UTILIZAREA SIMULATORULUI DLX

1. Scopul lucrarii

Scopul lucrarii este de a ajuta la întelegerea conceptelor legate de procesarea pipeline a instructiunilor precum si a aspectelor arhitecturale specifice procesoarelor RISC. Lucrarea de fata prezinta o descriere, nu foarte detaliata, a simulatorului DLX, un simulator scris pentru sistemul Windows. Pentru a utiliza simulatorul sunt necesare cunostinte minime de lucru în Windows (minim Windows 3.0), cum ar fi: startarea aplicatiei, lucrul cu ferestre (minimizare, maximizare), defilarea într-o fereastra, activarea sau aducerea unei ferestre în "*foreground*", etc.

2. Memento teoretic

Simulatorul DLX descrie modul de functionare al procesorului DLX (DeLuXe), un procesor pipeline, folosit drept exemplu în lucrarea "*Computer Architecture - A quantitative approach*", scrisa de J. Hennessy si D. Patterson. Procesorul DLX are cinci nivele distincte în procesarea instructiunilor. În nivelul **IF** (fetch instructiune) - se calculeaza adresa grupului de instructiuni ce trebuiesc citite din memoria principala. Al doilea nivel: **ID** (decodificare instructiune) - decodifica instructiunile aduse, se citesc operanzii din setul de registri generali, se calculeaza adresa de salt (pentru instructiunile de ramificatie). În timpul celei de-a treia faze de procesare a pipe-ului: **EX** (executa instrutiune) se executa operatii aritmetico-logice, de deplasare si rotire asupra operanzilor care pot fi numere întregi sau flotante, se calculeaza adresa de acces la memorie (pentru instructiunile LOAD sau STORE).. Accesarea / scrierea datelor din / în memoria principala, prin instructiunile LOAD sau STORE se face în nivelul patru al pipe-ului **MEM**. În final, avem al cincilea nivel **WB** (scriere date) - în care, unitatile functionale preiau rezultatul final al

instructiunilor aritmetico-logice sau data citita din memorie, si o depun pe magistrala rezultat, de unde este copiata în registrul destinatie corespunzator.

Modul de lucru al simulatorului este urmatorul: dupa încarcarea unui program de test scris în asamblare pentru procesorul DLX, majoritatea informatiilor relevante pentru CPU (registrii, memorie, intrari/iesiri) pot fi vizualizate si modificate în timpul rularii pas cu pas sau continuu a benchmark-ului. WinDLX ofera totodata, statistici despre comportamentul pipeline în timp al procesorului.

Cele trei benchmark-uri sunt programe de calcul cu numere întregi, scrise în asamblare. Benchmark-ul PRIM.s genereaza un tabel cu primele *Count* (numar arbitrar) numere prime aflate în memorie începând cu adresa *Table*. Pentru calculul *celui mai mare divizor comun* a doua numere sunt necesare modulele INPUT.s - care citeste de la tastatura doua numere întregi si GCM.s - care realizeaza efectiv calculul divizorului si-l afiseaza pe ecran. Factorialul unui numar, introdus de la tastatura cu ajutorul modulului INPUT.s, memorat în registrul 1 al procesorului DLX, este calculat si afisat pe ecran în benchmark-ul FACT.s.

3. Desfasurarea lucrarii

3.1. Pornirea si configurarea WinDLX

Pornirea WinDLX se face, ca orice aplicatie Windows, printr-un dublu click pe iconita fisierului executabil WinDLX. Pe ecran vor aparea o fereastra principala si sase ferestre copil minimizate. Pentru reinitializarea procesorului se alege optiunea *Reset all* din meniul *File*. Pe ecran va aparea fereastra "ResetDLX" si intentia de resetare a procesorului trebuie cofirmata printr-un click pe butonul OK.

WinDLX este capabil sa lucreze cu câteva configuratii. Salvarea configuratiei poate fi facuta la sfârsitul unei sesiuni sau prin activarea optiunii *Save* în meniul de configurare. Poate fi modificata structura, timpul necesar

nivelelor pipeline, capacitatea memoriei si alti parametri de control ai simularii. Pentru alegerea configuratiei standard avem de parcurs urmatorii pasi: din meniul *Configuration* selectam optiunea *Floating Point Stages* si verificam existenta urmatoarelor setari:

	Numar	Cicli Întârziere
Unitate de adunare	1	2
Unitate de înmultire	1	5
Unitate de împartire	1	19

Daca este necesar, schimbarea setarilor se face prin editarea valorilor corecte în câmpurile respective. Validarea noilor valori se face printr-un click pe butonul *OK*.

