ARHITECTURA CALCULATOARELOR 2003/2004
CURSUL 12

4.2.6 Operațiuni pe magistrală

Până acum, am discutat doar cicluri de magistrală normale, cu un master (de obicei CPU-ul) citind de la un slave (de obicei memoria) sau scriind unuia. De fapt, mai există câteva tipuri de cicluri de magistrală. Acum le vom analiza.

În mod normal, transferul se face cuvânt cu cuvânt, câte unul pe rând. Totuşi, când cache-ul este folosit, este de dorit să aducem întreaga linie cache odata. De multe ori **blocurile de transfer** pot fi mai eficiente decât transferul succesiv individual. Când o citire de bloc este

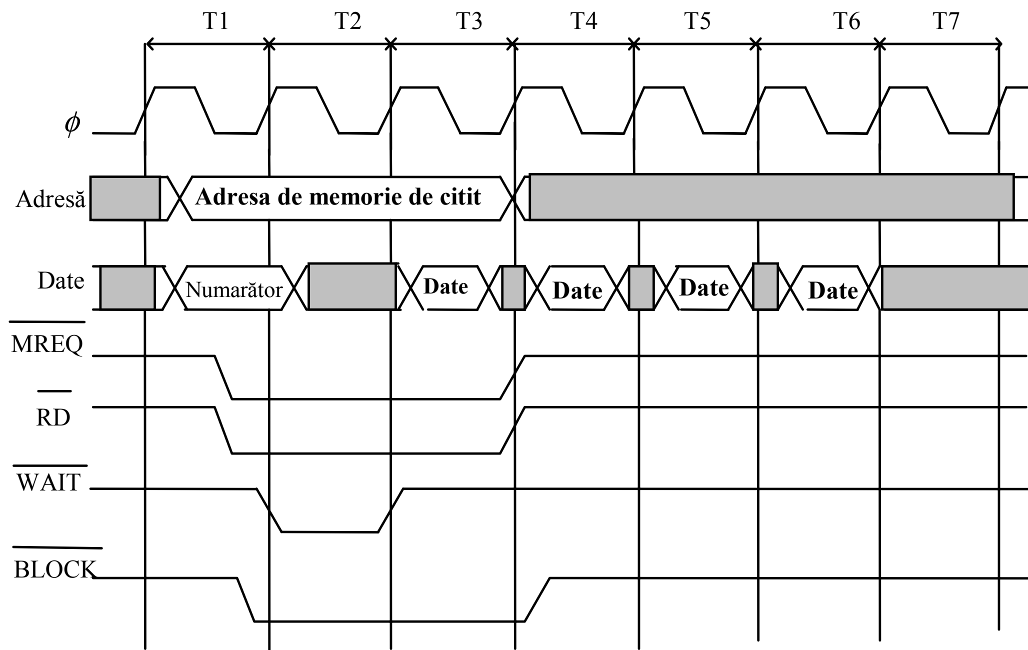

Figura 4.16 Transferul datelor pe blocuri

startată, masterul de magistrală transmite slaveului numărul de cuvinte care vor fi transferate, de exemplu, punând numărul de cuvinte pe liniile de date în timpul T_1. Înloc să returneze un singur cuvânt, slaveul oferă câte un cuvânt în timpul fiecărui ciclu, până când numărătoarea s-a terminat. Figura 4.16 este o versiune modificată a unei figuri anterioare, cu un semnal $\overline{BLOCK}$ care este folosit să indice atunci când un transfer de bloc este solicitat. În acest exemplu, citirea unui bloc de 4 cuvinte ia 6 cicluri în loc de 12.

Există de asemenea şi alte tipuri de cicluri de magistrală. De exemplu, pe un sistem multiprocesor cu **două sau mai multe procesoare pe aceeaşi magistrală**, este adesea necesar să se asigure că doar un procesor foloseşte structura de date din memorie. Un mod tipic de a

asigura aceasta, este de a avea o variabilă în memorie, care este 0 atunci când nici un procesor nu foloseşte structura de date, şi 1 când aceasta este folosită. Dacă un procesor vrea să obţină ccesul la structura de date, el trebuie să citească variabila, şi dacă aceasta este 0 atunci să o seteze la 1. Necazul este, cu puţin ghinion, ca două procesoare s-ar putea să citească variabila pe două cicluri de bus consecutive. Dacă fiecare vede că variabila este 0, atunci fecare o seteaza la 1 şi crede că este singurul procesor care foloseşte structura de date. Această secvenţă de evenimente conduce la haos.

