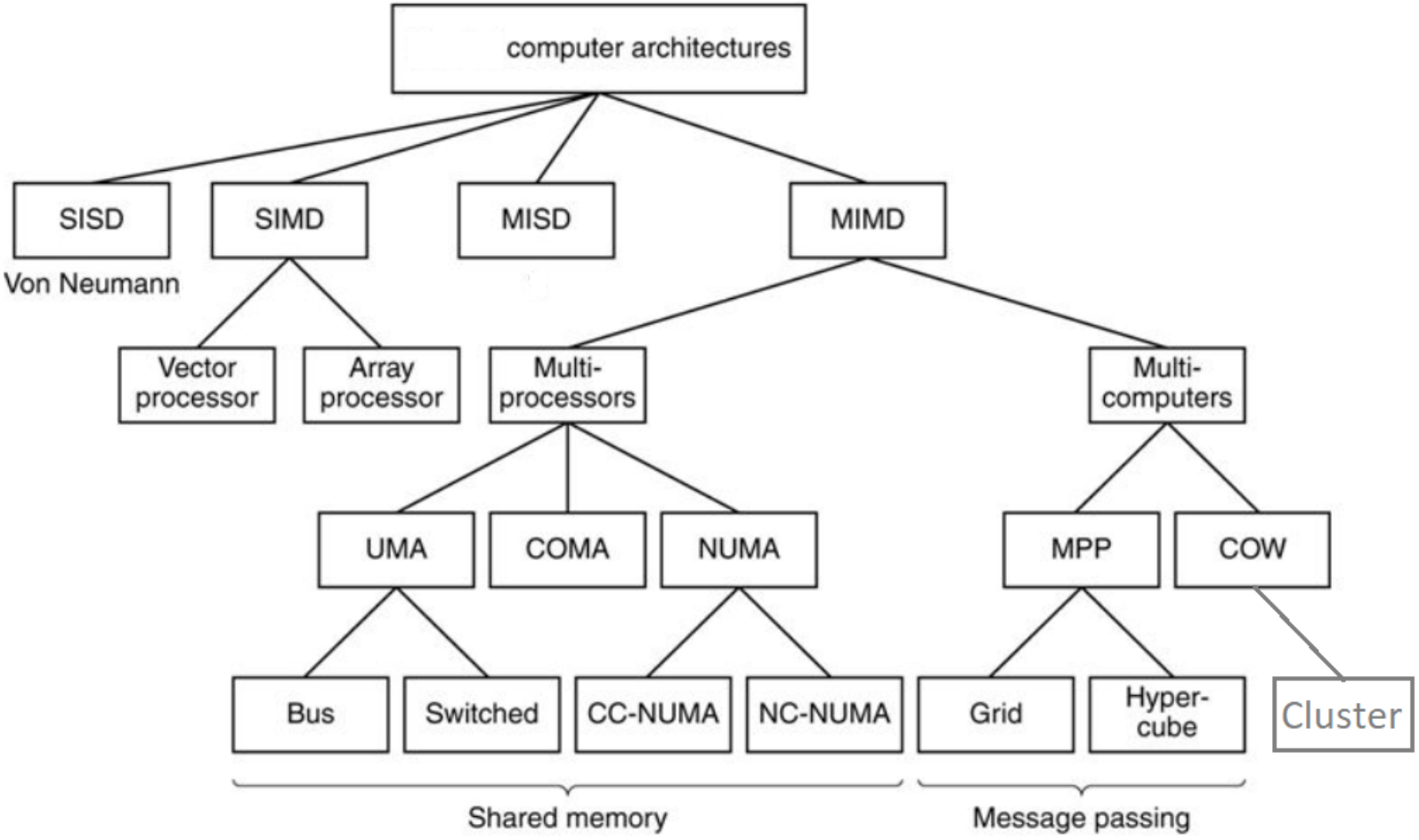
Taxomomii
Modele de calcul Paralel

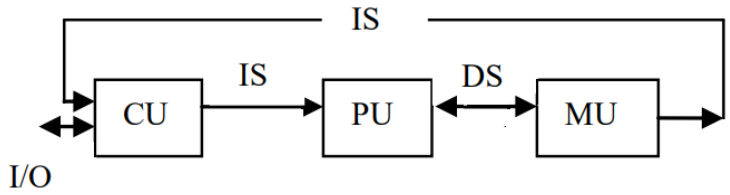
Arhitecturi de calculatoare



Taxonomii

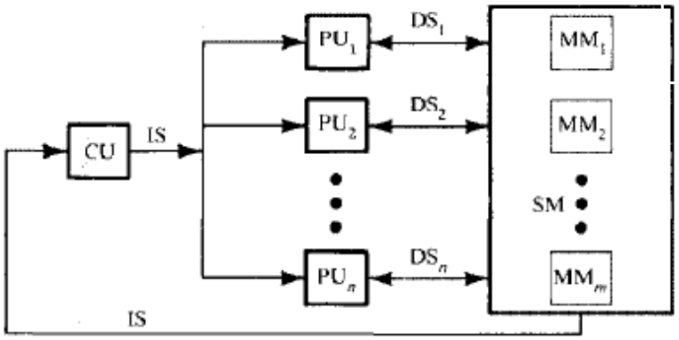
1. ***Taxonomia lui Flynn (bazata pe relatia dintre fluxul de instructiuni si fluxul de date:***

- SISD (single instruction single data flow)
- SIMD (single instruction multiple data flow)
- MISD (multiple instruction single data flow)
- MIMD (multiple instruction multiple data flow)

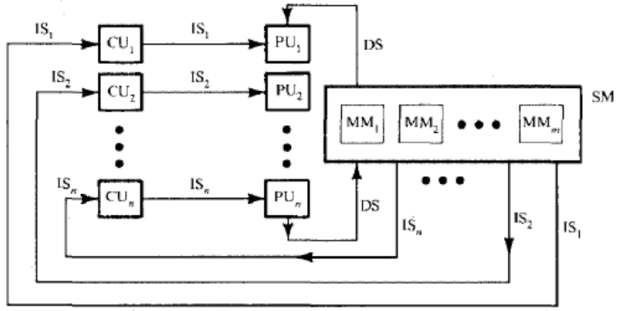


(a) SISD Uniprocessor Architecture

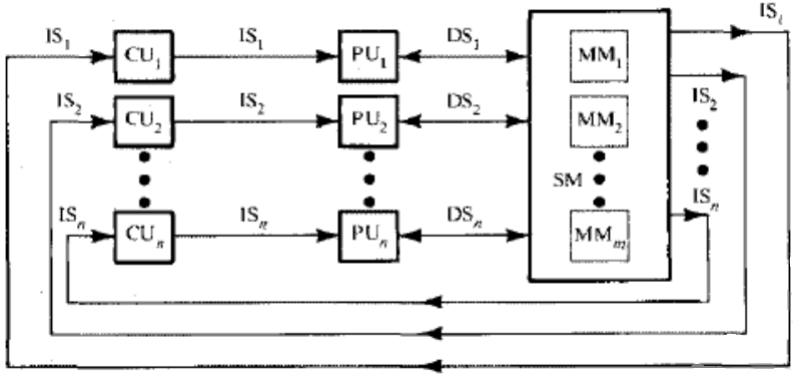
(b) SIMD Architecture



(c) MISD Architecture (the Systolic Array)



(d)MIMD architecture



Taxonomii

Taxonomia lui Shore:

- masina seriala pe cuvant paralela pe biti
- masina paralela pe cuvant seriala pe biti
- masina ortogonala
- masina masiv neconectat
- masina masiv conectat

Structurala

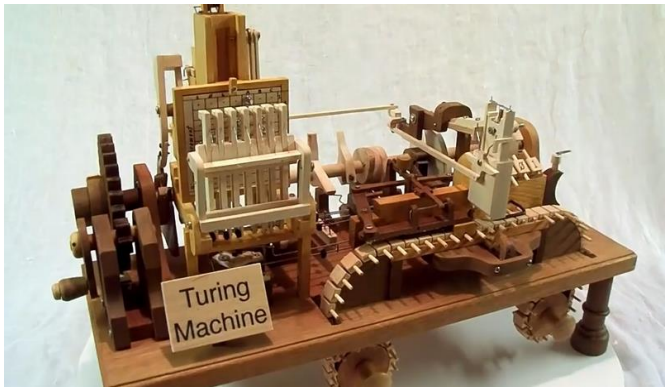
- uniprocesoare seriale
- unicalculatoare paralele bazata pe paralelism functional si pipeline
- masive de procesoare

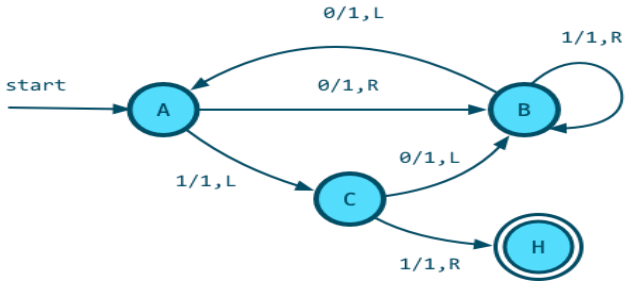
Modele ale calculului paralel

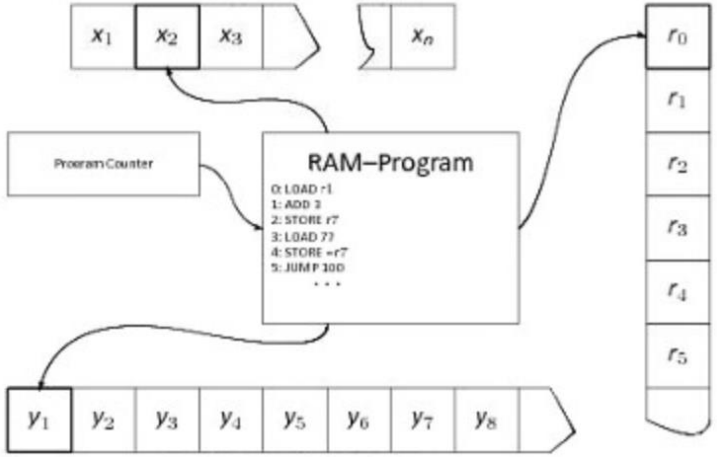
1. Model RAM (Random Acces Machine)
2. PIPELINE
3. PROCESOARE DE VECTORI
4. MULTIPROCESOARE cu memorie partajata
5. MULTIPROCESOARE cu transfer prin mesaje
6. PRAM (Parallel Random Acces Machine)
7. PROCESARE SISTOLICA
8. PROCESARE DATA FLOW

O mașină cu acces aleatoriu (RAM):

- o bandă de intrare infinită X cu poziția capului citit i ∈ N,
- o bandă de ieșire infinită Y cu poziția capului de scriere o ∈ N,
- un registru contor de program PC
- un program format dintr-un număr finit de instrucțiuni P [1], ..., P [m],
- o memorie constând dintr-o succesiune infinită de registre r0,r1,r2







RAM = (X ,Y, S, s0, f, Z)

S multimea finita a starilor automatului;
s0 starea initiala;

X alfabetul finit al benzii de intrare;

Y alfabetul finit al benzii de iesire;

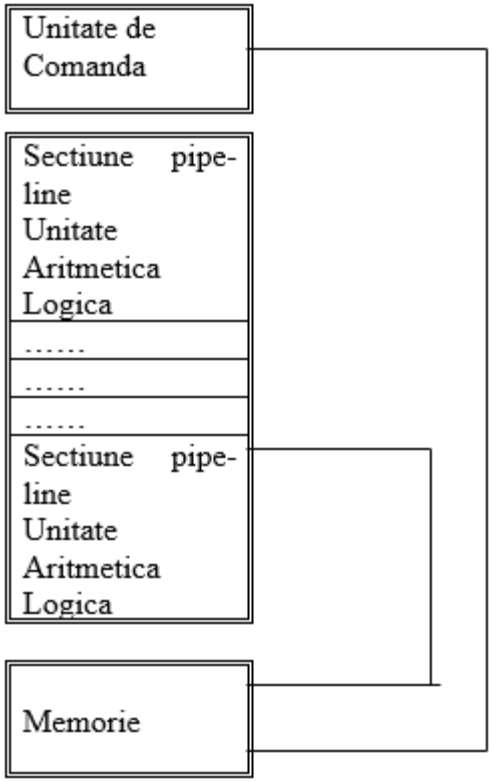
f functia de tranzitie, eventual partial definite
f : S x X -->S x Y

Z inclus in S - multimea starilor finale

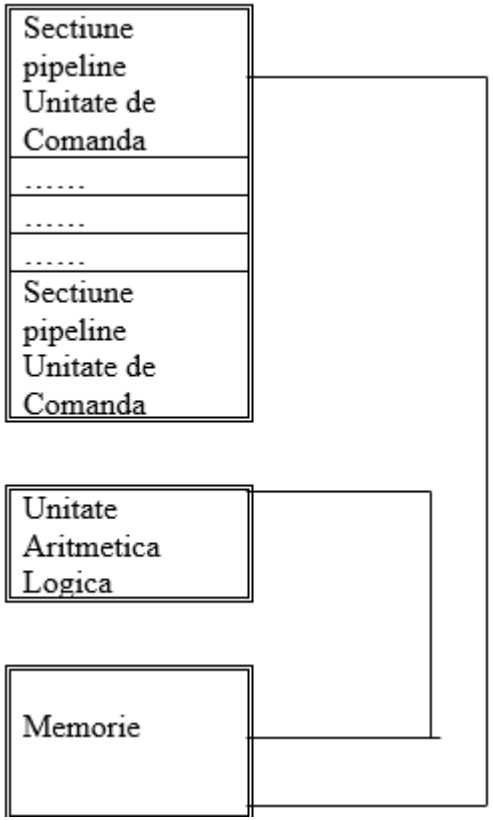
Structura pipeline.

Conceptul de pipeline consta in a imparti un task T in subtask-uri T_1 , T_2 , ...,T_k si de a le atribui unor elemente de procesare.
Paralelismul se poate realiza fie la nivel de:

- prelucrare aritmetica;



citire, interpretare, executie instructiuni



Procesoare de vectori.

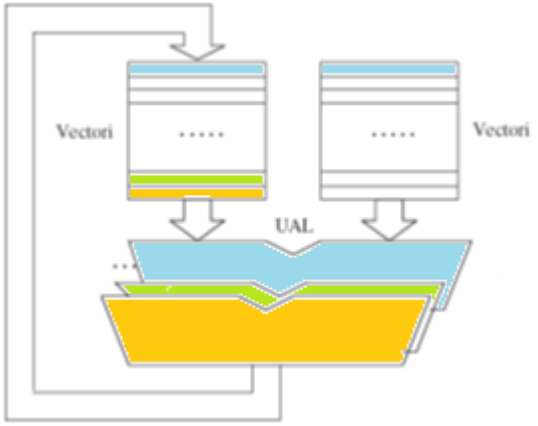
Multiprocesoare cu memorie partajata.

