Probleme propuse pentru examenul de la disciplina Structura şi Organizarea Calculatoarelor

== MOODLE TYPE ==

Prof. dr. ing. Vasile Gheorghiță GĂITAN

Ş. l. dr. ing. Ionel ZAGAN

Cap. 1 Introducere

1. Fie două implementări diferite M1 şi M2 ale aceluiaşi set de instrucţiuni. Există trei clase de instrucţiuni (A, B şi C) în setul de instrucţiuni. M1 are un ceas de 80 de MHz şi M2 are un ceas de 100 MHz. Numărul mediu de cicli pentru fiecare clasă de instrucţiuni şi frecvenţa lor (pentru un program tipic) sunt după cum urmează:

Instruction Class	Machine M1 – Cycles/Instruction Class	Machine M2 – Cycles/Instruction Class	Frequency
A	1	2	60%
B	2	3	30%
C	4	4	10%

   a) Calculează CPI mediu pentru fiecare maşină M1 şi M2.
   b) Calculează MIPS mediu pentru fiecare maşină M1 şi M2.
   c) Care maşină are raportul MIPS mai mic? Pentru care clasă individuală de instrucţiuni trebuie să schimbi CPI-ul, şi cât de mult, pentru ca această maşină să aibă aceleaşi performanţe sau mai bune faţă de maşina cu MIPS-ul mai mare (poţi schimba doar CPI-ul pentru una dintre clasele de instrucţiuni de pe maşina mai lentă)?

2. Presupuneţi că aveţi un procesor care execută un program constând din 50% înmulţiri în virgulă flotantă, 20% împărţiri în virgulă flotantă, iar restul de 30% sunt alte instrucţiuni (legea lui Amdahl).
   a) Managementul doreşte ca acest procesor să ruleze de 4 ori mai repede. Se poate face ca împărţirea să meargă de 3 ori mai repede şi înmulţirea de 8 ori mai repede. Puteţi îndeplini cerinţa managementului doar cu o singură îmbunătăţire, şi care este aceasta?
   b) Dogbert a preluat compania înlăturând toţi managerii precedenţi. Dacă realizaţi ambele îmbunătăţiri, la înmulţire şi la împărţire, care este viteza procesorului astfel îmbunătăţit relativ la procesorul original?

3. Presupunem că putem îmbunătăţii performanţele în virgulă flotantă ale unui procesor cu un factor având valoarea 15 (aceleaşi instrucţiuni în virgulă flotantă sunt de 15 ori mai rapide pe noul procesor). Ce procentaj dintre instrucţiuni trebuie să fie în virgulă mobilă pentru a atinge o îmbunătăţire a vitezei de cel puţin 4 ori?

4. Aşa cum este definit MIPS-ul se poate defini MFLOPS-ul care înseamnă milioane de instrucţiuni în virgulă flotantă per secundă. Dacă procesorul A are un MIPS mai mare decât procesorul B, atunci în mod obligatoriu are şi un MFLOPS mai mare decât B?

5. Se consideră benchmark-ul SPEC. Numiţi doi factori care influenţează performanţele rezultate pentru orice arhitectură particulară.

6. Cum a afectat calculatorul dezvoltarea tranzistorilor? Ce a înlocuit tranzistorul?

7. Presupunem că o echipă de proiectare realizează îmbunătăţirea unui procesor prin adăugarea unui modul hardware MMX (multimedia extension instruction). Atunci când un calcul se realizează în modul MMX pe hardware-ul MMX acesta este de 10 ori mai rapid decât în modul normal de execuţie. Se va numi timpul petrecut utilizând modul MMX, procentul de îmbunătăţire media.
   a. Ce procent mediu de îmbunătăţire este necesar pentru a obţine o îmbunătăţire generală a vitezei de 2?
   b. Ce procent din timpul de execuţie este petrecut în modul MMX dacă se realizează ă mărire de două ori a vitezei ? (Sfat: trebuie calculat noul timp general).
   c. Ce procent de îmbunătăţire mediu este necesar pentru a obţine o jumătate din maximum de mărire a vitezei (one-half) care se poate obţine utilizând modul MMX?

