b magistrale a fost abordat pentru cîteva situaţii particulare (4 x 3 x 2), (4 x 4 x 3), (6 x 4 x 1), (6 x 4 x 2), (6 x 4 x 3), (6 x 4 x 4), (16 x 8 x 3) prezentate în fig.

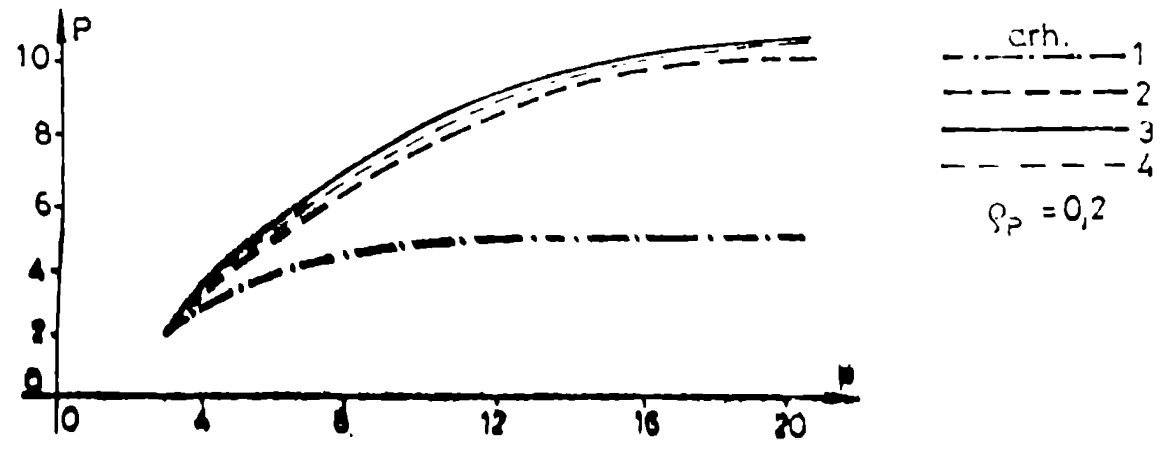
Modelele aproximative utilizate pun în evidenţă pentru toate cazul de bună comportare cu a programelor de simulare. Presupunerea făcută că fiecare memorie comună este adresată cu aceeaşi probabilitate, maxmizează valoarea puterii de prelucrare căci valori mai mari pentru o MC ar implica că ea devine punctul critic al SMM dar valori mai mici ar duce la încărcări sporite pentru celelalte memorii. Presupunerea că toate procesoarele generează aceiaşi fracţiune din traficul pe magistrală este dezavantajoasă ea conducînd la valori mai slabe ale puterii de prelucrare.

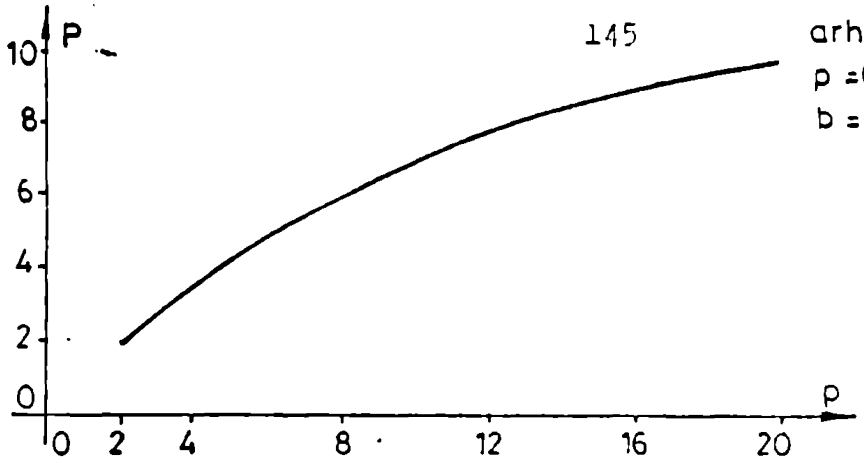
Creşterea eficienţei prin creşterea numărului de magistrale este neglijabilă pentru valori mici ale lui dar devine semnificativă pentru valori mari ale lui

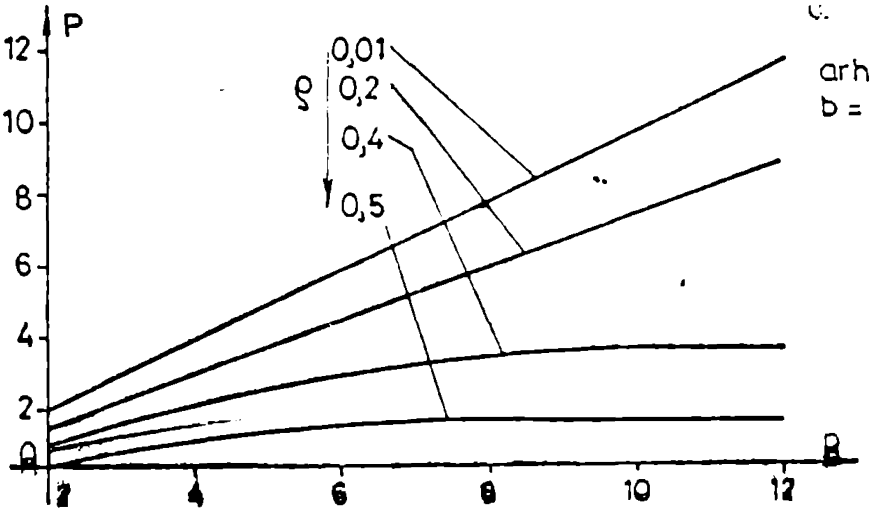

P/p
arh.
1
2
3
4
p = 2 ; b = 1
m = 1 sau 2
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1
ρP

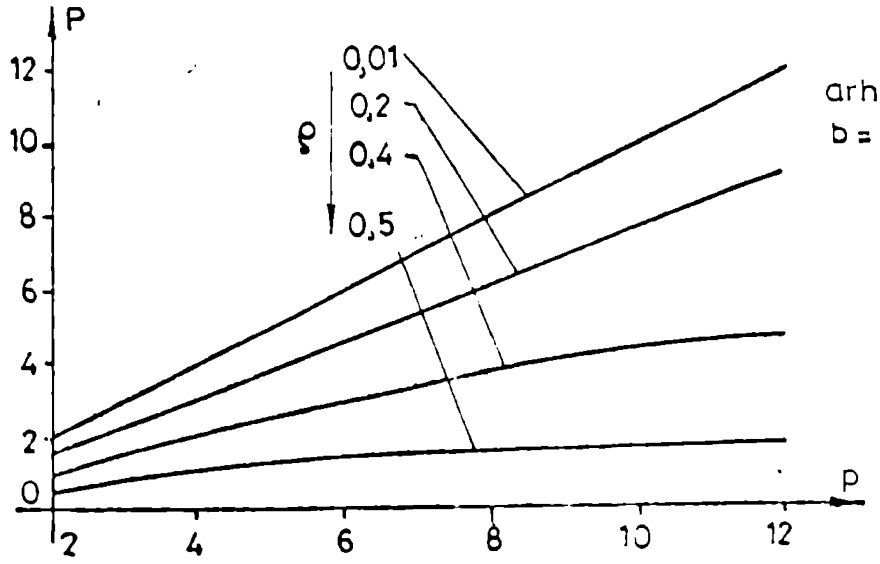

P/p
arh.
1
2
3
4
p = 4 ; b = 1
m = 1 sau 5
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1
ρP

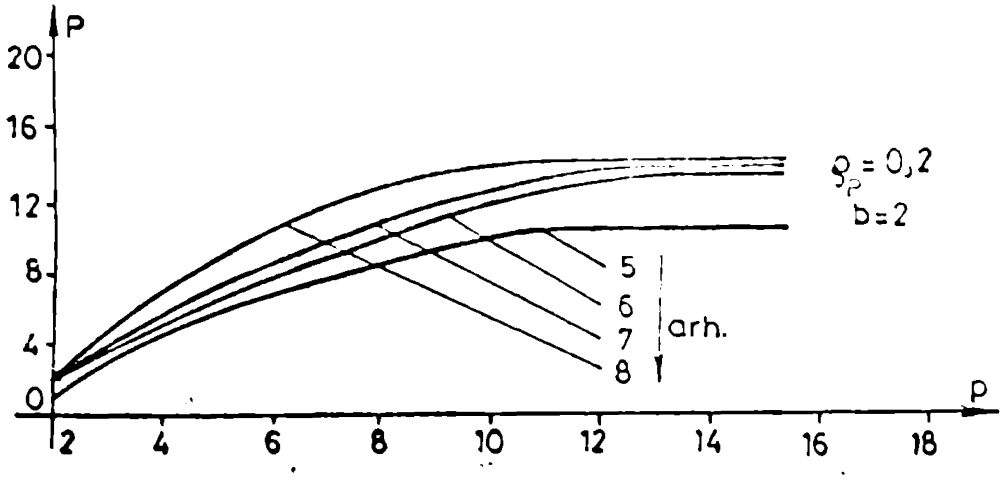

P/p
arh.
1
2
3
4
p = 8 ; b = 1
m = 1 sau 10
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1
ρP

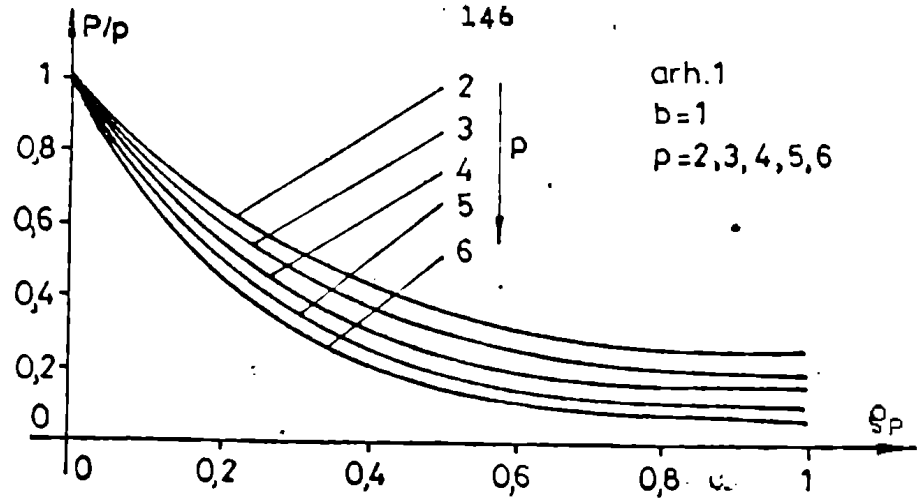

P
arh.
1
2
3
4
ρP = 0,2
10
8
6
4
2
0
0
4
8
12
16
20
n

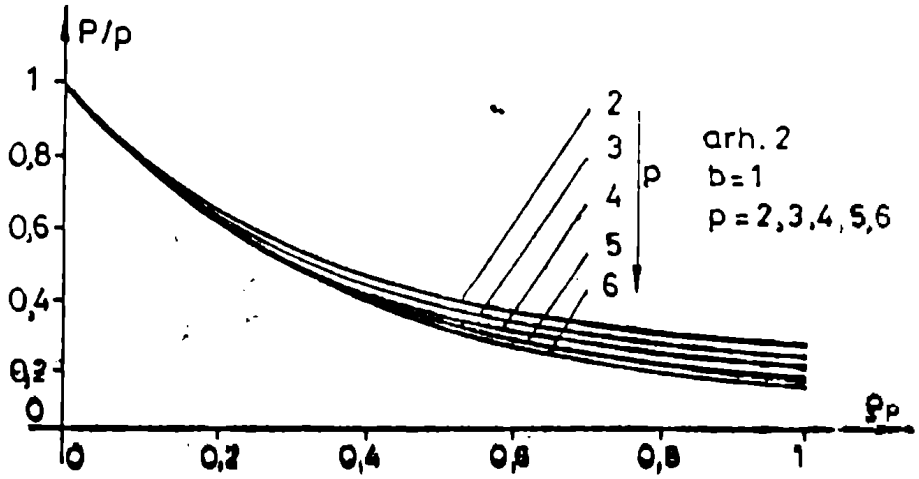

P
p
0
2
4
8
12
16
20
arh. 5
p =0,2
b = 2

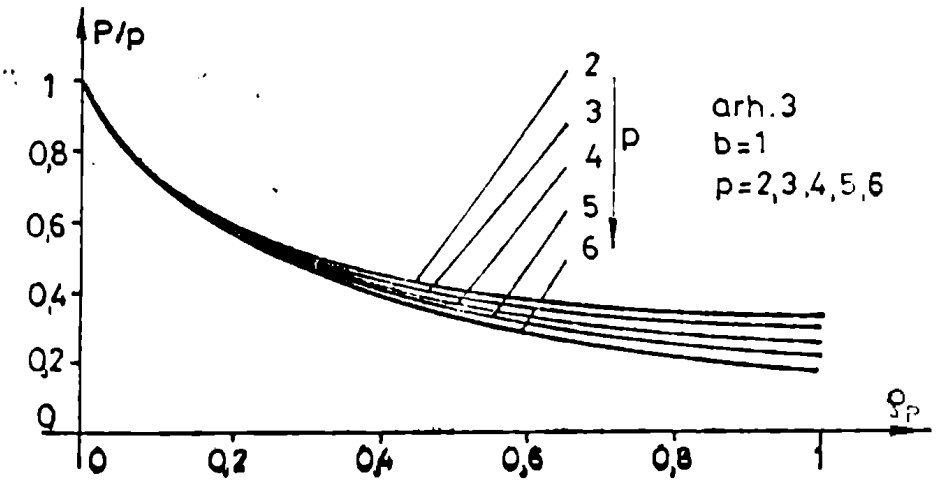

P
p
ρ
0,01
0,2
0,4
0,5
arh. 6
b = 2

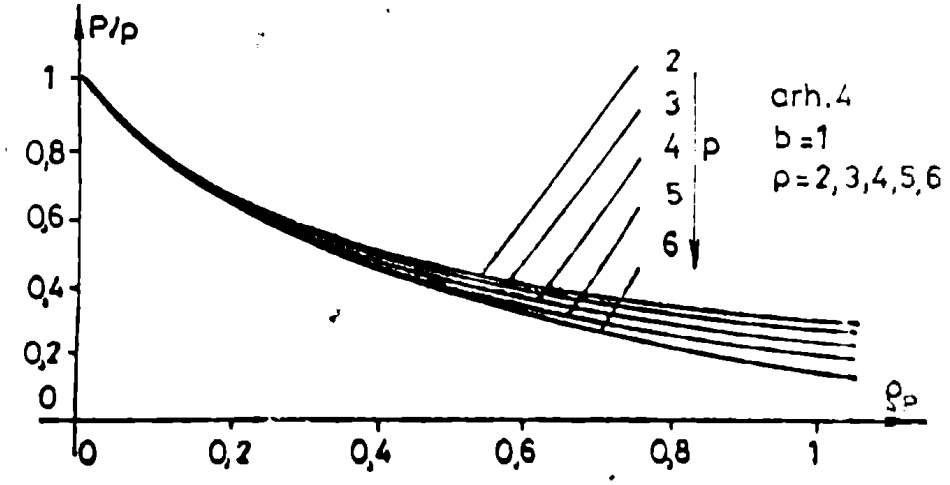

P
p
ρ
0,01
0,2
0,4
0,5
arh. 7
b = 2

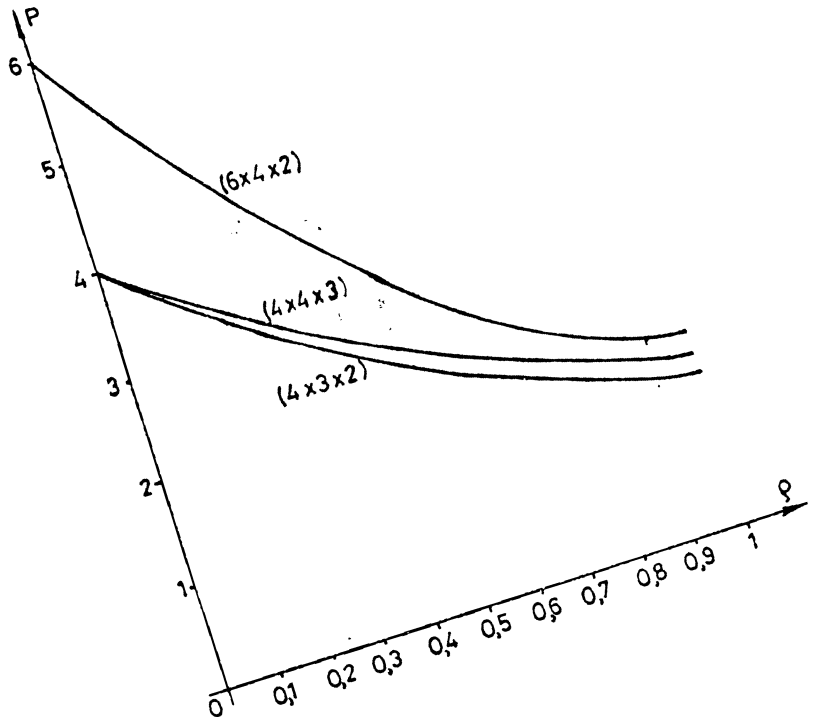

P
p
5
6
7
8
arh.
ρp = 0,2
b=2


P/p
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1
2
3
4
5
6
p
arh.1
b=1
p=2,3,4,5,6
ρp


P/p
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1
2
3
4
5
6
p
arh. 2
b=1
p=2,3,4,5,6
ρp


P/p
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1
2
3
4
5
6
p
arh.3
b=1
p=2,3,4,5,6
ρp


P/p
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1
2
3
4
5
6
p
arh.4
b=1
p=2,3,4,5,6
ρp



5. ANALIZA IN TIMP REAL A COMPORTARII SISTEMELOR AUTOMATE UTILIZIND SMI

In abordarea problemelor tehnico-ştiinţifice, tehnico-economice şi mai ales în cadrul mai restrîns al SA, cu mijloacele teoriei sistemelor au fost evidenţiate trei probleme generale BEL : analiza (stabilirea obiectivului şi a performanţei); sinteza structurii, identificarea structurală şi parametrică; sinteza comenzii (sau a programului în cadrul structurii precizate).

Complexitatea aparatului matematic utilizat şi volumul calculelor care intervin în rezolvarea problemelor enumerate justifică utilizarea SMI ca instrument practic de lucru. SMI pot interveni în analiza SA în două moduri diferite:

- ca mijloc de prelucrare a informaţiei apriorice sau de rezolvare a problemelor matematice ale diferitelor etape ale procesului de analiză (off-line);

- ca instrumente de prelucrare în timp real a informaţiei de lucru, prin îndeplinirea în sistemul analizat a rolului unuia sau mai multor blocuri (în conexiune directă, on-line).

O analiză riguroasă a comportării unui sistem pune cercetătorul în faţa unei dileme formulată astfel de P. Rykhoff [RYK]: "Gîndirea umană prezintă rezerve nesecate privind abilitatea de a analiza situaţiile în care relaţiile dinamice, cauză-efect, joacă un rol esenţial. Limitările inerente ce apar în acest proces sînt cauzate de faptul că analiza poate fi făcută numai prin imaginarea unui model, o proiecţie a acelei părţi a realităţii complexe care prezintă interes".

