Curs 7. Operarea in Banda de Asamblare (BA).

7.1. Introducere.

BA reprezinta o tehnica de implementare a procesoarelor, care presupune derularea simultana a mai multor instructiuni, ale unui program, aflate in faze diferite de executie.

BA consta in mai multe etaje sau segmente interconectate serial, fiecare avand functii specifice.

Timpul necesar deplasarii unei instructiuni cu un segment, in BA, poarta numele de ciclu al masinii. Datorita interconectarii in serie a etajelor, ciclul este dictat de catre cel mai lent etaj. De regula, un ciclu masina corespunde unei perioade de ceas (uneori doua si, mai rar, mai multor perioade de ceas). Ceasul poate avea una sau mai multe faze.

Productivitatea (Throughput) este data de numarul de instructiuni executate in unitatea de timp.

In conditii ideale, timpul de executie pentru o instructiune, in BA, este:

Timpul necesar executiei instructiunii fara BA/ Numarul segmentelor BA (performanta BA).

Cresterea de viteza (Speed-up), in cazul ideal, este egala cu numarul segmentelor BA. Datorita neechilibrarii perfecte a segmentelor, in ceea ce priveste timpii de executie, apare o regie (overhead), care reduce performanta BA.

Se poate spune ca BA asigura o reducere a timpului mediu de executie pe instructiune. Aceasta reducere poate fi vazuta sub trei forme:

1. O micsorare a numarului de cicluri de ceas pe instructiune (CPI), in cazul in care se pleaca de la un procesor, in care instructiunile se executra in mai multe cicluri de ceas.
2. O micsorare a perioadei ceasului, in cazul in care se considera un procesor care executa instructiunile intr-un singur ciclu masina.
3. O combinatie intre 1 si 2.

BA exploateaza paralelismul la nivelul unui program secvential, in situatia in care procesul este transparent pentru utilizator.

In fig. 7.1 se prezinta structura DLX in conditiile executiei instructiunii in mai multe cicluri de ceas.

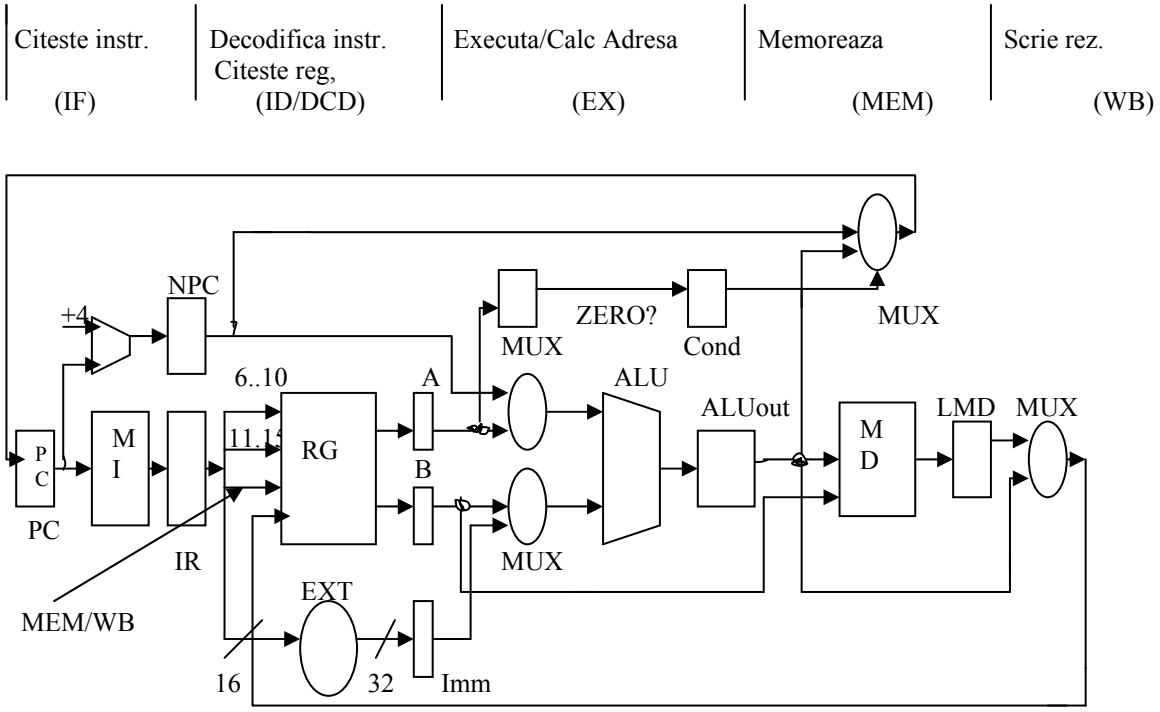

Fig. 7.1. Structura Unitatii de Executie in care sunt incluse Memoriile de instructiuni (MD) si de date (MD), in cazul procesorului DLX, care executa instructiunile in mai multe cicluri de ceas.

Fiecare segment al BA este prevazut cu registre pentru stocarea informatiei prelucrate sau necesare controlului.

Elementele de memorare sunt urmatoarele: MI, MD, RG/Regs, PC si registrele “temporare”: LMD, Imm, A, B, IR , NPC, cond Registrele temporare stocheaza valorile intre ciclurile de ceas ale instructiunii. Alte elemente de memorare sunt parti vizibile ale starii masinii si stocheaza valorile intre instructiunile succesive.

Implementarea Uex a DLX asigura efectuarea instructiunilor in 4 sau 5 cicluri de ceas. Daca se examineaza PC si RG se constata ca primul este utilizat in fazele IF si Mem, iar al doilea in fazele ID si WB, pentru scriere. Aceste operatii de "scriere inapoi" introduc o serie de probleme in operarea in BA.

In cadrul capitolului anterior s-a vazut ca:

1. Instructiunile de ramificare necesita pentru derulare 4 cicluri de ceas, iar celelalte necesita 5 cicluri . In conditiile in care frecventa de aparitie a instructiunilor de ramificare este de 12%, se obtine CPI = 4,88.
2. Implementarea de fata nu conduce nici la cea mai buna performanta, nici la o economie de hardware.
    2.1 CPI poate fi redus prin terminarea instructiunii lor UAL pe durata ciclului MEM. Ponderea instructiunilor UAL este de 44%, ceea ce va duce la CPI = 4,44. Astfel, rezulta o crestere a performantei: 4,88/4,44 = 1,1.
    2.2. Orice incercare de reducere a CPI va conduce la o crestere a perioadei ceasului T, deoarece vor fi necesare mai multe activitati in ciclul dat. Se va face un compromis intre perioada ceasului si numarul de cicluri CPIi.
    2.3. Se poate incerca implementarea Unitatii de Comanda (UC) cu un automat finit avand un numar redus de stari.
    2.4. O reducere de hardware se poate obtine utilizand o singura memorie pentru program si date, cat si o singura UAL pentru operatiile UAL si pentru incrementarea CP

7.2. Trecerea la Banda de Asamblare pentru DLX.

In acest caz, in fiecare ciclu de ceas se poate lansa o noua instructiune. Fiecare segment se va contine o instructiune intr-o faza data a derularii sale (fig. 7.2.)

