Arhitectura Calculatoarelor

# Curs 2: Sinteza circuitelor digitale

**E-mail: florin.oniga@cs.utcluj.ro**

**Web: http://users.utcluj.ro/~onigaf, secțiunea Teaching/AC**

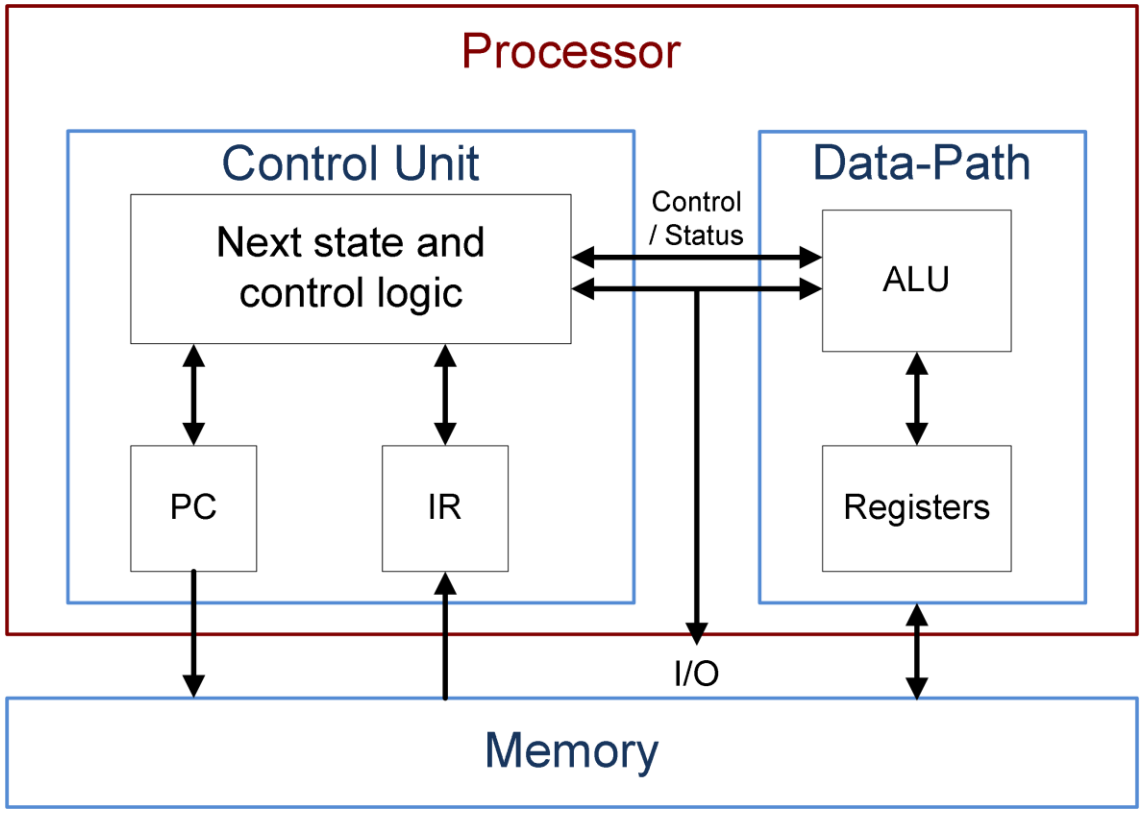
# Principalele subiecte

- RTL – Register Transfer Level
- Sinteza de nivel înalt pentru arhitecturi digitale (High-Level Synthesis - HLS)

# RTL – Register Transfer Level

**Sistem digital format din:**

- Unitatea căilor de date (Data-Path) – registre, magistrale, logică de procesare
- Unitatea de comandă/control (Control Unit) – determină secvența de operații de procesare a datelor în calea de date



# RTL – Register Transfer Level

- Un sistem se descrie la nivel RTL prin transferul informației între elementele de memorare ale sistemului

Operație RT: rdest ← f(rsrc1, rsrc2, … rsrcn)

- **RTL Abstract** → o specificație comportamentală; nu se ia în considerare structura sistemului, nici caracteristicile de timp ale resurselor
- **RTL Fizic sau concret** → descrie implementarea specificației comportamentale pe structura selectată, cu granularitatea semnalului de ceas, ținând seama de constrângeri de resurse și de timp
- **Metoda de proiectare RTL** (definiție) - convertirea unei descrieri comportamentale (RTL abstract) într-o descriere structurală (RTL fizic / concret).

# RTL – Register Transfer Level

- Specificarea RTL concret include următoarele trei componente pentru un sistem digital:
  1. Setul de registre / memorii
  2. Unitățile funcționale, care execută operaţiile necesare (interconectare cu registre / memorii)
  3. Unitatea de control care supervizează secvenţa de operaţii ale sistemului
- RTL reprezintă o notație algebrică folosită pentru a defini operații la nivelul calculatorului
- RTL **nu** este executată de calculator → este folosită pentru a explica funcționarea calculatorului
- **Micro-operație** - descrie un transfer unic între registre:

  Ex: X ← Y + Z
- **Propoziție (statement) RTL** - (condiție) → {micro-operație,…, micro-operație};

  Ex: (c>0) → X ← Y + Z

# RTL – Register Transfer Level

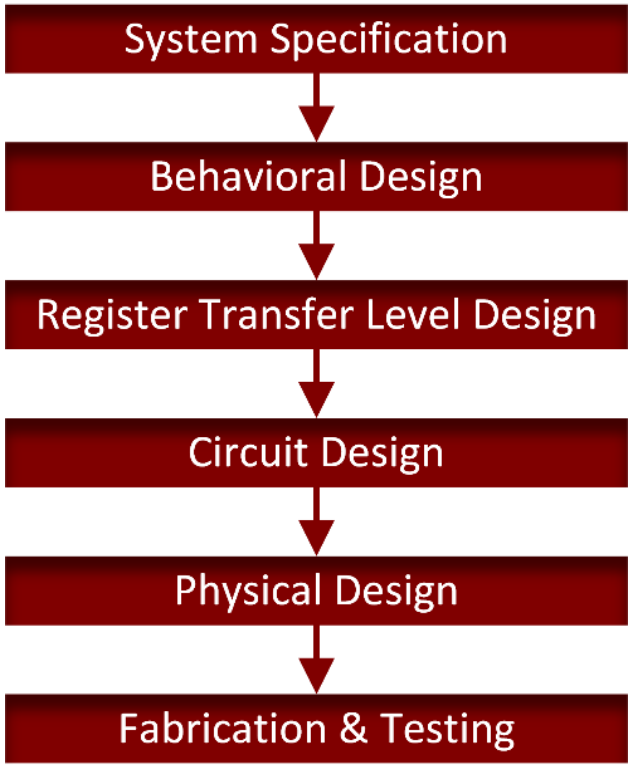
Register Transfer Notation (RTN) (o variantă posibilă, notația nu este standard)

| | |
|---|---|
| ← | Atribuire |
| =, ≠ | Test pentru egalitate / inegalitate |
| \|\| | Concatenare pentru șiruri de biți |
| X ← Y | Transfer de date din registrul Y în registrul X |
| X ← 0 | Ștergere registrul X |
| X ← Y + Z | Adună registrul Y cu registrul Z, încarcă rezultatul în X |
| X ←Y v Z | SAU între Y și Z, încarcă rezultatul în registrul X |
| DR ← M[AR] | Încarcă în DR conținutul memorie de la adresa dată de AR |
| R1 ← >> R1 | Deplasare logică la dreapta cu o poziție pentru registrul R1 |
| R2 ← << R1 | Identic, dar la stânga |
| X ← Y, A ← B | Transfer paralel (concurent) |
| (cond) → A ← B | Dacă cond = 1 atunci copiază B în A |
| S0 → A← B | Dacă starea curentă e S0, copiază B în A |
| P·(a·b) → R2 ← R3 | Dacă starea curentă este P, și dacă a ȘI b este adevărată atunci copiază R3 în R2; unde a și b sunt semnale |

# Sinteza de nivel înalt pentru arhitecturi digitale - High-Level Synthesis (HLS)

**HLS este o metodologie care:**

- de la descriere comportamentală la nivel algoritmic, **trece la**
- descriere structurală similară cu RTL concret: registre, unități funcționale (+, -, *), memorii, interconexiuni (mux, magistrale),
- **respectând** constrângerile (timp, suprafață, putere).