Pentru a preveni această situaţie, sistemele multiprocesor au adesea un ciclu de bus special citeşte – modifică - scrie, care permite oricărui procesor să citească un cuvânt din memorie, să-l examineze, să-l modifice şi să-l scrie înapoi în memorie, totul fără să elibereze magistrala. Acest tip de ciclu, controleză capacitatea procesoarelor de a folosi magistrala.

Un alt tip important ciclu de magistrală este acela pentru **manipularea întreruperilor**. Când procesorul comandă unui dispozitiv I/O să facă o operaţie, el aşteaptă deobicei o întrerupere când lucrul este gata. Semnalarea întreruperii solicită magistrala.

Din moment ce mai multe dispozitive trebuie să cauzeze o întrerupere simultan, acelaşi fel de probleme de arbitrare sunt prezente şi aici, aşa cum am avut şi cu ciclurile de magistrală normale. Soluţia uzuală este aceea de a atribui priorităţi dispozitivelor, şi de a folosi un arbitru centralizat, care să dea prioritate celui mai critic dispoziv. Controlerele de întreruperi standard, într-un cip, există şi sunt considerabil folosite. IBM PC-ul şi succesorii săi au folosit cipul Intel 8259A, ilustrat în figura 4.17.

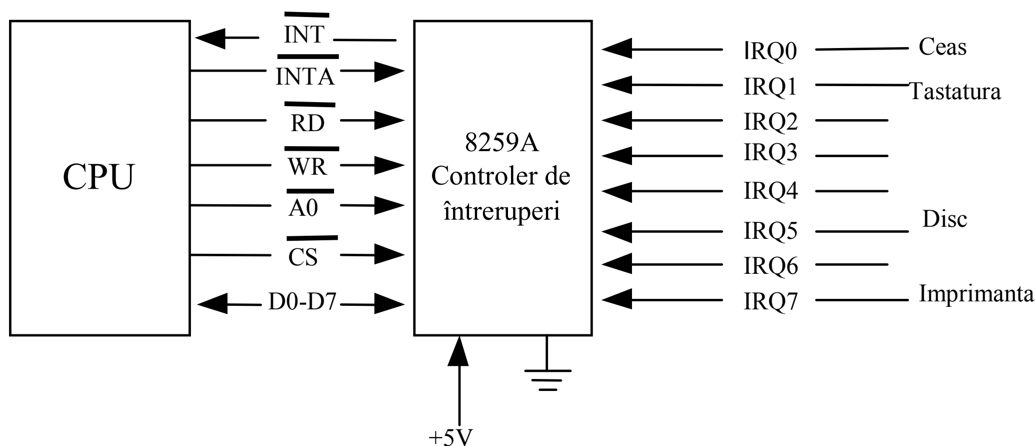

Figura 4.17 Gestionarea întreruperilor cu 8259A

Până la 8 controlere pot fi conectate direct la 8 intrări IRx la 8259A. Când oricare dintre aceste dispozitive cauzează o întrerupere, acestea activează linia de intrare. Când una sau mai multe intrari sunt activate, 8259A-ul activează INT (INTerrupt), care conduce direct la pinul de întrerupere al procesorului. Când procesorul este capabil să suporte întreruperea, el trimite un semnal înapoi procesorului 8259A pe INTA (Interrupt Acknowledge). În acel moment, 8259A trebuie să specifice care intrare cauzează întreruperea, trimiţând acel număr de ordine al intrării, pe magistrala de date. Această operaţie cere un ciclu de magistrală special. Procesorul foloseşte atunci acel număr ca index, într-o tabelă de pointeri, numiţi vectori de întrerupere, pentru a găsi adresa procedurii care tratează întreruperea.

Cipul 829A are registri în interiorul său, pe care procesorul îi poate citi şi scrie folosind ciclul de magistrală normal şi pinii $\overline{RD}$ (read - citire), $\overline{WR}$ (write - scriere), $\overline{CS}$ (chip select - selectare cip) şi $\overline{A0}$. Când programul care tratează întreruperea e gata şi poate să o ia pe următoarea, el scrie un cod special într-unul din regiştrii, care forţează cipul 8259A să inactiveze INT, dacă între timp nu se tratează o alta întrerupere. Aceşti registri pot fi scrişi astfel încât să pună cipul 8259A într-unul din mai multe moduri, să mascheze un set de întreruperi, şi să permită alte caracteristici.

