CUPRINS

Limbajul de simulare GASP

1. Introducere

Mult timp calculatoarele analogice s-au folosit ca simulatoare pentru rezolvarea problemelor exprimate ca ecuații diferențiale dar care nu puteau fi soluționate analitic. În timp ce metodele de simulare fizice operează eficient şi au un grad înalt de realism, datorită similarității lor cu sistemele pe care le imită, ele au două mari dezavantaje: cost ridicat şi rigiditate.

Calculatoarele analogice trebuie să fie conectate şi deconectate de la o problemă la alta şi folosesc echipamente electromecanice cu scopuri speciale, care sunt greu de adaptat la noi utilizări. Astfel, spre exemplu, pentru simularea zborurilor spațiale şi pentru instruirea piloților şi astronauților se fobosesc sisteme complexe de calculatoare analogice, precum şi unele echipamente electromecanice deosebit de sofisticate.

Calculatoarele digitale operează logic mai degrabă decât fizic, sunt programabile şi deci flexibile şi adaptabile la diverse tipuri de probleme. Programate să descrie sistemele logic şi numeric, ele pot fi folosite ca simulatoare, mult mai generale decât cele analogice, capabile să manipuleze aceste descrieri astfel încât un program să poată reproduce comportamentul unui sistem real.

Un calculator analogic consideră timpul într-o manieră directă, continuă aşa cum procesul de simulare analogică este realizat prin trecerea unui mediu real cum ar fi aerul, apa sau electricitatea. Ieşirile dintr-un simulator analogic sunt măsurate de cantitățile fizice luate după o perioadă de timp şi prin urmare sunt de asemenea continue (figura 1).

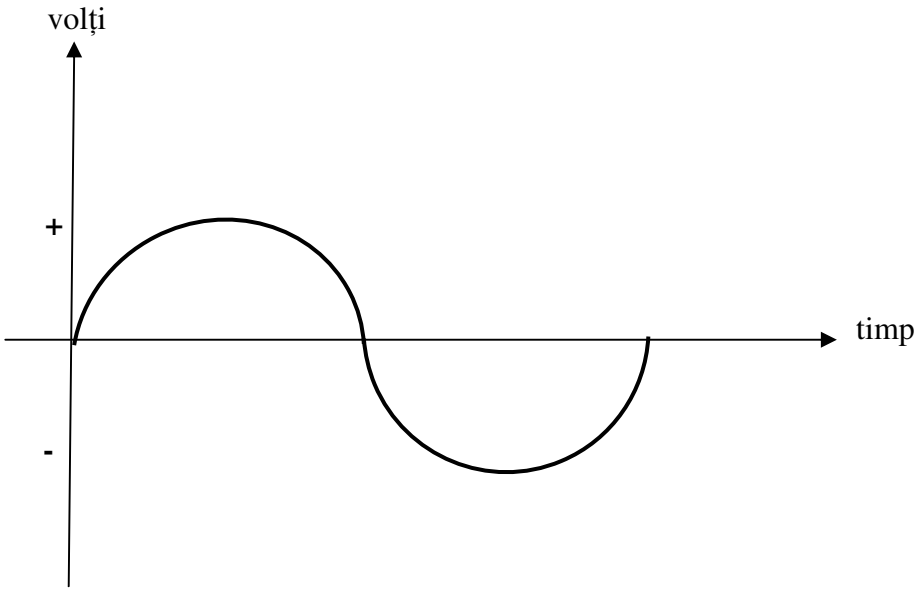


Fig.1 - *Răspunsul măsurat dintr-un simulator analogic*

Un calculator digital folosit ca simulator în timp continuu este un program pe calculator care imită operațiile unui calculator analogic, acționând în conformitate nu cu procese fizice ci cu modele care descriu procesele reale. Timpul este avansat în incremente mici, discrete, egale, pentru a aproxima scurgerea continuă a timpului în sistemul fizic simulat. Măsurătorile sunt făcute la intervale discrete, egale, dar aproximarea este pentru o funcție continuă (figura 2).

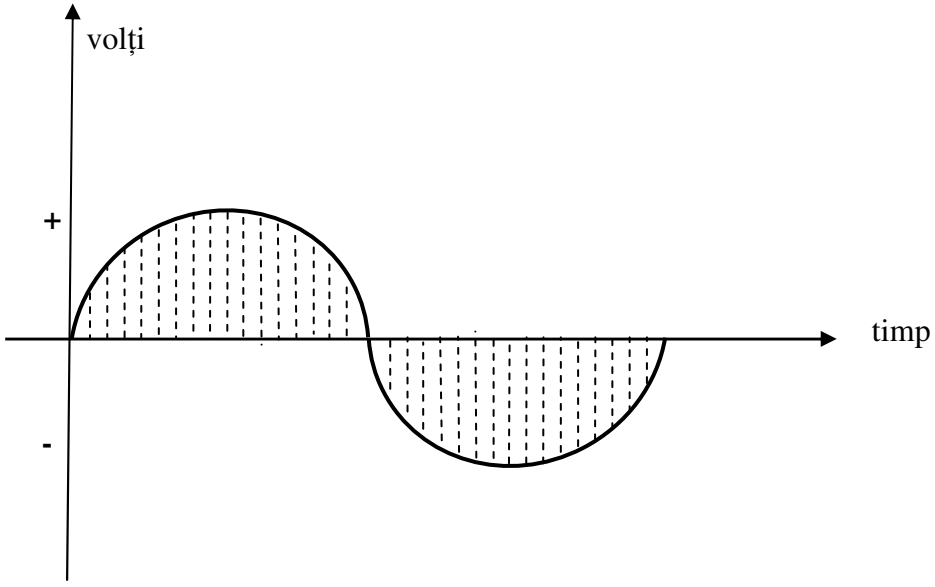


Fig.2 - *Răspunsul măsurat dintr-un simulator digital folosit ca simulator în timp continuu*

Un calculator digital folosit ca simulator în timp-discret lucrează în mod diferit. Timpul este spart în intervale discrete, nu neapărat egale, care reprezintă duratele interacţiunilor dintre diferite elemente ale unui sistem. Legăturile logice mai degrabă decât cele matematice, mută sistemul dintr-un punct de interacţiune în altul, prin subprograme denumite evenimente. Acest gen de simulare este cunoscut sub numele de **simulare evenimente - discrete** şi reproduce comportarea sistemului.

Măsurarea performanţei sistemului se face la **momente-eveniment** întâmplătoare şi reprezintă măsurători discrete (figura 3).

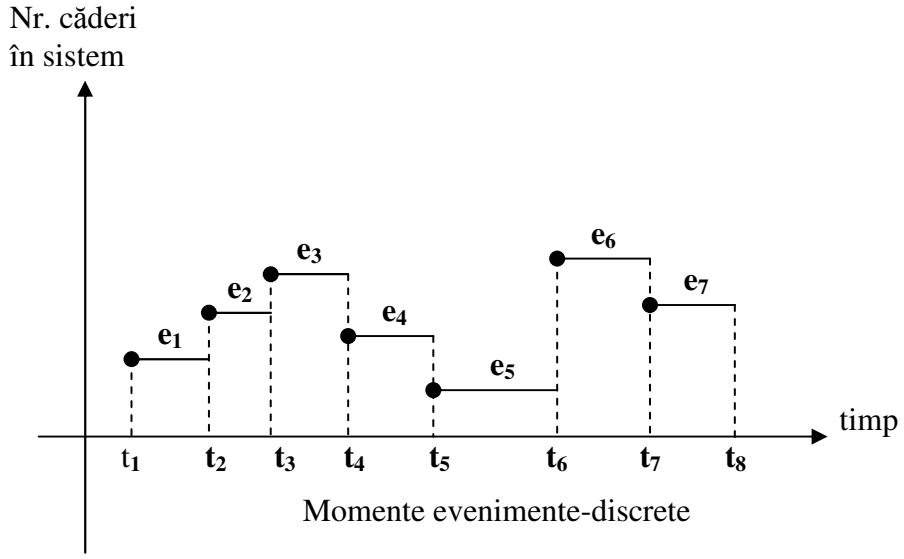


Fig.3.- *Răspunsul măsurat dintr-un simulator timp-discret*

Prin studiul comportării unui sistem, observând dinamica lui şi măsurând caracteristicile lui funcţionale putem profita de *"pătrunderi"* în privinţa performanţelor lui, a parametrilor critici, a facilităţilor înguste, putem face predicţii privind comportarea sistemului sub variate condiţii de lucru înainte sau după ce sistemul a fost costruit.

Pentru a studia un sistem în acest mod, sistemul trebuie să poată fi descris în termeni numerici, conceptul de bază fiind acela de descriere a stării sistemului.

Dacă un sistem poate fi caracterizat printr-un set de variabile, fiecare combinaţie a valorilor variabilelor reprezentând o stare unică sau condiţie a sistemului, atunci manipularea de variabile simulează mişcarea sistemului de la o stare la alta. Aceasta precizează că simularea este reprezentarea comportamentului sistemului prin mişcarea lui de la o stare la altă stare în conformitate cu reguli de funcţionare bine definite.

Numele de simulare evenimente-discrete vine de la maniera în care procesele ce afectează dinamica sistemului sunt observate la momente discrete în perioada simulată. Procesele sunt descrise cu ajutorul evenimentelor care sunt blocuri ce construiesc dinamica sistemului.

Simularea este realizată cu programe care interacționează la momente discrete în timp, comportarea sistemului rezultând din schimbarea stărilor, schimbare care se produce datorită realizării evenimentelor sistemului.

Un simulator este un laborator artificial şi odată ce un sistem a fost modelat şi programat, pot fi făcute oricâte experimente şi trase concluzii despre sistem:

- fără construirea lui, în cazul unui sistem propus;
- fără perturbarea lui, dacă sistemul funcționează dar este costisitor sau nesigur pentru experiment;
- fără distrugerea lui, dacă scopul unui experiment este de a determina limitele sale de rezistență.

Simulatorii pot fi utilizați în proiectare, în determinarea performanței şi în analiza comportamentului unor sisteme complexe.

Una dintre cele mai importante utilizări ale simulării este investigarea efectului schimbărilor într-un sistem asupra performanțelor sale, indiferent dacă simularea s-a folosit pentru evaluarea sau proiectarea sistemului.

Simularea se recomandă să fie aplicată la rezolvarea problemelor complexe, de dimensiuni mari sau când analiza matematică nu poate fi făcută şi implică următorii paşi:

Pasul 1: Se determină dacă problema respectivă se poate rezolva prin simulare; factorii cruciali sunt: costul simulării, realizabilitatea dirijării experimentelor reale, şi posibilitatea analizei matematice;

Pasul 2: Construirea unui model pentru rezolvarea problemei;

Pasul 3: Scrierea unui program pe calculator, prin codificarea modelului într-un program de simulare operațional;

Pasul 4. Studiul sistemului prin realizarea unor experimente cu ajutorul programului de simulare.

Pentru proiecte mari echipa care realizează simularea este alcătuită din analistul de sistem, constructorul modelului, programatorul calculatorului şi un statistician, iar pentru proiecte mici, analistul de sistem poate să realizeze singur toți aceşti paşi.

GASP facilitează construirea modelului şi prezintă un limbaj pentru descrierea sistemelor dinamice, precum şi concepte pentru exprimarea aspectelor relevante ale comportării sistemelor. GASP furnizează de asemenea programatorului un set de instrucțiuni FORTRAN proiectate pentru a realiza cele mai importante lucrări în programarea simulării.

Faptul că conceptele de modelare şi instrucțiunile FORTRAN sunt congruente, reprezintă legătura dintre activitățile de modelare şi de programare care sunt de asemenea importante pentru succesul studiului de simulare.

Toate limbajele de simulare, mai puțin GASP, furnizează concepte de modelare şi instrucțiuni de programare de un înalt nivel.

Multe limbaje sunt mai puternice şi mai eficiente însă GASP este atractiv pentru scopuri didactice şi utilizatori industriali cu calculatoare mici sau medii, deoarece:

- GASP este bazat pe FORTRAN, este modular şi poate fi utilizat pe orice calculator care are compilator FORTRAN;
- GASP este uşor de învățat şi poate fi rapid implementat pe orice sistem de calcul deoarece se bazează pe concepte de simulare uşor asimilabile de utilizatori;
- GASP poate fi convertit în alte limbaje de simulare, cu minim de efort şi pregătire, deoarece are concepte similare cu acestea;
- GASP este uşor de modificat şi de extins pentru a satisface cerințele unei aplicații particulare.

2. Concepte şi variabile GASP importante

Un **sistem / subsistem** reprezintă o colecție relativă de articole dintr-un sector limitat al realității, scopul fiecăruia fiind în întregime determinat de rațiunea pentru care a fost identificat şi izolat ca obiect de studiu pentru simulare.

Pentru a considera **scopul** unui sistem, trebuie să se analizeze sistemul, elementele şi conexiunile relevante, structura sa în cadrul **granițelor** sale. În timp ce granița unui sistem poate fi fizică, este mai bine să se determine o graniță în termeni de cauză şi efect. Dacă un anumit aspect al unui sistem este complet determinat de influențe din afara sistemului, atunci acel aspect este în afara granițelor sistemului. În terminologia sistemică elementele care sunt în afara granițelor sistemului, dar care îl pot influența constituie **mediul sistemului.**

Spre exemplu, dacă într-un model de organizare a producției, vânzările de produse sunt considerate intrări în model, modelul nu poate influența vânzările.

Pe de altă parte, modelul sistemului de producție nu conține un model cauză şi efect al vânzărilor, ci numai un model statistic, istoric sau predictiv al vănzărilor şi prin urmare organizația care se ocupă de vânzare face parte din mediu, fiind în afara granițelor definite ale sistemului.

În cadrul granițelor unui sistem sunt entități ca: oameni, echipamente, organizare, procese, materii prime care interacționează unele cu altele. Entitățile sunt de tipuri diferite, au caracteristici diferite şi sunt implicate în diferite tipuri de **activități.**

Scopul fiecărui model de simulare este determinat de problemele specifice pe care este proiectat să le rezolve şi constă în a reproduce activitățile unui sistem şi a cunoaşte comportamentul şi performanța potențială a acestuia.

Acest lucru se realizează definind **stări** ale sistemului şi construind activități care mişcă sistemul dintr-o stare în altă stare.

Un sistem se află într-o anumită stare particulară distinctă dacă toate entitățile lui se află în stări corespunzătoare cu acea stare.

O entitate este într-o stare particulară când **atributele** ei au valorile numerice specifice acelei stări.

De exemplu, un utilaj poate fi modelat astfel:

- Entitatea: strung;
- Atribute: viteza de prelucrare: 1000 rotații/minut;
  - cost per operație : 5$/oră;
  - starea procesării:

    0 - liber/neocupat;

    1 - în lucru;

    2 - în reparație;

    3 - defect/aşteptând reparația.

Un strung este în starea "neocupat" dacă atributul stării de procesare este zero, iar un atelier este în starea "neocupat", dacă toate strungurile sunt în această stare.

Rezultă că două din cele mai importante sarcini în modelarea simulării sunt identificarea entităților şi atributelor şi codificarea valorică a atributelor pentru caracterizarea stărilor sistemului.

Un sistem trece de la o stare la altă stare după cum entitățile lui determină activitățile să schimbe stările. Astfel, un strung prelucrează o piesă şi se schimbă de la starea "neocupat" la starea "ocupat"; un lift urcă un etaj şi se schimbă de la starea "liftul la etajul n", la starea "liftul la etajul n + 1" etc. Activitățile au fie un efect instantaneu asupra sistemului, fie după o perioadă de timp la sfârşitul căreia se creează o nouă stare a sistemului.

Conceptul de **eveniment** este esențial în simularea evenimente-discrete, evenimentul având loc la un anumit moment în timp, numit **moment-eveniment**, când fie începe, fie se termină o activitate care schimbă starea sistemului.

Comportarea sistemuiui este simulată prin schimbări de stare care au loc aşa cum se produc evenimentele după o perioadă de timp.

Realizarea unui eveniment poate schimba starea sistemului în mai multe moduri:

- alterând valoarea unuia sau mai multor atribute ale entităților;
- alterând relațiile existente între entități, şi/sau;
- schimbând numărul entităților din sistem.

Evenimentele, ca şi entitățile, au atribute: un atribut care defineşte momentul la care se produce evenimentul, un atribut care defineşte tipul evenimentului şi altele care descriu atributele entităților afectate de eveniment (de exemplu, evenimentul "Liftul începe să se mişte", are şi atributele: "Etajul la care se află liftul de obicei", şi "Direcția de mişcare a liftului").

În timp ce **tipurile de entități** prezente într-un sistem sunt constante (stații de servire şi clienți într-un sistem de aşteptare), **numărul de entități** implicate în activitățile sistemului în fiecare moment poate fi variabil.

De asemenea, **numărul de atribute** definite pentru fiecare tip de entitate şi **relațiile lor de legătură** sunt într-o oarecare măsură arbitrare şi depind de obiectivele modelului de simulare.

De exemplu, funcționarea unui sistem cu o singură stație de servire poate fi descrisă în termeni de evenimente, astfel: un client (entitate) soseşte (eveniment) la un anumit moment specific în sistem (un atribut al clientului).

Dacă stația de servire (entitate) este liberă (un atribut al stației) se determină durata servirii pe baza distribuției duratei de servire a stației (un atribut al stației de servire). Momentul de terminare a servirii unui client (atribut al clientului şi al stației de servire) se determină ca suma dintre momentul intrării lui în servire şi durata servirii, şi un eveniment de terminare a servirii este programat să se producă la acest moment.

Dacă staţia de servire este ocupată când clientul soseşte, acesta intră în firul de aşteptare (un fişier care ordonează clienţii care aşteaptă pentru servire) într-o poziţie bazată pe sistemul de priorităţi asociat pentru clienţi. Prioritatea şi numărul de clienţi în firul de aşteptare sunt atribute ale firului de aşteptare. Valorile particulare specifice pentru a ordona clienţii, sunt atribute ale clienţilor. Structura sistemului entitate – atribut - relaţii de legătură se stabileşte în GASP prin declaraţii de date. Entităţile sunt memorate ca **vectori şi matrici,** elementele acestora reprezentând atribute.

De exemplu, instrucţiunea DIMENSION SERV(5,3) poate reprezenta 5 staţii de servire dintr-un sistem, fiecare având următoarele trei atribute:

- situaţia servirii;
- ritmul de servire;
- numărul de clienţi serviţi.

În acest caz, SERV(3,1) reprezintă situaţia servirii pentru staţia 3, SERV(1,2) reprezintă ritmul de servire pentru staţia 1, iar SERV(5,3) reprezintă numărul de clienţi serviţi de staţia 5.

Relaţiile de legătură sunt schimbătoare, au un caracter dinamic, iar datele care le exprimă sunt stocate în fişiere într-o arie NSET şi sunt prelucrate prin rutine speciale, ca FILEM (pentru a stoca o cantitate în fişier) şi RMOVE (pentru a lua o cantitate din fişier).

Entităţile, atributele şi legăturile alcătuiesc **structura statică** a unui model de simulare şi au rolul de a descrie starea sistemului. Descrierea modului de funcţionare a sistemului se face cu ajutorul evenimentelor.

Cheia succesului în simularea evenimente - discrete constă în capacitatea de a organiza evenimentele sistemului astfel încât succesiunea în care ele sunt executate în cadrul programului de simulare să corespundă succesiunii în care ele se produc în sistemul real, păstrându-se astfel legătura în timp dintre evenimentele simulate şi cele reale. În acest sens, în studiile de simulare se consumă mult timp şi efort în proiectarea de programe care organizează şi planifică evenimentele într-un model astfel încât sistemul modelat să evolueze în timp într-o manieră realistă. Un asemenea program este deseori numit **ceasul simulării** şi utilizarea lui eliberează programatorul de sarcina de a ordona evenimetele în succesiunea lor cronologică.

În GASP, variabila TNOW (ceasul simulării) este în mod automat actualizată astfel încât să semnifice trecerea timpului de simulare şi este iniţializată cu variabila TBEG la începutul simulării.

Pentru a facilita operațiile pe calculator implicate în recunoaşterea tipului de eveniment, fiecare eveniment simulat are asignat un cod numeric numit **cod - eveniment** definit de programator. În timpul simulării variabila JEVNT păstrează codul evenimentului care se produce la momentul TNOW. Zona de memorie NSET, este folosită pentru stocarea evenimentelor, entităților, precum şi a atributelor asociate acestora sub formă de articole în fişiere. Pentru a prelua avantajele procedurii FORTRAN de stocare a matricilor în serie pe coloane, NSET nu este stocată în COMMON, iar locația de început a lui NSET este transmisă ca argument la fiecare subprogram.

Când se compilează un subprogram şi se face o referire la o celulă, NSET (I,J), poziția celulei se calculează automat ca MXX * (J - I) + I locații, de la locația de start, unde MXX reprezintă numărul de linii din NSET.

Toate subprogramele GASP sunt compilate cu numărul de coloane ID = 1. În felul acesta, toată memoria disponibilă poate fi alocată pentru NSET, prin atribuirea unei valori corespunzătoare pentru ID (creşterea numărului de coloane) în programul principal fără a fi necesară compilarea subprogramelor. Dacă creşterea numărului de coloane face imposibilă tipărirea tuturor coloanelor unui rând, se recomandă interschimbarea liniilor şi coloanelor.

Fiecare înregistrare din aria NSET conține valorile atributelor şi indicatorii poziției relative. Atributele sunt stocate în primele IM linii din NSET, iar indicatorii în ultimele două linii: MX = IM + 1, MXX = IM + 2.

Înregistrările unui fişier au o ordine fizică însă ele sunt considerate de GASP într-o ordine logică. Înlănțuirea logică este asigurată pentru fiecare înregistrare cu ajutorul a doi indicatori de poziție relativă aflați în liniile MX şi MXX, care indică înregistrarea succesoare şi respectiv, înregistrarea predecesoare. Pentru ultima înregistrare a unui fişier JO, MLE(JQ), indicatorul succesor are valoarea, NSET(MX,MLE(JQ)) = 7777, iar pentru prima înregistrare MFE(JQ), indicatorul predecesor are valoarea NSET(MXX, MFE(JQ)) = 9999.

Deoarece aria NSET este partiționată în mai multe fişiere păstrarea legăturilor dintre înregistrările care aparțin aceluiaşi fişier este realizată prin indicatorii predecesori şi succesori. Plasarea înregistrărilor în fiecare fişier se face după valorile atributului definit de variabila KRANK(JQ), cu o procedură proprie de ordonare LVF (Lower-Values-First) dacă variabila de ordonare INN(JQ) = 1, sau HVF (Higher-Values-First) dacă lNN(JQ) = 2.

Programatorul nu va utiliza în mod direct aria NSET, citirea sau înscrierea unei înregistrări făcându-se prin intermediul unui vector ATRIB cu IM componente ATRlB(1),..., ATRIB(IM) în care sunt plasate atributele, sistemul de indicatori fiind actualizat în mod automat de GASP.

3. O metodă de programare a simulării

Fiecare **model de simulare** conține:

- un set de programe-eveniment care descriu reguli de funcționare a sistemului;
- vectori şi matrici în care se stochează informațiile;
- o rutină executiv care dirijează fluxul de informații şi de control în cadrul modelului.

Un **program de simulare** este constituit din subprograme şi un program executiv care organizează şi controlează execuția lor, performanțele acestuia reflectând pe cele ale sistemului simulat. Orice program de simulare poate fi privit ca asocierea a două structuri logice în strânsă legătură şi anume:

- **logica tehnologică,** ce reprezintă tehnologia sau ştiința de a studia un sistem;
- **logica structurală,** care controlează funcționarea unui model în timpul simulării şi execută sarcinile asociate programului de simulare.

Fiecare model de simulare este unic datorită logicii lui tehnologice, adică a descrierii sistemului şi a regulilor lui de funcționare. Deşi este posibil să se stabilească într-o oarecare măsură anumite reguli sau principii generale ale construirii modelului, succesul aplicării acestor principii este în mare măsură dependent de experiența constructorului de modele.

Există o logică structurală comună pentru toate programele de simulare. GASP oferă programatorului o logică structurală particulară cu care poate controla simularea. Programele de simulare cer răspuns la trei întrebări de bază pentru control: CE, CÂND, şi CUM.

CE şi CÂND se referă la organizarea în timp a unui program: CE eveniment urmează? CÂND va fi programat evenimentul următor? CUM se referă la metodele de realizare a evenimentelor şi la logica schemelor de control CE şi CÂND.

CE

Un program de simulare este într-o anumită fază de funcționare când execută funcția corespunzătoare unui modul format din unul sau mai multe programe. GASP este capabil să realizeze următoarele funcții specifice cerute de fiecare simulare (figura 4):

- inițializarea stării sistemului;
- controlul evenimentelor;
- monitorizarea programului şi manipularea erorilor;
- calcule statistice şi generarea de rapoarte;

- memorarea şi regăsirea informaţiilor;
- colectarea datelor;
- generarea variabilelor aleatoare.

Primele patru funcţii reprezintă modulele de bază ale controlului GASP, iar celelalte sunt orientate pe algoritm şi deci asigură capacitatea de prelucrare a datelor. În orice moment, un program GASP poate fi în unul din modulele de bază şi poate să apeleze orice modul orientat pe algoritm. Programele care fac posibilă realizarea sarcinilor de control şi de prelucrare a datelor, precum şi rutinele-eveniment scrise de programatorul simulării reprezintă CE-ul fiecărui program GASP. Ele fac ca programele să poată fi apelate pentru execuţie în orice moment în timpul simulării.