*Ca o comparaţie* (!): sinteza VHDL porneşte de la descrierea RTL, foloseşte tehnici de sinteză logică pentru optimizarea circuitului, şi generează descrierea la nivel de "porţi şi fire – gates and wires" mapabilă pe circuitul digital configurabil (ex. FPGA).

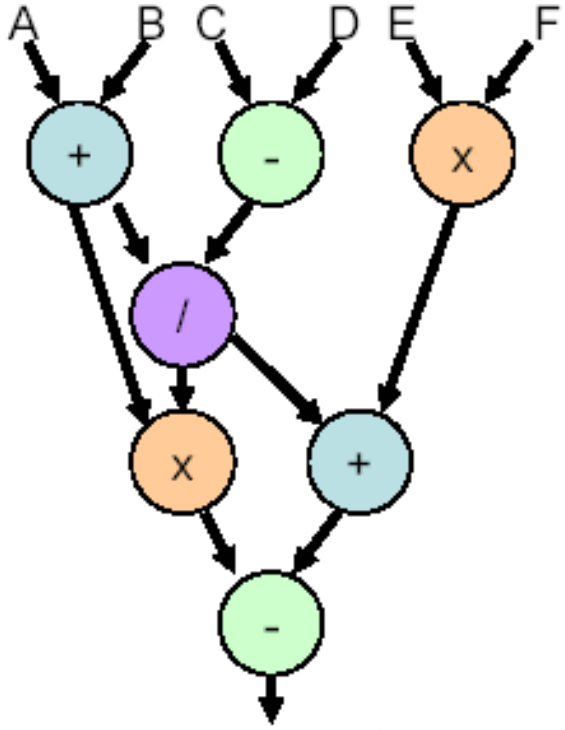
# HLS – Operațiile de bază

- **Alocarea resurselor** — selectarea tipului și numărului componentelor hardware care vor fi utilizate
- **Planificare** (Scheduling) — Atribuirea operațiilor la perioadele semnalului de ceas
- **Legarea modulelor** (Binding) — Legarea (maparea) operațiilor la componentele hardware alocate
- **Sinteza unității de comandă** — Se stabilește tipul unității de comandă și modul de folosire a semnalului de ceas

# Model de calcul orientat pe flux de date

**Grafuri de flux de date / Data-flow graphs (DFGs)** (or Data-Dependency graphs DDG) reprezintă relațiile de precedență și paralelismele în calcule. DFG conține:

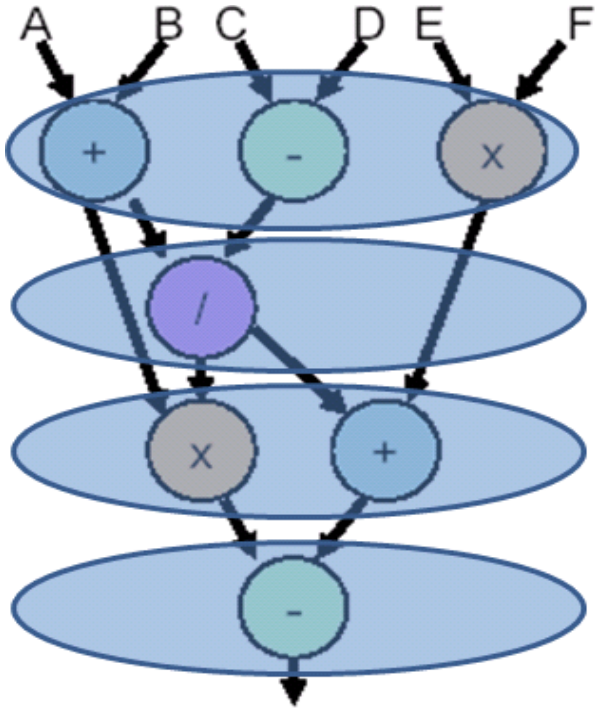
- **Noduri**: reprezentând operații
- **Muchii, Săgeți:** reprezentând relații de precedență.




| Calcul Secvențial |
|---|
| tmp_1 = A+B |
| tmp_2 = C–D |
| tmp_3 = E*F |
| tmp_4=tmp_1/tmp_2 |
| tmp_5=tmp_1*tmp_4 |
| tmp_6=tmp_4+tmp_3 |
| Result=tmp_5–tmp_6 |

**DFG** – exemplu cod secvențial

# Model de calcul orientat pe flux de date




| Calcul Concurent |
|---|
| tmp_1=A+B, tmp_2=C–D, tmp_3=E*F |
| tmp_4=tmp_1/tmp_2 |
| tmp_5=tmp_1*tmp_4, tmp_6=tmp_4+tmp_3 |
| Result=tmp_5–tmp_6 |
| |
| |
| |

**DFG** – exemplu cod concurent (paralel)

- Fiecare nod consuma date de la intrări, efectuează transformări și livrează date de ieșire
- Toate nodurile pot fi executate concurent, dacă (!) fluxul de date / resursele o permit

# Atribuire unică

Pregătirea codului programului pentru graful de flux de date, analiză și conversie la forma de atribuire unică, cu 2 intrări și o ieșire pentru fiecare operație (fiecare rezultat de operație se atribuie unei variabile unice, ex. x devine x1, x2, x3):

| ORIGINAL | | Atribuire unică |
|---|---|---|
| x ← a + b; | → | x1 ← a + b; |
| y ← a * c; | → | y ← a * c; |
| z ← x + d; | → | z ← x1 + d; |
| x ← y – d; | → | x2 ← y – d; |
| x ← x + c; | → | x3 ← x2 + c; |

# Compilare / Tehnici de Optimizare

Este o optimizare comportamentală, fără a avea nevoie de cunoștințe despre implementarea la nivel de circuit.

## Reducerea înălţimii arborelui

x = a + b + c + d;  se poate rescrie ca:

x = a + b;  x = x + c; x = x + d; → trei adunări în serie.

sau  p = a + b; q = c + d; x = p + q; → primele două adunări pot fi efectuate în paralel

## Propagarea constantelor şi a variabilelor

constante

a = 0; b = a + 1; c = 2*b; înlocuit de

a = 0; b = 1; c = 2;

variabile

a = x; b = a + 1; c = 2*a; înlocuit de

~~a = x~~; b = x + 1; c = 2*x;

unde a = x poate fi eliminat ca fiind cod “mort”

## Eliminarea sub-expresiilor comune

a = x + y; b = a + 1; c = x + y; înlocuit de

a = x + y; b = a + 1; c = a;

## Reducere complexităţii operator

b=3*x;  înlocuit cu

t = x <<1; b = x + t;