Când mai mult de 8 dispzitive I/O sunt prezente, poate fi folosit încă un cip 8259A cascadat. În cazul extrem, toate cele opt intreruperi pot fi conectate la ieşirile altor 8 cipuri 8259A permiţând până la 64 componente I/O să lucreze într-o reţea de întreruperii în două faze. Cipul 8259A are câţiva pini dedicaţi să suporte aceste cascadări, care au fost omişi pentru simplificare.

Capitolul 5: Nivelul microarhitecturii

Nivelul microarhitecturii este nivelul imediat superior nivelului logic numeric. Sarcina acestui nivel este implementarea ISA (Instruction Set Architecture), nivelul superior lui (vezi figura de la inceputul capitolului 4). Proiectarea nivelului microarhitecturii depinde de pe de o parte de tipul nivelului ISA implementat, iar pe de altă parte de costul şi performanţele avute în vedere pentru respectivul sistem de calcul. Multe dintre ISA moderne, în special cele de tip RISC, au instrucţiuni simple care de cele mai multe ori se pot executa într-un ciclu de ceas. ISA mai complexe, ca în exemplul Pentium II, au instrucţiuni care au nevoie de mai multe cicluri de ceas pentru a fi executate. Executarea unei instrucţiuni poate cere localizarea operanzilor în memorie, citirea lor şi stocarea rezultatelor la loc în memorie. Secvenţierea operaţiilor corespunzătoare unei singure instrucţiuni conduce adeseori la abordări diferite pentru controlul lor decât acelea pentru ISA simple.

5.1 Un exemplu de microarhitectură

În mod ideal acest subiect ar trebui introdus prin explicarea principiilor generale ale proiectării microarhitecturii. Din pacate nu există astfel de principii generale: fiecare caz este unul special. În cosecinţă vom prezenta un exemplu. Pentru a exemplifica ISA s-a ales o submulţime a unei maşini Java virtuale. Această submulţime conţine numai instrucţiunile care lucrează cu întregi şi ca urmare o vom identifica cu sigla IJVM (Integers Java Virtual Machine). Se va prezenta microarhitectura pe care se va implementa IJVM.

Multe dintre aceste arhitecturi au fost adeseori implementate prin microprogramare cum am discutat în capitolul 3. Deşi IJVM este mic, este un punct de început bun pentru a descrie instrucţiunile şi secvenţele de control.

Microarhitectura va conţine un microprogram (în ROM) a cărui treabă este să extragă, decodifice şi să execute instrucţiunile IJVM. Nu s-a putut folosi interpretorul Sun JVM pentru microprogram pentru că este nevoie de un microprogram simplu care să conducă eficient porţile individuale din hardware. În contrast interpretorul Sun JVM a fost scris în C pentru portabilitate şi nu poate controla hardwareul la nivelul de detaliu de care este nevoie. Din

moment ce hardwareul folosit constă decât din componentele primare descrise anterior, în teorie, după ce a înțeles complet acest capitol, cititorul ar trebui să fie capabil să se duca să cumpere o geantă mare plină de tranzistori şi să construiască acest ansamblu al maşinii JVM. Studenții care au realizat această aplicație cu succes li se va acorda un extracredit (şi un examen psihiatric complet)!!!

Un model convenabil pentru proiectarea microarhitecturii este să gândim proiectul ca o problema de programare, unde fiecare instrucțiune la nivel ISA este o funcție care este apelată de programul principal. În acest model, programul principal este simplu, o buclă infinită care determină funcția care va fi apelată, apelează funcția, după care incepe iar, ca în figura 3.3, "Un interpretor pentru un calculator simplu (scris in Java)" din cursul 8.

Microprogramul are un set de variabile, care alcătuiesc starea calculatorului şi care pot fi accesate de toate funcțiile. Fiecare funcție schimbă cel puțin câteva dintre variabile. Spre exemplu, Program Counter-ul (PC) este o parte a stării. Aceasta indică locația de memorie conținând următoarea funcție (ex: instrucțiune ISA) care va fi executată. În timpul execuției fiecărei instrucțiuni PC este avansat la punctul unde va fi executată următoarea instrucțiune. Instruicțiunile IJVM sun scurte. Fiecare instrucțiune are câteva câmpuri, de obicei 1 sau 2, fiecare dintre ele are semnificatii specifice. Primul câmp al fiecărei instrucțiuni este opcode (codul de operație), care identifică instrucțiunea spunând dacă este o ADD sau o BRANCH, sau altceva. Multe instrucțiuni au un camp adițional care specifică operandul. De exemplu instructiunile care accesează o variabilă locală au nevoie de un câmp pentru a spune numele variabilelor. Acest model de execuție, uneori numit "fetch - execute cycle", este folositor unei abordări abstracte şi poate fi de asemenea baza pentru o implementare ISA ca IJVM care are instrucțiuni complexe. Mai jos vom descrie cum funcționează, cum arată microarhitectura, şi cum este controlată de microinstrucțiuni, fiecare dintre ele controlând calea datelor (data path) pentru un ciclu. Împreună, lista de microinstrucțiuni formează microprogramul.