Analiza unui sistem automat poate, ca atare, fi gîndită ca reprezentînd construcţia şi studiul unui model care surprinde aspectele esenţiale ale sistemului şi care oferă date despre aceste aspecte într-o formă cît mai edificatoare. In acest sens nu trebuie uitat nici o clipă scopul în care a fost construit modelul (utilizarea lui potenţială):

- interpretarea comportării trecute a sistemului, înţeleasă de "condensarea cunoştinţelor disponibile şi a datelor măsurate, restrîngerea numărului parametrilor relevanţi;

- prezicerea comportării viitoare a sistemului, prognoze, detectarea tendinţelor eventual a regimului staţionar;

- acumularea datelor trecute şi prezente în scopul refacerii modelului după fiecare set de măsurători (măsurări indirecte sau estimarea cantităţilor ce nu se pot măsura direct);

- asigurarea cunoştinţelor despre sistem în scopul asigurării unui regim de funcţionare automată.

Procesul de obţinere (identificare) a unui model este perfect ilustrat prin următoarea schemă elaborată de [EYK2]:

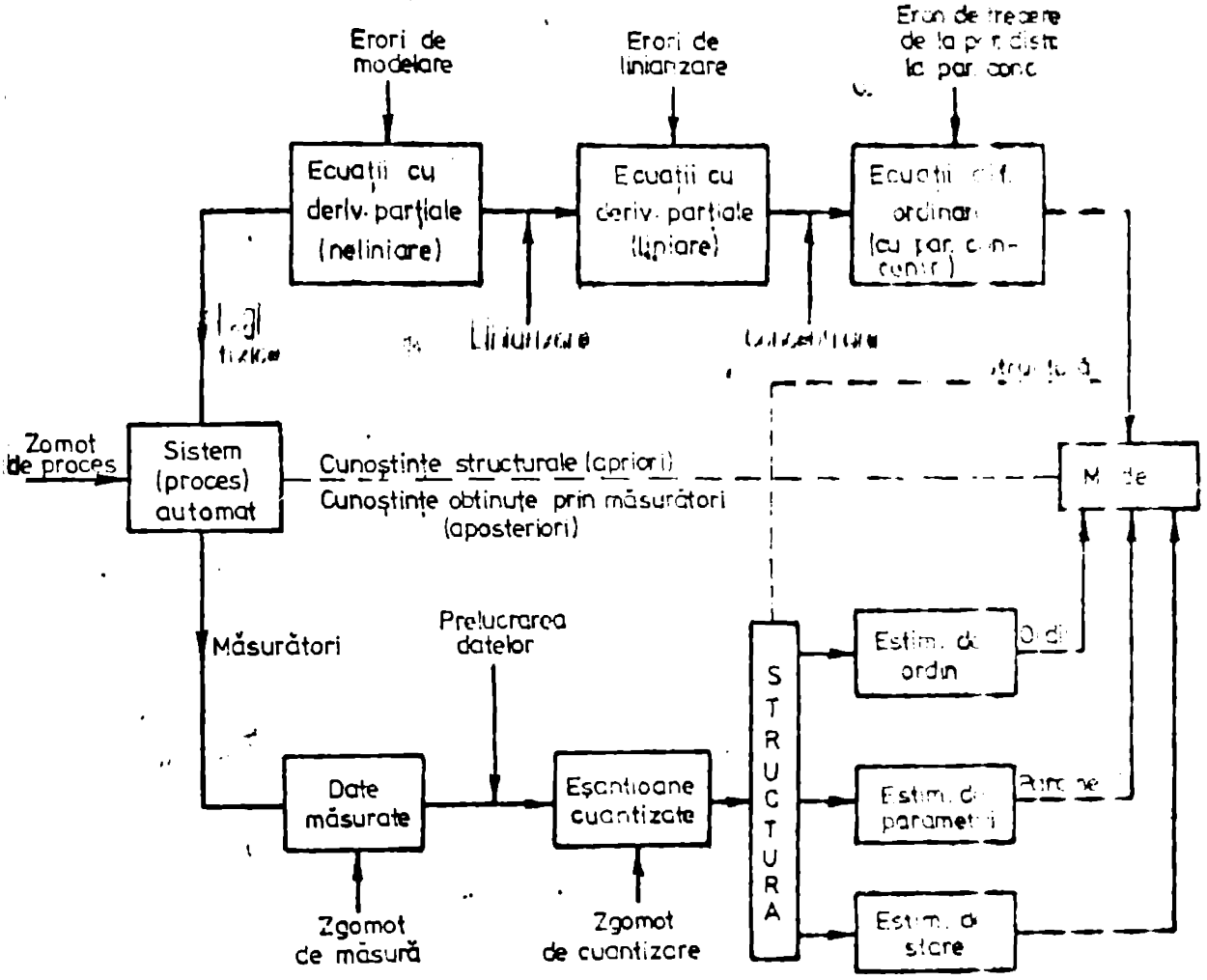


Fig. 6-1

Analiza în timp real a comportării SA presupune o bună cunoaştere a tipurilor de modele utilizabile, a relaţiilor dintre ele şi a utilităţii lor în analiza unor aspecte particulare ale comportării SA.

Primul pas pe drumul construcţiei unui model constituie culegerea datelor urmată de o prelucrare preliminară în sensul conversiei analog, numerice a eşantionării şi cuantizării acestor date.

În literatura de specialitate se găsesc amplu comentate referitoare la acest subiect.

Următorul pas este cel al alegerii unei structuri corecte a modelului ce va servi pentru analiza comportării SA.

Termenii de modelare şi cel de simulare vor fi utilizaţi în cele ce urmează într-un mod puc în evidenţă în. ig.
Modelarea surprinde relaţiile dintre sistemele reale i modele; simularea surprinde relaţiile dintre calculator (S M) şi modele. Pentru sistem real am desemnat partea din lumea reală avută în vedere. Din punctul de vedere al simulării sistemul real este privit ca o sursă de date privind comportarea acelui sistem. Modelul constă într-un set de instrucţii care specifică modul de generare a unor date de comportare de forma dorită. Modelele pot fi specificate în diverse forme: modele matematice (ecuaţii diferenţiale cu diferenţe); programe care specifică unui S M modul în care să genereze date de comportare pornind de la un model matematic. De i modelele însele nu generează date se va vorbi în continuare de date generate de model sau de comportarea modelului în sensul că modelul este cel care asigură instrucţiile pe baza cărora S M generează comportarea dorită.

Se va distinge în continuare între comportarea modelului şi structura modelului. Comportarea este modul de exteriorizare a sistemului pe cînd structura este ceea ce asigură comportarea respectivă. Din punctul de vedere al reproducerii comportamentului se disting: modelele replicative (asigură date de comportare deja cunoscute); modele predictive (asigură date de comportare înainte ca ele să fie obţinute de la sistemul real); modele structurale (asigură nu numai o comportare similară dar reproduce şi modul în care operează sistemul real).

Simularea trebuie să asigure o comportare cît mai apropiată de a sistemului real pe baza unor modele matematice i a unor programe cît mai fidele. Relaţia de modelare este cea care asigură măsura mai mare sau mai mică în care modelul matematic generează date de comportare corecte.

In mod formal un model este descris printr-un set de componente (părţi ale modelului); variabile descriptive (se vor pentru a descrie starea componentelor la diverse momente de timp), şi interacţiuni între componente (legile după care interacţionează componentele modificîndu-şi stările determinînd astfel evoluţia în timp a comportării sistemului).

Modelul trebuie să fie complet (să asigure o descriere completă a tuturor situaţiilor relevante), consistent (să nu conţină opţiuni contradictorii) şi să nu fie ambiguu (să specifice fără dubiu calea de urmat pentru fiecare situaţie posibilă).

In continuare se vor trece în revistă cele mai importante categorii de modele. O clasificare de bază a modelelor este cea referitoare la baza de timp utilizată modelului: modele cu timp continuu şi modele cu timp discret (t R respectiv t Z). O altă clasificare are în vedere mulţimea valorilor variabilelor descriptive: modele cu stări discrete, modele cu stări continue şi modele cu ... de ambele tipuri. Modelele cu timp continuu pot fi subclasificate în : modele cu evenimente discrete şi ecuaţii diferenţiale. O altă clasificare este dată în funcţie de includerea în model a variabilelor aleatoare: modele determinist şi modele stohastice. În funcţie de măsura în care modelul ţine cont de interacţiunea mediu - sistem real, se disting: modele autonome şi modele neautonome sau închise, respectiv deschise. În funcţie de măsura în care legile de interacţiune ale unui model depind de variabila timp se pot distinge : modele invariante în timp şi modele variabile în funcţie de timp.

Elementele de bază ale analizei comportamentului unui sistem real sînt: sistemul real propriu-zis; cadrul experimental, modelul ideal (modelul de bază), modelele simplificate, sistemul de calcul (SC). Aceste elemente reprezintă aspecte conceptuale majore ale procesului de modelare şi simulare. Deşi am menţionat sistemul real va fi privit numai ca o sursă de date privind comportarea sa. Variabilele descriptive ale unui sistem pot fi clasificate în: observabile şi neobservabile. Variabilele neobservabile nu pot fi măsurate direct şi totuşi ele pot influenţa în mod hotărîtor comportarea sistemului. Variabilele observabile cele mai comune sînt variabilele de intrare şi cele de ieşire aflate în interrelaţie de la cauză la efect.

Mulţimea tuturor perechilor de valori intrare/ieşire ce pot fi obţinute în urma unui experiment se defineşte pentru acesta ca urmează ca fiind comportarea intrare/ieşire a sistemului real.

Cadrul experimental este cel care fixează modul în care este observat sistemul şi în care decurge experimentul. Fiecărui cadru experimental îi este asociată o anumită comportare a sistemului real. Orice model obţinut în aceste ipoteze este valid numai pentru cadrul experimental respectiv.

Modelul ideal sau modelul de bază este un model valid în orice cadru experimental. Cum un astfel de model este imposibil de obţinut se recurge cel mai adesea la modele simplificate care surprind caracteristicile esenţiale ale sistemului real şi a căror validitate este limitată la unul sau la cîteva cadre experimentale.

Sistemul multimicroprocesor este dispozitivul de calcul cu ajutorul căruia sînt generate perechile intrare/ieşire al modelului simplificat. Comportarea sistemului se obţine pas cu pas de la un moment la altul. Procesul este denumit: simulare. Instrucţiile care pun în operă procesul de simulare, pe cicluri (iterativ, inductiv, recursiv) sînt furnizate de modelul simplificat. Simularea impune importante constrîngeri asupra programelor de simulare în ceea ce priveşte: timpul (timp real), necesarul de memorie, efortul de programare şi depanare.

5.1. MODELE MATEMATICE UTILIZATE PENTRU ANALIZA COMPORTĂRII SISTEMELOR UTILIZÎND SIM

5.1.1. CONVENTII SI NOTATII

S-a menţionat anterior că componentele unui model sînt descrise de o mulţime de variabile descriptive. Se spune că un set de variabile descriptive constituie un set de variabile de stare dacă la orice moment de timp t valorile acestor variabile (singure) determină în mod unic valorile tuturor variabilelor descriptive la orice moment următor t. Un model definit astfel se numeşte model pe stare. Programul de simulare accesat unui astfel de model satisface următoarele proprietăţi :

- Iniţializarea programului: după programul de simulare de calculat valorile variabilelor descriptive la momentul t dîndu-se valorile lor la momentul t, singurele locaţii de memorie ce trebuie iniţializate sînt cele ce corespund variabilelor de stare.

- Repetarea unei rulări: dacă trebuie repetate calculele variabilelor descriptive la momentul t dîndu-se valorile lor a momentul, datorită apariţiei unor erori la prima rulare, rezultatele vor fi aceleaşi cu condiţia ca variabilele de stare să fie iniţializate cu aceleaşi valori.

- Tratarea întreruperilor: dacă se doreşte programul de calcul al variabilelor descriptive să nu fie afectat de eventuale întreruperi în urma unei întreruperi, în stivă trebuie salvată şi starea programului şi vectorul variabilelor de stare. Aceiaşi situaţie se recomandă şi în cazul situaţiilor de avarie: salvarea conţinutului vectorului variabilelor de stare pentru a relua execuţia programului de simulare de la început şi nu din punctul în care a apărut avaria.

- Din cele spuse anterior rezultă modul iterativ în care decurge procesul de simulare la momentele $t_1, t_2, \ldots, t_n$ de simulare. Programul poate simula tranziţia modelului de la t_{i-1} la t_i

pentru orice pereche (t_i, t_{i+1}) de momente de calcul. Acest tip de simulare se numeşte simulare discretă. Dacă legile de interacţiune ale modelului nu depind de timp ci doar de variabilele de stare modelul simulat este invariant în timp.

5.1.2 Simularea discretă a modelelor invariante în timp

Se dau valorile $x_1, \ldots, x_m$ ale variabilelor de stare la momentul $t = t_i$. Se cer valorile variabilelor de stare la momentul $t = t_{i+1}$. Procedura de calcul are următoarele operaţii elementare:

Pas 1 : se iniţializează locaţiilor de memorie ataşate variabilelor de stare, locaţii notate cu x1, ... ,x ;

Pas 2 : se iniţializează contorul corespunzător "ceasului modelului" cu valoarea t_i;

Pas 3 : se aplică locaţiilor x1, ... ,xm rutinele care realizează legile de interacţiune pentru calculul valorilor noi ale locaţiilor de stare cît şi pentru calculul celorlalte variabile descriptive;

Pas 4 : se incrementează contorul-ceas;

Pas 5 : se verifică dacă conţinutul contorului depăşeşte valoarea M+N. Dacă da se opreşte calculul; dacă nu se trece la pasul 3. Procedura simulează corect modelul dacă calculele legate de o tranziţie sînt exacte. Dacă nu oricît ar fi de mică acumulată la fiecare iteraţie poate genera erori grosolane. Procedura este secvenţială în sensul că dispozitivul de calcul execută operaţiile una după alta. Procedura este implementată de o secvenţă respectivă este repetată de un număr de ori.

În ceea ce priveşte structura pasului 4 se observă că el constă într-o macroinstrucţie (eventual o subrutină) care acceptă o listă de variabile de stare şi în cazul nostru o listă de variabile descriptive de intrare şi produce o listă cu noile valori ale variabilelor de stare cît şi o listă cu variabilele de ieşire. Subprogramul materializează în fond următoarele funcţii

$$\delta : X \times U \to X$$

$$\beta : X \times U \to Y$$

unde X - mulţimea valorilor variabilelor de stare, U - mulţimea valorilor variabilelor de intrare, Y - mulţimea valorilor variabilelor de ieşire. În acest caz se pot defini în mod unic traiectoriile de stare şi cele de ieşire cu condiţia specificării stărilor iniţiale şi a traiectoriei de intrare.

Perechea compusă dintr-o traiectorie de intrare și o traiectorie de stare asociată se numește trajectorie intrare-stare. Mulțimea tuturor acestor perechi poartă denumirea de comportare pe stare a modelului. În mod similar se definește noțiunea de traiectorii intrare-ieșire. Mulțimea tuturor acestor perechi poartă denumirea de comportare intrare-ieșire a modelului.

5.2 Simulare paralelă

Am arătat că un model poate fi privit ca fiind alcătuit din mai multe componente care interacționează. Această interacțiune are loc de cele mai multe ori de natură paralelă. Spre deosebire de interacțiunea secvențială, cea paralelă poate implica mai multe acțiuni ce se desfășoară simultan. Natura paralelă a interacțiunilor se datorează faptului că mai multe, chiar toate componentele modelului pot fi simultan active.

Deși cele mai multe modele sînt întrinsec paralele, pînă la apariția sistemelor multimicroprocesor cele mai multe dispozitive de calcul lucrau secvențial. În cazul prelucrării secvențiale comportarea generată (simulată) de calculator nu poate fi o replică fidelă în timp cu a celei descrise de model. Acest lucru se datorește faptului că acțiunile paralele ale modelului sînt secvențializate pierzîndu-se astfel cea mai importantă proprietate a comportării acesta de a se desfășura în timp real.

În modelele matematice discrete se pot privi trei tipuri de funcții:

- funcții algebrice (operații algebrice) de tipul

$y_i = FA(x_1, \ldots, x_n) \quad i = 1, \ldots, m$

- funcții de timp

$u_i = FT(t) \quad i = 1, \ldots, r$

- funcții dinamice (cu memorare)

$x_i = SIM(x_1, \ldots, x_n, u_1, \ldots, u_r)$

Funcțiile dinamice nu sînt altceva decît ecuațiile cu diferențe corespunzînd dinamicii sistemelor (în sensul teoriei sistemelor) sub forma modelelor pe stare. Ele conțin ca argumente variabilele de stare a căror valori trebuie inițializate o dată cu începutul execuției. Intern o funcție de tranziție a stării determină evoluția stării sub influența evoluției mărimilor de intrare u_i iar o funcție de ieșire determină traiectoria variabilelor de ieșire corespunzînd evoluției determinate a stării. În cazul modelelor discrete în timp o anumită componentă de ieșire este o funcție de elementul de întîrziere. Orice model discret poate fi descris cu ajutorul acestei dinamici.

Utilizînd un limbaj formal simplu bazat pe implementarea celor trei tipuri de funcţii se va pune în evidenţă paralelismul inerent modelelor discrete cît şi problemele legate de ordonarea proceselor corespunzătoare. Limbajul rezultat prezintă totodată marele avantaj de a transpune imediat descrierea grafică a modelelor discrete liniare (diagramele bloc).

Fie lista funcţiilor programului de simulare $S_1, \ldots, S_n$.

Se verifică pentru început că descrierea generală are sens:

i) se verifică dacă vre-o variabilă apare în membrul stîng a două sau mai multe instrucţii distincte; dacă da descrierea este eronată o variabilă fiind multiplu definită; într-o diagramă bloc situaţia ar corespunde la două arce convergente în acelaşi punct (fizic nu de sumare).

ii) se înlătură formal toate instrucţiile de apelare a funcţiilor FT şi SIM ; programul conţine acum numai instrucţii de tipul FA.

iii) pe mulţimea instrucţiilor rămase se introduce o relaţie de precedenţă (de ordine) de următorul tip: $u \to y$ dacă u precede y ori de cîte ori y este variabila de ieşire a unei instrucţiuni în care u figurează ca argument $y = FA(\ldots,u,\ldots)$. Se notează $u \to y$.

iv) relaţia $\to$ este tranzitivă $u \to y$, şi anume

$$u = v_0 \to v_1 \to \ldots \to v_{n-1} \to v_n = y$$

corespunzător $u \to y$ ori de cîte ori în diagrama bloc corespunzătoare există o cale de la u la y traversînd numai funcţii FA.

v) se verifică dacă există vre-o variabilă y astfel încît $y \to y$ dacă da programul este incorect şi trebuie corectat. Dacă descrierea este corectă se trece la evidenţierea paralelismului.

vi) tuturor variabilelor ce constituie elementele iniţiale pentru relaţia $\to$ li se ataşează rangul 0. Rangul oricărei alte variabile este lungimea celui mai lung şir de la cel puţin o din variabilele de rang 0. In sensul definiţiilor date rangul unei variabile este numărul maxim de funcţii FA necesare pentru calculul acelei variabile la momentul t dacă se dau toate variabilele de rang 0 la momentul t.

vii) fiecărei instrucţii de simulare de tip FA i se asociază un rang în felul următor: rangul (S_i) = max rang (v_j), $i = 1,\ldots,n$ unde S_i este $y = FA(v_1,\ldots,v_p)$. Instrucţiunea de simplă FA nu poate fi calculată decît ce toate argumentele ei au fost calculate.

viii) se verifică dacă toate variabilele inițiale (de rang o) sînt ieșiri a unor FT sau funcții SIM. Dacă nu descrierea nu este validă deoarece înseamnă că există variabile de intrare care nu sînt generate din exterior (FT) și/sau nu este disponibilă dintr-o iterație precedentă

ix) se ordonează toate instrucțiile S_i după rang, toate instrucțiile pînă la același rang pot fi prelucrate în paralel.

x) toate instrucțiile de tip FT trebuie plasate la începutul programului de simulare; toate instrucțiile de tip SIM trebuie plasate la sfîrșitul programului. Ambele categorii FT și SIM permit prelucrarea instrucțiilor lor în paralel.