Procesoare de vectori.

In cadrul acestui model, exista un set de instructiuni care trateaza vectorul ca operand simplu. (SIMD) sau
elemente de procesare vectoriala.(UAL de tip pipeline si registre de vectori).

O astfel de structura o gasim in sisteme monoprocesor care au ca extensie procesare de vectori.

Acopera o gama redusa de probleme cu caracter vectorial.



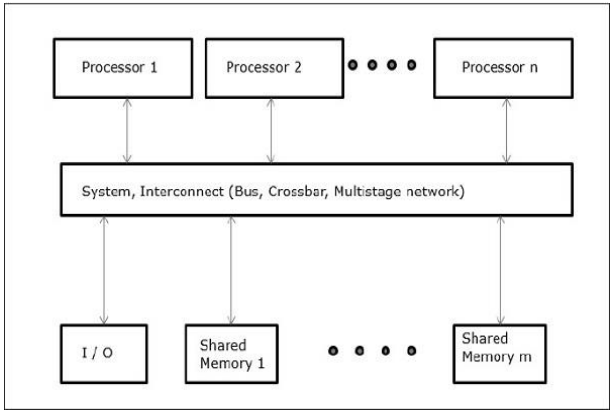
Multiprocesoare cu memorie partajata.

Acest model descrie structurile in care fiecare procesor contine propria unitate de prelucrare, propria memorie si propria unitate de comanda si comunica prin intermediul unei retele de comutare cu module de memorie partajate.

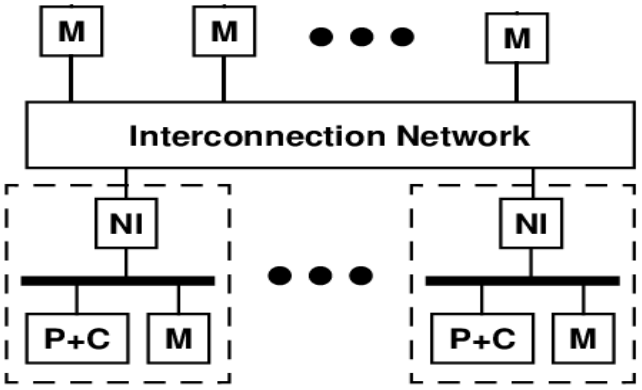
Fiecare procesor executa propriul set de instructiuni din memoria locala sau memoria partajata.

Reteaua de comutare permite schimbul de informatii intre procesoare si intre procesoare si memoria partajata.

UMA



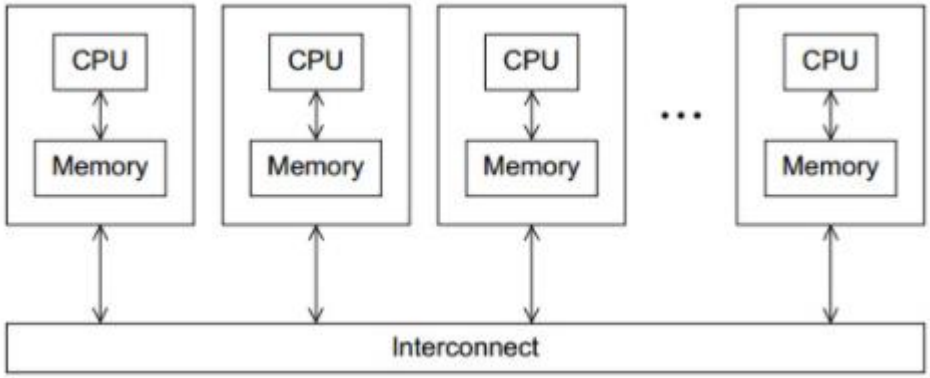
NUMA



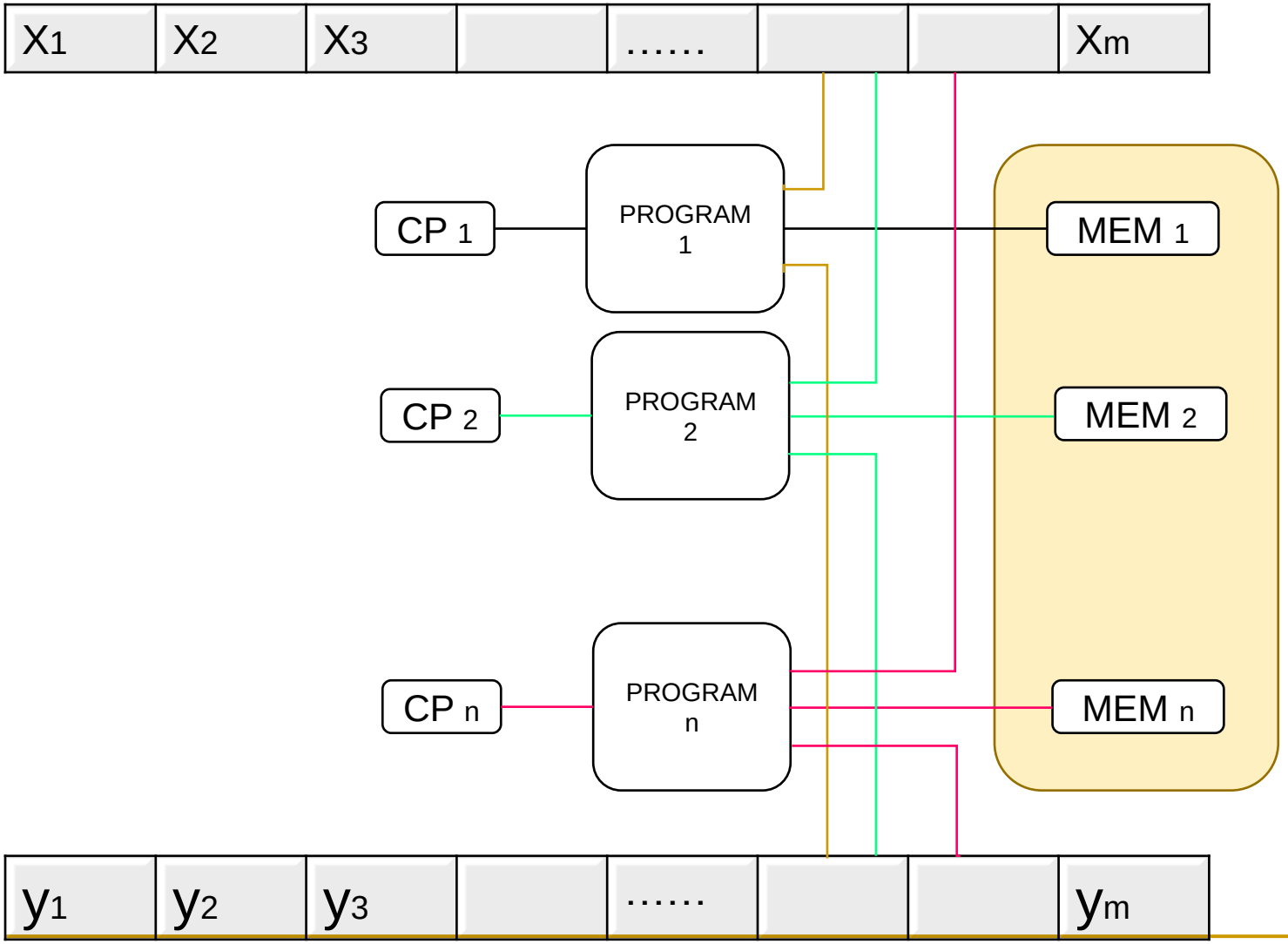
Multiprocesoare cu transfer prin mesaje

Multiprocesoare cu transfer prin mesaje.

- In acest model memoria este distribuita fiecarui procesor astfel incit fiecare procesor dispune de propriul program si propria memorie de date.
- Comunicarea datelor partajate este realizata prin schimburi de mesaje prin intermediul unei retele de comutare mesaje.
- Acest model este diferit de modelul cu memorie partajata deoarece interconexiunea se realizeaza prin mesaje si nu prin acces direct (comutare de circuite).



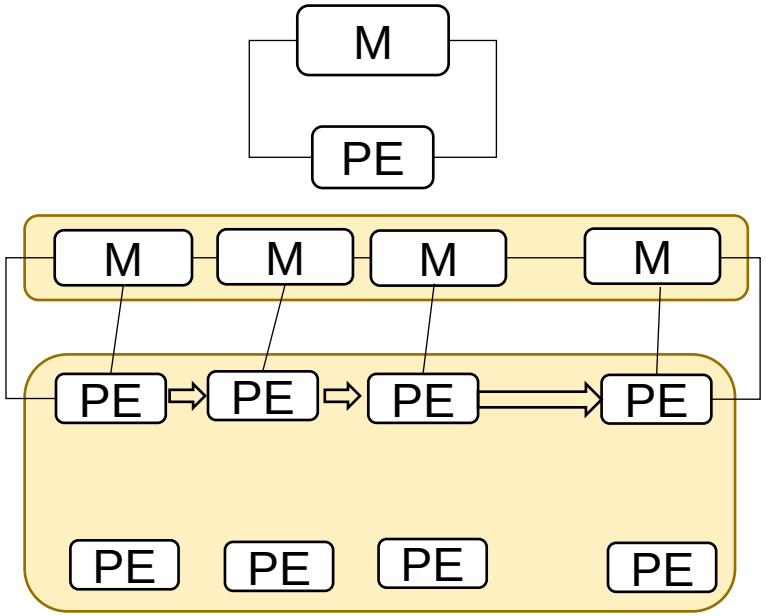
Modelul PRAM -parallel RAM

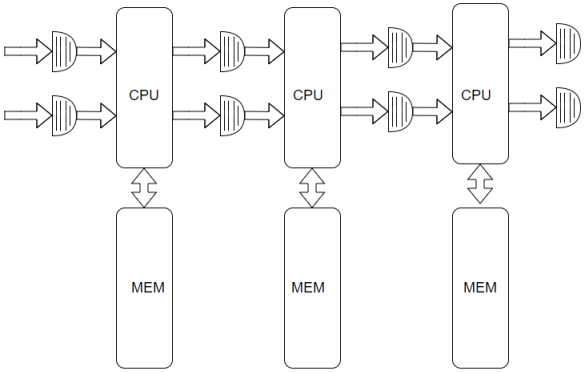


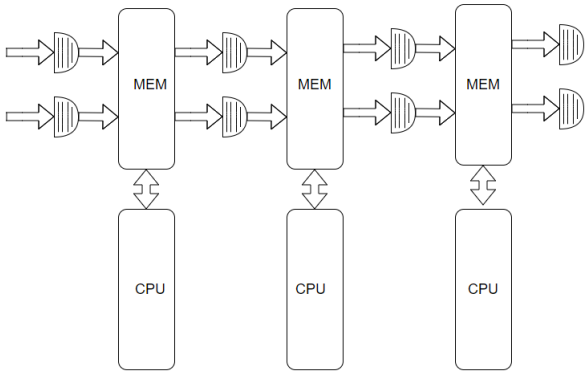
Procesare sistolica

Procesare sistolica.

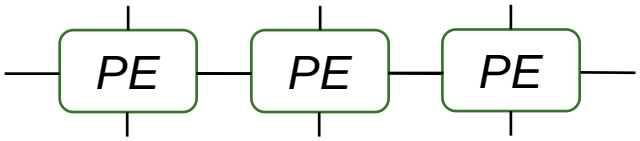
Modelul procesarii sistolice consta din elemente de procesare identice aranjate intr-o structura pipeline.
Procesarea sistolica este caracterizata prin interconexiuni simple si regulate ceea ce permite integrarea VLSI.

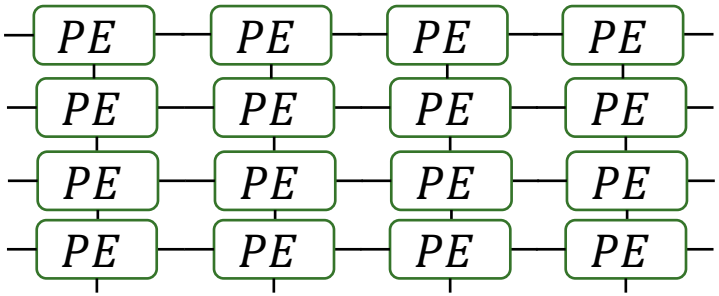


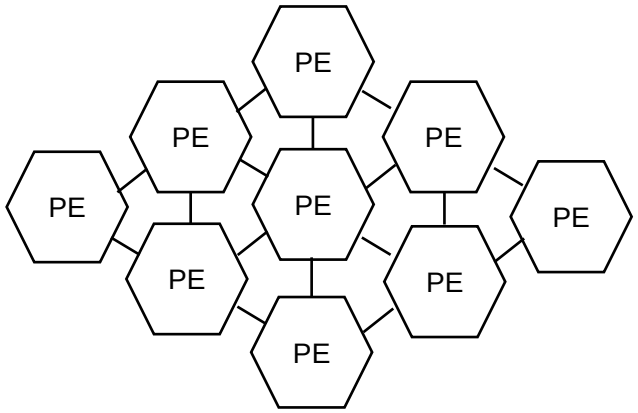


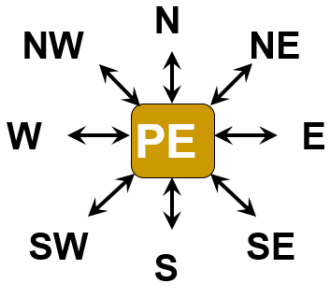


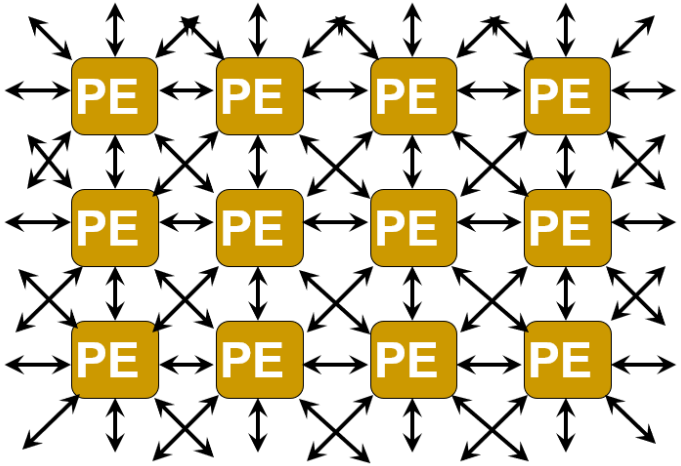
Modalitati de conexiune Sistolica





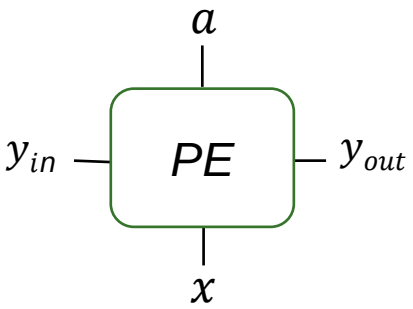






Exemplu

- Fie
- $A=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & an_2 & \cdots & a_{nn} \end{bmatrix}$ $Y=\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$ $X=\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ $Y_0=\begin{bmatrix} y_{01} \\ y_{02} \\ \vdots \\ y_{0n} \end{bmatrix}$
- Dorim sa calculam
- $Y=A*X+Y_0$
- Presupunem ca avem un element de procesare sistolica