Setarea dimensiunii memoriei simulate a procesorului DLX se face printr-un click pe optiunea *Memory* din meniul *Configuration*. Aceasta trebuie sa fie 0x8000. Teoretic, capacitatea memoriei poate fi între 512 octeti (0x200) si 16 Mbytes (0x1000000). În practica, ea este limitata la configuratia si managementul memoriei din sistemul MS-Windows. Cu *OK* se revine în fereastra parinte.

La selectarea optiunii *Symbolic addresses* din meniul de configurare, adresele vor fi afisate ca expresii de tipul: "symbol + offset". Altfel, adresele sunt reprezentate ca valori hexazecimale.

Optiunea *Absolute Cycle Count* permite contorizarea absoluta a ciclilor de ceas, începând cu ciclul 0 (de la restartarea procesorului). Altfel, ciclii de tact sunt numerotati relativ, de exemplu, ciclu curent este 0, iar cei precedenti sunt -1, -2, etc.

Optiunea *Enable Forwarding* valideaza sau invalideaza mecanismul de forwarding. Reamintim ca forwarding-ul (numit de altfel si *bypassing* sau uneori *scurtcircuitare*) este o tehnica hardware simpla de reducere a penalitatilor

cauzate de hazardurile de date, rebuclând la o intrare, iesirea unei unitati functionale sau, la procesoarele superscalare folosind statii de rezervare. [1]

3.2. Încarcarea programelor de test

Pentru a putea starta simularea, cel putin un program trebuie încarcat în memoria principala. Astfel, se selecteaza optiunea *Load Code or Data* din meniul *File*. Va aparea o fereastra de dialog din care se pot selecta un numar arbitrar de module cu extensia "*.s*", reprezentând coduri sursa DLX. Daca este confirmata selectia prin apasarea butonului *Load*, toate modulele selectate vor fi asamblate si codul este scris în memoria DLX. Erorile raportate vor apare într-o fereastra de dialog separata iar datele scrise în memorie vor fi invalide. Dupa o încarcare cu succes putem reseta procesorul DLX prin intermediul unei ferestre de dialog.

Daca s-a selectat un fisier nedorit, se insereaza respectivul fisier si se apasa butonul *Delete*. Daca nu se selecteaza nimic, atunci nici un fisier nu va fi încarcat în memorie.

Posibilitatea de încarcare multipla a modulelor în acelasi timp permite definirea simbolurilor globale care pot fi folosite în mai multe module. Ulterior, va fi posibil sa încarcam module fara a tine cont de ordinea de încarcare.

Pentru exemplificare vom considera benchmark-ul *fact.s* si rutina necesara *input.s*.

- se executa click pe **fact.s**.
- se apasa butonul *Select*.
- se executa click pe **input.s**.
- se apasa butonul *Select*.
- se apasa butonul *Load*.

Dupa executia acestor pasi simularea poate începe.

3.3. Simularea propriu-zisa

3.3.1. Fereastra Pipeline

În aceasta fereastra sunt descrise cele 5 nivele pipeline ale executiei instructiunilor.