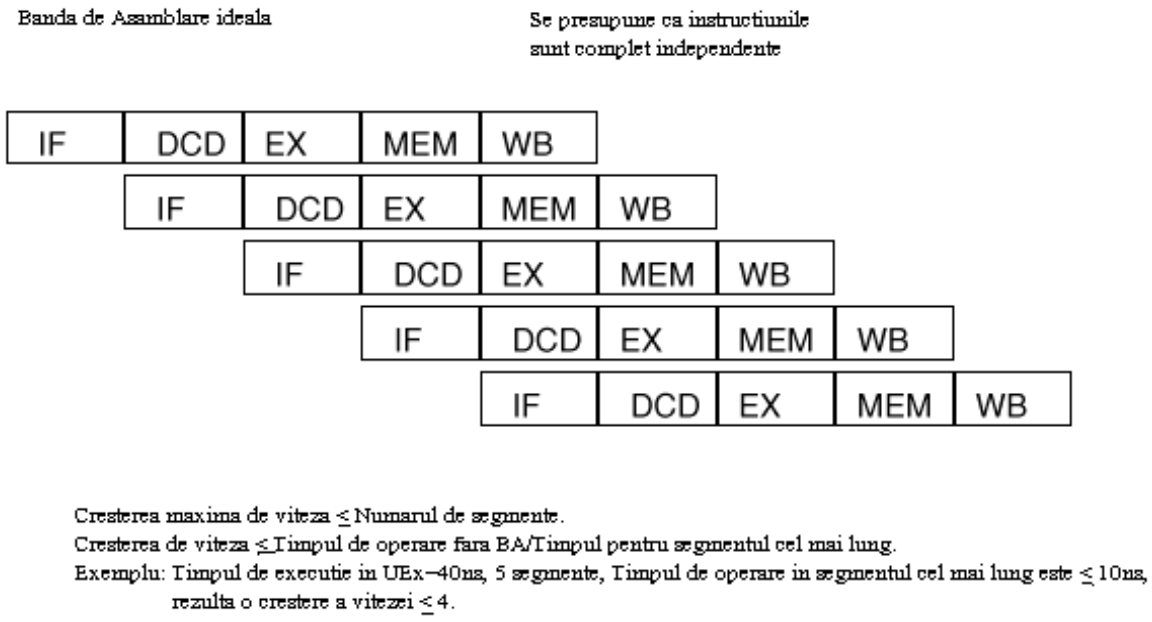

Fig. 7.2. In fiecare segment se va afla o instructiune intr-o faza data a derularii

Unitatile functionale principale sunt folosite in cicluri diferite. Suprapunerea fazelor unor instructiuni diferite va introduce putine conflicte. BA poate fi imaginata sub forma unei serii de UEx. deplasate in timp (fig.3).

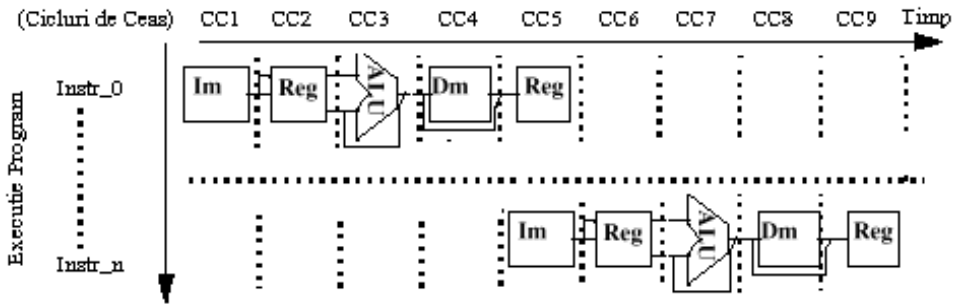

Fig. 7.3. Banda de Asamblare vazuta ca o serie de Uex. deplasate in timp.

Unitatea de Executie Uex. foloseste memorii diferite pentru instructiuni si date, pentru eliminarea unora dintre conflicte. Registrele generale RG/Reg sunt folosit in fazele ID/DCD si WB. In legatura cu aceste registre pot aparea conflicte daca se citeste/scrie din /in acelasi registru. Pentru noua instructiune, CP se modifica in faza IF. In cazul ramificarii, CP se modifica in faza MEM. Acestea impun ca ramificarile sa fie tratate intr-o maniera aparte.

Intrucat segmentele BA trebuie sa fie active in fiecare perioada de ceas, toate operatiile trebuie sa se sfarseasca intr-o perioada de ceas.

Functionarea in BA presupune ca, in fiecare segment sa fie disponibile toate informatiile necesare operatiilor de la acel nivel, inclusiv IR si NPC. Acest deziderat se realizeaza folosind la limita intre segmente registre/latch-uri destinatie/sursa marcate cu numele segmentelor sursa/destinatie

In fig. 7.4 este aratata structura DLX pentru operare in BA.

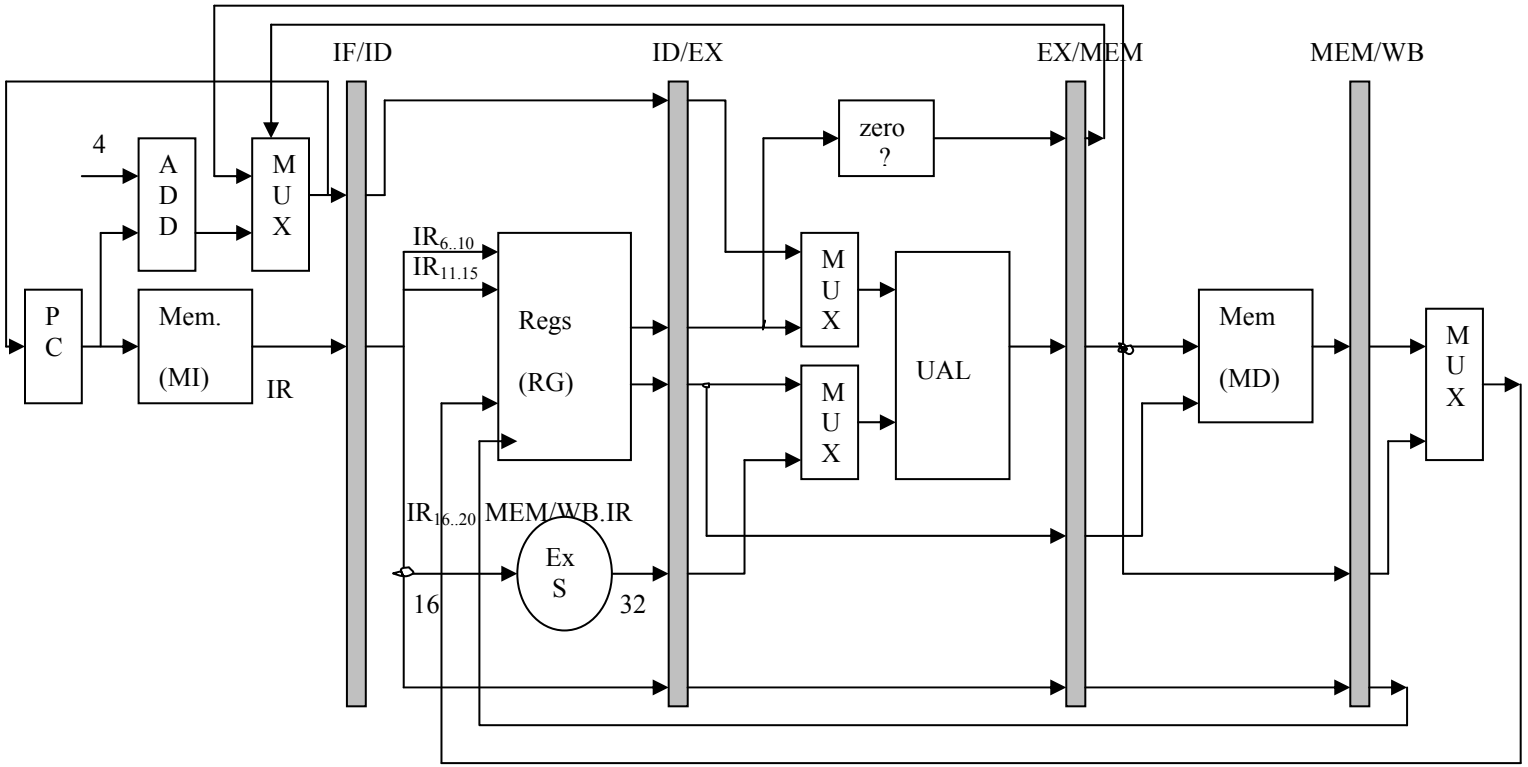

Fig. 7.4. Unitatea de Executie DLX in varianta Banda de Asamblare

Operatiile in cadrul sectiunilor Benzii de Asamblare

Sectiunea	Toate Instructiunile		
IF	IF/ID.IR ← MI[PC]; IF/ID.NPC ← (if EX/MEM.cond {EX/MEM.UALies} else {PC + 4}); IF/ID.PC ← (if EX/MEM.cond {EX/MEM.UALies} else {PC + 4});		
ID	ID/EX.A ← Regs[IF/ID.$IR_{6..10}$]; ID/EX.B ← Regs[IF/ID.$IR_{11..15}$]; ID/EX.NPC ← IF/ID.NPC; ID/EX.IR ← IF/ID.IR; ID/EX.Imm ← IF/ID.$(IR_{16})^{16}$ ## $IR_{16..31}$		
	Instructiunile UAL	**Instructiunile Incarca/Memoreaza**	**Instructiunile de Ramificare**
EX	EX/MEM.IR ← ID/EX.IR; EX/MEM.UALies ← ID/EX.A op ID/EX.B; sau EX/MEM.UALies ← ID/EX.A + ID/EX.Imm; EX/MEM.cond ← 0;	EX/MEM.IR ← ID/EX.IR; EX/MEM.UALies ← ID/EX.A + ID/EX.Imm; sau EX/MEM.cond ← 0; EX/MEM.B ← ID/EX.B	EX/MEM.UALies ← ID/EX.NPC + ID/EX.Imm; sau EX/MEM.cond ← (ID/EX.A op 0);