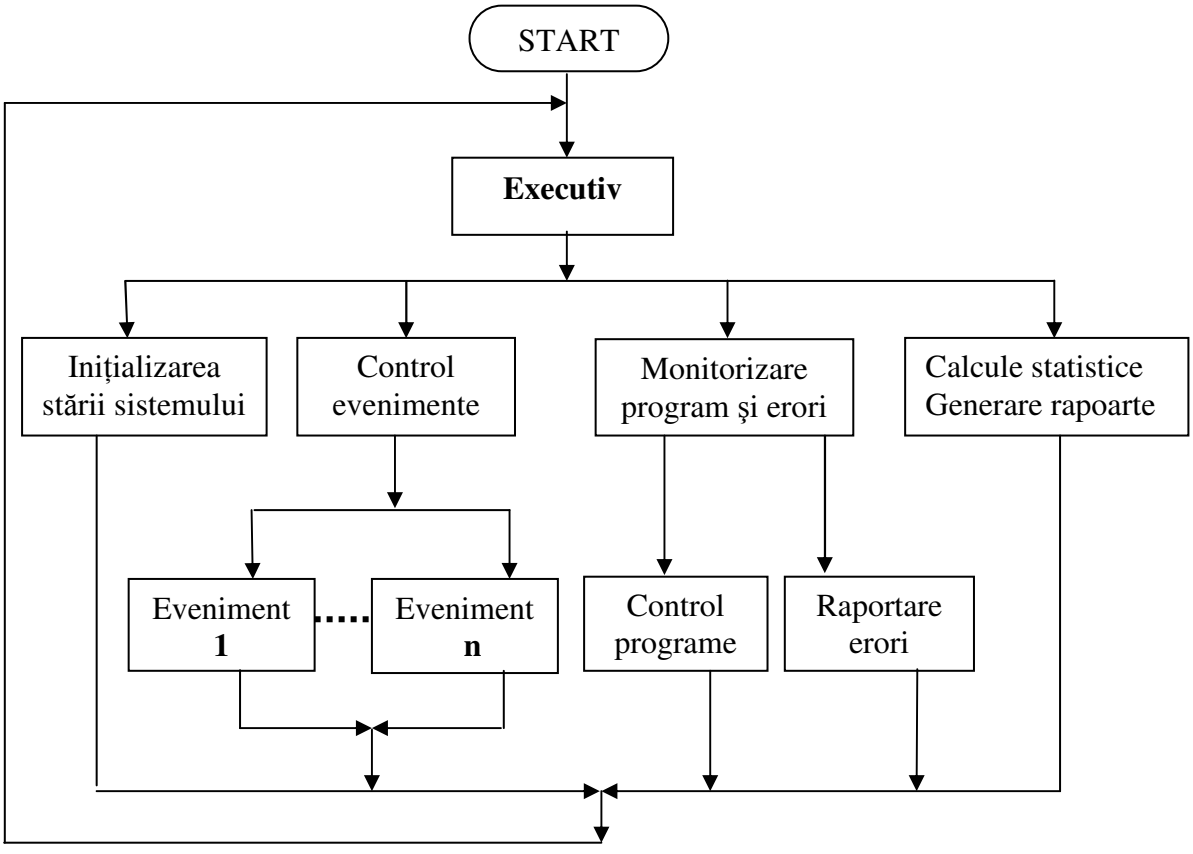


Fig.4. – *Modulele de bază ale controlului GASP*

CÂND

Un rol important în orice simulare îl are programul care decide CÂND trebuie executat fiecare subprogram - eveniment. Acest program acționează ca "executiv" prin orientarea fluxului de informații şi de control într-o manieră ordonată în conformitate cu regulile modelului. În GASP controlul programului de simulare se realizează prin două funcții:

- **funcția executivă**, care este din punct de vedere logic orientativă şi dirijează simularea comutând de la modul la modul aşa cum o cere logica programului;
- **funcția de selectare a evenimentelor**, care este orientată în timp şi conduce în cadrul fiecărui modul de simulare selectând rutinele-eveniment ce trebuie executate în conformitate cu momentul la care evenimentele sunt programate să se producă. Programarea evenimentului este un concept de bază al simulării în general şi al GASP în particular.

În plus, flexibilitatea simulării se poate obține prin definirea **evenimentelor de control** în afară de evenimentele-activități ale sistemului. Evenimentele de control sunt instrucțiuni de orientare în timp pentru executiv care-i permit să comute de la modul la modul în funcție de timp sau de o bază logică.

Spre exemplu, se poate planifica evenimentul de **"a intra în modulul de output**" când ceasul simulatorului atinge 1000 ore sau când cererea depăşeşte 100 unități. În limbajele de simulare există multe alte forme sugerate pentru executiv şi programele de control a evenimentelor.

CE şi **CÂND -** un sistem de control al programului.

Forma de organizare utilizată în GASP este ierarhizată, adică există câteva nivele de control, fiecare având autoritate peste nivelele subordonate lui. Cel mai înalt nivel de control îl are nivelul executiv, care determină ce trebuie să facă programul în fiecare moment din perioada simulată şi direcționează controlul spre modulul care trebuie să realizeze sarcina selectată. Controlul trece temporar de la nivelul executiv la modulul care se execută şi apoi revine la nivelul executiv. Conceptul de organizare a controlului în GASP se realizează prin distribuirea informației între nivelul executiv şi modulele programului printr-un fond de date comun.

Conceptele de **control ierarhizat, modul, fond de date comun** asigură flexibilitate în proiectarea şi adaptarea programelor, deoarece schimbările în modelele de simulare impun schimbări în datele de intrare, în elementele sistemului sau în logica evenimentelor.

O structură modulară permite ca logica evenimentelor să fie schimbată uşor deoarece fiecare eveniment este un subprogram separat. Fondul comun de date permite modificarea automată a datelor în tot sistemul printr-o singură schimbare în datele de intrare. Pregătirea rapoartelor care sintetizează rezultatele rulărilor de simulare se simplifică prin utilizarea de programe de raportare-standard care îşi preiau informațiile necesare din fondul comun de date.

Un alt avantaj al acestei abordări îl constituie faptul că modulul de depanare este simplificat, acesta furnizând punctele de acces din care pot fi extrase rezultatele programelor, fără a perturba logica producerii evenimentelor sistemului dat.

CUM - Rutina executiv

Pe baza informațiilor pe care le primeşte de la fiecare eveniment procesat, rutina executiv poate să realizeze sarcina de monitorizare a programului, să genereze un raport statistic al avansării unui program să selecteze evenimentul următor, sau să termine o rulare de simulare. Funcția de control-executiv este îndeplinită de subrutina GASP, care conține:

a) un fond de date de legătură (prin instrucțiuni COMMON);

b) un set de evenimente planificate ale sistemului, ordonate în timp;

c) instrucțiuni logice care se ocupă cu controlul programului;

d) apelări de subrutine pentru:

- inițializarea datelor (DATAN);
- monitorizarea şi verificarea erorilor (MONTR);
- selectarea evenimentelor (EVNTS);
- colectarea datelor de ieşire (COLCT, TMST, HISTO);
- raportarea rezultatelor (SUMRY).

Pentru a înțelege simularea, trebuie înțeles modul cum lucrează subrutina GASP, care are ca primă sarcină introducerea datelor referitoare la:

- parametrii care descriu variabilele independente ale unui experiment de simulare;
- atributele entităților pentru inițializarea sistemului;
- evenimentele care încep simularea.

Inițializarea datelor dă o imagine a sistemului la începutul simulării şi furnizează instrucțiuni pentru punerea sistemului în mişcare prin programarea evenimentelor inițiale pentru activități. Momentul de realizare al unui eveniment este un atribut inclus în structura înregistrării fişierului de evenimente.

GASP furnizează un sistem de memorare şi de regăsire a informației prin intermediul unor subrutine specializate, atât pentru fişierul ce conține evenimentele în ordine cronologică, cât şi pentru alte fişiere ce conțin informații uzuale ce trebuie păstrate sau prelucrate în timpul simulării.

Pentru a scrie un program de simulare este necesar să se specifice schimbările de stare ale sistemului care se produc la momente-eveniment, precum şi evenimentele viitoare care sunt generate de realizarea evenimentelor.

GASP a fost proiectat astfel încât să reducă în măsura în care este posibil efortul de programare, singurele programe ce trebuie scrise fiind: un program principal, subrutinele eveniment şi subrutina de selectare a evenimentelor, numită EVNTS.

Programul principal inițializează variabilele non-GASP folosite în subrutinele eveniment şi apelează subrutina executiv GASP care preia controlul până la sfârşitul simulării. Subrutina de selectare a evenimentelor EVNTS, este ceva mai mult decât o instrucțiune GO TO calculat. Componentele unui program tipic de simulare scris în GASP sunt ilustrate în figura 5.

Un **prototip** pentru programul principal şi subrutina EVNTS de selectare a evenimentelor, al unui sistem de servire cu o singură stație este exemplificat în continuare.

În subrutina EVNTS, "I" reprezintă codul evenimentului, care poate fi:

- I = 1, "sosire client" (subrutina ARRIV);
- I = 2, "servire client" (subrutina ESERV);
- I = 3, "terminarea simulării" (subrutina ENDSM).

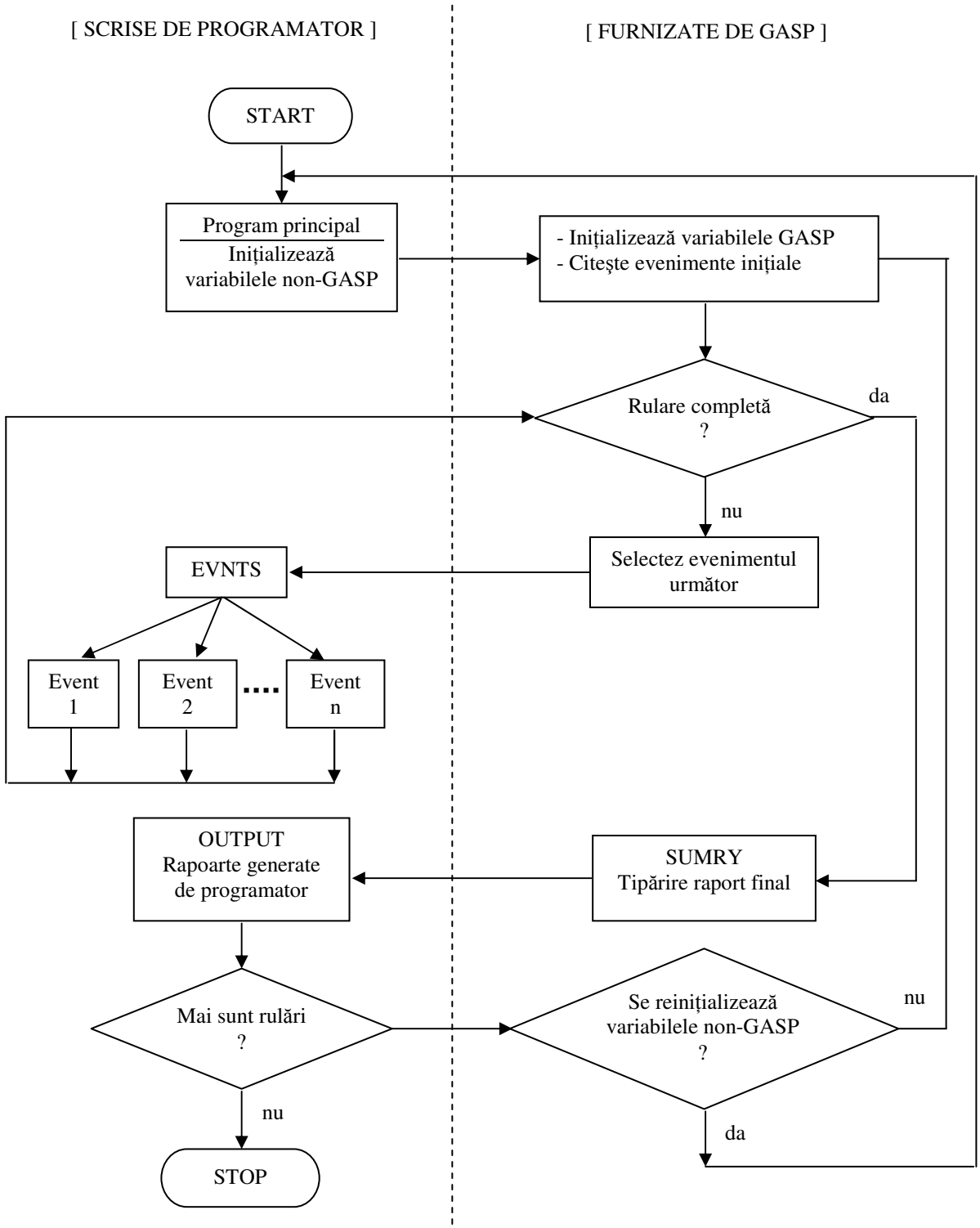


Fig.5. - *Componentele unui program de simulare cu GASP*

Programatorul trebuie să scrie programul principal şi să fixeze ordinea subrutinelor eveniment pentru a avea un program de simulare complet, aşa cum este ilustrat în exemplul următor:

```
C       MAIN PROGRAM (program principal)
        DIMENSION NSET (6,ID)
C       ID: k (număr de coloane ce trebuie specificat)
        COMMON (listă variabile GASP)
        COMMON (listă variabile non-GASP)
C       Inițializarea variabilelor non-GASP
        CALL GASP(NSET)
C       Dacă sunt mai multe rulări, se inserează instrucțiunea GO TO
C       la reinițializarea variabilelor non-GASP, sau
C       la, din nou, CALL GASP
        CALL EXIT
        END

        SUBROUTINE EVNTS (INSET)
        DIMENSION NSET (6,ID)
C       ID: nr.de coloane ce trebuie specificat
        COMMON (listă variabile GASP)
        COMMON (listă variabile non-GASP)
        GO TO (1,2,3),I
1       CALL ARRIV (NSET)
        RETURN
2       CALL ESERV(NSET)
        RETURN
3       CALL ENDSM (NSET)
        RETURN
        END
```

4. Principalele funcții realizate de limbajul de simulare GASP

Cele mai importante funcții realizate de limbajul de simulare **GASP** şi subprogramele corespunzătoare sunt prezentate în tabelul 1.

Tabel 1

Nr. crt.	Funcția	Subprograme
F1	Executiv + controlul simulării	GASP (NSET)
F2	Inițializarea datelor	DATAN (NSET)
F3	Stocarea şi regăsirea informației	SET(JQ,NSET) FILEM(JQ,NSET) RMOVE(KCOL,JQ,NSET) FIND(XVAL,MCODE,JQ,JATT,KCOL,NSET)
F4	Colectarea datelor	COLCT(X,N,NSET) TMST(X,T,N,NSET) HISTO(X_1,A,W,N)
F5	Calcule statistice şi generări de rapoarte	PRNTQ(JQ,NSET) SUMRY(NSET)
F6	Monitorizarea şi raportarea erorilor	MONTR(NSET) ERROR(J,NSET)
F7	Generarea variabilelor aleatoare	DRAND(ISEED,RNUM) NPOSN(J,NPSSN) FUNCTION UNFRM(A,B) FUNCTION RNORM(J) FUNCTION RLONG(J) FUNCTION ERLNG(J)
F8	Alte rutine ajutătoare (tip funcții)	SUMQ(JATT,JQ,NSET) PRODQ(JATT,JQ,NSET) AMIN(ARG_1,ARG_2) AMAX(ARG_1,ARG_2) IMAX($IARG_1$,$IARG_2$)

F1. Execuţia şi controlul simulării

Această funcţie este realizată de subrutina **GASP(NSET)** referită şi ca "executivul GASP", care constituie nucleul limbajului de simulare GASP. Subrutina executiv este chemată numai de programul principal şi preia controlul până la terminarea simulării. Schema logică generală a acestei subrutine este prezentată în figura 6.

GASP cheamă mai intâi subrutina DATAN care iniţializează toate variabilele GASP din COMMON şi pregăteşte fişierele.

Este chemată apoi subrutina MONTR care tipăreşte conţinutul fişierelor iniţiale pentru verificări. GASP începe simularea folosind subrutina RMOVE pentru a lua următorul (primul) eveniment din fişierul de evenimente (fişierul 1). Este evident că cel puţin un eveniment iniţial trebuie înserat în fişierul de evenimente de către subrutina DATAN.

Cu instrucţiunea CALL RMOVE (MFE(1),2,NSET) se scoate din fişierul 1 următorul eveniment şi se memorează atributele lui în vectorul ATRIB. Momentul şi codul evenimentului sunt primele două atribute ale oricărui eveniment şi prin urmare, GASP le transferă variabilelor corespunzătoare, actualizându-le astfel:

TNOW = ATRIB(1);

JEVNT = ATRIB(2).

Apoi se face un test pentru a determina dacă evenimentul selectat este un eveniment de monitorizare sau un eveniment scris de programator. Există două coduri de evenimente pentru monitorizare:

- codul 100, care impune schimbarea codului variabilei JMNIT din 0 în 1 şi invers; dacă JMNIT = 1 fiecare eveniment este tipărit aşa cum se produce;
- codul 101, care impune chemarea subrutinei MONTR pentru tipărirea rezultatelor intermediare (necesare pentru depanare). Subrutina MONTR poate fi modificată pentru obţinerea unor informaţii suplimentare în timpul simulării.

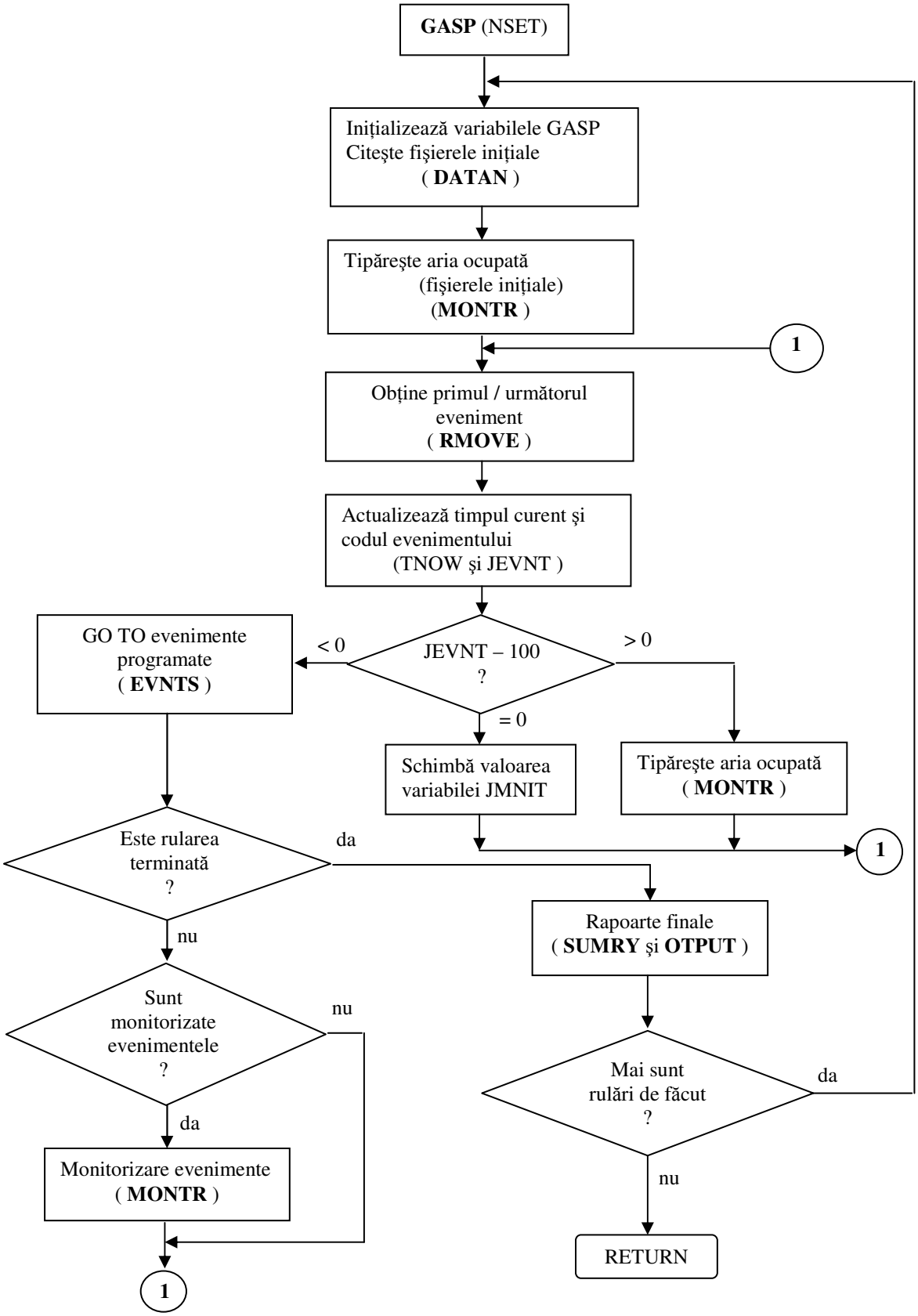


Fig.6.- *Schema de principiu a subrutinei GASP*

Când codul evenimentului este mai mic decât 100, însemnă că are loc un eveniment scris de programator; subrutina EVNTS cheamă rutina-eveniment corespunzătoare codului şi după executarea acesteia, controlul este returnat la GASP prin subrutina EVNTS.

În acest punct sunt verificate condiţiile pentru a determina dacă o rulare de simulare este terminată:

- dacă MSTOP = 0, un eveniment special ce trebuie scris de programator va poziţiona MSTOP la -1 pentru terminarea rulării de simulare;
- dacă MSTOP > 0, rularea este terminată când TNOW ≥ TFIN, variabila TFIN definind timpul de terminare al rulării de simulare. În acest caz, se folosesc automat subrutinele SUMRY şi OTPUT: SUMRY pentru calcularea şi imprimarea rapoartelor standard finale şi OTPUT scrisă de programator pentru calcularea şi tipărirea altor informaţii dorite de utilizatori.

Când se foloseşte prima condiţie de oprire MSTOP = 0, dacă se doreşte utilizarea subrutinelor SUMRY şi OTPUT setăm variabila NORPT = 0, iar dacă nu, setăm NORPT > 0.

Variabilele NRUN şi NRUNS sunt folosite pentru a indica numărul rulărilor de simulare făcute şi respectiv, numărul de rulări rămase incluzând-o şi pe cea în curs de execuţie.

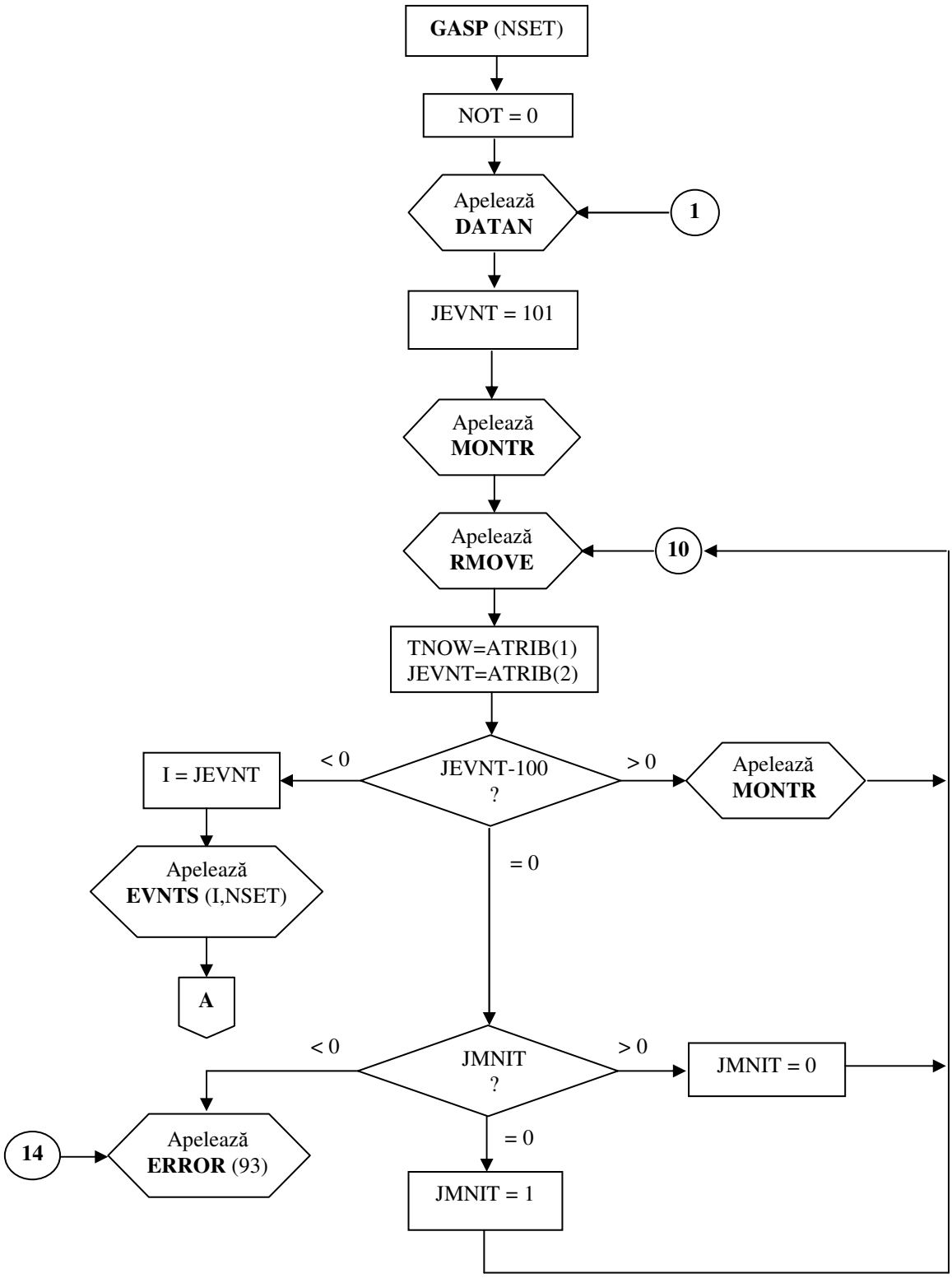
Aceste funcţii sunt ilustrate în schema logică detaliată a subrutinei GASP (figura 7).

Când NRUNS = 1, controlul revine la programul principal, unde este responsabilitatea programatorului să facă una din acţiunile următoare:

- cheamă EXIT;
- reiniţializează variabilele non-GASP şi apoi cheamă GASP;
- cheamă GASP (nu se doreşte reiniţializarea).

Atâta timp cât NRUNS > 1 este chemată subrutina DATAN şi o variabilă de control NEP specifică variabilele GASP care trebuie să fie reiniţializate. Prin această procedură se poate face un număr de rulări cu aceleaşi valori iniţiale ale variabilelor non-GASP şi apoi încă un set de rulări cu valori iniţiale noi.

Când NRUNS = 0, simularea este terminată şi execuţia programului este oprită.