8.

a. Dacă procesorul A are ceasul mai mare ca procesorul B, şi procesorul A are şi MIPS-ul mai mare decât procesorul B, explicaţi când procesorul A va fi mai rapid decât procesorul B. Presupuneţi că există două implementări ale aceleiaşi arhitecturi a setului de instrucţiuni.
b. Procesorul A are ciclul de ceas de 20ns şi un CPI efectiv de 1,5 pentru unele programe, şi procesorul B are ciclul de ceas de 15ns şi un CPI efectiv de 1 pentru unele programe. Care procesor este mai rapid pentru acest program şi cu cât?

9. Presupunem că un segment de program constă dintr-o parte pur secvenţială care ia pentru execuţie 25 de cicli, şi o buclă iterativă care ia 100 de cicli per iteraţie. Presupuneţi că iteraţiile din buclă sunt independente, şi nu pot fi paralelizate. Dacă bucla se execută de 100 de ori, care este creţterea maximă de viteza posibilă dacă se utilizează un număr infinit de procesoare (în comparaţie cu un singur procesor)

10. Procesorul A are un CPI general de 1,3 şi poate rula la o frecvenţă de ceas de 600 MHz. Procesorul B are un CPI de 2,5 şi poate rula la o frecvenţă de ceas de 750MHz. Avem un anumit program pe care dorim să-l executăm. În urma compilării pe procesorul A rezultă 100.000 de instrucţiuni. Câte instrucţiuni trebuie să aibă programul compilat pe procesorul B, pentru ca cele două procesoare să aibă, pentru acest program, exact acelaşi timp de execuţie?

11. Imaginaţi-vă că puteţi realiza o cursă bazată pe benchmark pentru a compara două calculatoare pe care credeţi că vreţi să le cumpăraţi. Precizaţi o listă cu 5 programe de benchmark sau un scenariu de utilizare pe care vreţi să-l folosiţi pentru a crea o suită proprie, personalizată de benchmark. Pentru fiecare program l-aţi selectat, justificaţi-l. Pentru suita de benchmark-uri ca întreg, discutaţi o metodă pentru calculul mediei ponderate a diferitelor programe în execuţie.

12. O echipă de proiectanţi trebuie să aleagă pentru un procesor simplu cu o singură instrucţiune pe eeeciclu, între o implementare pipeline sau fără pipeline. În tabel sunt prezentaţi câţiva parametrii de proiectare pentru cele două opţiuni.

Parameter	Pipelined Version	Non-Pipelined Version
Clock Rate	500MHz	350 MHz
CPI for ALU instructions	1	1
CPI for Control instructions	2	1
CPI for Memory instructions	2.7	1

a. Pentru un program cu 20% instrucţiuni ALU instrucţiuni, 10% instrucţiuni de control şi 70% ezinstrucţiuni de lucru cu memoria, care va fi proiectul mai rapid? Furnizaţi valoarea medie a CPI pentru fiecare caz.
b. Pentru un program cu 80% instrucţiuni ALU instrucţiuni, 10% instrucţiuni de control şi 10% instrucţiuni de lucru cu memoria, care va fi proiectul mai rapid? Furnizaţi valoarea medie a CPI pentru fiecare caz.

13. Un proiectant doreşte să îmbunătăţească performanţele generale a unui procesor dat cu respectarea unei ţinte de tip suită benchmark şi se ia în considerare o îmbunătăţire X care se aplică la 50% instrucţiunilor originale executate dinamic, şi le măreşte viteza de execuţie fiecăreia dintre ele cu un factor 3. Managerul de proiectare are unele îngrijorări despre complexitate şi costul efectiv pentru X, şi sugerează ca proiectantul să ia în considerare o îmbunătăţire Y alternativă. Îmbunătăţirea Y, dacă se aplică numai la câteva dintre instrucţiunile executate dinamic le va face doar cu 75% mai rapide. Determinaţi ce procent dintre toate instrucţiunile executate dinamic trebuie îmbunătăţite utilizând optimizarea Y pentru a obţine aceeaşi creştere de viteză propusă de îmbunătăţirea X.