## Rearanjarea codului

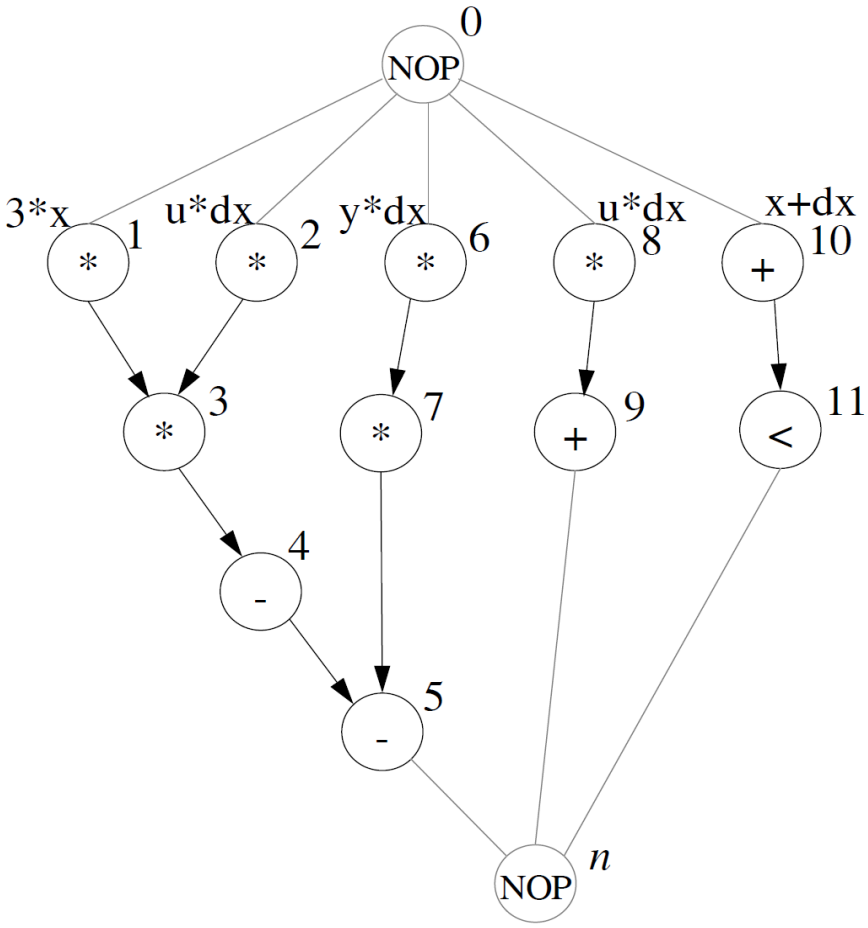
for (i = 1; i <= a*b){ }  înlocuit de

t = a*b; for (i = 1; i<= t) { }.

# HLS - Exemplul 1

Să considerăm următorul program (rezolvarea unei ecuații diferențiale, [1]):

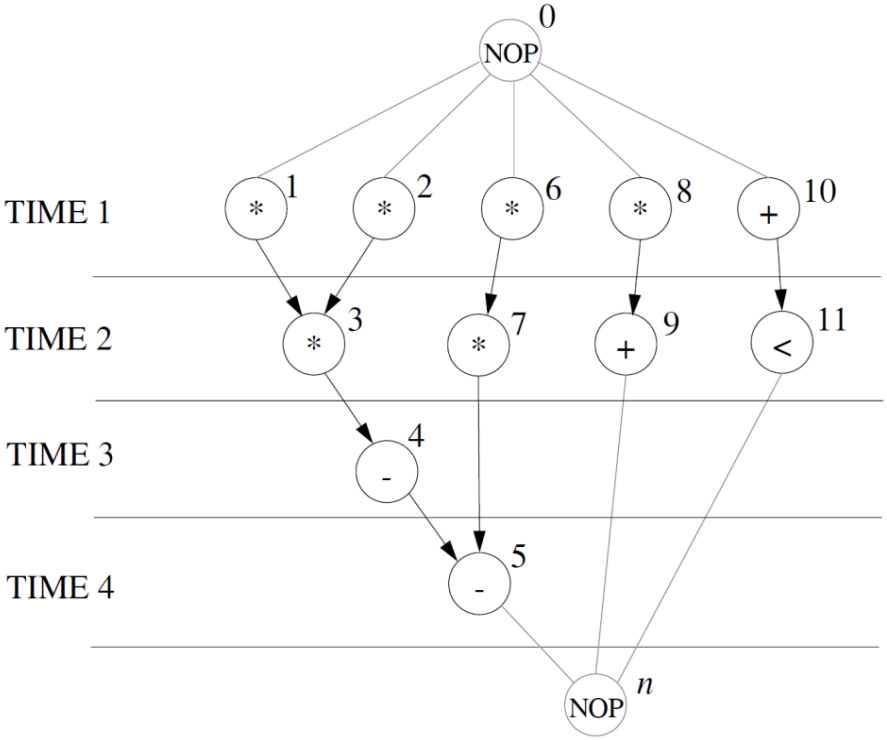
```
repeat
        xl = x + dx;
        ul = u - (3 * x * u * dx) - (3 * y * dx);
        yl = y + u * dx;
        c = xl < a;
        x = xl; u = ul; y = yl;
until (c);
```



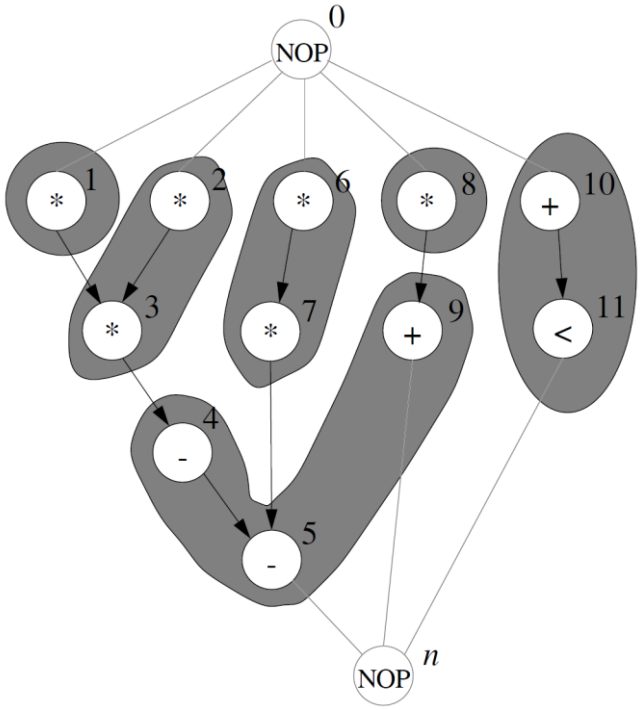

## Alocare pe noduri în DFG

xl = x + dx v10
ul = u - (3 * x * u * dx) - (3 * y * dx) v1- v7
yl = y+ u*dx v8, v9
c = x<a v11

# Planificare și atribuirea operațiilor




Graf de secvențiere pentru Exemplul 1;
necesita maxim 6 Înmulțitoare și 5 ALU



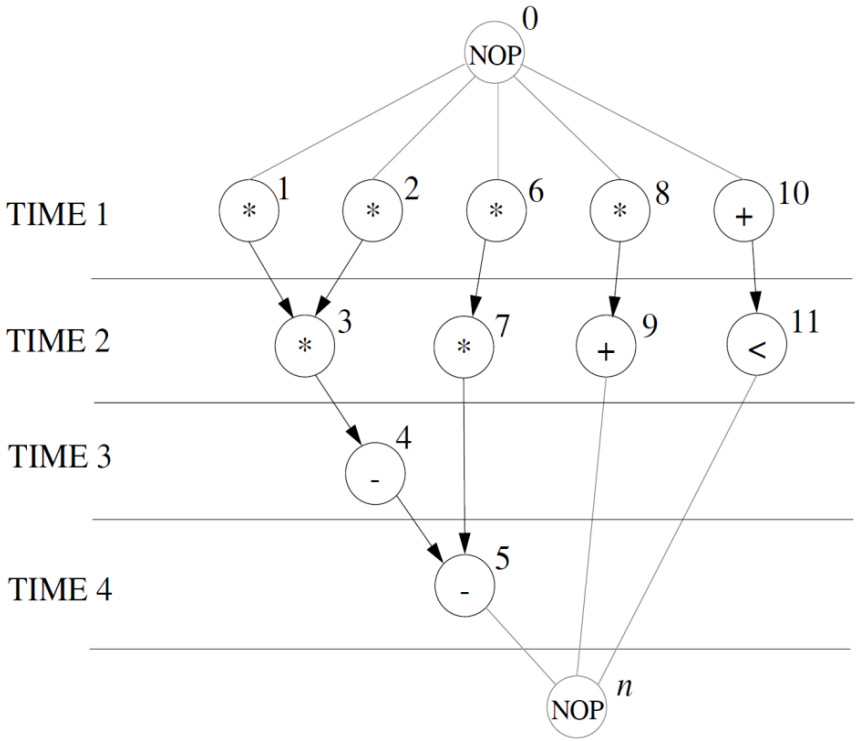

**Alocare la resurse**
minim 4 Înmulțitoare și 2 ALUs

- Operațiile se execută în segmentele de timp corespunzătoare cu pozițiile lor
- **Refolosirea Resurselor** (sharing sau partajare) => circuit secvențial
  - O componentă poate fi utilizată pentru mai multe operații
  - Operațiile corespunzătoare se pot planifica secvențial – nu concurent