GASP (NSET)
NOT = 0
Apelează DATAN
1
JEVNT = 101
Apelează MONTR
Apelează RMOVE
10
TNOW=ATRIB(1)
JEVNT=ATRIB(2)
JEVNT-100 ?
< 0
I = JEVNT
Apelează EVNTS (I,NSET)
A
> 0
Apelează MONTR
= 0
JMNIT ?
< 0
JMNIT = 0
> 0
= 0
JMNIT = 1
14
Apelează ERROR (93)

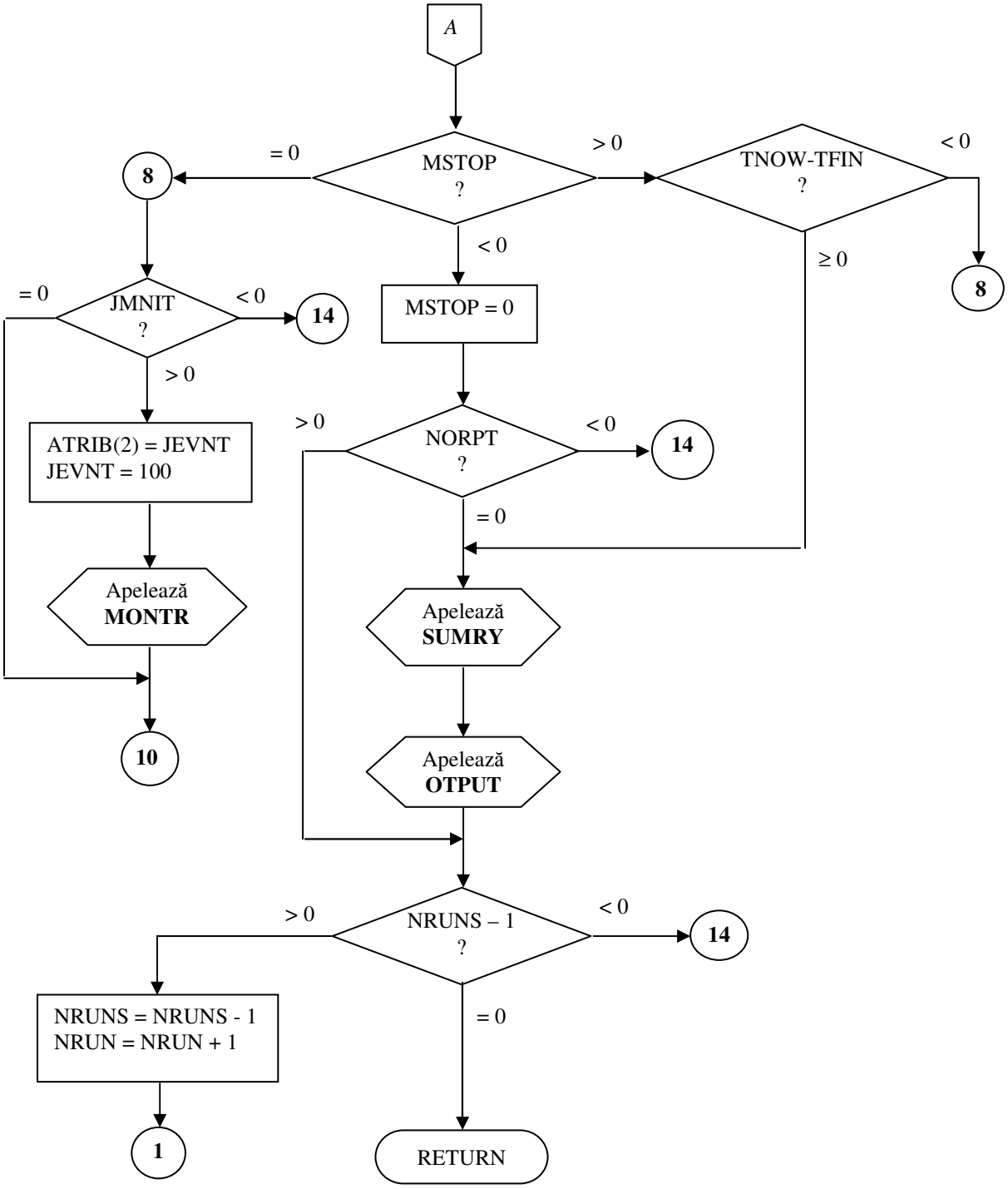


Fig. 7.- *Schema detaliată a subrutinei GASP*

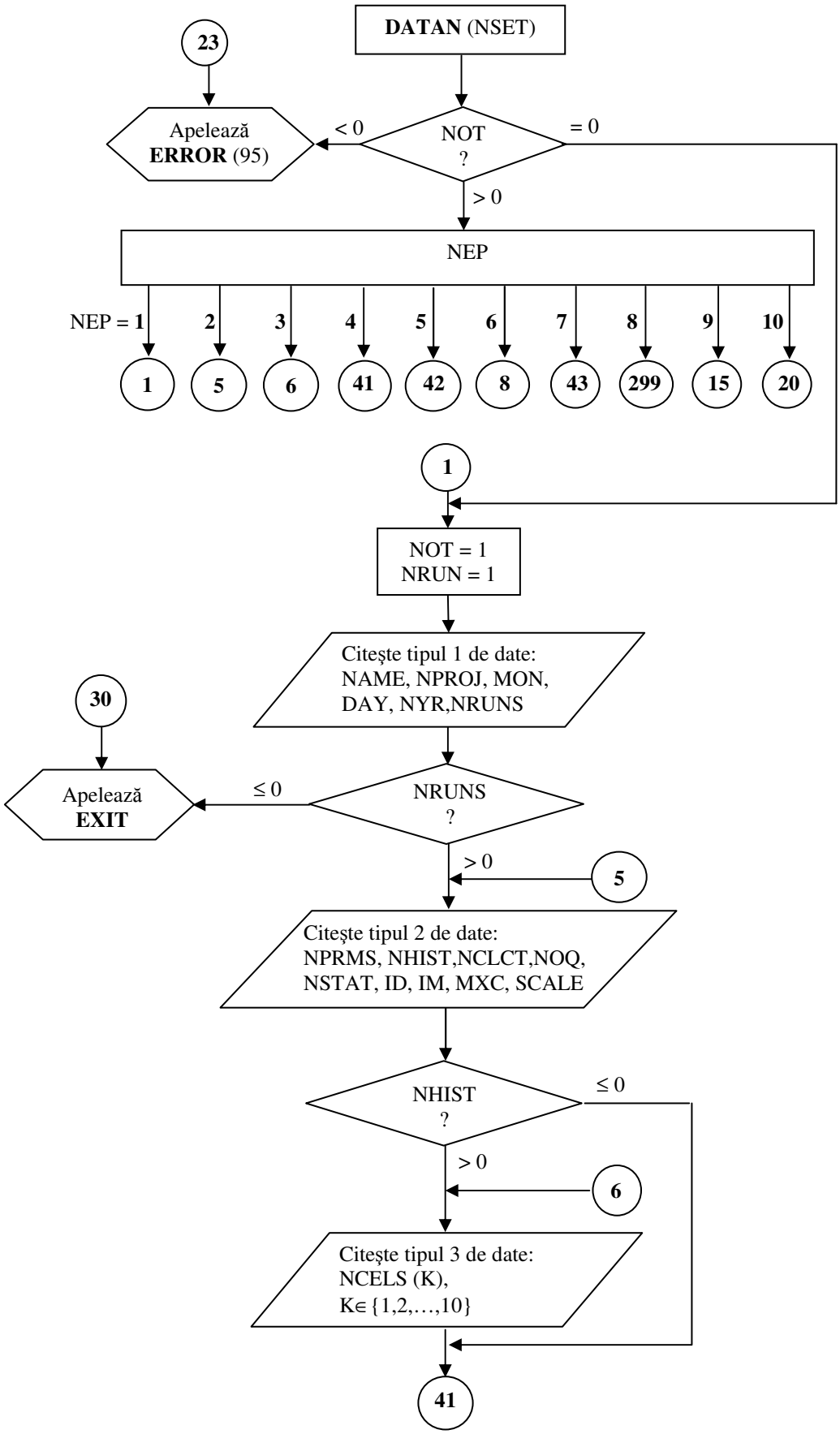
F2. Inițializarea datelor

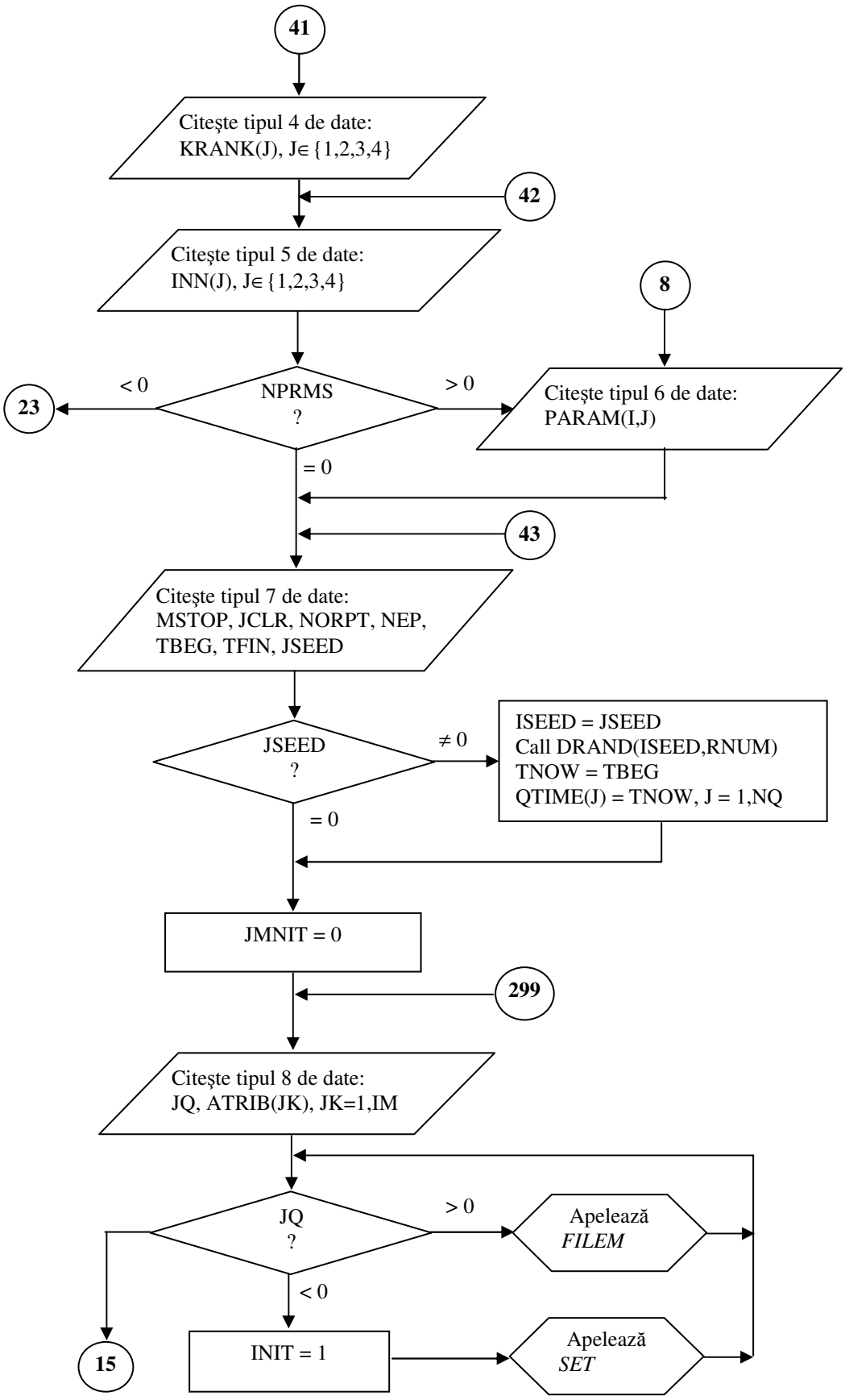
Pentru inițializarea datelor se foloseşte subrutina DATAN care este apelată numai de subrutina GASP şi realizează următoarele funcții generale (figura 8):

- inițializează variabilele GASP care permit începerea simulării;
- oferă posibilitatea citirii valorilor pentru constantele utilizate în diferite subrutine GASP;
- asigură citirea înregistrărilor fişierului inițial şi a evenimentelor endogene şi exogene care se produc în timpul rulării programului de simulare. Evenimentele cauzate de condiții externe sunt referite ca evenimente exogene iar cele generate din celelalte evenimente ale simulării sunt denumite evenimente endogene;
- permite efectuarea unor rulări succesive prin folosirea opțiunii de citire a noilor valori pentru constantele GASP şi /sau a noilor înregistrări inițiale.

Variabilele de control NOT şi NEP determină câte tipuri şi ce tipuri de date sunt folosite. Subrutina GASP setează NOT = 0. Când NOT = 0 sunt folosite toate tipurile de date şi apoi se setează NOT = 1, ceea ce implică obligativitatea verificării variabilei NEP în rularea următoare. Valoarea lui NEP dă cel mai mic tip de date cerut într-o rulare de simulare.

De exemplu, când NOT > 0, dacă NEP = 1 sunt cerute toate cele 8 tipuri de date; dacă NEP = 5 sunt cerute tipurile de date 5, 6, 7 şi 8; dacă NEP = 9 sau 10 tipurile de date sunt ignorate.


DATAN (NSET)
23
Apelează
ERROR (95)
< 0
NOT
?
= 0
> 0
NEP
NEP = 1 2 3 4 5 6 7 8 9 10
1 5 6 41 42 8 43 299 15 20
1
NOT = 1
NRUN = 1
Citeşte tipul 1 de date:
NAME, NPROJ, MON,
DAY, NYR,NRUNS
30
Apelează
EXIT
≤ 0
NRUNS
?
> 0
5
Citeşte tipul 2 de date:
NPRMS, NHIST,NCLCT,NOQ,
NSTAT, ID, IM, MXC, SCALE
NHIST
?
≤ 0
> 0
6
Citeşte tipul 3 de date:
NCELS (K),
K∈ {1,2,…,10}
41


41
Citeşte tipul 4 de date:
KRANK(J), J∈ {1,2,3,4}
42
Citeşte tipul 5 de date:
INN(J), J∈ {1,2,3,4}
8
< 0
NPRMS
?
> 0
23
Citeşte tipul 6 de date:
PARAM(I,J)
= 0
43
Citeşte tipul 7 de date:
MSTOP, JCLR, NORPT, NEP,
TBEG, TFIN, JSEED
JSEED
?
≠ 0
ISEED = JSEED
Call DRAND(ISEED,RNUM)
TNOW = TBEG
QTIME(J) = TNOW, J = 1,NQ
= 0
JMNIT = 0
299
Citeşte tipul 8 de date:
JQ, ATRIB(JK), JK=1,IM
JQ
?
> 0
Apelează
FILEM
< 0
15
INIT = 1
Apelează
SET

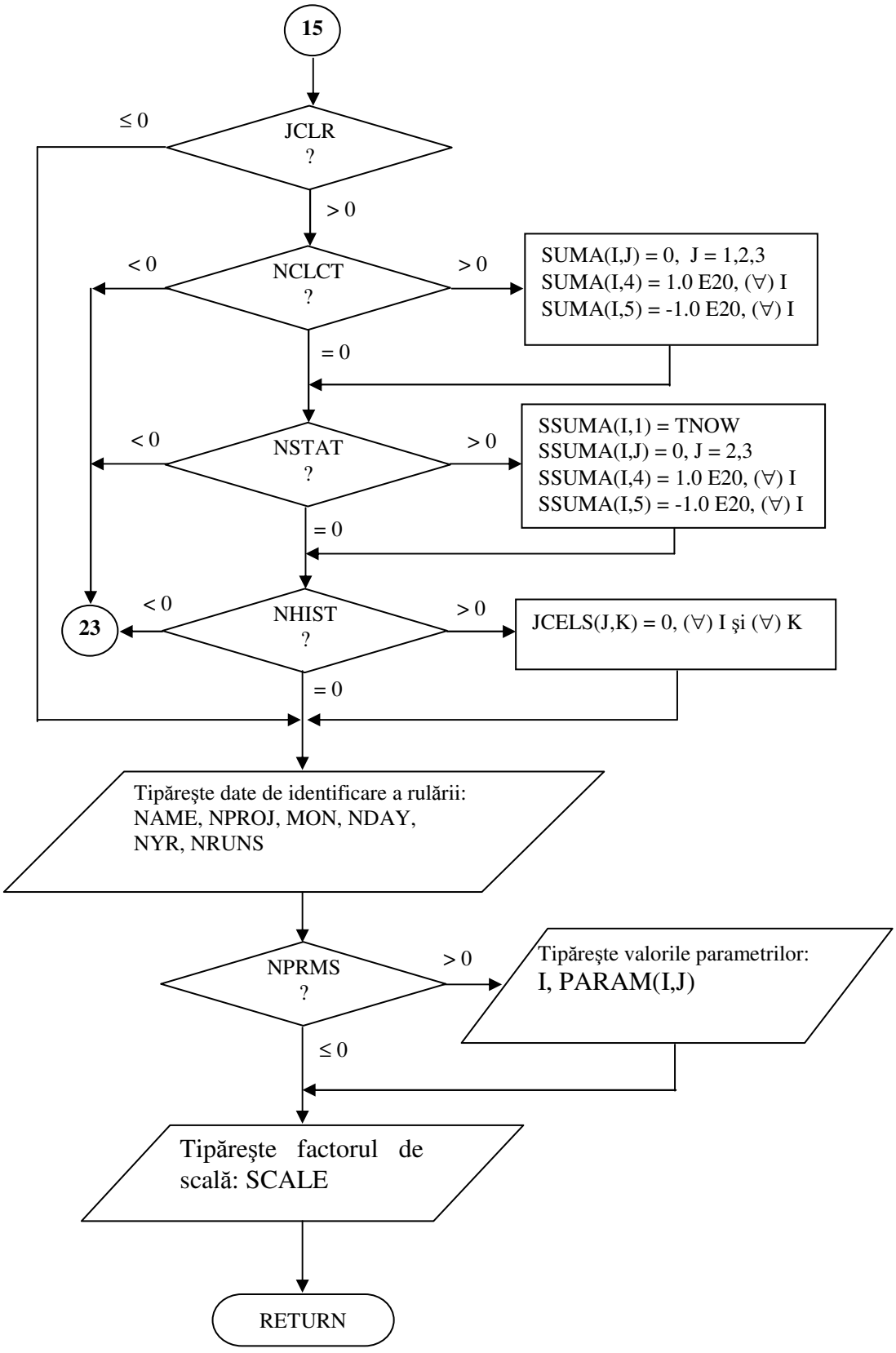


Fig. 8. – *Schema logică de calcul a subrutinei DATAN*

Primul tip de date citite şi iniţializate de subrutina DATAN sunt variabilele GASP ale căror cod şi semnificaţie sunt prezentate în tabelul 2.

Tabel 2

Cod variabilă	Semnificaţie variabilă
NAME	Nume programator
NPROJ	Număr proiect
MON	Număr lună
NDAY	Număr zi
NYR	Număr an
NRUNS	Număr total de rulări

Unele rulări pot fi terminate prin instrucţiunea CALL EXIT atribuind o valoare zero sau negativă variabilei NRUNS.

Dacă NRUNS > 0 se citeşte tipul 2 de date care iniţializează variabilele GASP prezentate în tabelul 3.

Tabel 3

Cod variabilă	Semnificaţie variabilă
NPRMS	Număr de seturi de parametri ale distribuţiilor statistice
NHIST	Număr de histograme cerute pentru acea simulare
NCLCT NSTAT	Număr de variabile pentru care sunt calculate statistici în subrutinele COLCT, respectiv în TMST
ID	Număr de coloane din NSET
IM	Număr maxim de atribute asociate oricărei înregistrări din NSET
NOQ	Număr de fişiere din NSET
MXC	Număr maxim de celule folosite în histograme
SCALE	Parametru folosit pentru a evita erori de trunchiere datorate utilizării unor variabile în virgulă fixă

Apoi se testează numărul de histograme utilizate în simulare; dacă NHIST > 0, subrutina DATAN citeşte tipul 3 de date şi iniţializează numărul de celule utilizate de fiecare histogramă.

Variabila NCELS(K) indică numărul de celule necesare pentru histograma K, neincluzând celulele de început şi de sfârşit şi deci verifică relația:

NCELS(K) ≤ MXC - 2

Dacă nu se doreşte utilizarea nici unei histograme, tipul 3 de date nu este cerut.

Tipurile 4 şi 5 de date sunt utilizate în mod obligatoriu împreună. Tipul 4 de date conține valorile vectorului KRANK, variabila KRANK(J) specificând atributul după care este sortat fişierul J. Tipul 5 de date, indică valorile pentru vectorul INN, variabila INN(J) specificând codurile pentru procedura de ordonare: 1 pentru LVF şi 2 pentru HVF, aşa cum a mai fost menționat anterior. De exemplu, dacă KRANK(2) = 3 şi INN(2) = 1, atunci fişierul 2 se va sorta crescător (LVF) după atributul 3, deci înregistrările cu valori mici ale atributului 3 sunt prioritare.

Tipul 6 de date se foloseşte numai dacă sunt necesari parametri distribuțiilor statistice pentru generarea variabilelor aleatoare. Dacă NPRMS ≥ 1, parametri stocați în zona PARAM sunt asociați la câte un subprogram pentru a genera variabile aleatoare conform distribuțiilor statistice.

Subrutina DATAN citeşte apoi tipul 7 de date pentru a inițializa variabilele de control a căror semnificații se precizează în tabelul 4.

Chiar dacă un program scris de utilizator nu necesită generarea unui număr aleator, subrutina DATAN cere ca JSEED să nu fie zero pentru a inițializa timpul curent la TBEG.

Variabila de control JMNIT = 0 indică faptul că nu se cere tipărirea datelor despre fiecare eveniment aşa cum se produce.

Tipul 8 de date se foloseşte pentru a inițializa aria NSET şi pentru a însera înregistrările inițiale în NSET. Fiecare dată tip 8 conține numărul fişierului JQ şi atributele înregistrării ATRIB. La început JQ < 0 şi aria NSET va fi inițializată.

Tabel 4

Cod variabilă	Semnificație variabilă
MSTOP = 0 MSTOP > 0 MSTOP < 0	- Evenimentul sfârşit de simulare este furnizat de programator; - Simularea e termină când TNOW ≥ TFIN; - Programatorul a indicat că simularea s-a terminat şi se scot rapoarte finale la cerere;
JCLR = 0 JCLR = 1	- Nu se inițializează variabilele de colectare a statisticilor; - Se inițializează variabilele de colectare a statisticilor (SUMA, SSUMA, JCELS);
NORPT = 0 NORPT = 1	- Se tipăresc rapoarte finale; - Nu este cerut nici-un rezumat final;
NEP	Tipul de date de la care începe următoarea rulare de simulare;
TBEG	Timpul de începere a simulării (valoarea inițială a lui TNOW);
TFIN	Timpul de terminare a simulării, dacă MSTOP > 0;
JSEED < 0 JSEED > 0	Sursa numărului aleator ISEED este setată la JSEED;

Când JQ > 0 valorile din ATRIB sunt înserate într-o înregistrare care va fi scrisă în fişieruI JQ, într-o poziție conformă cu atributul de plasare KRANK(JQ) şi cu procedura de ordonare a înregistrărilor INN(JQ).

O valoare JQ = 0 în datele de tip 8 semnifică faptul că toate datele de inițializare au fost citite. Deoarece GASP are nevoie de cel puțin un eveniment inițial pentru a începe simularea, rezultă că fişierul de evenimente trebuie să conțină cel puțin o înregistrare.

În continuare DATAN verifică variabila de control JCLR pentru a determina dacă trebuie inițializate variabilele de colectare a statisticilor; dacă JCLR > 0 le inițializează, iar în caz contrar, nu. În ultima parte se obține un output cu datele de identificare a rulării, valorile parametrilor şi factorul de scală pentru a ajuta la o eventuală depanare şi pentru a determina până unde a progresat rularea de simulare.

F3. Stocarea şi regăsirea informației

Mecanismul pentru stocarea şi regăsirea informațiilor necesare pentru realizarea unei simulări este asigurat de subrutinele SET, FILEM, FIND şi RMOVE. Acest mecanism asigură stocarea atributelor asociate entităților şi evenimentelor sub formă de înregistrări, precum şi înlănțuirea înregistrărilor în fişiere distincte pe baza unui atribut de plasare, a unei proceduri de ordonare şi a unui sistem de indicatori.

Variabilele MFE(JO) şi MLE(JQ) indică prima şi respectiv, ultima înregistrare a fiecărui fişier JQ din NSET.

Înlănțuirea înregistrărilor în fişiere este asigurată de indicatorii MX şi MXX care punctează pentru fiecare înregistrare, înregistrarea succesoare şi respectiv, înregistrarea predecesoare. Pentru fiecare fişier, aceşti indicatori au valori prestabilite care indică ultima înregistrare (nu are succesor), MX = 7777 şi respectiv prima înregistrare (nu are predecesor), MXX = 9999. Ultima înregistrare disponibilă din NSET este recunoscută prin valoarea indicatorului MX = 8888.

a) SUBRUTINA SET (JQ, NSET)

Subrutina SET reprezintă nucleul sistemului de memorare şi regăsire a informației şi execută următoarele funcții:

- inițializează zona de stocare a informațiilor;
- actualizează sistemul de indicatori care asigură înlănțuirea înregistrărilor pentru toate operațiile efectuate cu fişiere;
- întreține statistica după numărul înregistrărilor din fiecare fişier.

Datorită complexității schemei logice de calcul şi a dificultății descrierii acesteia se va prezenta doar o schemă generală a subrutinei SET (figura 9).

Dintre valorile statistice calculate pentru fiecare fişier JQ, menționăm:

- numărul curent de înregistrări, NQ(JQ);
- numărul maxim de articole MAXNQ(JQ), înregistrate în perioada [0, TNOW];
- numărul de articole cumulate în timp ENQ(JQ);
- momentul anterior ştergerii sau înregistrării unui articol QTIME(JQ);
- media şi abaterea medie pătratică a numărului de articole din fiecare fişier.

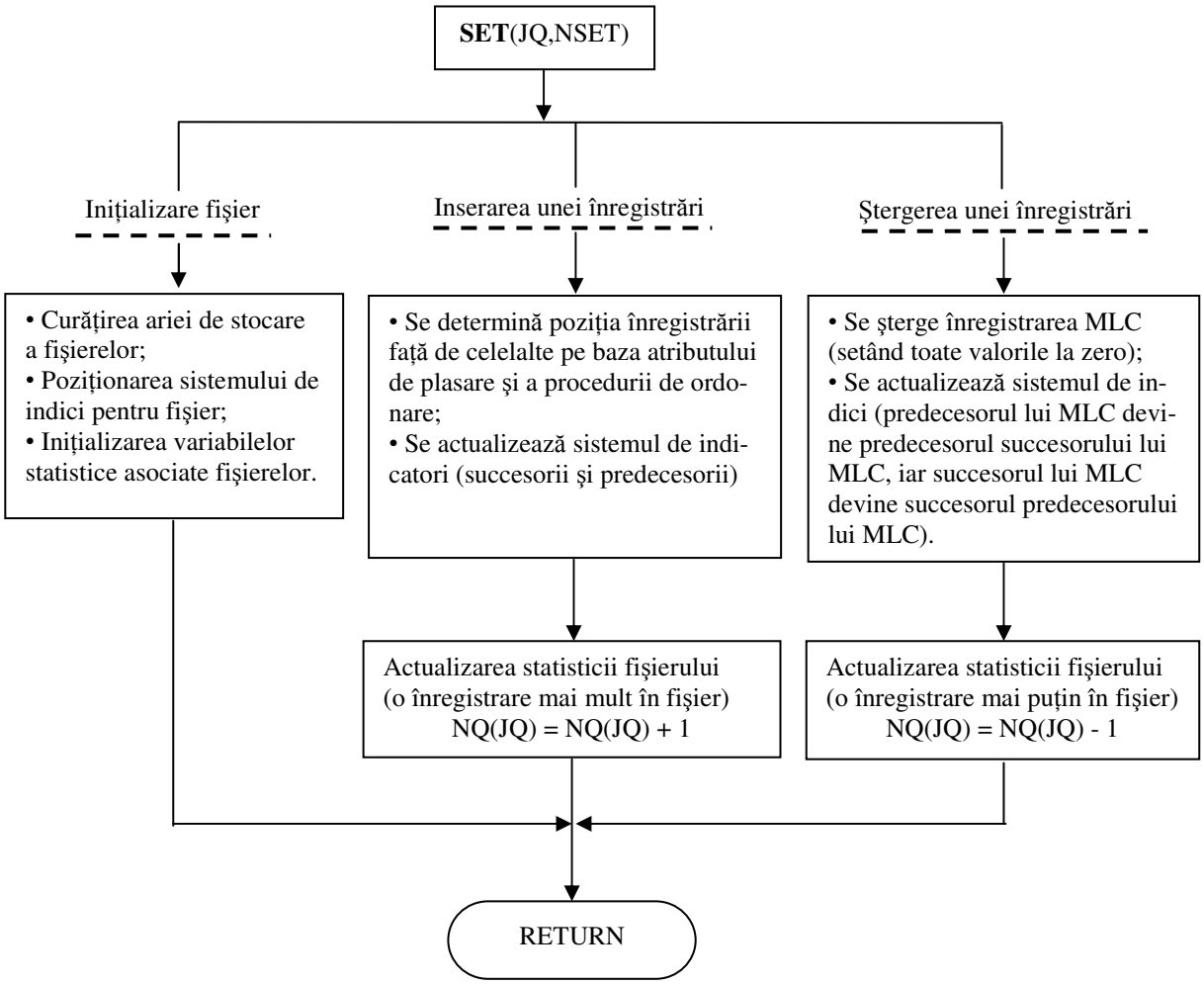


Fig.9.- *Schema de principiu a subrutinei SET*

b) SUBRUTINA FILEM(JQ, NSET)

Subrutina FILEM are rolul de a scrie o înregistrare în fișieruI JQ din aria NSET. Mai întîi se testează dacă prima coloană disponibilă, indicatä de variabila MFA, depășește numărul maxim de coloane ID, prevăzut pentru NSET, caz în care se tipărește un mesaj de eroare și apoi este apelată subrutina ERROR care furnizează informații suplimentare necesare pentru depanare. Schema logică a subrutinei FILEM este prezentată în figura 10.

