ANEXA_1

LIMBAJUL DE PROGRAMARE HARDWARE AHPL.

(**AHPL - A Hardware Programming Language**).

1. INTRODUCERE

In functie de scopul urmarit, un calculator numeric poate fi descris folosind diverse mijloace. Astfel, la nivelul etapei de proiectare, se folosesc adesea diagrame bloc, organigrame, limbaje specializate, tabele de cablaj etc.

In cadrul acestei lucrari calculatorul numeric este descris in termenii unor secvente de transferuri ale datelor intre diferitele primitive functionale ale calculatorului, in vederea implementarii unui algoritm dat. Se considera ca, dupa terminarea acestei etape de proiectare, elaborarea schemelor logice si a schemelor de cablaj reprezinta operatii mecanice, de rutina, care se pot automatiza cu usurinta.

Sistemele numerice de calcul prelucreaza in principal date structurate sub forma de vectori binari, stocati in registre sau memorii organizate sub forma de matrici, cu acces la linii.

In acest context apare evidenta utilitatea unui limbaj prevazut cu facilitati pentru manipularea vectorilor si matricilor.

Un asemenea limbaj a fost propus de K.E. Iverson sub numele APL (A Programming Language). Limbajul APL cunoaste o larga raspindire, ca limbaj conversational in sistemele cu multiacces IBM 360/370 si chiar in cadrul calculatoarelor personale IBM-PC compatibile. Trebuie mentionata, de asemenea, si implementarea realizata in tara noastra pe sistemele FELIX C-256/512.

APL este un limbaj foarte puternic. El permite descrierea concisa a algoritmilor si verificarea lor prin executie, pe calculator. Pentru scopurile urmarite de proiectare, limbajul APL trebuie completat cu instructiuni si alte constructii, care permit o implementare directa in hardware.

Un asemenea limbaj AHPL (A Hardware Programming Language) a fost propus de catre F. Hill si G. Peterson. AHPL, ca limbaj de descriere a transferurilor intre registre, asigura o corespondenta directa intre notatia folosita si implementarea ei in hardware.

AHPL foloseste, pe linga constructiile proprii, si un subset de constructii din APL, care isi gasesc un corespondent in hardware. Frecvent, in scopul facilitarii comunicarii si documenarii, in programele AHPL se mai utilizeaza unele notatii APL, care descriu algoritmul de operare al unui calculator numeric.

In momentul traducerii in hardware a descrierii AHPL a modulului numeric dat, aceste notatii nu sunt luate in consideratie.

Autorii limbajului AHPL au elaborat si un compliator pentru hardware, care are ca intrare descrierea AHPL a sistemului numeric, iar ca iesire - schema sistemului, la nivel de diagrame.

Indiferent de implementarea unui algoritm, este o buna practica aceea de a descrie mai intai algoritmul in APL si de a-l executa pe un calculator, prevazut cu un compliator de APL.

In continuare algoritmul descris in APL poate fi implementat in hardware, firmware (microprogram) sau poate fi executat direct, fiind implementat in software.

Conventii privind operanzii folositi in AHPL.

Operanzii manipulati in limbajul AHPL reprezinta marimi constante sau variabile.

Constantele sunt date numerice sau alfanumerice. Ele se reprezinta prin numere standard, respectiv - prin litere standard sau numere cuprinse intre semnele apostrof.

Variabilele pot fi scalari, vectori, matrici.

Este important de facut distinctie intre variabile si valori.In limbajele conventionale de programare o variabila reprezinta un nume prin care se face o referire la un operand; o valoare reprezinta marimea pe care o ia efectiv operandul. In AHPL, prin variabila se intelege numele unui registru al carui continut este manipulat printr-o instructiune a programului, iar prin valoare - data care se plaseaza in registru.

Tipurile de operanzi in AHPL se vor deosebi prin conventii tipografice: litere mici pentru scalari, majuscule pentru vectori si majuscule ingrosate pentru matrici.

Transferul unei constante (binare - in cazul de fata) intr-un registru AC se noteaza astfel:

$$AC \longleftarrow 0,1,1,1,1,0,1,0$$

Transferul continutului unui registru RD, intr-un registru AC, fara a se modifica continutul lui RD se reprezinta prin instructiunea:

$$AC \leftarrow RD$$

Un vector constituie o colectie de operanzi avand o structura unidimensionala. Numarul componentelor vectorului reprezinta dimensiunea vectorului. Pentru a specifica dimensiunea unui vector oarecare (AC) se va folosi notatia ρAC, unde ρ reprezinta operatorul “dimensiune”. Daca AC are 16 biti (ranguri binare), atunci $\rho AC = 16$. In aceasta lucrare, in cazul in care nu se

vor face mentiuni speciale, pozitiile bitilor individuali vor fi specificate prin indici cu originea 0, plasata in extrema stanga: AC_0 ,AC_1 ,.....,$AC_{\rho AC-1}$

Trebuie observat ca aceasta notatie este destul de greoaie in textele dactilografiate, ceea ce face ca, in multe cazuri, indicii sa fie plasati pe aceeasi linie cu numele registrului, de exemplu: ACi - bitul de rang i din registrul AC. Pentru a specifica un grup de biti adiacenti, dintr-un registru se foloseste notatia: $A_{i:j}$ sau A(i:j), unde sunt inclusi si bitii i si j.

Operanzii de tip matricial se reprezinta sub forma unui tablou bidimensional, constituit din elemente cu indici inferiori si superiori. Indicii inferiori specifica coloanele, iar cei superiori liniile. Se considera matricea **M** avind ρ1**M** coloane si ρ2**M** linii, unde ρ1 si ρ2 reprezinta notatiile pentru operatorii care aplicati lui **M** furnizeaza numarul de coloane si numarul de linii ale tabloului **M**.

$$\begin{vmatrix} \mathbf{M}^0{}_0 \ \mathbf{M}^0{}_1 \ \ldots\ldots\ldots\ldots\ldots\ldots\mathbf{M}^0{}_{\rho 1\mathbf{M}-1} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots \mathbf{M}^i{}_j \ldots\ldots\ldots\ldots\ldots \\ \mathbf{M}^{\rho 2M-1}{}_0 \ \mathbf{M}^{\rho 2M-1}{}_1 \ldots\ldots\ldots\ldots\ldots \mathbf{M}^{\rho 2M-1}{}_{\rho 1M-1} \end{vmatrix}$$

Linia i a matricii **M** se specifica prin notatia $\mathbf{M}^i$ sau **M<i>,** iar coloana j - prin notatia $\mathbf{M}_j$ sau **M**[j]. Liniile succesive i...k, ale matricii **M** se noteaza prin $M^{i:k}$ sau **M**<i:k>, iar coloanele succesive j...l - prin $\mathbf{M}_{j:l}$ sau **M**[j:l].

Conventii privind operatorii APL si AHPL.

Operatorii folositi sunt *operatorii primitivi* si *operatorii de tip mixt*.

Operatorii primitivi manipuleaza, de regula, operanzi de tip scalar, desi ei pot fi extinsi atat la vectori, cat si matrici. Acesti opeatori se pot referi la o singura variabila (operatori unari) sau la doua variabile (operatori binari). In cele ce urmeaza se prezinta operatorii primitivi aritmetici, logici si relationali. In scopul facilitarii comunicarii, in textele ce reprezinta programe AHPL pot fi intalniti toti acesti operatori. Implementari directe in hardware au insa numai operatorii logici. Operatorii aritmetici si relationali sunt specifici limbajului APL.

Operatorii aritmetici manipuleaza numere reale:

x + y adunare; suma algebrica, (APL),

x - y scadere; diferenta algebrica, (APL),

x ∗ y inmultire; inmultire algebrica, (APL),

x % y impartire algebrica; (APL).

|x valoare absoluta; (APL).

Operatorii logici manipuleaza numere binare (vectori binari):

$\overline{x}$ NU (APL si AHPL),

$x \cap y$ SI (APL si AHPL),

$x \cup y$ SAU (APL si AHPL),

$x \oplus y$ SAU-Exclusiv (APL si AHPL).

Operatorii relationali sunt de forma:

(x $\Re$y) unde $\Re \in \{ <, \leq, =, \geq, > \}$ (APL).

In versiunile existente de APL, incercarea de a folosi alte tipuri de variabile, cu exceptia celor logice, conduce la eroare de domeniu.

In raport cu operatorii aritmetici si relationali nu exista limitari privind folosirea variabilelor.

In exemplele urmatoare se considera:

x = 1, y = -3, W = (4,-5,0,2) si U = (1,2,1,-1).