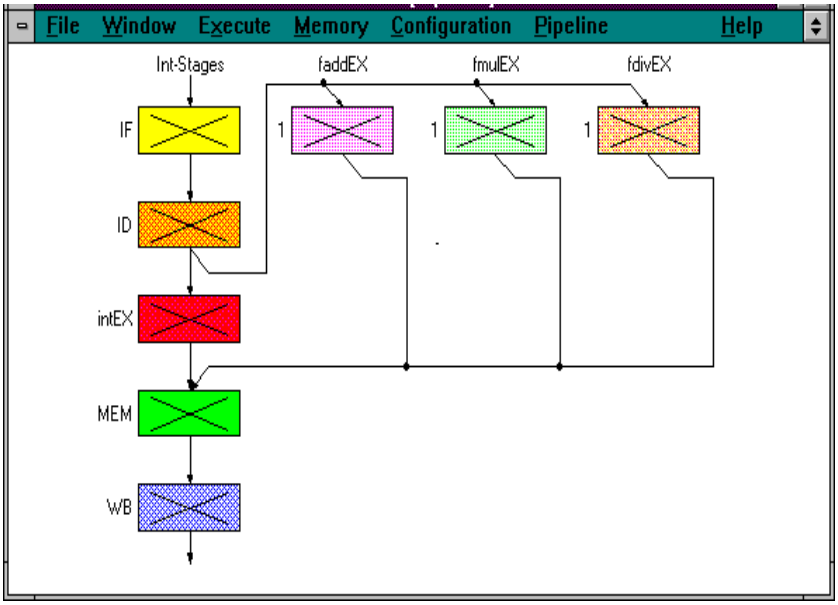

Figura 1. Fereastra pipeline

La nivelul **IF** în fiecare ciclu de tact o noua instructiune este extrasa din memorie si depusa în registrul instructiunii. PC este incrementat pentru a pointa la urmatoarea instructiune. La nivelul **ID** are loc executia branch-urilor pentru a reduce stagnarile. La nivelul **IntEX** au loc operatii aritmetice (mai putin înmultire si împartire) si calcul efectiv al adresei pentru instructiunile cu referire la memorie. În **FaddEX** se executa operatii de adunare si scadere în simpla sau dubla precizie. Unitatile **FmulEX** si **FdivEX** executa operatii de înmultire si împartire atât asupra întregilor (cu semn si fara semn) cât si asupra flotantilor în simpla si dubla precizie. Singurele instructiuni ale DLX active la nivelul **MEM** sunt LOAD si STORE. Calculul adresei de citire sau scriere s-a efectuat în nivelul anterior. Rezultatul este scris în registrii în nivelul **WB**. El provine fie

din memorie fie dintr-o operatie ALU. Operatia de scriere se face în prima jumatate a ciclului de ceas, astfel încât o instructiune aflata pe nivelul ID sa poata citi rezultatul în a doua jumatate a ciclului de ceas, eliminând necesitatea de bypass-ing a rezultatului instructiunii.

3.3.2. Fereastra Cod

Instructiunile DLX încarcate în memorie si adresele lor sunt afisate (în hexazecimal) si dezasamblate în fereastra de Cod.

$TEXT	0x20011000	addi r1,r0,0x1000
main+0x4	0x0c00003c	jal InputUnsigned

Stabilirea unui punct de întrerupere pe o instructiune se face marcând cu Bxx respectiva instructiune, unde xx este tipul întreruperii. Daca o instructiune se afla pe un nivel pipeline de executie culoarea caracteristica a nivelului pipeline este folosita ca o culoare de fond pentru instructiune si este etichetata adecvat.

Defilarea sus/jos prin memorie se face folosind tastele cu sageti sau PgUp/PgDown. Dupa executia unuia sau mai multor pasi ai codului DLX, ultimele instructiuni executate sunt afisate.

Meniul Code contine urmatoarele optiuni:

- *From Address* - adresa de început a codului ce va fi afisat în fereastra de cod; poate fi introdusa printr-o cutie de dialog. Specificarea adresei se face printr-o valoare întreaga, operatori sau simboluri.
- *Set Breakpoint* - pe instructiunea selectata din fereastra de Cod se stabileste un punct de întrerupere.
- *Delete Breakpoint* - punctul de întrerupere de pe instructiunea selectata este sters.

Pentru exemplificare, vom începe simularea secventei de instructiuni descrisa anterior.

Din meniul *Execute* se alege optiunea *Single Cycle*. Tasta F7 are acelasi efect. Vom observa ca prima linie din ferestra cu adresa $TEXT este colorata galben. Apasând F7 vom avansa în simulare cu un pas. Aceasta va determina schimbarea culorii primei linii în orange si urmatoarea linie este colorata galben. Aceste culori arata nivelul pipeline în care se afla fiecare instructiune. Astfel, instructiunea **jal InputUnsigned** este în nivelul IF iar precedenta **addi r1, r0, 0x1000** este în nivelul secund ID. Celelalte nivele sunt marcate cu o cruce, aratând ca nu se proceseaza nici o operatie semnificativa în ele. Tastând F7 în continuare, culorile din fereastra de cod vor fi rearanjate, introducând rosu pentru al treilea nivel intEX. Nivelul MEM va fi colorat în verde, iar WB în albastru.

3.3.3. Diagrama ciclurilor procesorului

În aceasta diagrama este afisat tot ce se întâmpla în fiecare ciclu si în fiecare nivel al pipe-ului. Fiecare coloana reprezinta starea pipe-ului în fiecare ciclu de tact. Starea curenta a pipe-ului (ultima coloana) este colorata cu gri. Executând dublu-click pe o instructiune selectata vom obtine mai multe detalii despre instructiunea respectiva, în fiecare ciclu de tact (instructiunea dezasamblata, adresa ei, codificarea în hexazecimal, starea de executie - nivelul pipeline, modul în care s-a încheiat - normal, cu eroare, întrerupt, numarul ciclului când a început executia, numarul de cicli de executie, etc).