Sectiunea	Instructiunile UAL	Instructiunile Incarca/Memoreaza
MEM	MEM/WB.IR ← EX/MEM.IR; MEM/WB. UALies← EX/MEM.UALies;	MEM/WB.IR ← EX/MEM.IR; MEM/WB.LMD ← MD[EX/MEM.UALies]; sau MD[EX/MEM.UALies] ← EX/MEM.B
WB	Regs[MEM/WB.$IR_{16..20}$] ← MEM/WB.UALies; sau Regs[MEM/WB.$IR_{11..15}$] ← MEM/WB.UALies;	Regs[MEM/WB.$IR_{11..15}$] ← MEM/WB.LMD;

Observatii privind BA:

- Banda de Asamblare mareste productivitatea, adica numarul de instructiuni executate in unitatea de timp.
- Timpul de executie (latenta) a unei instructiuni nu se micsoreaza, ci se poate mari datorita regiei inerente. Latch-urile necesita un timp de stabilire si introduc o intarziere in aparitia semnalului la iesire. Trebuie avuta in vedere intarzierea sau/si alunecarea ceasului. Cand ciclul ceasului este comparabil cu regia introdusa de latch-uri + alunecarea ceasului, nu mai ramane timp pentru operatii utile in cadrul ciclurilor.

Exemplu: Se considera o masina fara BA, avand un ceas de 10 ns, in conditiile in care pentru operatiile UAL si ramificari sunt necesare 4 cicluri de ceas, iar pentru operatiile cu memoria 5 cicluri de ceas. Frecventele relative ale acestor operatii sunt urmatoarele: 40%, 20% si 40%. Se admite ca, la implementarea in BA, se mai adauga 1ns, datorita alunecarii ceasului.

Timpul de executie fara BA = Ciclul ceasului x CPI

= 10 ns x ((40% + 20%) x 4 + 40% x 5)

= 44 ns.

Timpul de executie in BA = 11 ns.

Cresterea de viteza = 44/11 = 4 ori

7.3. Problemele Benzii de Asamblare: Hazardele.

Hazardele impiedica executia urmatoarei instructiuni din flux. Exista trei clase de hazarde:

1. **Hazarde structurale**. Acestea rezulta ca urmare a conflictelor care apar atunci cand hardware-ul nu poate suporta toate combinatiile posibile ale executiei instructiunilor in regimul de suprapunere ale diverselor faze ale acestora.

2. **Hazardele de date** apar atunci cand o instructiune depinde de rezultatul instructiunii anterioare, intr-un mod impus de suprapunerea instructiunilor in BA.
3. **Hazardele de control** rezulta datorita plasarii in BA a instructiunilor de ramificare, cat si a altor instructiuni, care modifica PC.

Hazardele structurale.

Operarea in BA necesita structurarea in acest scop a unitatilor functionale si duplicarea resurselor pentru a permite toate combinatiile posibile de instructiuni. Daca unele combinatii de instructiuni nu se pot executa, rezulta hazardul structural. Cauzele care conduc la hazardele structurale pot fi urmatoarele:

- unele unitati functionale nu sunt complet organizate in BA;
- secventa de instructiuni in cauza nu se poate executa la o frecventa de o instructiune pe ciclu;
- unele resurse nu au fost multiplicate suficient pentru a permite toate combinatiile posibile de instructiuni, de exemplu RG/Regs poseda numai o singura intrare de scriere, in timp ce, uneori, sunt necesare simultan doua porturi de intrare.

Cand apare o asemenea situatie BA va intarzia, cu unul sau mai multe cicluri, introducerea instructiunii urmatoare. Este ca si cum s-ar introduce o instructiune neoperationala. Aceasta situatie mai poarta numele de *intarziere*, *stall*, *pipeline-bubble* sau simplu *bubble*. Rezolvarea pe aceasta cale a hazardurilor structurale va conduce la cresterea valorii lui CPI.

Ca exemplu, se poate da prezenta unei singure memorii, atat pentru instructiuni, cat si pentru date (fig. 7.5). Pentru rezolvare se va introduce o *intarziere*/*stall* (fig. 7.6).

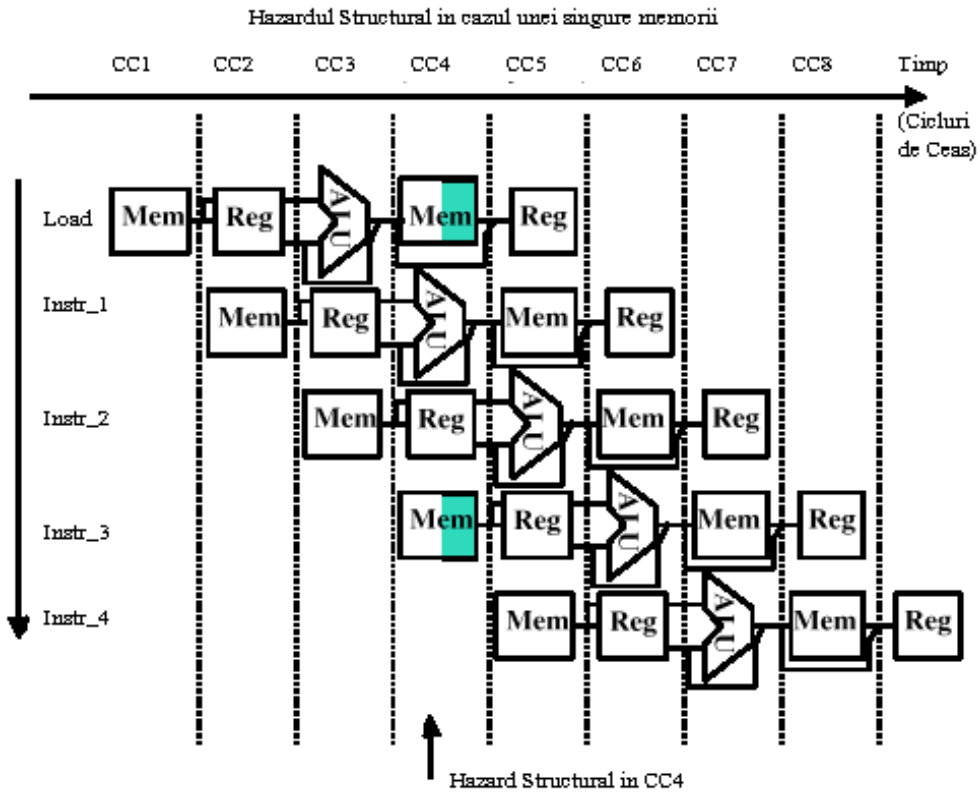

Fig. 7.5. Hazard Structural in cazul unei singure memorii.