5.1.1 Calea de date

Calea de date este parte a CPU-ului care conține ALU, intrările şi ieşirile ei. Calea de date, a exemplului ales, de microarhitectură este arătată în figura 5.1. Pentru că a fost atent optimizată pentru interpretarea programelor IJVM este aproape similară cu calea de date folosită în majoritatea maşinilor. Contine un număr de regiştri de 32 biti, cărora li s-au dat simbolic nume ca PC, SP şi MDR. Deşi unele dintre aceste nume sunt familiare este important să ințelegem că aceşti regiştri nu sunt accesibili decât la nivelul microarhitecturii (de către microprogram). Li s-au dat aceste nume pentru că de obicei conțin o valoare a variabilei corespunzătoare de acelaşi nume în nivelul de arhitectură ISA. Mulți regiştri pot conduce conținutul lor în magistrala B. Ieşirile ALU conduc registrul de deplasare şi apoi magistrala C, a cărei valoare poate fi scrisă în unul sau mai mulți regiştri în acelaşi timp. Deocamdată nu există magistrala A.

ALU este identic cu cel arătat în capitolul anterior corespunzător subiectului. Funcțiile lui sunt determinate de 6 linii de control. Diagonala scurtă etichetată "6" în figura indică existența a 6 linii de control ALU. Liniile F_0 şi F_1 sunt folosite pentru determinarea operației ALU, ENA şi ENB sunt folosite pentru a activa individual intrările, INVA este folosit pentru a inversa intrarea din stânga şi INC pentru a forța un transport (carry) în bitul cel mai puțin semnificativ, "low-order", efectiv adăugând 1 la rezultat. Oricum, nu toate cele 64 combinații de linii de control ALU fac ceva folositor.