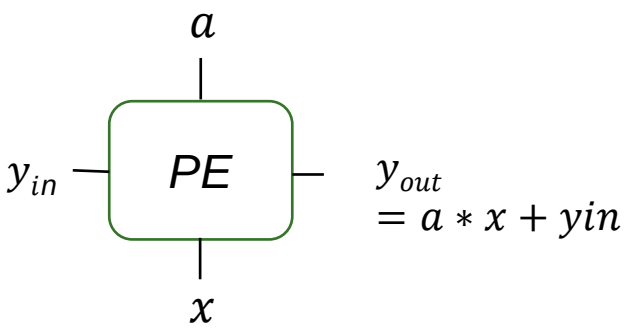


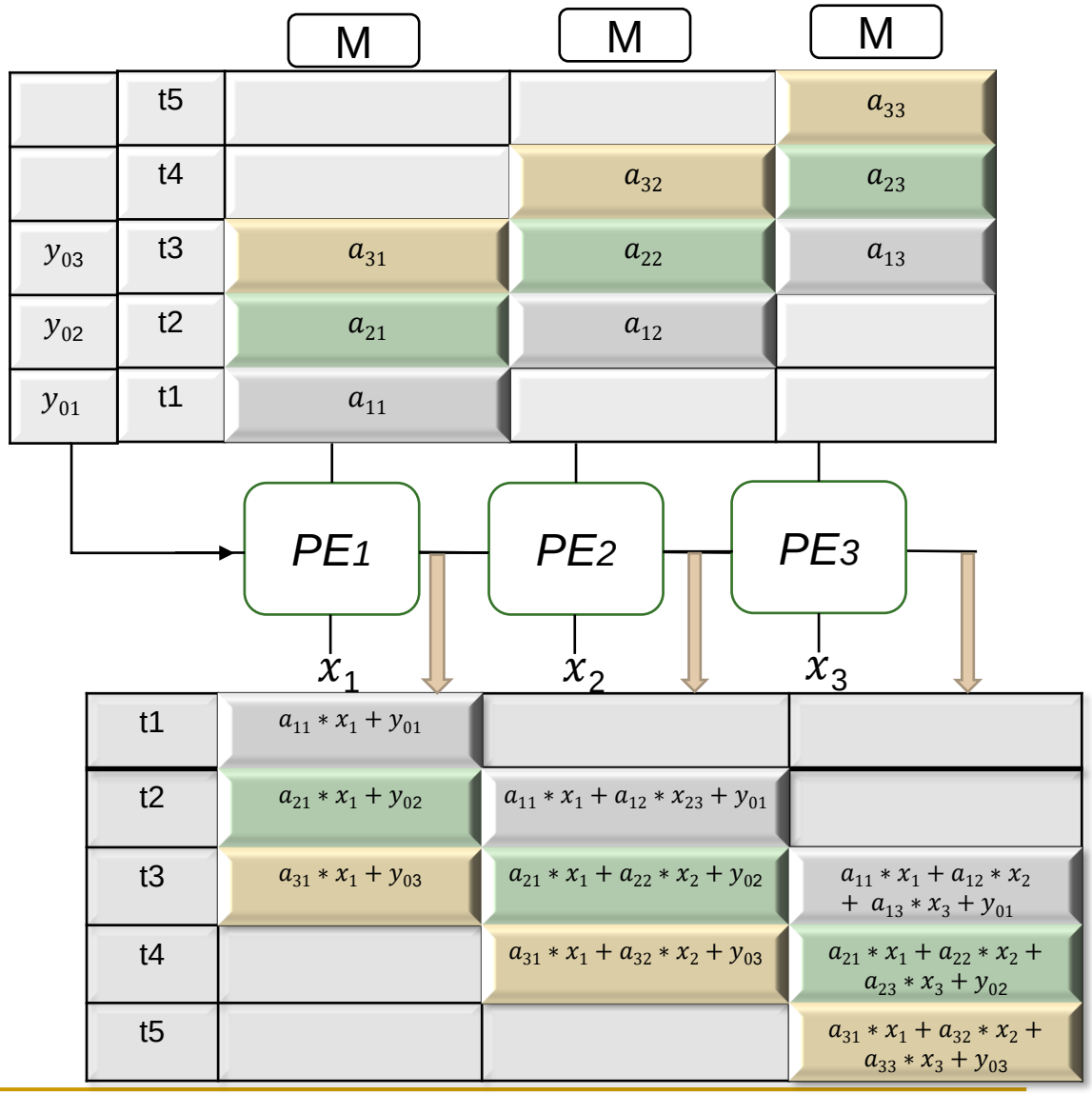
$$y_{out} = a * x + yin$$

Y	A					X	Y_0
y_1	a_{11}	a_{12}	a_{13}		a_{1n}	x_1	y_{01}
y_2	a_{21}	a_{22}	a_{23}		a_{2n}	x_2	y_{02}
y_3	a_{31}	a_{32}	a_{33}		a_{3n}	x_3	y_{03}
y_n	a_{n1}	a_{n2}	a_{n3}		a_{nn}	x_n	y_{0n}

Exemplu pentru n=3, structura unidimensionala

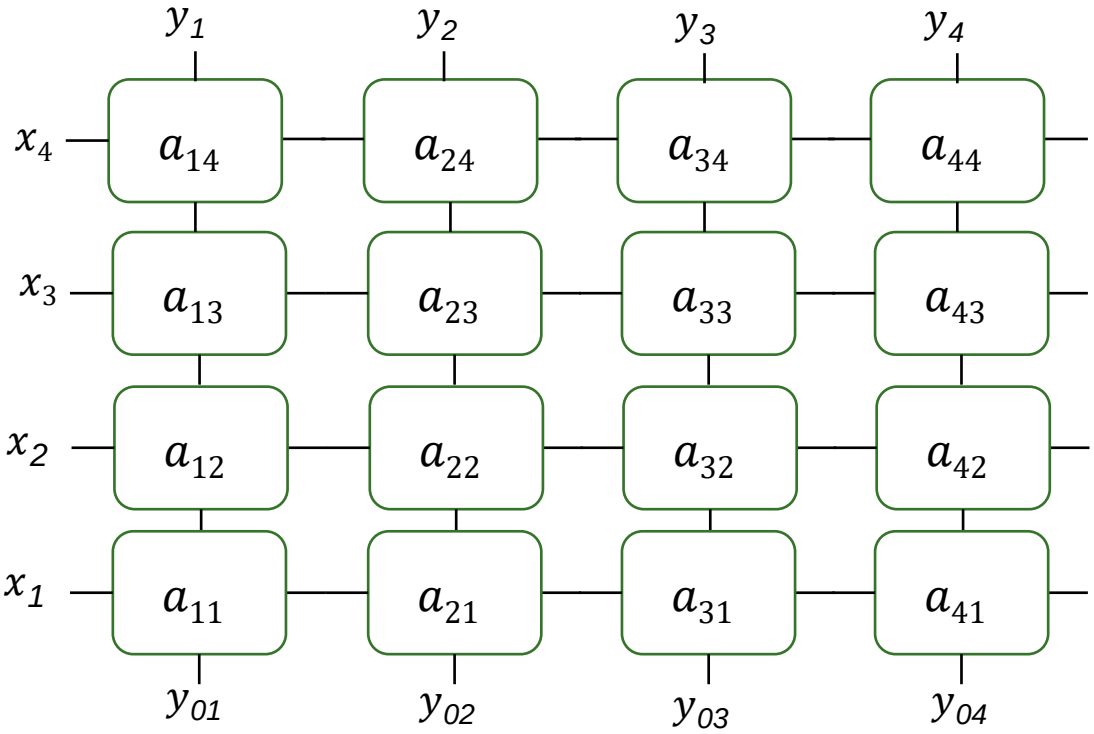
- $y_i = \sum_{j=1}^{n} a_{ij} * xj + y_{0i}$
- $y_1 = a_{11} * x_1 + a_{12} * x_2 + a_{13} * x_3 + y_{01}$
- $y_2 = a_{21} * x_1 + a_{22} * x_2 + a_{23} * x_3 + y_{02}$
- $y_3 = a_{31} * x_1 + a_{32} * x_2 + a_{33} * x_3 + y_{03}$

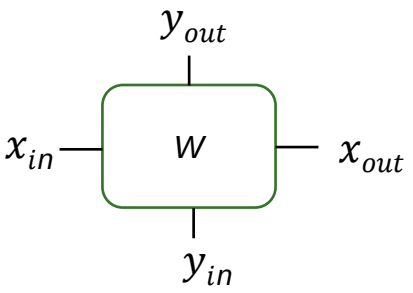




Exemplu pentru n=4, structura bidimensionala

- $y_i = \sum_{j=1}^{n} a_{ij} * xj + y_{0i}$
- $y_1 = a_{11} * x_1 + a_{12} * x_2 + a_{13} * x_3 + a_{14} * x_4 + y_{01}$
- $y_2 = a_{21} * x_1 + a_{22} * x_2 + a_{23} * x_3 + a_{24} * x_4 + y_{02}$
- $y_3 = a_{31} * x_1 + a_{32} * x_2 + a_{33} * x_3 + a_{34} * x_4 + y_{03}$
- $y_4 = a_{41} * x_1 + a_{42} * x_2 + a_{43} * x_3 + a_{44} * x_4 + y_{04}$

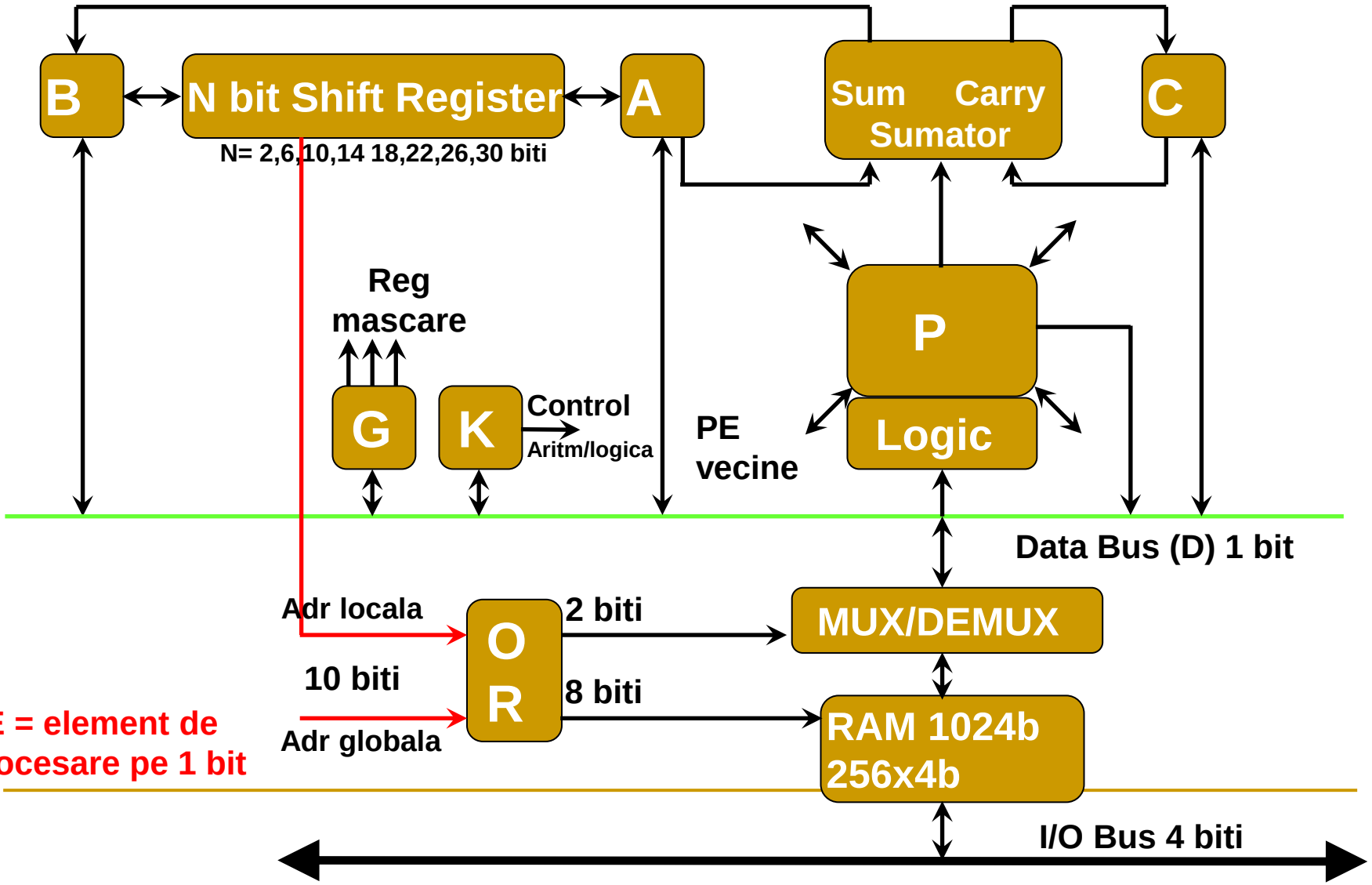




$$y_{out} = w * xin + yin$$

$$x_{out} = xin$$

Exemplu de masina sistolica, Arhitectura Masinii Blitzen – PE



PE = element de procesare pe 1 bit

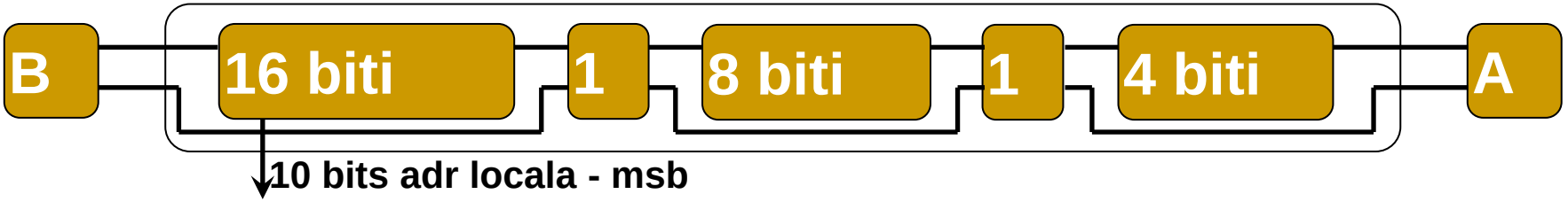
Elementele PE

- P – registru pe 1 bit
  - Operanzi
  - Operatori in functiile aritmetice
  - Utilizat la rutare: primeste ops de la PE-uri adiacente
- K – registru de control aritmetico-logic
  - Faciliteaza aparitia ops aritmetice inverse (+/-)
  - Algoritmi de impartiri (non-returning-division)
- G – registru de mascare
  - Se indica daca operatia se executa sau nu
- A & B – registru pe 1 bit
  - Preiau rezultatele de la sumator
  - Contribuie la suma