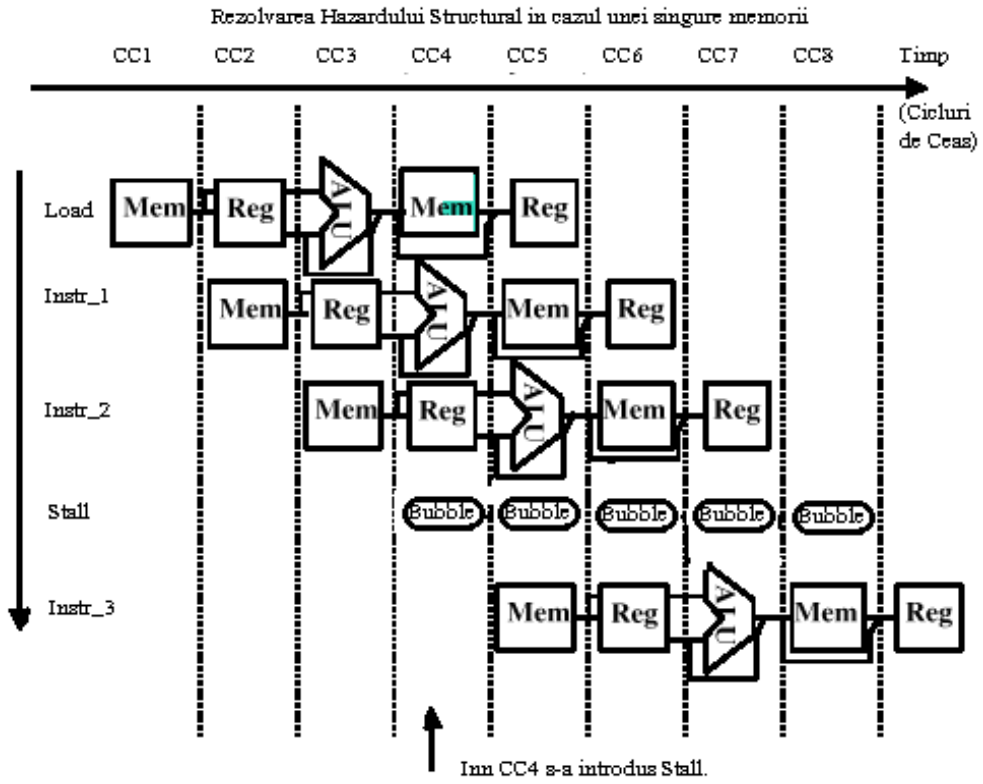

Fig. 7.6. Rezolvarea Hazardului Structural, in cazul unei singure memorii.

Performanta BA cu intarzieri.

Intrazierile (stall), introduse pentru a rezolva hazardele in BA, conduc la degradarea performantei sistemului.

Cresterea de viteza cu BA.

$$\textit{Cresterea de viteza cu BA} = \frac{\textit{Timpul mediu de executia a instructiunii fara BA } (TMI_{FBA})}{\textit{Timpul mediu de executia a instructiunii cu BA } (TMI_{CBA})}$$

$$= \frac{CPI_{FBA} \times T_{FBA}}{CPI_{CBA} \times T_{CBA}} = \frac{CPI_{FBA}}{CPI_{CBA}} \times \frac{T_{FBA}}{T_{CBA}}$$

unde T este perioada ceasului.

Operarea in BA *conduce la reducerea valorilor CPI si T*. In mod ideal $CPI_{CBA} = 1$ (ceea ce se va nota cu $CPII_{CBA}$)

a) Se considera cazul in care are loc reducerea numarului de perioade de ceas pe instructiune, la aceeasi perioada a ceasului:

CPI_{CBA} = $CPII_{CBA}$ + Cicluri de Intarziere pe Instructiune cu BA ($CIPI_{CBA}$)

= 1 + $CIPI_{CBA}$

Daca se neglijeaza regiile pentru BA si se considera etajele echilibrate**:**

$$\textit{Cresterea de viteza} = \frac{CPI_{FBA}}{1 + CIPI_{CBA}}$$

Un caz simplu si important este acela in care toate instructiunile necesita acelasi numar de cicluri de ceas, care este egal cu numarul de segmente ale BA, denumit si *adancimea* BA. Astfel, CPI_{FBA} = *adancimea* BA.

$$\textit{Cresterea de viteza cu } \text{BA} = \frac{\textit{adancimea } \text{BA}}{1 + \text{CIPI}_{\text{CBA}}}$$

Daca nu sunt intarzieri, se poate considera ca viteza de executie are o crestere egala cu numarul de segmente ale BA.

b) Daca se considera ca introducerea BA conduce la o micsorare a perioadei ceasului, atunci, presupunand ca $\text{CPI}_{\text{FBA}} = \text{CPII}_{\text{CBA}} = 1$, se va obtine:

$$\textit{Cresterea de viteza cu } \text{BA} = \frac{\text{CPI}_{\text{FBA}}}{\text{CPI}_{\text{CBA}}} \times \frac{\text{T}_{\text{FBA}}}{\text{T}_{\text{CBA}}} = \frac{1}{1 + \text{CIPI}_{\text{CBA}}} \times \frac{\text{T}_{\text{FBA}}}{\text{T}_{\text{CBA}}}$$

In cazul in care segmentele BA sunt perfect echilibrate nu apare regie ceea ce face ca T_{CBA} sa fie mai mic decat T_{FBA} cu un factor egal cu *adancimea* BA:

$$\textit{adancimea } \text{BA} = \frac{\text{T}_{\text{FBA}}}{\text{T}_{\text{CBA}}}$$

Ceea ce consuce la o crestere de viteza:

$$\textit{Cresterea de viteza cu } \text{BA} = \frac{1}{1 + \text{CIPI}_{\text{CBA}}} \times \frac{\text{T}_{\text{FBA}}}{\text{T}_{\text{CBA}}} = \frac{1}{1 + \text{CIPI}_{\text{CBA}}} \times \textit{adancimea } \text{BA}$$

Hazarde de Date.

Un efect important al operarii in BA se refera la modificarea temporizarii relative a instructiunilor prin suprapunerea lor. Aceasta circumstanta este de natura sa conduca la aparitia hazardelor de date si de control.

Hazardul de date apare atunci cand BA modifica ordinea de accese citeste/scrie la operanzi, astfel incat ele apar in alta succesiune decat in executia secventiala a instructiunilor pe un calculator fara BA.

Clasificarea hazardelor de date.

Se considera doua instructiuni *i* si *j*, astfel incat *i* precede *j* ($i \prec\prec j$). Hazardele de date posibile sunt urmatoarele:

- **Citeste Dupa Scriere** (**RAW** – Read After Write): *j* incearca sa citeasca o data inainte ca aceasta sa fie scrisa de catre *i*. Astfel, se citeste o valoare veche. Aceasta este un exemplu de dependenta de date, care poate fi solutionata partial prin *ocolire* (*bypass*).
- **Scrie Dupa Scriere** (**WAW** – Write After Write): *j* incearca sa scrie un operand inainte ca acesta sa fie scris de catre *i*, ceea ce face ca rezultatul lui *i* sa ramana scris. Acesta este un exemplu de antedependenta si se solutioneaza prin efectuarea scrierilor numai in cadrul aceluiasi segment (WB), desi in unele cazuri datele sunt disponibile intr-un segment anterior.
- **Scrie Dupa Citire** (**WAR** – Wraite After Read): *j* incearca sa scrie la o destinatie inainte ca aceasta sa fie citita de catre *i*, ceea ce face ca *i* sa obtina o valoare incorecta. Acesta este un exemplu de dependenta de date de iesire si se solutioneaza prin BA de lungime fixa. WAR si WAW reprezinta restrictii de memorare, care pot fi solutionate prin redenumirea registrelor.

Exemple de hazarde de date.

Fie secventa urmatoare de instructiuni intr-o BA:

add **r1**, r2, r3

sub r4, **r1**, r5

and r6, **r1**, r7

or r8, **r1**, r9

xor r10, **r1**, r11

Toate instructiunile care urmeaza dupa instructiunea *add* (r1 ← r2 + r3), utilizeaza valoarea lui r1, care va fi actualizata abia in ciclul WB.

In figura 7.7 se prezinta diagrama dependentelor retardate corespunzatoare secventei de instructiuni de mai sus, cauzate de scrierea cu intarziere a rezultatului adunarii in r1.

Instructiunile *sub* si *and* ridica probleme mari. Instructiunea *or* s-ar putea rezolva prin efectuarea scrierii in prima jumatate a ciclului si a citirii in cea de-a doua jumatate a acestuia. Instructiunea *xor* se executa corect.

Inlocuirea insertiei de *stall*-*uri/intarzieri*, in cazul hazardelor de date, prin tehnicile: *avansarii* (forwarding), *ocolirii* (bypassing) sau a *scurt-circuitarii*.

Rezultatul instructiunii *add* este necesar pentru *sub*, dupa producerea acestuia. Astfel, rezultatul trebuie deplasat, din locul in care a fost produs, EX/MEM.Reg, in segmentele in care este necesar.