Instructiune:	**Rezultat:**
$z \leftarrow x + y$	z = -2
$z \leftarrow \overline{x}$	z = 0
Z ←W + U	Z = (5,-3, 1,1)
Z ←W * U	Z = (4,-10,0,-2)
Z ← W % U	Z = (4,-2.5,0,-2)
z ← (x < y)	z = 0
z ← (x > y)	z = 1

Operatorii de tip mixt sunt extrem de puternici, deoarece permit manipularea unor combinatii de scalari, vectori si matrici. In cele ce urmeaza se vor prezenta *operatorii de tip mixt*, cu specificarea limbajelor in care se utilizeaza.

Notatie	Denumire	Semnificatie	Observatii
X,Y	Inlantuire/ Concatenare	X_0 ,...,$X_{\rho X-1}$,Y_0 ,....,$Y_{\rho Y-1}$	APL si AHPL
M!N	Inlantuire linii.	O matrice cu ρ2**M**+ρ2**N** linii cu liniile lui M peste liniile lui N.	AHPL
k↑X	Extragere	Se extrag primele k elemente din vextorul X.	APL
k↓X	Eliminare	Se elimina primele k elemente din vectorul X.	APL
⊥X	Decodificare	Echivalentul zecimal al vectorului binar X	APL
n⊤p	Codificare binara.	Un vector cu n elemente binare, obtinut prin reprezentarea numarului zecimal p in baza doi.	APL si AHPL
@/X	Reducere	X_0 @X_1 @X_2 @.....@$X_{\rho X-1}$	APL si AHPL
X ← @/**M**	Reducere linie	$X_i \leftarrow @/M^i$	AHPL si APL
X ←@//**M**	Reducere coloana	$X_j \leftarrow @/M_j$	AHPL si APL
X ←U/Y	Comprimare	Vectorul X este obtinut din vectorul Y prin su-primarea rangurilor X_i pentu care U_i =0 U este vector binar.	APL si AHPL in descrierea unitatilor logice combinationale
A ← U/**M**	Comprimare linii	**A** ← U^i/$\mathbf{M}^i$	Idem linii.
A ← U//**M**	Comprimare coloane	$\mathbf{A}_j$← U/$\mathbf{M}_j$	Idem coloane

Nota1: @ este un operator logic sau aritmetic.
Nota2: **A** si **M** sunt matrici, iar U este un vector binar.

In exemplele care urmeaza se ilustreaza utilizarea operatorilor de tip mixt.

Inlantuirea.

Fie: X = (1,2,3,4) si Y = (5,6,7),

daca $Z \leftarrow X,Y$ atunci, rezulta: Z = (1,2,3,4,5,6,7)

Inlantuirea permite descrierea operatiilor de deplasare si rotire ale vectorilor si matricilor.

In cele ce urmeaza se vor prezenta notatiile AHPL pentru deplasari si rotiri de vectori.

Fie vectorul $A = (A_0 ,A_1 ,A_2 ,.....,A_{\rho A-1})$

Deplasare logica - dreapta (cu inserta unui zero in bitul 0).

$A \leftarrow 0, A_0 ,A_1 ,A_2 ,.....,A_{\rho A-2}$

Deplasare logica - stanga (cu insertia unui zero in bitul de rang ρA-1).

$A \leftarrow A_1 ,A_2 ,.....,A_{\rho A-1},0$

Deplasare aritmetica dreapta (cu extinderea bitului de rang 0 in bitul de rang 1).

$A \leftarrow A_0, A_0 ,A_1 ,A_2 ,.....,A_{\rho A-2}$

Deplasarea aritmetica stanga este identica cu deplasarea logica stanga.

Rotire -dreapta.

$A \leftarrow A_{\rho A-1}, A_0 ,A_1 ,A_2 ,.....,A_{\rho A-2}$

Rotire - stanga.

$A \leftarrow A_0 ,A_1 ,A_2 ,.....,A_{\rho A-1,} 0$

In operatiile de deplasare si rotire, inaintea bitului de rang 0 si dupa bitul de rang ρA-1, se pot plasa biti individuali reprezentand indicatori de conditii (de exemplu, din unitatea de executie, bitul de transport). Cele mai multe microprocesoare dispun de instructiuni care implementeaza asemenea operatii de deplasare. Ele sunt folosite pentru instructiunile de transfer conditionat al comenzii.

In continuare sint ilustrate operatiile de rotire/deplasare circulara a liniilor unei matrici.

Fie:
$$\mathbf{M} = \begin{vmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{vmatrix}$$

Deplasare circulara in sus.

Daca $\mathbf{N} \leftarrow \mathbf{M}^{1:2} \ ! \ \mathbf{M}^{0}$, atunci rezulta: $\mathbf{N} = \begin{vmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{vmatrix}$

Deplasare circulara in jos.

Daca $\mathbf{N} \leftarrow \mathbf{M}^{2}\ !\ \mathbf{M}^{0:1}$, atunci rezulta: $\mathbf{N} = \begin{vmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{vmatrix}$

Extragerea.

Fie: X = (1,2,3,4,5,6)

daca $Y \leftarrow 4\uparrow X$, atunci rezulta: Y = (1,2,3,4) in APL

Eliminarea.

daca, $Y \leftarrow 4\downarrow X$ atunci rezulta: Y = (4,5,6) in APL.

In AHPL notatiile echivalente vor fi urmatoarele:

$Y \leftarrow X_{0:3}$ (extragere) si respectiv $Y \leftarrow X_{4:6}$ (eliminare)

Codificare binara.

Fie: x = 13 si y = 17

daca: $X \leftarrow \top 13$ si $Y \leftarrow \top 17$, atunci: X = (1,1,0,1) si Y = (1,0,0,0,1)

Decodificarea binara.

Fie: X = (1,1,0,1) si Y = (1,0,0,0,1)

daca: $x \leftarrow 4\perp X$ si $y \leftarrow 5\perp Y$, atunci: x = 13 si y = 17.

Reducere.

Fie @ un operator binar care se aplica unui vector binar X,rezultatul operatiei va fi de forma:

$x \leftarrow (\ldots.((X_0 @ X_1) @ X_2) @ \ldots.. X_{\rho X-1})$

Expresia:

$x \leftarrow +/X$ este echivalenta cu :

$$x \leftarrow \sum_{i=0}^{\rho X-1} X_i$$

Daca X este un vector logic (cu componente binare), atunci urmatoarele operatii vor furniza informatii privind:

$x \leftarrow +/X$, numarul de unitati din vectorul X;

$x \leftarrow \cup/X$, prezenta a cel putin unei unitati diferita de zero, daca $x <> 0$;

$x \leftarrow \cap/X$, prezenta tuturor componentelor egale cu unu, daca x = 1.

Fie matricea **M**:

$$\mathbf{M} \leftarrow \begin{vmatrix} 1\ 0\ 0\ 1 \\ 0\ 1\ 0\ 0 \\ 1\ 0\ 1\ 1 \end{vmatrix}$$ atunci: +/**M** = (2,1,3); +//**M** = (2,1,1,2,) si +/(+//(**M**)) = 6.

Comprimare.

Fie U = (1,0,0,1) si A = (1,2,3,4), atunci: X ← U/A, conduce la rezultatul: X = (1,4)

Considerind matricea **M** si vectorul U de mai sus, comprimarea pe linii **N** ← U/**M** va furniza matricea **N**, de forma:

$$\mathbf{N} \leftarrow \begin{vmatrix} 1\ 1 \\ 0\ 0 \\ 1\ 1 \end{vmatrix}$$

Comprimarea coloanelor va fi exemplificata cu un vector logic V, de forma: V = (1,1,0).

N ←V//**M** va avea aspectul urmator:

$$\mathbf{N} \leftarrow \begin{vmatrix} 1\ 0\ 0\ 1 \\ \\ 0\ 1\ 0\ 0 \end{vmatrix}$$

Ca un ultim exemplu, penru a ilustra forta limbajului AHPL se va scrie un program pentru cautarea intr-o lista de cuvinte plasate intr-o memorie, in vederea extragerii acelor cuvinte pentru care primii 4 biti sint egali cu 1.

Se considera ca lista de cuvinte binare, de cite 16 biti, se afla intr-o memorie **M**, cu un numar de ρ2**M** cuvinte. Se va forma un vector U cu ρ2TM componente, care vor fi 0 pentru acele cuvinte, care nu indeplinesc conditia impusa si 1, in caz contrar. In final se va genera o matrice **N** obtinuta prin comprimarea coloanelor lui **M** dupa vectorul U.