Simularea funcțiilor de dinamică este tratată separat. Dacă se consideră cazul general cînd SIM implementează un sistem dinamic continuu oarecare apare o problemă deosebită care va fi subliniată în cele ce urmează. Fie

$$y_j = SIM(x_1,\ldots,x_n, u_1,\ldots,u_r)$$

și $y_j(t_i)$, x_1 $x_k(t_i)$ valorile la momentul t_i. In simularea numerică este fixat un moment următor t_{i+1}. Programul de simulare dispune numai de valorile la momentul t_i, din care trebuie să estimeze valorile la momentul t_{i+1} fără să știe ce se întîmplă pe durata t_i pînă în t_{i+1} adică fără să calculeze traiectoriile intrărilor, stărilor și ieșirilor pe intervalul t_i, t_{i+1}). In cele ce urmează se va presupune că aceste traiectorii sînt constante pe intervalele respective.

Se va considera cel mai natural model, cel al sistemelor discret de formă:

$$x_{k+1} = A x_k + Bu_k \qquad (A,B,C)$$

$$y_{k+1} = Cx_{k+1} \qquad k\in Z, x\in R^n, u\in R^r, y\in R^s$$

matricile utilizate avînd următoarele dimensiuni și semnificații: $x_k(nx1)$ și $u_k(rx1)$ starea și intrarea curentă, x_{k+1} și $y_{k+1}(sx1)$ starea și ieșirea următoare, A (nxn) matricea de tranziție a stărilor, B (nxr) matricea de intrare, C (sxn) matricea de ieșire.

Se subliniază că modelul introdus poate să reprezinte un model matematic simplificat. Caracterul limitat al resurselor (timp de calcul și memorie) care pot fi alocate constituie factorul determinant care impune selectarea și adaptarea corespunzătoare a algoritmilor la restricțiile și particularitățile utilizării în timp real. Un mijloc important prin care se poate realiza această adaptare constă în reducerea dimensiunii modelelor matematice cu ajutorul cărora se analizează dinamica SA.

In capitolul de față se va utiliza un fel abstract matematic $S = (X, U, Y, \delta, \beta, T)$ care reflectă complet structura unui sistem real, fiind în măsură să genereze comportamentul său intrare, ieșire. Acesta este modelul matematic al sistemelor dinamice liniare invariante în timp sisteme ce vor fi analizate în cele ce urmează. In continuare se va analiza relația dintre structură și comportarea sistemelor.

Procesul de modelare presupune stabilirea de relații între perechi de sisteme. Aceste relații asigură: validitatea reprezentării sistemului real prin modelul matematic, validitatea modelului simplificat relativ la modelul matematic, corectitudinea programului de simulare relativ la modelul simplificat. Fiecăruia dintre aceste nivele de reprezentare a sistemului îi corespunde o relație între perechi de sisteme în cadrul acelui nivel. Acest tip de relații vor fi denumite în cele ce urmează relații de conservare sau morfisme. Exemple de asemenea morfisme vor fi date în paragrafele următoare. In continuare vom specifica morfismele ce corespund fiecăruia din nivelele la care poate fi specificat un sistem.

Cadrul experimental poate fi specificat astfel: aplicăm sistemului segmente de intrare definite prin perechea (U, ω) și observăm segmentele de ieșire definite prin perechea (Y, ς). Notăm $\omega \in (U, T)$ și $\varsigma \in (Y, T)$. Se adoptă convenția ca domeniile celor doi ω și ς coincid (intervalul de observație este același). În acest caz perechea (ω, ς) este denumită pereche intrare/ieșire sau pe scurt I/E. Experimentul conduce la obținerea unei mulțimi de perechi I/E care definesc o relație I/E. In concluzie o relație experimentală I/E (REIE) se definește astfel:

(T, U, Ω, Y, R) unde T este timpul, U și Y mulțimea variabilelor de intrare respectiv ieșire, $\Omega \in (U, T)$ mulțimea segmentelor de intrare, $R \subset \Omega \times (Y, T)$ cu $(\omega, \varsigma) \in R$ relația I/E, $\mathrm{dom}(\omega) = \mathrm{dom}(\varsigma)$. Această structură va fi notată prin simbolul R.

REIE sau R asigură suma cunoștințelor despre sistemul real conceput ca o cutie neagră. Apar imediat două probleme. Prima este problema trecerii de la structură la comportare: dacă se cunoaște structura din cutie se poate descrie într-un fel sau altul comportarea sa exterioară. A doua problemă este reciprocă: pornind de la comportare să se deducă structura. Prima problemă este cea care va fi analizată în cele ce urmează, constituind de fapt procesul de simulare.

Pentru ca experimentele de simulare să fie concludente şi repetabile modelul trebuie de fiecare dată adus în aceeaşi stare iniţială. In acest caz fiecărui segment de intrare îi va corespunde un unic segment de ieşire asociate printr-o funcţie f. Dacă experimentul se repetă pentru un număr mai mare de stări iniţiale se obţine un număr corespunzător de asemenea funcţii f. Putem deci defini:

O funcţie experimentală I/E (FEIE) ca fiind (T, U, Ω, F) unde T, U, Ω şi Y au semnificaţiile cunoscute iar

$$F = \{ f \in F \mid f \in \Omega \times (Y, T),\ \varphi = f(\omega) \}$$

Relaţia dintre R şi F este $R = \bigcup_{f \in F} f$.

FEIE reprezintă un nivel de cunoaştere superior REIE, ultimul pierzînd informaţia referitoare la grupările funcţionale f care au fost reunite.

La nivelul FEIE se deţin informaţii despre starea iniţială dar nu se ştie nimic despre starea finală în urma experimentului. Se defineşte din acest motiv o structură mai generală ca cea de sistem experimental intrare/ieşire SEIE sau pur şi simplu sistem $S = (T, U, \Omega, X, Y, \delta, \beta)$ cu semnificaţiile din paragraful 1

în care $\delta : X \times U \to X$ este dinamica care generează starea finală.

Avînd specificată structura sistemului se poate în sfîrşit genera comportarea sa prin descrierea traiectoriei stării şi a ieşirii ca răspuns la un segment de intrare, pe durata intervalului de experimentare.

Fiecărei stări $x \in X$ i se asociază funcţia:

$$\beta_x : \Omega \to Y \qquad \beta_x(\omega) = \beta(\delta(x, \omega))$$

Se notează cu $B_c = \{ \beta_x \mid x \in X \}$ comportarea I/E a sistemului S (generează aceiaşi informaţie intrare/ieşire ca şi sistemul real).

După cum s-a mai menţionat, dat fiindcă prin ipoteză am luat în considerare numai sisteme liniare invariante în timp, sistemul S poate fi formalizat prin tripletul (A, B, C) cu semnificaţiile din paragraful

Dacă pe mulţimea segmentelor de intrare se defineşte o operaţie de compunere astfel încît

$\omega = \omega_1 \circ \omega_2 \circ \ldots \circ \omega_n$ prin operaţia de compunere se defineşte natural prin segmentarea domeniului T printr-o mulţime $\{t_1, t_2, \ldots t_n\}$, $t_0 < t_1 < t_2 < \ldots < t_n < t$

atunci:

şi $\delta_\omega(x, \{t_1, t_2, \ldots t_n\}) = \delta(\ldots \delta(x, \omega_{(0,t_1)}), \ldots) \omega_{(t_n, t)}$

Altfel spus traiectoriile I/E se obțin prin aplicarea iterativă a segmentelor de intrare de rang i :

$$\delta(x,\omega) = \delta(\delta(\ldots\delta(\delta(x,\omega_1),\omega_2)\ldots),\omega_n)$$

iar această formalizare a sistemului S poartă denumirea de formalizare iterativă.

Putem concluziona că un sistem specificat printr-o diagramă de sisteme iterative definite poate fi convertit printr-o procedură standard într-un alt sistem iterativ care să genereze comportarea globală a sistemului real.

Dacă fiecare componentă a sistemului poate fi implementată printr-un program de simulare SMM, atunci întreaga diagramă poate fi implementată ca un program de simulare.

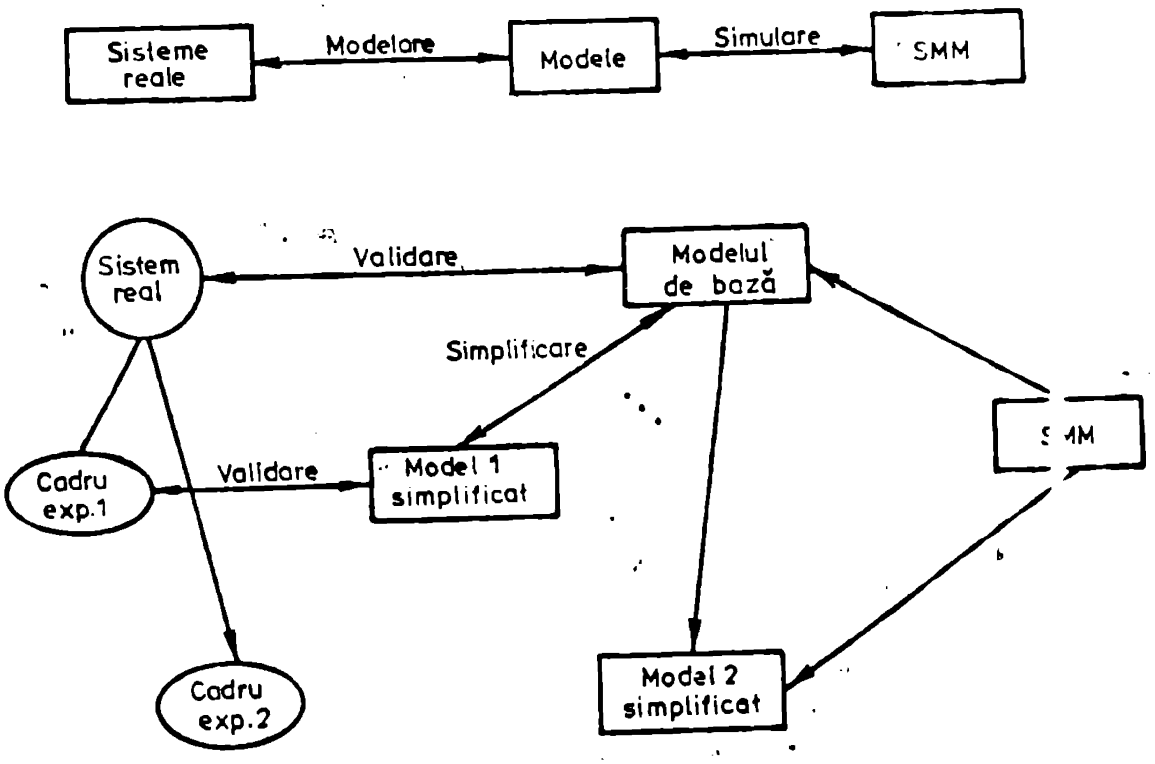


Fig. 6.2-1

5.3. ANALIZA COMPORTARII SA PRIN SIMULAREA CU REALIZARI MINIMALE

In cazul realizărilor minimale se utilizează următoarele matrici restrînse:

$$u^{*}_{k\,\|\,n} = [u_k \;\; u_{k-1} \ldots, u_{k-n+1}]$$

$$y_{*,k+1\|n} = [y_{k+1}, y_{k+2}, \ldots, y_{k+n}]$$

$$H_{\|\,n}^{\|\,n+1} = \begin{bmatrix} H_1 & H_2 \ldots & H_{n+1} \\ H_2 & H_3 \ldots & H_{n+2} \\ \ldots & & \\ H_n & H_{n+1} \ldots & H_{2n} \end{bmatrix}$$

$$M_{\|\,n} = [B, \; AB, \; \ldots, \; A^{n-1}B]$$

$$N_{\|\,n} = [C, \; CA, \; \ldots; \; CA^{n-1}]'$$

$H = N\,M$ în sensul că $y_* = N(Mu^*)$

$M = [M_1 \; M_2 \; M_3 \ldots]$ și $N = [N_1 \; N_2 \; N_3 \ldots]$

fiind matricile de accesibilitate observabilitate ale realizării.

In acest scop se presupune că pentru $i=1,2,\ldots$ matricea ^{-i}N are rang maxim n adică toate stările sînt accesibile prin aplicarea unor intrări adecvate u^{*}_{1}, caz în care M se notează $[AM] \rightarrow$ iar realizarea se zice accesibilă.

^{-i}N are rang maxim n adică orice stare poate fi aflată (observată) dacă se dau ieșirile $y_{*,k+1}$ ^{i}N (unde N se va fi notat $\leftarrow$ iar realizarea corespunzătoare se spune observabilă.

Realizările $(^{i}A, \; ^{i}B, \; ^{i}C)$ care sînt atît accesibile cît și observabile se zic minimale.

Orice două realizări minimale ale aceluiași sistem sînt echivalente.

o. Modelul Hankel

$$y_{*,k+1} = H\,u^{*}_{k} \quad \text{unde}$$

$$H_{\|\,n}^{\|\,n+1} = H_{\|\,n}^{\|\,n} \begin{bmatrix} 0 & \ldots & 0 & -\alpha_1 \\ 1 & & & -\alpha_2 \\ & \ddots & & \vdots \\ & & 1 & -\alpha_n \end{bmatrix}, \quad \operatorname{rang} H = n$$

$$H_{\|\,1}^{\|\,n} \cdot H_{\alpha} = [\beta_1 \ldots \beta_n] \neq 0, \quad H_{\alpha} = \begin{bmatrix} -\alpha_2 & \ldots & -\alpha_n \\ \vdots & & \vdots \\ -\alpha_n & & \end{bmatrix}$$

1. Modelul Hankel - coloană

$$\left[\begin{array}{c|c} 1_A & 1_B \\ \hline 1_C & 0 \end{array}\right] = \left[\begin{array}{ccc|c} 0 & & \alpha_1 & 1 \\ 1 & & \alpha_2 & 0 \\ & \ddots & \vdots & \vdots \\ & 1 & \alpha_n & 0 \\ \hline H_1 & \cdots & H_{n-1} H_n & 0 \end{array}\right]$$

1D. Modelul Hankel - linie

$$\left[\begin{array}{c|c} 1D_A & 1D_B \\ \hline 1D_C & 0 \end{array}\right] = \left[\begin{array}{c|c} 1_A & 1_B \\ \hline 1_C & 0 \end{array}\right]^T, \quad \text{rang} \begin{Vmatrix} 1D \\ H \end{Vmatrix} = r$$

$$H_\alpha \cdot {}^{1D}B = [\beta_1 \ldots \beta_n]$$

2. Modelul Bezout - coloană

$$\left[\begin{array}{c|c} 2_A & 2_B \\ \hline 2_C & 0 \end{array}\right] = \left[\begin{array}{ccc|c} 0 & & 1 & 0 \\ \vdots & \ddots & & \vdots \\ \alpha_1 & \alpha_2 \ldots & \alpha_n & 1 \\ \hline \beta_1 & \beta_2 \ldots & \beta_n & 0 \end{array}\right] \quad \text{rang} \begin{Vmatrix} 2 \\ S \end{Vmatrix}$$

2D. Modelul Bezout - linie

$$\left[\begin{array}{c|c} 2D_A & 2D_B \\ \hline 2D_C & 0 \end{array}\right] = \left[\begin{array}{c|c} 2_A & 2_B \\ \hline 2_C & 0 \end{array}\right]$$

3. Ecuația internă cu diferențe

$$y_{k+1} = [\beta_1 \ldots \beta_n] \begin{Vmatrix} S \\ \ldots \\ n \end{Vmatrix} \cdot S_{k+n+1} =$$

$$= [\alpha_1 \ldots \alpha_n] \begin{Vmatrix} S \\ \ldots \\ n \end{Vmatrix} + u_k, \quad \text{rang} \begin{Vmatrix} S \\ \ldots \\ 2n \end{Vmatrix}$$

$$S = \begin{bmatrix} T_\alpha \\ -T_\beta \end{bmatrix}, \quad T_\alpha = \begin{bmatrix} -\alpha_1 & \ldots & -\alpha_n & 1 \\ \ldots & & & \\ & & & \\ & -\alpha_1 & & -\alpha_n \end{bmatrix}$$

$$T_\beta = \begin{bmatrix} -\beta_1 & \ldots & -\beta_n & 0 \\ & & & \\ & & & \\ & -\beta_1 & & -\beta_n \end{bmatrix}$$

3D. Ecuţia cu diferenţe

$$y_{k+1} = [\alpha_1 \dots \alpha_n]\, y_{*,k-n\,\|\,n} + [\beta_1 \dots \beta_n]\, u_{*,k-n\,\|\,n}$$

4,4D. Fracţie raţională

$$y_{*,k} = \frac{\beta_1 + \beta_2 z + \dots + \beta_n z^{n-1}}{-\alpha_1 + \alpha_2 z - \dots - \alpha_n z^{n-1} + z^n}\, u_{*,k}$$

numitorul $\neq 0$, numărătorul şi numitărul sînt polinoame prime.

Modelele 0,1, 1D, 2, 2D, 3, 3D, 4, 4D sînt echivalente, lucru pus în evidenţă prin următoarea diagramă comutativă [VA] :

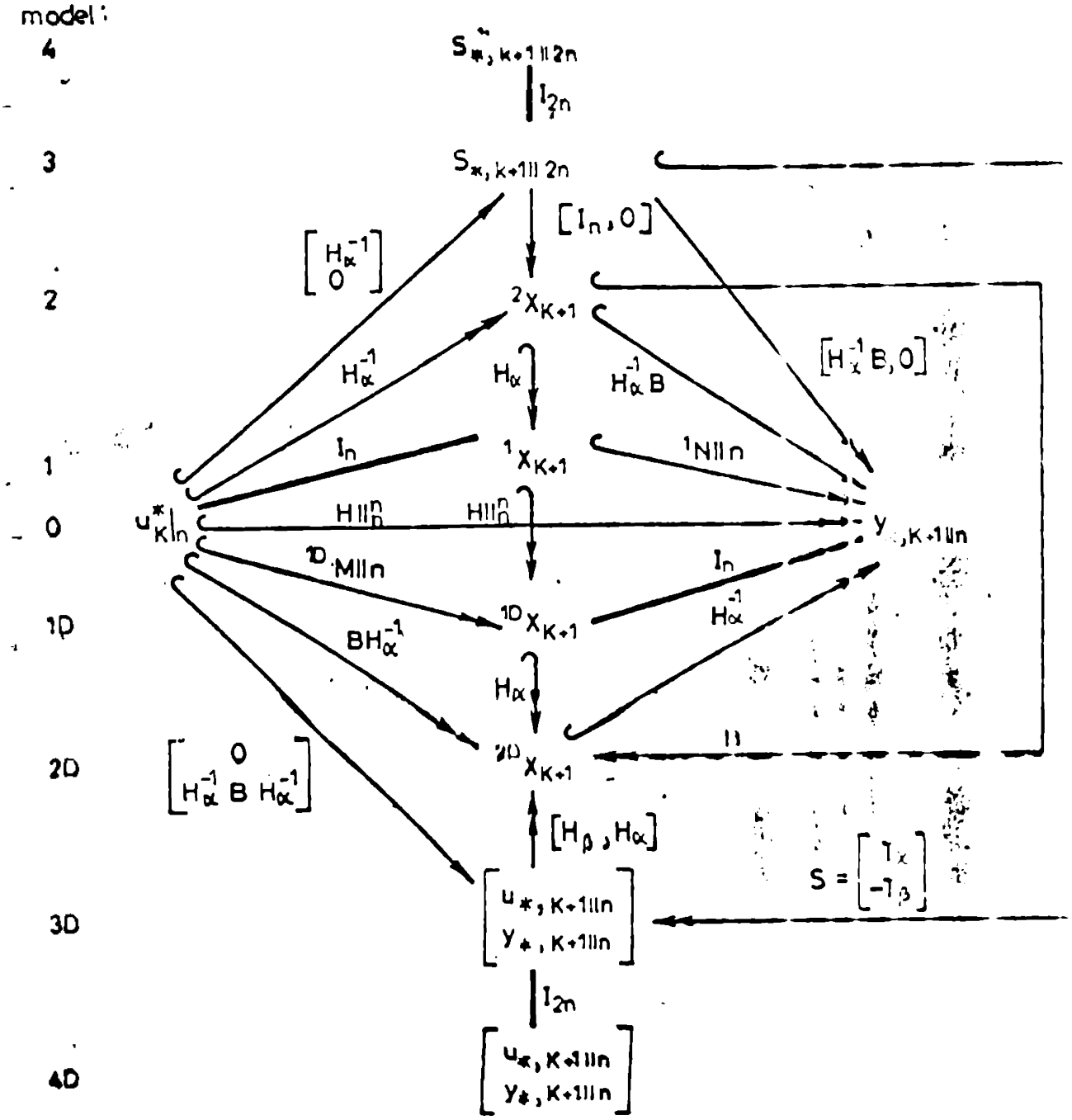


Fig. 6.3-1

UTILITATEA MODELELOR o,1,1D,2,2D,3,3D,4,4D IN ANALIZA COMPORTARII SA.