14. Descrieţi paşii care transformă un program scris într-un limbaj de nivel înalt, precum C, într-o reprezentare care este direct executată de procesor.

15. Presupuneţi că un afişor color utilizează 8 biţi pentru fiecare dintre culorile de bază (roşu, verde şi albastru) per pixel şi un cadru având mărimea de 1280 x 1024.
    a. Care este mărimea minimă în octeţi al buffer-ului pentru cadru care memorează cadrul?
    b. Când de mult durează trimiterea unui cadru pe o reţea cu viteza de 100 Mbit/s?

16. Se consideră trei procesoare P1, P2 şi P3 care execută acelaşi set de instrucţiuni. P1 are frecvenţa de ceas de 3 GHz şi un CPI de 1,5, P2 are 2,5 GHz şi un CPI de 1 şi P3 are 4GHz şi un CPI de 2,2.
    a. Care procesor are performanţele cele mai mari exprimată în MIPS?
    b. Dacă fiecare procesor execută un program în 10 secunde, calculaţi numărul de cicli şi numărul de instrucţiuni.
    c. S-a încercat reducerea timpului cu 30% dar aceasta a condus la o creştere cu 20% a CPI-ului. Care frecvenţă de ceas trebuie pentru a avea această reducere?

17. Se consideră două implementări diferite ale aceleiaşi arhitecturi a setului de instrucţiuni. Instrucţiunile pot fi divizate în 4 clase în funcţie de CPI (clasele A, B, C şi D). P1 are frecvenţa de ceas de 2,5 GHz şi CPI-urile 1,2,3 şi 3, iar P2 are un ceas de 3 GHz ţi CPI-urile 2,2,2 şi 2. Dându-se un program cu contorul dinamic de instrucţiuni de 1.0E6 instrucţiuni divizate în clase după cum urmează: clasa A 10%, clasa B 20%, clasa C 50%, şi clasa D 20%, care implementare este mai rapidă?
    a. Care este CPI-ul global pentru fiecare implementare?
    b. Găsiţi ciclurile de ceas necesare în ambele cazuri.

18. Compilatoarele pot avea un impact profund asupra performanţelor unei aplicaţii. Presupunem că pentru un program, compilatorul A generează un contor dinamic pentru instrucţiuni de 1.0E9 şi are un timp de execuţie de 1,1s, în timp ce compilatorul B generează un contor dinamic pentru instrucţiuni de 1.2E9 şi are un timp de execuţie de 1,5s.
    a. Găsiţi CPI-ul mediu pentru fiecare program dat fiind faptul că durata unui ciclu de ceas al procesorului este de 1ns.
    b. Presupuneţi că programele compilate se execută pe două procesoare. Dacă timpul de execuţie pe cele două procesoare este acelaşi, cât de rapid este ceasul procesorului care rulează codul compilat de compilatorul A în raport cu ceasul procesorului care rulează codul compilat de compilatorul B?
    c. Se dezvoltă un nou compilator care utilizează doar 6.0E8 instrucţiuni şi care are un CPI mediu de 1,1. Care este creşterea de viteză utilizând acest compilator versus compilatoarele A şi B de pe procesorul original?

19. Procesorul Pentium IV Prescott, realizat în 2004, avea un ceas de 3,6 GHz, la o tensiune de lucru de 1,25V. Presupuneţi că, pe medie, acesta consuma 10W de energie statică şi 90W de energie dinamică. Nucleul I5 Ivy Bridge, realizat în 2012, are un ceas de 3,4 GHz la o tensiune de 0,9V. Presupuneţi că în medie consumă 30W de putere statică şi 40W de putere dinamică.
    a. Pentru fiecare procesor găsiţi încărcare capacitivă medie.
    b. Găsiţi procentul din puterea totală disipată ce corespunde puterii staticii şi raportul dintre puterea statică şi cea dinamică pentru fiecare tehnologie.
    c. Dacă puterea totală disipată este redusă cu 10%, cât de mult trebuie redusă tensiunea pentru a menţine acelaşi curent de scurgere? Notă: puterea este definită ca produsul între tensiune şi curent.