Meniul Diagrama ciclului de tact contine urmatoarele optiuni:

- *Display Forwarding*
- *Display Cause of Stalls*
- *Delete History*
- *Set History Length*

În continuare vom prezenta detalii privind stagnari sau forwarding în executie. Stagnarile sunt reprezentate prin chenare colorate în culoarea corespondenta nivelului pipeline în care se stagneaza. Distingem urmatoarele tipuri de stagnari:

- **R-Stall** - stagnare datorata unui hazard RAW. O sageata rosie închis va indica instructiunea care cauzeaza stagnarea (dupa al carei operand se asteapta).
- **T-Stall** - stagnare datorata unei exceptii Trap. O instructiune Trap ramâne în nivelul IF, pâna când nici o alta instructiune nu se mai afla în pipe.
- **W-Stall** - stagnare datorata unui hazard WAW. Dependenta dintre instructiunile implicate este sugerata tot printr-o sageata de culoare rosu închis.
- **S-Stall** - hazard structural. Nu exista suficiente resurse pentru deservirea instructiunilor.
- **Stall** - daca o instructiune în flotant se afla în nivelul MEM, urmatoarea instructiune va fi oprita în nivelul intEX, etichetat cu "Stall", pâna la extragerea datelor necesare din memorie.

Desi este posibila aparitia oricarui tip de hazard mentionat mai sus, pe cele trei benchmark-uri studiate nu vom întâlni hazarduri structurale sau de tipul WAW, în concluzie fiind suficiente doar câte o unitate de executie pentru fiecare nivel al pipe-ului. Totusi, pentru programe de test care utilizeaza succesiv mai multe instructiuni de (adunare/scadere, înmultire sau împartire) în flotant pot apare hazarduri structurale pentru anumite configuratii arhitecturale.

Exemplu:

Consideram urmatoarea configuratie arhitecturala, fara mecanism de forwarding:

	Numar	Cicli Întârziere
Unitate de adunare	1	2
Unitate de înmultire	1	10
Unitate de împartire	1	19

si urmatoarea secventa de instructiuni:

*i*1: *multd f2, f2, f0*
*i*2: *multd f4, f4, f0*

În urma executiei acestei secvente de instructiuni vom observa aparitia unor întârzieri în procesare datorate hazardului structural generat de prezenta celei de a doua instructiuni de înmultire.

Hazardurile de tip WAW pot apare doar în cazul unui scheduling soft aplicat benchmark-urilor sau daca în vreunul din programele de test exista urmatoarea secventa de instructiuni:

*i*1: *ins_float R2, R2, R0 se executa în 30 ciclii*
*i*2: *ins_integer R2, R4, R6 se executa în 5 ciclii*

Instructiunea pe întregi (*i*1) se încheie mai repede decât cea în flotant (*i*2) si modifica continutul registrului *R2* înainte ca operatia în flotant sa se încheie; rezultatul ramas în *R2,* dupa executia celor doua instructiuni este astfel, eronat.

Daca optiunea *Display Forwarding* este validata, forwarding-ul este reprezentat printr-o sageata verde indicând spre sursa si destinatia mecanismului. Istoria contine informatii despre instructiunile deja încheiate. Se poate specifica dimensiunea istoriei (în instructiuni, de la 0 la 100). Daca aceasta este 0, înseamna ca doar instructiunea care se executa în pipe este afisata în diagrama ciclului de tact.

Pentru o întelegere mai buna, vom exemplifica în continuare pe diagrama ciclului de tact de mai jos.