1. U ← ρ2**M**T0 /* se genereaza valoarea initiala a vectorului U
2. i ← 0 /* se initializeaza contorul i
3. → ((∩/($\mathbf{M}^{i}_{0} \cap \mathbf{M}^{i}_{1} \cap \mathbf{M}^{i}_{2} \cap \mathbf{M}^{i}_{3}$)) = 0)/(5) /* se testeaza conditia
4. U_i ←1 /* se atribuie valoarea 1 componentei care nu indepplineste conditia impus
5. i ←i + 1 /* se incrementeaza contorul i
6. → (i < ρ2**M**)/(3) /* transfer conditionat al comenzii
7. **N** ←U//**M**

Se poate observa ca vectorul U se pute genera simplu prin opeatia:

$U \leftarrow (\mathbf{M}_0 \cap \mathbf{M}_1 \cap \mathbf{M}_2 \cap \mathbf{M}_3)$

Deci, tot programul secvential de mai sus se poate inlocui cuexpresia:

$\mathbf{N} \leftarrow (\mathbf{M}_0 \cap \mathbf{M}_1 \cap \mathbf{M}_2 \cap \mathbf{M}_3)//\mathbf{M}$

Conventii AHPL pentru descrierea logicii combinationale.

S-a aratat ca procesul de prelucrare a datelor consta in trans-ferul acestora intre registrele sursa si registrele destinatie prin intermediul unor retele logice combinationale. Acestea din urma reprezinta resurse hardware, care pot fi folosite in mod repetat, cu diverse argumente, in puncte diferite ale secventei AHPL, care descrie algoritmul.

Se considera o retea logica combinationala, care realizeaza operatia NOR (SAU-NU) asupra rangurilor a doi vectori de 4 biti.

$C \leftarrow (A_0 \cap B_0 , A_1 \cap B_1 , A_2 \cap B_2 , A_3 \cap B_3)$

Daca expresia de mai sus apare in mod repetat in secventa AHPL, se poate folosi o notatie prescurtata:

C ←NOR(A;B), care este asemanatoare unei notatii de subrutina intr-un limbaj de programare. In contextul de fata NOR(A;B) va fi mentionata ca o functie sau o unitate logica combinationala.

Intr-un program AHPL, ea va fi descrisa o singura data, dupa care poate fi apelata in mod repetat, cu diferite argumente.

Sub forma generala descriera unei asemenea functii fi delimitata de titlu si de sfirsit (END).

```
UNIT:NOR(A;B)
    conexiuni
    .....
    conexiuni
END
```

Dupa titlu pot urma declaratii privind vectorii manipulati, lungimea lor etc. Conexiunile sunt instructiuni care se traduc direct in hardware sub forma unor legaturi intre intrarile si iesirile unor circuite logice combinationale.

Intr-un alt paragraf se vor da mai multe exemple privind descrierea unor unitati logice combinationale: BUSFN, DCD, ADD etc.

2. Descrierea sistemelor numerice la nivelul transferurilor intre registre folosind limbajul AHPL.

Sistemele numerice de calcul au un caracter complex, ceea ce conduce la o serie de dificultati, atat in privinta descrierii lor functionale, cat si a proiectarii lor.

Plecand de la modul lor de functionare, ele pot fi partajate in subsisteme sau module mai simple, cu specificatii bine precizate in legatura cu semnalele de intrare si iesire, care vor reprezenta date, comenzi, stari, sincronizari etc, pe de-o parte, cat si in legatura cu algoritmul de operare.

Sub aspect functional, un sistem numeric poate fi descris prin proceduri si functii, plecand de la algoritmul pe care trebuie sa-l execute. Astfel, intr-o prima etapa, descriera sistemului numeric va fi asemanatoare cu cea a unui program de mare complexitate.

Procedurile vor avea ca implementari fizice modulele, in timp ce functiile vor fi realizate prin unitati logice combinationale.

In cadrul modulelor pot fi evidentiate unitatile de executie si de comanda. In descrierea modulelor se intalnesc in mod frecvent functii a caror implementare conduce la scheme logice combinationale. Pentru prezentarea modalitatilor de descriere a modulelor si functiilor se va folosi forma BNF (Backus-Naur Form).

Structura unei proceduri (modul) este urmatoarea:

MODULE: < nume modul >
    < lista de declaratii >
    < lista de pasi AHPL >
    ENDSEQ
    < lista de instructiuni individuale nesincronizate si instructiuni >
END

< lista de declaratii >:= < MEMORY (memorii) >
    < INPUTS (intrari) >
    < OUTPUTS (iesiri) >
    < BUSES (magistrale) >
    < LABELS (etichete) >
    < ONE-SHOTS (monostabile) >
    < COMBUSES (magistrale decomunicatii)>

< MEMORY >: < lista de bistabile >;< lista de registre >;<lista de memorii >

< lista de bistabile >:= < nume bistabil >;...;< nume bistabil >

< lista de registre>:= < nume registru[i]>;...;<nume registru[j] >

< i, j >:= dimensiunile registrelor

< i, j >:= 1|2|.....|n

< lista de memorii >:= < nume memorie[m;n] >;...;< nume memorie[r;q]

< m,r >:= numar de cuvinte in memorii

< n,q >:= numar de biti in cuvintele de memorie

< m, n, q, r >:= 1|2|.....|n

< INPUTS >: < lista de vectori >;< lista de semnale individuale>

< lista de vectori >:= < nume vector[i] >;...;< nume vector[j] >

< lista de semnale individuale >:= < nume semnal >;...;< nume semnal >

<OUTPUTS >: < lista de vectori >;...;< lista de semnale individuale >

< BUSES >: < lista de magistrale >

< lista de magistrale >:= < nume magistrala[i] >;...;< nume magistrala[j] >

< i,j >:= dimensiunile magistralelor

< LABELS >: < lista de etichete >

< lista de etichete >:= < nume eticheta >;...; < nume eticheta >

(etichetele sunt folosite pentru a referi campuri din anumite registre - subregistre etc)

< ONE-SHOTS >: < lista de monostabili >

< lista de monostabili >:= < numemonostabil[i] >;...;< nume monostabil[j] >

< i,j >:= duratele semnalelor la iesirile monostabilelor, date in multipli ai perioadei de tact

< COMBUSES >: < lista de magistrale de comunicatii >

< lista de magistrale de comunicatii >:= < nume magistrala[i] >;< nume magistrala[j]

(magistrala de comunicatii trebuie sa fie conectata la un set de linii de intrare/iesire in/din modul >

Exemplu de declaratii: Se considera un modul FILTRU DE CUVINTE plasat intre doua sisteme numerice A (emitator) si B (receptor), care permite trecerea unor vectori binari ce indeplinesc anumite conditii. Modulul poseda registrele de intrare si iesire REGIN[16] si

REGIES[16], registrul intern A[4] si bistabilul a, intrarea X[16] si iesirile Z[16], gatain, gataies.

Descrierea modulului va incepe astfel:

MODULE: FILTRU DE CUVINTE

MEMORY: REGIN[16]; REGIES[16]; A[4], a

INPUTS: X[16]

OUTPUTS: Z[16]; gatain; gataies

In continuare se va prezenta secventa de pasi AHPL. Un pas AHPL contine toate operatiile elementare, sincrone cu o perioada a ceasului, care au loc la nivelurile resurselor unitatilor de executie si comanda. Fiecare pas de comanda va consta in instructiuni AHPL de atribuire si/sau o instructiune AHPL de ramificare/control).

< secventa de pasi AHPL >:= < pas AHPL>
< pas AHPL >
.....
< pas AHPL >

< pas AHPL >:= < lista de instructiuni de atribuire >;< instructiune de ramificatie >

< instructiune de atribuire >:= < nul >|< transfer sincron >|< conexiune >|< transfer sicron; conexiune >

< instructiune de ramificare >:= < nul >|< $\rightarrow$ (F)/(S_j) >|< $\rightarrow$ (S_j) >|< DEAD END-fara continuare >

< nul >:= < absenta instructiunii de atribuire >|< instructiune implicita de ramificare la pasul urmator >

< transfer sincron >:= < VD <- VLCS >|< VD $\leftarrow$ MLCS*F >|< MD*F $\leftarrow$ VLCS >

VD - vector destinatie. El reprezinta fie un registru simplu, fie un vector format din unul sau mai multe elemente de memorie asamblate pe baza unor operatori de selectie. Indicii superiori si inferiori, care afecteaza operanzii, sunt in mod obligatoriu constante.

Se folosesc urmatorii operatori de selectie:

A_j - elementul j din vectorul

$A_{m:n}$ - elementele m, pina la n, din vectorul A

, - inlantuire (concatenare)

A ! B - inlantuire pe linii

M^j - linia j din matricea M

$M_{m:n}$ - liniile m, pina la n, din matricea M.

VLCS reprezinta un vector logic combinational sursa, constituit din expresii logice combinationale evaluate, ale caror argumente (operanzi) pot fi: elemente de memorie, intrari, functii logice, magistrale, constante binare.