Modelul matricial o este modelul care pune cel mai pregnant în evidenţă cauzalitatea SA, motivează valabilitatea tuturor celorlalte modele finite. Spre deosebire de celelalte modele care sînt duale cîte două, acesta este autodual. Utilitatea lui teoretică este fundamentală, cea practică este mai limitată.

Modelele duale 1,1D evidenţiază în modul cel mai clar relaţia dintre răspunsul la impuls şi realizările minimale. Problemele ridicate de realizările minimale, realizările minimale aproximative îşi găsesc cea mai naturală rezolvare cu ajutorul modelelor pe stare de tip Hankel TSO , DYE , ICK . Aceste modele mai sînt cunoscute sub denumirea de forma normală de controlabilitate şi forma normală de observabilitate. ACK .

Modelul 2 asigură cele mai simple metode pentru calculul secvenţei de intrare necesare pentru atingerea unei stări finite stări, pentru calculul legii de reglare în sisteme cu reacţie. El este de asemenea foarte util în analiza SA cu reglare optimală KAL . In acest caz utilizarea modelului 2 permite simplificarea rezolvării ecuaţiei Riccati reducînd substanţial numărul variabilelor avute în vedere. Acest model este cunoscut şi sub denumirea de formă normală de reglare ACK .

Modelul 2D (forma normală pentru observator ACK) are avantajul că determinarea stărilor prin observator este mult simplificată dacă se utilizează această formă. Estimarea stării poate fi însoţită şi de realizarea reacţiei pe stare sau în alt fel analiza SA cu estimare şi compensare în cazul în care eroarea tinde asimptotic la zero. KAL , VAN . Această formă este utilă şi în cazul analizei comportării filtrelor Kalman (reconstrucţia optimală a stării), cît şi în analiza metodelor de identificare a parametrilor.

Modele 3 cuplează cel mai strînş datele măsurabile ale vectorilor de intrare şi ieşire (suprapuşi în timp) de problema realizării minimale; sînt mult utilizate în analiza problemelor de identificare şi predicţie BOD .

Modelele 4 (funcţii de transfer în z) sînt utilizate în analiza comportării externe, şi în analiza stabilităţii BJ. In ultima vreme teoria clasică a analizei în plan z este înlocuită de teoria polinoamelor formale KAL , ICK .

D. In limbajul categoriilor un sistem este specificat prin :

P- proces de intrare (înlocuieşte mărimea de intrare U)

δ: $XP \to X$ (generalizează dinamica $\delta : X \times U \to X$)

I un obiect al lui $\mathcal{K}$ denumit obiect al stării iniţiale (generalizează noţiune de variabilă iniţială de stare x_o)

ζ: $I \to X$ este un $\mathcal{K}$-morfism denumit morfism al stării iniţiale (specifică un set de stări iniţiale atagînd starea $\zeta(i)$ fiecărui i dintr-o mulţime de indici ai stărilor iniţiale). In mod obişnuit $I = \{o\}$ şi $\zeta(o) = x_o$ este starea iniţială.

Y este un obiect din $\mathcal{K}$ denumit obiect ieşire (generalizează noţiune de mulţime de ieşiri din Ens)

β: $X \to Y$ este un $\mathcal{K}$-morfism denumit morfism ieşire (generalizează noţiunea de funcţie de ieşire ($\beta : X \to Y$ din Ens).

Dacă se alege $\mathcal{K}$ = R - Mod=categoria R - modulelor, (.-dinamicile sînt funcţiile liniare:

δ : $XF \to X$, - $F : X \to X$, $x \to Fx$

P-dinamorfismele h : ${}^1X \to {}^2X$, $h^1F = h^2F$) atunci se obţine categoria în care evoluează sistemele liniare [ARB].

In acest caz se ajunge la definiţia clasică a sistemelor decompozabile:

D. Un sistem decompozabil este definit în limbajul categoriilor ca fiind

A : $X \to X$ o dinamică 1_X

B : $U \to X$.

C : $X \to Y$ A,B,C liniare.

pe scurt (A,B,C)

Comportarea unui sistem va putea fi studiată prin rezolvarea problemei realizării, care va conduce la obţinerea unui model care pă evidenţiază comportarea dorită.

D. Fie (A,B,C) un sistem din categoria cu dinamici liberă peste I,

δ_I : $(IP^*)\ P \to IP^*$

şi cu incluziunea generatorilor :

η_I : $I \to IP^*$

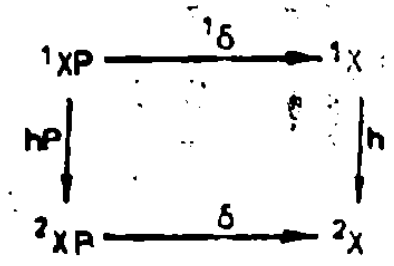


Fig. 5.3-2

In acest caz IP^* va fi obiectul intrărilor lui (A,B,C). Morfismul de accesibilitate

$$r : \dot{I}P^* \longrightarrow X$$

este unica extensie la un dinamorfism a morfismului de stări iniţiale $\xi : I \to X$, adică unicul $\mathcal{K}$- morfism r pentru care diagramele din fig. 5.3-3 comută:

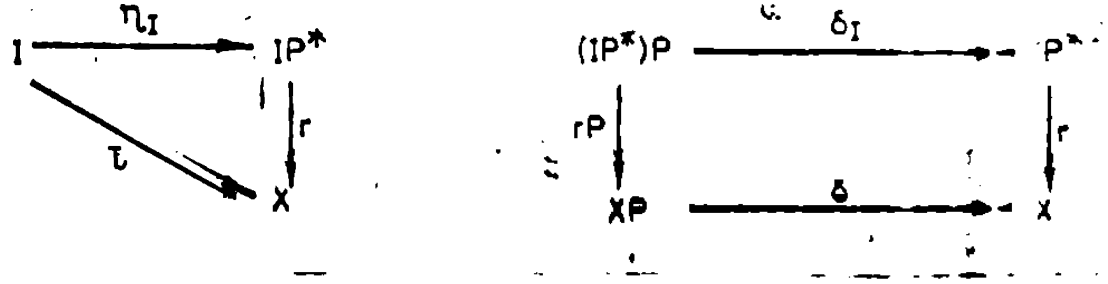


Fig. 5.3-3

Comportarea (sau răspunsul) lui (A,B,C) este morfismul

$\beta r : IP^* \longrightarrow Y$ adică

$$IP^* \xrightarrow{r} X \xrightarrow{\beta} X$$

Pentru I, P şi Y daţi orice $\mathcal{K}$-morfism

$$f : IP^* \longrightarrow Y$$

este un morfism comportare (răspuns). Se spune că un model (A,B,C) realizează pe f, sau este o realizare a lui f dacă şi numai dacă

$$f = \beta r \quad \text{pentru } (A,B,C)$$

Utilizând noţiunea de conucleu (coker) $[\alpha, \gamma]$, $[\alpha', \gamma']$, se poate defini coker - accesibilitatea:

D. Sistemul (A,B,C) din categoria K este coker - accesibil dacă şi numai dacă r:

$$IP^* \longrightarrow X$$

este un conucleu (α, γ) dacă diagrama din figură comută:

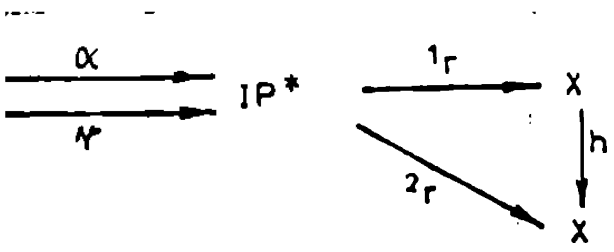


Fig. 5.3-4

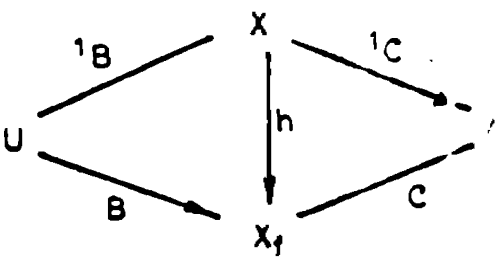


Fig. 5.3-5

In categoria K-Mod a sistemelor dinamice liniare orice funcţie surjectivă este un conucleu astfel încât orice diferen

liniar este poker - accesibil dacă este accesibil în sensul tradiţional.

Realizările sistemelor decompozabile

Dacă se dă un morfism comportare (răspuns):

$$f : U^{*} \longrightarrow Y$$

trebuie găsit sistemul $(A,B,C)_f$ care se constituie o realizare minimală accesibilă a lui f, în sensul că:

i) $(A.B.C)_f$ este o realizare a lui f,

$$f = C\, r_f$$

ii) Aplicaţia de accesibilitate

$$U^{*} \xrightarrow{r_f} X \text{ este un conucleu}$$

iii) $(A,B,C)_f$ este minimală în sensul că există un unic dinamorfism h care face comutativă diagrama din fig.

După cum s-a procedat în primul paragraf U^{*} se construieşte cu ajutorul funcţiilor in_j :

$$in_j : U \longrightarrow U^{*} : u \longmapsto (\ldots, o, u)$$

iar Y_{*} cu ajutorul funcţiilor :

$$\tilde{\pi}_k : Y_{*} \longrightarrow Y : (y_o, y_1, \ldots y_k, \ldots) \longmapsto y_k$$

astfel încît:

$$r : U^{*} \longrightarrow X$$

este unic definit prin ecuaţia:

$$r \cdot in_j = A^j B$$

în timp ce aplicaţia de observabilitate:

$$\sigma : X \longrightarrow Y_{*}$$

este unic definită prin:

$$\tilde{\pi}_n \cdot \sigma = BC^n$$

Răspunsul total, al sistemului este dat:

$$\tilde{f} = \sigma \cdot r : U^{*} \longrightarrow Y$$

iar comportarea sau răspunsul propriu-zis:

$$f = \tilde{\pi}_o \tilde{f} : U^{*} \longrightarrow Y$$

Reciproc dacă se dă $\tilde{f} : U^{*} \longrightarrow Y_{*}$, spaţiul stărilor realizării minimale este :

$$X_f = \tilde{f}\,(U^{*})$$

Dacă se notează $\tilde{f}\,(U^{*}) = Im(\tilde{f})$

se poate apela la factorizarea

$$U^{*} \xrightarrow{f} Y_{*} = U^{*} \xrightarrow{p} Im(\tilde{f}) \xrightarrow{i} Y_{*}$$

unde p este surjectivă (toate stările realizării minimale sînt accesibile) iar i este injectivă (toate stările realizării minimale sînt observabile).

Dacă se definește A_f astfel ca

$$A_f \cdot \tilde{f} = \tilde{f}_z$$

$$B_f = p \cdot in_o$$

$$C_f = \tilde{\pi}_o i$$

atunci $(A,B,C)_f$ este o realizare minimală a lui f.

Dacă se consideră categoria $\mathcal{M}$ ale cărei obiecte sînt sisteme decompozabile (A,B,C) cu dinamicile (X,A), aplicațiile de intrare $B : U \rightarrow X$ și aplicațiile de ieșire $C : X \rightarrow Y$ ale cărei morfisme sînt "simulările" h :

$$h : {}^1(A,B,C) \rightarrow {}^2(A,B,C)$$

$$h : {}^1X \rightarrow {}^2X ,$$

atunci diagrama din fig ... comută.

Categoria $\mathcal{M}$ permite generalizarea modelelor liniare monovariabile la intrare și la ieșire la sisteme multivariabile liniare, biliniare cît și la cazul automatelor finite [VAN]. Vaneeck introduce sistemul adjunct cu codinamica Δ :

$$XP \xrightarrow{\delta} X \xrightarrow{\Delta} XZ$$

functorii P și Z fiind adjuncți și arată că $\mathcal{M}$ - dinamorfismele sînt cazuri generale ale tuturor extensiilor de modele menționate.

Odată stabilit modelul discret el trebuie convertit într-un algoritm de calcul. Acest algoritm constă din trei tipuri de operații : adunare/scădere, înmulțire și întîrziere unitară. Diversele modele propuse pretind un număr diferit de operații aritmetice (timp) necesită cantități variabile de memorie în plus efectul erorilor de trunchiere afectează în mod particular fiecare alegere.

In urma studiilor făcute MAT , TZA , ACK în literatura de specialitate se consideră oportune următoarele realizări recursive: pentru modele de ordine reduse modelul 2, 2D, pentru modele de ordine mai mare realizări serie și cascadă cu elemente de bază de tip 2.

[illegible] [MAU] [illegible], luînd în considerare toate sursele de zgomot sînt minimizate.

6.4 Implementarea modelelor discrete pe procesoare cu unitate de comandă microprogramată

Pentru implementare s-a utilizat un procesor bazat pe familia I3000. Utilizînd aceleași principii, implementarea se

poate face utilizînd circuite şi tehnologii mai performante.

În continuare se trec succint în revistă caracteristicile esenţiale ale procesorului cu unitate de comandă microprogramată construit şi adaptat nevoilor aplicaţiilor de timp real a comportării SA. El constituie elementul de prelucrare principal al SMM al cărui MA a fost fundamentat în capitolele anterioare.

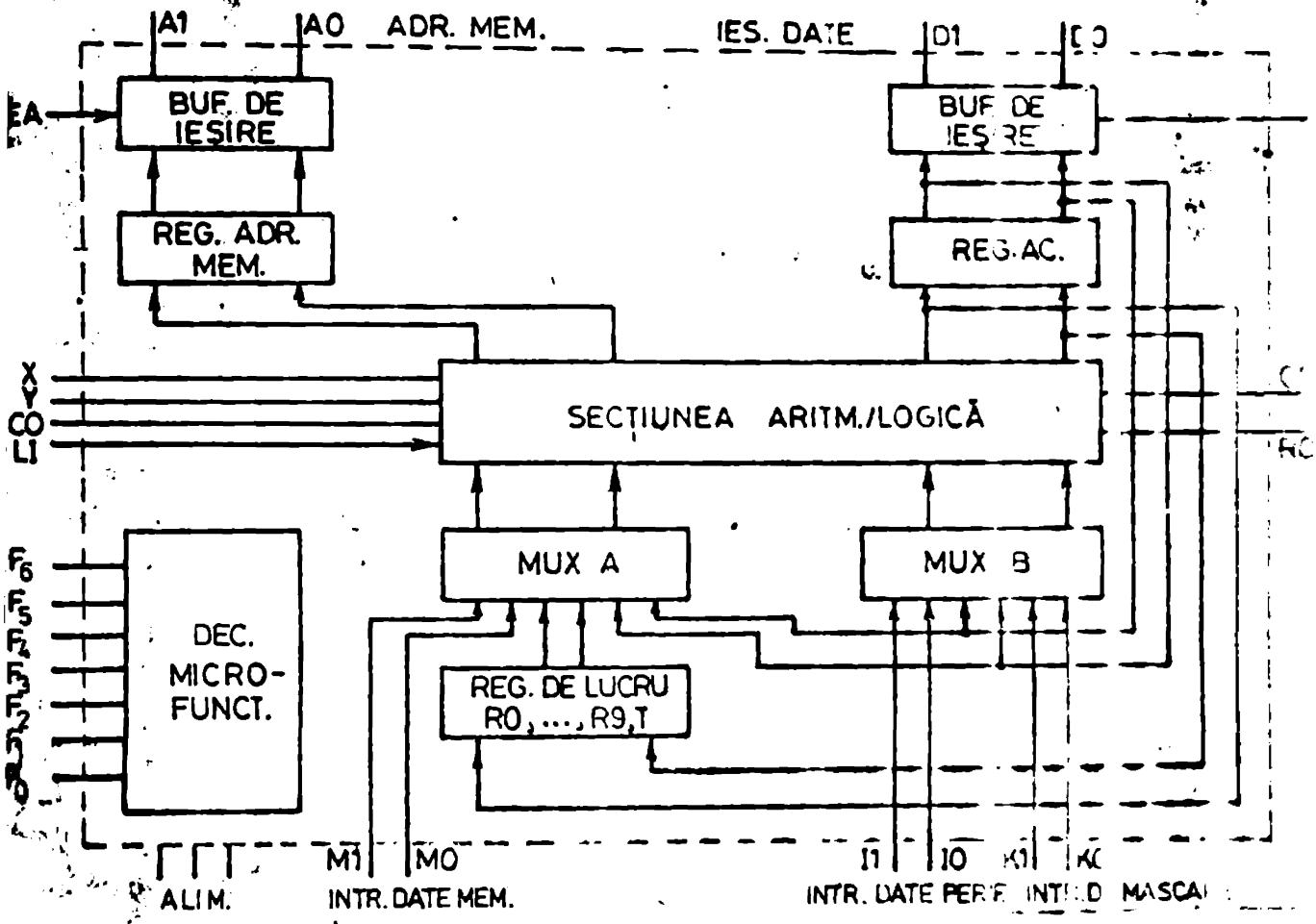
Unitatea de control al microprogramului comandă secvenţa de execuţie a microinstrucţiilor din memoria microprogram. Ea îndeplineşte următoarele funcţii: selecţia următoarei microinstrucţii pe baza conţinutului registrului de adresă microprogram; decodificarea şi testarea datelor furnizate prin ieşirile magistrale pentru a determina secvenţa de microinstrucţiuni; salvarea bitului de transport de la procesorul central; comanda intrării de transport sau deplasare a procesorului central; comanda întreruperii microprogramelor.