Elementele PE

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

- Sumator
  - ADD: A + P + C → B, C
  - HADD: A + C → B, C
- Shift Register (SR) – Registru de deplasare



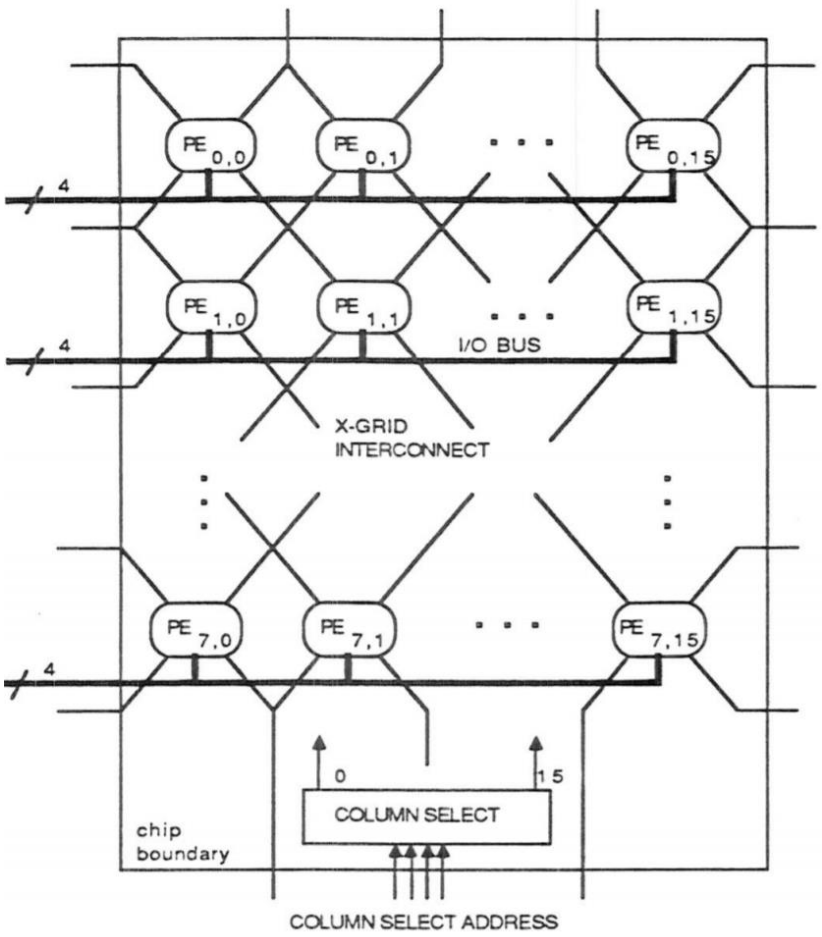
  - N stagii = 2, 6, 10, 14, 18, 22, 26, 30
  - 30 biti + A & B = 32 biti
  - Bitul obtinut prin deplasare nu e neaparat salvat
  - La fiecare pas SR se deplaseaza cu o pozitie R/L
  - SR-ul poate fi sters

Memoria & I/O Bus

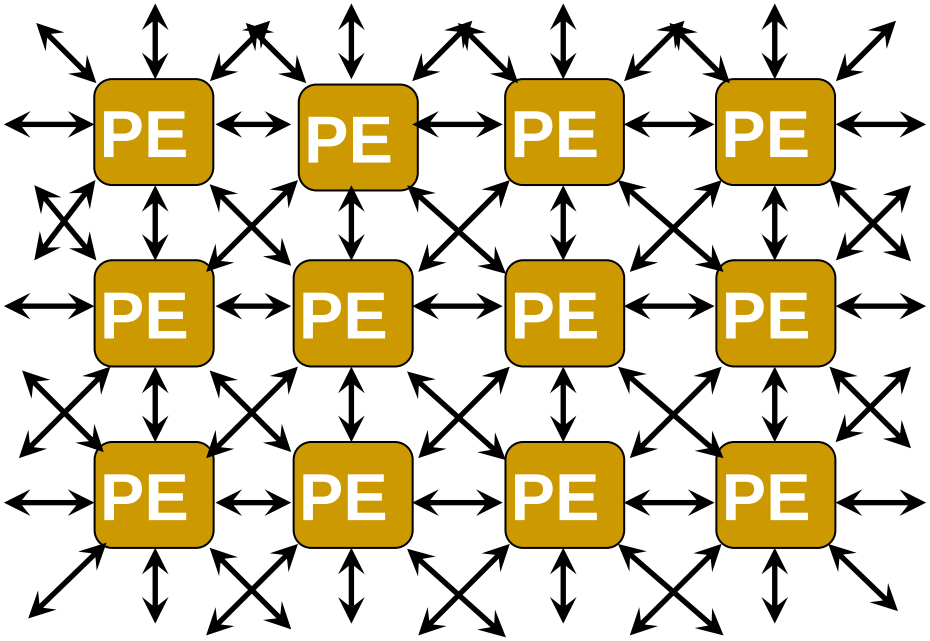
- Accesul la memoria RAM
  - Cu o adresa globala de forma 2^p (p e multiplu de 2)
  - Cu o adresa locala → specific masinii Blitzen (10 biti)
- I/O Bus
  - Formata din 4 biti si utilizata pentru conectarea PE-urilor la resurse externe de stocare a datelor
  - O magistrala I/O este folosita in comun de catre 16 PE-uri

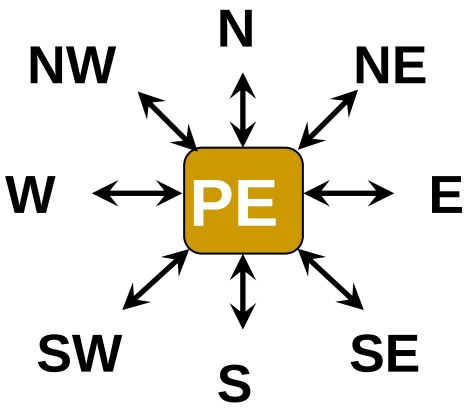
Chip-ul Blitzen

- Un Chip Biltzen este format din
  - 8 magistrale
  - 8 linii a cate 16 procesoare fiecare → 128 PE/chip, fiecare cu cate 1K de RAM
  - Fiecare PE functioneaza la 20MHz cu 8x4=32 biti/ciclu



Un Sistem Blitzen



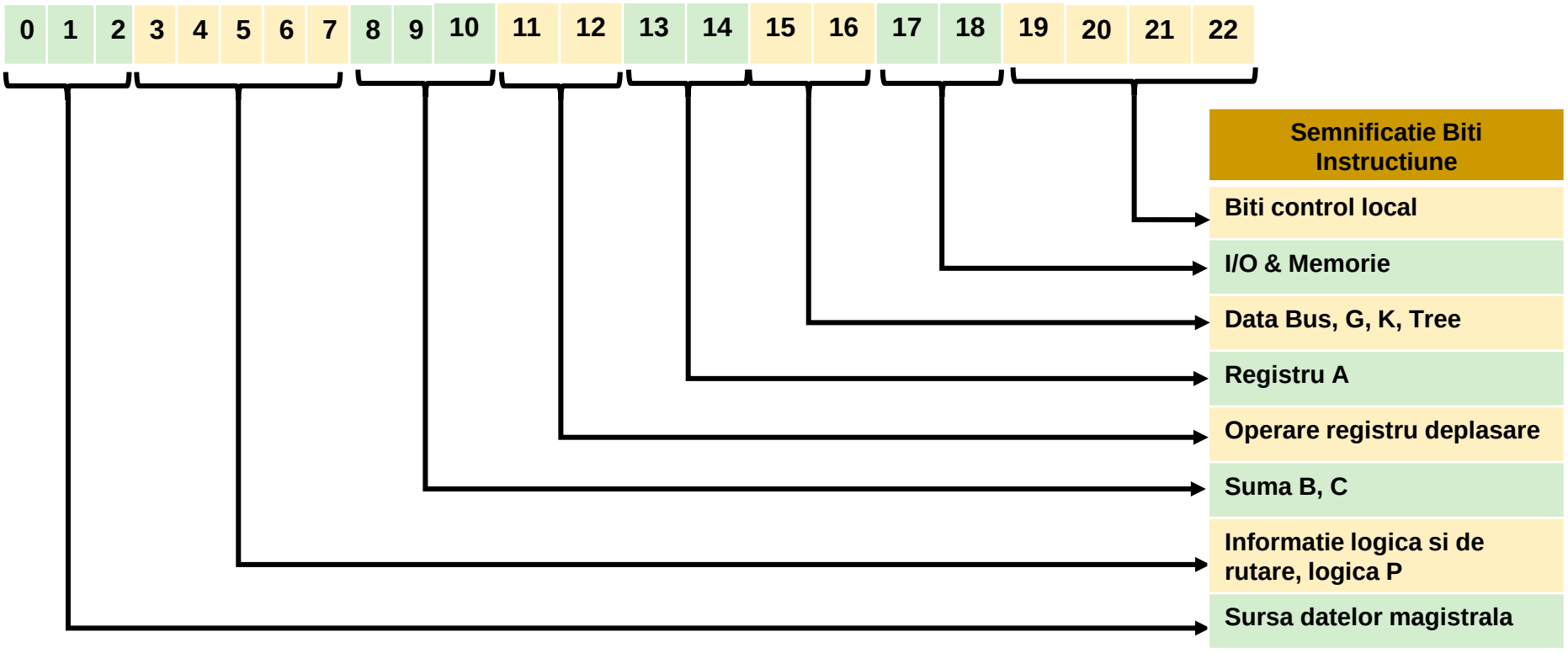


Structura X-Grid

- Sunt 8 directii de rutare
  - N, S, E, W, NE, SE, SW, NW
- Chip-urile Blitzen se grupeaza pe linii de 128 chip-uri → 128 x 128 PE = 16384PE (sistem Blitzen)

Formatul Instructiunilor

- La fiecare ciclu se executa o instructiune intr-un pipeline cu 3 stagii: Decodare, Broadcast, Executie
- Instructiunile sunt pe 23 de biti organizati astfel:



Semnificatia Bitilor Instructiune

- Bitii 0, 1, 2: sursa datelor de pe magistrala
  - Registrii A, B, C
  - Registrii G sau K
  - Registrul P
  - Memorie (cand este adresata memoria se executa un read 1b)
- Bitii 3, 4, 5, 6, 7:
  - Operatia logica ce trebuie operata in P
  - Operatia de rutare catre unul din cei 8 vecini
  - Configurarea registrului de deplasare (no assigns to P)
- Bitii 8, 9, 10: operatii executate de sumator
  - Suma & Carry din sumator catre registrii B si C
  - Pot fi utilizate si pentru incarcarea lui B & C

Semnificatia Bitilor Instructiune

- Bitii 11,12: operatii de deplasare/shiftare
  - Stanga (L)/Dreapta (R)/Stergere (D)
- Bitii 13,14: operatii cu registrul A
  - Setare directa
  - Setare cu continului registrului de deplasare/shiftare
- Bitii 15,16: rezultatul e transferat din magistrala de date
  - Catre registrii G, K
  - Catre OR-ul ierarhic (Tree)
- Bitii 17,18: prelucrarea informatiilor din
  - Memoria locala (R/W operations)
  - I/O Bus (I/O operations)

Semnificatia Bitilor Instructiune

- Bitii 19,20,21,22: control local
- Bit 19: operatii mascate & nemascate ale registrului P
  - Bit 19 = 0: operatiile P nemascate
  - Bit 19 = 1: operatiile P mascate;
- Bit 20 : identifica operatii mascate efectuate de memorie, sumator, SR, & registrii A, B sau C
  - Bit 20 = 0: operatiile nemascate
  - Bit 20 = 1: operatiile mascate;
- Bit 21 : poate identifica daca operatiile sunt transmise sau daca a avut loc un broadcast. Este folosit pt:
  - Bit 21 = 0: operatiile curente
  - Bit 21 = 1: op complementate
- Bit 22 : operatii cu
  - Bit 22 = 0: adresa globala
  - Bit 22 = 1: cu cei 10 biti din registrul de deplasare (adr locala)

Particularitati Blitzen

- PE-urile lucreaza la nivel de bit
- Registrul de Shiftare/Deplasare este bidirectional
- Blitzen-ul are memorie on-chip de 1k (1024 b)
- Este posibila mascarea separata a diverselor tipuri de operatii cu memoria sau cu SR (de deplasare)
- Exista un control local al adresei de memorie (Local Address Modifier)
- Bitul K de la Blitzen permite o executie a datelor de tip MIMD
- Masina Blitzen in anumite aplicatii este mult mai performanta ca MPP

Exemple

- Mai multe instructiuni elementare distincte se pot executa in acelasi timp:
  - ADD + preluare date din memorie + preluare de pe magistrala de date
- Exemple de microinstructiuni:
  - SET_C // C ← 1
  - ADD // A + P + C → B, C
  - MOV_BD // pune B pe magistrala de date D
  - MOV_MD(ADDR) // pune de la ADDR pe magistrala D
  - MOV_DP // pune in P continutul de pe D
  - ROUTE_E // trimite P catre E si incarca in P din W

Adunarea a doua Numere (8b)

```
#include "blitzen.h"     /*contine setul de instructiuni */
#define XADDR   100
#define YADDR   200
#define WADDR   300
#define NUMBITS 8
main (){
// suma W = X + Y;
// valori cum ar fi "X0" fac referinta la bitul 0 din X
set_route(GRID);
  load_file("in1.ism",XADDR,XADDR,8);   //citesc X si Y
  load_file("in2.ism",YADDR,YADDR,8);
   CLR_C;                  // clear C,
for (int i=0; i< NUMBITS; i++){
        MOV_MD(XADDR+i); //* A <- X(i), data de la addresa XADDR pe mag da date D
        MOV_DA;           // pune in A continutul de pe D
   END;
        MOV_MD(YADDR+i); //* P <- Y(i)
        MOV_DP;
   END;
        ADD;             //* adunare, rezultat in B
   END;
        MOV_BD;          //* W(i) <- B
        MOV_DM(WADDR+i);
   END;

     }

     save_file("sum.osm",WADDR,WADDR,8);  //salvare rezultat
     zyg_end();
}
```

Categorii de Algoritmi

- Algoritmii ce se preteaza la utilizarea masinii Blitzen fac parte din urmatoarele categorii
- 1. Embarrassingly Parallel Algorithms (algoritmi “rusinosi” paraleli)
  - Fiecare PE actioneaza independent (paralelism maxim);
  - Exemplu: Valoarea de prag (apartenenta la domeniu)
- 2. Near Embarrassingly Parallel Algorithms
  - Un PE are nevoie de informatii pe care le va primi de la alte PE
  - Exemplu: tratarea imaginilor (filtre etc); detectia contururilor