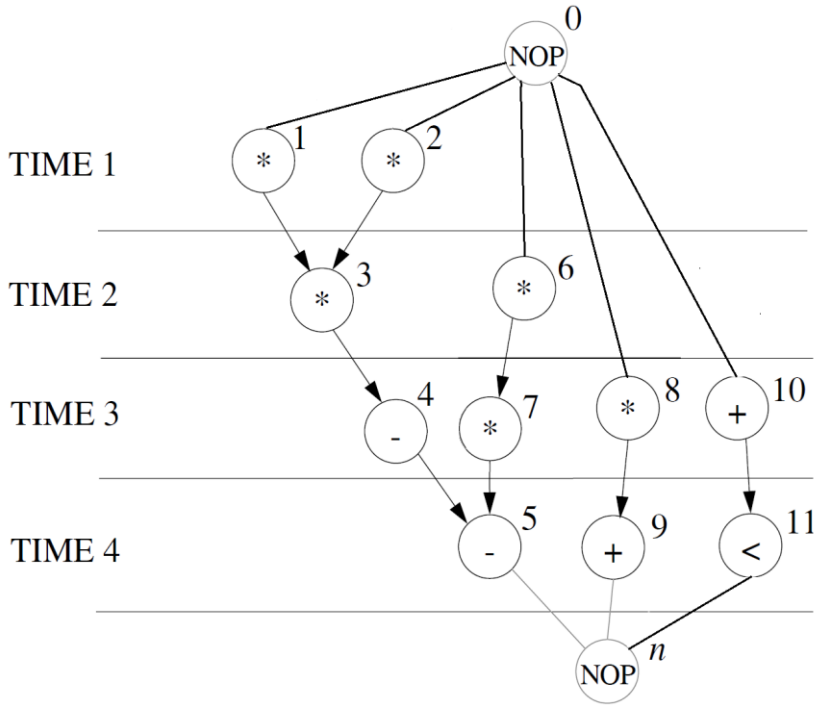
# Planificare fără constrângeri de resurse

**As Soon As Possible (ASAP):** o operație este planificată imediat după ce operațiile predecesoare au fost terminate, momentul de început al unei operații este cel mai devreme posibil

**As Late As Possible (ALAP):** se bazează pe folosirea valorilor maxime pentru momentele de început ale operațiilor, momentul de început este cel mai târziu




Exemplul 1: Planificare **ASAP**




Exemplul 1; Planificare **ALAP** cu același număr de pași

# Mobilitatea

**Mobilitatea** definește intervalul de start pentru o operație

Cum se calculează:

1. Se planifică ASAP și ALAP.
2. Diferența în planificare determină mobilitatea sau libertatea operației (operațiile în calea critică, cea mai lungă, sunt cele cu mobilitate zero).

La ce folosește? Pentru o planificare eficientă.

**Mobilitate zero** pentru o operație înseamnă că operația poate fi începută doar la momentul dat de timp pentru a satisface constrângerile globale de latenţă (calea critică)

**Exemplul 1:**

- mobilitatea operaţiilor (1 – 5) este zero,
- (6 şi 7) este 1,
- (8, 9, 10 şi 11) este 2.

# Sinteza și optimizarea căilor de date

**Modele hardware pentru HLS**: Unităţi funcţionale (ALU, sumatoare, înmulțitoare, etc.), Registre, Bloc/Fişier de registre (register file), Memorii, MUX-uri, Magistrale (BUS), Bufere tri-state

**Parametrii care definesc modelul hardware**:

- *Strategia de temporizare:* ceasul folosit (front crescător, etc.).
- *Interconectare:* bazată pe MUX şi/sau BUS
- *Sincronizarea (temporizarea) unităților funcționale:*
  - Cu un singur ciclu.
  - Cu operaţii multi-ciclu.
  - Pipelining (linie de asamblare).

# Sinteza căilor de date / optimizări

## Sinteza căilor de date

- Generarea structurii căilor de date din DFG secvențiat
- Doi paşi: **alocare** (allocation) şi **legare** (binding)

Alocarea alege unitățile funcționale (UF) şi registrele din biblioteca de componente

- Se selectează componentele care îndeplinesc cel mai bine criteriile de proiectare

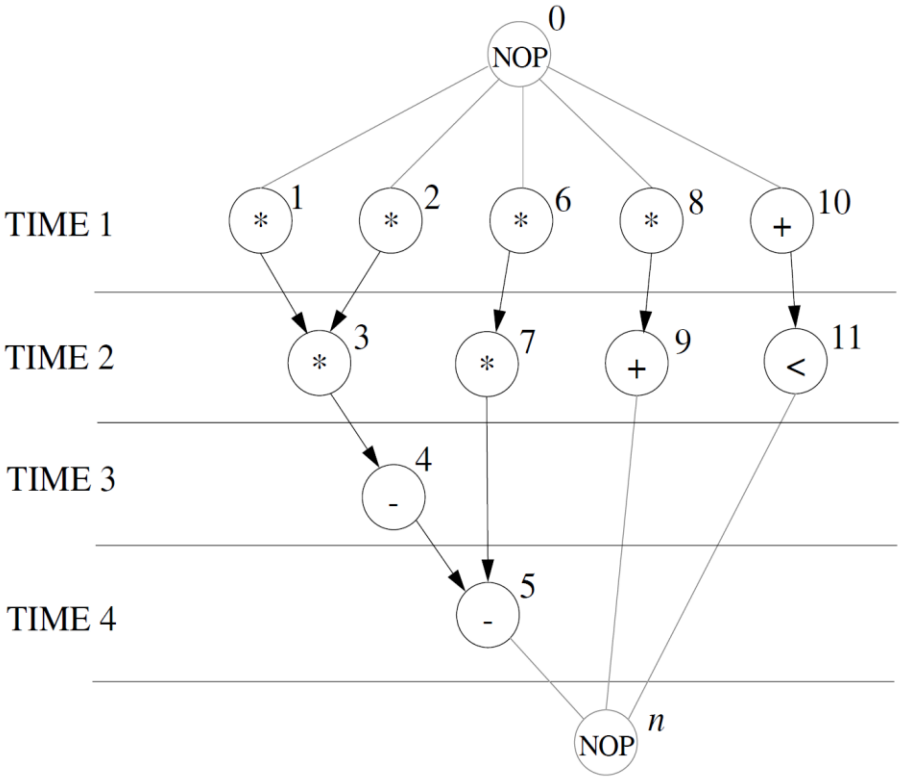
Legarea asociază operațiile cu UF-urile, variabilele cu registre, cu scopul de a conecta componentele astfel încât costul de interconectare (număr de MUX-uri, BUS-uri) să fie minimizat

Algoritmul de legare a registrelor (Register-binding): Timpul de viaţă al variabilelor este definit ca numărul de perioade de ceas în care variabila este necesară

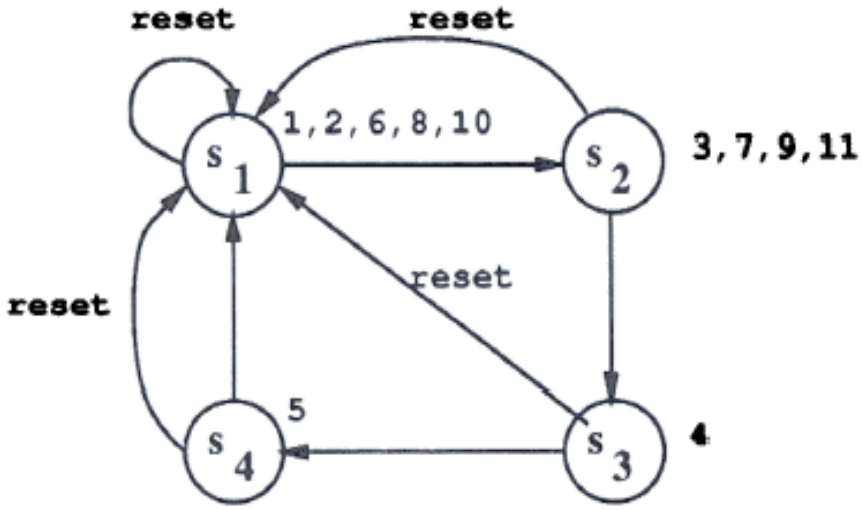
- Analizează timpul de viaţă al tuturor variabilelor
- Stabileşte numărul necesar de registre

# Sinteza unităţii de control

- Fluxul de control descris prin planificare este implementat folosind o mașină (automat) cu stări finite (FSM), cu o stare pentru fiecare pas
- Semnalele de control generate pentru fiecare stare activează unităţile funcţionale, registre, MUX-urile, selecţiile BUS-urilor, etc.