20. Presupuneţi că un procesor are pentru instrucţiunile aritmetice, load/store şi de salt un CPI de 1,2, respectiv 5. Presupuneţi de asemenea că un program executat pe un singur procesor necesită execuţia a 2,56E9 instrucţiuni aritmetice, 1,28E9 instrucţiuni load/store şi 256 de milioane de instrucţiuni de salt. Presupuneţi că fiecare procesor are un ceas de 2 GHz. Presupuneţi că, dacă programul este paralelizat pentru a se executa pe nuclee multiple, numărul de instrucţiuni aritmetice şi load/store per procesor este divizat cu 0,7 x p (unde p este numărul de procesoare), dar numărul de salturi per procesor rămâne acelaşi.

a. Calculaţi timpul total de execuţie pentru acest program pe 1, 2, 4 şi 8 procesoare, şi prezentaţi creşterea relativă de viteză a celor 2, 4 şi 8 procesoare la un singur procesor.
b. Dacă CPI-ul instrucţiunilor aritmetice se dublează care este impactul execuţiei programului pe 1, 2, 4 şi 8 procesoare?
c. La ce valoare trebuie redus CPI-ul instrucţiunilor load/store pentru un singur procesor pentru ca acesta să aibă aceleaşi performanţe cu patru procesoare ce utilizează valorile originale ale CPI?

21. Presupunem un wafer de 15 cm diametru care are un cost de 12, conţine 84 de die, şi are 0,020 defecte/cm^2. Presupuneţi ca un wafer de 20 de cm diametru are un cost de 15, conţine 100 de die, şi are 0,031 defecte/ cm^2.
    a. Calculaţi yield pentru ambele wafer-e.
    b. Calculaţi costul per die pentru ambele wafer-e.
    c. Dacă numărul de die per wafer este crescut cu 10% şi defectele pe unitatea de arie cresc cu 15%, calculaţia aria die şi yield.
    d. Presupuneţi că un proces de fabricaţie îmbunătăţeşte yield de la 0,92 la 0,95. Calculaţi defectele pe unitatea de arie pentru fiecare versiune a tehnologiilor, fiind dată aria die ca fiind 200 mm^2.

22. Rezultatele pentru benchmark-ul SPEC CPU2006 bzip2 ce rulează pe un AMD Barcelona are un contor de instrucţiuni de 2,389E12, un timp de execuţie de 750s şi un timp de referinţă de 9650s.
    a. Calculaţi CPI-ul dacă perioada ceasului este de 0,333ns.
    b. Calculaţi SPECratio.
    c. Calculaţi creşterea timpului CPU dacă numărul de instrucţiuni ale banchmark-ului creşte cu 10% fără a afecta CPI-ul.
    d. Calculaţi creşterea timpului CPU dacă numărul de instrucţiuni ale banchmark-ului creşte cu 10% iar CPI-ul creşte cu 5%.
    e. Calculaţi modificările SPECratio pentru această schimbare.
    f. Presupuneţi că dezvoltaţi o nouă versiune de procesor AMD Barcelona cu un ceas de 4 GHz. Aţi adăugat câteva instrucţiuni adiţionale setului de instrucţiuni astfel încât numărul de instrucţiuni a fost redus cu 15%. Timpul de execuţie s-a redus la 700s şi noul SPECratio este 13,7. Găsiţi noul CPI.
    g. Această valoare este mai mare decât aceea obţinută la 1.11.1 deşi ceasul a crescut de la 3 la 4 GHz. Determinaţi situaţia în care creşterea CPI este similară creşterii frecvenţei ceasului. Dacă nu sunt similare explicaţi de ce.
    h. Cu cât s-a redus timpul CPU?
    i. Pentru un al doilea benchmark, libquantum, presupuneţi că timpul de execuţie este 960ns, CPI de 1,61,şi ceasul este de 3 GHz. Dacă timpul de execuţie este redus cu 10% fără a afecta CPI-ul şi cu un ceas de 4 GHz, determinaţi numărul de instrucţiuni.
    j. Determinaţi ceasul necesar pentru o nouă reducere cu 10% a timpului CPU menţinând numărul de instrucţiuni şi CPI-ul.
    k. Determinaţi ceasul dacă CPI se reduce cu 15% şi timpul CPU cu 20%, în timp ce numărul de instrucţiuni rămâne neschimbat.