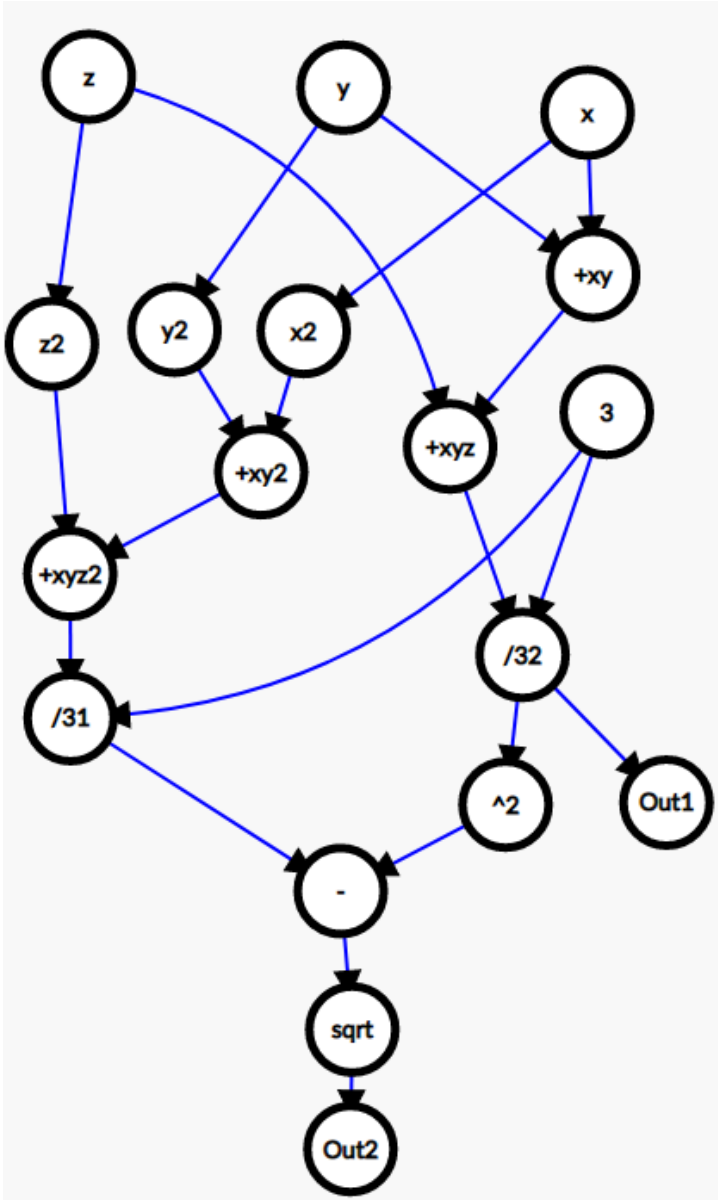
Regula de activare:

- O instrucțiune este activată (adică executabilă) dacă sunt disponibili toți operanzii.
- Observați că în modelul von Neumann, o instrucțiune este activată dacă este indicată de PC.
- Regula de calcul sau regula de declanșare, specifică când se execută efectiv o instrucțiune activată.
- O instrucțiune este declanșată (adică executată) când devine activată.
- Efectul declanșării unei instrucțiuni este prelucrarea datelor sale de intrare (operanzi) și generarea datelor de ieșire (rezultate).

Exemplu

- function Med_Dev:
- Calculeaza media si deviatia standard a trei variabile

```
function Med_Dev(x,y,z: real returns
   real,real);
 let
  Out1 := (x + y + z)/3;
  Out2 := SQRT((x**2 + y**2 + z**2)/3 - Out1**2);
 in
  Out1, Out2
 endlet
endfun
```



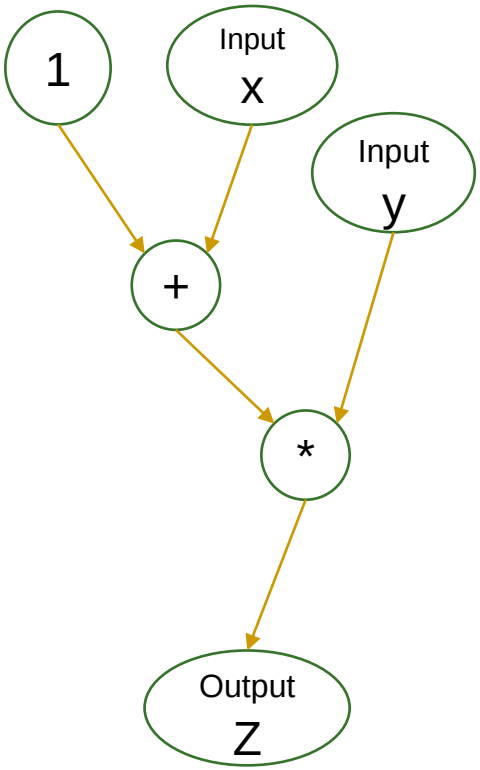
Modelul Data Flow

- In Model bazat pe flux de date (data flow):
- execuția unei instructiuni de prelucarea este efectuata doar de disponibilitatea operandului!
- Nu dispune de PC
- lipsesc cele două caracteristici ale modelului von Neumann care devin blocaje în exploatarea paralelismului

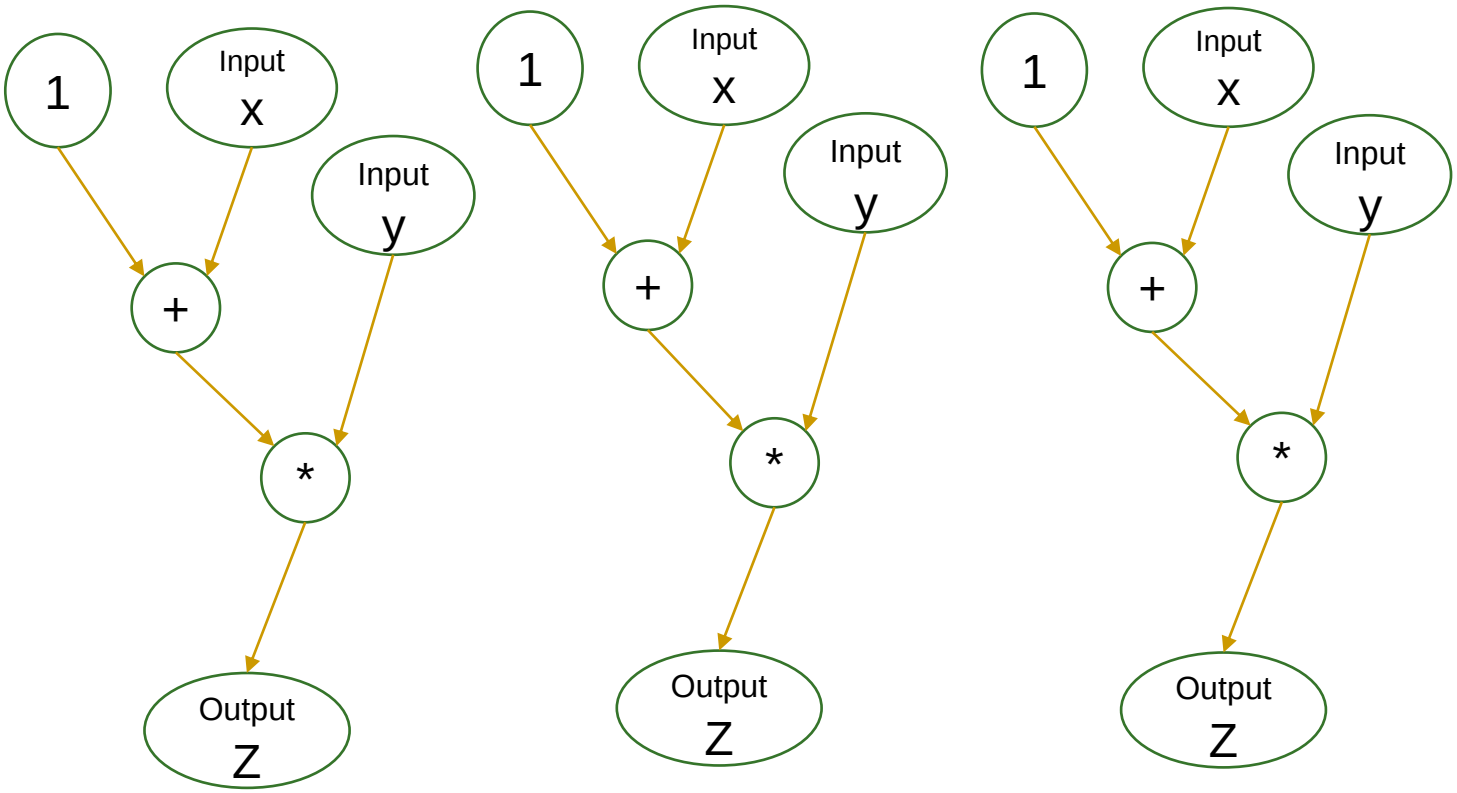
Exemplu

- Un program de flux de date este compilat într-un graf de flux de date care este un graf direcționat format din noduri, care reprezintă instrucțiuni și arce, care reprezintă dependențe de date între instrucțiuni.
- Grafiul fluxului de date este similar cu un graf de dependență utilizat în reprezentările intermediare ale compilatoarelor.
- În timpul executării programului, datele se propagă de-a lungul arcurilor în pachete de date, numite jetoane (token-uri).
- Acest flux de jetoane permite unele dintre noduri (instrucțiuni) sa inceapa executia și le declanșează.