Operatorii din expresii sunt de tip logic: $\cap$(SI), $\cup$ (SAU),$^-$ (NU), $\cap$ / (SI aplicat elementelor vectorului), $\cup$/ (SAU aplicat elementelor vectorului), $\oplus$/ (SAU-Exclusiv), SYN (sincronizare).

In absenta parantezelor, in expresiile din VLCS, operatorii logici si de selectie vor avea urmatoarele prioritati:

1. Negatia si Sincronizarea,
2. Toti operatorii de selectie, cu exceptia inlantuirii,
3. $\cap$
4. $\cup$ sau $\oplus$,
5. Inlantuirea.

MD - *matrice de memorie sau asamblaj de registre* de memorare.

MLCS - *matrice logica combinationala sursa*, constituita din asamblaje de vectori logici combinationali.

F - *vector de selectie*, ale carui componente sunt expresii logice combinationale evaluate, care se exclud mutual:

$F = (f_1 (x_1 ,...,x_n),.....,fm (x_1 ,...,x_n),$

unde: $(f_i \cap f_j = 0)$ si $(\cup/F) = 1$; $i,j = 1,...,n; i = j; x_I \in \{ 0,1 \}$.

Expresiile: MLCS*F si MD*F sunt echivalente cu expresiile:

F/MLCS si F/MD (selectie pe linii).

< conexiune > := < BUS = VLCS >|< BUS = MLCS*F >|< Z = VLCS >|< Z = MLCS*F >

Exemple:

$D \leftarrow A_{m:n} , B_{p:q}$

$D \leftarrow A + B,C$

Fie matricea logica combinationala sursa: (A ! B ! C), unde A, B, C sunt vectori logici combinationali, si vectorul logic:

F = (d, e, f).

In aceste conditii se pot scrie urmatoarele instructiuni de atribuire:

1. D ←(A ! B ! C) * (d, e, f)

echivalenta cu:

D ←d ∩ A U e ∩B U f ∩ C

2. (A ! B ! C) * (d, e, f) ←D

echivalenta cu:

d ∩A ∪e ∩B ∪f ∩C ←D

3. BUS = (A ! B ! C) * (d, e, f)

reprezenand o conexiune conditionala la magistrala BUS a unuia dintre vectorii A, B, sau C.

In instructiunea de tip ramificatie, F are semnificatia prezentata mai sus, iar E constituie un vector ale carui componente (E_i) sunt numere/etichete de pasi AHPL, din secventa data, la care poate avea loc ramificarea (transferul comenzii).

De exemplu, ramificarea:

$\rightarrow (\overline{x}, \quad x)/(E_i, E_j)$

asigura transferul comenzii la linia Ei , daca x = 1 sau la linia Ej , daca x = 0.

DEAD END marcheaza sfarsitul comenzii, terminarea operarii in sensul ca instructiunea de comanda nu mai genereaza trecerea la pasul urmator din secventa, acesta lipsind.

Dupa END SEQ (sfarsitul secventei de pasi AHPL), in descrierea modulului sunt plasate instructiunile individuale de atribuire nesincronizate, conexiunile permanente, operatiile de comanda asincrone, cum ar fi operatia RESET(1), care forteaza reluarea secventei numerotate de pasi AHPL de la linia 1.

Descrierea AHPL a unei functii.

UNIT: < nume de functie >(<parametri>)

< declaratii >

< secventa de instructiuni de conexiune, contorizare,

transfer al comenzii >

END

< parametri > := < lista de intrari >

< lista de intrari >:= < VLC >|< MLC >

< declaratii >: < INPUTS >

< OUTPUTS >

< INPUTS >: < lista de intrari separate prin ; >

< OUTPUTS >:< vector logic combinational >

< secventa de instructiuni > := < conexiune >| < atribuire de valoare unui indice >|< transfer conditionat al comenzii >

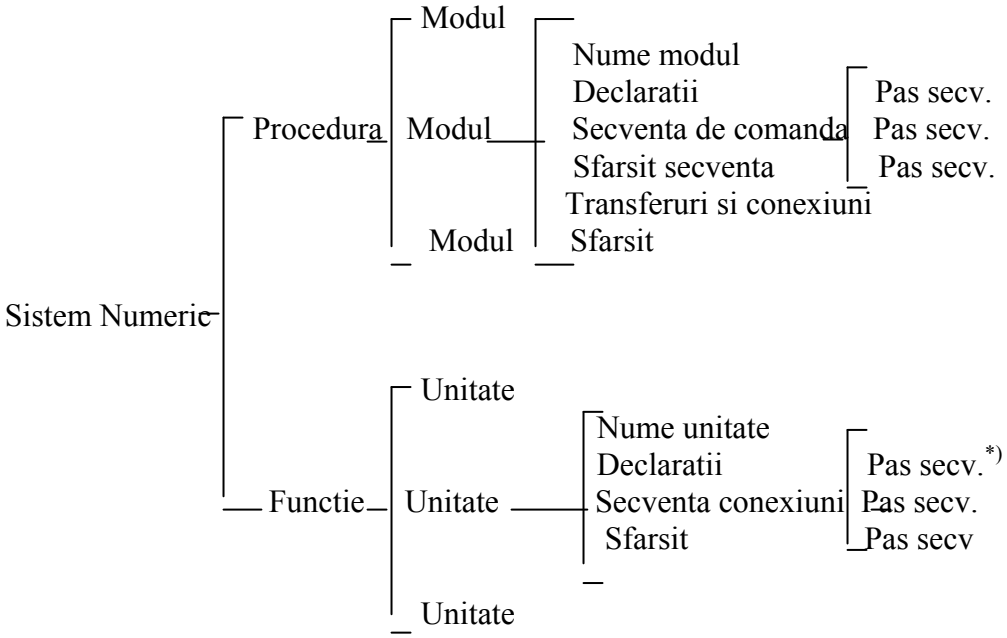
< atribuire de valoare unui indice > := < nume indice ⇐ valoare binara | zecimala >

< transfer conditionat al comenzii >:= < ⇒ (F)/(S) >

Efectul executiei unui program AHPL, ce descrie o unitate, reprezinta compilarea - generarea schemei hardware pentru acea unitate combinationala. Numai instructiunile de conexiune genereaza efectiv hardware. Instructiunile care manipuleaza indici si cele de transfer asigura efectuarea unor operatii de ciclare pentru generarea unor copii multiple ale aceleiasi scheme.

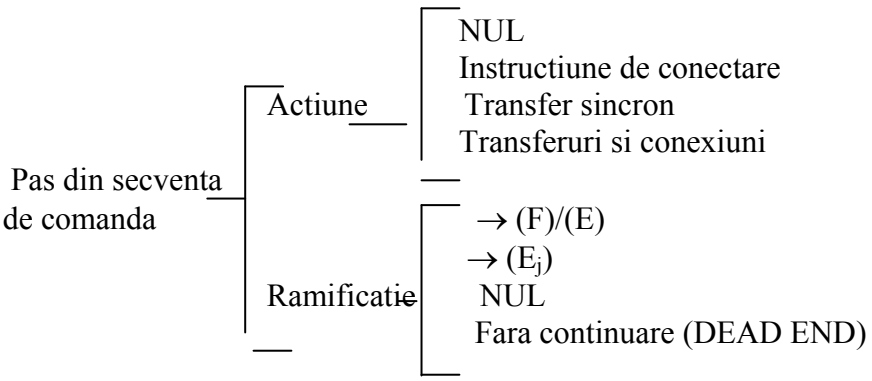
Folosirea functiilor, in cadrul unor module, presupune descrierea lor ca unitati, o singura data, in vederea compilarii/generarii schemei logice combinationale specifice. Unitatea poate fi apelata ca functie, in cadrul instructiunilor de atribuire (transfer | conexiune), din secventa de pasi AHPL ai unui modul.

In rezumat, structura bloc a unui sistem numeric consta in module si functii:

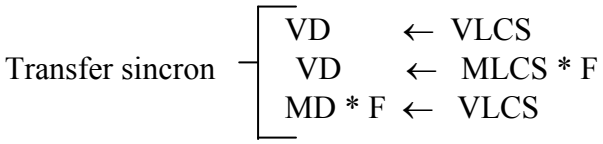


*) Specifica o conexiune

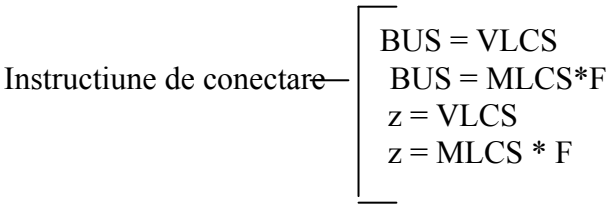
Sintaxa unui pas de comanda din secventa:



Transferul sincron poate avea unul din urmataoarele formate:



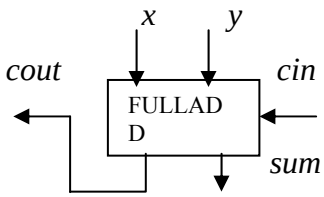
Instructiunea de conectare poate avea unul din urmatoarele formate:



Exemple de descrieri AHPL ale unor UNITATI (Circuite Logice Combinationale) si ale unor module.