23. Secţiunea 1.10 citează ca şi capcană utilizarea unui subset al ecuaţiilor pentru performanţă ca metrică pentru performanţă. Pentru a ilustra aceasta să considerăm următoarele două procesoare. P1 are un ceas de 4GHz, un CPI mediu de 0,9 şi necesită execuţia a 5.0E9 instrucţiuni. P2 are un ceas de 3 GHz, un CPI mediu de 0,75, şi necesită execuţia a 1,0E9 instrucţiuni.
    a. O interpretare greşită uzuală este să considerăm că procesorul cu ceasul mai mare are performanţe mai bune. Verificaţi dacă este adevărat pentru P1 şi P2.
    b. O altă interpretare greşită este aceea de a considera că procesorul care execută mai multe instrucţiuni va avea un timp CPU mai mare. Considerând că procesorul P1 execută o secvenţă de 1,0E9 şi că CPI-ul celor două procesoare nu se schimbă, determinaţi numărul de instrucţiuni pe care le poate executa P2 în acelaşi timp necesar ca P1 să execute cele 1,0E9 instrucţiuni.
    c. O altă interpretare greşită comună este aceea a utilizării MIPS pentru a compara performanţele a două procesoare, şi se consideră că procesorul cu MIPS mai mare este mai performant. Verificaţi dacă este valabil pentru P1 şi P2.
    d. O altă interpretare greşită este utilizarea MFLOPS. Presupuneţi că 40% din instrucţiunile executate atât de P1 cât şi de P2 sunt instrucţiuni în virgulă flotantă. Găsiţi interpretarea MFLOPS greşită pentru programe.

24. O altă capcană citată în secţiunea 1.10 este aceea că ne aşteptăm să îmbunătăţim performanţele generale ale unui calculator prin îmbunătăţirea doar a unui aspect. Să considerăm un calculator

care rulează un program care necesită 250s, cu 70s petrecute executând instrucţiuni FP, 85 de secunde executând instrucţiuni L/S.şi 40s executând instrucţiuni de salt.

a. Cât de mult se reduce timpul total dacă timpul pentru operaţiile FP se reduce cu 20%?
b. Cât de mult se reduce timpul pentru operaţiile INT dacă timpul total se reduce cu 20%?
c. Poate fi redus timpul total cu 20% reducând numai operaţiile de salt?

25. Presupuneţi că un program necesită execuţia a 50 x 106 instrucţiuni FP, 110 x 106 instrucţiuni INT, 80 x 106 instrucţiuni L/S, şi 16 x 106 instrucţiuni de salt. CPI pentru fiecare tip de instrucţiune este de respectiv 1, 1, 4, şi 2. Presupuneţi că procesorul are ceasul de 2 GHz.
    a. Cât de mult trebuie îmbunătăţit CPI-ul instrucţiunilor FP dacă dorim ca programul să ruleze de două ori mai repede?
    b. Cât de mult trebuie îmbunătăţit CPI-ul instrucţiunilor L/S dacă dorim ca programul să ruleze de două ori mai repede?
    c. De câte ori este îmbunătăţit timpul de execuţie al programului dacă CPI pentru instrucţiunile INT şi FP este redus cu 40% şi CPI pentru instrucţiunile L/S şi de salt este redus cu 30%.