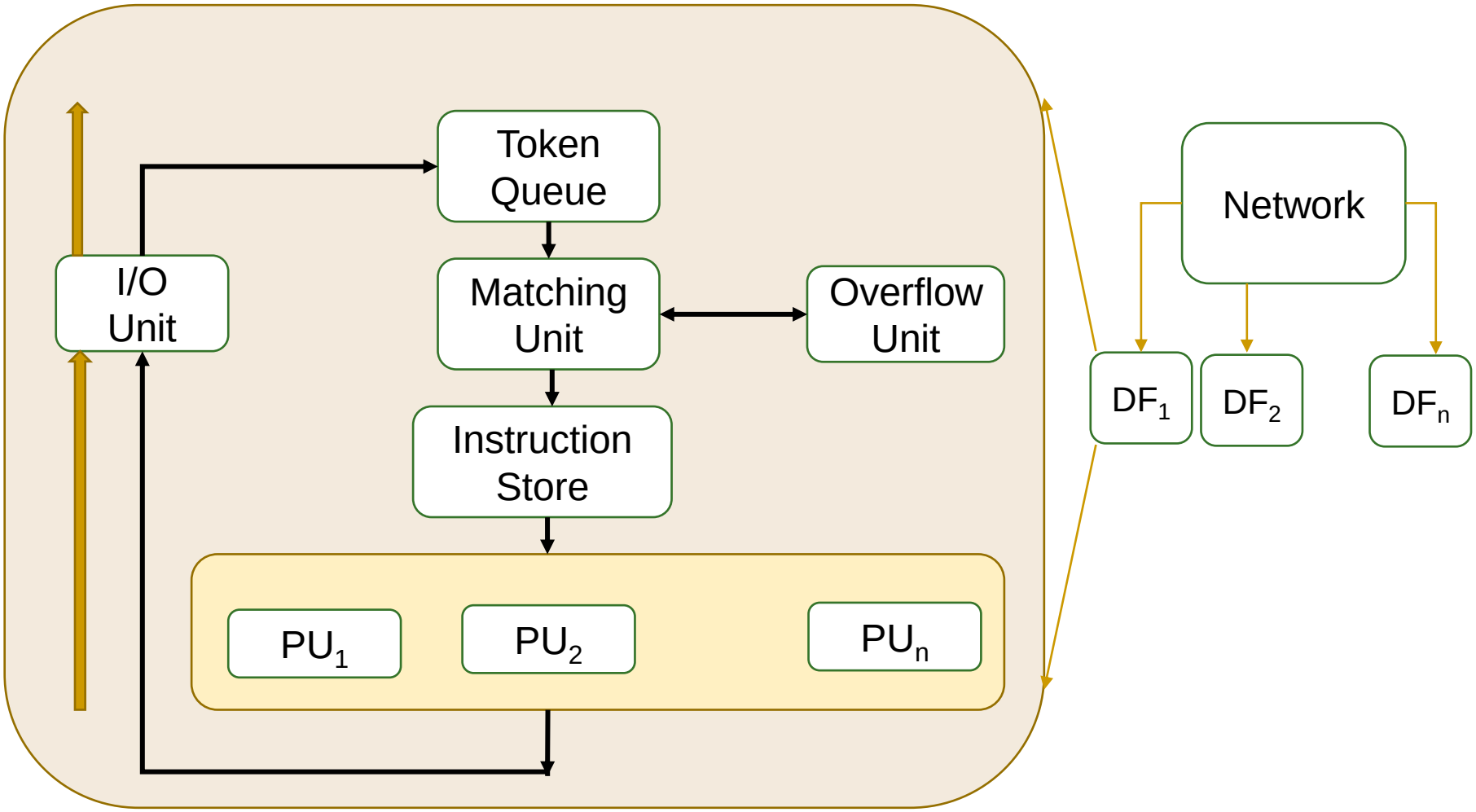
Z=(x+1)*y



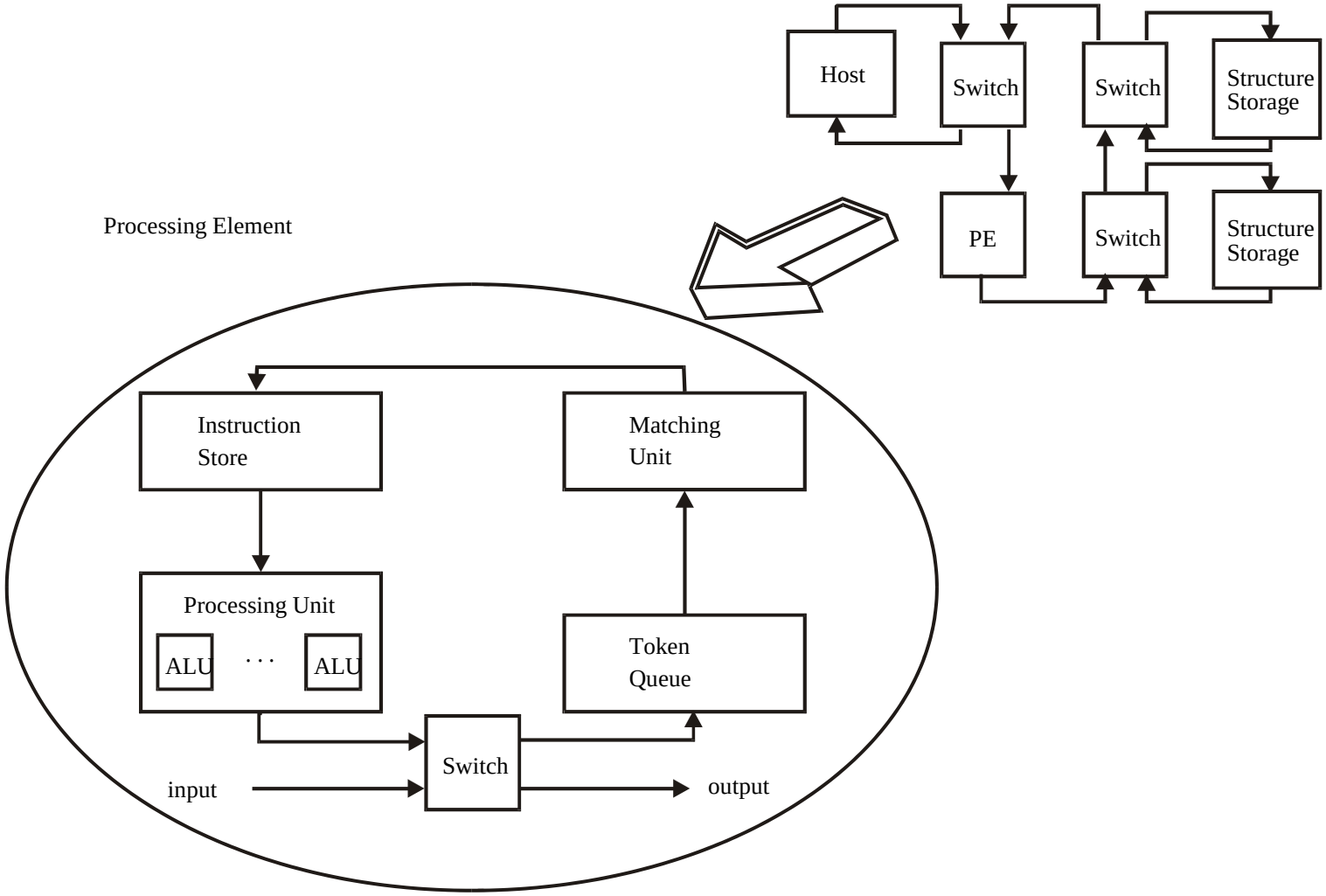
Executie


1
Input
x
Input
y
+
*
Output
Z
1
Input
x
Input
y
+
*
Output
Z
1
Input
x
Input
y
+
*
Output
Z

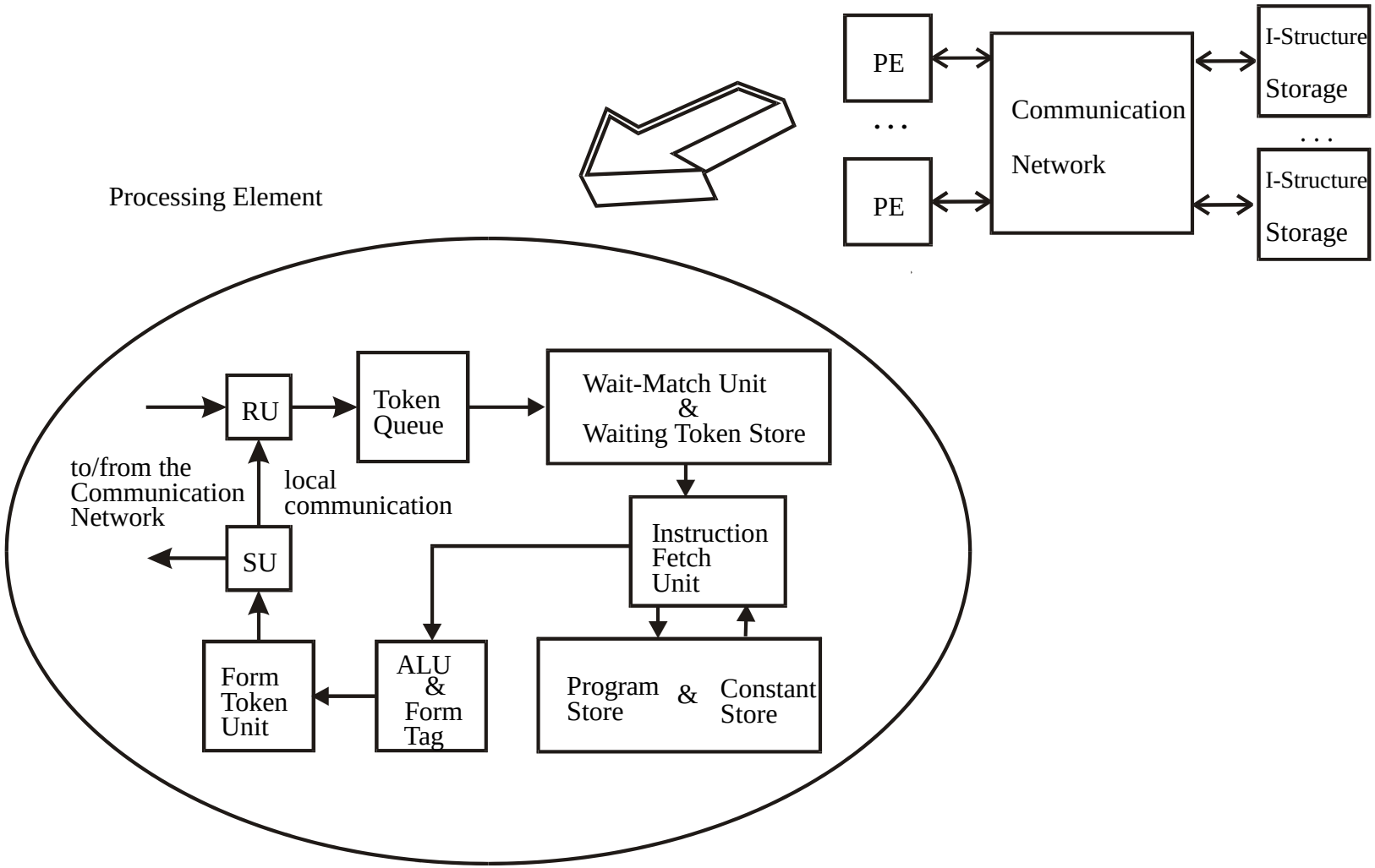
Manchester Dataflow Computer



Manchester Dataflow Machine



MIT Tagged-Token Dataflow Architecture



Caracteristici importante ale grafului asociat fluxului de date

- Funcționalitate:
  - *Evaluarea unui graf de flux de date este echivalentă cu evaluarea funcției matematice corespunzătoare.*
- Compozibilitate:
  - *Grafurile fluxului de date pot fi combinate pentru a forma grafuri noi.*

Dataflow Static

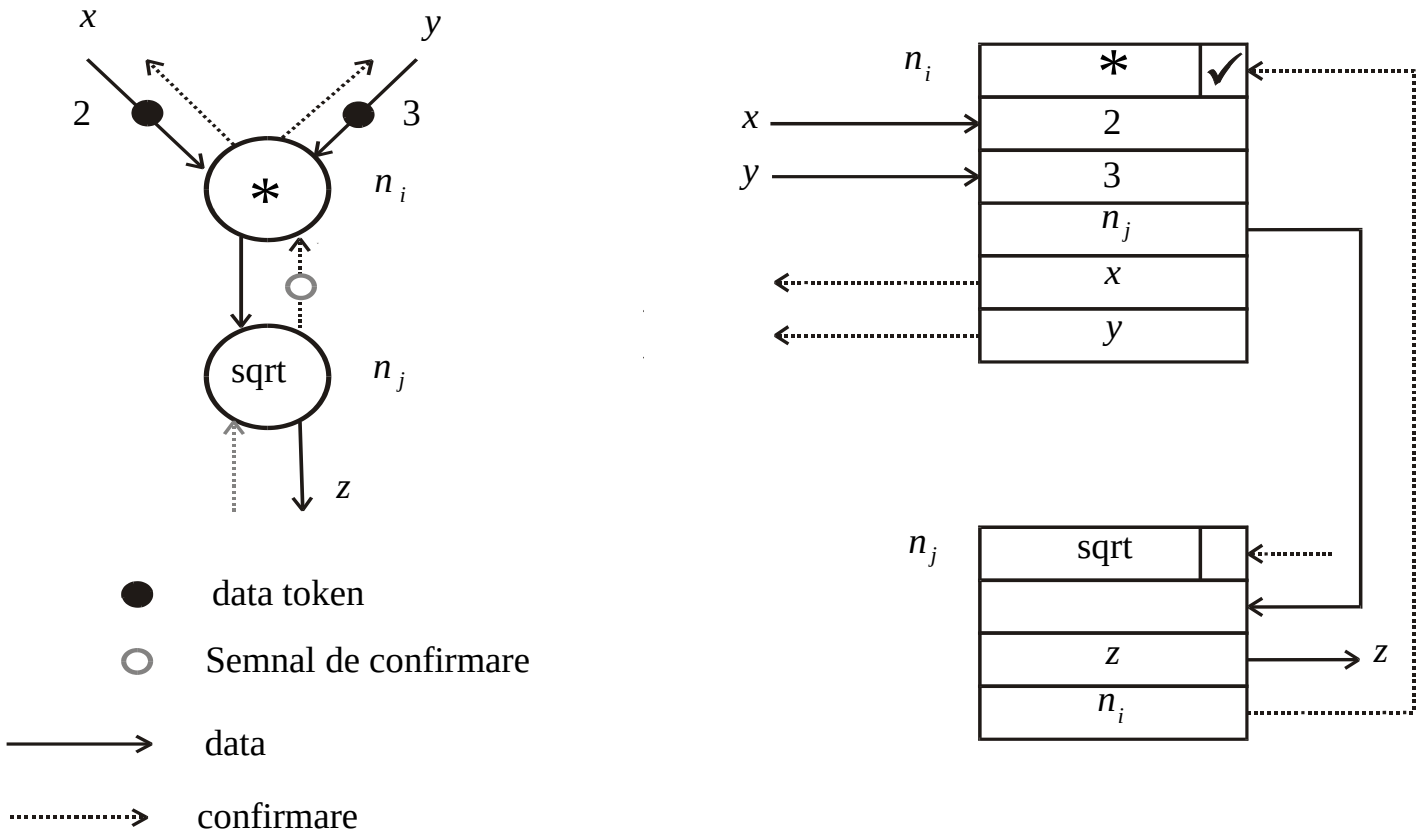
- Un graf de flux de date este reprezentat ca o colecție de șabloane de activitate, fiecare conținând:
  - codul instrucțiunii reprezentate,
  - sloturi de operand pentru păstrarea valorilor de operand,
  - câmpurile de adresă de destinație, referindu-se la sloturile de operand din șabloanele de activitate din secvente care trebuie să primească valoarea rezultatului.

Deficiente dataflow static:

- Iterațiile consecutive ale unei bucle pot fi doar executate sub forma de pipeline.
- Datorită token-urilor (jetoanelor) de confirmare, traficul de token-uri este dublat.
- Lipsa suportului de programare care este esențial pentru limbajul de programare modern
  - fără apeluri de procedură,
  - fără recursivitate.

Avantaj: model simplu de implementat

Dataflow : Template-ul de activitate



Semale de confirmare

- Abordarea statică a fluxului de date permite cel mult un simbol pe orice arc.
- Nu pot fi token-uri (jetoane) diferite destinate aceleiași destinații.
- Extinderea regulii de bază pentru declansare se face astfel:
  - Un nod activat este declanșat dacă nu există nici un indicativ pe oricare dintre arcurile sale de ieșire.
- Implementarea restricției prin semnale de confirmare (jetoane suplimentare), transmise de-a lungul arcurilor suplimentare de la nodurile consumatoare catre cele producătoare.
- Regula de declansare poate fi modificată la forma sa originală:
  - Un nod este declanșat în momentul în care devine activat.

- *Atentie: restrictiile structurale sunt ignorate presupunând resurse nelimitate!*

Dataflow Dinamic

- Fiecare iterație de buclă sau invocare de subprogram ar trebui să poată executa în paralel ca o instanță separată a unui subgraf reentrant.
- Replicarea este doar conceptuală.
- Fiecare *token* (jeton) are un *tag (o etichetă):*
  - adresa instrucțiunii pentru care este destinată valoarea particulară a datelor
  - informații de context
- Fiecare arc poate fi văzut ca o *multime* care poate conține un număr arbitrar de Token-uri cu tag-uri diferite.
- Regula de activare și declanșare este:
  - Un nod este activat și declanșat de îndată ce jetoanele cu etichete identice sunt prezente pe toate arcele de intrare.

- Restrictiile structurale sunt ignorate!

Interpretare de tip U

- Fiecare *token* constă dintr-un nume de activitate și date
  - numele activității cuprinde tag-ul.
- Tag (eticheta) are:
  - o adresă de instrucțiune **n**
  - câmpul contextual **c** care identifică în mod unic contextul în care urmează să fie invocată instrucțiunea,
  - numărul de inițiere **i** care identifică iterația de buclă în care are loc această activitate.
- De mentionat faptul ca **c** este el însuși un nume de activitate.
- Deoarece instrucțiunea de destinație poate necesita mai multe intrări, fiecare token poartă și numărul portului său de *destinație* ***p***.