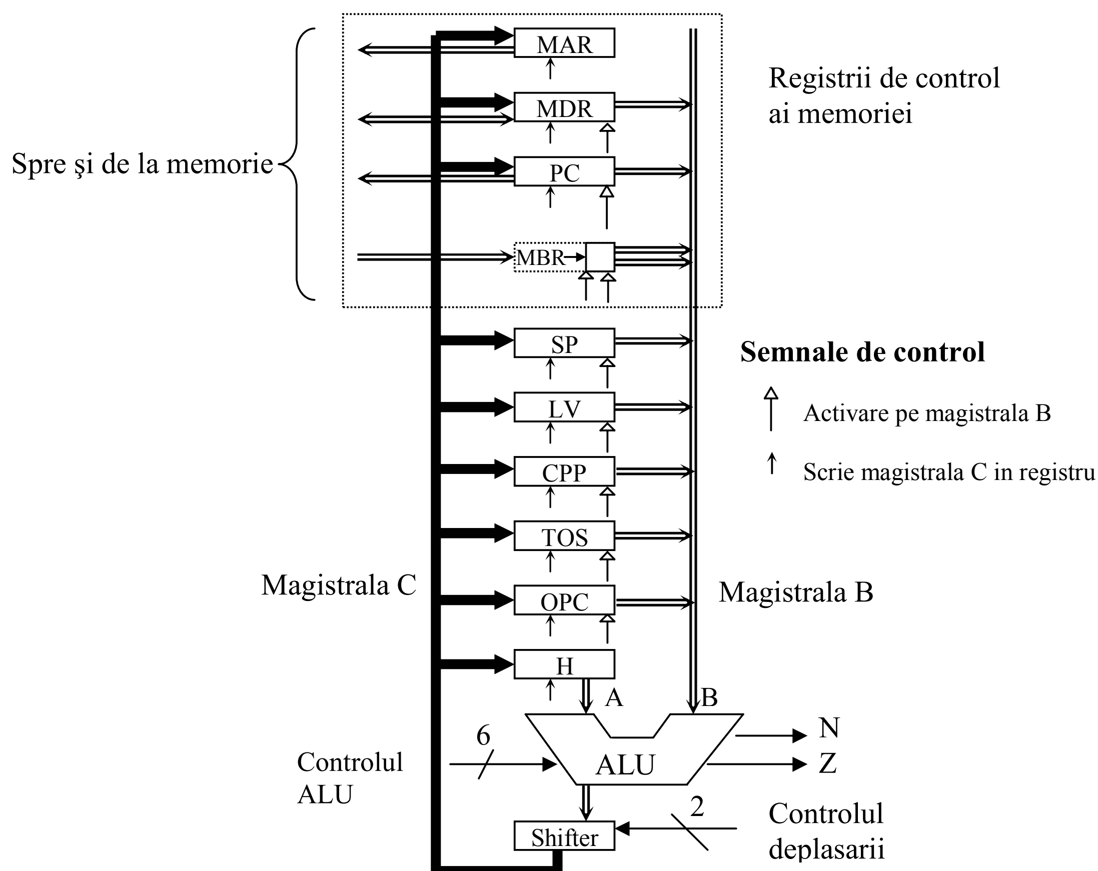

Figura 5.1 Cale de date pentru microarhitectura exemplului ISA IJM

F_0	F_1	ENA	ENB	INVA	INC	Funcție
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A+B
1	1	1	1	0	1	A+B+1
1	1	1	0	0	1	A+1
1	1	0	1	0	1	B+1
1	1	1	1	1	1	B-A
1	1	0	1	1	1	B-1
1	1	1	0	1	1	-A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
0	1	0	0	0	1	1
0	1	0	0	1	0	-1

Figura 5.2 Functiile ALU pentru microarhitectura exemplului ISA IJM

Unele dintre cele mai interesante combinații sunt arătate în figura 5.2. Nu toate din aceste funcții sunt necesare pentru IJVM, dar pentru varianta completă JVM multe dintre ele ar fi

folositoare. În multe cazuri există multiple posibilități pentru a obține acelaşi rezultat. În acest tabel, + înseamnă plus aritmetic, iar – înseamnă minus aritmetic, deci de exemplu, -A înseamnă complementul față de 2 al lui A.

ALU din figura 5.1 are nevoie de două intrări de date: intrarea stânga (A) şi intrarea dreapta (B). Ataşat de intrarea din stânga este un registru H (H de la holding). Ataşată de intrarea din dreapta este magistrala B, care poate fi încărcată din oricare din cele 9 surse, indicate de cele 9 săgeți care-o ating. Un proiect alternativ, cu două magistrale complete are o altă abordare.

H poate fi încărcat inițial alegând o funcție ALU care transferă numai intrarea din dreapta (de la magistrala B) la ieşirea ALU. O asemenea funcție este adunarea intrărilor ALU, doar cu ENA negat astfel încât intrarea stângă este forțată la 0. Adunând 0 la valoarea de pe magistrala B se păstrează valoarea de pe magistrala B. Acest rezultat poate trece prin registrul de deplasare (shifter) nemodificat şi înregistrat în H.

Suplimentar la funcțiile de mai sus, încă două linii de control pot fi folosite independent pentru a controla ieşirea din ALU. SLL 8 (Shift Left Logical) deplasează conținutul spre stângă cu 1 byte, forțând cei mai puțini semnificativi 8 biți, la 0. SRA 1 (Shift Right Arithmetic) deplasează conținutul spre dreapta cu un bit, lăsând cei mai semnificativi biți neschimbați.

Este explicit posibil să se citească şi să se scrie acelaşi registru într-un singur ciclu. De exemplu, este permis sa se pună SP pe magistrala B, să se dezactiveze intrarea din stânga a ALU, să se activeze semnalul INC, şi să se scrie rezultatul în SP, deci incrementând SP cu 1 (vezi a 8-a linie din tabel). Cum poate fi un registru citit şi scris în acelaşi ciclu fără a face probleme? Soluția este că citirea şi scrierea sunt de fapt executate la timpi diferiți în acelaşi ciclu. Când un registru este selectat ca fiind intrarea dreapta ALU, valoarea sa este pusă mai devreme în ciclu pe magistrala B şi este ținută acolo în continuu până la sfârşitul ciclului. Atunci ALU îşi face treaba, producând un rezultat care trece prin shifter în magistrala C. Aproape de sfârşitul ciclului, când se ştie că iesirile ALU şi shifterului sunt stabile, un semnal de ceas declanşează stocarea conținutului magistralei C într-unul sau mai mulți regiştri. Unul dintre registrii poate fi foarte bine unul pe care l-a furnizat magistrala B prin intrările sale. Secvențierea precisă a căii de date face posibilă scrierea şi citirea aceluiaşi registru într-un ciclu, cum va fi descris mai târziu.