2.1. Sumatorul Complet (FULLADD).

Sumatorul Complet reprezinta un circuit logic combinational cu trei intrari: *x, y, cin* si doua iesiri: *sum, cout*. Intrarile *x* si *y* sunt intrari de date de cate un bit (corespunzatoare unui rang oarecare *i*, in cazul adunarii a doi vectori binari X si Y), iar *cin* este intrarea de transport (din rangul inferior). Iesirea *sum* corespunde sumei, pentru rangul curent, iar *cout* constituie transportul in rangul urmator.



Ecuatiile logice pentru *sum* si *cout* sunt urmatoarele:

$$sum = (x \cap \overline{y} \cap \overline{cin}) \cup (\overline{x} \cap y \cap \overline{cin}) \cup (\overline{x} \cap \overline{y} \cap cin) \cup (x \cap y \cap cin)$$

$$cout = (x \cap y) \cup (x \cap cin) \cup (y \cap cin)$$

Aceste ecuatii se pot implementa cu ajutorul portilor SI, SAU, NU.

Folosind si porti SAU-EXCLUSIV, expresiile de mai sus devin mai simple:

$$sum = x \oplus y \oplus cin$$

$$cout = ((x \oplus y) \cap cin) \cup (x \cap y)$$

Cu ajutorul ecuatiilor de mai sus se poate prezenta o secventa AHPL, de descriere a Sumatorului Complet, sub forma unei Unitati:

UNIT: FULLADD(*x* ; *y* ; *cin*)

INPUTS: *x* ; *y* ; *cin*

OUTPUTS: FULLADD[2]

1. $a = x \oplus y$
2. $b = x \cap y$
3. $sum = a \oplus cin$
4. $c = a \cap cin$
5. $cout = b \cup c$
6. $FULLADD_0 = cout$
7. $FULLADD_1 = sum$

END

Primele cinci instructiuni reprezinta pasi de conectare, iar ultimele doua instructiuni constituie iesirile.

Se poate observa ca secventa care descrie un Sumator Complet are un caracter “spatial”, in sensul ca pasii reprezentati corespund iesirilor/intrarilor portilor logice, care intra in componenta unei retele spatiale. Pasii 1-7 nu sunt legati de vreun mecanism de temporizare. Elementul timp intervine, in acest context, numai in legatura cu intarzierea in propagarea

semnalelor prin porti.. Intrarile *x*, *y* si *cin* trebuie sa fie stabile pana la obtinerea rezultatelor *sum* si *cout*.

Pe baza Unitatii FULLADD se poate genera un sumator cu transport succesiv pentru numere binare de cate 16 biti.

Sumator, ADD, cu transport succesiv, pentru numere binare de cate 16 biti.

Pentru realizarea unui asemenea sumator, se vor utiliza, in calitate de blocuri constructive, componentele Sumatorului Complet: $FULLADD_0$ si $FULLADD_1$. Acestea vor fi chemate succesiv, in cadrul unor cicluri, pentru fiecare bit al sumatorului ADD. In programul AHPL, pentru controlul ciclurilor, se va introduce un indice *i*, care va fi initializat, decrementat si testat. Operatiile corespunzatoare manipularii indicelui *i* vor fi marcate cu sageti cu corp dublu, care vor fi tratate intr-o maniera specifica de catre compilatorul, care va genera schema lui ADD.

UNIT: ADD(*X*; *Y*)

INPUTS *X*[16]; *Y*[16]

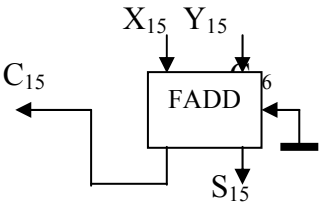
OUTPUTS: *ADD*[17]

1. $C_{16} = 0$
2. $i \Leftarrow 15$
3. $C_i = \text{FULLADD}_0 (X_i; Y_i; C_{i+1})$
4. $S_i = \text{FULLADD}_1 (X_i; Y_i; C_{i+1})$
5. $i \Leftarrow i - 1$
6. $\Rightarrow (i \geq 0)/(3)$
7. $ADD = C_0, S_{0:17}$

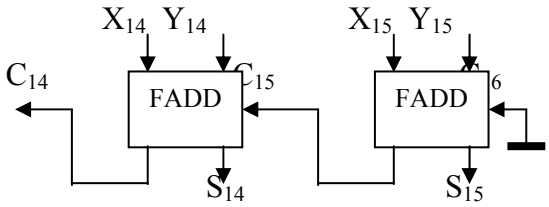
END

Compilarea manuala a acestui program AHPL, va genera elementele schemei sumatorului ADD, in maniera prezentata mai jos la nivelul primelor doua si al ultimei parcurgeri a programului.

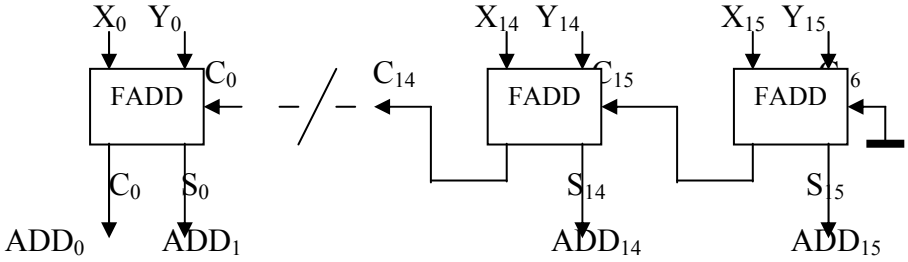
1.



2.

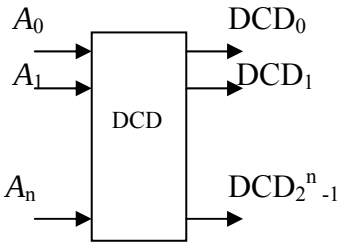


3.



2.2. Decodificatorul DCD.

Decodificatorul DCD(*A*), unde *A* este un vector binar de *n* biti, este un circuit combinational cu *n* intrari si 2^n iesiri. Pentru oricare vector binar, aplicat la intrare, numai o singura iesire a decodificatorului va fi activa.



Descrierea DCD(*A*) in AHPL este urmatoarea:

UNIT: DCD(*A*)
INPUTS: A[*n*]
OUTPUTS: DCD(2^n)
1. i $\Leftarrow$ 0
2. $DCD_i = \cap/((n \; T \; i) / A), (\overline{n \; T \; i}) \; \overline{/ A})$
3. i $\Leftarrow$ *i* + 1
4. i $\Rightarrow$ (i < 2^n)/(2)

END

Primul pas initializeaza indexul *i*, pentru a stabili iesirea decodificatorului, care urmeaza sa fie evaluata. Operatiile inceteaza dupa evaluarea tuturor iesirilor decodificatorului DCD.

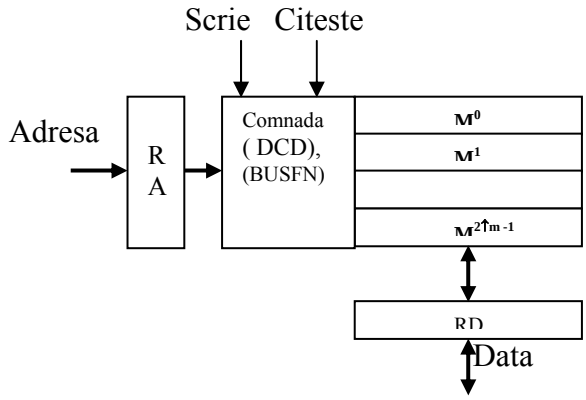
Pentru a intelege cum se evalueaza iesirile decodificatorului, sa considera cazul particular $n = 4$ si $i = 5$.

$$DCD_5 = \cap/((4\ T\ 5)/A), \overline{(4\ T\ 5)/A}$$
$$= \cap/((0,1,0,1)/A), \overline{(0,1,0,1)/A}$$
$$= \cap/((A_1, A_3), (\overline{A_0}, \overline{A_2})) = \overline{A_0}, A_1, \overline{A_2}, A_3$$

In cazul in care $A = A_0, A_1, A_2, A_3 = (0, 1, 0, 1)$ se obtine $DCD_5 = 1$. Celelalte iesiri DCD_j, pentru care $j \in \{0, 1, 2, \ldots, 2^{n-1}\} \cap j \neq 5$, vor lua valoarea 0.