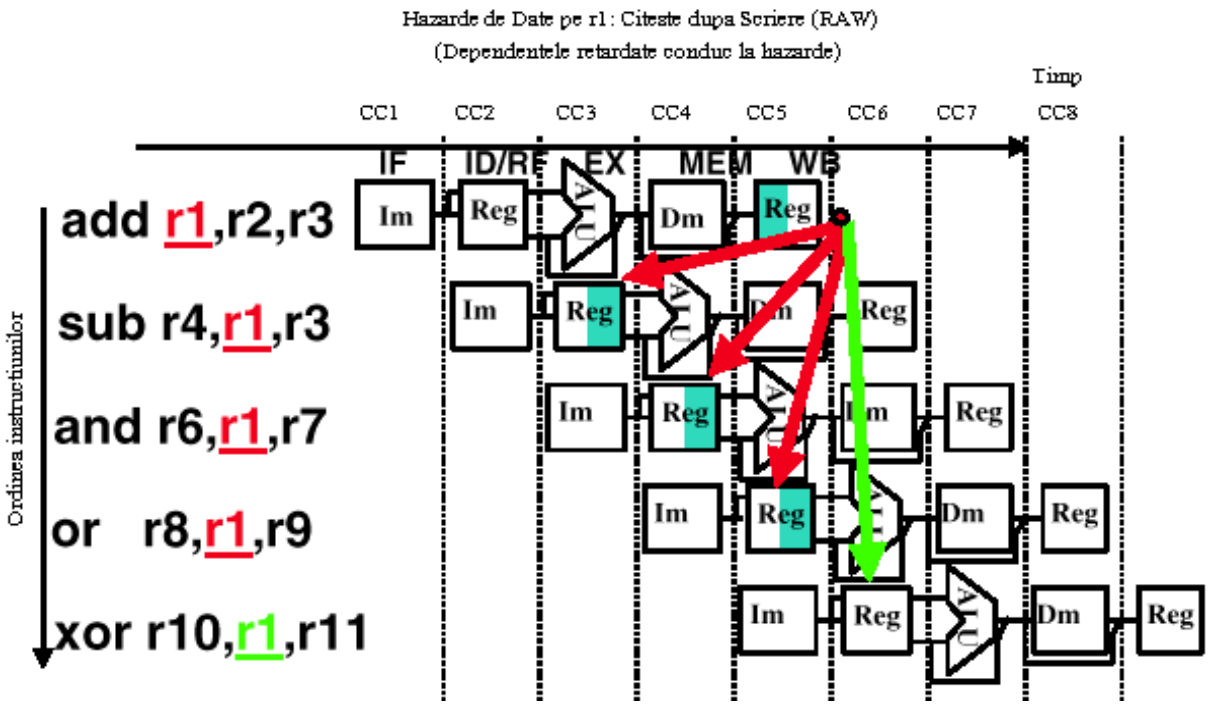

Fig. 7.7. Hazarde de Date pe r1: Citire dupa Scriere (RAW).

In figura 7.8 se prezinta solutia de eliminare a hazardului de date prin tehnica avansarii (forwarding).

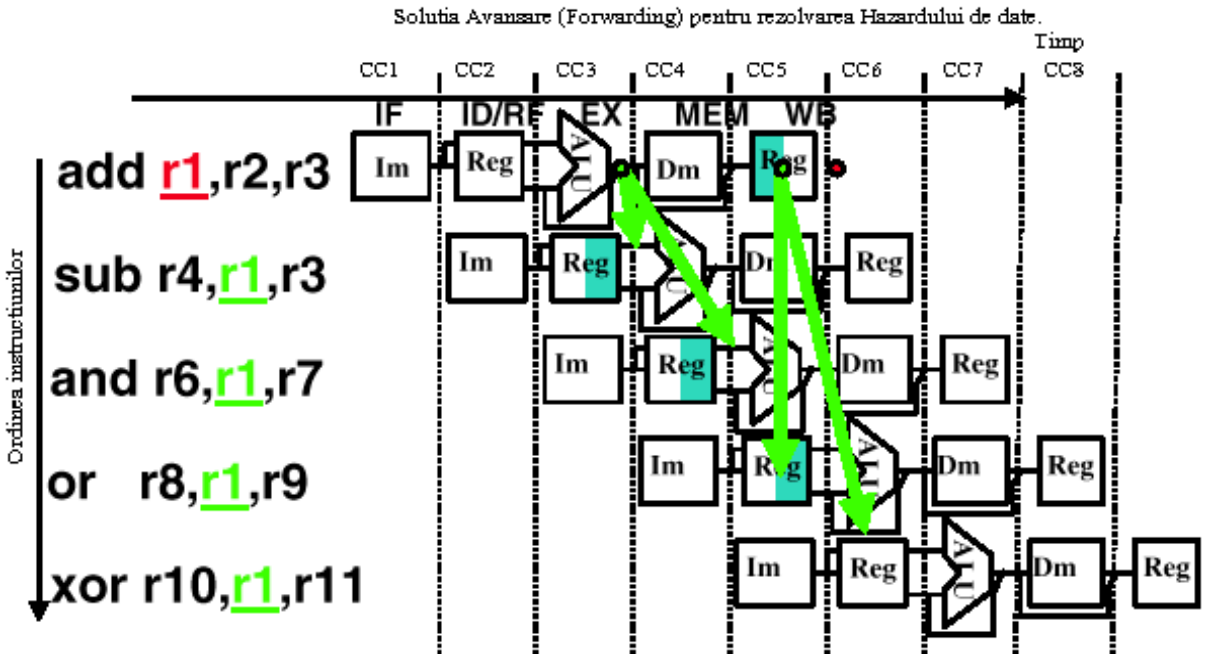

Fig. 7.8. Solutia pentru rezolvarea hazardului de date prin tehnica Avansare (Forwarding).

Daca hardware-ul de ocolire (bypass) detecteaza faptul ca operatia anterioara a scris un registru, care corespunde unei surse pentru operatia UAL curenta, logica de control selecteaza rezultatul obtinut, folosind ocolirea (bypassing). In acest fel rezultatul este fortat la intrarea UAL, in locul valorii ce s-ar fi citit din registru.

Hazarde de date care necesita intarzieri (*stall*).

In practica se intalnesc hazarde de date care nu se pot rezolva prin tehnica avansarii (fig. 7.9). Fie secventa urmatoare de instructiuni:

lw **r1**, 0(r2)

sub r4, **r1**, r6

and r6, **r1**, r7

or r8, **r1**, r9

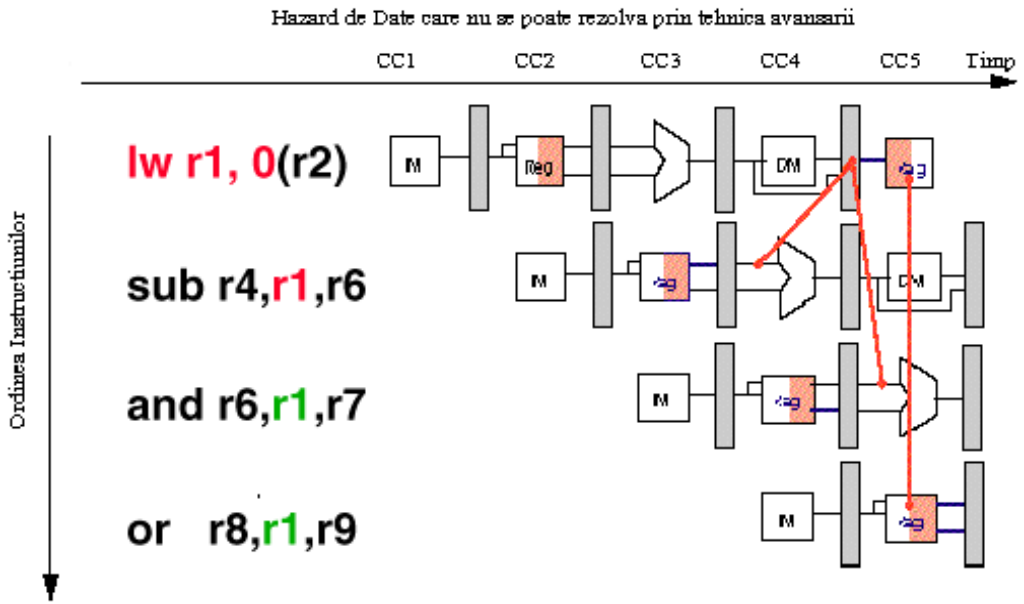

Fig. 7.9. Hazard de Date, care nu se poate rezolva prin tehnica avansarii.