5.1.2 Secvențierea în timp (timing) a căii de date

Secvențierea în timp a acestor evenimente este prezentată în figura 5.3. Aici este produs un scurt puls la începutul fiecărui ciclu de ceas. Acesta poate fi derivat din ceasul principal, cum este arătat într-un paragraf anterior. La frontul căzător al pulsului, sunt setați biții care vor conduce porțile. Aceasta necesită un timp finit şi cunoscut Δw. Apoi este selectat registrul de care este nevoie pentru magistrala B şi condus pe magistrala B. Este nevoie de Δx înainte ca valoarea să se stabilizeze. Atunci ALU şi shifterul încep să opereze pe date valide. După încă Δy, ALU şi shifterul au ieşiri stabile. După un adițional Δz rezultatele sunt propagate de-a lungul magistralei C către regiştrii unde pot fi încărcate pe frontul crescător al următorului puls. Încărcarea trebuie să fie declanşată rapid pe front astfel încât chiar şi dacă unii dintre regiştrii de intrare sunt schimbați, efectele nu vor fi simtițe pe magistrala C decât mult după ce

regiştri au fost încărcaţi. De asemenea, pe frontul crescător al pulsului registrul care conduce magistrala B se opreşte din această acţiune, pregătindu-se pentru următorul ciclu. MPC, MIR, şi memoria sunt menţionate în figura 5.1.

Este important să realizăm că, chiar dacă nu există elemente de stocare în calea de date, timpul de propagare prin aceasta este finit. Schimbarea valorilor de pe magistrala B nu cauzează schimbarea magistralei C decât mai târziu după un timp finit (din cauza întârzierilor finite care apar la fiecare pas). Ca urmare, chiar dacă o stocare schimbă unul dintre regiştrii de intrare, valoarea va fi preluată în registru mult înainte ca valoarea (în acest moment incorectă) pusă pe magistrala B (sau H) să ajungă la ALU. Pentru a face acest proiect să meargă e nevoie de o secvenţiere rigidă, un ciclu de ceas lung, un timp de propagare minim prin ALU cunoscut, şi o încărcare rapidă a regiştrilor din magistrala C. Oricum cu o proiectare atentă calea de date poate fi proiectată să funcţioneze corect. Un mod oarecum diferit de a privi ciclul căii datei este să o gândim descompusă în subciclurile implicite. Începutul subciclului 1 este declanşat de frontul căzător al ceasului.

Activităţile care se petrec în timpul subciclurilor sunt arătate mai jos împreună cu lungimea subciclurilor.

1. Semnalul de control este setat (Δw).
2. Regiştri sunt încărcaţi pe magistrala B (Δx).
3. ALU şi shifterul operază (Δy).
4. Rezultatele sunt propagate de-a lungul magistralei C înapoi în regiştri (Δz).
5. Pe frontul crescător al urmatorului ciclu de ceas, rezultatele sunt stocate in registrii.

Subciclurile pot fi gandite mai bine ca fiind implicite. Prin aceasta înţelegem că nu există impulsuri de ceas sau alte semnale explicite, care să anunţe ALU când să opereze sau să determine intrarea rezultatelor pe magistrala C. În realitate ALU şi shifterul merg tot timpul. intrările lor sunt eronate până la un moment Dw + Dx după frontul descrescător al ceasului.