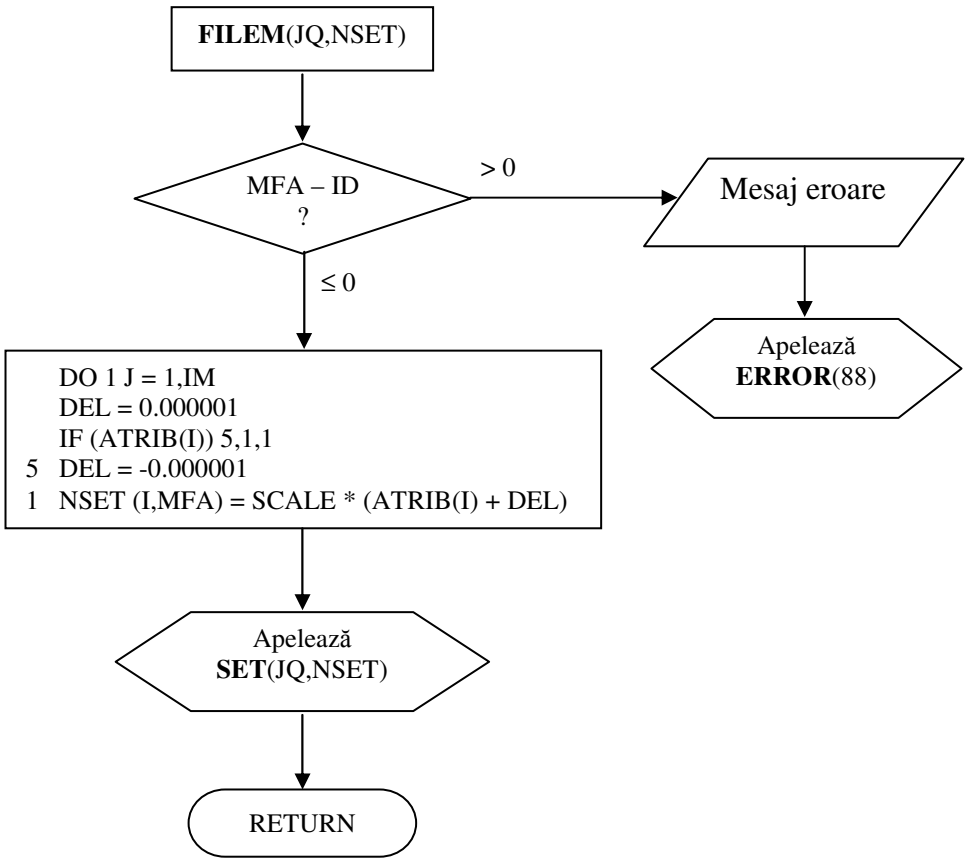


Fig.10.- *Schema logică a subrutinei FILEM*

Înainte de scriere, valoarea fiecărui atribut este multiplicată cu un factor de scală, indicat de programator pentru a exclude o eroare de trunchiere semnificativă. Acest lucru este necesar deoarece în această subrutină se trece de la virgulă mobilă la virgulă fixă pentru a reduce necesarul de memorie.

Dacă există coloane disponibile, subrutina FILEM înscrie atributele în prima coloană disponibilă indicată de variabila MFA şi cheamă apoi subrutina SET care actualizează noua coloană disponibilă precum şi sistemul de indici care asigură înlănțuirea corectă a înregistrărilor în fişier.

c) Subrutina RMOVE (KCOL, JQ, NSET)

Această subrutină este apelată pentru a lua înregistrarea indicată de argumentul KCOL din fişierul JQ şi de a o stoca în vectorul ATRIB, unde valorile atributelor sunt transformate în virgulă mobilă (prin împărțire cu factorul de scală) pentru a putea fi utilizate în calcule.

Apoi este chemată subrutina SET care şterge înregistrarea indicată de KCOL din fişierul JQ şi actualizează sistemul de indicatori pentru a asigura o înlănțuire corectă a înregistrărilor rămase. Schema logică a subrutineiRMOVE este prezentată în figura 11.

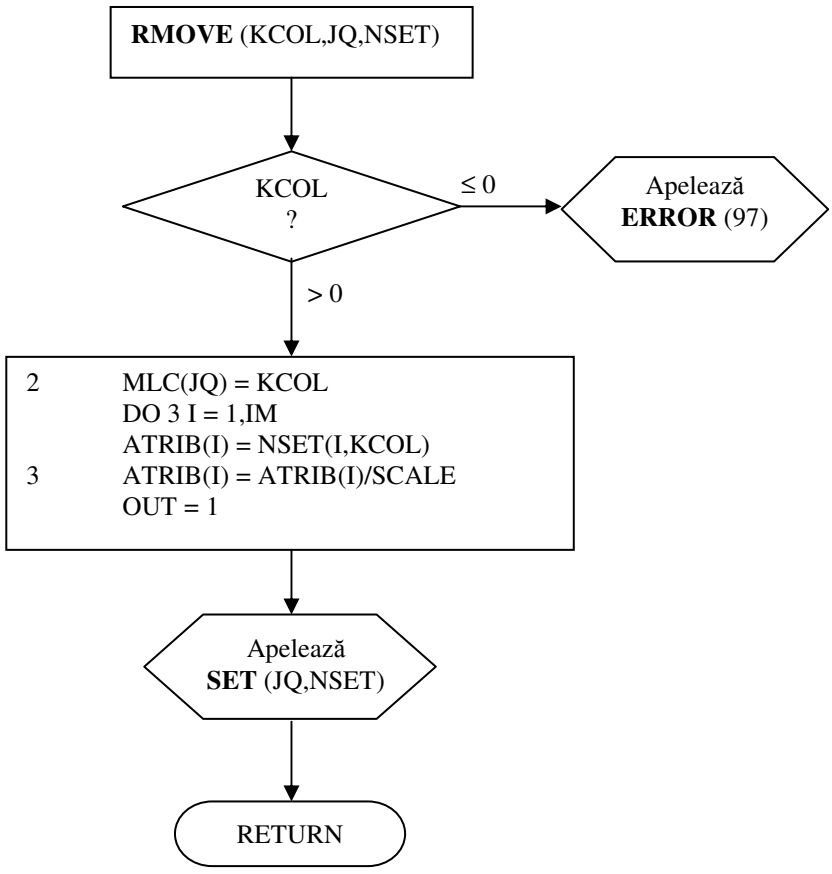


Fig.11.- *Schema logică a subrutinei RMOVE*

d) Subrutina FIND (XVAL, MCODE, JQ, JATT, KCOL, NSET)

Această subrutină este folosită pentru a localiza înregistrarea (KCOL) din fişierul JQ, din aria NSET, care conţine un atribut (JATT) a cărui valoare (XVAL) îndeplineşte anumite condiţii fixate prin MCODE .

Pentru a folosi subrutina FIND, programatorul trebuie să specifice următoarele argumente:

- XVAL - valoarea utilizată pentru a localiza înregistrarea care conţine o valoare atribut ce îndeplineşte o condiţie specificată prin MCODE;
- MCODE - un cod care specifică relaţia dintre valoarea atributului şi XVAL, care poate fi:

  MCODE = 1: valoarea maximă mai mare decât XVAL;

  MCODE = 2: valoarea minimă mai mare decât XVAL;

  MCODE = 3: valoarea maximă mai mică decât XVAL;

  MCODE = 4: valoarea minimă mai mică decât XVAL;

  MCODE = 5: valoarea atributului egală cu XVAL;
- JQ - fişierul care conţine înregistrarea ce trebuie localizată;
- JATT - număr de atribut, de succesor sau de predecesor folosit în localizarea înregistrării;

Înregistrarea care conţine rezultatul căutării este specificată în KCOL, iar dacă în fişierul JQ nu există nici o înregistrare care să satisfacă condiţia specificată, atunci KCOL este setat la zero.

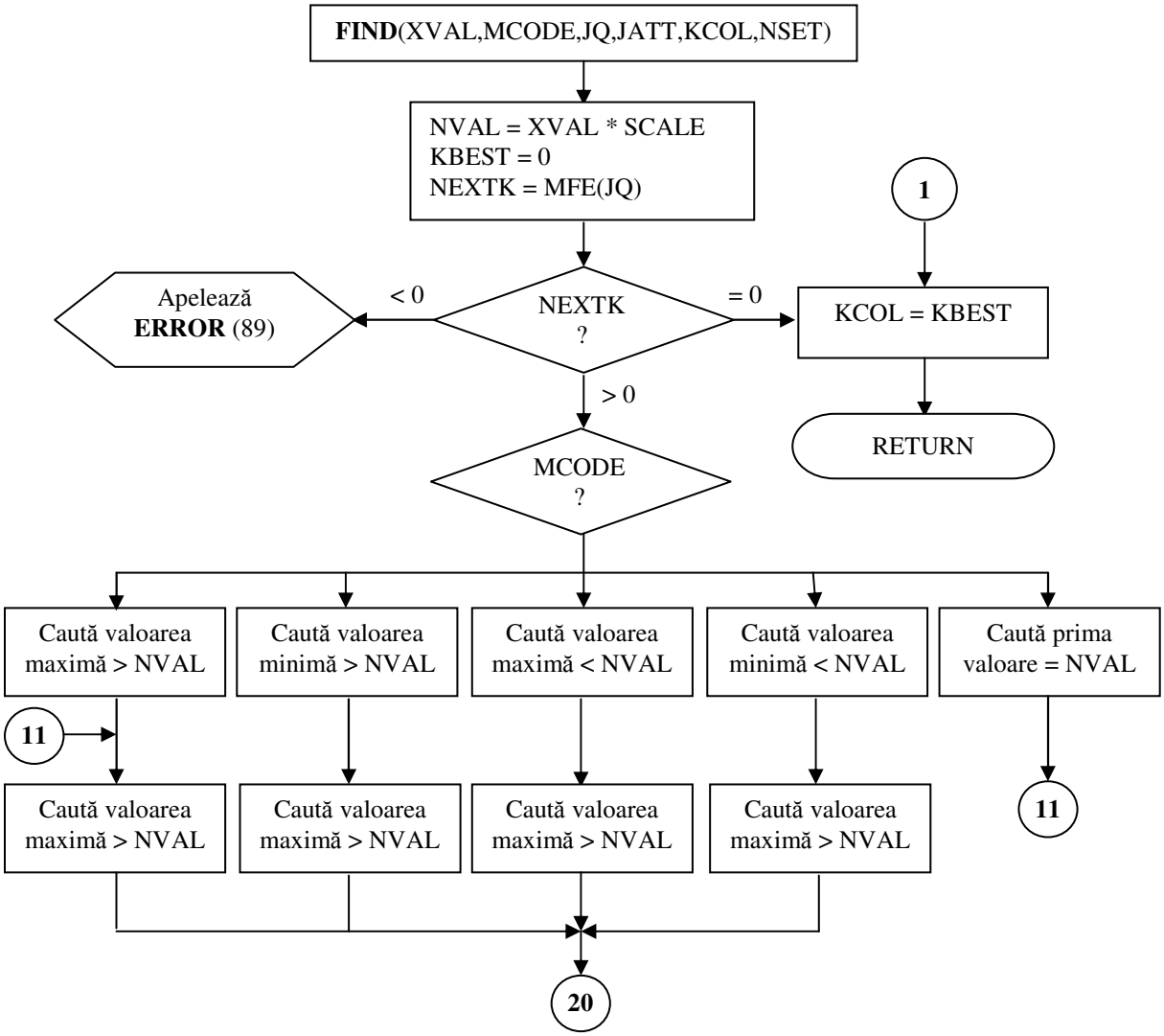
De exemplu, să presupunem că se doreşte găsirea înregistrării care conţine un articol al cărui preţ maxim să fie de 10$, preţurile fiind stocate în atributul 4 al înregistrărilor din fişierul 2. Instrucţiunea folosită pentru căutarea înregistrării va fi construită astfel:

CALL FIND (10., 3, 2, 4, KCOL, NSET)

iar înregistrarea găsită, **KCOL,** va fi stocată în vectorul ATRIB pentru a fi folosită cu instrucţiunea:

CALL RMOVE (KCOL, 2, NSET).

Schema logică a subrutinei FIND este prezentată în figurila 12.


FIND(XVAL,MCODE,JQ,JATT,KCOL,NSET)
NVAL = XVAL * SCALE
KBEST = 0
NEXTK = MFE(JQ)
1
Apelează
ERROR (89)
< 0
NEXTK
?
= 0
KCOL = KBEST
> 0
RETURN
MCODE
?
Caută valoarea
maximă > NVAL
Caută valoarea
minimă > NVAL
Caută valoarea
maximă < NVAL
Caută valoarea
minimă < NVAL
Caută prima
valoare = NVAL
11
Caută valoarea
maximă > NVAL
Caută valoarea
maximă > NVAL
Caută valoarea
maximă > NVAL
Caută valoarea
maximă > NVAL
11
20

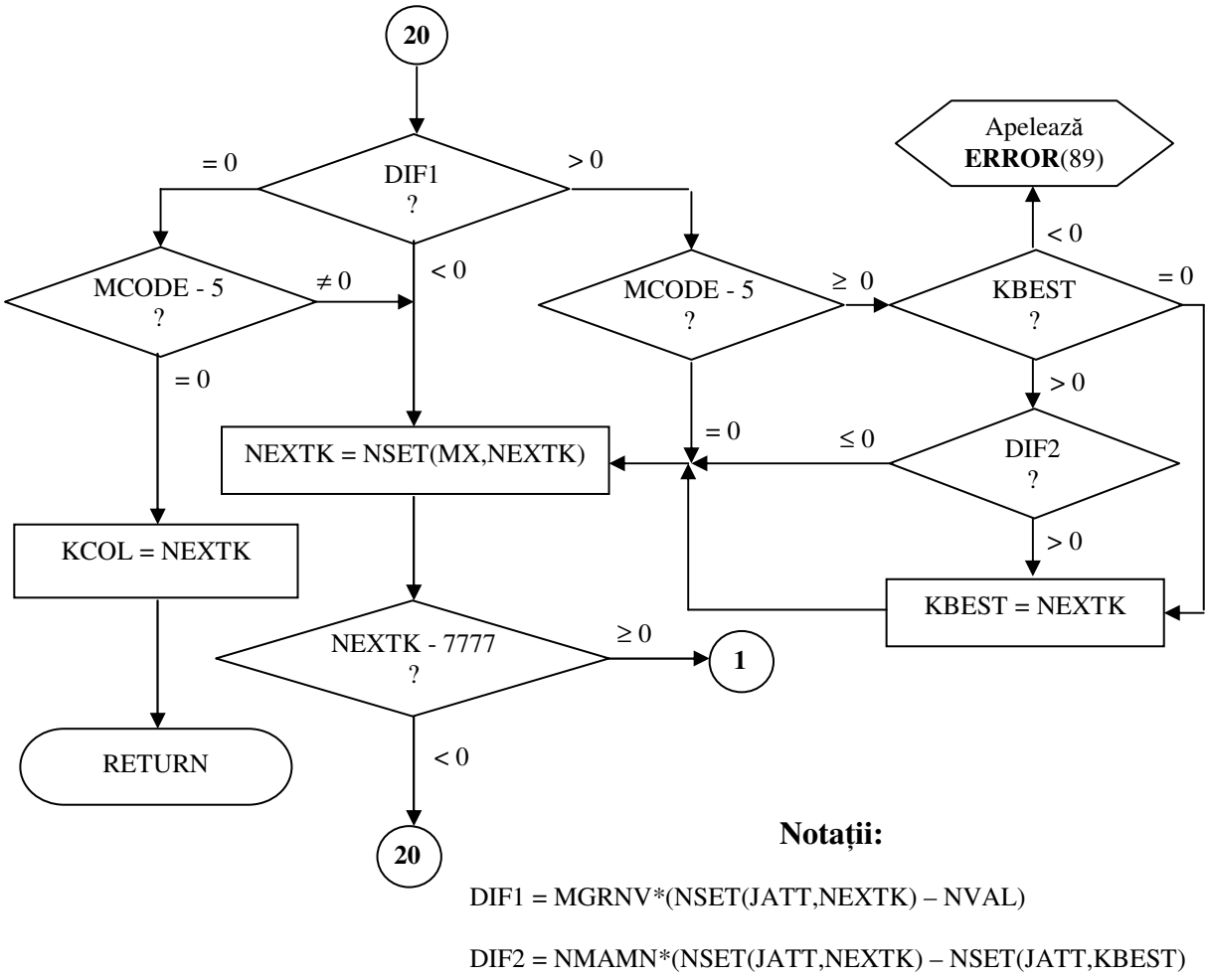


Fig.12.- *Schema logică de calcul pentru subrutina FIND*

F4. Colectarea datelor

În timpul simulării subrutinele COLCT şi TMST sunt folosite pentru colectarea datelor în vederea estimării parametrilor distribuţiilor care descriu variabilele simulării iar subrutina HISTO este folosită pentru construirea de histograme care arată frecvenţa de apariţie a valorilor unei variabile într-un interval dat.

a) Subrutina COLCT(X, N, NSET)

Această subrutină colectează valorile X pe care le primeşte variabila aleatoare cu codul N în timpul simulării. Când este apelată, COLCT adună valoarea X la suma tuturor observaţiilor anterioare cu cod N, adună X^2 la suma pătratelor observaţiilor anterioare, adună 1 la numărul de observaţii şi determină dacă X este valoarea minimă sau maximă observată

până în acel moment. Aceste valori sunt stocate în SUMA (N,J), J ∈ {1,2,3,4,5}.

Schema logică de calcul a acestei subrutine este ilustrată în figura 13.

Spre exemplu, instrucțiunea CALL COLCT (5., 3, NSET) determină următoarele calcule pentru variabila cu codul 3:

SUMA (3, 1) = SUMA (3,1) + 5

SUMA (3, 2) = SUMA (3, 2) + 5^2

SUMA (3, 3) = SUMA (3,3) + 1

SUMA (3, 4) = AMIN (SUMA (3,4), 5)

SUMA (3, 5) = AMAX (SUMA (3, 5), 5).

Numărul maxim de variabile pentru care sunt colectate date este fixat de parametrul NCLCT. Aceste valori sunt folosite apoi de subrutina SUMRY la calcularea şi imprimarea mediei pătratice, a numărului de observări, a minimului şi maximului variabilei cu codul N.

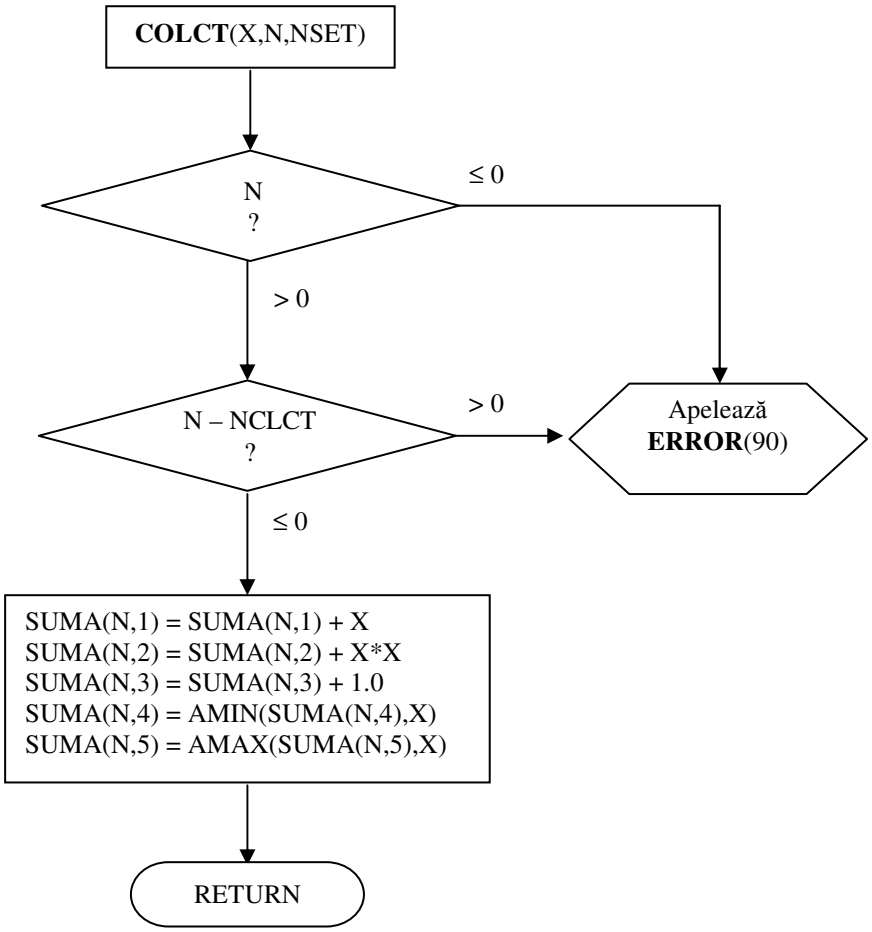


Fig.13.- *Schema logică a subrutinei COLCT*

b) Subrutina TMST(X, T, N, NSET)

Această subrutină se foloseşte dacă valoarea primită de o variabilă se menține aceeaşi pe toată perioada dintre două observări succesive. Când este apelată, colectează valoarea X observată a variabilei cu codul N pe perioada TT calculată ca diferență între momentul observării curente T şi momentul observării anterioare, cu relația:

TT = T - SSUMA (N,1)

după care SSUMA (N,1) este actualizată cu momentul curent.

Se calculează valorile X * TT şi X*X*TT care se adună la SSUMA (N,2) şi respectiv la SSUMA (N,3). Apoi se determină dacă valoarea X este minimă sau maximă observată până la momentul curent şi în caz afirmativ valoarea găsită se reține după caz, în SSUMA (N,4) sau respectiv, în SSUMA (N,5).

Numărul maxim de variabile pentru care sunt colectate date este fixat de parametrul NSTAT.

Datele stocate în SSUMA sunt folosite de subrutina SUMRY pentru a calcula şi tipări:

- media şi abaterea medie pătratică a variabilelor în funcție de timp;
- durata totală după care s-au colectat statistici;
- valorile minime şi maxime ale variabilelor observate în timpul simulării.

Valoarea argumentului X trebuie să fie valoarea unei variabile înainte de a fi schimbată printr-un eveniment de simulare.

De exemplu, instrucțiunea CALL TMST(4., TNOW, 3, NSET), determină calcularea intervalului dintre ultimele observări succesive TT = TNOW - SSUMA(N,1), adună TT*4 la SSUMA (3,2) şi execută celelalte calcule indicate mai sus.

Schema logică a subrutinei TMST(X, T, N, NSET) este ilustrată în figura 14.

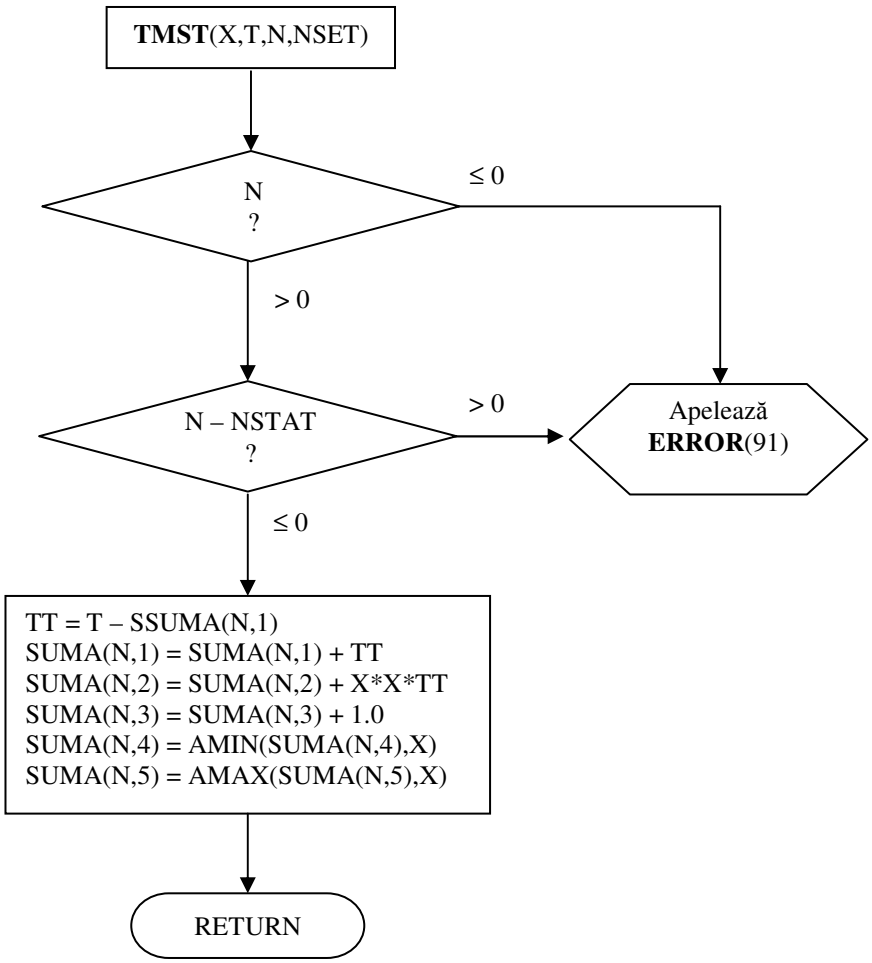


Fig.14.- *Schema logică a subrutinei TMST*

c) Subrutina HISTO (X_1, A, W, N)

Subrutina HISTO calculează şi tabelează frecvențele valorilor unei variabile în cadrul unor intervale specificate.