Dupa cum se observa *lw* furnizeaza data la sfarsitul lui CC4 (MEM), in timp ce *sub* solicita data la inceputul aceluiasi ciclu. Ocolirea ar trebui sa se realizeze invers in timp, ceea ce nu este posibil. In acest caz este necesar un hardware suplimentar, care are rolul de a interbloca BA. Acesta detecteaza conditia de hazard si blocheaza BA, prin intarziere (stall - bubble), pana cand hazardul este eliminat (fig. 7.10). In aceste conditii CPI creste.

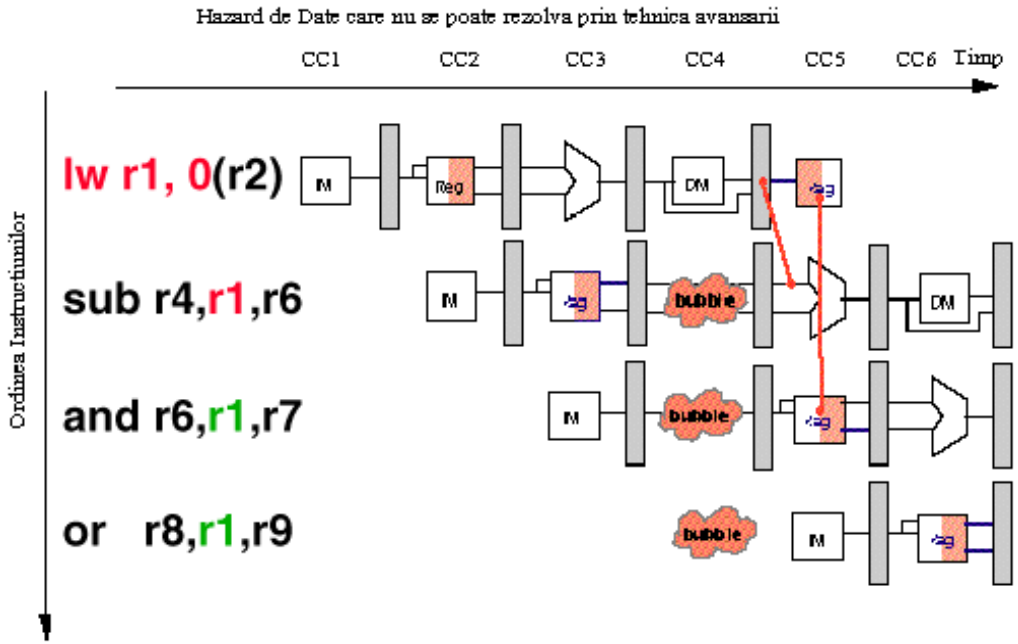

Fig. 7.10. Introducerea de intarzieri (stall/bubble) pentru rezolvarea hazardului de date introdus de *lw*.

Planificarea efectuata de catre compilator pentru a evita hazardele de date.

Generarea codului, pentru calculul expresiei a =b + c, conduce la necesitatea introducerii intarzierilor in BA.

lw r1, b	IF	ID	EX	MEM	WB				
lw r2, c		IF	ID	EX	MEM	WB			
add r3, r1, r2			IF	ID	stall	EX	MEM	WB	
sw a, r3				IF	stall	ID	EX	MEM	WB

Instructiunea *add* trebuie intarziata pentru a permite incarcarea lui c. Instructiunea *sw* nu mai trebuie intarziata deoarece, prin ocolire, rezultatul UAL poate fi trecut direct la intrarea memoriei, in vederea stocarii.

Compilatorul trebuie sa planifice operarea in BA, pentru a elimina intarzierile (*stall*), prin rearanjarea codului. Astfel, nu trebuie sa se genereze cod cu incarcare urmata de utilizarea imediata a registrului destinatie. Aceasta tehnica poarta numele de "*planificarea* BA" (*pipeline scheduling*) sau "*planificarea instructiunii*" (*instruction scheduling*).

Exemplu: Sa se genereze un cod pentru a evita blocarile in urmatoarele secvente:

a = b + c; d = e – f

Codul planificat este urmatorul:

lw Rb, b
lw Rc, c
lw Re, e : interschimb de instructiuni
add Ra, Rb, Rc
lw Rf, f
sw a, Ra; interschimb de instructiuni sw/lw
sub Rd, Re, Rf
sw d, Rd

Compilatoarele moderne planifica, in acelasi bloc de baza, o secventa de instructiuni, fara tansferuri in/out, cu exceptia inceputului si sfarsitului secventei. Planificarea unor asemenea blocuri de baza este mult simplificata, deoarece fiecare instructiune din bloc va fi executata, din moment ce prima instructiune a fost executata. In figura 7.11 se prezinta in %, pentru diferite programe rezultatele planificarii/neplanificarii instructiunilor de incarcare

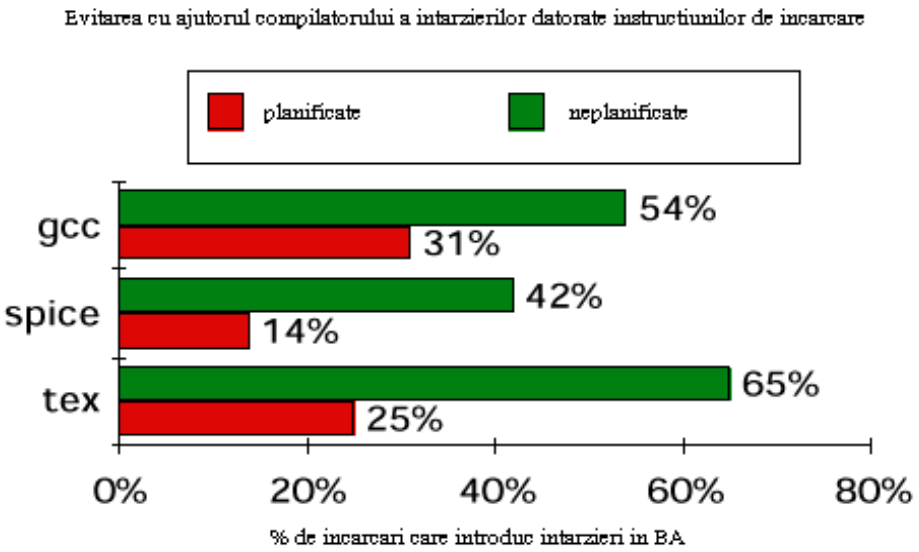

Fig. 7.11. Rezultatele obtinute, cu ajutorul compilatorului, privind instructiunile de incarcare, care introduc intarzieri.

Din cele prezentate mai sus, rezulta ca hardware-ul BA trebuie modificat pentru implementarea tehnicii avansare (*forwarding*). In figura 7.12. se prezinta modificarile aduse multiplexoarelor de la intrarea UAL.

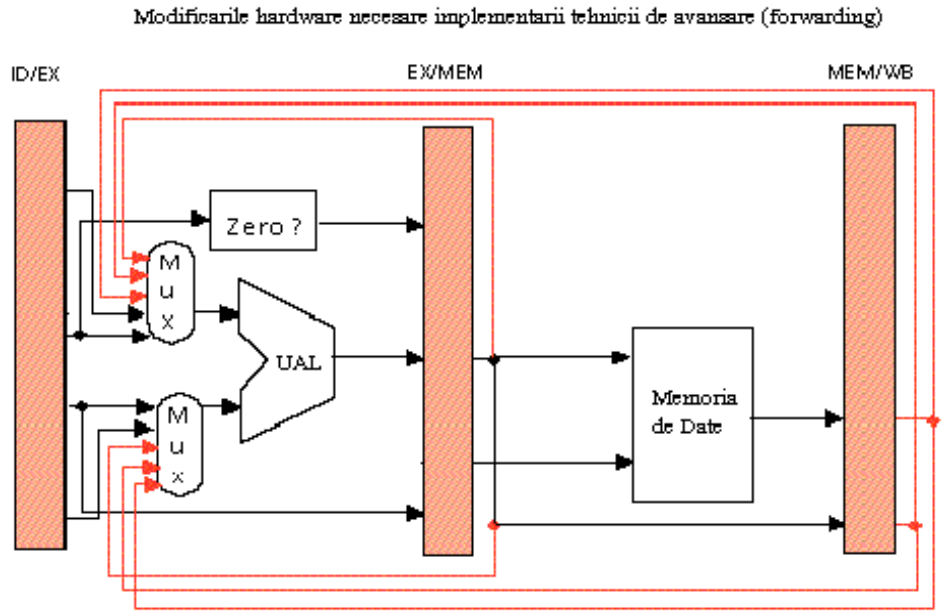

Fig. 7.12. Modificarile hardware impuse de implementarea tehnicii de avansare (*forwarding*).