2.3. Circuitul, BUSFN, de citire a unui cuvant dintr-o memorie M.

Se considera o memorie **M**, alcatuita din mai multe registre $\mathbf{M}^0$, $\mathbf{M}^1$, ……, $\mathbf{M}^{2\uparrow m-1}$. Memoria primeste informatiile de adresa de la un registru RA si efectueaza operatiile de citire/scriere prin intermediul unui registru RD.



Memoria M dispune de o unitate de comanda, care asigura controlul operatiilor de scriere si citire, pe baza semnalelor primite din exterior.

Operatiile de citire si scriere se pot descrie, la nivelul transferurilor intre registre, dupa cum urmeaza:

- **Citire**

  i. RA ← Adresa
  (i +1). RD ← BUSFN(**M**; DCD(RA)); Citeste = 1

- **Scriere**

  j. RA ← Adresa; RD ← Data
  (j +1). **M*** DCD(RA) ← RD; Scrie = 1

BUSFN(**M**; DCD(RA)) reprezinta o functie logica combinationala, care are ca argumente o matrice **M**, cu ρ2**M** linii si ρ1**M** coloane, pe de-o parte, si un vector DCD(RA), cu 2^m componente, pe de alta parte, unde ρ2**M** $= 2^m$.

Daca in locul vectorului DCD(RA) se utilizeaza un vector R[r], iar ρ1**M** este egal cu *p*, se poate obtine urmatoarea descriere AHPL:

UNIT: BUSFN(**M** ; R)

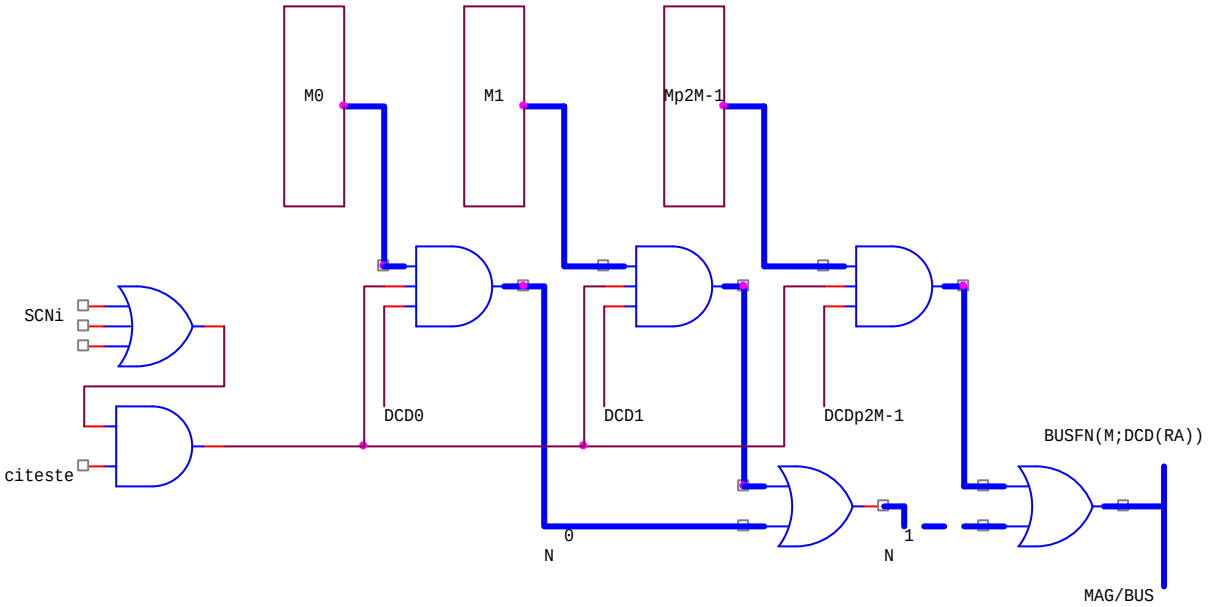
INPUTS: **M**[*r* : *p*]

OUTPUTS: BUSFN[p]

1. $\mathbf{N}^0 = \mathbf{M}^0 \cap R_0$
2. $i \Leftarrow 1$
3. $\mathbf{N}^i = (\mathbf{M}^i \cap R_i) \cup \mathbf{N}^{i-1}$
4. $i \Leftarrow i + 1$
5. $i \Rightarrow (i < r)/(3)$
6. BUSFN = $\mathbf{N}^{r-1}$

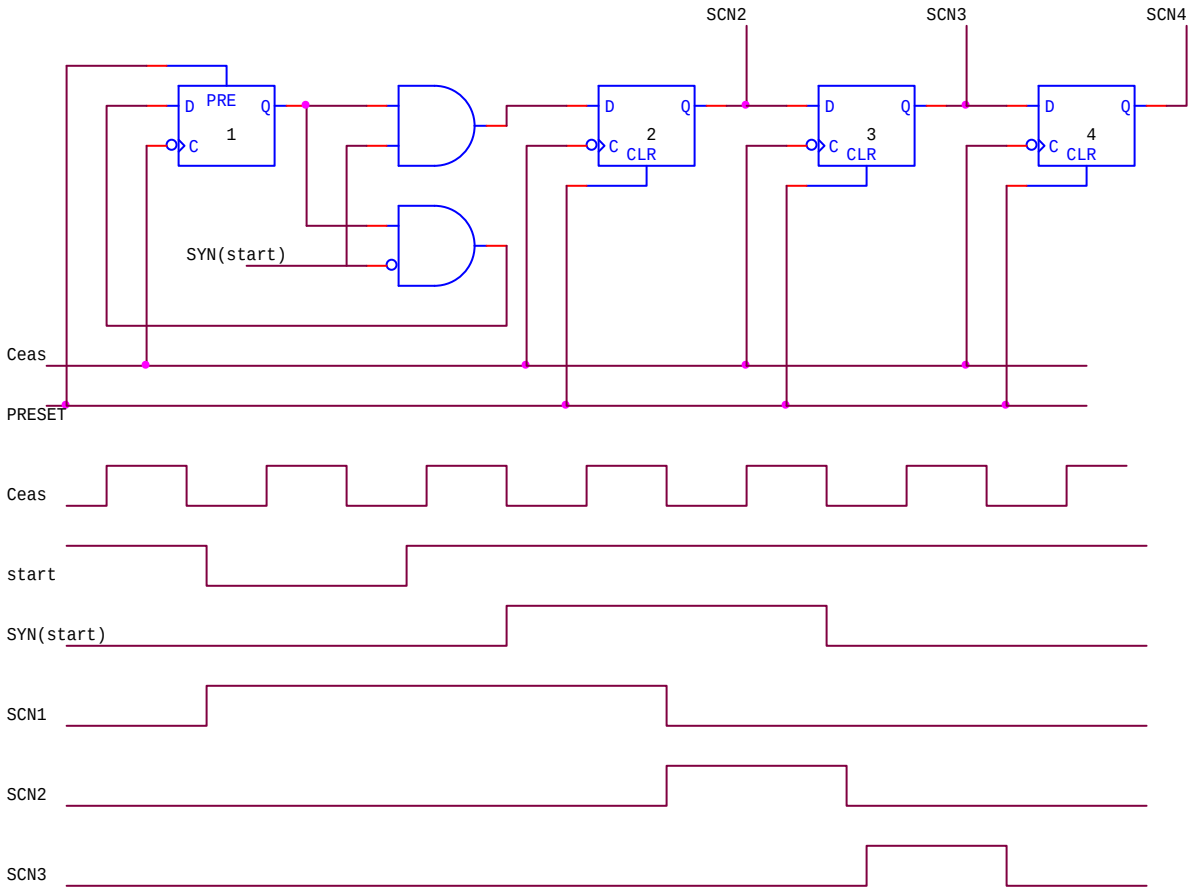
END

O schita a implementarii unitatii BUSFN, in contextul operatiei de citirere, este data mai jos.

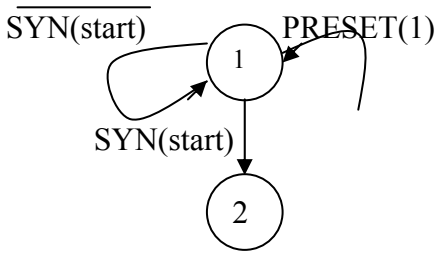


2. 4. Implementarea secventei start/reset (PRESET).

Implementarea primilor pasi dintr-o secventa AHPL, din cadrul descrierii unui modul a carui operare este lansata de semnalul *start*, este prezentata mai jos. Primul bistabil (1), cat si celelalte bistabile (2), (3), (4) ale schemei de comanda au fost fortate, printr-un semnal *reset*/PRE/CLR asincron, in starile $Q_1 = 1$ si, respectiv, $Q_2 = 0$, $Q_3 = 0$, $Q_4 = 0$. Comanda *reset*, in cazul de fata PRESET, are un caracter asincron si apare, in secventa AHPL, intre specificatiileENDSEQ si END, sub forma: PRESET(1), unde (1) constituie linia AHPL, la care se efectueaza saltul (un GO TO 1 asincron).