- Token-ul se poate reprezenta prin $\langle c.i.n, data \rangle_p$

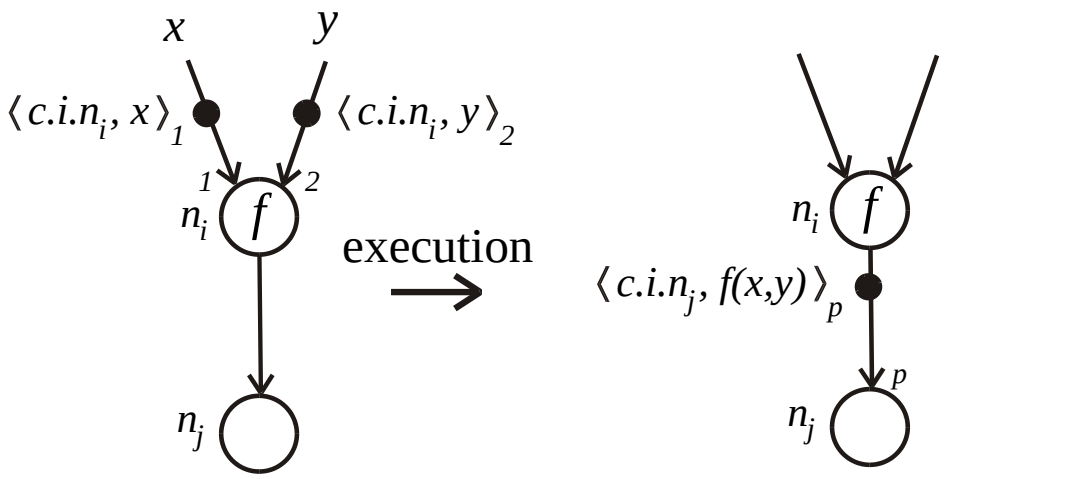
< c-context . i-iteratia . n-adresa , data >

Interpretare de tip U

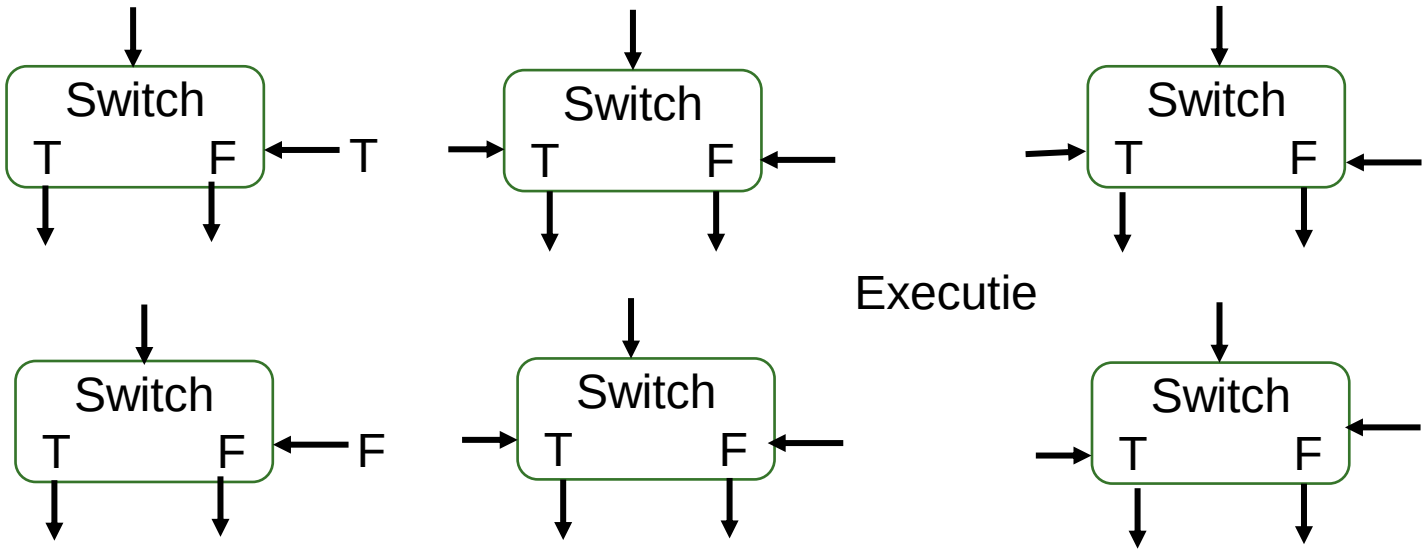
$\langle c.i.n, data \rangle_p$ *< c-context . i-iteratia . n-adresa , data >*

- dacă nodul n_i îndeplinește o funcție f și
- dacă portul p al lui n_j este destinația lui n_i,
- atunci avem

$$in: \{\langle c.i.n_i, x\rangle_1, \langle c.i.n_i, y\rangle_2\} \qquad out: \{\langle c.i.n_j, f(x,y)\rangle_p\}$$



Implementare decizie

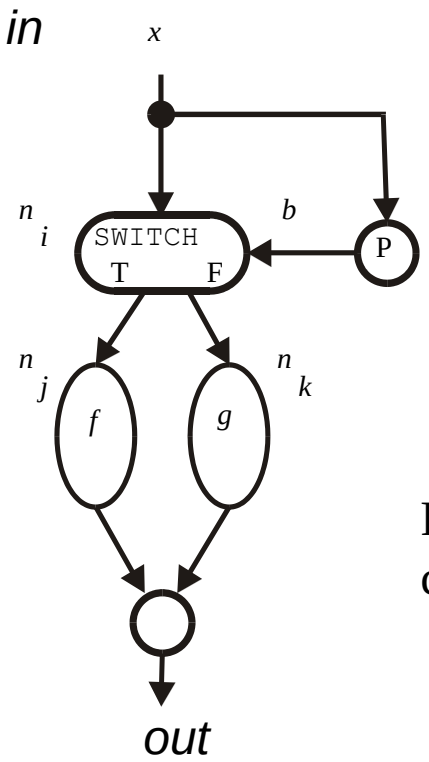


T/F sunt complementare
0/1 sau 1/0

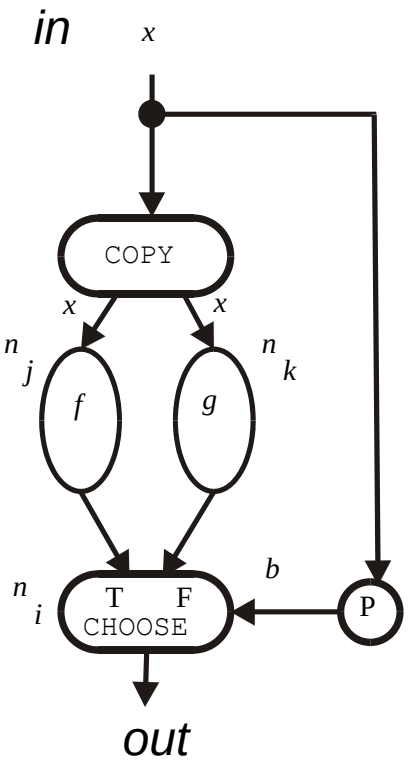
Implementare ramificatie (decizie)

$\langle c.i.n, data \rangle_p$ *< c-context . i-iteratia . n-adresa , data >*

$$in : \{\langle c.i.n_i, x\rangle_{data}; \langle c.i.n_i; b\rangle_{control}\} \qquad out : \begin{cases} \{\langle c.i.n_j, x\rangle\} if & b = T \\ \{\langle c.i.n_k, x\rangle\} if & b = F \end{cases}$$



Ramificatie clasica



Evaluare speculativa a ramificatiei

L, L^{-1}, D, and D^{-1} Operatori pentru impementarea Buclelor

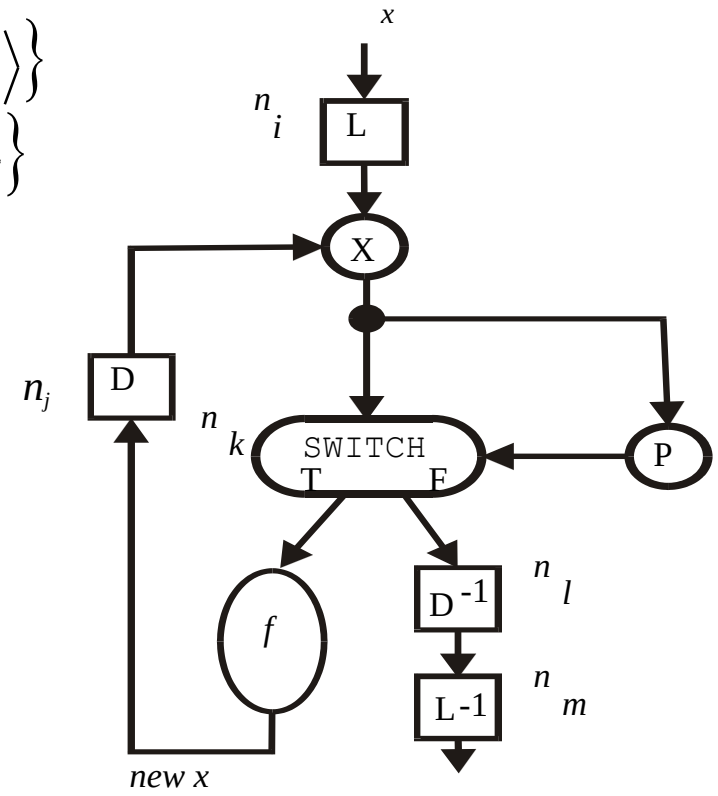
$\langle c.i.n, data \rangle_p$ *< c-context . i-iteratia . n-adresa , data >*

$L: \quad in: \{\langle c.i.n_i, x \rangle\}$ $\qquad out: \{\langle c'.1.n_k, x \rangle\}, \quad unde \quad c' = \langle c.i.n_i \rangle$

$D: \quad in: \{\langle c'.j.n_j, x \rangle\}$ $\qquad out: \{\langle c'.j+1.n_k, x \rangle\}$

$D^{-1}: \quad in: \{\langle c'.k.n_l, x \rangle\}$ $\qquad out: \{\langle c'.1.n_m, x \rangle\}$

$L^{-1}: \quad in: \{\langle c'.1.n_m, x \rangle\}$ $\qquad out: \{\langle c'.i.n_n, x \rangle\}$



- L: Initalizare, context bucla
- D: incrementeaza contor bucla
- D^{-1}: initializeaza contor la 1
- L^{-1}: reface contextul original

Nota: **c** este el însuși un nume de activitate

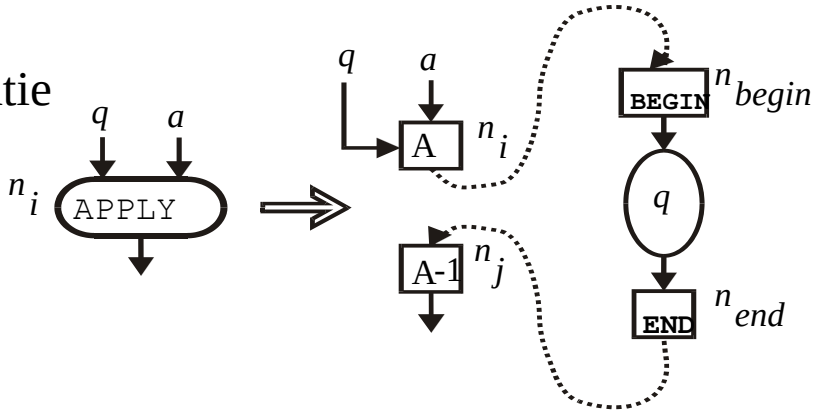
A, A^{-1}, BEGIN, and END Operatori pentru functii

- A: $in : \{\langle c.i.n_i, q\rangle_{func}, \langle c.i.n_i, a\rangle_{\arg}\}$ $out : \{\langle c'.1.n_{begin}, a\rangle\}$

  unde $c' = \langle c.i.n_j \rangle$ si n_j este adresa operatorului A^{-1}

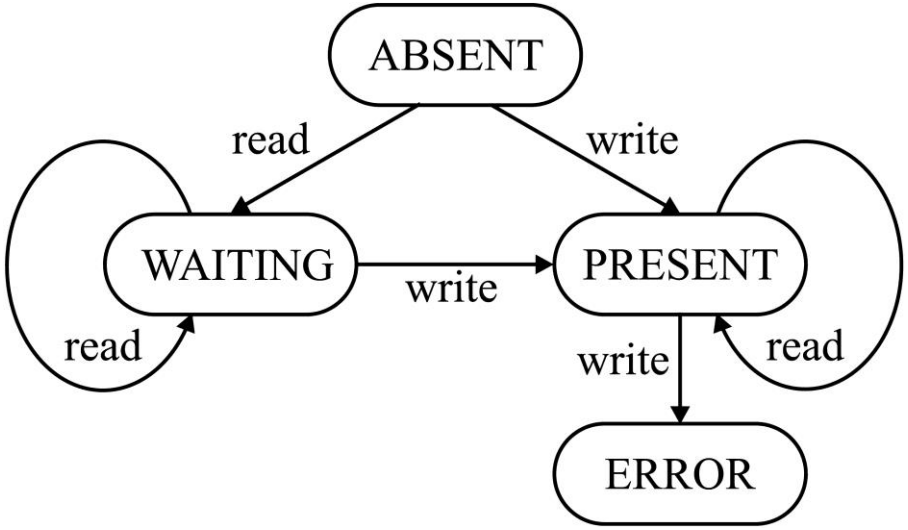
- END: $in : \{\langle c'.i.n_{end}, q(a)\rangle\}$ $out : \{\langle c.i.n_j, q(a)\rangle\}$

- A: creaza un nou context
- BEGIN: replica token-ul pentru fiecare ramificatie
- END: returneaza rezultatul
- A^{-1}: replica iesirea pentru succesori



Structura I

- Statutul fiecărui element al structurii I poate fi:
  - ***prezent****:* elementul poate fi citit, dar nu scris,
  - ***absent***: o cerere de citire trebuie amânată, dar este permisă o operație de scriere în acest element,
  - ***așteptare***: cel puțin o cerere de citire a elementului a fost amânată.



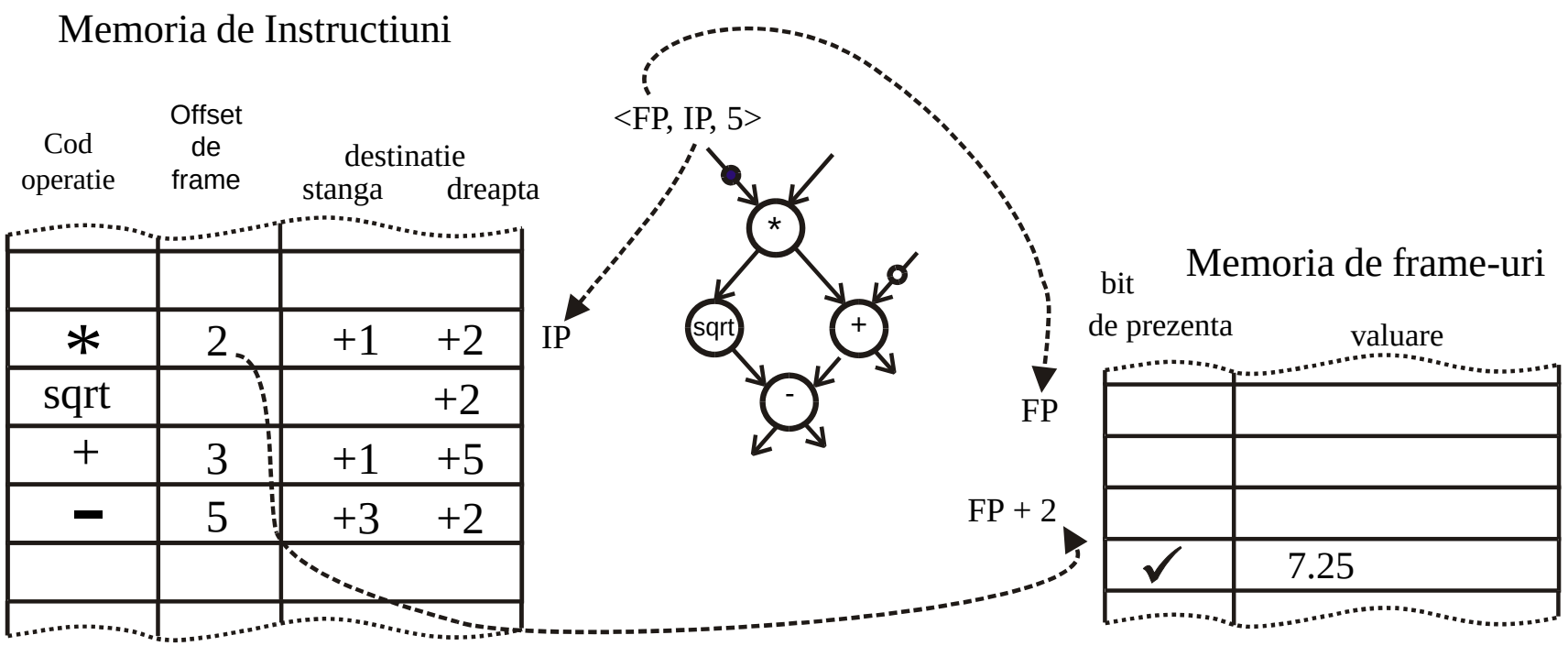
structura I

- Sunt definite trei operații elementare pe structurile I:
  - ***alocare:*** rezervă un număr specificat de elemente pentru o nouă structură I,
  - ***I-fetch:*** recuperează conținutul elementului structura - I specificat (dacă elementul nu a fost încă scris, atunci această operațiune este amânată automat),
  - ***I-store:*** scrie o valoare în elementul structura - I specificat (dacă acel element nu este gol, se raportează o condiție de eroare).
- Aceste operații elementare sunt utilizate pentru a construi noduri SELECT și ASSIGN.
- ***I-fetch*** este implementată ca operație de memorie în fază divizată:
  - o cerere de citire emisă unei structuri I este independentă în timp de răspunsul primit și, prin urmare, nu provoacă o așteptare de către PE emitent.