Hazardele de Control.

Hazardele de Control pot conduce la reducerea performantei BA intr-o masura mai mare decat Hazardele de Date. Atunci cand o instructiune de ramificare (Br/Beq) este executata, in functie de indeplinirea/neindeplinirea conditiei, ramificarea este/nu este realizata. Daca ramificarea nu are loc, in mod normal, PC nu se va modifica pana la sfarsitul segmentului MEM, dupa calculul adresei efective si dupa efectuarea comparatiei.

Cea mai simpla metoda consta in a introduce intarziere in BA, in momentul in care se detecteaza o instructiune de ramificare, pana ce se ajunge la segmentul MEM, care determina noua valoare pentru PC. O astfel de solutie este ilustrata in figura 7.13.

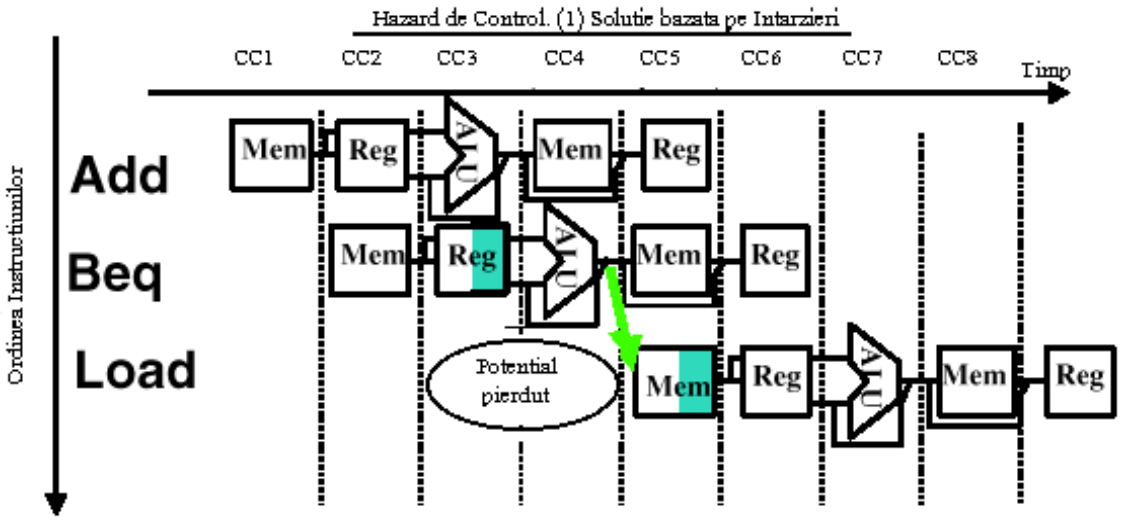

Fig. 7.13. Hazard de Control. Solutie bazata pe introducerea de Intarzieri in BA.

Decodificarea instructiunii de ramificare (Br) se realizeaza in segmentul ID. Primul segment,IF, al instructiunii, care urmeaza dupa Br, trebuie repetat imediat ce se stie rezultatul ramificarii. Astfel, primul ciclu IF este de intarziere/*stall,* deoarece nu efectueaza o activitate utila. Acest *stall* se poate implementa prin fortarea in zero a registrului IF/ID pentru trei cicluri. Daca ramificarea nu are loc, repetarea segmentului IF nu este necesara intrucat urmeaza sa fie executata instructiunea deja citita.

In conditiile in care instructiunile Br au o pondere de 30%, CPI real va creste, ajungand sa fie egal cu 2. Pentru a evita, pe cat este posibil, aceasta situatie se propun urmatoarele solutii:

- stabilirea cat mai devreme, in BA, a faptului ca ramificarea va avea loc sau nu va avea loc;
- in cazul efectuarii ramificarii, calcularea noii valori pentru PC, cat mai devreme posibil.

Astfel, ciclurile de intarziere necesare pentru evitarea Hazardului de Control, legat de instructiunile de ramificare, se pot reduce, ca numar, prin efectuarea testului ZERO? si prin calculul adresei tinta in segmentul ID, al BA.

Ambele solutii se vor implementa de o astfel de maniera incat sa se introduca o intarziere de un singur ciclu.

In figura 7.14 este prezentata solutia bazata pe predictia ramificarii. In cazul in care directia derularii programului a fost corecta nu se pierde nici un ciclu, la executia instructiunii Br. Daca directia a fost gresita, se va pierde un ciclu, prin suprimare instructiunii lansate si trecerea la noua directie. In situatia in care probabilitatea alegerii unei directii corecte este de 50%, numarul de cicluri pentru instructiunile Br va fi calculat astfel: $CPI_{Br} = 1 \times 0,5 + 2 \times 0,5 = 1,5$.

Considerand ca instructiunile de ramificare au o pondere de 20% si ca celelalte instructiuni se executa intr-un singur ciclu se va obtine:

$$CPI = 1,5 \times 0,2 + 1 \times 0,8 = 1,1.$$

In aceste conditii cresterea numarului mediu de cicluri pe instructiune va fi de 10%

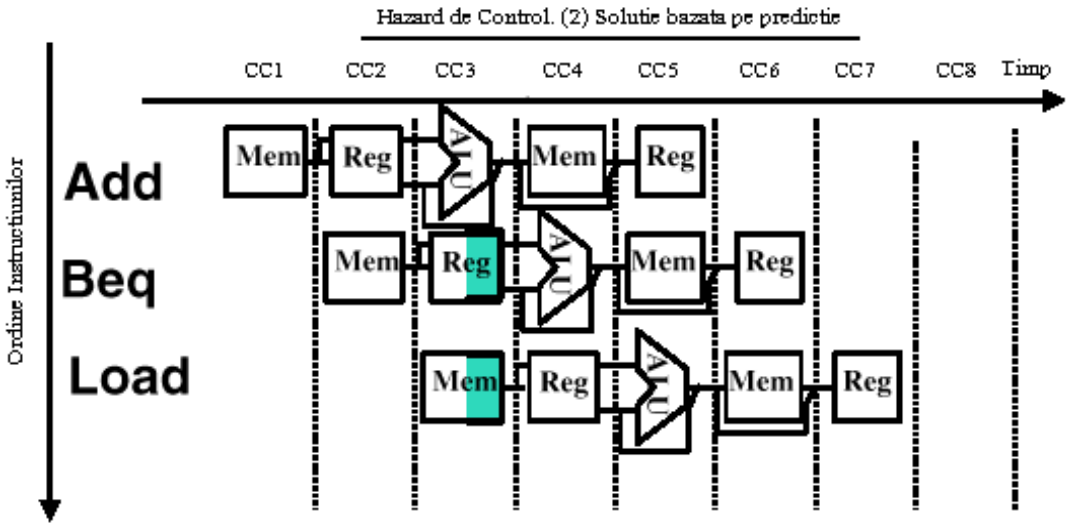

Fig. 7.14. Hazard de Control. Solutie bazata pe predictie (predictie corecta).

In figura 7.15 se prezinta o solutie bazata pe intarzierea ramificarii, prin introducerea unei instructiuni dupa instructiunea Add. Astfel, ramificarea are loc dupa lansarea acestei noi instructiuni, "neutre", din punctul de vedere al deciziei. In aceste conditii impactul instructiunii Br asupra lui CPI este 0 cicluri de ceas, daca se reuseste sa se gaseasca instructiunea potrivita, care va fi plasata dupa Add. Solutia nu este foarte utila intrucat se mai lanseaza o instructiune in secventa data initial.