Semnificația argumentelor este următoarea:

X_1 - o valoare a unei variabile pentru care se calculează frecvența;

A - limita inferioară a intervalului doi;

W - lungimea fiecărui interval (exceptând primul şi ultimul);

N - codul histogramei/variabilei.

O ilustrare a histogramei cu codul N = 1 pentru A = 4.0, W = 3 şi NCELS(1) = 6 este dată în tabelul 5.

Tabel 5

Număr intervale (IC)	1	2	3	4	5	6	7	8
Intervale	(-∝,4)	[4,7)	[7,10)	[10,13)	[13,16)	[16,19)	[19,22)	[22,∝)
Frecvența vectorilor JCELS(1,IC)	3	7	14+1	10	12	4	2	11

Variabila NCELS(1) = 6 indică numărul de intervale pentru histograma 1 a căror lungime este finită (în total sunt 8 intervale).

De exemplu, instrucțiunea CALL HISTO (7.5, 4., 3., 1*)* va determina creşterea cu 1 a valorii JCELS(1,3).

Schema logică a subrutinei HISTO este prezentată în figura 15.

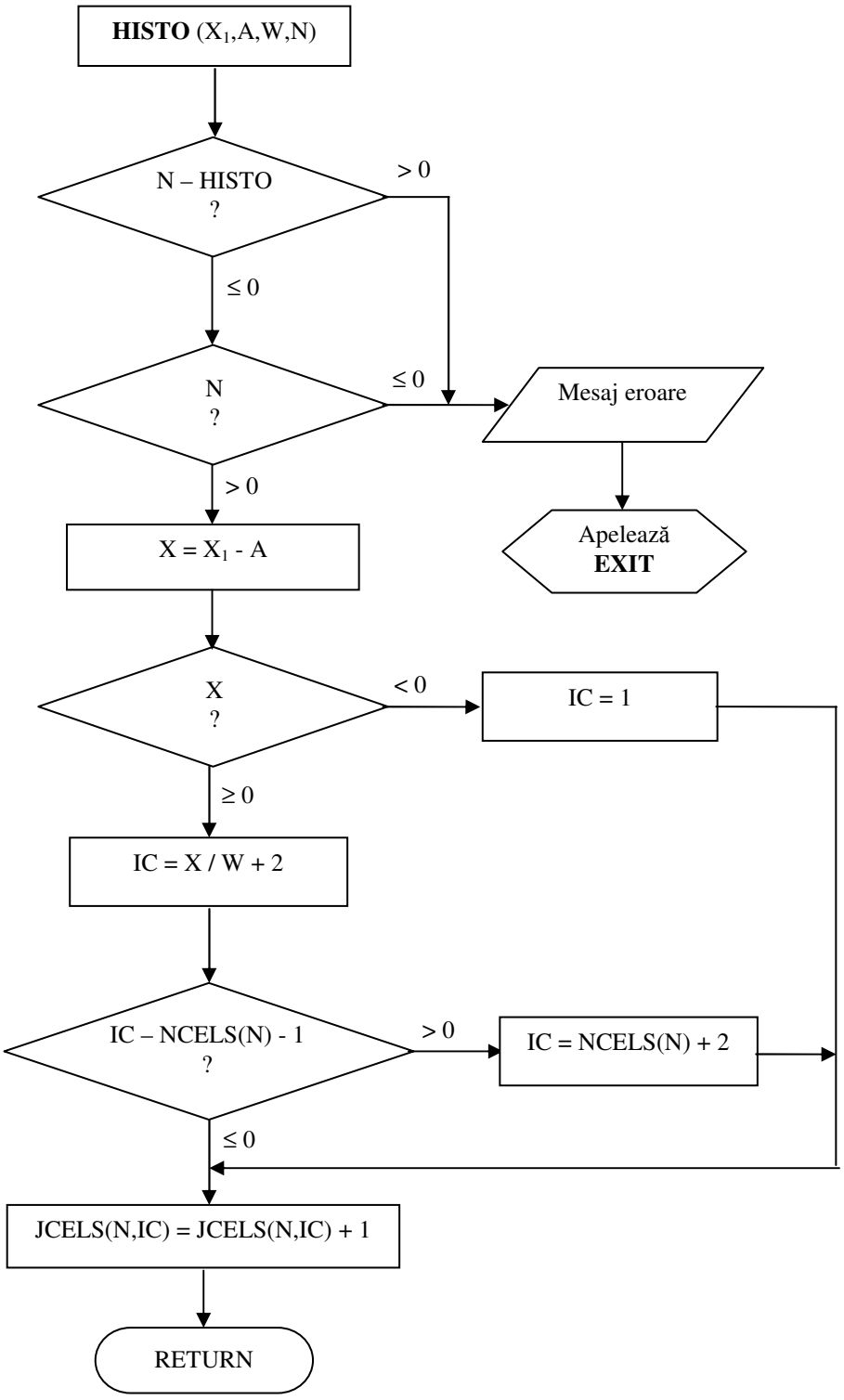


Fig.15.- *Schema de calcul a subrutinei HISTO*

F5. Calcule statistice şi raportări

Subrutinele PRNTQ şi SUMRY pot fi apelate în orice moment în timpul simulării pentru calcularea şi raportarea unor informații statistice.

a) Subrutina PRNTQ (JQ, NSET)

Această subrutină calculează şi tipăreşte pentru fiecare fişier JQ, următoarele statistici:

- media aritmetică cumulată în timp a numărului de înregistrări;
- abaterea medie pătratică a numărului de înregistrări;
- numărul maxim de înregistrări existente în fişier de la ultima sa inițializare.

PRNTQ tipăreşte, de asemenea, conținutul fişierului JQ în ordine, începând cu prima înregistrare MFE (JQ) şi terminând cu ultima înregistrare, MLE (JQ).

Schema logică de calcul pentru subrutina PRNTQ(JQ, NSET) este prezentată în figura 16.

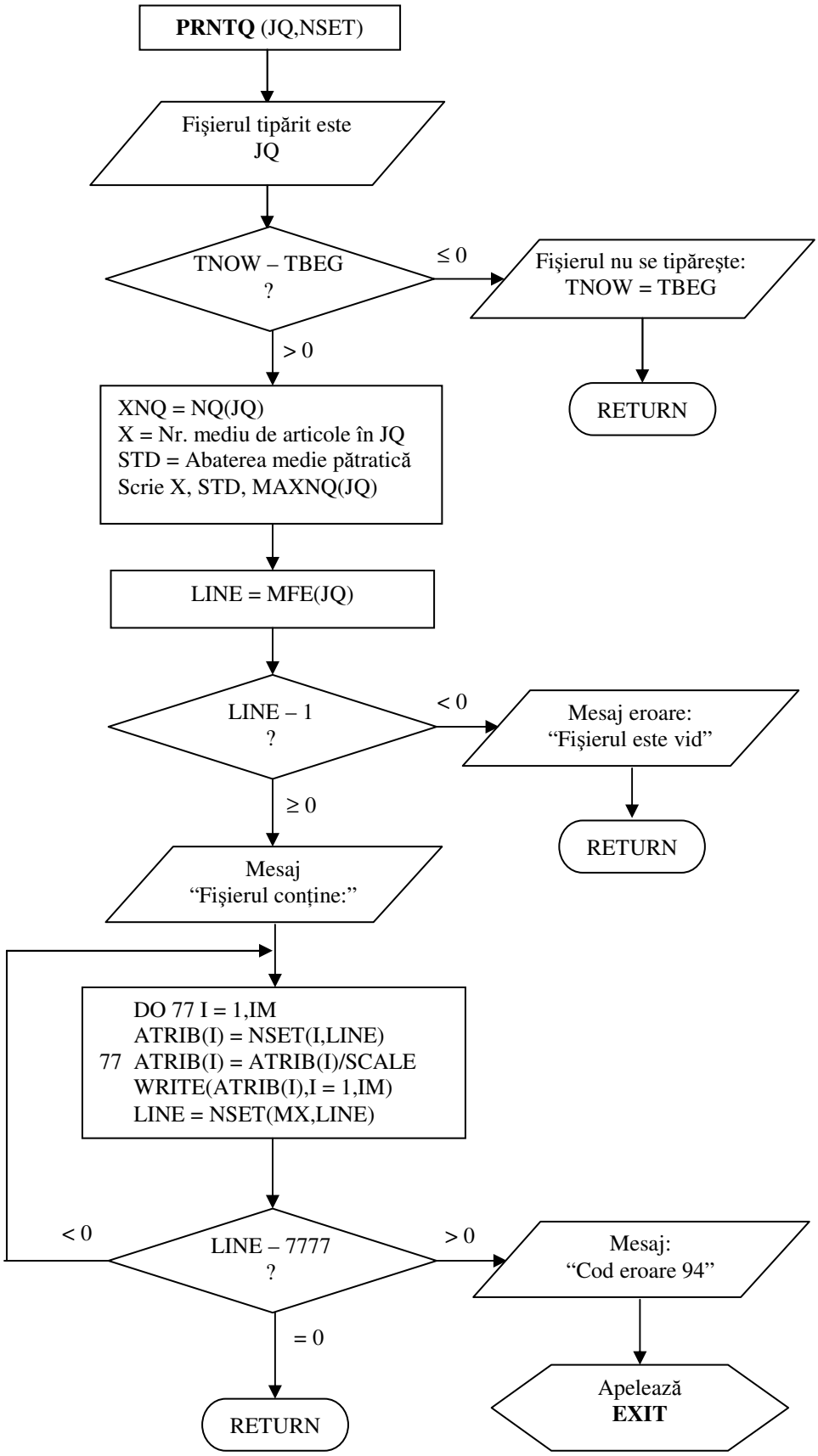


Fig. 16 – *Schema logică a subrutinei PRNTQ*

b) Subrutina SUMRY (NSET)

Subrutina SUMRY este principala subrutină de output din GASP, care prelucrează datele colectate în subrutinele COLCT, TMST şi HISTO şi tipăreşte rezultatele obținute.

Din datele furnizate de subrutina COLCT, subrutina SUMRY calculează şi imprimă (afişează) media, abaterea medie pătratică, numărul de observații, valoarea minimă şi maximă pentru fiecare variabilă pentru care s-au colectat date.

Din datele colectate de subrutina TMST, subrutina SUMRY calculează şi imprimă (afişează) media, abaterea medie pătratică, valoarea minimă şi maximă, timpul cumulat pentru fiecare variabilă pentru care s-au colectat date şi timpul total.

Valorile statistice sunt calculate numai până la ultima schimbare. Pentru a include şi valoarea variabilei la sfârşitul simulării (pe ultima perioadă), subrutina evenimentului "sfârşit de simulare" apelează subrutina TMST pentru fiecare variabilă ce trebuie actualizată.

Din datele colectate şi repartizate pe intervale de subrutina HISTO, subrutina SUMRY tipăreşte datele din histogramă şi apelează apoi subrutina PRNTQ, care prezintă informațiile statistice pentru toate fişierele folosite în simulare.

Schema logică a subrutinei SUMRY(NSET) este prezentată în figura 17.

F6. Monitorizarea şi raportarea erorilor

Subrutinele MONTR şi ERROR ajută utilizatorul să facă unele modificări pentru a obține informații suplimentare în timpul sau la terminarea unei rulări de simulare în vederea depanării şi a verificării modului de funcționare a programelor.

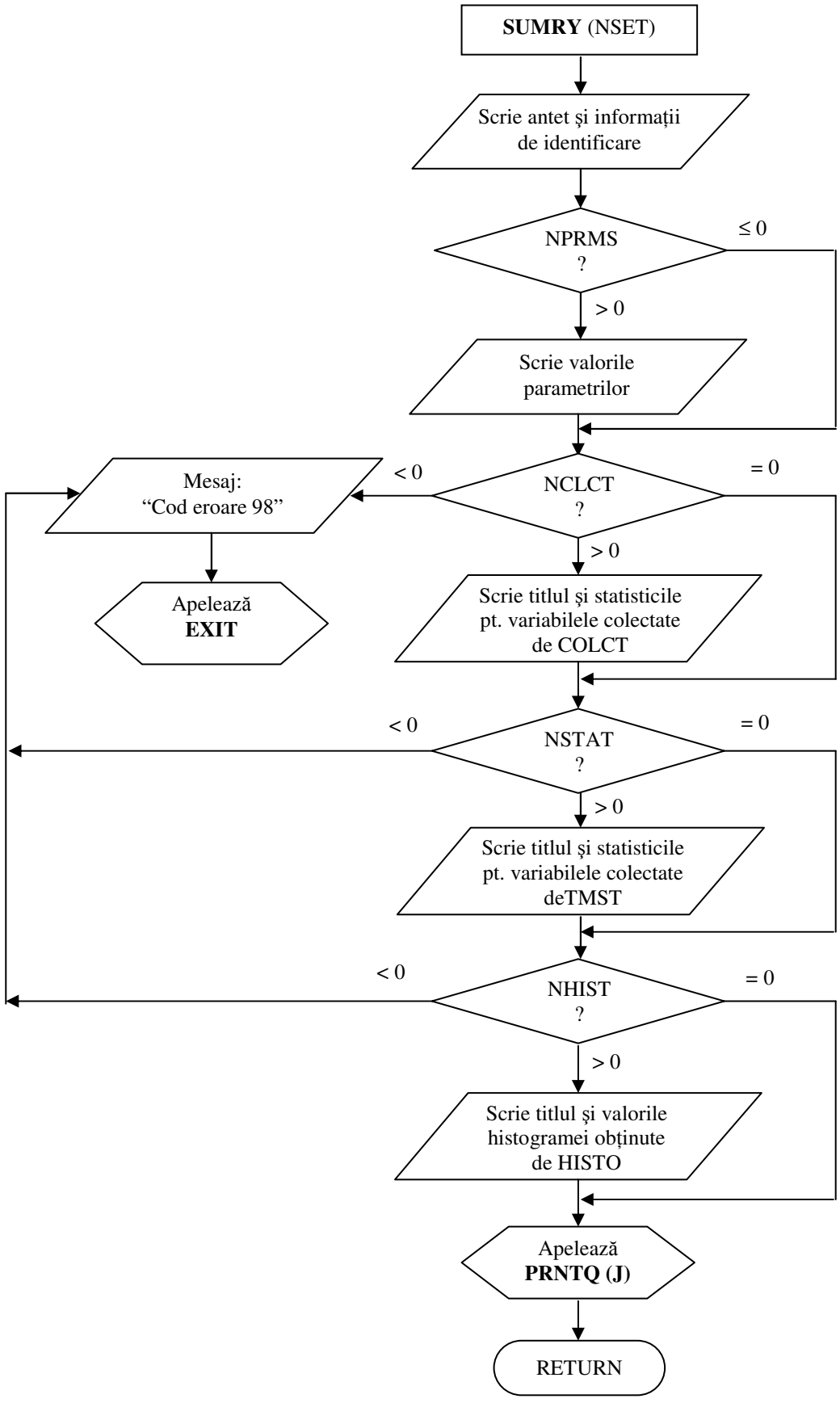


Fig.17. – *Schema logică a subrutinei SUMRY*

a) Subrutina MONTR (NSET)

Prin opțiunea de a imprima toate fişierele sau o succesiune de evenimente ce trebuie luate din fişierul de evenimente, subrutina MONTR oferă capacitatea de conducere selectivă a evenimentelor pentru depanarea şi înțelegerea funcționării programului de simulare. Codurile evenimentelor de monitorizare trebuie înscrise în fişierul de evenimente înaintea începerii simulării.

În felul acesta, un cod 100 determină tipărirea informațiilor despre fiecare eveniment până când se întâlneşte un nou cod 100, iar un cod 101 determină tipărirea tuturor fişierelor din NSET. Schema logică a acestei subrutine este prezentată în figura 18.

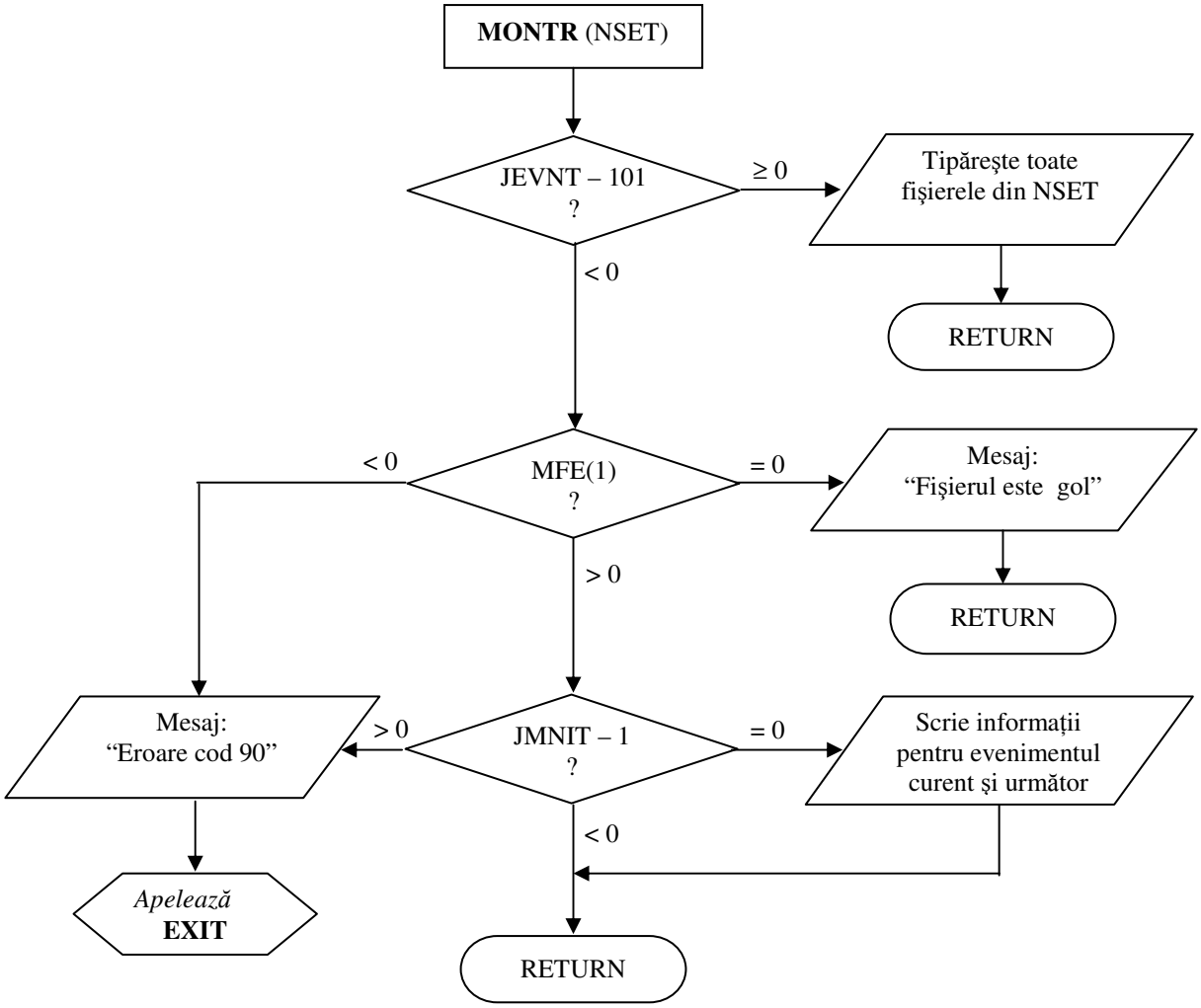


Fig. 18 – *Schema logică a subrutinei MONTR*

b) Subrutina ERROR (J,NSET)

Subrutina ERROR poate fi apelată de orice subrutină când este detectată o eroare, cu excepția subrutinelor PRNTQ, SUMRY şi MONTR care dau propriul lor mesaj de eroare.

Această subrutină realizează următoarele funcții (figura 19).

- tipăreşte momentul curent de simulare TNOW şi codul subrutinei J în care eroarea a fost detectată;
- apelează subrutina MONTR pentru tipărirea fişierelor din NSET;
- apelează subrutina PRNTQ care tipăreşte conținutul fişierului de evenimente;
- apelează subrutina SUMRY care prezintă un sumar al rezultatelor obținute până la momentul la care s-a produs eroarea;
- cheamă subrutina EXIT pentru terminarea simulării.

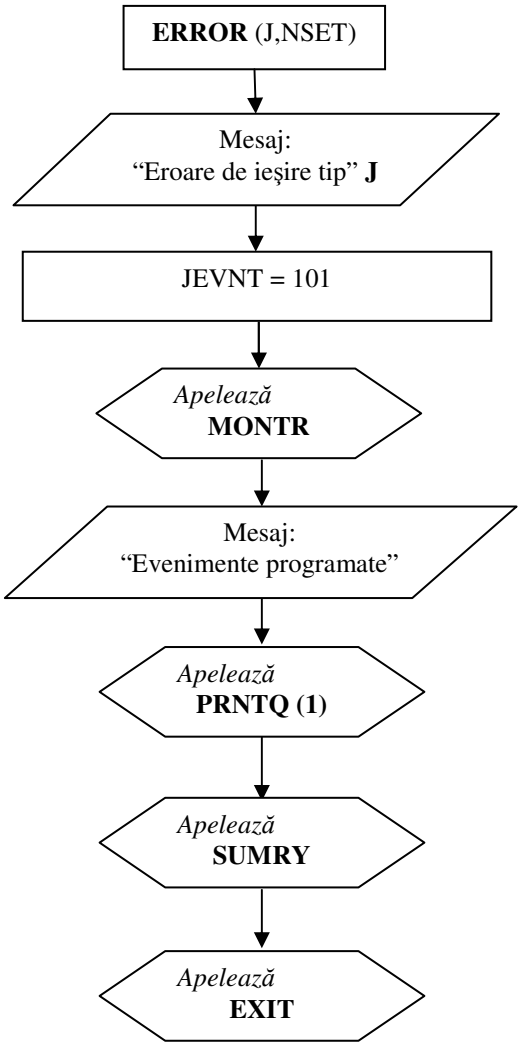


Fig. 19 – *Schema logică a subrutinei ERROR*

F7. Generarea de variabile aleatoare

Pentru generarea de variabile aleatoare trebuie precizate, printr-un vector de parametrizare PARAM (J,I), valorile parametrilor (media, valoarea minimă, valoarea maximă, abaterea medie pătratică) ce caracterizează fiecare tip de distribuție (tabelul 6).

În felul acesta, când se apelează o subrutină de generare, este suficient să se indice doar vectorul care conține parametrii distribuției respective.

Tabel 6

Coloana	1	2	3	4
PARAM (J,I)	Media μ	Valoarea minimă	Valoarea maximă	Abaterea standard σ

Subprogramele nu permit generarea de variabile aleatoare în afara limitelor definite; astfel, dacă valoarea generată este mai mică decât valoarea minimă sau mai mare decât valoarea maximă, atunci valoarea generată se consideră valoarea minimă, respectiv valoarea maximă. Prin alegerea atentă a valorilor minime şi maxime, probabilitatea de generare a valorilor extreme poate deveni suficient de mica în timpul generării valorilor pentru distribuția dată.

Subprogramele pot fi uşor modificate pentru a genera valori dintr-o distribuție trunchiată, folosind metoda respingerii valorilor generate în afara limitelor specificate. Aceasta poate conduce însă la creşterea excesivă a timpului de rulare în cazul unor limite fixate arbitrar, deoarece procentul de respingere a valorilor generate devine foarte mare. Se impune astfel alegerea atentă a limitelor minime şi maxime pe baza testării subrutinelor.

GASP furnizează subprograme pentru generarea de variabile aleatoare prin metoda transformatei inverse pentru distribuțiile uniformă, Erlang, exponențială şi Poisson, iar pentru distribuțiile normală şi lognormală, foloseşte metoda de transformare analitică.

Pentru a ilustra modul de utilizare a vectorului care specifică parametrii unei distribuții, vom prezenta în continuare subrutinele pentru generarea de numere aleatoare pentru distribuția ERLANG şi pentru distribuția POISSON.

a) Funcţia ERLANG(J)

Această funcţie se foloseşte pentru generarea unor numere aleatoare dintr-o distribuţie ERLANG (distribuţia GAMMA cu parametru întreg), a cărei densitate de probabilitate este:

$$f_x(x) = \begin{cases} \dfrac{1}{(k-1)!}\mu(\mu \cdot x)^{k-1} \cdot e^{-\mu \cdot x}, x > 0, k \in N \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad , in\ rest \end{cases}$$

Valorile parametrilor pentru utilizarea funcţiei ERLANG sunt prezentate în tabelul 7.

Tabelul 7

Coloana	1	2	3	4
PARAM (J,I)	1/μ	Valoarea minimă	Valoarea maximă	k

De remarcat că valoarea din coloana 1 nu reprezintă media, aceasta fiind k/μ. Distribuţia exponenţială este un caz particular al distribuţiei ERLANG, pentru k = 1 şi media μ:

$$f_x(x) = \begin{cases} \mu \cdot e^{-\mu \cdot x}, x > 0, \mu > 0 \\ 0 \qquad , in\ rest \end{cases}.$$

Utilizând metoda transformatei inversei se pot genera numere aleatoare x_n, distribuite după legea exponenţială, după cum urmează:

- se integrează funcţia de distribuţie exponenţială şi se egalează cu un număr aleator y_s, uniform distribuit în (0,1):

$$F_x(x) = \int_0^x \mu \cdot e^{-\mu \cdot z} \cdot dz = 1 - e^{-\mu \cdot x} = y_s ,$$

din care rezultă: $x_s = -\dfrac{1}{\mu} \cdot \ln(1 - y_s)$;

- cum 1- y_s este tot un număr aleator se poate înlocui cu y_s şi obținem:

$$x_s = -\frac{1}{\mu} \cdot \ln(y_s);$$

- deoarece o variabilă aleatoare distribuită ERLANG este suma a *k* variabile aleatoare independente distribuite exponențial, iar relațiile următoare sunt echivalente:

$$\sum_{i=1}^{k} x_i = -\frac{1}{\mu} \cdot \sum_{i=1}^{k} \ln(y_i) = -\frac{1}{\mu} \ln \prod_{i=1}^{k} y_i \ ,$$

în schema logică a subrutinei ERLANG se face mai întâi produsul a *k* numere generate uniform în (0,1) şi apoi se determină valoarea variabilei aleatoare de tip ERLANG.