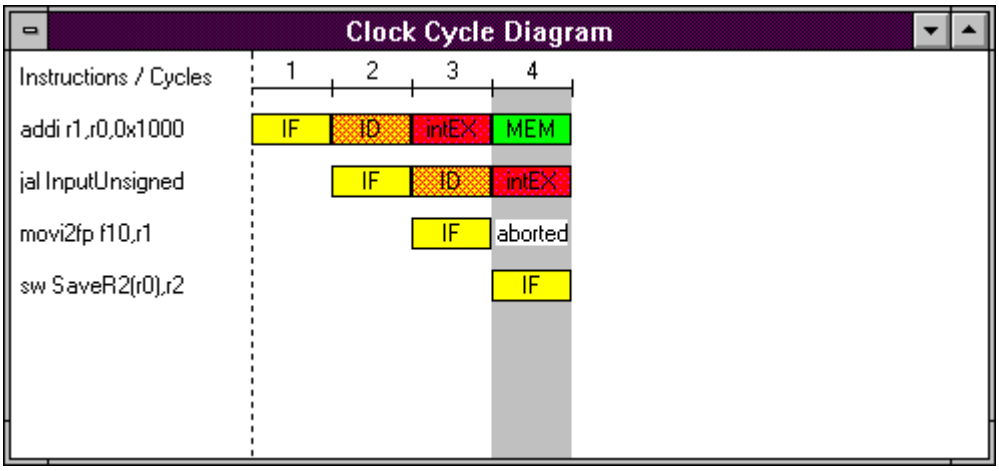

Figura 2. Diagrama ciclului de tact

În diagrama alaturata observam ca simularea este în al patru-lea ciclu, prima instructiune este în nivelul MEM, a doua în intEX si a patra în IF. A treia instructiune este întrerupta. Motivatia consta în faptul ca a doua instructiune **jal,** este un salt neconditionat. Acest lucru e cunoscut abia dupa cel de-al treilea ciclu, dupa ce instructiunea de salt a fost decodificata. În acest timp, instructiunea imediat urmatoare celei de salt (**movi2fp**) a trecut de faza IF, însa instructiunea care se va executa efectiv dupa instructiunea de salt se afla la alta adresa. Astfel, executia instructiunii **movi2f** trebuie întrerupta, lasând loc gol în pipe.

Saltul se va face la eticheta *InputUnsigned*. Pentru a gasi valoarea adresei simbolice se alege optiunea *Symbols* din meniul *Memory*. Fereastra care apare arata corespondenta dintre simbolurile folosite si valorile numerice ale adresei. Este preferabila sortarea dupa *nume* decât dupa *valoare* a adreselor, pentru o regasire mai usoara a acestora. Caracterul ‘G’ scris dupa valoare semnifica un simbol global, iar ‘L’ un simbol local. Astfel, *InputUnsigned* este un simbol global, existent în modulul Input.s, semnificând valoarea adresei 0x144.

3.3.4. Fereastra Breakpoint

În fereastra Breakpoint, putem vizualiza, stabili, schimba si sterge punctele de întrerupere. E posibila stabilirea a maxim 20 de puncte de întrerupere. Pentru manevrarea unui punct de întrerupere, se selecteaza breakpoint-ul respectiv (prin tastele cu sageti sau mouse), iar printr-un dublu-click este permisa modificarea punctului de întrerupere sau a caracteristicilor acestuia. Meniul Breakpoint contine urmatoarele optiuni:

- *Set*
- *Delete*
- *Delete All*
- *Change*

Dupa activarea optiunii Set, pe ecran apare o caseta de dialog prin intermediul careia se stabilesc:

– *adresa* punctului de întrerupere; poate fi o expresie întreaga arbitrara (poate include simboluri sau operatori). Adresa trebuie sa fie multiplu de 4 (aliniere la cuvânt de 4 octeti). În caz contrar, adresa e convertita la multiplu de 4.

– *tipul* punctului de întrerupere; poate fi IF, ID, EX, MEM sau WB, însemnând ca simularea s-a întrerupt daca instructiunea specificata a ajuns în nivelul pipeline specificat de tip. Un breakpoint pe o citire (în cazul instructiunilor Load/Store) se realizeaza în momentul în care în memorie am ajuns pe cuvântul (sau o parte a lui) specificat de adresa punctului de întrerupere. Un breakpoint pe o scriere (în cazul instructiunilor Store sau exceptiilor Trap) se realizeaza când un acces de scriere se face pe cuvântul (sau o parte a lui) specificat de adresa punctului de întrerupere.

Valorile introduse sunt implicite pentru data urmatoare când se va activa optiunea Set. Stergerea unui punct de întrerupere se face selectând optiunea Delete, iar stergerea tuturor punctelor de întrerupere se face cu varianta Delete All. În acest caz e necesara confirmarea actiunii. Prin optiunea Change poate fi

modificata adresa sau tipul punctului de întrerupere selectat prin intermediul casetei de dialog.

Exemplificare:

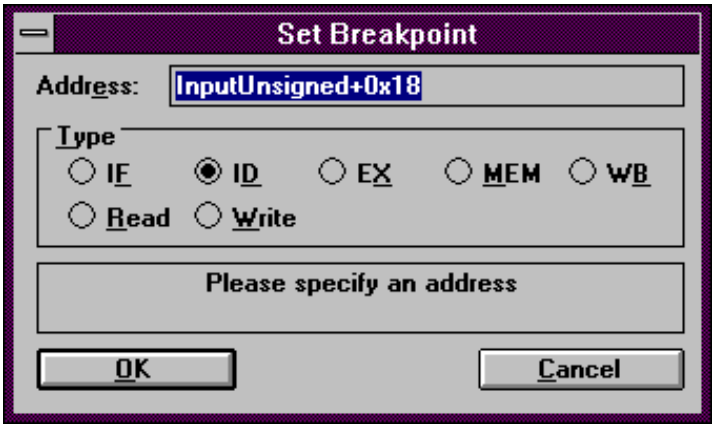

Figura 3. Fereastra Breakpoint

Când examinam fereastra cod observam instructiuni similare care se repeta. Astfel, parcurgerea pas cu pas a codului sursa al benchmark-ului devine plictisitor si ineficient. Pentru accelerarea acestui proces vom introduce puncte de întrerupere.