Elementul central de prelucrare I-3oo2 (EPC) conţine circuite ce reprezintă o "felie" cu lăţimea de 2 biţi a părţii de prelucrare a unui calculator numeric. Pentru a constitui un procesor complet pentru un cuvînt cu o lăţime dată de N biţi e necesar să se conecteze N/2 elemente de prelucrare. Un astfel de circuit de tipul I-3oo2 poate realiza următoarele operaţii: operaţii aritmetice; operaţii logice; incrementări/decrementări; generare de transport anticipat.

Elementul central de prelucrare I-3oo2 realizează funcţiile aritmetico logice şi de registru ale unei "felii" de 2 biţi dintr-un procesor central.

Datele de la surse exterioare cum ar fi memoria principală sînt aduse la elementul de prelucrare prin una din cele trei magistrale de intrare. Datele sînt trimise spre exterior prin cele 2 magistrale de ieşire. În interiorul elementului de prelucrare datele se memorează într-unul din cele 11 registre generale sau în acumulator (AC). Datele de pe magistralele de intrare, din unul din registrele generale sau din acumulator sînt transmise la intrarea unităţii aritmetice şi logice sub controlul a două multiplexoare interne. Intrările şi ieşirile suplimentare servesc la propagarea transportului la operaţii de deplasare şi la selecţia

microfuncţiei. Organizarea logică a elementului central de prelucrare se dă în fig. 54-1:



Magistrala microfuncţiilor este formată din 7 linii F0-F6 care sînt decodificate intern pentru a selecta funcţia unităţii aritmetice şi logice şi pentru a genera adresa registrului cu care se lucrează şi comanda multiplexoarelor A şi B.

Intrările magistralei M se folosesc pentru a aduce date de la memoria principală la elementul de prelucrare. Datele de pe această magistrală intră în unitatea aritmetică şi logică printr-un multiplexor.

Memoria rapidă conţine 11 registre notate R0-R9 şi T. Ieşirile registrelor sînt multiplexate divers pentru a fi introduse în unitatea aritmetică şi logică. Ieşirea unităţii aritmetice şi logice e dusă şi la intrarea memoriei rapide.

Pentru memorarea rezultatului unei operaţii aritmetice sau logice se foloseşte un registru separat numit acumulator (AC). Ieşirea acumulatorului e dusă printr-un multiplexor la intrarea în unitatea aritmetică şi logică şi de asemenea e dusă pe magistrala D printr-un buffer cu trei stări. In mod similar magistrala D se foloseşte pentru a transmite date spre memoria principală sau spre dispozitivele de intrare/ieşire.

Aceste multiplexoare selectează cele două intrări ale unităţii aritmetice şi logice conform specificaţiei de la magistrala microfuncţiei. Intrările multiplexorului A sînt : magistrala M, memoria rapidă (registrele) şi acumulatorul. Multiplexorul B selectează fie magistrala I fie acumulatorul fie magistrala K. De fiecare dată se efectuează un SI logic între informaţia selectată a multiplexorului B şi informaţia de pe magistrala K pentru a se putea realiza măscări şi testări de biţi.

Unitatea aritmetică şi logică realizează o serie de operaţii aritmetice şi logice : adunare în complement de doi, incrementare, decrementare, SI, SAU, complementare şi SAU-EXCLUSIV. Rezultatul operaţiei poate fi memorat în acumulator sau într-unul din registrele memoriei rapide. Liniile de intrare stînga (LI) şi ieşire dreapta (RO) se folosesc pentru realizarea operaţia de deplasare dreapta. Intrarea şi ieşirea de transport (CI şi CO) se folosesc pentru propagarea normală a transportului. CO şi RO sînt duse spre exterior prin buffere cu trei stări. In plus, mai sînt prevăzute ieşirile X şi Y pentru realizarea transportului anticipat la cuvinte de orice lungime. Posibilitatea de a masca intrarea unităţii aritmetice şi logice cu magistrala I măreşte flexibilitatea unităţii de prelucrare. La operaţiile aritmetice, circuitele de transport sînt folosite pentru realizarea operaţiei SAU între biţii cuvîntului mascaţi în mod corespunzător de magistrala K. In operaţiile aritmetice magistrala I se foloseşte pentru mascarea unor cîmpuri din operand. O altă funcţie realizată de magistrala K este aceea de a genera constante prin microprogram.

O ieşire separată a unităţii aritmetice şi logice este dusă de registrul de adrese care comunică cu magistrala I printr-un buffer cu trei stări. In mod convenţional registrul de adrese şi magistrala A se folosesc pentru a trimite adresa către memoria principală. Registrul de adrese şi magistrala A, se pot de asemenea folosi pentru selectarea unor dispozitive externe cînd se execută o operaţie de intrare/ieşire.

De-a lungul fiecărui microciclu la intrările F ale elementului central de prelucrare, se aplică o microfuncţie. Microfuncţia e decodificată, operanzii sînt selectaţi de către cele două multiplexoare şi operaţia specificată este executată de către unitatea aritmetică şi logică. Pe frontul coborîtor al tactului rezultatul se înscrie în acumulator sau în registrul selectat din memoria rapidă. In plus la unele operaţii datele referitoare la adresă sînt înscrise în

registrul de adresă al memoriei. O nouă microfuncţie se poate aplica doar după frontul crescător al tactului. Înainte tactul spre elementul de prelucrare în anumite microoperaţii acesta poate fi omis. Circuitele de transport şi deplasare sunt sincronizate impulsul de tact, ieşirile lor se pot folosi pentru a realiza o serie de testări nedistructive ale conţinutului registrului. Conţinutul registrelor nu se modifică decât în absenţa tactului. Microoperaţia ce se efectuează este determinată de grupul de funcţii şi grupul de registre selectate de informaţia de pe magistrala F. Grupul de funcţie este specificat de cei mai semnificativi trei biţi ai magistralei F, F4-F6 iar grupul de registre de cei mai puţini semnificativi patru biţi ai magistralei F, F0-F3. Grupul R se referă la registrele R0-R9, T şi AC notate R.. Grupurile RII şi RIII se referă doar la registrele T şi AC.

FUNCŢIILE CPE

Simbol		Funcţie CPE	CO	M.A.	CI
MOVE(Rn)		Rn AC	0 CO		
MOVA(Rn)	K	AC K Rn	1 CO		(-1
MOVM(AT)	K	M K AT	1 CO		(-1
MOVI(AT)	K	I K AT	1 CO		(-1
MARR(Rn)		Rn MAR	0 CO		
MARR1(Rn)		Rn MAR Rn+1 Rn	n.c. CO		
MARM(AT)		M MAR M AT	0 CO		
ADDE(Rn)	K	(AC K)+Rn Rn,AC	n.c. CO		(-1
ADDEC(Rn)	K	(AC K +Rn+C Rn, AC	n.c. CO		(-1
ADDR(Rn)	K	(AC K) +Rn Rn	n.c. CO		(-1
ADDRC(Rn)	K	(AC K) +Rn+C Rn	n.c. CO		(-1
ADDM(AT)	K	(AC K) +M AT	n.c. CO		(-2
ADDI(AT)	K	I K +AT AT	n.c. CO		(-1
KADR(Rn)	K	Rn K MAR Rn +K Rn	n.c. CO		(-1
KADM(AT)	K	M K MAR M +K AT	n.c. CO		(-1
INCE(Rn)		Rn+1 Rn,Ac	n.c. CO		
INCR(Rn)		Rn+1 Rn	n.c. CO		
INCM(AT)		M+1 AT	n.c. CO		
DECA(Rn)		AC-1 Rn	n.c. CO		-1
DECT(AT)		AT-1 AT	n.c. CO		-1
DECI(AT)		I-1 AT	n.c. CO		-1
DECR(Rn)		Rn-1 Rn -1 MAR	n.c. CO		-1
ANDR(Rn)	K	AC K Rn Rn	l.r. CO		(-1
ANDM(AT)	K	AC K M AT	l.r. CO		(-1
ANDI(AT)	K	I K AT AT	l.r. CO		(-1
IORR(Rn)	K	(AC K) Rn Rn	AC K CC		(-1
IORM(AT)	K	(AC K) M AT	AC K CC		(-1
IORI(AT)	K	(I K) AT AT	I K CC		(-1
XNRR(Rn)	K	(AC K) + Rn Rn	AC K CC		(-1
XNRM(AT)	K	(AC K) + M AT	AC K CC		(-1
XNRI(AT)	K	(I K +AT AT	I K CC		(-1
MSKR(Rn)	K	Rn K Rn	l.r. CO		(-1
MSKM(AT)	K	M K AT	l.r. CO		(-1
TSTR(Rn)	K	Rn K Rn	l.r. CO		(-1
TSTM(AT)	K	M K AT	l.r. CO		(-1
TSTI(AT)	K	I K AT AT	l.r. CO		(-1
NOTR(Rn)		Rn Rn	0 CO		
NOTM(AT)		M AT	0 CO		

NEGT(AT)		AT+1 AT	a.c.	CO	
CLRR(Rn)	CI	0 Rn	l.r.	CO	(0)
SETR(Rn)		-1 Rn	l.r.	CO	(-1)
SHRT(AT)	LI	LI AT_H AT_H AT_L AT_L RO	AT_1	0	(mod)

a.c. = transport aritmetic

l.r. = rezultat logic

Structura cuvîntului instrucţie. Instrucţiile au fost codificate prin cîte unul sau două cuvinte de 16 biţi. În cadrul primului cuvînt, octetul superior reprezintă codul operaţiei. Codul operaţiei se aplică la intrările CX, PI ale circuitului I3001, care reprezintă unitatea de control al microprogramului, şi este folosit pentru intrarea într-o secvenţă de microinstrucţii specifică fiecărei instrucţii. Acest lucru este ilustrat în fig. în care se prezintă organizarea structurii de control microprogramate şi fluxul de informaţii în cadrul ei.

Structura cuvîntului microinstrucţie este prezentată în fig. După cum reese din această figură, se lucrează după principiul microprogramării orizontale, printr-o microinstrucţie fiind programate operaţii care sînt executate în paralel de către mai multe resurse ale sistemului, lucru evidenţiat clar în fig.

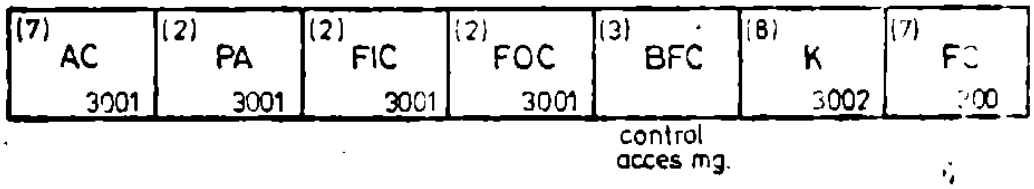


Fig. 5.4-2

Cîmpul AC (Adress control) este interpretat în circuitul I3001 şi folosit pentru determinarea adresei următoarei microinstrucţii, în unele situaţii el contribuind şi la stabilirea codului registrului general vizat în circuitul I3002.

Cîmpul PA (Page adress) reprezintă adresa paginii de memorie de microprogram, memorie care este organizată pe 4 pagini. Această organizare permite o flexibilitate mai mare de codificare în memoria de microprograme şi a fost adoptată pentru că, deşi există 3 biţi neutilizaţi în cuvîntul microinstrucţie, lungimea fizică a acestuia fiind, inevitabil, egală cu 32 biţi.

Cîmpul FIC (Flag in control) realizează controlul poziţionării fanioanelor C şi Z ale circuitului I3oo2 cu informaţia de pe linia de intrare FI.

Cîmpul FOC (Flag out control) realizează controlul informaţiei care se plasează pe linia FO şi e folosită drept intrare stînga (LI) sau intrare transport (CI) pentru circuitul I3oo2, în funcţie de operaţia care se execută în acest circuit.

Cîmpul BCF (Bus control field) poartă o informaţie folosită pentru controlul magistralelor sau pentru inhibarea tactului circuitului I3oo2 în timpul operaţiilor de testare a conţinutului unui registru.

Cîmpul K (Masca) reprezintă codul explicit la intrarea de mascare al circuitului I3oo2.

Cîmpul FC (Function control) reprezintă codul funcţiei pe care o execută I3oo2 şi codul registrului la care se referă aceasta.

Pentru a micşora volumul de memorie de microprograme utilizată şi mai ales pentru a conferi maşinii emulate viteză, s-a adoptat o structură cu registre pipe-line pe două nivele. In consecinţă, la un moment dat sînt în execuţie operaţii programate prin trei microinstrucţii.

Din punct de vedere al numărului de faze, structura adoptată este monofazică.

Avînd în vedere modul de selectare a registrelor circuitului I3oo2 şi modul de codificare a sursei de comandă respectiv a destinaţiei rezultatului la microprocesorul 3c3o, a fost aleasă următoarea alocare a registrelor: R0=B, R1=C, R2=D, R3=E, R4=H, R5=L, R6 = registrul de memorie, R7=A, R8=PC, R9=SP, T=(P),AC = registrul acumulator pentru microprocesor

5.4.1 Instrucţiile speciale MM ale unităţii microprogramate.

Unitatea microprogramată (UM) pe 16 biţi proiectată să reprezintă elementul de prelucrare de bază al MA propus.

Fiecare astfel de UM are implementat un set de instrucţii de bază (emularea instrucţiilor Intel 8086) şi un set de instrucţii speciale de modelare în timp real bazate pe modelele 1,... 4 analizate. Cele mai utile şi performante sînt cele prevăzute modelele de tip 2 şi 4 ale MM. Aceste instrucţiuni sînt de tip specializat în octetul superior codul operaţiei, urmînd ca specificarea adresei tabelei cu parametrii modelului. Algoritmii modelului 2 sînt comuni şi modelului 4 (ordinul n = 2)

$$y_k = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \; u_k$$

Modelul de tipul 2 corespunzător este :

$$\begin{matrix} x_1 \\ x_2 \end{matrix}_{k+1} = \begin{matrix} 0 & 1 \\ -b_2 & b_1 \end{matrix} \begin{matrix} x_1 \\ x_2 \end{matrix}_k + \begin{matrix} 0 \\ 1 \end{matrix} u_k ;$$

$$y_k = \begin{matrix} a_2 - a_1 b_2, & a_1 - a_0 b_1 \end{matrix} \begin{matrix} x_1 \\ x_2 \end{matrix}_k + a_0 u_k .$$

Implementarea instrucţiei SIM2 corespunzătoare este dată la nivel de microprogram în fig.

Pentru simularea MA de ordinul n = 2 se utilizează posibilităţile de prelucrare paralelă ale MA propus.

Modelul 4 este descompus sub forma :

$$y_k = H_0 + H_1(z) + H_2(z) + \ldots H_n(z)$$

în care H_0, H_1, H_2 avînd ordinul n = 2 pot fi simulate în paralel pe UM ale MA. Prelucrarea concurentă conduce la un spor de viteză proporţional cu numărul de procesoare, grevat însă de necesităţile de sincronizare. În acest scop numărul şi dimensiunile schimburilor de informaţii între elementele de prelucrare a fost ele următorii parametrii: u_k, x_{1k}, x_{2k} şi y_k.

Pentru a facilita o implementare avantajoasă a setului de instrucţii al procesorului, o emulare a instrucţiunilor uP 18086 MURA - şi în special a instrucţiunii de simulare numerică SIM2, prin prisma vitezei de calcul şi costului (dictat cu precădere de volumul memoriei de microprograme) resursele "firmware" ale UM au fost completate cu resurse "hardware", aşa cum se evidenţiază

în fig. la care se vor face referiri în continuare, considerîndu-se, ilustrativ, instrucţiunea SIM2. În mod similar se implementează SIM1, SIM3, SIM4.

După cum se va vedea algoritmul de realizare numerică implementat implică efectuarea unor operaţii de însumare algebrică şi de înmulţire. Întrucît pentru însumare este avantajoasă reprezentarea numerelor în complement de 2, iar înmulţirea se face mai eficient (mai ales în cazul unei structuri bazată pe familia de circuite I3000) dacă operanzii sînt reprezentaţi în semn - mărime, sînt folosite în mod corespunzător ambele aceste reprezentări.

Notaţiile utilizate în continuare au următoarea semnificaţie:

AC = controlul adresei ; ACMI = controlul adresei microinstrucţiilor; APU = adresa paginii următoare, AP = adresa de pagină ; RPAP = registru pipe line pentru adresa de pagină; AMI = adresa microinstrucţiei; BMAMI = bloc de modificare a adresei de microinstrucţie; MMP = memorie de microprogram; RPCPI = registru pipeline pentru controlul PI; RPCPO = registru pipeline pentru controlul PO; RPLSC, RP2SC = registre pipeline pentru sincronizare şi control; BSC = bloc sincronizare şi control; RPCF = registru pipeline pentru controlul funcţiei; BMF = bloc de modificare a funcţiei; RPCK = registru pipeline pentru controlul măştilor; LA = latch-uri auxiliare; CA = contor auxiliar; ROD = registru deplasare dreapta; DFS = detector funcţii speciale; MD = magistrală date; MA = magistrală de adrese; MC = magistrală de control.

Circuitul "latch" auxiliar LA este destinat să păstreze, în general, primul cuvînt din cadrul instrucţiunii curente, pentru a putea fi utilizat la modificarea adresei de microinstrucţiune generată de MCU şi/sau a funcţiei (de regulă a codului registrului referit în cadrul ei) microprogramată pentru CPE; în cazul instrucţiunii SIM2, circuitul LA păstrează înmulţitul curent.

Registrul de deplasare dreapta ROD păstrează, în cazul instrucţiunii SIM2, înmulţitorul curent.

Contorul auxiliar CA este destinat să numere paşii în cadrul operaţiei de înmulţire; el este iniţializat cu conţinutul nul, iar la sfîrşitul unei înmulţiri, este lăsat în starea iniţială.

Detectorul funcţiilor speciale DFS are rolul de a sesiza prezenţa în execuţie a funcţiilor CLA , CLA (identice "firmware funcţia în CLA, utilizată normal, în speţă pentru anularea conţinutului registrului AC bibliografie) şi de a concura în lansarea operaţiilor suplimentare specifice lor, şi anume:

- închiderea în circuitul "latch" a informaţiei prezentă la intrările sale, sincronizat cu semnalul de tact MCPECL, pentru funcţia CLA;

- încărcarea registrului de deplasare RDD cu informaţia prezentă la intrările sale, reprezentînd înmulţitorul în cadrul instrucţiunii ARN22, forţat cu semn negativ, pentru funcţia CLA

- validarea impulsurilor de deplasare a conţinutului RDD şi de numărare în CA, în cazul funcţiei AIA, a cărei semantică este adunarea în registrul AC a conţinutului acestuia cu rezultatul funcţiei "SI" între informaţiile de pe magistralele I şi K; se precizează că în timpul execuţiei funcţiei AIA, dacă bitul cel mai puţin semnificativ curent al RDD este "I" , se dezactivează registrul pipe-line RPCK, ceea ce implică anularea măştii K.

Se menţionează că la atingerea conţinutului 15_{10} în CA, se va modifica, în BMAMI, adresa microinstrucţiunii următoare generată de MCU, astfel încît să se părăsească forţarea ciclică a secvenţei de înmulţire curentă.

Blocul de sincronizare şi control BSC asigură sincronizarea tuturor activităţilor şi resurselor şi generează semnalele de comandă şi control.

Detalii privind funcţionarea UM sînt specificate în protocoalele contractului "Sisteme electrice de transport cu motoare liniare" nr. 198 din 1983, 1984 şi 1985 IPA-CCSIP - IP Craiova, secţiunea privind proiectarea, realizarea şi testarea unui procesor cu unitate microprogramată bazat pe familia de componente I3ooo.