Schema logică pentru generarea unor numere aleatoare dintr-o distribuție ERLANG (distribuția GAMMA cu parametru întreg), a cărei densitate de probabilitate este cunoscută este prezentată în figura 20.

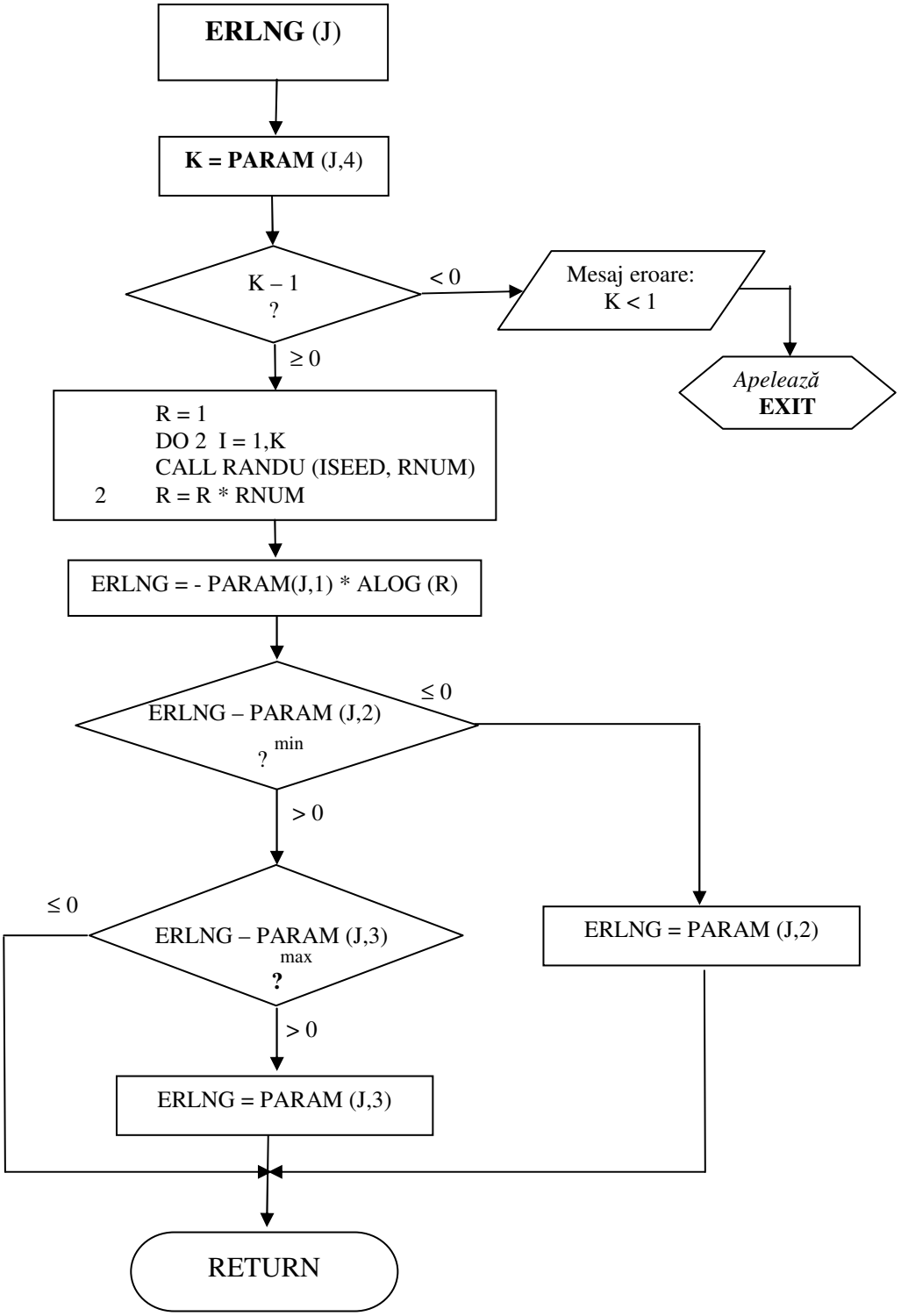


Fig. 20 – *Schema logică a subrutinei ERLNG*

b) Subrutina NPOSN (N, NPSSN)

Această subrutină este folosită pentru generarea de numere aleatoare dintr-o distribuție POISSON discretă, care are forma:

$$P(N=n)=p(n)=\begin{cases}\dfrac{\lambda^{n-n_0}\cdot e^{-\lambda}}{(n-n_0)!}, \lambda>0, n\in\{n_0, n_0+1,\cdots\}\\ 0 \qquad\qquad , in\ rest\end{cases}$$

în care:

- P(N = n) reprezintă probabilitatea ca variabila aleatoare N să se producă de n ori într-o unitate de timp;
- λ este valoarea medie;
- n_0 este valoarea minimă.

Parametrii utilizați pentru distribuția Poisson sunt ilustrați în tabelul 8.

Tabel 8

Coloana	1	2	3	4
PARAM (J,I)	λ - n_0	n_0	Valoarea maximă	-

Pentru a genera un număr cu codul NPSSN distribuit Poisson se testează mai întâi dacă λ - n_0 > 6, caz în care numărul generat se obține dintr-o distribuție normală pe baza proprietății care ne asigură că pentru valori mari ale mediei normalizate, distribuția normală dă o bună aproximare a distribuției Poisson.

În caz contrar se foloseşte *p(n)* şi din teoria probabilității rezultă că dacă evenimentele se produc unul câte unul la un moment dat şi numărul de produceri ale unui eveniment pe unitatea de timp este distribuit Poisson cu media λ, atunci intervalele dintre producerea evenimentelor sunt distribuite exponențial cu media 1/λ.

Rezultă că valorile lui *n* vor fi distribuite Poisson dacă satisfac inegalitatea:

$$\sum_{i=1}^{n} t_i \le 1 < \sum_{i=1}^{n+1} t_i \ ,$$

unde fiecare t_i este distribuit exponențial cu media 1/λ, adică $t_i = -\frac{1}{\lambda} \cdot \ln y_i$, iar y_i este un număr aleator.

Având în vedere aceste valori, inegalitatea se scrie:

$\sum_{i=1}^{n}\left(-\frac{1}{\lambda}\right) \cdot \ln y_i \le 1 < \sum_{i=1}^{n+1}\left(-\frac{1}{\lambda}\right) \ln y_i$, sau multiplicând cu (-λ), inegalitatea devine:

$$\sum_{i=1}^{n} \ln y_i \ge -1 > \sum_{i=1}^{n+1} \ln y_i \ .$$

Ultima relație poate fi înlocuită succesiv cu relațiile echivalente:

$$\exp\left(\sum_{i=1}^{n} \ln y_i\right) \ge \exp(-1) > \exp\left(\sum_{i=1}^{n+1} \ln y_i\right)$$

$$\prod_{i=1}^{n} \exp(\ln y_i) \ge \exp(-1) > \prod_{i=1}^{n+1} \exp(\ln y_i)$$

$$\prod_{i=1}^{n} y_i \ge \exp(-1) > \prod_{i=1}^{n+1} y_i \ .$$

Rezultă că trebuie generate *n + 1* numere aleatoare până când produsul lor îndeplineşte ultima dublă inegalitate şi atunci *n* este numărul generat după distribuția Poisson.

Schema logică pentru generarea de numere aleatoare după distribuția Poisson de parametri dați, este prezentată în figura 21.

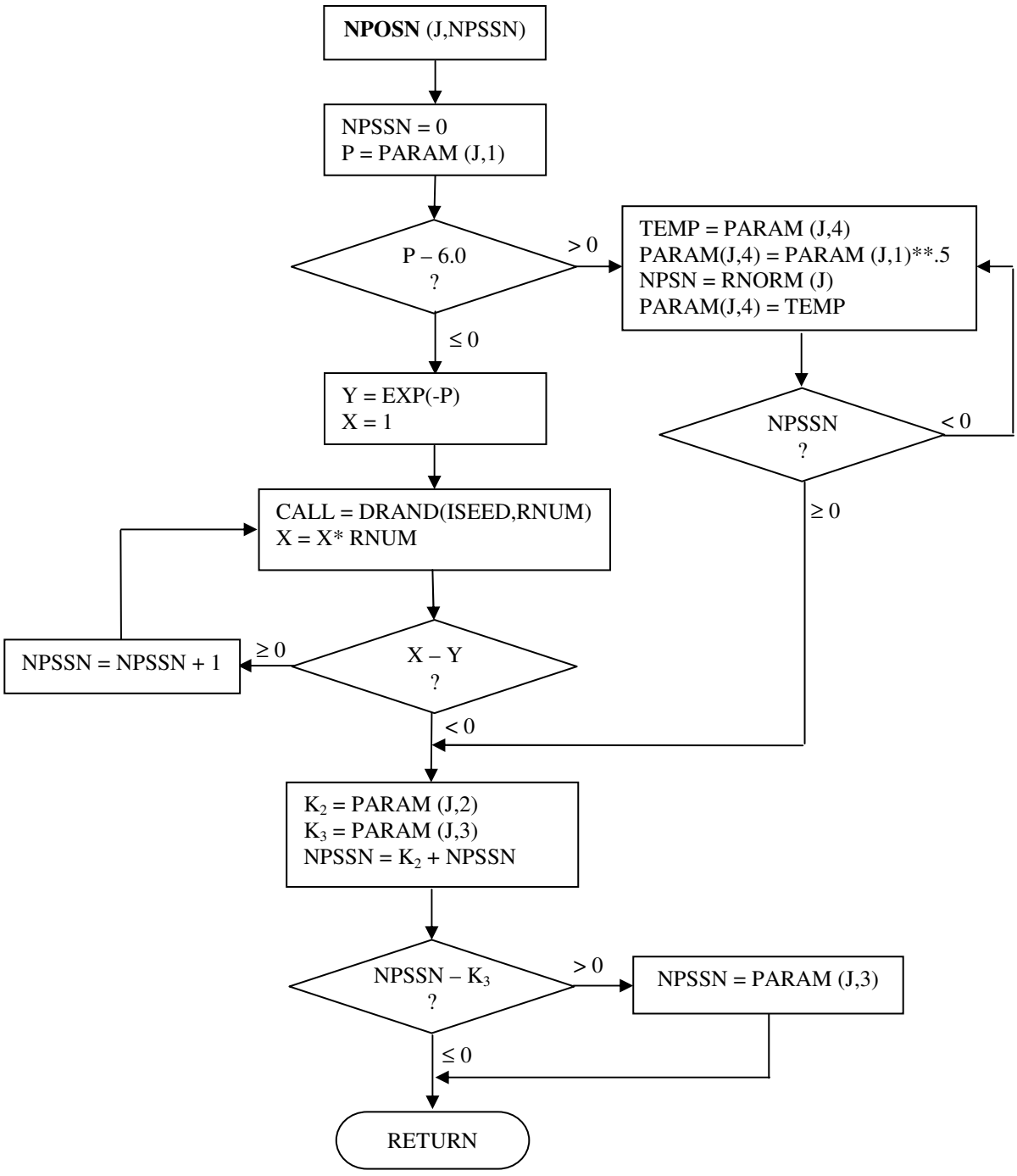


Fig. 21 – *Schema logică a subrutinei NPOSN*

F8. Alte funcţii ajutătoare

În afara funcţiilor care determină minimul sau maximul dintre două variabile, GASP mai conţine şi funcţiile SUMQ şi PRODQ care obţin informaţii cumulative ale unor atribute specificate din fişiere.

F8.1) Funcţia SUMQ (JATT, JQ, NSET)

Această funcţie calculează suma uturor valorilor atribut stocate în JATT, începând cu prima înregistrare şi terminând cu ultima înregistrare a fişierului JQ (figura 22).

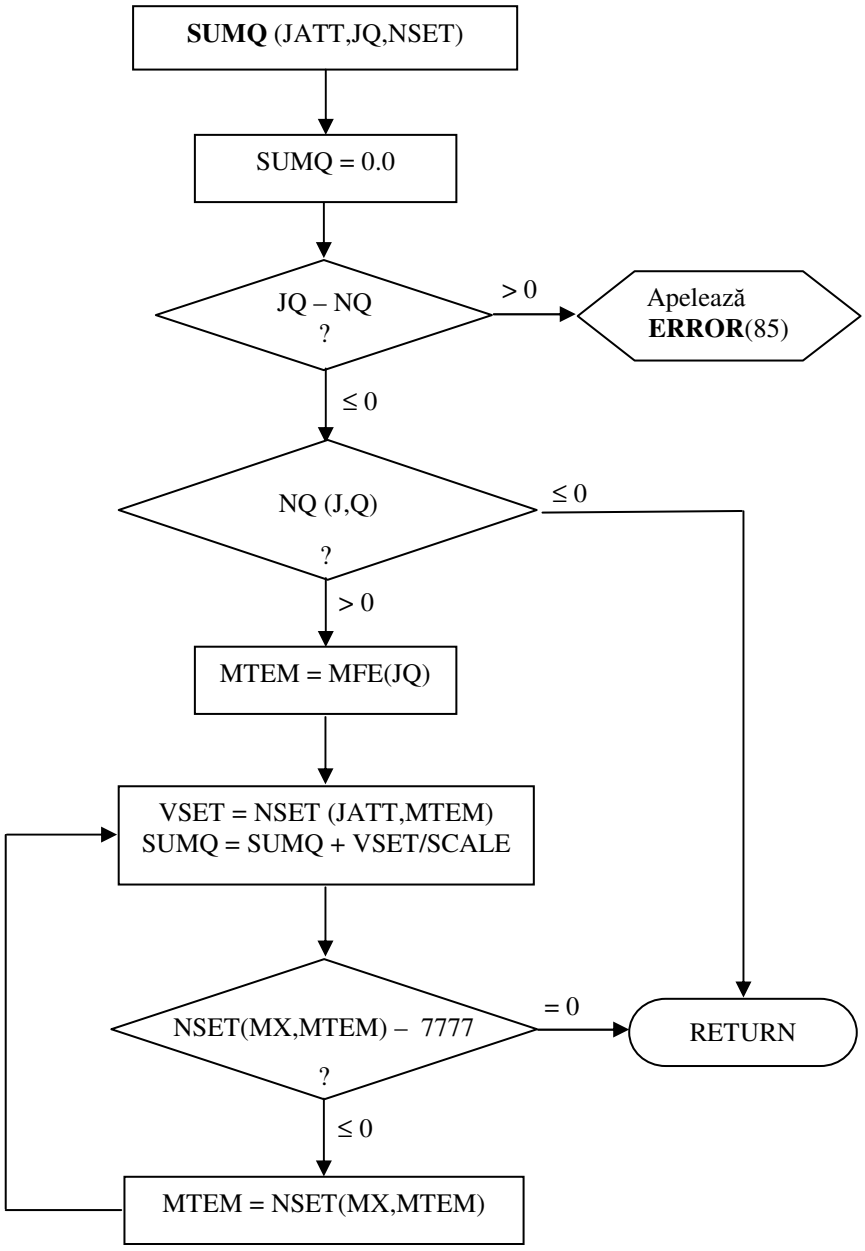


Fig. 22 – *Schema logică a subrutinei SUMQ*

F8.2) Funcția PRODQ (JATT, JQ, NSET)

Această funcție calculează produsul valorilor atributului JATT din fiecare înregistrare a fişierului JQ. Procedura de calcul este similară cu cea descrisă pentru funcția SUMQ, exceptând faptul că valorile sunt multiplicate în loc să fie însumate. Schema logică a subrutinei PRODQ este reprezentată în figura 23.

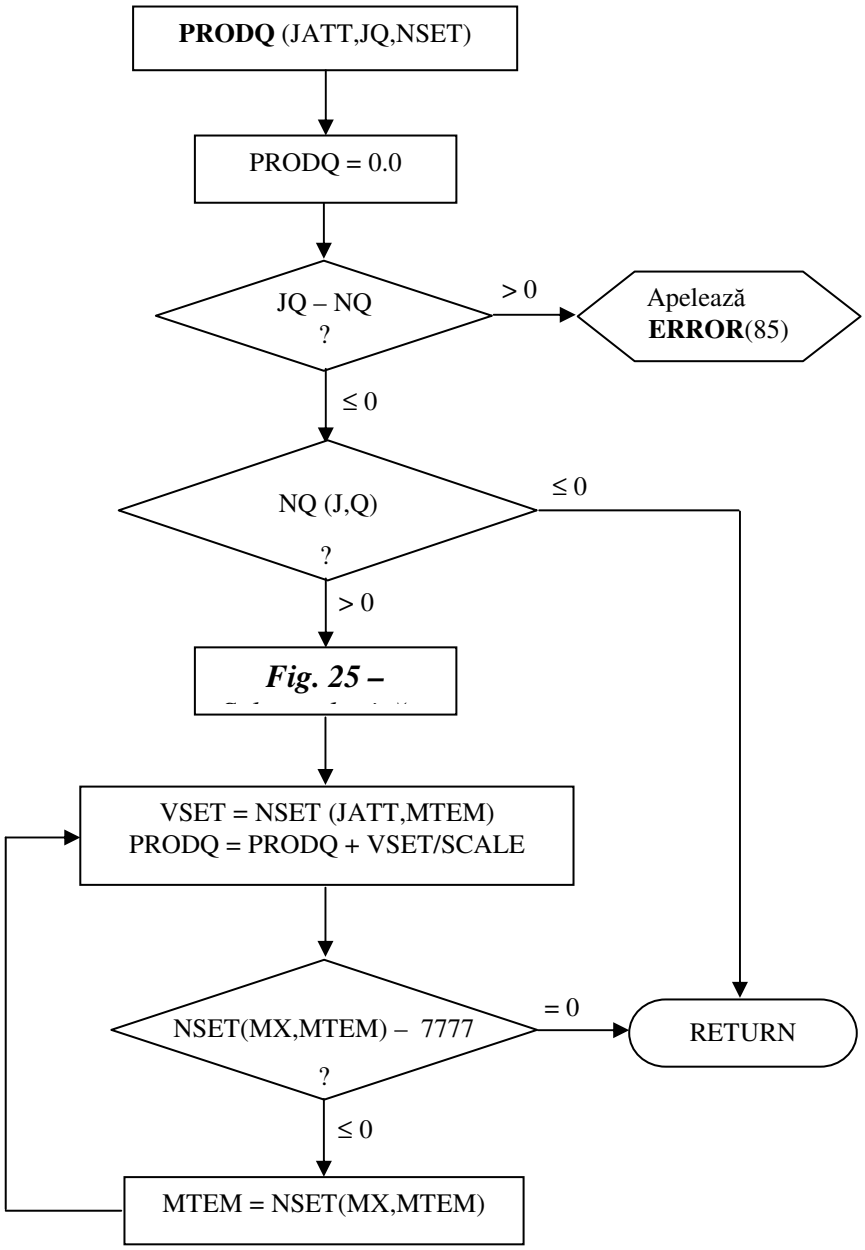


Fig. 23 – *Schema logică a subrutinei PRODQ*

5. O comparaţie între limbajele de simulare GASP, GPSS şi SIMSCRIPT

Un rezumat al celor mai importante concepte rezultate din practica şi studiul acestor limbaje de simulare este necesar pentru a ilustra relaţia dintre ele.

Toate limbajele de simulare furnizează instrucţiuni şi concepte pentru:

- reprezentarea stării sistemului la un anumit moment în timp (modelarea statică);
- mişcarea sistemului de la o stare la alta (modelarea dinamică);
- realizarea unor activităti relevante (generarea datelor, analiza datelor etc.) necesare pentru conducerea experimentelor de simulare.

Modelarea statică şi modelarea dinamică constituie nucleul oricărui limbaj de simulare.

5.1. Concepte de modelare statică

GASP consideră că sistemele sunt formate din entităţi care sunt descrise de atribute şi sunt stocate în fişiere. O stare a sistemului poate fi schimbată numai dacă sunt create sau distruse entităţi, dacă sunt schimbate valorile atributelor, sau dacă se modifică conţinutul fişierelor.

SIMSCRIPT consideră în mod similar sistemele, entităţile şi atributele, însă substituie fişierul cu un mecanism mai general denumit “set”.

Entităţile sunt împărţite în două categorii:

- **temporare,** care sunt în mod fizic create şi distruse prin instrucţiuni;
- **permanente,** care au atributele stocate ca masive.

SIMSCRIPT, ca şi GASP, foloseşte “pointeri” pentru a înlănţui entităţile care sunt membrii din seturi, respectiv care sunt înregistrări conţinute în fişiere.

GPSS lucrează cu **tranzacţii**, **facilităţi** şi **stocări**. Tranzacţiile au parametrii care le descriu şi sunt generate şi încheiate în mod dinamic. Tranzacţiile pot fi grupate în “lanţuri - utilizator” care pot fi manipulate ca şi fişierele şi seturile.

Facilităţile şi stocările reprezintă entităţi permanente şi au proprietăţi de capacitate limitată. Tabelul 9 pune în contrast conceptele modelării statice folosite de cele trei limbaje.

Tabel 9.

Concept	GASP	SIMSCRIPT	GPSS
Obiect static	Entități	Entitate permanentă	Facilitate/stocare
Obiect dinamic	Înregistrare în fişier	Entitate temporară	Tranzacție
Caracteristică	Atribut	Atribut	Parametru
Unitate de legătură	Fişier	Set	Lanț utilizator

Prin urmare, orice sistem modelat în SIMSCRIPT sau GPSS se află într-o relație puternică cu un model GASP, în măsura în care interesează modelul static al stării lui. Din acest punct de vedere, nu este dificil a translata un model GASP într-un model SIMSCRIPT sau GPSS, sau invers.

Există însă alte caracteristici care fac dificilă această translatare iar în cazul simulărilor pentru probleme mari şi complexe, translatarea poate deveni chiar imposibilă.

5.2. Concepte de modelare dinamică

GASP se bazează pe evenimente care se produc la momente în timp la care starea sistemului se schimbă. Evenimentele sunt reprezentate prin programe pe calculator care caracterizează trecutul şi prezentul sistemului şi determină schimbările stărilor lui. De asemenea, programele pot să prevadă stările viitoare şi pot să planifice schimbările acestor stări în cadrul altor evenimente.

SIMSCRIPT are concepte similare şi conține un program de evenimente şi subrutine eveniment. În plus față de GASP, are un mecanism pentru planificarea automată a unor evenimente speciale exogene.

GPSS se bazează pe un punct de vedere diferit asupra dinamicii sistemului. Se construieşte o diagramă - bloc alcătuită din 45 de tipuri diferite de blocuri, fiecare din ele realizând o funcție orientată pe o simulare specială. Pregătirea unei diagrame - bloc se face

vizualizând modul în care o anumită tranzacție "curge" prin sistem. Tranzacțiile comută de la bloc la bloc aşa cum ele se mişcă prin sistemul real. Sistemul GPSS generează un calendar de evenimente după care are loc producerea lor.

Anumite blocuri permit reprezentarea avansului sau întărzierii în timp, iar altele reprezintă operații logice executate fără decalaje în timp. Toată simularea GPSS are loc în contextul celor 45 de blocuri, aşa cum, în mod similar, simularea GASP are loc în contextul FORTRAN-ului şi a subrutinelor şi funcțiilor standard. Tabelul 10 ilustrează aceste diferențieri în conceptele de modelare dinamică.

Tabel 10

Concept	GASP	SIMSCRIPT	GPSS
Model	Program compus din subrutine eveniment	Program compus din subrutine eveniment	Diagramă bloc
Avans în timp	Programarea evenimentului	Programarea evenimentului	Tranzacția diagramei flux

Rezultă că translațiile din GASP în SIMSCRIPT şi GPSS sunt directe, în timp ce translațiile din GASP sau SIMSCRIPT în GPSS sunt mai dificile. Deşi modelele statice au o structură asemănătoare, modelele lor dinamice sunt complet diferite.

6. Aplicații ale simulării cu limbajul GASP

6.1. Simularea stocurilor cu limbajul GASP

Situațiile de stocare apar frecvent în procesele industriale, comerciale, de aprovizionare etc. şi continuă să rețină atenția specialiştilor din domeniul cercetării operaționale. Interesul lor este justificat de economiile obținute prin îmbunătățirea deciziilor luate în probleme de stocare.

Sistemele de stocare implică înmagazinarea de articole, cereri ale beneficiarilor care solicită cantități diferite din unul sau mai multe articole, şi o politică de stoc care să specifice **când** şi **ce** cantitate trebuie comandată pentru reaprovizionare.

Teoria stocurilor încearcă să răspundă la aceste întrebări într-o manieră optimă. Soluțiile depind de veniturile şi de costurile asociate sistemului de stocare. Venitul obținut este funcție de cantitatea vândută şi de prețul unitar de vânzare. Costurile sunt mult mai complexe şi includ: costuri de stocare, costuri de lansare a comenzii, costuri de penalizare, pierderi datorate lipsei de articole în stoc (cereri nesatisfăcute) etc. Obiectivul il constituie determinarea celei mai bune po!itici de stoc având în vedere maximizarea profitului mediu, minimizarea cheltuielilor totale, minimizarea probabilității ruperii stocului etc.

În continuare este prezentat un exemplu simplificat pentru a se ilustra modul de utilizare a limbajului GASP în simularea problemelor de stocare.

Definirea problemei

O unitate comercială de desfacere a unor produse către populație doreşte să analizeze politica stocului (Q,R) de comandă a Q unități de fiecare dată când nivelul stocului curent scade sub R, pentru valorile (36,16), (36,9) şi (18,9).

Din analiza unității comerciale se cunosc următoarele date:

- cererea este distribuită Poisson cu o medie de 5 buc./săptămână;
- produsul se cumpără cu 40 u.b. şi se vinde cu 65 u.b.;
- cheltuielile de stocare sunt de 0,20 u.b. pentru fiecare u.b./an;
- cheltuielile de penalizare sunt de 20 u.b. pentru fiecare cerere nerezolvată;
- costul de lansare a unei comenzi este de 3 u.b./comandă;
- durata de la lansarea până la primirea unei comenzi este de 3 săptămâni.

Se face ipoteza că stocul inițial este de 31 bucăți şi că nu sunt comenzi restante.

Obiectivul simulării este de a studia efectul diferitelor valori ale politicii de stoc pentru unitatea comercială care suportă penalizări în cazul nesatisfacerii integrale a unei cereri. Simularea politicii de stoc se face pentru o perioadă de 6 ani şi se obțin următoarele statistici:

- profitul mediu pe an;
- stocul mediu de siguranță;
- numărul mediu de comenzi;
- numărul mediu de pierderi la vânzare;
- vânzările totale.

Procedura

Funcționarea unității comerciale este simulată utilizând limbajul GASP pe baza a trei evenimente:

- cererea unui client pentru un produs, DMAND;
- recepția unei comenzi de la angrosist, RECPT;
- sfârşitul rulării de simulare, ENDSM.

Entitățile folosite pentru această simulare sunt:

- disponibilul existent în stoc (stocul net/fizic);
- situația stocului, dată de inventarul din registre, calculată ca suma dintre stocul fizic şi volumul comenzilor din care se scade volumul comenzilor returnate.