În fereastra de cod vom selecta linia cu adresa 0x0000015c (instructiunea **trap 0x5 –** instructiune analoaga întreruperilor **int n** de la procesoarele INTEL 80x86). Instructiunea reprezinta un apel sistem în urma caruia pe ecran va fi afisat un mesaj. Se selecteaza optiunea *Set Breakpoint*, iar pe ecran apare o fereastra identica cu cea din figura 3. Tipul implicit al breakpoint-ului este ID. Se confirma cu butonul OK.

În consecinta, pe linia din fereastra de cod pe care se afla instructiunea **trap 0x5**, va aparea expresia "BID", semnificând ca programul se întrerupe când instructiunea respectiva este în faza de decodificare. Pentru examinarea punctelor de întrerupere deja definite se executa click pe iconita *Breakpoint*. Declansarea simularii se face selectând optiunea *Run* din meniul *Execute* sau tasta F5. O fereastra ne va informa ca ne aflam pe nivelul ID si s-a atins primul

punct de întrerupere. Daca studiem diagrama ciclului de ceas vom observa ca simularea este în ciclul 14, iar instructiunea **trap 0x5** se afla în stagnare.

Motivatia consta în faptul ca nivelele pipeline ale DLX sunt golite de fiecare data când se întâlneste o instructiune trap, pentru a evita toate posibilitatile problemei. Daca vom activa fereastra *Display DLX-IO* din meniul *Execute*, în fereastra va aparea mesajul: " An integer value:".

3.3.5. Fereastra Registru

În aceasta fereastra sunt afisate valorile tuturor registrilor. Este posibila totodata si modificarea continutului registrilor generali - GPR, a registrilor flotanti - FPR, a PC-ului si registrului de stare - FPSR, dar nu si continutul registrilor speciali. Distingem mai multe tipuri de afisare si reprezentare a continutului registrilor: hexadecimal, decimal, flotant simpla si dubla precizie. Selectia unui registru se face prin dublu click pe numele acestuia.

Procesorul DLX are urmatorii registri vizibili în programe:

- GPR (R0..R31) - 32 registri de uz general pe 32 de biti; R0 (ca si la procesorul MIPS R2000) este întotdeauna cablat la 0.
- FPR - set de registri flotanti care pot fi folositi ca 32 de registri în simpla precizie (F0 ÷ F31), sau perechi de registri par-impar în dubla precizie (D0 to D30).
- FPSR - registru de stare folosit atât pentru comparatii cât si pentru exceptii în virgula flotanta. Toate instructiunile de transfer în/din registrul de stare lucreaza cu registrii de uz generali. Instructiunile de salt conditionat testeaza bitul de comparatie din registrul FPSR.

- PC – program counter-ul contine întotdeauna adresa urmatoarei instructiuni ce urmeaza a fi extrasa din memorie. Instructiunile de ramificatie (salturi conditionate sau neconditionate) pot altera continutul PC.

De asemenea, DLX contine si registrii interni, care nu sunt vizibili programelor DLX.

- IMAR - registrul adresei de memorie a instructiunii; este initializat cu continutul program counter-ului în timpul fazei IF. Spre deosebire de PC, acest registru este în conexiune directa cu memoria sistem.
- IR - registrul instructiunii; în timpul fazei IF acest registru se încarca cu urmatoarea instructiune ce va fi executata.
- A, B - registrii operanzi ai ALU; în faza de decodificare cei doi registrii sunt cititi si trimisi unitatii arimetico-logice. WinDLX prezinta, de asemenea, doi pseudoregistrii AHI si BHI care contin cei mai semnificativi 32 de biti ai registrilor cu valori flotante în dubla precizie.
- BTA - registrul de adresa a tintei branch-urilor; în nivelul ID se calculeaza adresa tinta a instructiunilor branch sau jump si se scrie în acest registru. Daca branch-ul este *taken* (se executa salt) adresa se încarca si în program counter.
- ALU - registrul rezultat al operatiilor aritmetico-logice; WinDLX prezinta un pseudoregistru ALUHI ce are acelasi rol ca si registrele AHI si BHI.
- DMAR - registrul adresa de memorie a datei; adresa datelor în cazul referintelor la memorie este transferata în acest registru. Accesul la memorie se face în timpul fazei MEM.
- SDR - registrul de memorare a datei; data care urmeaza a fi scrisa în memorie este transferata în acest registru. WinDLX prezinta, de asemenea, pseudoregistrul SDRHI.