In continuare, operarea automatului cu stari complet decodificate, la nivelul primului bistabil, este prezentata sub forma diagramei de tranzitii:

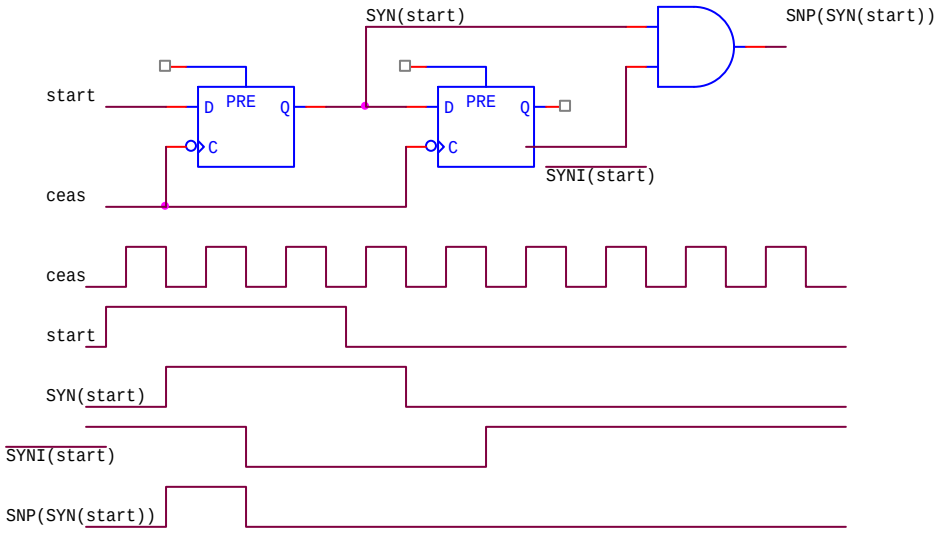


Dupa cum se constata, automatul va parasi starea (1) numai in conditiile in care semnalul SYN(*start*) va trece pe nivel ridicat. In caz contrar, starea (1) se va extinde pe un numar indefinit de perioade de ceas.

2.5. Sincronizarea semnalului *start*.

In cele prezentate mai sus, s-a vazut necesitatea existentei unui semnal de *start*, pentru activarea automatului de comanda al unui modul. De cele mai multe ori semnalul *start* este dat manual, ceea ce face ca el sa fie asincron cu ceasul sistemului si sa aibe o durata relativ mare.

Sincronizarea semnalului *start*, specificata prin operatorul SYN(*start*), se poate realiza cu ajutorul diferitelor scheme. Mai jos se prezinta una dintre acestea, impreuna cu diagramele de semnale.



Se poate usor observa ca, cel de-al doilea bistabilul reproduce sub forma negata, la iesirea Q, semnalul SYN(*start*), intarziat cu o perioada de ceas. Prin efectuarea produsului logic

intre SYN(*start*) si semnalul $\overline{\text{SYNI}(start)}$ se obtine un semnal de *start sincronizat*, de tip nivel, cu durata unei perioade de ceas: SNP(SYN(*start*)).

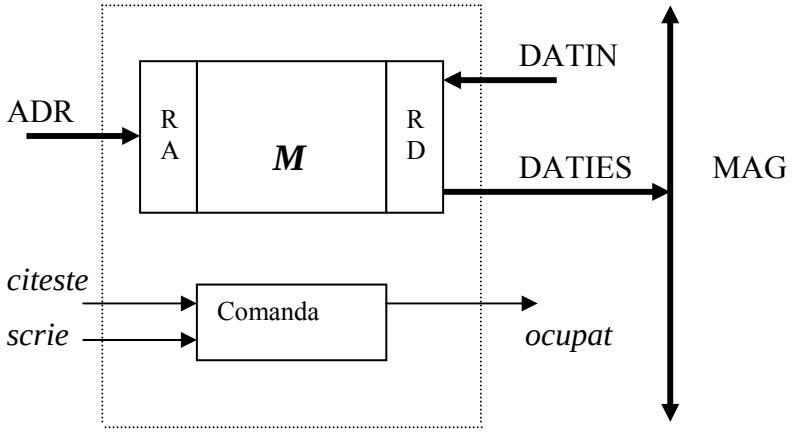
2.6. Comanda citirii instructiunii dintr-o memorie principala asincrona.

Memoriile principale ale calculatoarelor numerice reprezinta module, care opereaza asincron, de cele mai multe ori.

Secventa AHPL, pentru citirea unei instructiuni dintr-o memorie **M**, prevazuta cu registrele de adresa RA si registrul de date RD, in registrul de instructiuni RI, are urmatorul aspect:

2. RA ← CP /* CP este contorul programului
3. RD ← BUSFN(**M**;DCD(RA))
4. RI ← RD

In cazul unei memorii asincrone, al carei model logic este dat mai jos, adresele si datele presupun, pe de-o parte, folosirea unor magistrale: ADR, DATIN, DATIES, iar comanda, pe de alta parte, utilizarea unor semnale de control *citeste*, *scrie* si de stare *ocupat*.



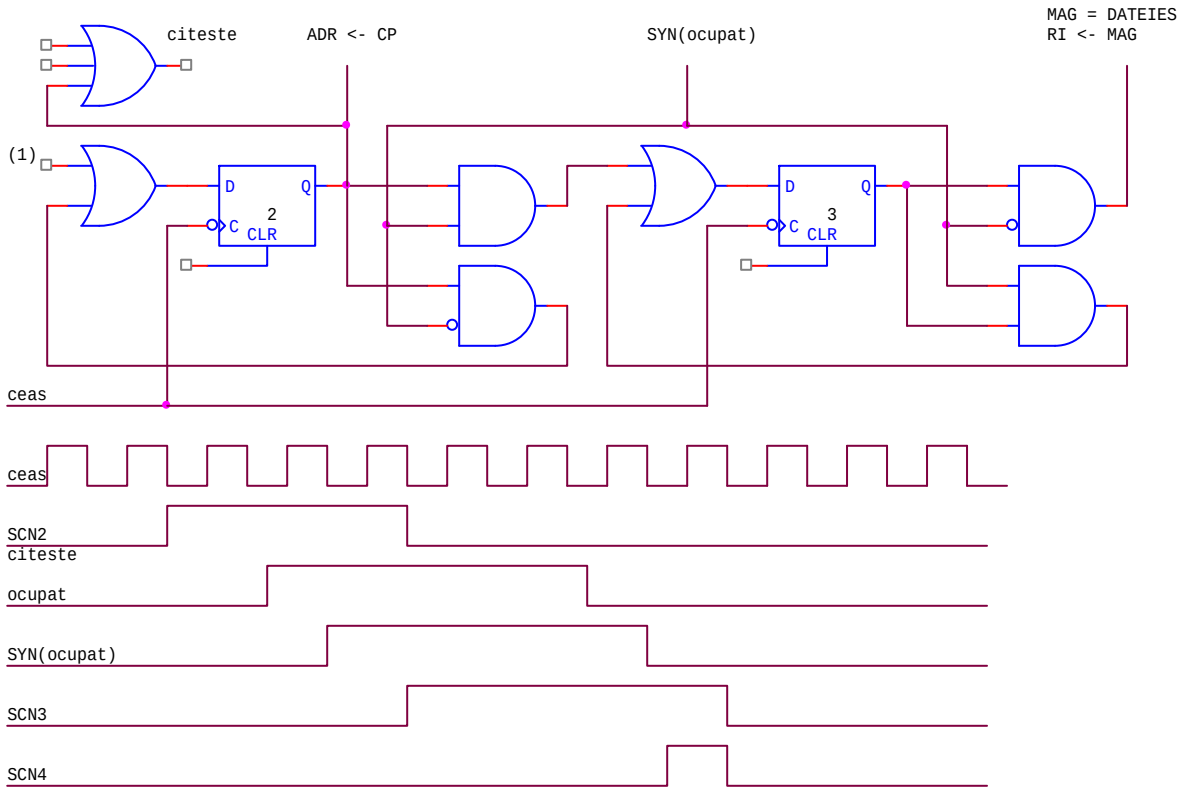
Modelul logic al modulului memoriei principale.