Exemplul 1: Planificare ASAP

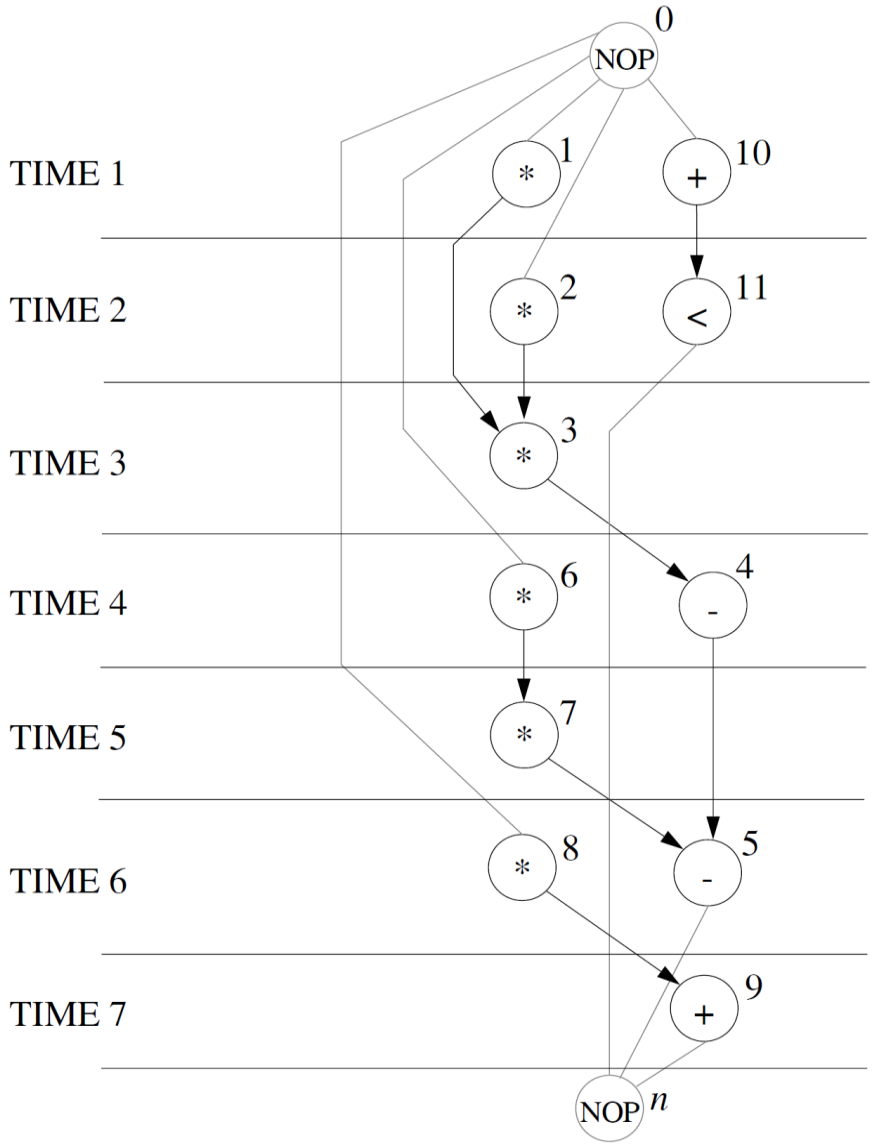



Control: diagrama tranziției stărilor FSM. Numerele de lângă noduri sunt referinţe la semnalele de activare (dacă sunt necesare)

# Proiectare cu constrângeri

**Exemplul 1**: proiectarea căilor de date şi a controlului considerând că avem la dispoziție doar:

**1 unitate de multiplicare şi 1 ALU**

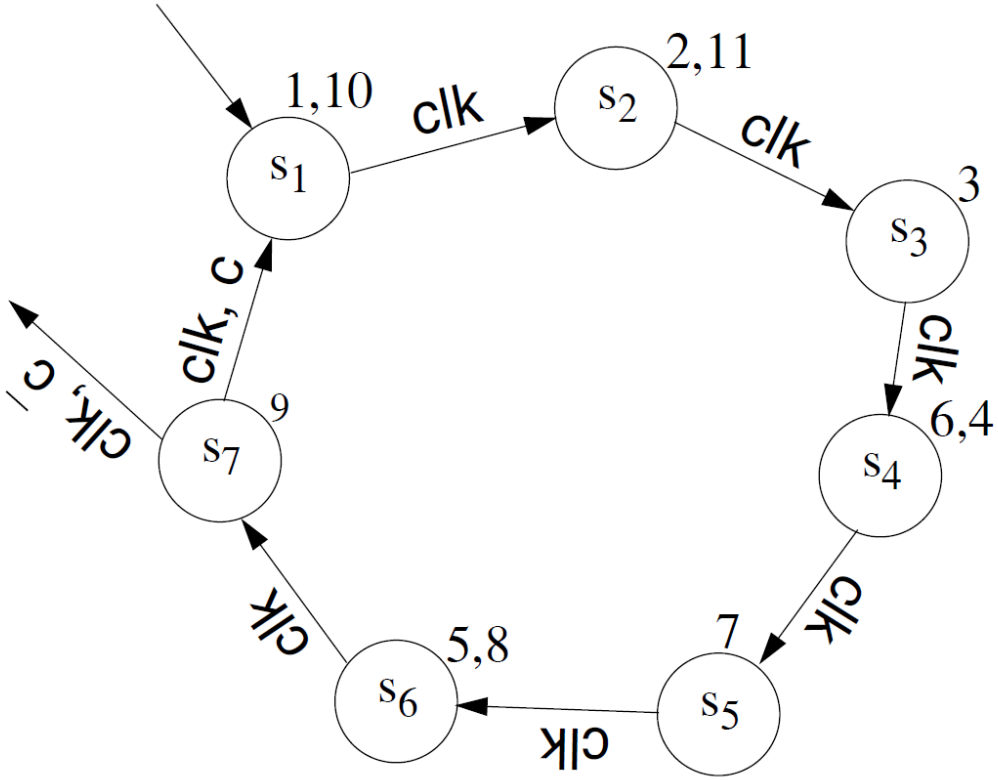



Exemplu 1: graf de secvenţiere cu 1 multiplicator, 1 ALU

# Proiectare cu constrângeri

**Unitatea de control (pentru exemplu anterior) - FSM**

- O stare în fiecare perioadă de ceas
- În fiecare stare se generează semnalele de control necesare pentru executarea respectiv stocarea rezultatelor pentru operațiile necesare




Exemplu 1: 1 multiplicator, 1 ALU; diagrama de tranziţie a stărilor

# Sinteza circuitelor de tip pipeline

- **Pipeline** (linie de asamblare) este o tehnică uzuală de îmbunătățire a performanței (**productivitatea**) unui circuit
- La o implementare pipeline **circuitul este partiţionat** într-un şir liniar de stagii / etaje, fiecare etaj execută concurent o operaţie pe un set distinct de date, şi furnizează rezultatele sale către următorul stagiu.
- **Între etaje consecutive** sunt necesare **registre cu rol de stocare** a rezultatelor din stagiul anterior, **respectiv cu rol de intrare** pentru stagiul următor

# Constrângeri de proiectare şi optimizare

- Timp → viteza; latență, lățime de bandă, frecvență de ceas;
- Spațiu → cost; logică combinațională multi-nivel, partajarea componentelor secvențiale, pipeline;
- Putere → viața bateriei, mod sleep;
- Testare → crash; BIST (Built In Self Test);

**Stiluri de proiectare:**

- Optimizarea latenţei (timpul de răspuns): un singur ciclu de ceas, cale de date cu logică combinaţională cu paralelism multilevel, fără constrângeri de resurse, şi frecvenţă de ceas redusă
- Optimizarea lăţimii de bandă (productivitate): cale de date pipeline, frecvenţă crescută de ceas
- Optimizarea spațială (dpdv a resurselor): partajare, reutilizarea componentelor, cale de date multi-ciclu
- Optimizarea spațială (dpdv a comunicației): căi de date bazate pe MUX-uri sau cu 1, 2, 3 magistrale, multi-ciclu.