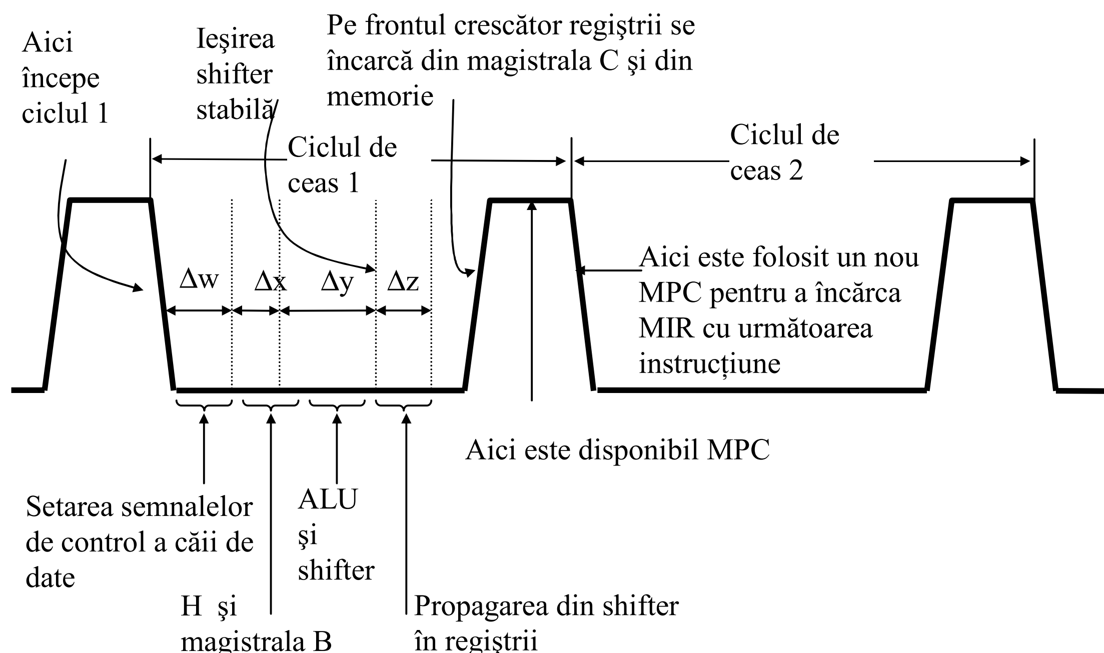

Figura 5.3 Secvenţierea căii de date

De asemenea iesirile lor sunt eronate până la un moment Dw + Dx + Dy după frontul descrescător al ceasului. Singurele semnale explicite care conduc calea de date sunt frontul descrescător al ceasului, care porneşte ciclul căii de date şi frontul crescător al ceasului, care încarcă regiştrii din magistrala C. Celelalte legături subciclice sunt implicit determinate de timpul de propagare moştenit de la circuitele implicate.

Este responsabilitatea proiectanților să se asigure ca momentul Dw + Dx + Dy + Dz să apară suficient de devreme înaintea frontului crescător al ceasului pentru a avea tot timpul funcționale valorile de încărcare ale regiştrilor.

5.1.3 Operații cu memoria

Maşina prezentată are 2 căi diferite de comunicare cu memoria: un port de memorie adresabil pe cuvânt pe 32 de biți şi un port de memorie adresabil pe octet, pe 8 biți. Portul de 32 de biți este controlat de 2 regiştrii, MAR (Memory Address Register), şi MDR (Memory Data Register), ca în figura 5.1. Portul de 8 biți este controlat de un registru, PC, care citeşte un octet în cei mai puțin semnificativi 8 biți ai MBR. Acest port nu poate decât să citească date din memorie; el nu poate scrie date în memorie. Fiecare dintre aceşti regiştri (şi ceilalți regiştrii din figura 5.1) sunt conduşi de 1 sau 2 semnale de control. O săgeată deschisă sub un registru indică un semnal de control care activează ieşirea registrului pe magistrala B. Din moment ce MAR nu are o conexiune la magistrala B, el nu are un semnal de activare. Nici H nu are unul pentru că este întotdeauna activat, fiind singura intrare din stânga posibilă pentru ALU.

O săgeată neagră sub registru indică un semanl de control care scrie (încarcă) registrul de pe magistrala C. Din moment ce MBR nu poate fi încărcat de pe magistrala C, el nu poate avea un semnal de scriere (deşi are alte 2 semnale de activare descrise în continuare). Pentru a iniția scrierea sau citirea memoriei, regiştrii corespunzători de memorie trebuie încărcați, apoi un semnal de scriere sau citire e trimis la memorie (nu apare pe figură).

MAR conține adrese de cuvânt, pentru ca valorile 0, 1, 2 etc. să se refere la cuvinte consecutive. PC conține adrese de octet pentru ca valorile 0, 1, 2 etc să se refere la octeți consecutivi. Deci punând un 2 în PC şi inițiind o citire a memoriei se va citi octetul 2 din memorie şi se va pune în cei mai puțin semnificativi 8 biți ai MBR. Punând un 2 în MAR şi inițiind o citire a memoriei se vor citi octeții 8-11 (adică, cuvântul 2) din memorie şi vor fi puşi în MDR. Această diferență în funcționare este necesară pentru că MAR şi PC vor fi folosite ca referințe pentru 2 părți diferite din memorie. Nevoia pentru această distincție va deveni clară mai târziu. Pentru moment este suficient să spunem că, combinația MAR/MDR este folosită pentru a citi şi a scrie cuvinte de date la nivelul ISA şi combinația PC/MBR este folosită la citirea programelor executabile la nivel ISA, care constă într-un şir de octeți. Toti ceilalți regiştri care conțin adrese folosesc adrese de cuvânt, precum MAR.