Implementare cu token explicit

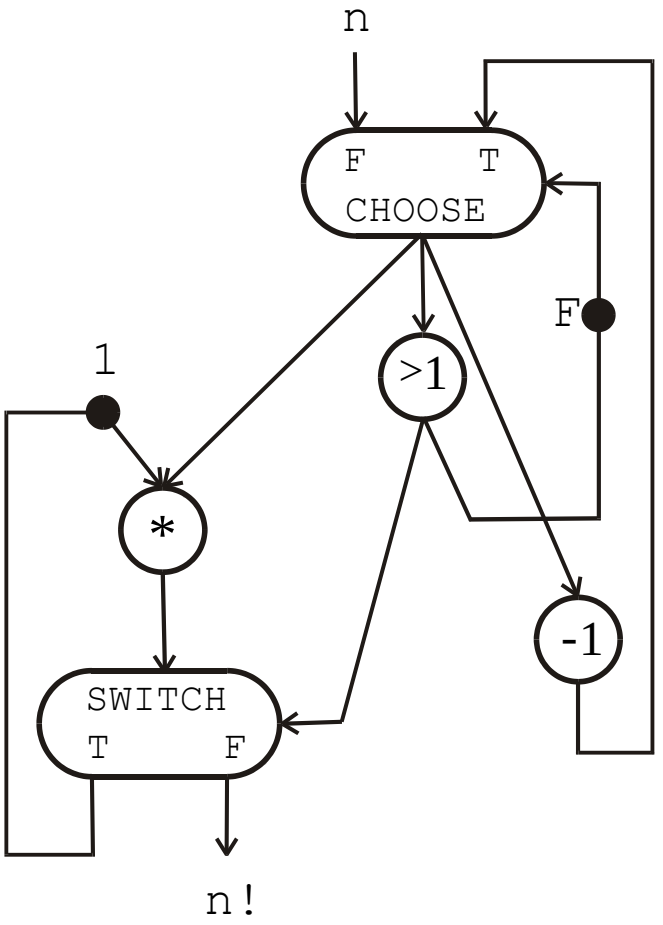
- **Scop**: implementarea eficientă a potrivirii token-urilor.
- **Ideea de bază**: se aloca un frame separat în memoria de frame-uri pentru fiecare iterație de buclă activă sau invocare de subprogram.
- Un ***frame*** este format din sloturi în care fiecare slot deține un operand care este utilizat în activitatea corespunzătoare.
- Deoarece accesul la sloturi este direct nu este necesară nici o căutare asociativă.

Explicit Token



Exemplu calcul factorial n!

```
initial j = n; k = 1
 while j > 1 do
   j = j - 1;
   k = k * j;
return k
```



Exemplu de prelucrare vectori

Input d,e,f

//a,b,c,d,e,f

//vectori de date

$C_0=0$

For i=1 to n

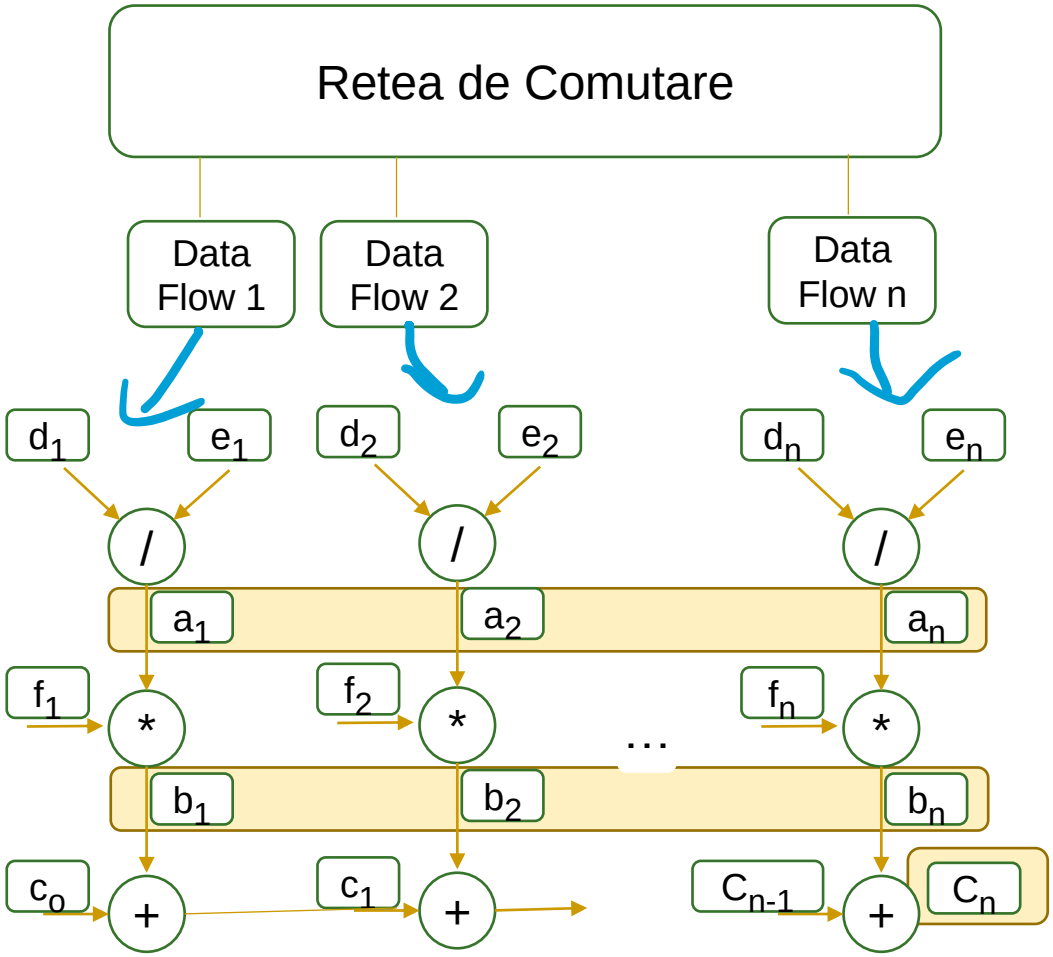
$a_i=d_i / e_i$

$b_i=a_i*f_i$

$c_i=b_i+c_{i-1}$

End for

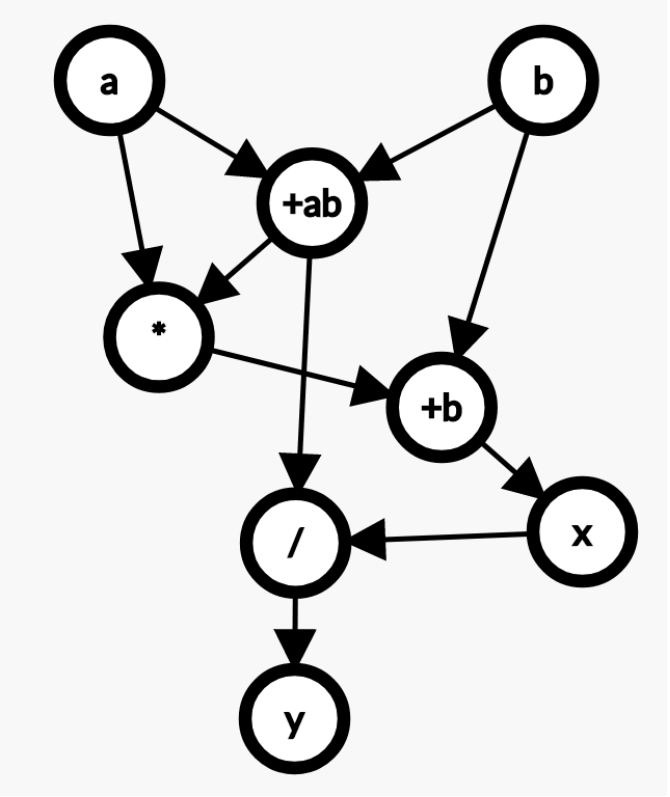
Output a,b,c_n



Ordinea instructiunilor irelevanta

Exemplu:

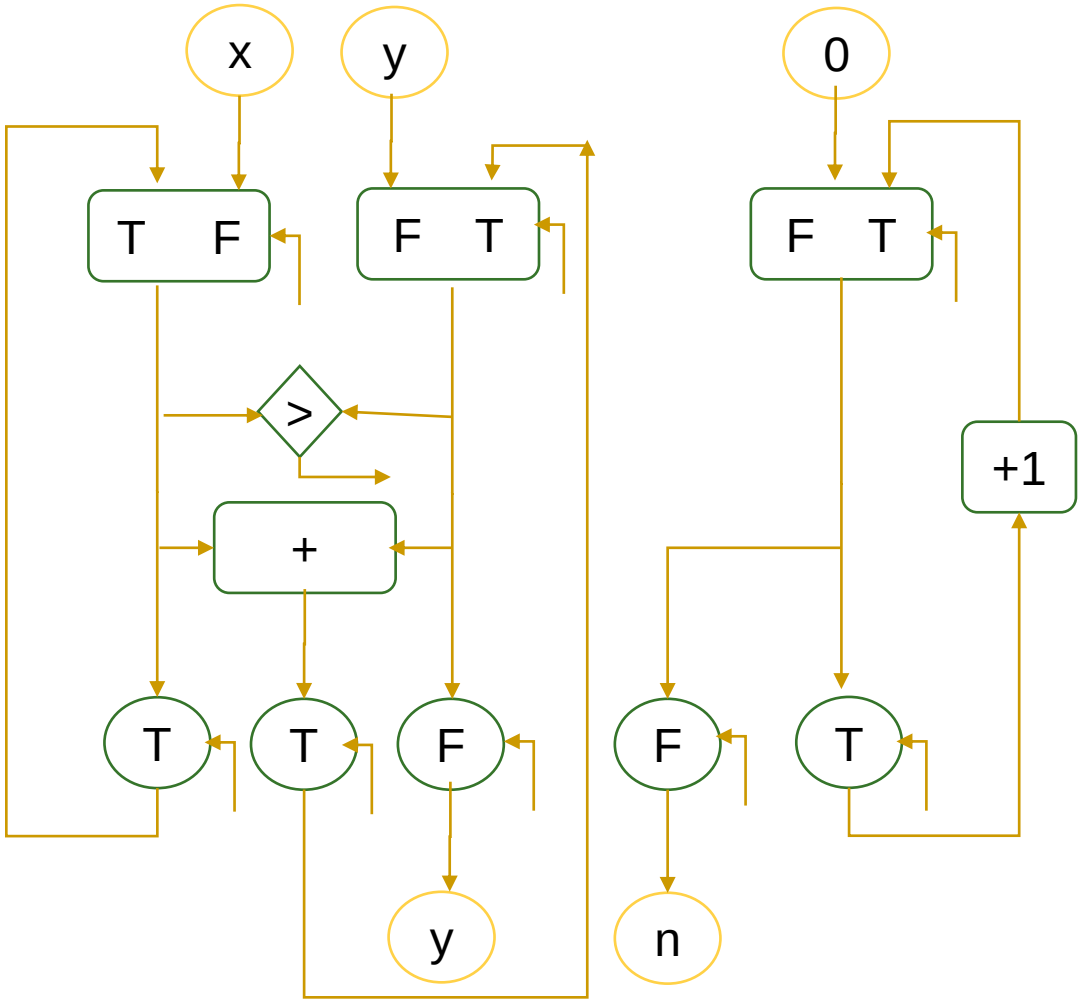
```
input a,b
  y = (a+b)/x
  x = (a*(a+b))+b
output y,x
```



Exemplu de bucla

```
input y,x
 n = 0
 while y<x do
     y =  y + x
     n =  n + 1
 end
output y,n
```

Nu s-a folosit array



Caracteristici de baza

- Executia instructiunilor este bazata pe disponibilitatea datelor
- Datele sunt pastrate in cadrul instructiunii
- Disponibilitatea datelor este verificata de o unitate specializata
- Tokenul associat datelor este verificat de o unitate specializata
- Executia instructiunilor este asincrona

Avantaje/ Dezavantaje

Avantaje

- Potential ridicat de parallelism
- Viteza crescuta de executie
- Usor de integrat comunicatia si sincronizarea

Dezavantaje

- Overhead-ul mare pentru control
- Manipularea dificila a structurilor de date

Unde se foloseste data flow in microprocesoarele actuale

Cea mai recentă generație de microprocesoare superscalare afișează o execuție dinamică în afara ordinii, denumită flux de date local sau flux de date micro.
Colwell și Steck 1995, în prima lucrare despre PentiumPro:
Fluxul instrucțiunilor in Arhitectura Intel este prevăzut cu mecanismul prin care instrucțiunile sunt decodificate în micro-operații (μops), sau serie de μops, iar aceste μops sunt plasate într-un set de microperatii executabile speculative, care se vor executa în afara ordinii de operații (out of order),
Vor fi executate pe principiul data flow (când operanzii sunt gata) și utilizate în ordinea programului sursă.

Fiecare instrucțiune este gata să fie executată imediat ce toți operanzii sunt disponibili.

Exemplu care evidentiaza diferite structuri elementare

Consideram adunarea a doi vectori

$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ $Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

Z= X+Y

$z_i = x_i + y_i$

Fiecare componenta are mantisa si exponent (caracteristica)

$x_i = x_{ic}\ x_{im}$

Operatia de adunare are 4 operatii elementare organizate in pipeline

- Compararea caracteristicilor
- Deplasarea mantisei (daca este cazul)
- Adunarea mantiselor
- Normalizarea rezultatului
- Aceste operatii sunt independente – structura pipeline
- Consideram aceste operatii dureaza ~t (pentru simplificare)