# Constrângeri de proiectare şi optimizare

## Proiectare cu un singur ciclu (Single cycle design)

- **Pentru**: latenţă optimă.
- **Resurse**: unităţi funcţionale combinaţionale (UF), fără constrângeri de resurse.
- **Comunicaţie**: directă, punct la punct între UF-uri.
- **Frecvenţa de ceas**: definită de calea critică, cea mai lungă întârziere compusă a unităţilor UF interconectate serial.

## Proiectare pipeline

- **Pentru**: lăţime de bandă optimă.
- **Resurse**: UF-uri combinaţionale, registre între etaje (inter stage registers), anumite constrângeri de resurse
- **Comunicaţie**: prin intermediul registrelor pipeline inter-stagii. Unităţile UF sunt izolate prin intermediul registrelor inter-stagii şi pot funcţiona în paralel pe faze diferite de execuţie.
- **Frecvenţa de ceas**: definită de cea mai înceată UF şi de încărcarea (overhead) introdusă de registre.

## Proiectare cu mai multe cicluri de ceas (Multi-ciclu)

- **Pentru**: frecvenţă mare de ceas cu constrângeri spaţiale (reutilizarea resurselor).
- **Resurse**: număr limitat de UF-uri, registre temporare.
- **Comunicaţie**: prin intermediul registrelor temporare, MUX-uri şi BUS-uri. Rezultatele intermediare al UF-urilor sunt stocate în registre pentru utilizare viitoare.
- **Frecvenţa de ceas**: definită de cea mai înceată UF şi de încărcarea (overhead) introdusă de registre.

# Problema 1

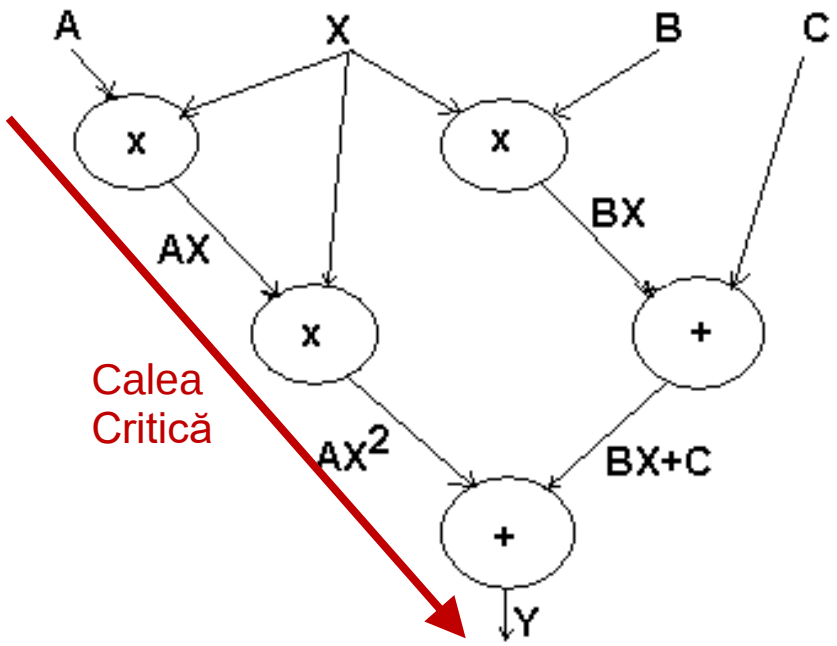
- Proiectarea unei UF specifice pentru o aplicație, folosind componente standard.
- Se dă $Y = AX^2 + BX + C$; Să se proiecteze (cale de date plus control) un circuit care să calculeze Y. Se va folosi metoda HLS.

# Soluția 1 - Proiect cu ciclu unic, V1

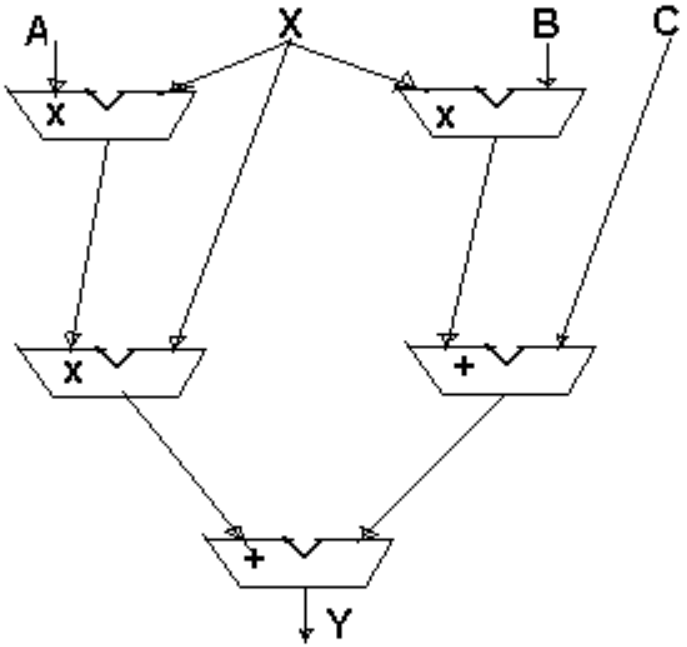
Rescriem ecuaţia, operaţii cu 2 intrări și 1 ieşire

- A·X, B·X, (A·X)·X, (B·X)+C, ((A·X)·X + (B·X)+C

DFG corespunzător, arată dependentele si precedentele pentru operații.




DFG

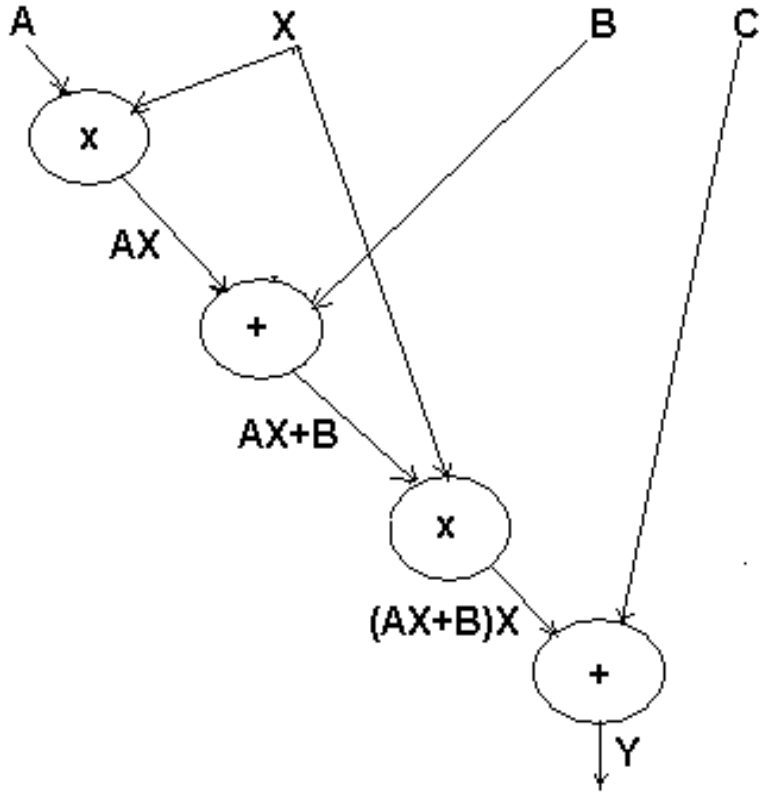



Calea de date

- **Resurse** pentru ciclu unic, V1: 3 Înmulţitoare si 2 Sumatoare.
- **Calea critică** (AX, $AX^2$, Y) (x, x, +) defineşte timpul de răspuns: (2·x + 1·+).
- **Execuţie controlată strict de fluxul de date**. UF si conexiunile definesc comportamentul circuitului. Schimbarea datelor schimba rezultatul operaţiilor.