În implementarea reală fizică există decât o singură memorie reală şi este orientată pe octeți. Situația apărută ca urmare a faptului că MAR este lăsat să numere în cuvinte (necesar pentru felul în care este definit JVM) cât timp memoria fizică numără în octeți este gestionată cu un simplu truc. Când MAR este pus pe magistrală de adrese, cei 32 de biți nu apar pe cele 32 de linii de adrese, 0-31, direct. Înloc de aceasta, bitul 0 al MAR este legat de linia de adresă 2 a magistralei, bitul 1 al MAR este legat de linia de adresă 3 a magistralei şi tot aşa mai departe.

Cei 2 biți superiori ai MAR nu sunt luați în seamă din moment ce nu este nevoie de ei decât pentru adrese cuvânt peste 2^{32}, ceea ce nu este folosit în maşina noastră de 4 GB. Folosind această corespondență, când MAR este 1, adresa 4 este pusă pe magistrală; când MAR este 2, adresa 8 este pusă pe magistrală şi aşa mai departe. Acest truc este ilustrat în figura 5.4.
Aşa cum a fost menționat mai sus, citirea datelor din memorie prin portul de memorie de 8 biți este returnată în MBR, un registru de 8 biți. MBR poate fi copiat pe magistrala B în una dintre cele 2 căi: cu semn sau fără semn. Când valoarea fără semn este necesară, cuvântul de 32 de biți pus pe magistrala B conține valoarea MBR în cei 8 biți mai puțin semnificativi şi 0 în cei 24 de biți superiori. Valoarea fără semn este folositoare la indexarea într-o tabelă sau când un întreg pe 16 biți trebuie să fie asamblat din 2 octeți consecutivi (fără semn) în fluxul de instrucțiuni.

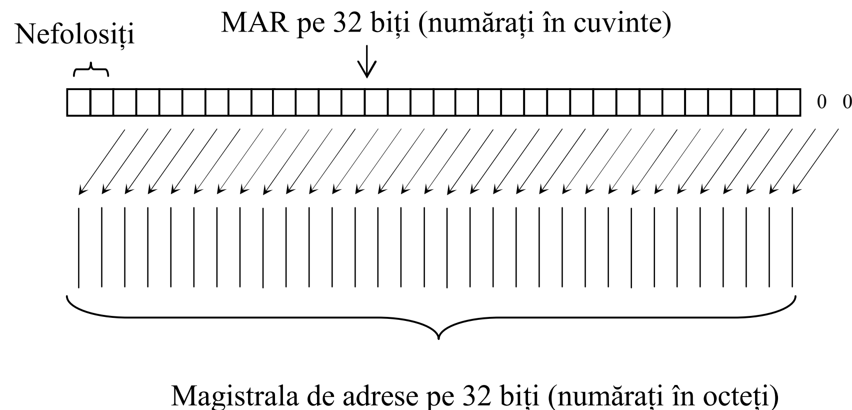

Figura 5.4 Utilizarea registrului MAR

O altă opțiune pentru a converti MBR-ul de 8 biți într-un cuvânt de 32 de biți presupune să îl tratăm ca pe o valoare cu semn între –128 şi 127 şi să folosim această valoare pentru a genera un cuvânt pe 32 de biți cu aceeaşi valoare numerică. Această conversie este făcută prin duplicarea bitului de semn MBR (cel mai din stânga bit) în cei 24 de biți superiori ai magistralei B, proces cunoscut sub denumirea de extensie de semn. Când se alege această opțiune, cei 24 de biți superiori vor fi ori toți 0 ori 1, depinzând de cel mai din stânga bit din MBR-ul de 8 biți, dacă e 0 sau 1.

Alegerea dacă MBR-ul de 8 biți este convertit într-o valoare de 32 de biți cu sau, respectiv, fără semn pe magistrala B este determinată de unul dintre cele 2 semnale de control (săgețile deschise de sub MBR). Nevoia pentru aceste două opțiuni este motivul prezenței celor două săgeți. Abilitatea de a face MBR-ul de 8 biți să funcționeze ca o sursă de 32 de biți pentru magistrala B este indicată de pătratul punctat din stânga MBR-ului în figura 5.1.