În acest exemplu nu sunt comenzi returnate, relansarea comenzilor se face pe baza situației stocului şi se foloseşte numai fişierul de evenimente având ca atribut 1 momentul evenimentului, iar ca atribut 2 codul evenimentului (1 = cerere, 2 = recepție, 3 = sfârşit de simulare). Variabilele non - GASP folosite în această simulare sunt date în tabelul 11.

Pe perioada de simulare **veniturile totale** se calculează ca produs între numărul total de vânzări şi diferența dintre prețul de vânzare şi prețul de achiziție, iar cheltuielile totale se determină ca sumă între:

- cheltuielile totale de lansare, calculate ca produs între costul de lansare a unei comenzi şi numărul total de comenzi;
- cheltuielile totale de stocare, calculate ca produs între costul unitar de stocare şi stocul cumulat în timp pe perioada simulată;
- cheltuielile totale de achiziție, calculate ca produs între prețul unitar de achiziție şi stocul cumulat în timp pe perioada simulată;
- cheltuielile totale de penalizare, calculate ca produs între costul unitar de penalizare (pierdere datorată nevinderii unei unități de produs) şi numărul total de pierderi la vânzare.

Cu aceste elemente/definitii, profitul mediu pe săptămână se calculează ca diferență între veniturile totale şi cheltuielile totale raportat la numărul de săptămâni din perioada de simulare. Utilizând notațiile din tabelul 11, profitul mediu pe săptămână se poate calcula pentru o perioadă de simulare de T săptămâni, cu relația:

$$P = [(SPOU-PCOU)*SALE-(CPO* TORDS+CCHG*PCOU* TIIN+ULOSE*SLOST)]/T$$

Tabel 11

Cod variabilă	Semnificaţie	Valoare iniţială
AVIN	Disponibilul de stoc mediu (buc./ săptămână)	Output
AVSS	Stocul mediu de siguranţă	Output
P	Profitul mediu pe săptămână	Output
CCHG	Cheltuieli de întreţinere a stocului ($ / $ - săpt.)	0,003846
CPO	Cost de lansare pe comandă ($ / comandă)	3
PCOU	Preţ unitar de achiziţie ($ / buc.)	40
POS	Situaţia stocului	31
STOCK	Disponibilul în stoc	31
Q	Cantitatea comandată	Se citeşte
R	Nivelul de recomandă	Se citeşte
SALE	Număr total de vânzări	0
SLOST	Număr de pierderi la vânzare	0
SPOU	Preţ unitar de vânzare ($ / buc.)	65
TIIN	Stocul cumulat în timp pe perioada simulată	-
TLEAD	Timpul dintre lansarea şi primirea comenzii (săptămâni)	3
TORDS	Număr total de comenzi	0
ULOSE	Pierderea datorată nevânzării ($ / buc.)	20
XL	Durata medie între cereri (săptămâni)	0.2

Simularea începe prin citirea de către programul principal a valorilor variabilelor asociate costurilor şi politicii de stoc. Dacă Q este pozitiv se inițializează variabilele asociate cu vânzările totale, numărul total de comenzi, numărul de pierderi la vânzare, disponibilul în stoc şi situația stocului. Apoi este apelată subrutina GASP care va conduce simularea până la sfârşit. Când Q este negativ, simularea este terminată (condiție de oprire a simulării).

Simularea stocului poate fi realizată pentru oricâte seturi de valori ale parametrilor politicii de stoc (Q,R) se doresc. Pentru fiecare rulare tipurile de date necesare sunt citite cu subrutina DATAN.

Subrutina EVNTS transferă temporar controlul la una din cele trei subrutine eveniment scrise de programator: DMAND, RECPT, sau ENDSM.

Schema logică pentru subrutina DMAND, care exprimă cerea unui client pentru un produs, este prezentată în figura 24.

Când o cerere se întâmplă, următoarea cerere este programată să se realizeze la TNOW plus un timp extras dintr-o distribuție exponențială cu media XL. Se determină apoi caracterul cererii curente. Dacă nu există nici un produs în stoc care să satisfacă cererea, numărul de pierderi la vânzare este mărit cu 1 şi se revine la subrutina care a chemat (cerere nesatisfăcută).

Dacă există un produs disponibil, vânzările sunt crescute cu 1, se calculează statistici după numărul de unități în stoc, iar stocul şi inventarul sunt micşorate cu 1. Apoi se compară inventarul cu nivelul de recomandă şi dacă $\boldsymbol{PCS \leq R}$, se face o comandă pentru Q unități.

Se programează un eveniment RECPT la TNOW plus timpul dintre lansarea şi primirea comenzii şi stocul este mărit cu Q unități, după care se revine la subrutina care a chemat. Înainte de a creşte disponibilul cu cantitatea comandată Q, sunt colectate statistici după:

- disponibilul de stoc cumulat pe perioada de timp de la ultima schimbare a disponibilului din inventar, utilizând subrutina TMST;
- disponibilul de stoc la momentul când o comandă este primită, denumit nivelul stocului de siguranță, utilizând subrutina COLCT.

Evenimentul sfârşit de simulare ENDSM, determină apelarea subrutinelor PRNTQ şi SUMRY pentru actualizarea statisticilor referitoare la stoc şi pentru calcularea valorilor medii care apar în rapoartele finale, apoi poziționează variabilele de control pentru terminarea simulării.

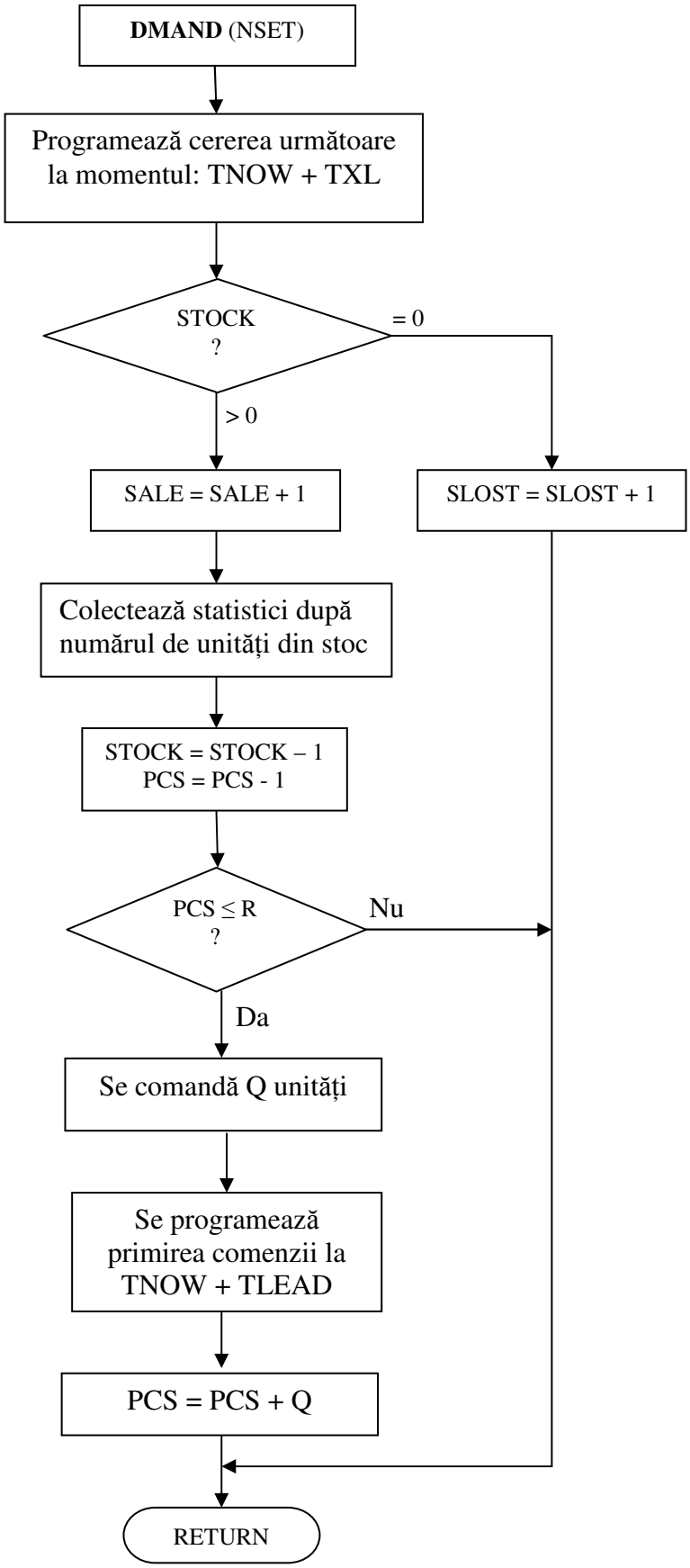


Fig. 24 – *Schema logică a subrutinei DMAND*

Rapoarte de simulare

În tabelul 12 sunt prezentate rezultatele finale, **exprimate ca valori medii pe săptămână,** obținute în trei rulări de simulare pentru cele trei seturi de valori ale politicii de stoc (Q,R), pe o perioadă de 6 ani.

Tabel 12

Nr. rulare	Q	R	Valori medii pe săptămână					
			Vânzări	Pierderi	Nr. comenzi	Stoc	Stoc siguranță	Profit
1	36	18	5.07	0.13	0.14	21.09	3.16	120.40
2	36	9	4.45	0.76	0.12	15.76	0.00	933.00
3	18	9	3.79	1.41	0.21	7.13	0.05	648.00

O analiză a rezultatelor prezentate în tabelul 12 ilustrează faptul că nivelul de recomandă R, trebuie să fie relativ mare pentru a obține un profit mediu pe săptămână mai mare. Deoarece costul unei pierderi la vânzare (20 $ / buc.) este mult mai mare decât costul de întreținere în stoc (0,039*40 ≅ 0.16 $ / buc / săpt.), nivelul de recomandă trebuie să fie cel puțin egal cu cererea medie în timpul perioadei de reaprovizionare (λ•TLEAD ≅ 5 buc/săpt.•3 săpt ≅ 15 buc). Rezultă astfel condiția: ***R ≥ l5 buc*** şi deci politica (36,18) este cea mai bună dintre cele trei considerate. Pe de altă parte, deoarece costul de lansare a unei comenzi este mic, cantitatea comandată Q nu trebuie să fie mare.

Analiza rezultatelor obținute prin experimentul de simulare conduce la concluzia că o îmbunătățire a profitului mediu necesită creşterea nivelului de recomandă R şi descreşterea cantității comandate Q.

6.2. Simularea proiectului unui sistem de informare

Se doreşte analiza proiectului unui sistem de servire cu informații ce s-ar putea instala în aeroporturi, gări sau hoteluri, care să furnizeze informații despre vreme, sosirile şi plecările avioanelor şi trenurilor, rezervarea de camere etc., prin simulare cu limbajul GASP, în vederea luării unei decizii privind fabricarea noului produs.

Din punct de vedere al utilizatorului, sistemul constă din staţii care-i permit să formuleze cereri de informare, utilizând un cod fixat la fiecare staţie.

În exemplul considerat, cele 6 staţii sunt depărtate una de alta, astfel că fiecare utilizator poate folosi o singură staţie. Dacă staţia este ocupată, utilizatorul intră în şir şi aşteaptă rândul său pentru a lansa cererea. După lansarea cererii utilizatorul aşteaptă afişarea răspunsului pe care-l citeşte cu viteza lui proprie. Fiecare staţie este asociată cu o diviziune pe un analizor (scanner) care se roteşte şi identifică staţia de la care a fost lansată cererea. Analizorul transferă cererea într-un buffer, capabil să păstreze trei mesaje, care este conectat la un calculator ce poate să prelucreze simultan cele trei mesaje. Dacă buffer-ul este plin, analizorul se opreşte şi aşteaptă până când acesta devine disponibil.

Transferul răspunsului are loc direct între buffer şi staţie şi nu necesită utilizarea analizorului. Parametrii estimaţi în vederea simulării sistemului sunt:

- utilizatorii sosesc la sistem cu o rată medie de 5 utilizatori pe minut, după o distribuţie exponenţială a timpului dintre sosiri;
- cele şase staţii au aceeaşi densitate de trafic şi deci, toate staţiile au o probabiliate egală de a fi utilizate;
- durata lansării unei cereri şi durata citirii răspunsului sunt uniform distribuite în intervalul (0.3, 0.5) respectiv, în intervalul (0.6, 0.8) minute;
- durata rotirii analizorului între două puncte este egală cu durata de analiză şi reprezintă 0.0027 minute;
- transferul informaţiei de la unitatea de analiză la buffer necesită 0.0117 minute, iar de la buffer la staţie 0.0397 minute;
- timpul necesar pentru a găsi răspunsul la o cerere este uniform distribuit în intervalul (0.05, 0.10) minute.

Această problemă este specifică tipului de studiu care trebuie făcut când se proiectează un nou echipament. În acest caz, echipamentul este hard-ul necesar pentru a satisface nevoile sistemului de informare. Studiul intenţionează să descopere dacă echipamentul cu performanţele arătate de parametrii din enunţul problemei satisface sau nu cererile estimate ale clienţilor.

Două statistici sunt de interes major şi reprezintă obiectivul simulării prin analiza lor şi anume:

- distribuţia timpului de aşteptare al unui utilizator între momentul în care termină formularea cererii şi momentul în care este afişat răspunsul său;

- dimensiunea medie probabilă a şirului în faţa fiecărei staţii.

Un program principal şi mai multe subrutine sunt folosite pentru simularea proiectului sistemului de informare.

Programul principal stabileşte fondul comun de date, citeşte variabilele non-GASP (tabel 13) şi cheamă subrutina principală GASP, care conduce simularea pănă la sfârşit, cedând temporar controlul celorlalte subrutine aşa cum o cere logica programului.

Tabel 13

Cod variabile non-GASP	Semnificaţie variabile
NARC	Nr. clientului care soseşte în sistem
JBUFF	Dacă JBUFF = 1, buffer-ul este plin şi analizorul se opreşte; Dacă JBUFF = 0 se continuă
NSCAN	Nr.staţiei ce urmează să fie analizată
NSTA(I)	Nr.de utilizatori care aşteaptă la staţia I∈ {1,2,…,6}
JRPLY(I)	Indică sub formă codificată starea liniei I şi anume: 1 - linie deschisă şi se lansează o cerere; 2 - cererea a fost lansată; 3 - cererea a fost transferată în buffer şi se procesează; 4 - răspunsul a fost trimis şi urmează să fie citit.

Unele subrutine reprezintă chiar tipurile de evenimente implicate în modelul de simulare al sistemului de informare, şi anume:

- ARRVL - sosirea unui utilizator la o staţie;
- RQEST - lansarea unei cereri pentru obţinerea de informaţii;
- SCAN - sosirea analizorului la un punct terminal;
- ANSER - sosirea unui răspuns la o staţie;
- ENDSV - sfârşitul servirii unui utilizator.

Deoarece simularea se termină când timpul curent TNOW, depăşeşte timpul fixat pentru simulare TFIN, nu este necesar un eveniment sfârşit de simulare.

Fişierele şi atributele asociate pentru această simulare sunt prezentate în tabelul 14.

Tabel 14

Atribute	Fişier 1	Fişier 2	Fişier 3
	Evenimente	Cereri formulate	Cereri în buffer
ATRIB(1)	Momentul evenimentului	Momentul lansării cererii	Momentul citirii răspunsului
JTRIB(1)	Codul evenimentului	20	30
JTRIB(2)	Nr. staţiei	Nr. staţiei	Nr. staţiei

Pentru fişierele 2 şi 3 valorile stocate în JTRIB(1) sunt folosite pentru a determina când trebuie tipărit conţinutul lor.

Procedura de simulare

Subrutina ARRVL este apelată de fiecare dată când un utilizator soseşte în sistem. Un număr aleator de la 1 la 6 este generat dintr-o distribuţie uniformă pentru a determina la ce staţie soseşte un utilizator. Subrutina verifică apoi dacă staţia este liberă şi în acest caz programează momentul terminării timpului de transmitere a cererii. Dacă staţia este ocupată, utilizatorul intră în firul de aşteptare al staţiei şi NSTA(I) se măreşte cu o unitate. Apoi se calculează următorul moment de sosire şi se programează evenimentul sosire corespunzator.

Subrutina RQEST este apelată imediat ce utilizatorul şi-a terminat cererea pentru informare. Cererea este plasată în fişierul 2 şi indicatorul JRPL(I) este setat la 2 pentru a arăta că o nouă cerere a fost lansată.

Subrutina SCAN controlează analizorul, chemându-l de fiecare dată pentru a semnaliza lansarea unei noi cereri de la o staţie asociată unui punct de pe analizor. Dacă se identifică o nouă cerere sosită la un punct de pe analizor şi buffer-ul nu este plin, subrutina programează momentul citirii răspunsului şi plasează cererea în fişierul 3.

Dacă nici o cerere nu este sesizată sau dacă cererea se poate transfera în buffer, subrutina programează momentul următorului punct ce va fi analizat.

Aceste acţiuni pot fi întrerupte de un buffer plin, caz în care subrutina opreşte analizorul prin neprogramarea unui alt eveniment SCAN şi returnează controlul la subrutina conducătoare GASP, fără nici o altă acţiune.

Subrutina ANSER este chemată ori de câte ori se citeşte răspunsul la o cerere. Se calculează timpul necesar pentru citirea răspunsului şi se programează un eveniment sfârşit de servire. Subrutina şterge apoi mesajul din buffer şi dacă analizorul a fost oprit datorită unui buffer plin, se apelează subrutina SCAN. Dacă buffer-ul nu a fost blocat, controlul este returnat la GASP.

Statisticile referitoare la timpul scurs din momentul lansării cererii sunt colectate în subrutinele TMST şi HISTO.

Subrutina ENDSV este chemată ori de câte ori un utilizator a primit răspuns la cererea sa. Subrutina colectează statistici şi începe servirea următorului utilizator din şir.

Simularea se opreşte când timpul curent depăşeşte timpul fixat pentru simulare, adică TNOW ≥ TFIN.

Rapoarte de simulare

Din rezultatele finale prezentate în raportul sumar GASP (tabel 15), se observă că dimensiunea medie probabilă a şirului de aşteptare, NSTA(I), ia valori cuprinse între 1.5769 la staţia 2 şi 10.0429 la staţia 6. Se observă de asemenea că timpul mediu pentru a obţine un răspuns este de 0.1444 minute (variabila T_1). În felul acesta chiar dacă clienţii sosesc după media de unul la fiecare 0.2 minute, sistemul proiectat nu va furniza serviciul corespunzător pentru a face faţă cererii anticipate de informaţii ale utilizatorilor.

Tabel 15

Nume variabilă	Cod	Medie	Abatere standard	Minim	Maxim	Timp total
NSTA(1)	1	4.1817	1.4547	0.0000	7.0000	38.8895
NSTA(2)	2	1.5769	1.1885	0.0000	5.0000	39.9977
NSTA(3)	3	1.6388	1.2277	0.0000	4.0000	39.2231
NSTA(4)	4	3.5326	1.9493	0.0000	7.0000	39.1148
NSTA(5)	5	3.7533	2.4236	0.0000	12.0000	39.7055
NSTA(6)	6	10.0429	3.8217	0.0000	15.0000	39.9057
T_1	1	0.1444	0.0170	0.1094	0.1875	175(Obsevaţii)

În consecinţă, proiectul noului produs va fi respins, urmând să fie realizat un nou proiect al sistemului cu performanţe îmbunătăţite, care să satisfacă cerinţele estimate de informare ale utilizatorilor.

6.3. Simularea proiectului unui lift

Definirea problemei

Pentru o clădire cu 6 etaje a fost proiectat un lift cu o capacitate de 7 persoane. Constructorul liftului a estimat că timpul de trecere al liftului între etaje este de 0.10 minute, timpul de oprire al liftului la un etaj este de 0.05 minute, timpul consumat de fiecare pasager pentru a intra în lift este de 0.005 minute iar pentru a ieşi din lift este de 0.20 minute.

Beneficiarul clădirii a făcut o analiză a traficului de pasageri şi a elaborat un studiu din care rezultă informaţii asupra frecvenței relative cu care pasagerii solicită liftul pentru fiecare etaj şi frecvența relativă cu care pasagerii de la un etaj doresc să meargă la un alt etaj. Aceste valori sunt prezentate în tabelul 16.

Tabel 16

Etaj inițial	Probabiliatea inițială a etajului	Probabilitatea ca un pasager să meargă de la etajul I la etajul J. [P(I,J)]					
		Etaj destinație J					
I	F(I)	1	2	3	4	5	6
1	0.279	0.0	0.2	0.2	0.2	0.2	0.2
2	0.135	0.4	0.0	0.1	0.1	0.1	0.3
3	0.130	0.4	0.1	0.0	0.1	0.1	0.3
4	0.130	0.4	0.1	0.1	0.0	0.1	0.3
5	0.116	0.5	0.1	0.05	0.05	0.0	0.3
6	0.210	0.3	0.2	0.2	0.2	0.1	0.0

Studiul traficului arată de asemenea că pasagerii sosesc în grupuri de maxim 8 pasageri/grup şi că numărul de indivizi într-un grup este distribuit Poisson, cu o medie de 3 pasageri pe grup. Sosirile grupurilor sunt distribuite exponențial cu un timp mediu de 2 minute. Proiectantul doreşte să ştie dacă sistemul lift propus este adecvat.

Qbiectivele simulării

Obiectivul acestei simulări este de a studia sistemul lift propus pentru a se obține măsura performanțelor sale în vederea evaluării lui. Evaluarea se face prin examinarea folosirii liftului pe baza numărului mediu de pasageri în lift, timpul pe care pasagerii îl petrec în lift şi mărimea şirului de aşteptare la fiecare etaj.

Procedura de simulare

Pentru a simula sistemul lift sunt necesare două evenimente şi anume:

- sosirea liftului la un etaj;
- sosirea unui grup de pasageri care solicită liftul la un etaj.

Când liftul soseşte la un etaj pasagerii care au ajuns la destinație coboară. Pasagerii care aşteaptă pentru servire în direcția în care liftul pleacă intră în lift numai dacă este suficient spațiu disponibil. Evenimentul sosire pasageri programează următorul eveniment sosire pasageri şi determină numărul de pasageri din grupul care soseşte. Valorile etajului de origine şi de destinație sunt obținute prin eşantionare. Informația referitoare la pasageri este menținută într-un fişier asociat cu un număr de etaj. Liftul se mişcă spre etajul următor prin programarea unui eveniment sosire lift la timpul curent, plus timpul cheltuit la etaj, plus timpul pentru trecere la etajul următor. Principalele variabile non-GASP asociate cu această problemă sunt ilustrate în tabelul 17.

Tabel 17

Cod variabile	Definiții variabile
FLOOR(I)	Probabilitatea ca un grup de pasageri să provină de la etajul I
P(I,J)	Probabilitatea ca un pasager care soseşte la etajul I să meargă la etajul J
NFOE	Numărul etajului la care se găseşte liftul
NCAP	Numărul maxim de pasageri care pot intra în lift
TPGI	Timpul per pasager de intrare în lift
TPGO	Timpul per pasager de ieşire din lift
TTT	Timpul de trecere a liftului între etaje
NDIR	Direcția de mişcare a liftului (1-în sus; 2-în jos)
NTE	Numărul de pasageri din lift

O considerație de proiectare implică plasarea fiecărui pasager ca o înregistrare într-un anumit fişier pentru a putea obține diverse statistici, şi anume:

- un fişier pentru evenimente;
- un fişier pentru pasagerii din lift;
- un fişier pentru pasagerii care aştepată liftul la fiecare etaj.

În acest caz, se asociază fiecărui pasager momentul lui de sosire, etajul la care aşteaptă şi etajul de destinație. Când un eveniment sosire lift se întâmplă, este cercetat tot fişierul pentru a găsi acei pasageri care doresc să meargă în direcția liftului.

Căutarea va consuma timp, deoarece toți pasagerii care aşteaptă vor fi considerați pasageri potențiali.

Pentru a reduce timpul de căutare, fişierul cu pasagerii care aşteaptă poate fi separat în pasageri care aşteaptă să urce şi pasageri care aşteaptă să coboare. În acest fel vor fi necesare 4 fişiere, exact limita impusă de subprogramele GASP precompilate.

O altă cale de a reduce durata simulării este de a avea fişiere pentru pasagerii care aşteaptă la fiecare etaj. Un pasager de la etajul la care liftul a sosit va intra în lift dacă este spațiu şi dacă liftul merge în direcția dorită. În acest caz sunt necesare 13 fişiere: un fişier pentru evenimente, 6 fişiere pentru pasagerii din lift (cele 6 destinații posibile) şi 6 fişiere pentru pasagerii care aşteaptă la fiecare etaj.

A patra structură de fişier posibilă stochează pasagerii după etajul de origine şi după direcția de mişcare. Aceasta reduce la minim timpul de căutare pentru găsirea pasagerilor candidați pentru intrarea în lift însă creşte la 17 numărul de fişiere: un fişier pentru evenimente, 6 pentru pasagerii din lift (destinația pasagerilor) şi 10 fişiere pentru păstrarea informației despre pasagerii care aşteaptă pentru servire, organizate după etajul de origine al pasagerilor şi după direcția de mişcare dorită. În toate cazurile numărul total de pasageri şi deci numărul de înregistrări în fişiere, rămâne acelaşi.

În programul principal sunt inițializate variabilele non-GASP: liftul este liber primul etajul şi direcția de mers este în sus. Evenimentul "sosirea pasagerului următor" este programat cu subrutina FILEM. Momentul sosirii pasagerului următor se calculează prin adăugarea unui eşantion dintr-o distribuție exponențială la momentul curent de simulare, TNOW. Numărul pasagerilor care sosesc la momentul TNOW se obține apelând subrutina NPOSN. Etajul la care se află pasagerii se obține utilizând o funcție de probabilitate discretă memorată într-o zonă specială, FLOOR. Etajul de destinație al fiecărui pasager se determină folosind o matrice de probabilitate de trecere P în care fiecare linie este o funcție de probabilitate.

Dacă etajul este fie 1, fie 6, direcția de mişcare a liftului se schimbă. La fiecare etaj se fac teste pentru a determina dacă există pasageri care părăsesc liftul şi/sau intră în lift făcându-se statisticile corespunzătoare. Apoi se calculează numărul de locuri disponibile din lift şi se completează cu pasagerii care doresc să meargă în direcția liftului.

După ce la un etaj toți pasagerii care ies şi intră au fost procesați, se calculează timpul cheltuit la etaj ca timpul pentru oprire la etaj, plus timpul pentru ca pasagerii să iasă, plus timpul ca pasagerii să intre în lift. Numărul etajului următor se determină prin adăugarea la numărul etajului curent a unui increment (±1), funcție de direcția de mişcare a liftului. Este programat apoi evenimentul "sosire lift".