5.4.2 Analiza comparativă a algoritmilor de simulare a comportării sistemelor automate implementaţi pe diverse procesoare de 16 biţi

Cei trei parametrii semnificativi vizaţi pentru evaluarea algoritmilor de prelucrare în timp real sînt : timpul de execuţie, cerinţele de memorie, precizia.

Au fost luaţi în considerare următorii parametri pentru cele mai populare procesoare pe 16 biţi : Z 8000 (4 Mhz) 8086 (5 Mhz) varianta standard şi I 3000 pe 16 biţi şi un procesor bit-sliced bazat pe familia I 3000 cu modificările aduse de autor.

Pentru sporirea vitezei de lucru s-a lucrat numai cu operaţii cu numere întregi (virgulă fixă), aritmetica cu virgulă mobilă mărind durata de execuţie de cel puţin trei ori.

Se consideră că în ipotezele prezentate precizia calculelor este satisfăcătoare. Comparaţiile între timpii de execuţie şi necesarul de memorie a fost făcut pentru următorii algoritmi RAJ , pentru câte o iteraţie:

i)Algoritm Kalman :

$$H(z) = \frac{1 - a z^{-1}}{K \cdot 0 \quad 1 - (c_1/C)\, z^{-1} + (c_2/C) z^{-2}}$$

ii) Algoritm de reglare :

$$H(z) = \frac{a_0 + a_1 z^{-1}}{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}$$

iii) Filtru trece-jos Butterworth:

$$H(z) = \frac{b (z+1)^4}{(z-p_1)(z-p_2)(z-p_3)(z-p_4)}$$

Timpii de execuţie şi spaţiul de memorare necesari pentru o iteraţie:

Procesor / Algoritm	I 8086 5 Mhz	Z 8000 4 Mhz	I 3000 6 Mhz	I 3000 *** 6 Mhz	SIM *** (3 x 4 x 2)
i) **	250 us 130 oct	171,9 us 128 oct	101,6 us 22 oct 598 ****	58,6 us 22 oct 340 ****	58,6 us 22 oct
ii) **	637 us 294 oct	445,2 us 308 oct	263,2 us 48 oct	117,2 us 48 oct	58,6 us 52 oct
iii)	5380,9 us 3114 oct *	3042,8 us 2896 oct *	406,4 us 100 oct **	234,4 us 100 oct **	130,6 us 124 oct **

* aritmetică în virgulă mobilă;

** aritmetică în virgulă fixă ;ş

*** unitate centrală cu instrucţia SIM microprogramată, şi cu înmulţirea cablată ;

**** numărul de microinstruoţii ale instruoţiei SIM.

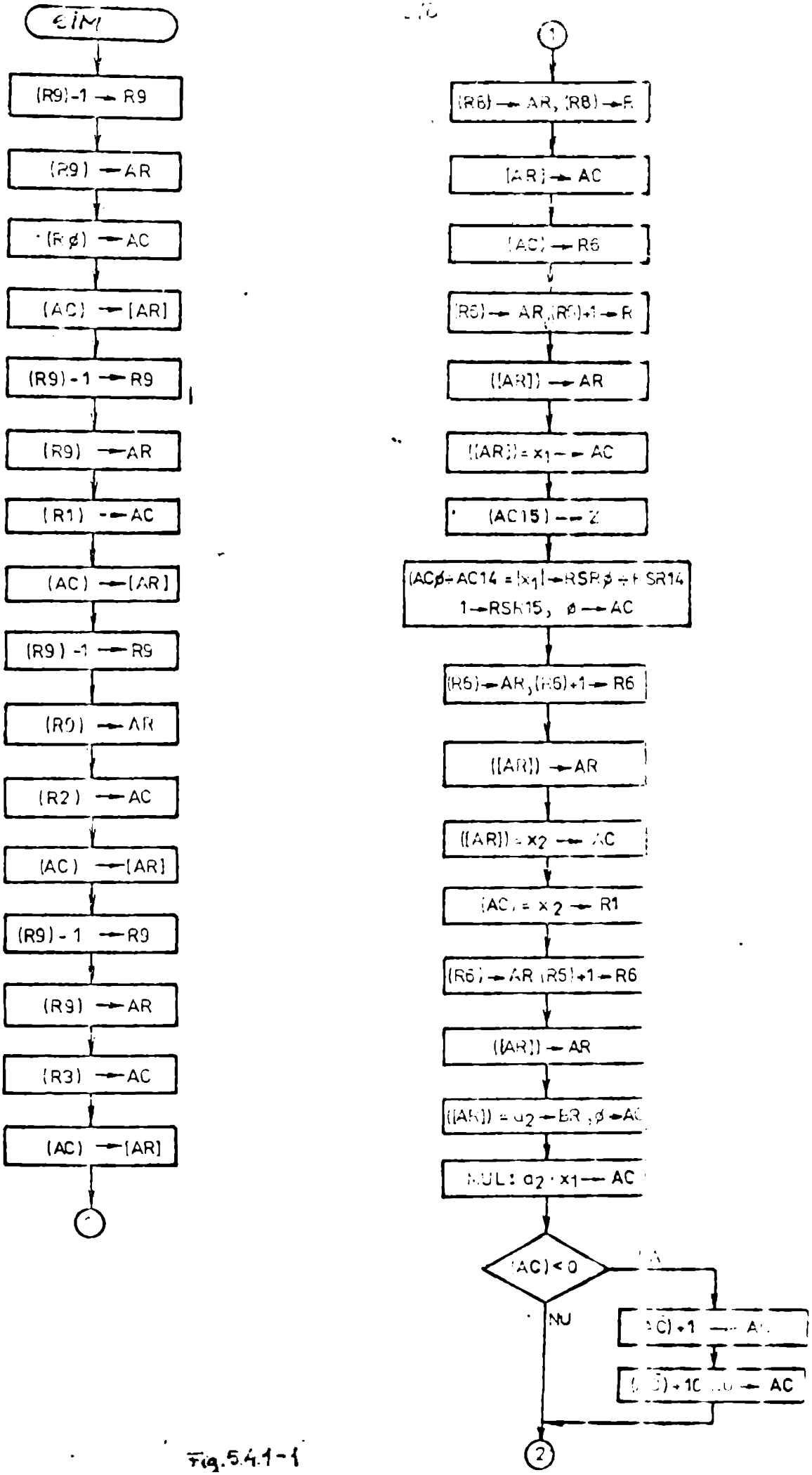
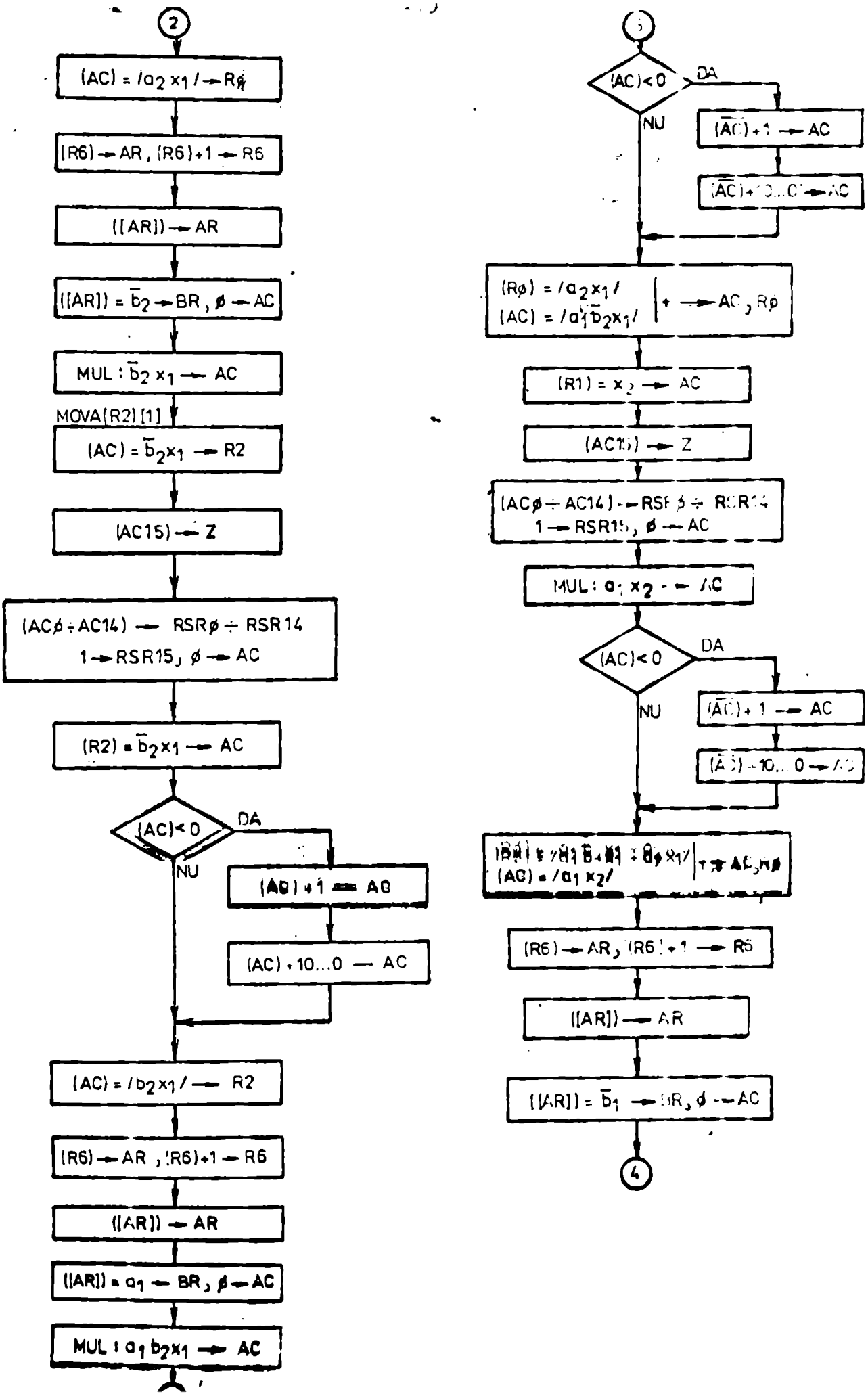
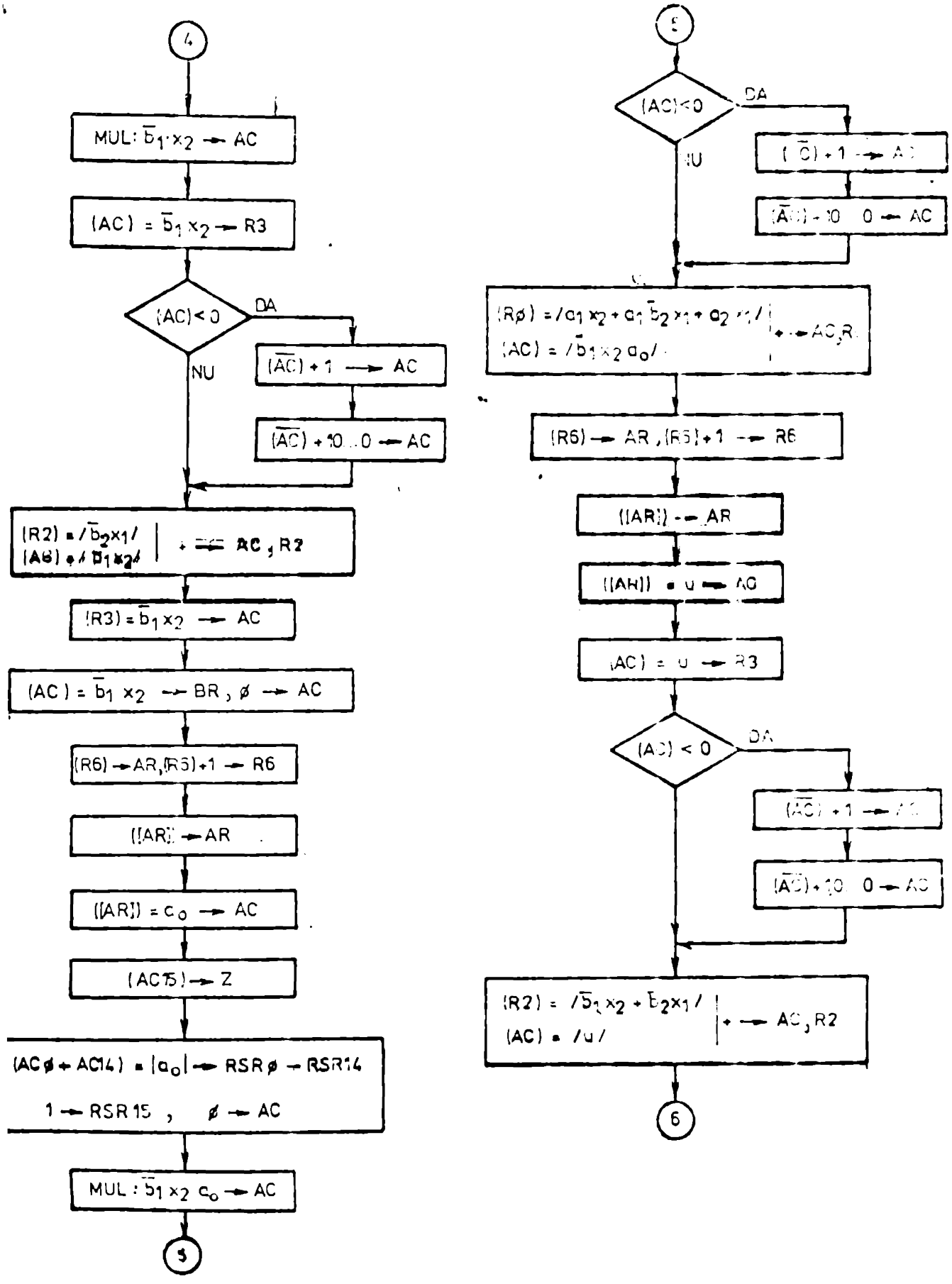
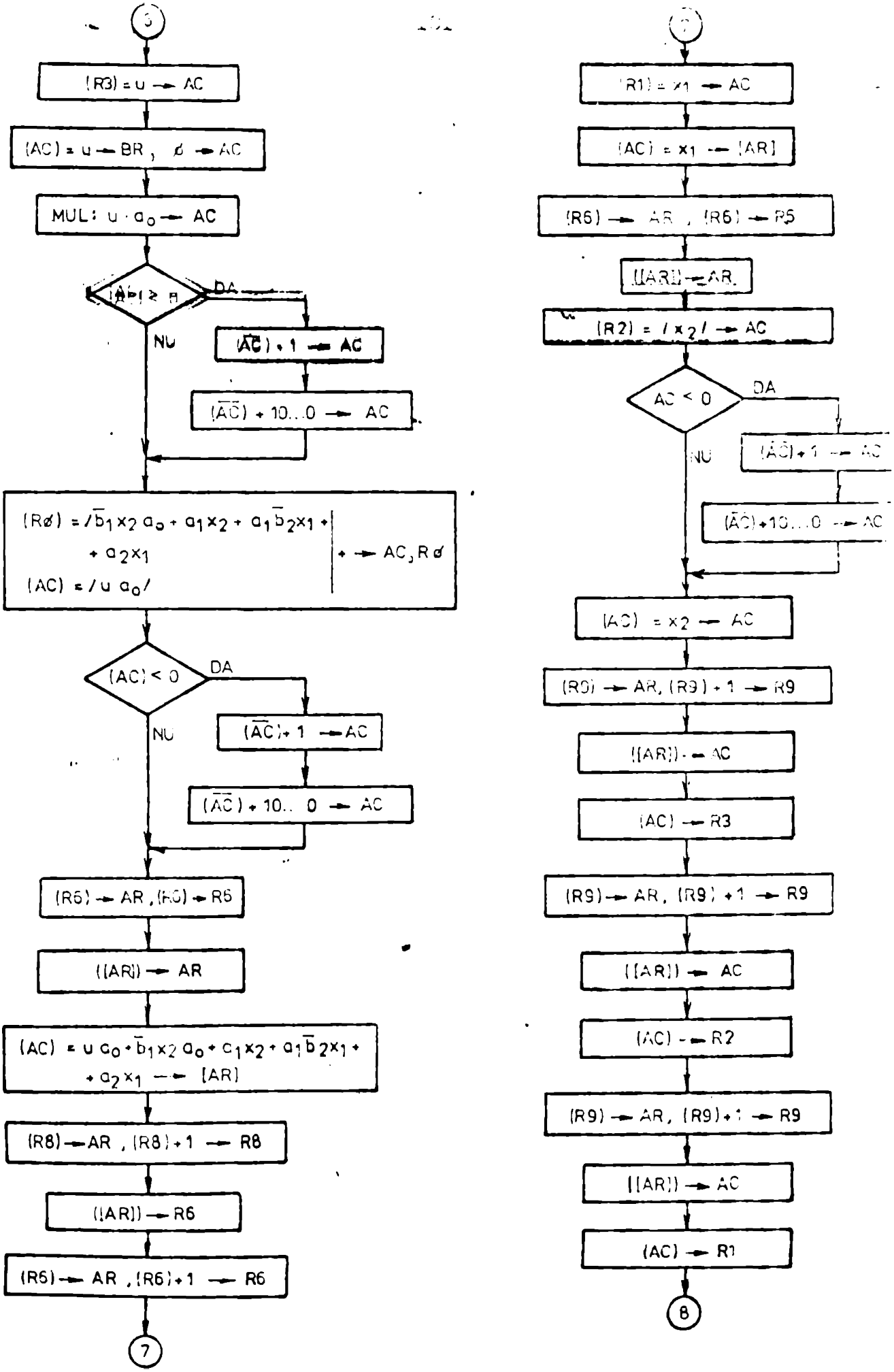
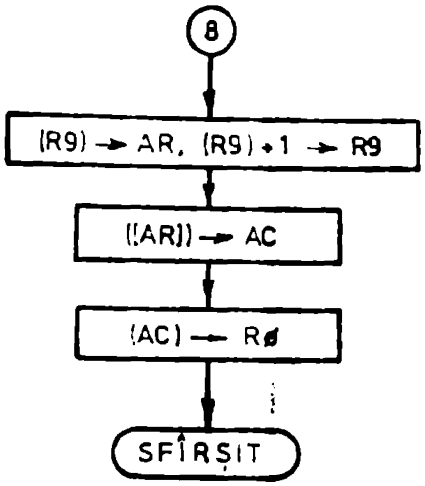


Fig.5.4.1-1


2
(AC) = /a2 x1 / → Rø
(R6) → AR, (R6)+1 → R6
([AR]) → AR
([AR]) = b̄2 → BR, ø → AC
MUL : b̄2 x1 → AC
MOVA(R2)(1)
(AC) = b̄2x1 → R2
(AC15) → Z
(ACø ÷ AC14) → RSRø ÷ RSR14
1 → RSR15, ø → AC
(R2) = b̄2x1 → AC
(AC)< 0
DA
NU
(AC) + 1 → AC
(AC) + 10...0 → AC
(AC) = /b2x1/ → R2
(R6) → AR, (R6)+1 → R6
([AR]) → AR
([AR]) = a1 → BR, ø → AC
MUL : a1 b2x1 → AC
3
(AC)< 0
DA
NU
(AC) + 1 → AC
(Rø) = /a2x1/
(AC) = /a1b̄2x1/ + → AC, Rø
(R1) = x2 → AC
(AC15) → Z
(ACø ÷ AC14) → RSRø ÷ RSR14
1 → RSR15, ø → AC
MUL : a1 x2 → AC
(AC)< 0
DA
NU
(AC) + 1 → AC
(AC) = /a1 x2/
(R6) → AR, (R6) + 1 → R6
([AR]) → AR
([AR]) = b̄1 → BR, ø → AC
4