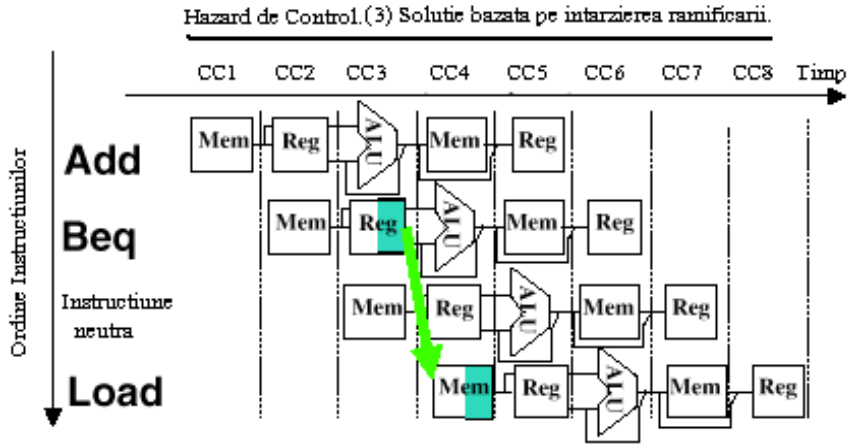

Fig. 7.15. Hazard de Control. Solutie bazata pe intarzierea ramificarii.

In figura 7.16 se prezinta modificarile aduse structurii din fig. 7.4 in vederea rezolvarii hazardului de Control datorat instructiunilor de ramificare.

In aceasta noua structura, pentru instructiunile Br in cadrul ciclurilor IF, ID,…, WR vor avea loc urmatoarele transferuri:

Segmentul BA	**Instructiunea Br**
IF	IF/ID.IR ← Mem[PC]
	IF/ID.NPC ← (PC + 4)
	PC ← (if cond[Regs{IF/ID.IR$_{6..10}$}]{IF/ID(NPC + (IR$_{16}$)^{16} # # IR $_{16..31}$)} else (PC + 4))
ID	ID/EX.A ← Regs[IF/ID.IR$_{6..10}$]; ID/EX.B ← Regs[IF/ID.IR$_{11..15}$];
	ID/EX.IR ← IF/ID.IR;
	ID/EX.Imm ← (IR$_{16}$)^{16} # # IR $_{16..31}$;
EX	
MEM	
WR	

Nota:Transferurile reprezentate in rosu sunt noi sau modificate

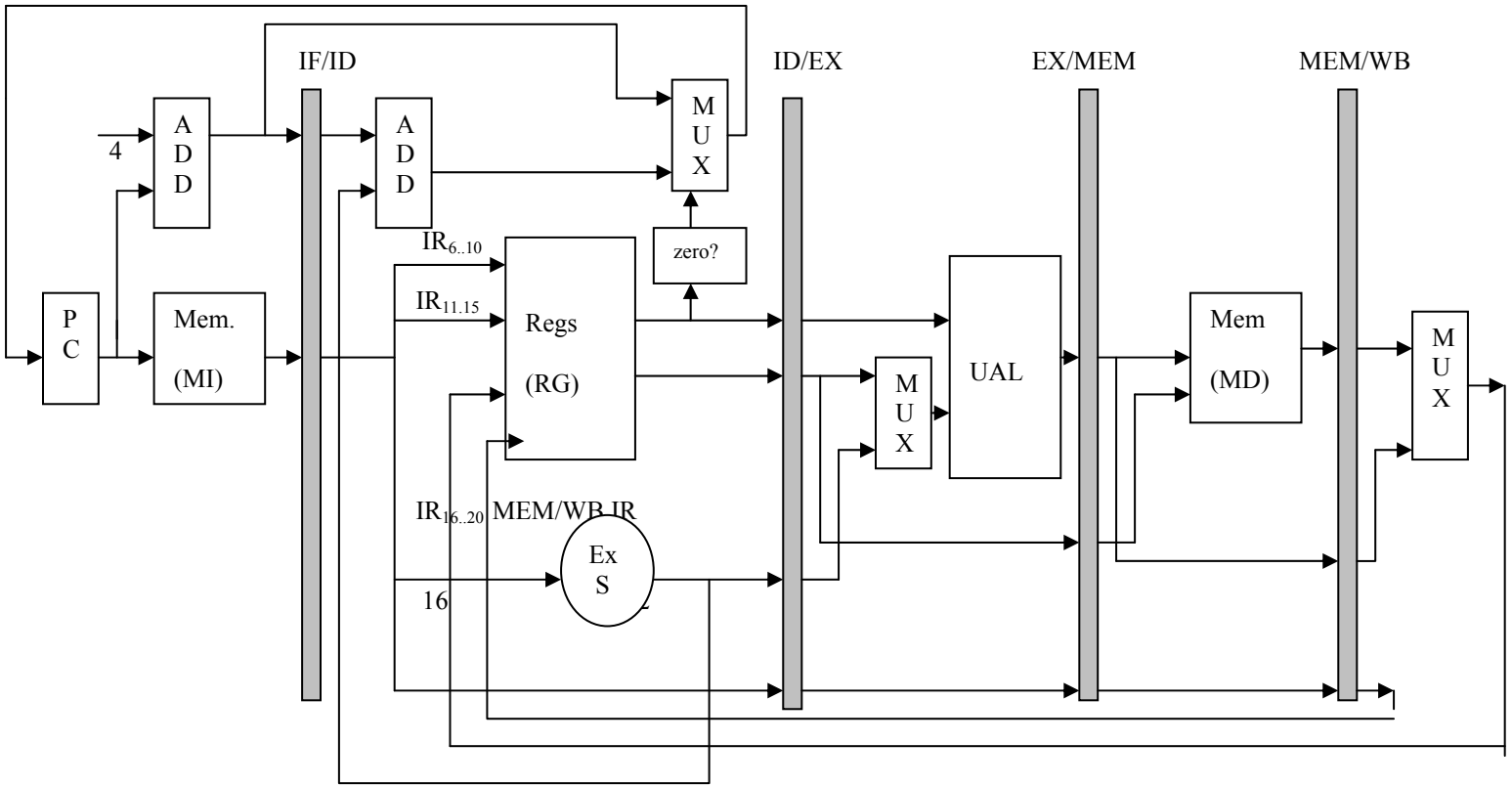

Fig. 7.16. Unitatea de Executie DLX in varianta Banda de Asamblare, cu modificarile efectuate pentru instructiunile Br.

Problema. Sa se compare, din punctul de vedere al performantelor, implementarile bazate pe memorii biport si monoport.

Structura A este prevazuta cu memorie biport.

Structura B are o memorie monoport, dar BA de care dispune are o frecventa de ceas mai mare cu 5%.

Ambele structuri au CPI = CPII = 1.

Instructiunile de incarcare au o pondere de 40%.

Se reaminteste formula pentru cresterea de viteza:

Cresterea de viteza = [adancimea BA/(1 + CIPI)] × (T_{FBA}/T_{CBA}) /* Dar, Cicluri de Intarziere pe Instructiune (CIPI) =0

Cresterea de viteza pentru A = adancimea BA/(1 + 0) × (T_{FBA}/T_{CBA})

= adancimea BA

Cresterea de viteza pentru B = adancimea BA/(1 + 0,4 × 1) × [$T_{FBA}/(T_{CBA}/1,05)$]

= (adancimea BA/1,4) × 1,05

= 0,75 × adancimea BA

Cresterea de viteza pentru A/ Cresterea de viteza pentru B = adancimea BA/(0,75 × adancimea BA) = 1,33

Structura A este de 1,33 ori mai rapida.

Concluzii:

- Hazardele limiteaza performanta.
  - Hazardele structurale necesita pentru rezolvare mai multe resurse hardware;
  - Hazardele de date se rezolva prin tehnica avansarii, prin planificarea de catre compilator.
  - Hazardele de control se solutioneaza prin evaluarea cat mai timpurie a conditiei, prin intarzierea ramificarii, prin predictie
- Cresterea lungimii BA mareste impactul hazardurilor.
- Intreruperile, Setul de instructiuni si Instructiunile in VM complica BA;
- Compilatoarele reduc costul hazardelor de date si control:
  - Introduc cicluri de intarziere pentru instructiunile de incarcare;
  - Introduc cicluri de intarziere pentru instructiunile de ramificare;
  - Predictia ramificarii.