In aceste conditii secventa AHPL, de citire a unei instructiuni, din memorie, capata urmatorul aspect:

2. ADR ← CP; citeste = 1
   → $\overline{\text{SYN}(\text{ocupat})}$/(2)
3. nul
   → SYN(ocupat)/(3)

4. FARA INTARZIERE

MAG = DATIES; RI ← MAG



Se poate constata faptul ca, pasul (4), FARA INTARZIERE, este amorsat in ultima parte a pasului anterior (3), ceea ce conduce la castigarea unei perioade de tact.

2.7. Exemple de module descrise in AHPL.

2.7.1.Unitate de comanda pentru o masina unealta.

Se cere proiectarea unei unitati de comanda, pentru o masina unealta, care foloseste o memorie **ROM**[1024:18] pentru a stoca patru secvente, de cate 256 cuvinte x 18 biti, structurati sub forma a trei campuri, pentru a specifica pozitia uneltei in trei dimensiuni. Pozitia curenta a

uneltei este memorata intr-un registru PR[18], care furnizeaza, conform structurarii sale pe cele trei campuri, vectori binari, de cate sase biti, pentru trei convertoare numeric/analogice CN/, care comanda echipamentul de actionare al masinii unelte. Comunicarea cu operatorul se realizeaza cu ajutorul unui bistabil *start*/*stop* *ss* (controlat din exterior de catre semnalele de comanda *start* si *stop*) si al unui registru RSECV[2] (comandat din exterior prin semnalul SECV[2], in care se memoreaza numarul secventei, care se executa: 00, 01, 10, 11.

Prin fortarea in unu a bistabilului *ss*, se va lansa in executie secventa al carei numar este stocat in RSECV.

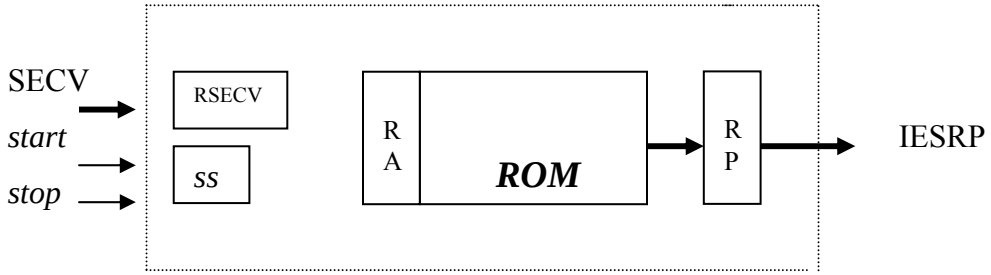
Daca, pe parcursul functionarii sistemului, operatorul forteaza in unu intrarea *stop*, bistabilul *ss* va lua valoarea zero, secventa curenta se va termina si in registrul RP se va stoca un vector binar cu 18 componente egale cu zero, iar unitatea de comanda va trece in starea initiala (1).

In cazul terminarii normale a secventei curente vor avea loc operatiile:

ss $\leftarrow$ 0 ; PR $\leftarrow$ 18 T 0 ; $\rightarrow$ (1)

Modificarile continuturilor lui ss si RSECV vor fi sincronizate cu ceasul.

Solutie: Pentru implementare, pe langa resursele mentionate mai sus, se va mai introduce un registru de adrese RA[10] de 10 biti. Cele patru secvente sunt notate cu A, B, C, D si vor avea, in zecimal, urmatoarele adrese de start: 0, 256, 512, 768. Frecventa ceasului este compatibila cu rata impusa de citirile efectuate de masina unealta. Un cuvant se citeste din memorie intr-o singura perioada de tact.



MODULE: Unitate_de _comanda_pentru_o_masina_unealta

MEMORY: **ROM**[1024:18]; RP[18]; RA[10]; RSECV[2]; *ss*

INPUTS: SECV[2]; *start*; *stop*

OUTPUTS: IESRP[18]

1. RSECV $\leftarrow$ SECV

   $\rightarrow (\overline{ss}, ss)/(1,2)$

2. RA $\leftarrow$RSECV$_0$, RSECV$_1$, 8 T 0

3. RP $\leftarrow$ BUSFN(**ROM**; DCD(AR))

4. RA $\leftarrow$ INC(RA)

   $\rightarrow ((\cap /RA_{2:9} \cap ss), \overline{ss}, \overline{(\cap /RA_{2:9} \cap ss)})/(5, 6, 3)$

5. *ss* $\leftarrow$ 0

6. RP $\leftarrow 18 \top 0$

   $\rightarrow (1)$

ENDSECV

ss $\leftarrow$ (1 ! 0) * (*start*, *stop*)

IESRP = RP

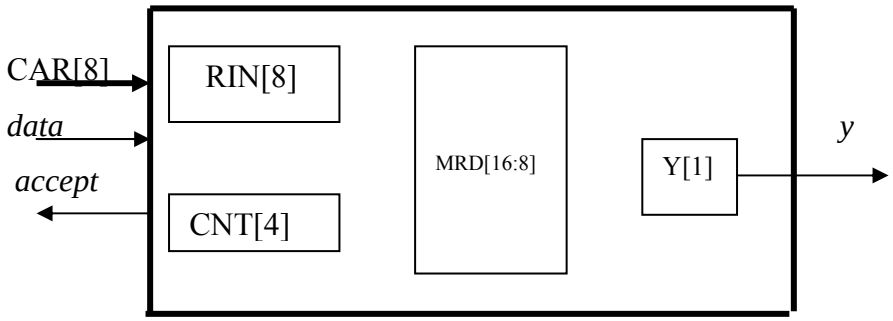
END

Plecand de la descrierea AHPL, de mai sus, sa se detalieze sectiunea de executie si sectiunea de comanda la nivelul schemelor cu porti, bistabile, registre, inclusiv semnalele de comanda.

2.7.2.Verificator de caractere duble.

Sa se proiecteze un modul numeric destinat verificarii caracterelor duble, prezente intr-un sir de caractere de cate 8 biti. Modulul poseda o linie de intrare, *data*, care va fi activata, pe nivel ridicat, de catre sursa de caractere, atunci cand un nou caracter este disponibil pe cele 8 linii de intrare CAR. Tranzitiile pe intrarile *data* si CAR[8] sunt sincronizate cu ceasul verificatorului de caractere. O linie de iesire, *accept*, va furniza un semnal de tip nivel, cu durata unei perioade de tact, dupa acceptarea caracterului de la intrarea CAR. O linie de iesire, *y*, conectata la un bistabil, Y, va fi activa pe nivel ridicat, in cazul in care cel mai recent caracter receptionat reprezinta o dublura a oricarui alt caracter din sirul de 16 caractere receptionate anterior. Iesirea *y* va trece in 0 atunci cand accept este fortat in 1, ceea ce indica receptionarea unui nou caracter. Intervalul intre spsirile a doua caractere succesive este suficient de mare pentru a permite verificarea seriala a existentei vreunei dubluri.

Sa se prezinte descrierea in AHPL a verificatorului de caractere duble.



Solutie: Verificatorul de caractere duble va contine un tablou de 8 registre de deplasare, a cate 16 biti fiecare. Acest tablou de registre va fi vazut ca o memorie **MRD** de 16 cuvinte a cate 8 biti: **MRD**[16:8]. In schema trebuie prevazute un registru de intrare RIN[8] si un contor de caractere CNT[4]. Caracterul curent, receptionat in RIN, este comparat cu fiecare caracter, stocat in MRD, in procesul rotirii caracterelor catre linia superioara. CNT va contoriza verificarea celor 16 caractere din **MRD.** Dupa terminarea verificarii ultimului caracter din **MRD**, caracterul curent, din RIN, este fortat in $\mathbf{MRD}^{15}$, in timp ce cuvantul, inregistrat de cel mai mult timp, care se afla in $\mathbf{MRD}^{0,}$ este pierdut. La detectarea unui caracter dublu semnalul *y* este fortat in 1.

Descrierea AHPL a modulului.

MODULE: Verificator_de_caractere_duble

MEMORY: MRD[16:8]; RIN[8]; CNT[4]; Y[1]

INPUTS: CAR[8]; *data*

OUTPUTS: *accept*; *y*

1. $\rightarrow (\overline{data},\ data)/(1, 2)$
2. *accept* = 1; $y \leftarrow$ 0;RIN $\leftarrow$ CAR; CNT $\leftarrow 4_{\top} 0$
3. Y * $\cup$/(RIN $\oplus$ $\overline{\text{MRD}^0}$) $\leftarrow$1; CNT $\leftarrow$ INC(CNT); **MRD** $\leftarrow$ ($\mathbf{MRD}^{1:15}$! $\mathbf{MRD}^0$)
   $\rightarrow$ ($\cap$/CNT, $\cap$/CNT)/(4,3)
4. **MRD** $\leftarrow$ ($\mathbf{MRD}^{1:15}$! RIN) ; $\rightarrow$(1)

ENDSEQUENCE

y =Y

END