4
MUL: $\bar{b}_1 x_2 \rightarrow$ AC
(AC) = $\bar{b}_1 x_2 \rightarrow$ R3
(AC) < 0
DA
NU
$(\overline{AC}) + 1 \rightarrow$ AC
$(\overline{AC}) + 10\ldots0 \rightarrow$ AC
(R2) = $/\bar{b}_2 x_1/$
(AC) = $/\bar{b}_1 x_2/$
$+ \rightarrow$ AC, R2
(R3) = $\bar{b}_1 x_2 \rightarrow$ AC
(AC) = $\bar{b}_1 x_2 \rightarrow$ BR, $\emptyset \rightarrow$ AC
(R6) $\rightarrow$ AR, (R6) + 1 $\rightarrow$ R6
((AR)) $\rightarrow$ AR
((AR)) = $c_0 \rightarrow$ AC
(AC15) $\rightarrow$ Z
(AC$\emptyset$ ← AC14) = $|a_0| \rightarrow$ RSR$\emptyset$ → RSR14
1 → RSR15, $\emptyset \rightarrow$ AC
MUL: $\bar{b}_1 x_2 c_0 \rightarrow$ AC
5
(AC) < 0
DA
NU
$(\overline{C}) + 1 \rightarrow$ AC
$(\overline{AC}) + 10\ldots0 \rightarrow$ AC
(R$\emptyset$) = $/a_1 x_2 + a_1 \bar{b}_2 x_1 + a_2$
(AC) = $/\bar{b}_1 x_2 a_0/$
$+ \rightarrow$ AC, R
(R6) $\rightarrow$ AR, (R6) + 1 $\rightarrow$ R6
((AR)) $\rightarrow$ AR
((AR)) = $\emptyset \rightarrow$ AC
(AC) = $\emptyset \rightarrow$ R3
(AC) < 0
DA
$(\overline{AC}) + 1 \rightarrow$ AC
$(\overline{AC}) + 10\ldots0 \rightarrow$ AC
(R2) = $/\bar{b}_1 x_2 + \bar{b}_2 x_1/$
(AC) = $/a_0/$
$+ \rightarrow$ AC, R2
6


5
(R3) = u → AC
(AC) = u → BR , ø → AC
MUL: u · a0 → AC
|AC| ≥ B
DA
NU
(AC) + 1 → AC
(AC) + 10...0 → AC
(Rø) = /b̄1 x2 a0 + a1 x2 + a1 b̄2 x1 + + a2 x1
(AC) = /u a0/
+ → AC, Rø
(AC) < 0
DA
NU
(AC) + 1 → AC
(AC) + 10... 0 → AC
(R6) → AR , (R6) → R6
((AR)) → AR
(AC) = u a0 + b̄1 x2 a0 + a1 x2 + a1 b̄2 x1 + + a2 x1 → (AR)
(R8) → AR , (R8) + 1 → R8
((AR)) → R6
(R6) → AR , (R6) + 1 → R6
7
7
(R1) = x1 → AC
(AC) = x1 → (AR)
(R6) → AR , (R6) → R5
((AR)) → AR
(R2) = /x2/ → AC
AC < 0
DA
NU
(AC) + 1 → AC
(AC) + 10...0 → AC
(AC) = x2 → AC
(R9) → AR, (R9) + 1 → R9
((AR)) → AC
(AC) → R3
(R9) → AR, (R9) + 1 → R9
((AR)) → AC
(AC) → R2
(R9) → AR, (R9) + 1 → R9
((AR)) → AC
(AC) → R1
8



Zonă adrese parametrii	
ZAP	adr. x_1
+1	adr. x_2
+2	adr. a_2
+3	adr. $-b_2$
+4	adr. a_1
+5	adr. $-b_1$
+6	adr. a_0
+7	adr. u
+8	adr. y

Fig. 5.4.1-2

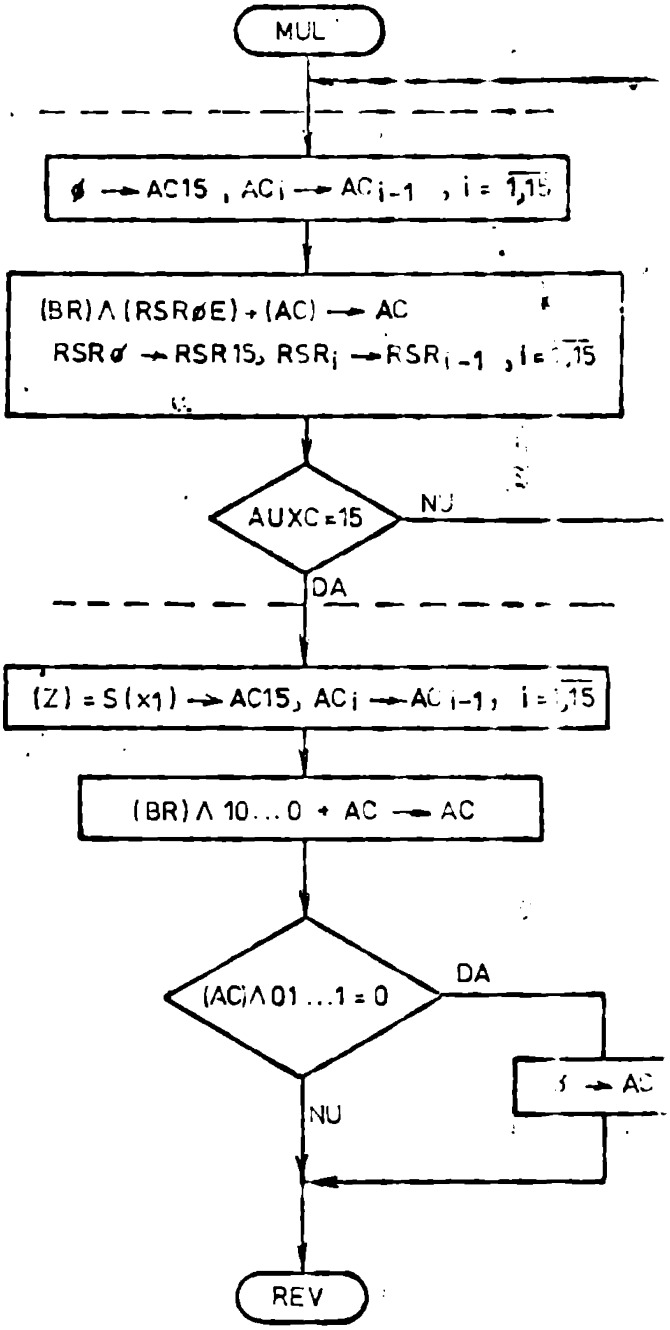


Fig 5.4.1-3

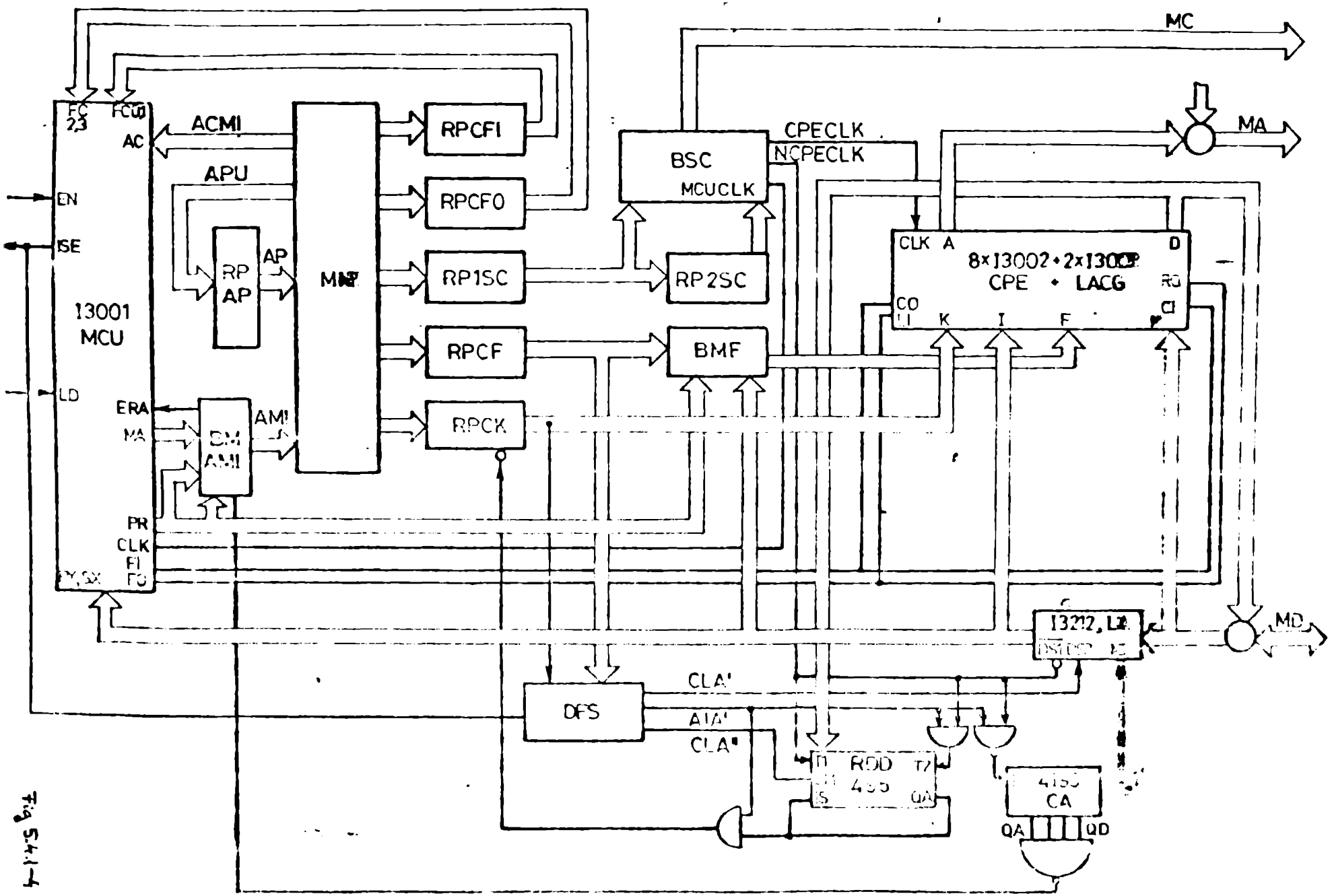


Fig 5.4.1-4

6 CONCLUZII

Scopul acestei lucrări este acela de a-şi aduce contribuţia, într-o manieră originală la conceperea arhitecturilor multimicroprocesor facilitate de cele mai moderne realizări în domeniuş magistralelor pe plan naţional şi mondial (Multibus I, VMEBUS - existente pe piaţă, MULTIBUS II, NUBUS, FUTURE BUS anunţate pentru sfîrşitul anului 1985) DAL , la evaluarea performanţelor acestor arhitecturi şi la utilizarea lor în domeniul analizei performanţelor SA. In acest sens au fost urmărite următoarele obiective :

1) Analiza şi clasificarea arhitecturilor multiprocesor utilizate în aplicaţiile în timp real.

2) Asigurarea bazei teoretice a modelării arhitecturilor multimicroprocesor în scopul evaluării performanţelor acestora.

3) Elaborarea unor modele analitice aproximative suficient de riguroase şi a unor modele de simulare care să permită proiectanţilor o alegere judicioasă a unor arhitecturi în funcţie de cerinţele impuse de aplicaţia specifică.

4) Alegerea modelelor matematice celor mai adecvate simulări în timp real a comportării SA.

6.1. CONTRIBUTII ORIGINALE

Pornind de la scopul şi obiectivele propuse se pot scoate în evidenţă următoarele contribuţii originale:

- Definirea structurii nivelurilor arhitecturale pe orizontală şi a subdiviziunilor verticale separînd componentele unui sistem multimicroprocesor utilizat în analiza în timp real a comportării SA.

- Sistematizarea într-o formă unitară a tendinţelor ce se manifestă în domeniul prelucrării în timp real utilizînd sisteme multimicroprocesor.

- Introducerea unui model arhitectural deosebit de adaptabil la cerinţele analizei comportării SA dar suficient de restrictiv în a-şi păstra caracteristicile de prelucrare în timp real.

- Stabilirea condiţiilor de aplicabilitate a diverselor metode şi modele de evaluare a performanţelor la cazul particular al structurilor multimicroprocesor.

- Utilizarea mecanismului de rendez-vous pentru elaborarea unui executiv pentru MA propus.

- Definirea unei relaţii de ordine totală pe mediul evenimentelor, legată de utilizarea secţiunilor critice asociate variabilelor sistem.

- Studiul cu ajutorul reţelelor de evaluare a proceselor de sincronizare bazate pe tehnica de rendez-vous.

- Introducerea relaţiei de echivalenţă pe mulţimea stărilor lanţului Markov exact asociat arhitecturii analizate.

- Definirea stării lanţurilor Markov comasate pe mulţimea cît S/ (S = mulţimea stărilor lanţului exact; = relaţie de echivalenţă).

- Introducerea unui criteriu (condiţii suficiente) care stabileşte condiţiile în care aplicaţia f : S S/ menţine proprietatea Markov a lanţului comasat (f omomorfism între S şi S/E).

- Introducerea unei strategii de simulare a interacţiunilor, utilizînd modele cu evenimente discrete, implementată prin algoritmi GPSS.

- Stabilirea ipotezelor de modelare şi a parametrilor MA şi alegerea indicilor de performanţă corespunzători.

- Utilizarea în condiţiile stabilite de criteriul de suficienţă, a lanţurilor Markov comasate pentru modelarea arhitecturilor MA.

- Utilizarea reţelelor de evaluare în modelarea arhitecturilor MA.

- Alegerea modelelor matematice cele mai adecvate pentru studiul comportării SA şi studiul domeniului lor de aplicabilitate prin analiza în domeniul limbajului categoriilor a generalităţii şi cuprinderii lor.

- Elaborarea unei proceduri de simulare discretă a modelelor invariante în timp.

- Implementarea acestei proceduri prin intermediul unor instrucţii specializate SIM pe procesoare cu unitate microprogramată. (UM).

- Elaborarea unor îmbunătăţiri hardware la firmware I 3ooo care să permită implementarea cu mare viteză a instrucţiilor SIM.

- Realizarea pe sistemul multi microprocesor dotat cu UM proiectată, a unor timpi de simulare a comportării SA superiori celor menţionaţi în literatura de specialitate.

6.2. VALOAREA APLICATIVA SI DIRECTII DE DEZVOLTARE VIITOARE

Aspectele prezentate în lucrare au fost valorificate în cadrul a 26 lucrări ştiinţifice din care una "Conducerea automată a proceselor" a apărută în editura Facla în anul 1985, şi în cadrul a 5 contracte de cercetare ştiinţifică. Pe viitor se conturează însă noi perspective de valorificare, avînd în vedere interesul crescînd pe care îl manifestă industria şi institutele de cercetare faţă de utilizarea sistemelor multimicroprocesor cu o viteză sporită de calcul pe de o parte iar pe de altă parte datorită apariţiei familiei MULTIPROM care pune la dispoziţia proiectanţilor o primă magistrală standardizată care facilitează construcţia modulară a arhitecturilor multimicroprocesor.

In acest context, din mulţimea direcţiilor posibile de dezvoltare ulterioară a aplicaţiilor MA propus se vor avea în vedere mai ales cele din domeniul automaticii : sinteza structurii SA, identificarea structurală şi parametrică, sinteza comenzii (sau a programului în cadrul structurii precizate).

Complexitatea aparatului matematic utilizat şi volumul calculelor care intervin în rezolvarea problemelor enumerate, viteza mare cu care se pretinde efectuarea lor justifică şi face imperios necesară utilizarea intensivă a sistemelor multimicroprocesor ca instrument practic de lucru, realitate reflectată şi în conţinutul lucrării de faţă.

Ca direcţii concrete de valorificare a rezultatelor prezentate se preconizează : continuarea cercetărilor legate de comanda procesului de levitaţie magnetică utilizînd SMM ; utilizarea calculului concurent în cadrul problemelor ridicate de captarea şi utilizarea eficientă a energiei eoliene ; utilizarea SMM în analiza şi comanda proceselor complexe ridicate de automatizarea prelucrărilor tehnologice prin metode neconvenţionale ; realizarea comenzilor adaptive în domeniul maşinilor unelte. Importante rezultate se pot obţine prin utilizarea MA propus şi în cadrul inteligenţei artificiale şi a roboţilor.

In final se pot aprecia că aspectele tratate în această lucrare sînt susceptibile de a fi perfecţionate şi dezvoltate în sensul adaptării la specificul domeniilor mai sus amintite şi eventual extinse şi asupra altor categorii de aplicaţii de interes pentru economia naţională.

BIBLIOGRAFIE

Notaţii utilizate în ordinea primei apariţii : Computer Design = CD; IEEE Transactionus on Industrial Electronic = IEEETIE; Editura tehnică=ET; Editura academiei = EA; Microprocessors Microsystems = =MM; Software Microsystems = SM; Microprocessor Microprogramming = MP; Mini-Micro Systems = MMS ; Editura ştiinţifică şi enciclopedică = ESE ; Editura didactică şi pedagogică = EDP; Electronic Design = ED; Computer Magazine = CM; Editura Militară = EM; IEEE Transaction on Computers = IEEETC; Comunications of ACM = CACM; IEEE Transactions on Audio Systems and Signal Processing = IEEE TASSP; Simpezion "Microprocesoare, microcalculatoare şi aplicaţii în economie" cat Automatică, SMA.

ALE Alexander P. Array Processor, dec.1981, CD.

ANG Ang Ws. Common element key to multiprocessors arhitectures, oct.1981, CD.

ASH Ashon S. ş.a. A up Based Multiloop Process Controller, feb.1983, IEEETIE

ACK Ackermann I, Abtastregelung, Springer Verlag, Berlin 1972.

BAB Bubuţia I, Dragomir UL., Mureşan I, Prostean C. Conducerea automată a proceselor, Facla 1985.

BAI Bailey C. Hardware, software trends, June,1984 MMS.

BAL1 B Baltac V. Optimizarea sistemelor de operare ale calculatoarelor numerice, Facla 1974.

BAL2 Baltac V. ş.a. Sisteme interactive şi limbaje conversaţionale, ET Buc.1984.

BAR Barthmaier P.I., Multiprocessing system mixes 8 and 16-bit uc, Feb. 1980, CD.

BEL2 Belea C., Vartolomei M, Metode algebrice şi algoritmi de sinteză optimală a sistemelor dinamice, ET 1985

BEL1 Belea C., Teoria sistemelor vol.II, EDP Buc.1985.

BER Berndt H. Software Support în Hardware, 13,1984, MP.

BEY Beyer O. ş.a. Stochastische Prozesse und Modelle Leipzig 1978.

BOI Bowen J. Software/Hardware integration on µP. febr.1985 MM.

BOL Bolomin A. ş.a. Multiprocessor Structures for Microprocessors dec.1984, SM.

BOW Bowen B.A., Buju IA, The Logicol Design of Multiple Microprocessor Systems, Prentice Hall 1980.

BRZ Brzozinski J. ş.a. A Family of Compatible Single and Multimicroprocessor Systems with 8 and 16 bit Microprocessors. 14,1984, MM.

BUD Budişan N., Teoria sistemelor - curs, IPTVT 1980.

BUH Bührer R. Hardware eines dynamisch konfigurierbaren Multiprozessors, Eidg. Techn. Hocschube Zuerich 1981.