# Soluția 2 - Proiect cu ciclu unic, V2

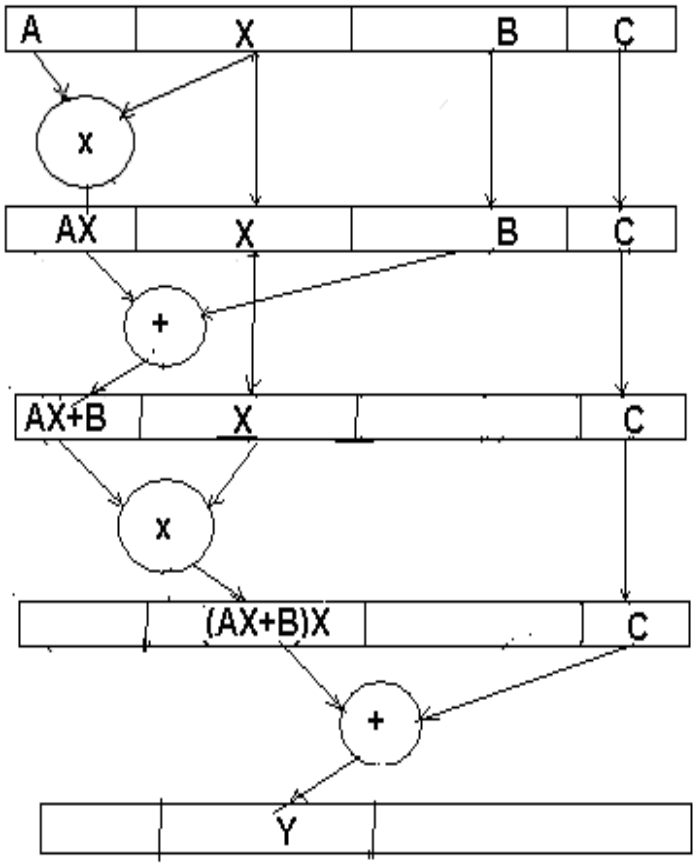
Rescriem ecuația Y = (A·X + B)·X + C




DFG Single cycle, V2

- **Resurse** pentru ciclu unic, V2: 2 Înmulțitoare și 2 Sumatoare.
- **Calea critică** definește timpul de răspuns: (2·x + 2·+).
- **2 soluții (V1 și V2)**: timpi de răspuns și resurse diferite.
- **Execuție controlată strict de fluxul de date**: UF si conexiunile definesc comportamentul circuitului. Schimbarea datelor schimba rezultatul operațiilor.

# Soluția 3 - Proiect Pipeline (bazat pe V2)



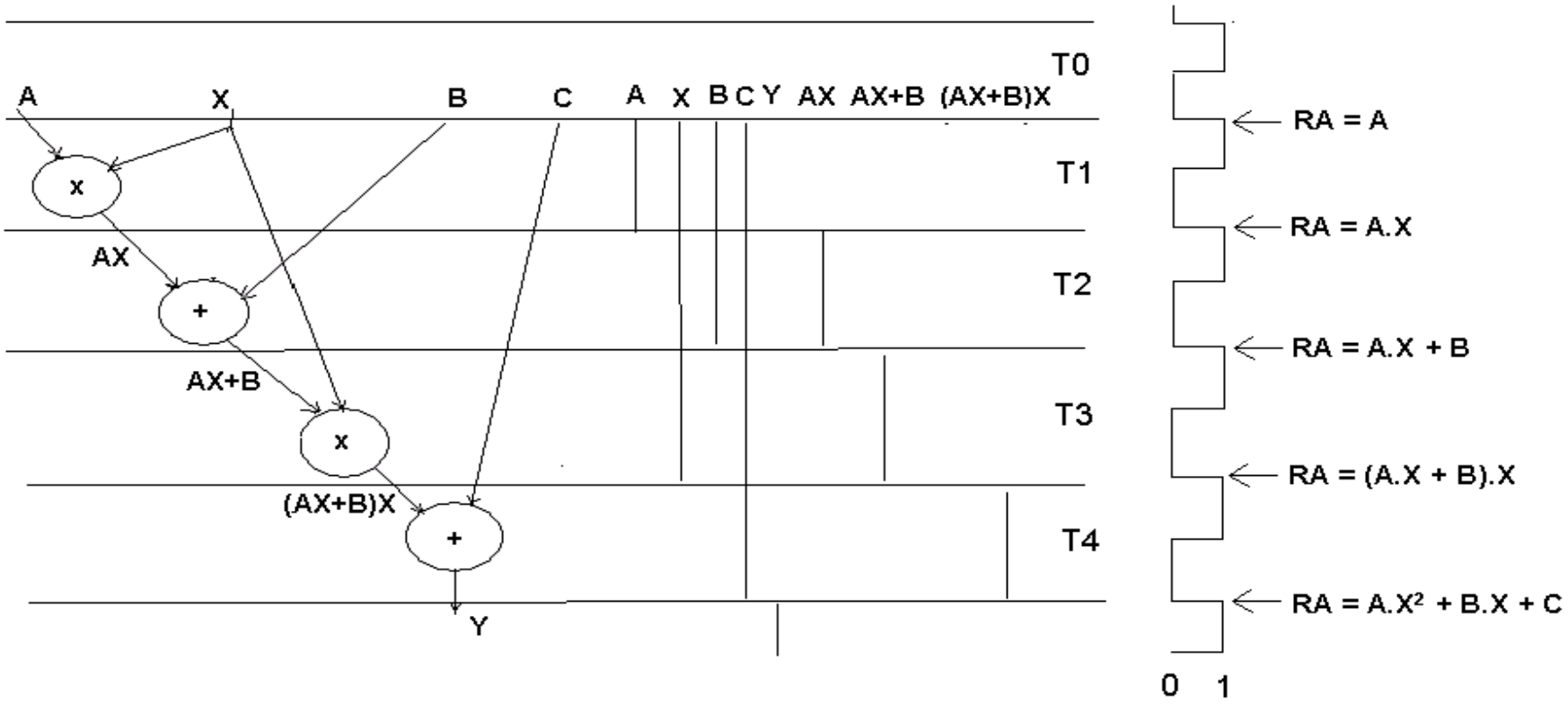

Patru etaje pipeline (același DFG ca la V2)

- Presupunem ca întârzierile Sumatorului și Înmulțitorului sunt egale => 4 Etaje balansate
- Frecventa ceas (productivitate) definita de etajul cel mai lent
- La fiecare ciclu de ceas un nou set de variabile A,,X, B, C poate intra in pipeline.
- Întârzierea circuitului (întârziere de propagare prin pipeline) se compune din întârzierile UF si registre.
- Întârzierea este mai mare decât in varianta Single cycle
- Proiectul pipeline consuma mai multe resurse decât proiectul single cycle
- După perioada de umplere a pipeline-ului, in fiecare ciclu obținem o nouă valoare Y
- Registrele dintre etaje impun disciplina semnalului de ceas.
  - Ex. Flip-Flop, front pozitiv, semnal ceas cu o faza.
- **Execuţie** dată de fluxul de date. UF si conexiunile definesc comportamentul circuitului. Schimbarea datelor schimba rezultatul operaţiilor
- Productivitatea este de 1 rezultat / perioadă de ceas.

# Solutia 4 - Proiect Multi-ciclu, V1

- **Specific**: Comunicare pe baza de MUX
- Selectam DFG din soluția 2 “Ciclu unic, V2” cu o constrângere de resurse: 1 Înmulțitor si 1 Sumator
- Pentru memorarea variabilelor folosim registre temporare, necesari pe durata de viață
- **Comunicarea** între registre și UF - MUX-uri și conexiuni directe.