- LDR - registrul de încarcare a datei; data citita din memorie se încarca în acest registru. Registrul LDRHI este destinat operatiilor cu date în flotant dubla precizie.

Pentru a continua simularea activam fereastra de cod si defilam prin ea pâna la linia cu adresa 0x00000194, la instructiunea **lw r2, SaveR2(r0)**. Punem un punct de întrerupere **(Code / Set Breakpoint / Ok)** si pe aceasta linie. Totodata stabilim un punct de întrerupere si pe instructiunea de la adresa 0x000001a4 **jr r31** (PC←(r31)). Continuând simularea cu F5, pe ecran va apare fereastra DLX Standard-IO în care se asteapta introducerea de la tastatura a unui întreg. Dupa citirea unei valori întregi de la tastatura (de exemplu: 20) si confirmarea cu Enter, simularea continua pâna se ajunge la cel de-al doilea breakpoint. Examinand simularea în ciclii 52 ÷ 56, în diagrama ciclului de ceas observam aparitia unor sageti rosii si verzi între instructiuni. Sageata rosie implica necesitatea unei stagnari datorate unui hazard de tip RAW (o instructiune are nevoie de rezultatul unei instructiuni precedente care nu este înca cunoscut). Sagetile verzi sugereaza folosirea mecanismului de forwarding (folosirea rezultatului unei instructiuni înainte ca acesta sa fie scris în registrul general destinatie).

În acest moment vom examina continutul registrilor generali activand fereastra Registru. Cu F5 continuam simularea pâna la urmatorul punct de întrerupere. Daca dorim ca simularea sa avanseze fara stabilirea unor noi puncte de întrerupere selectam optiunea Multiple Cycles din meniul Execute sau simplu F8. În fereastra nou creata se introduce numarul de cicli (de exemplu: 17) care se vor executa fara întrerupere. Defilam apoi prin diagrama ciclului de ceas pâna la cicli instructiune de la 72 la 78. Doua instructiuni în flotant dubla precizie (înmultire si scadere) sunt executate fiecare în unitati separate de executie în timpul fazei (EX), dar ambele necesita mai mult de un ciclu pentru executie. Astfel, instructiunea urmatoare acestora (j Fact.Loop) poate fi adusa din

memorie, decodificata si executata, dar apoi trebuie sa astepte un ciclu pentru a permite instructiuni subd sa încheie faza MEM.

3.3.6. Fereastra Statistica

Fereastra Statistica contine rezultatele statistice obtinute în urma simularii. Datele sunt clasificate în câteva grupuri si au urmatoarele semnificatii.

- Numarul total de cicli executati.
- Numarul de instructiuni care au fost deja transmise nivelului ID.
- Numarul de instructiuni curent executate în pipe.

Meniul Statistica contine unele optiuni pentru a controla afisarea informatiilor în fereastra Statistica:

- *Display* – cu optiunile:
  - *Hardware*
  - *Stalls*
  - *Conditional Branches*
  - *Load/Store-Instructions*
  - *Floating point stages instructions*
  - *Traps*
  - *All*
- *Detail info*
- *Reset*

Optiunea *Display Hardware* permite afisarea urmatoarelor informatii despre configuratia hardware a DLX:

- capacitatea memoriei (în octeti).
- numarul de unitati de executie care opereaza asupra datelor flotante precum si latenta lor.
- starea mecanismului de forwarding (validat sau nu).

Daca optiunea *Display Stalls* este activata vor fi afisate urmatoarele date statistice (relative sau absolute):

- numarul de stagnari datorate hazardurilor de date RAW. Daca mecanismul de forwarding este validat si optiunea *Detail info* este activata, numarul de stagnari va fi împartite dupa tipul instructiunii cauzatoare de stagnare, în: numar de instructiuni de load, numar de instructiuni de salt (branch/jump), numar de instructiuni în virgula mobila.
- numarul de stagnari datorate hazardurilor de date WAW.
- numarul de stagnari datorate hazardurilor structurale înainte de intrarea în nivelul de executie cu numere flotante.
- numarul de stagnari datorate executiei instructiunilor de salt conditionat (branch-uri taken).
- numarul de stagnari datorate instructiunilor trap.

Optiunea *Display Conditional Branches* determina afisarea informatiilor (relative sau absolute) referitoare la instructiunile de salt conditionat.