CAL1 Călin S., Lamitrache I. ş.a. Reglarea numerică a proceselor tehnologice, ET Buc.1984.

CAL2 Călin S., ş.a. Sisteme automate numerice Buc.1984, ESE.

CAN Cantoni A. ş.a. A Technique for Interrupt Distribution in a Multiprocessor System, Soft and Micros, ovt.1982

CAR Cartianu Gh. ş.a. Semnale circuite şi sisteme EDP Buc.1980

CER Cerneţki VI. ş.a. Metode matematice şi algoritmi în studiul sistemelor automate ET, Buc.1973.

CIO Cioffi G., Valzadi P., A Fully Distributed Arbiter for Multiprocessor Systems, IP 11, 1983.

CHA Chaudhuri P., Scheduler or Realtime Process Control, apr. 1985, MM.

CHC — Chance R.I. ş.a., Using DMA Devices for Data Or... Performing a Fast Fourier Transform in Real Time ... SM.

CHD — Chandra R.C. Design of a uP based process controller using systems Theory, march 84, IEC.

CHU — Chung K.L. Markov Chains, Springer Verlag, 1967.

CIU — Ciucu Gh., Tudor Ct., Teoria probabilităţilor şi aplicaţii, 1983, ESE.

CON — Conrad M., Hopkins W.D., Functional architecture Three central CPU-s, sept.1981, ED.

CRE1 — Creangă I., Retucher C., Simovici D., Introducerea algebrică în informatică, 1973, ESE.

CRE2 — Creangă I., Simovici D., Teoria algebrică a semigrupurilor cu aplicaţii EP Buc.1977.

CRE — Creţu V., Sisteme de operare în timp real, Teză de doctorat, IPTVT 1984.

DAH — DAHmke M., Microcomputer Operating System Byte Books,

DAL — Leedip Kumar Das ş.a. Demms – A Design tool for Multi, Microprocessor Systems, iulie 1983, MP.

DAVA — Davidoviciu ..., ş.a. Minicalculatoare şi microcalculatoare în conducerea proceselor industriale, EP Buc.1983.

DIM — Dimitriadis B., ş.a. On a Multimicroprocessor for Real Time Encoding of Regularly Decomposed Images, ... M.

DIR — Director S.W., A computational Approach, J. Wiley and Sons 1975.

DOD1 — Dodescu Gh. ş.a., Sisteme electronice de calcul şi prelucrare PDP Buc.1980.

DOD2 — Dodescu Gh. Modelarea sistemelor de operare, Buc.1981,

DON — Donald W.J. Distributed System Testbeds oct.1982, CM.

DOU — Douglas G.F., A New Frontier for Systems Designers, Jan. 1983, IEEESC.

DRA1 — Dragomir P., Dragomir A., Structuri algebrice, Facla 1977

DRA2 — Dragomir P., Dragomir M., Structuri algebrice, Facla 1981

DRAI — Dragu I., Iosif I.ş., Prelucrarea numerică a semnalelor discrete în timp EM Buc. 1980.

DRAT — Dragomir TL., Iscaiti St. Elemente de teoria sistemelor şi reglaj automat – curs IPTVT 1979.

DUM — Dumitrescu-I. Călin S., ş.a. Automatizări şi echipamente electronice, EDP Buc.1981.

... — ...lon F.D. Sisteme multiprocesor şi prelucrarea paralelă ... 1976.

... — ... identificarea sistemelor EŢ Buc.1977.

... — ... Trends and Progress in ... Edition, Benjamin Frech, 1981.

... — ... E., ... H., Multiple Microprocessor Systems ... 1983.

FAI — ... Information Board – ... year, march ...

FIL — ... 8 ... and using ECL III ..., June 1983, ...

FLO — ... Flynn ... Microcomputer Type ... Multiprocessor ..., 13, ..., ...

... — ... Design ... Structured DMA, ..., ...

GEB — ... Systems Engineering ..., ...

GCM — ... A ... Systems, ...

GRA — Gray R.M. ...

GRO — Groza V., ... viteză, Teză de doctorat, IPTV ...

HAN — Hansel ... Regul ...

HAMY Hamey R.D. Rethinking Network a Coordinated Approach, July 1985, MSS.

HAR Harper R., uP Based Multichannel Analyser Developed Using Polyforth, June 1983,ME.

HAS Hassan M.M., ş.a. Highly Concurrent Computing Structures, ian.1982, IEEETC.

HAT Hatton IP. Automatic Speech Analysis and Recognition, D, Reidel P.C. 1982.

HAY Haynes S ., Highly Barallel Computing, ian.1982, CD.

HEA Heal B., Multiprocessor solution in OCCAM fo an NP-complete problem, ME, aug.1985.

HEM Heath W.X., A System Executive for Real - Time Microcomputer Programs, IEEE Micro, June 1984.

HEI Heider G. Lot Operating System Aid in Component Design, sep. 1982, CD.

HEM HEmenway I.E., Advanced 16-bit operating system handles multiple tesks an real time oct.1983.ME.

HER Herzog H.I., A Design Perspective for Real Time Task Control in Distributed Systems, fen.1983, IEEETID.

HIL Hill F.I., Peterson G.R. Calculatoare numerice, ET Buc. 198o.

HIR Hirschmen D.A., ş.a. Standard Modules Offer Flexible Multiprocessor Syste: Design may 1979, CD.

ION Ion D.I., Radu N., Algebra, EDP Buc.1975.

IONT Ionescu T., Sisteme şi echipamente pentru conducerea proceselor, EDP Buc.1982.

ION V1 Ionescu V., Sinteza structurală a sistemelor liniare, ES 1979.

ION V3 Ionescu V., Teoria sistemelor, EDP Buc.1985.

ION V2 Ionescu V., Popea C.,Proceduri de sinteză a sistemelor automate IPB 1979.

IOS1 Iosifescu Gh., ş.a. Teoria probabilităţilor şi statistica matematică ET Buc.1966.

IOS2 Iosifescu M. Lanţuri Markov finite şi aplicaţii ET Buc.1977.

IOS3 Iosifescu M., ş.a. Elemente de modelare stohastică ET Buc. 1984.

JAS Jaswa R., Designing Interrupt Structures for Multiprocessors sep. 1978,CD.

JUR1 Jurcă I. A Multiprocessor System with Multitasking Facilites; Teză de doctorat, Pijnacker 1977.

JUR2 Jurcă I. Simularea sistemelor continue şi discrete, curs 198o.

JUR3 Jurcă I., Sisteme de operare, curs, IPTVT 1984.

KAL Kalman R.E., Falb P.L., Arbib M.A., Teoria sistemelor dinamice ET Buc. 1975.

KAT Katz P., Digital Control Using Microprocessors Prentice Hall, 1981.

KEM1 Kemeny G.I., ş.a. Iontroduction fo Finite Mathematics, Moscova 1963.

KEM2 Kemeny G.I., Mathematical Models in the Social Sciences. Radio,Moso.1972.

KIT Kittler I. ş.a. Pattern Recongntion, D.Reidel P.C.1981.

KNG Kung SY, VLS for Manlvely Parallel Signal Processors, dec.1983, ME.

KOI Koinakov G.P., DunchevLS, A Method for Real Time Numerical Integration, 14,1984,EP.

KOR Korn A.G., Multiprocessor Design Bargass Sboerminie Alternatives for Continues System Simulation, may 1981 CD.

KUN Kung H.T. Why Systolic Architectures Ian 1982, IEEETC.

LEE1 Lee A.M. Les files d'attente, Dunod, Paris 197o

LEE2. Lee A.M. Teoria așteptării cu aplicații ET Buc. 1976

LIL Lillen H. Microprocesserus : Electronique Industrialle apr,198o.

LUP Lupu C., ș.a. Microprocesoare, aplicații EM Buc.1982

MAR1 Marsan M.A. ș.a.Comparative, performance of Single Bus Multiprocessor Arhitectures. IEEETC, dec.1982.

MAR2. Marsan M.A. ș.a. Modelling Bus Contention and Memory Interference in a Multiprocessor System, IEEETC, Ian. 1983.

MAR3 Marsan M.A., Carra G, Bus and Memory Interference in Double Bus Multiprocessor Systems, Jul 1984, MP.

MAT Mateescu A. Semnale, circuite și sisteme, EDP Buc. 1984

MCO MC Cluro RM, Optimizing a dual-processor for UNIX, May 1984, MMS.

MEZ Mezzalira L. ș.a. Acquining Real-Time Information from a Multitasking Microcomputer, 11,1983,MI.

MIH1 Mihoc Gh., Ciucu G. Introducere în teoria așteptării ET Buc.1967.

MIH3 Mihoc Gh., ș.a. Procese stohastice, ESE Buc.1978.

MIH2 Mihoc Gh., ș.a. Modele matematice ale așteptării EA Buc.1973.

MIH4 Mihoc Gh., Micu D. Teoria probabilităților și statistică matematică, EDP Buc.198o.

MIH5 Mihoc Gh., ș.a. Modele de analiză statistică, ESE Buc. 1982.

MIN Mintzer F. ș.a. The Real - Time Signal Processor, feb. 1983, IEEETASSP.

MUN Munro A. ș.a. Real - Time Control Including Concurrency, june 1982, SM.

MURXX1 I. Babuția, I. Mureșan, ș.a. Determination of a Reduced Order Model for a Distillation Colum. IPTVT Timișoara 25, 198o.

MURXX2 Mureșan I., ș.a. Conducere cu calculator numeric a vehiculelor pe șine, 4-Th International Conference on Control Systems and Computer Science Buc. June 1981.

MURXX3 Mureșan I., ș.a. Tehnici de identificare și modelare - îndrumător, IPTVT 1981.

MURX4 Mureșan I., Babuția I., Mureșan V. Simulation Sprache zum Entwarf und zur Optiemierung von Diskreten Automaten Kolloquium über Rechventechnik und Dateriverarbeitung, iunie 1981, Magdeburg DDR.

MURX5 Mureșan I., Hentea T. Programarea calculatoarelor de proces - îndrumător IPTVT 1982.

MUR6 Hentea T., Mureș[illegible]. Programarea calculatoarelor de proces, IPTVT 198[illegible].

MUR7 Mureșan I. Abordarea problemelor de realizare minimală a sistemelor decompozabile utilizînd limbajul categoriilor. Lucrarea de licență în matematică, Univ. Timișoara 1982.

MUR8 Mureșan I., Babuția I., Mureșan V. Limbaj de simulare pentru proiectarea sistemelor de conducere discretă în industria construcțoare de mașini, Bul. IPTVT Timișoara 29, 198[illegible].

MUR9 Mureșan I. Sisteme multimicroprocesor. probleme privind alegerea, proiectarea și implementarea unei arhitecturi. Referat în cadrul specializării prin doctorat, IPTVT 1984.

MUR1o Mureşan I. Analiza comportării sistemelor cu levitaţie magnetică utilizînd SMA cu prelucrare în timp real. Referat în cadrul specializării prin doctorat, IPTVT 1984.

MUR11 Babuţia I., Dragomir T., Mureşan I., Proştean O. Partea II.-Conducerea automată a proceselor Facla 1985.

MUR12 Mureşan I., Savii G. Interfaţă bazată pe seria M68ooo pentru echipament cu disc flexibil, SMA IPTVT 1985.

MUR13 Mureşan I. Utilizarea sistemelor multimicroprocesor în cond proceselor, SMA IPTVT 1985.

MUR14 Mureşan I. Simularea numerică a fdtz pe sisteme multiprocesor, SMA, IPTVT 1985.

MUR15 Mureşan I., Mureşan V. Elemente privind proiectarea unui SOTR pentru un sistem multimicroprocesor, SMA IPTVT 1985.

MUR16 Mureşan V., Mureşan I. SOTR pentru un sistem bazat pe µP Z8o, SMA, IPTVT 1985.

MUR17 Mureşan I., Robu N. Implementarea pe un sistem cu UC microprogramată a setului de instrucţii I8o8o, SMM, IPTVT 1985.

MUR18 Mureşan I., Robu N. Configuraţia de sistem multimicroprocesor utilizată pentru conducerea în timp real a proc, SMM, IPTVT 1985.

MUR19 Mureşan I. Sistem multimicroprocesor cu familia M68ooo, SMA, IPTVT 1985.

MUR2o Mureşan I. Sistem multiprocesor realizat cu familia I8o86, SMA IPTVT 1985.

MUR21 Mureşan I., Marchiş I. Sistem biprocesor realizat cu µP I8o8o şi Z8o SMA, IPTVT 1985.

MUR22 Proştean O., Mureşan I. Vol.II. Tehnici de identificare şi modelare - curs IPTVT 1985.

MUR23 Mureşan I., Robu N. Emulator de I8o cu o unitate centrală microprogramată. Propunere de brevet de invenţie, 1985.

MUR24 Mureşan I., Creţu V. Implementarea funcţiilor nucleului superior al SO-SMM pe o unitate microprogramată. Propunere de brevet de invenţie 1985.

MUR25 Mureşan I., Mureşan V., Creţu V. Implementarea funcţiilor nucleului interior al SO-SMM pe o unitate microprogramată. Propunere de brevet de invenţie 1985.

MUR26 Mureşan I. Unitate centrală cu MC68ooo compatibilă cu familia MULTIPROM. Propunere de brevet de invenţie 1986.

NEC Necula A.M. Simularea sistemelor continue şi discrete IPB 1979.

NEL Nelson I.C.C., Perfoi MK, Design of a hardware arbitor for Multimicroprocessor systems feb 1984, MM.

NIC Nicolau Ecn., Popovici Al., Introducere în cibernetica sistemelor hibride ET Buc.1975.

PAD Padulo L., Arbib A.M. Teoria sistemelor, London 1974.

PAT Patel J.H. Performance of processor-memory interconections for multiprocessors, IEEETC, oct.1981.

PAU Păunescu P. Analiza şi concepţia sistemelor de operare ESE,1982.

PEA Pearson C.E., Handbook of Aplied Mathematics Van Nostrand 1974.

PEN Penescu C., Sisteme ET Buc.1975.

PET Peterson I.L., Silberschatz A. Operating System Concepts, Addison-Wesley 1985.

POP1 Pop V. Arhitectura sistemelor multimicroprocesor, Lucrările colocviului de cibernetică Timişoara 1981.

POP2 Pop V. Structura sistemelor de prelucrare a datelor numerice - curs IPTVT 1981.

POP Popescu N. Sisteme informatice cu funcţionare în timp real, EM Buc.1984.

PRA Prangasvili I.V. Microprocesoare, Energia, Moscova 1979.

PRE Preece C. ş.a. Time-shared communications software for real - time controllers, SM, oct.1981.

PRO Prostean O., Mureşan I. Vol.I. Tehnici de identificare şi modelare, IPTVT 1985.

OPP Oppenheim A.V., Schafer R.W. Digital Signal Processing, Prentice-Hall,1975.

RAB Rabiner LR., Gold B. Theory and Application of Digital Signal Procesing, Prentice Hall 1975.

RAD Radu O., Săndulescu Gh. Filtre numerice aplicaţii ET Buc.1979.

RĂD Rădăceanu E. Limbaje de simulare EM Buc.1981.

RAJ Rajulu RG., Rojanavan, Execution Time Analysis of Process Control Algorithms on Microprocessors IEEETIE, nov.1982.

REA Ready I.F. Operating Systems conform to aplication needs, dec.1984. MMS.

RIT Ritchie D.M. The Evolution of The UNIX, Micros. oct. 1984.

ROD Rodda L.ş.a. A. Hierarchical Arhitecture With Independent processors for Real - Time Systems, 15, 1985, IP.

ROG Rogojan Al. Calculatoare numerice, curs IPTVT 1970.

ROZ Rozanov Y. Processus aleatoires ED. MIR 1975.

RUM M.van Rumste, The iAPX432, a Next Generation of Microprocessor, 11, 1983, MM.

SAB1 Sabatier A Le Multibus et ses signaux EAI 2, 1979.

SAB2 Sabatier A. L'utilisation du Multibus EAI, 3, 1979.

SCHE Schell R,R., A Security Kernel for a Multiprocessor CM Julil 1983.

SCH Schmidt Herman - Sisteme Multiprocesor, Electronik, 1982.

SCO Scordalakes E.N. Arbiter Handles Shared Resources for Multiprocessor, CD, dec.1981.

SEI Seitz CH.L, The Cosmic Cube, CACM ian 1985.

SHO Shoja G.C. ş.a. Some experiences of impl. ADA conc. facilites on a distr. multiprocessor, SM, oct.1982.

SNY Lawrence Snyder, Introduction to the Configurable Highly Parallel Computer IEEE TC ian.1982.

SOU Soucek B. ş.a. Event-Trajan Correlation and Real-Time Microcomputer Systems, MM, 11, 1983.

SPR Spriet J.A., Vansteenkiste G.C., Computer Aided Modeling and Simulation, Accademic Press, 1982.

STA Stănciulescu Fl. Dinamica sistemelor mari, EA 1982.

STD Stanomir D., Stănăşilă O. Metode matematice în Teoria sistemelor, ED Buc. 1980.

STE Stearns S.D., Digital Signal Analysis, Hayden Book Comp., 1975.

STO Stănăşilă O. Noţiuni şi tehnici de matematică discretă EDP Buc.1985.

TEO Teodorescu D. Sisteme automate deterministe ET Buc. 1984.

TER1 Tertişco M., Stoica P. Identificarea şi estimarea parametrilor sistemelor, EA 1980.

TER2 Tertişco M., Stoica P., Popescu Th. Modelarea şi predicţia seriilor de timp. EA 1985.

THO Thomas H.W. Design and performance of a simple nucleus for real - time control, oct.1982, SM.

TUC Tucker S., iRMX86, oct. 1983, LM.

TUCM Tucker M., Integrated Software Spars Mini Market,1985, MMS.

TZA TZafestas S.G. Microprocessors in Signal Processing, Measurement and Control, D. Reidel P.C 1982.

VAU Vaughan I.G., Design of an operating System nucleus for real - time microcomputer applications June 1983, SM.

VER Verona A. Introducere în coomologia algebrelor EA 1974.

WEI Weissberger I.A., Analysis of Multimicroprocessor System Arhitectures, CD, June 1977.

WHI Whitty R.W. ş.a. Structural programming. A. Tutorial Guide, SM, june 1984.

WIT Witten I.H., An introduction fo the arhitecture of the Intel iAPX432, SM, apr.1983.

YEH Yehosu D.B., Martinez R. Aplications of µP in Control Problems, CACC San Francisco 1977.

YEN YEn K., Digitel Simulation Algorithms Using Parallel Processing, IEEETIE, aug.1982.

ZAB Zabludowski A. ş.a. Analysis of a Multimicroprocessor System with Time - Shared Bus, 13, 1984, MP.

ZAD Zadeh L.A. ş.a. Teoria sistemelor, ET. Buc.197o.

ZEI Zeigler B.P., Theory of Modelling and Simulation, J.Wiley Sons

*** Calculatoare electronice din generaţia a cincea, EA 1985.

*** Dicţionar de informatică ESE, Buc.1981.

*** iAPXA32 , VLSI Microminframe, Intel 1981

*** Intel 3ooo Schottky Bipolar LSI Microcomputer Set.

*** Intel Component Data Catalog 1982

*** Manual de utilizare SIMPUB

*** Mickrocomputer-Baugruppensystem SMP Siemens

*** The New Collins Concise English Dictionary

*** Vlitoral industriei de programe EA 1985.

*** Protocol contract nr.189/1o.12.1983,1984, 1985 "Sisteme electrice de transport acţionate cu motoare liniare" CCSIT IP Craiova - IPTV

*** Catalog Multiprom CIEL - IPA