# Solutia 4 - Proiect Multi-ciclu, V1




DFG și duratele de viață ale variabilelor pentru Y = (A.X + B).X + C; pe partea dreapta se indică momentele şi valorile de scriere în registrul RA (vezi unitatea de comanda pe paginile următoare)

# Soluția 4 - Proiect Multi-ciclu, V1

- X , + : in0 – stânga, in1 - dreapta
- 4 registre sunt necesare pentru valorile inițiale A, X, B, C: RA, RX, RB si RC.
- Din figura rezultă ca RA poate fi folosit pentru: A, A.X, A.X+B, (A.X+B).X si Y
- RA va memora 5 valori diferite.

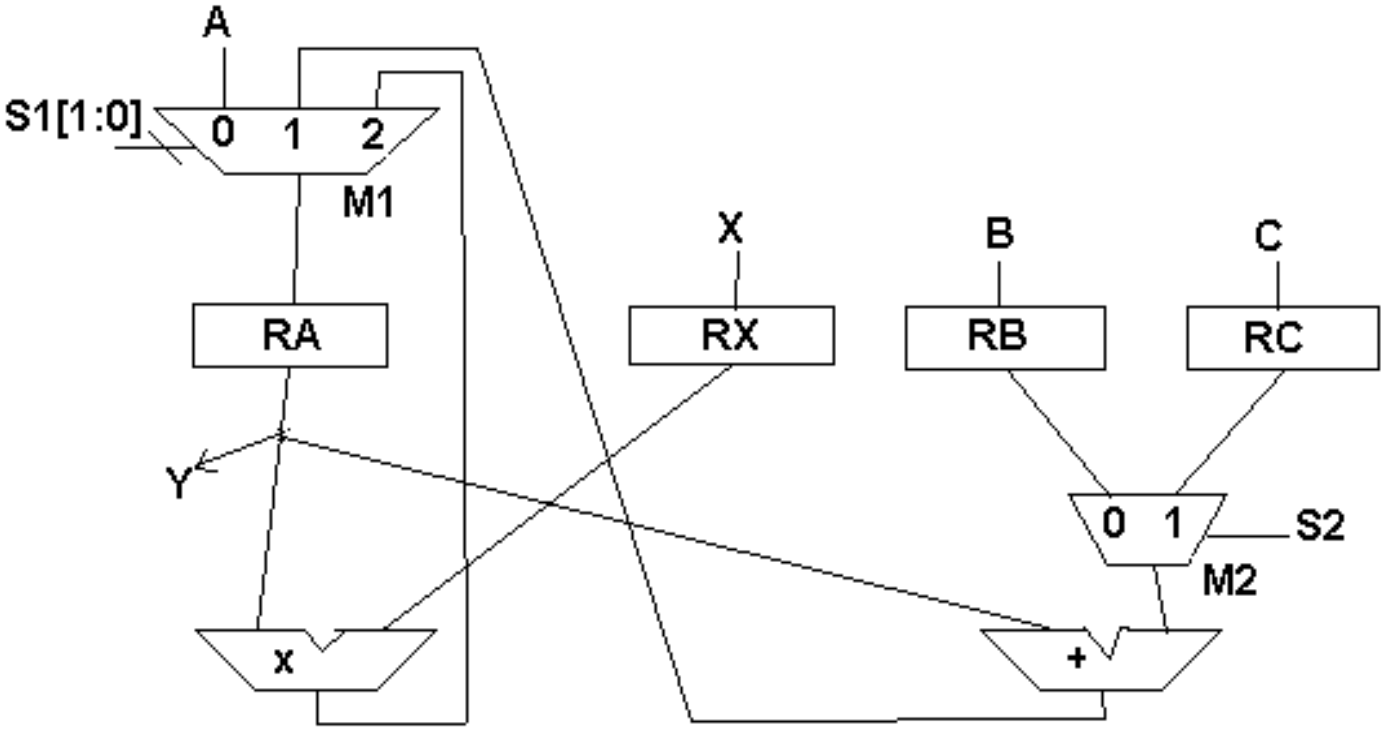
**Conexiunile necesare** intre UF si RA se prezintă in Tabelul 1

| | | Clock1 | Clock2 | Clock3 | Clock4 |
|---|---|---|---|---|---|
| Înmulțitor | in0 | A | | A.X+B | |
| Înmulțitor | in1 | X | | X | |
| Înmulțitor | Out | A.X | | (A.X+B).X | |
| Sumator | in0 | | A.X | | (A.X+B).X |
| Sumator | in1 | | B | | C |
| Sumator | Out | | A.X+B | | Y |
| Intrare RA | | A.X Înmulțitor | A.X+B Sumator | (A.X+B).X Înmulțitor | Y Sumator |

Tabel 1: Conexiunile dintre UF și RA

# Soluția 4 - Proiect Multi-ciclu, V1

**Înmulțitor in0**: conexiune directă la A
**Înmulțitor in1**: conexiune directă la X
**Sumator in0**: conexiune directă la A
**Sumator in1**: MUX2 (2:1)
**RA in**: MUX1 (3:1); o intrare pentru A, una pentru ieșirea înmulțitorului, o intrare pentru ieșirea sumatorului.




Calea de date Multi ciclu, conexiuni - MUX
Notaţii: Multiplier-x, Adder-+, MUX-M1,2, S1,2-semnale de control.

# Soluția 4 - Proiect Multi-ciclu, V1

**Unitate de Comandă** pentru transferuri între registre

Descriere RTL concret pentru Y = (A.X + B).X + C; si definirea semnalelor de comandă:

T0: Valorile initiale au fost incarcate

| | | | |
|---|---|---|---|
| T1: RA ← RA x RX; | S2 x; | S1 ← 2; | // RA = A.X |
| T2: RA ← RA + RB; | S2 ← 0; | S1 ← 1; | // RA = A.X + B |
| T3: RA ← RA x RX; | S2 x; | S1 ← 2; | // RA = (A.X + B).X |
| T4: RA ← RA + RC; | S2 ←1; | S1 ← 1; | // RA = A.$X^2$ + B.X + C |

Descriere RTL; MUX control; Trasare;

**Unitate de comandă - FSM cu 4 (5 cu T0) stări**

**Diferența Pipeline-Multi-ciclu:**

- **Pipeline**: la fiecare ciclu de ceas un nou set de variabile A, X, B, C poate intra in pipeline si se produce cate un rezultat după timpul de umplere.
- **Multi-ciclu**: la fiecare 5-lea ciclu se poate introduce un nou set de variabile si se produce un rezultat.
- Perioadele de ceas pot fi egale.

# Probleme de rezolvat

1. Pentru ecuaţia: $ax^2 + bx + c$, desenaţi DFG cu variantele:
   a. Planificarea execuţiei cu ASAP, ALAP
   b. Execuţie pipeline

   Comparaţi latenţele, laţimea de bandă şi costul fiecarei soluţii.
2. Pentru următoarea ecuaţie: $w = a\cdot(b\cdot x + z) – c\cdot(a\cdot y + x)$ se presupune:
   a. Implementare fără constrângeri.
   b. Implementare cu 1 sumator/scăzător şi o unitate de multiplicare 1; unitățile funcționale se consideră de tip combinațional.

   Pentru variantele a şi b arătaţi DFG, timpul de viaţă al variabilelor, numărul minim de registre necesare şi căile de date corespunzătoare pentru proiectare Multi-ciclu.

# Referinţe

[1] P. Eles, Z. Peng, „System Synthesis of Digital Systems”, Lecture slides, March 2000, http://www.ida.liu.se/~petel/SysSyn/lect1.frm.pdf

[2] P. Coussy, D.Gajski, M.Meredith, and A.Takach, „An introduction to high-level synthesis”, *IEEE Design Test of Computers*, 26(4):8 – 17, jul. 2009.

[3] Egon Boerger and Robert Staerk, “A Method for High-Level System Design and Analysis”, (Abstract State Machines chapter), Springer-Verlag 2003.