- numarul de salturi conditionate. Daca optiunea *Detail info* este activata, informatia este separata în numar de instructiuni de salt conditionat care se fac si numar de instructiuni de salt care nu se fac.

Selectând *Display Load/Store* în fereastra vor fi afisate numarul de instructiuni Load si Store executate. Activând si *Detail info* informatia va fi împartita în doua: numar instructiuni Load si numar instructiuni Store.

Daca optiunea *Display Floating Point Stage Instructions* este activa, se pot obtine informatii despre numarul total de instructiuni executate care au folosit nivelele de executie în virgula flotanta (faddEX, fmulEX oder fdivEX). Daca e validata optiunea *Detail info*, în fereastra informatia va apare segmentat în:

- numarul de instructiuni care au trecut prin nivelul faddEX.

- numarul de instructiuni care au trecut prin nivelul fmulEX.
- numarul de instructiuni care au trecut prin nivelul fdivEX.

Numarul de instructiuni Trap executate este afisat cu ajutorul optiunii *Display Traps*.

Optiunea *Display All* implica afisarea tuturor grupurilor de date statistice.

Selectând *Detail Info* se permite afisarea a unor informatii detaliate în fereastra Statistica.

Pentru o reluare a testelor este necesara resetarea tuturor parametrilor din fereastra Statistica. De asemenea, la pornirea sau restartarea procesorului este necesara o resetare a parametrilor statistici. Acest lucru se face selectând optiunea *Reset*. Ultimul ciclu de ceas simulat devine 0. Istoria si continutul pipeline-ului vor ramâne neschimbate.

Continuam exemplificarea pe programul de test, urmarind la finele executiei programului actualizarea parametrilor în fereastra Statistica.

Executia programului continua cu F5. Pe ecran va apare o caseta cu mesajul "Trap #0 occurred", aratând ca instructiunea **trap 0** a fost executata. Instructiunea **trap 0** nu este definita, ea fiind folosita pentru a sugera terminarea programului. Se confirma cu *OK* si se maximizeaza fereastra Statistica.

Fereastra Statistica este extrem de folositoare pentru a compara efectele modificarilor de configuratie asupra performantei procesorului (numar total cicli de executie). Vom examina avantajul introducerii mecanismului de forwarding. Folosind acest mecanism am obtinut urmatoarele rezultate: numarul total de cicli – 215, stagnari datorate (RAW – 17, Control – 25, Trap - 12), în total 54 de cicli. Închidem fereastra Statistica si vom reconfigura hardware procesorul. Pentru dezactivarea forwarding-ului se inhiba optiunea *Enable Forwarding*. Urmatorul avertisment care va apare pe ecran: "OK resets automatically the processor! Disable Forwarding?" trebuie confirmat cu *OK*. Stergem toate punctele de

întrerupere cu optiunea *Delete All* din meniul *Breakpoint*. Executam simularea benchmark-ului fara întrerupere astfel: *F5, 20* (numarul al carui factorial se calculeaza) *Enter* si *OK* pâna se executa instructiunea **trap 0**. Prin reexaminarea ferestrei Statistica observam ca, numarul de stagnari datorate instructiunilor de salt si trap ramâne acelasi dar numarul de stagnari datorate hazardurilor RAW creste de la 17 la 53, determinând cresterea numarului total de cicli de simulare la 236 fata de 215. Cunoscând aceasta informatie putem calcula *speed-up-ul* (câstigul de performanta obtinut prin forwarding). Raportul 236 / 215 = 1.098 este interpretat astfel: DLX cu forwarding este cu 9.8% mai rapid decât fara forwarding, pe benchmark-ul **fact.s**.

Concluzii

Lucrarea prezinta caracteristicile importante ale procesorului DLX, urmarind întelegerea conceptului de pipeline în general si a modului de operare a procesorului DLX în particular. Configuratia poate suferi modificari. Este interesant de observat daca introducerea unui sumator în virgula mobila este de folos, sau daca o unitate de împartire mai rapida (executia instructiunilor se face în mai putini cicli) justifica costul suplimentar. Ulterior, pot fi simulate efectele unei compilari optimizate prin rearanjarea liniilor în codul sursa, evitând astfel stagnarile datorate hazardurilor RAW.

Bibliografie

[1] **Hennessy J., Patterson D.** - *Computer Organization and Design*: *The Hardware / Software Interface*, Morgan Kaufmann Publishers, San Francisco, California, 1994

[2] **Gründbacher H. -** *Windows Deluxe Simulator - Tutorial*, University of Technology, Viena, 1992