Rapoarte de simulare

Rezultatele finale ale simulării sistemului lift folosind 17 fişiere sunt prezentate în tabelul 18. Când s-au utilizat 13 fişiere rezultatele au fost identice exceptând faptul că statisticile au fost combinate pentru pasagerii care aşteaptă la fiecare etaj, deoarece nu au fost create fişiere distincte pentru pasagerii care urcă, respectiv, coboară. De exemplu în varianta cu 13 fişiere, numărul mediu de pasageri care aşteaptă la etajul 2 este de 0.518, iar numărul maxim este 16 în timp ce în varianta cu 17 fişiere numărul mediu de pasageri care urcă este 0.276, care coboară 0.242, numărul maxim care urcă 10, iar care coboară 6.

Din rezultatele obținute, se observă că etajele 2 şi 6 reprezintă locuri înguste potențiale şi că mai departe trebuie facută o analiză a timpului cheltuit de pasagerii care provin de la aceste etaje.

Tabel 18

Statistici	Media	Deviația standard	Maxim
Timpul per pasager, în lift	2.078	2.270	12.48
Nr. pasageri în lift	2.208	2.332	7
Nr. de pasageri având ca destinație:			
etajul 1	0.603	1.251	7
etajul 2	0.327	0.683	3
etajul 3	0.292	0.683	4
etajul 4	0.309	0.714	4
etajul 5	0.714	0.553	3
etajul 6	0.460	0.951	7
Nr. de pasageri care aşteaptă pentru coborârea liftului la:			
etajul 1	0.802	1.905	8
etajul 2	0.276	0.954	10
etajul 3	0.267	0.847	4
etajul 4	0.164	0.531	3
etajul 5	0.069	0.360	4
Nr. de pasageri care aşteaptă pentru urcarea liftului la:			
etajul 2	0.242	0.031	6
etajul 3	0.192	0.741	5
etajul 4	0.499	1.331	5
etajul 5	0.150	0.568	5
etajul 6	1.376	2.924	19

6.4. Simularea unui atelier de reparații cu stații de lucru în serie

Situațiile de aşteptare apar frecvent în activitățile industriale, de comerț, personale şi au ca elemente de bază:

- un proces de sosire;
- un mecanism de servire;
- reguli de funcționare pentru noii veniți şi stațiile de lucru.

Când un nou sosit nu poate fi servit imediat, se înscrie în şir şi aşteaptă să fie servit. Situațiile de aşteptare implică nesiguranță datorită naturii stochastice a momentelor dintre sosiri şi durata sosirii unui client.

Acest exemplu prezintă simularea proiectului unui atelier de întreținere care execută două operații în serie (în cascadă). Problema ar fi tratată similar în cazul simulării proiectului unei linii de producție formată din două stații de lucru în serie.

Analiza datelor culese pentru acest tip de proiect, a rezultat că intervalul de timp dintre cererile pentru reparații este exponențial distribuit cu o medie de 0.4 u.t. Timpii de servire ai unei unități sunt de asemenea exponențial distribuiți, prima stație necesită în medie 0.25 u.t. iar cea de a doua stație 0.5. u.t. Unitățile sunt transportate automat de la stația de lucru 1 la stația de lucru 2 în 0.2 sau 0.1 u.t., dacă există zero şi respectiv o unitate în firul de aşteptare al celei de a doua stații.

Deoarece unitățile care trebuie reparate sunt destul de voluminoase iar spațiul disponibil este limitat, proiectul propus permite ca 4 unități să aştepte în fața primei stații de servire şi 2 unități în fața celei de a doua stații.

Dacă firul de aşteptare de la stația de lucru 2 este plin, adică dacă sunt două unități care aşteaptă pentru servire, prima stație de lucru este blocată şi nici o unitate nu poate părăsi acea stație. O stație de lucru blocată nu poate servi alte unități.

Politica companiei este de a subcontracta unitățile care nu pot să intre în sistem când firul de aşteptare de la stația 1 este plin (cele patru locuri de aşteptare sunt ocupate).

Condițiile inițiale pentru simulare sunt:

- ambele stații sunt ocupate cu sfârşitul servirii programate la momentul 1;
- prima unitate este programată să sosească la atelierul de întreținere la momentul de începere t = 0,1;
- în firul de aşteptare al stației de lucru 1 sunt trei unități, iar în firul de aşteptare al stației de lucru 2 nu se află nici-o unitate (zero unități.

Simularea este realizată pentru un orizont de simulare de 300 u.t.

Schema proiectului de sistem propus al unui atelier de întreținere-reparații cu două stații de servire, care este simlat, este reprezentat în figura 25.

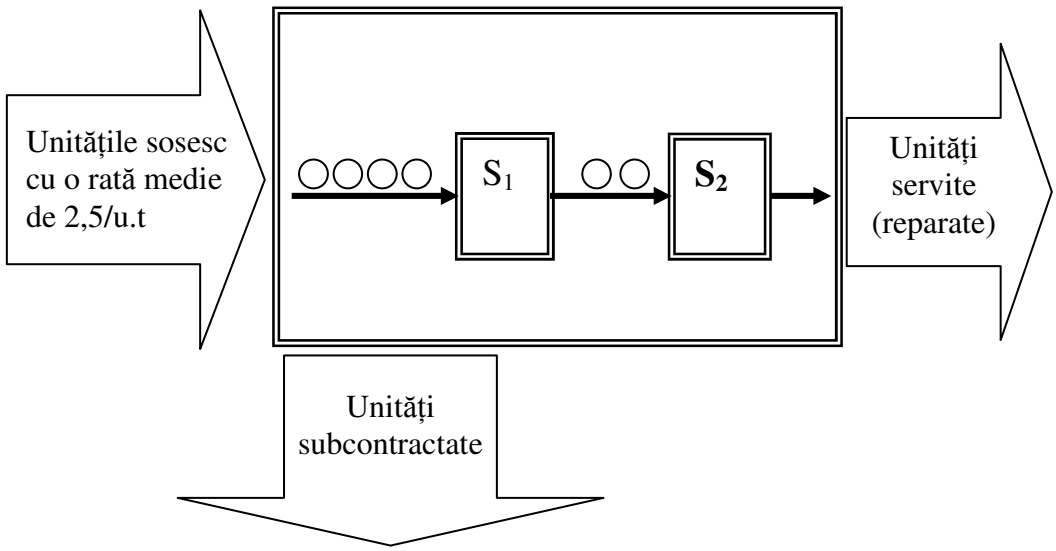


Fig.25 – *Proiectul de sistem al unei atelier de reparații cu două stații de servire*

Qbiectivele simulării

Acest exemplu ilustrează cum poate fi evaluat un proiect propus pe baza următoarelor statistici:

- gradul de utilizare a stațiilor de lucru;
- media şi deviația standard a numărului de unități în sistem;
- media şi deviația standard a numărului de unități care aşteaptă pentru servire la fiecare stație de lucru;
- media şi deviația standard a momentelor la care au fost terminate reparațiile;
- timpul mediu şi deviația standard a timpului cât o unitate este în sistem;
- numărul de unități subcontractate;
- cât la sută din timp stația de lucru 1 este blocată.

Procedura de simulare

Funcționarea atelierului de reparații este simulată, utilizând limbajul de simulare GASP, pe baza a trei evenimente:

- sosirea unei unități în sistem, ARRVL;
- terminarea servirii unei unități la fiecare stație, ENDSV;
- sfârşitul simulării, ENDSM.

Sunt folosite patru fişiere. Fişierul 1 este folosit pentru a stoca momentele programate ale evenimentelor. Atributele unităților care aşteaptă pentru servire în cozile 1 şi 2 sunt stocate în fişierele 2 şi 3. FişieruI 4 este folosit pentru a stoca momentul în care o unitate care blochează stația de lucru 1, a intrat în sistem.

Programul principal inițializează variabilele non-GASP (tabel 19) şi predă controlul subrutinei GASP, care conduce simularea până la sfârşit, predând temporar controlul celorlalte subrutine aşa cum logica simulării o cere.

Tabel 19

Variabile non-GASP	Definiție
XIST1	Nr. de unități în faza 1 (inclusiv cea din stația de servire 1)
XIST2	Nr. de unități în faza 2 (inclusiv cea din stația de servire 2)
XISYS	Nr. de unități în sistem
TLD	Momentul ultimei plecări din sistem
TBD	Timpul dintre plecările din sistem ale unităților
TITEM	Nr. total de unități care sosesc
CBALK	Nr. total de unități subcontractate
TISYS	Timpul cheltuit în sistem de o unitate
YBALK	Procentul de unități subcontractate
BLOCK	100 dacă stația 1 este blocată
XBUS(J)	- 0 dacă stațiile J=1,2 sunt libere; - 1 dacă stațiile J=1,2 sunt ocupate; - 0 dacă stația 1 nu este blocată.

Subrutina DATAN iniţializează variabilele GASP şi citeşte datele de intrare în zona de lucru NSET. Stocarea şi regăsirea informaţiei în zona NSET se realizează cu subrutinele FILEM şi respectiv, RMOVE. Regăsirea unei anumite înregistrări într-unul din cele patru fişiere se face cu subrutina FIND.

Subrutinele MONTR şi ERROR sunt folosite pentru monitorizare şi raportarea erorilor în scopul depanării.

Pentru calculul şi raportarea indicatorilor statistici sunt utilizate subrutinele COLCT, TMST şi HISTO.

Subrutina RANDU se foloseşte pentru generarea de numere pseudoaleatoare uniform distribuite în [0,1].

Subrutina EVNTS este folosită pentru generarea evenimentului următor care trebuie să se producă, ARRVL, ENDSV sau ENDSM.

Evenimentul sosire este generat de subrutina ARRVL. Deoarece sosirile sunt evenimente endogene, fiecare sosire generează sosirea următoare. Aceasta este prima funcţie executată în ARRVL. Apoi se măreşte cu 1 numărul de sosiri şi se testează dacă unitatea sosită poate sau nu să intre în sistem:

- dacă numărul unităţilor la staţia 1 este mai mare sau egal cu 5, şirul este plin şi unitatea sosită trebuie subcontractată;
- în caz contrar, unitatea poate să intre în staţia 1, sunt colectate statistici după numărul unităţilor în sistem şi se face un test pentru a determina dacă unitatea sosită trebuie plasată în şir sau direct la servire.

Schema logică a evenimentului sosirea unei unităţi în sistem, generat de subrutina ARRVL este prezentată în figura 26.

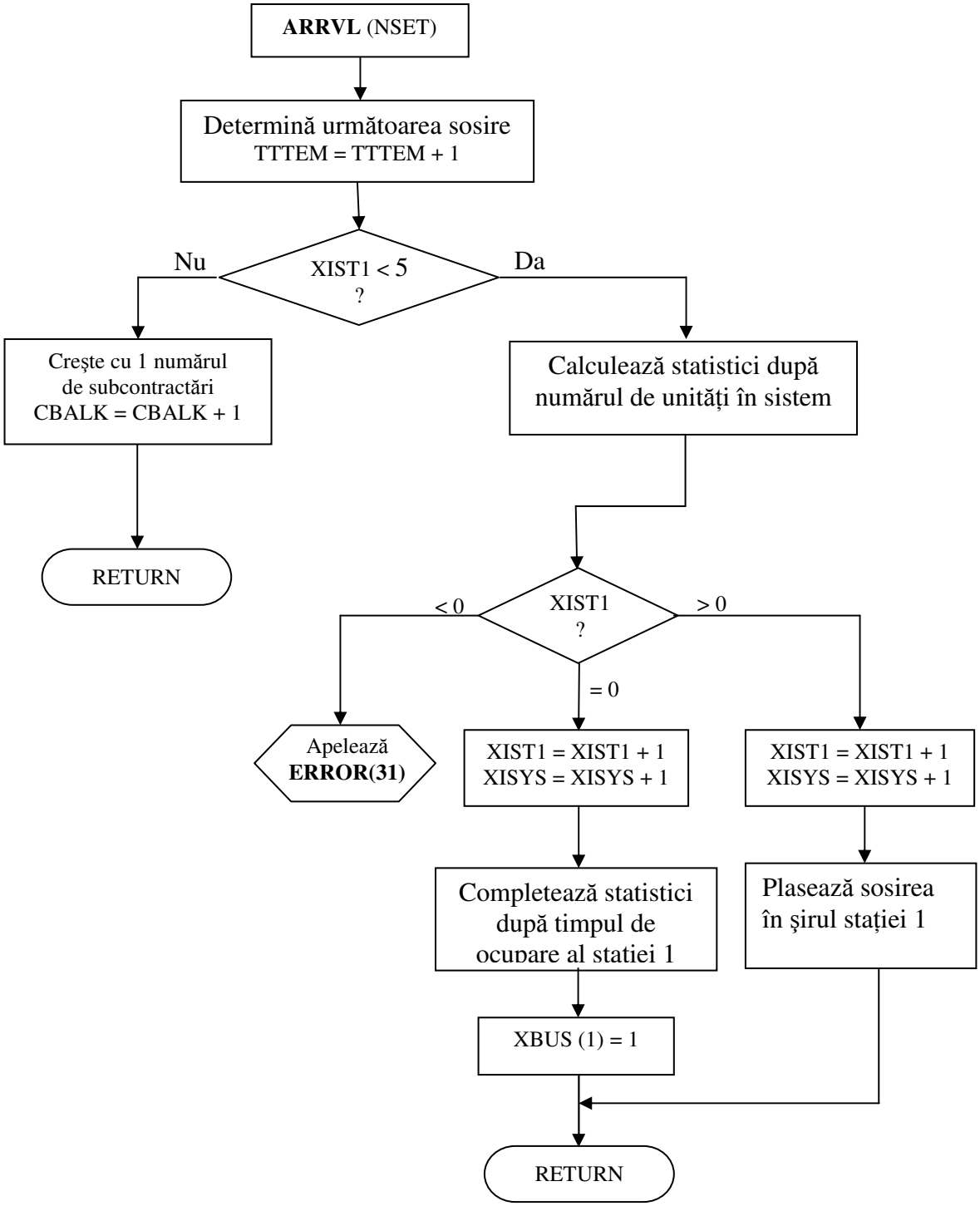


Fig. 26 – *Schema logică de calcul a subrutinei ARRVL – sosirea unei unităţi în sistem*

Subrutina ENDSV simulează evenimentul terminarea servirii la ambele staţii de lucru. Schema logică a acestei subrutine este prezentată în figurile 27.a şi 27.b.

a) Vom considera mai întâi un evenimet sfârşit de servire la staţia 1.

Deoarece funcţionarea staţiei 1 depinde de numărul de unităţi din staţia 2 se face un test pentru a determina acest număr. Astfel:

- Dacă sunt 3 unităţi în staţia 2, staţia 1 este blocată. Sunt calculate statistici pentru staţia 1 după timpul de ocupare şi blocare, se poziţionează staţia 1 la starea neocupată iar unitatea care a provocat blocarea este înscrisă în fişierul 4 pentru a reţine atributele sale, stocate anterior în evenimentul sfârşit de servire pentru prima staţie. Această procedură uşurează înscrierea unităţii care a produs blocarea în şirul staţiei 2 la momentul corespunzător.
- Dacä sunt două unităţi în staţia 2, unitatea care termină servirea la staţia 1 intră în şirul staţiei 2, la momentul curent TNOW, fără nici o întârziere de trecere;
- Dacă există o unitate în staţia 2, unitatea care termină servirea la staţia 1 intră în şirul staţiei 2, la momentul TNOW + 0.1;
- Dacă nu există nici o unitate la staţia 2, pentru unitatea care tocmai a terminat sosirea la staţia 1 se programează terminarea servirii pe staţia 2 la momentul TNOW+0.2+ durata servirii. Unităţile sunt prelucrate la staţia 2 în conformitate cu momentele sosirii lor (disciplina FIFO). În continuare se testează numărul de unităţi care aşteaptă la staţia 1. Dacă nu sunt unităţi care aşteaptă pentru servire, sunt colectate statistici după timpul de funcţionare, se poziţioneaza staţia 1 la starea "liberä" şi se revine la rutina care a chemat.

Dacă există cel puţin o unitate care aşteaptä servirea, se programează evenimentul "terminarea servirii", se colectează statisticile corespunzătoare şi se poziţionează staţia 1 la starea "ocupată.

b) Dacă evenimentul ENDSV este pentru staţia 2, se colectează statistici după numărul de unităţi în sistem, după timpul petrecut în sistem de unitatea care a terminat servirea şi după timpul dintre plecări din sistem. De asemenea, se micşorează cu 1 numărul de unităţi din sistem şi de la staţia 2. Se face apoi un test pentru a determina dacă sunt unităţi care aşteaptă servirea la staţia 2. Dacă nu, sunt colectate statistici după timpul de ocupare al staţiei 2 şi aceasta se poziţionează în starea "liberă".

Dacă sunt unităţi care aşteaptă servirea, se şterge prima unitate din fir şi pentru ea se programează evenimentul sfârşit de servire.

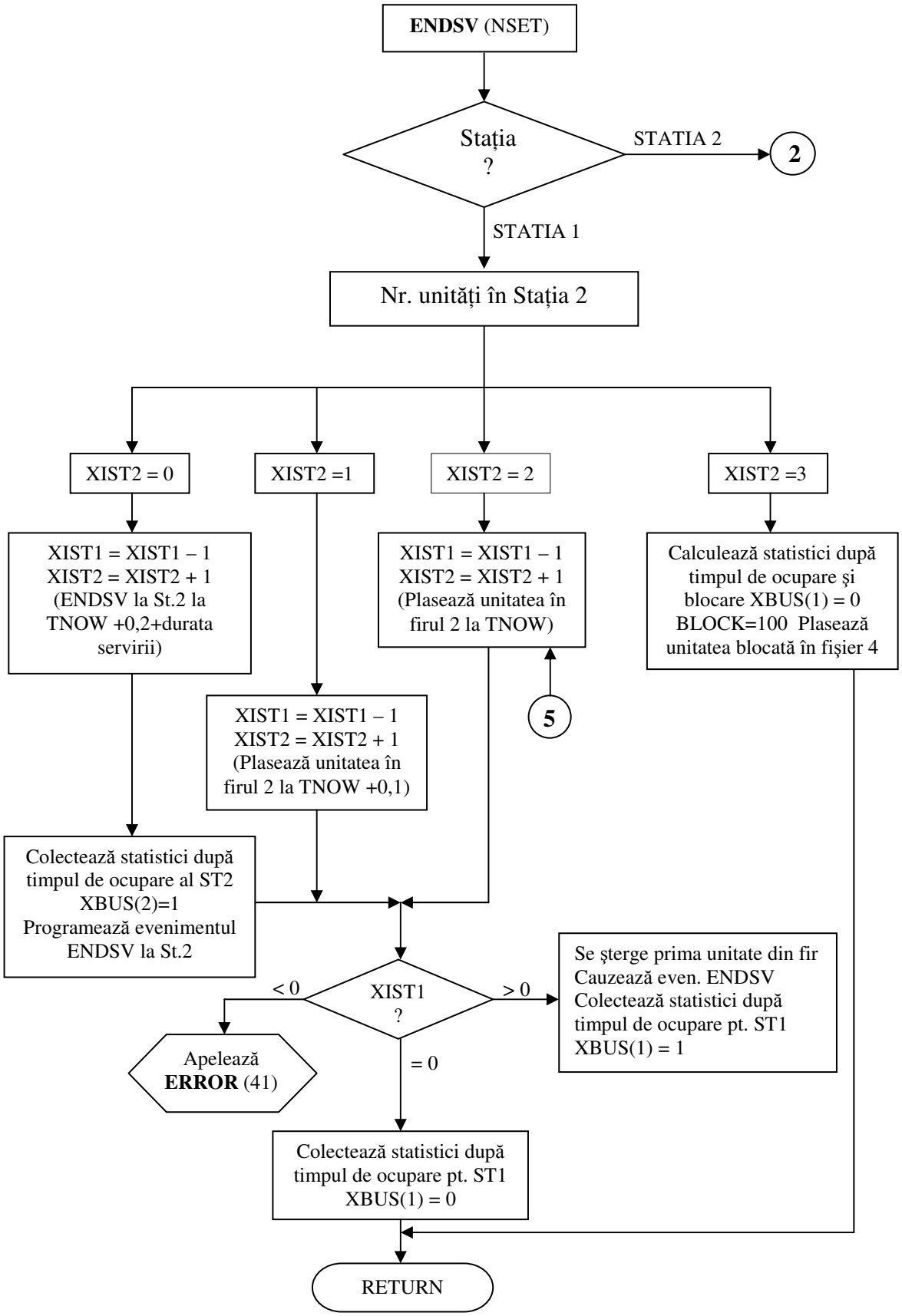


Fig.27.a – *Evenimentul sfârşit de servire pentru o unitate*

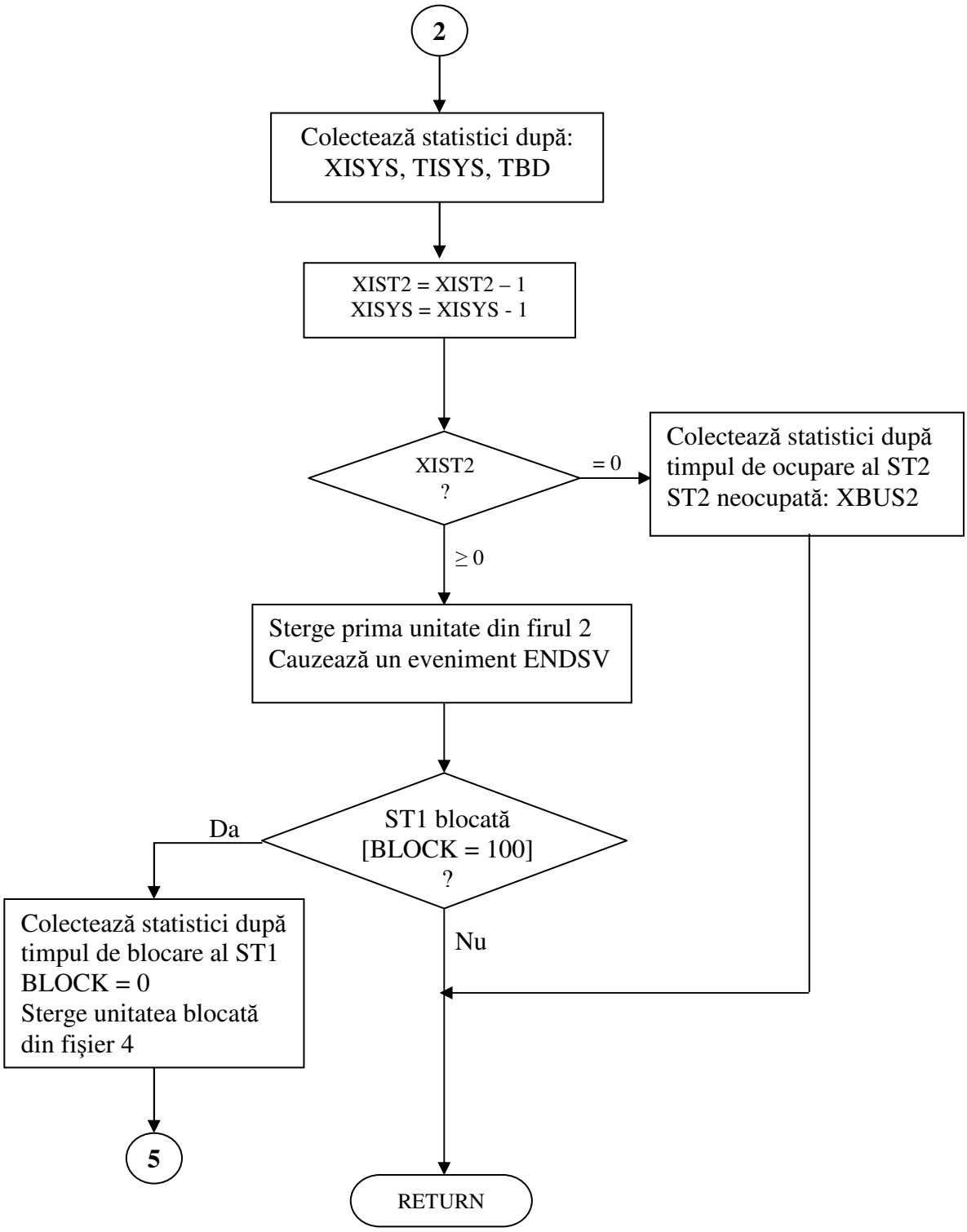


Fig.27.b – *Evenimentul sfârşit de servire pentru o unitate*

În continuare se testează dacă stația 1 este sau nu blocată. Dacă nu este blocată se revine la subrutina care a chemat; în caz contrar, sunt colectate statistici după timpul cât stația 1 a fost blocată. Apoi, stația 1 este deblocată, unitatea care a provocat blocarea este ştearsă din fişierul 4 şi se face un transfer la 5, care determină intrarea unei unități în stația 1 când există două unități în stația 2, precum şi ordonarea unităților în fir.

Când se produce un eveniment sfârşit de simulare ENDSM, sunt procesate toate unitățile intrate în sistem până la acel moment, sunt actualizate statisticile, iar variabilele de control vor indica sfârşitul simulării şi care sunt rapoartele finale dorite.

Subrutina OTPUT este folosită pentru a tipări parametrii de intrare asociați cu simularea atelierului de reparații, procentul şi numărul de unități subcontractate, numărul total de unități care sosesc pentru reparații.

Concluzii

Rezultatele simulării arată câteva fapte interesante despre proiectul propus. Timpul mediu de servire pentru o unitate prelucrată prin ambele stații a fost de 0.75 u.t., însă unitățile stau în sistem în medie 3.11 u.t. În felul acesta, în medie, aproximativ 2.36 u.t. sunt cheltuite de către fiecare unitate aşteptând pentru servire. Unitățile părăsesc sistemul cu un timp mediu între plecări de 0.54 u.t.

Deoarece timpul mediu de servire pentru stația 2 este de 0.5 u.t., această stație este foarte ocupată. Acest fapt este de asemenea ilustrat de indicele utilizării ei care este 0.91. Această valoare este extrem de mare în comparație cu utilizarea primei stații de lucru, care are o medie a utilizării de 0.47. Unul din motivele pentru utilizarea inferioară a stației de lucru 1 este blocarea care se produce datorită dimensiunii limitate a şirului de aşteptare pentru stația de lucru 2. Astfel, în medie timpul cât stația de lucru 1 a fast blocată este 45.33% şi de asemenea, circa 28% din unitățile care sosesc sunt subcontractate.

Bibliografie

1. ALAN PRITSKER., PHILIP KIVIAT, *Simulation with GASP II*, Seria Automatic Computation, USA, 1969;
2. ANDREICA MARIN, STOICA MARCEL, *Modelarea şi simularea proceselor economice*, Lito ASE, Bucureşti, 1994;
3. DOBRE ION, MUSTATA FLOARE, *Simularea proceselor economice*, Editura INFOREC, BUCUREşTI, 1996;
4. FLOARE MUSTATA, PAUN MIHAI, DOBRE ION, *Simularea numerică a proceselor economice - Aplicaţii,* Editura ASE, Bucureşti, 2000.
5. PAUN MIHAI, *Cercetări şi programe pentru optimizarea unor modele de cercetări operaţionale,* preprint, LCCE-ASE, Bucureşti, 1992;