26. Atunci când un program este adaptat să se execute pe procesoare multiple într-un sistem multiprocesor, timpul de execuţie pe fiecare procesor este alcătuit din timpul de calcul şi timpul dat de supracontrolul necesar pentru a bloca secţiunile critice şi/sau trimiterea datelor de la un procesor la altul. Presupuneţi un program care necesită 100s de execuţie pe un singur procesor. Atunci când se rulează pe p procesoare necesită t/ps precum şi 4s adiţionale pentru supracontrol, indiferent de numărul de procesoare. Calculaţi timpul de execuţie per procesor pentru 1, 2, 4, 8, 16, 32, 64, şi 128 de procesoare. Pentru fiecare caz, listaţi creşterea relativă de viteză la un singur procesor şi raportul dintre creşterea de viteză actuală şi creşterea de viteză ideală (fără supracontrol).

Probleme propuse pentru examenul de la disciplina Structura şi Organizarea Calculatoarelor

== MOODLE TYPE ==

Prof. dr. ing. Vasile Gheorghiță GĂITAN

Ş. l. dr. ing. Ionel ZAGAN

Cap. 2 Instrucţiunile: limbajul calculatorului

1. Înainte de anii 1980, calculatoarele erau construite cu seturi instrucţiuni din ce în ce mai complexe. MIPS este un calculator RISC. De ce a existat o mutare de la calculatoarele cu set complex de instrucţiuni la calculatoarele RISC.

2. Scrieţi următoarea secvenţă în cod assembler MIPS.
   x = x + y + z – q.
   Presupuneţi că x,y,z,q sunt memorate în $s1 - $s4.

3. În assemblerul MIPS, scrieţi o versiune în limbaj de asamblare a următorului segment de cod C:
   ```
   int    A[100], B[100];
   for (i=1; i<100, i++) {
    A[i] = A[i-1] +B[i];
   }
   ```
   La începutul acestui segment de cod, singurele valori din regiştrii sunt adresele de bază ale ariilor A şi B în regiştrii $a0 şi $a1. Evitaţi utilizarea instrucţiunilor de înmulţire.

4. Unele calculatoare un registru special pentru indicatori care conţin indicatori de stare. Aceşti biţi sunt includ adesea biţii *carry* şi *overflow*. Descrieţi diferenţa între funcţionalităţile acestor doi biţi şi daţi un exemplu de operaţie aritmetică care să conducă la starea lor pe valori diferite.

5. Setul de instrucţiuni MIPS include câteva instrucţiuni de deplasare. Acestea sunt deplasare logică la stânga, deplasare logică la dreapta, şi deplasare aritmetică la dreapta. Alte arhitecturi includ numai o instrucţiune de deplasare aritmetică la dreapta.
   a. De ce nu conţine MIPS un opcode pentru deplasare aritmetică la stânga?
   b. Cum se poate implementa în assembler o pseudo-operaţie de deplasare logică la stânga (LSL) pentru un procesor care nu are această instrucţiune? Fiţi siguri că instrucţiunea voastră LSL poate realiza deplasarea până la W-biţi, unde W este mărimea în biţi a cuvântului procesor.

6. Consideraţi următorul cod în assembler pentru partea 1 şi partea 2.
   ```
   r1 = 99

   Loop:
     r1 = r1 – 1
     branch r1 > 0, Loop
     halt
   ```

   a. Pe durata execuţiei codului de mai sus câte instrucţiuni dinamice sunt executate?
   b. Presupuneţi ca aveţi un procesor uniciclu (cu un singur ciclu procesor) standard funcţionând la 100 KHz, cât de lung va fi codul de mai sus până la completare?

7. Convertiţi funcţia C de mai jos în limbaj de asamblare MIPS. Fiţi siguri că, codul vostru în limbaj de asamblare poate fi apelat de un program Cl standard (altfel spus, fiţi sigur că respectaţi convenţiile MIPS de apel).

   ```
   unsigned int sum(unsigned int n)
   {
     if (n == 0) return 0;
     else return n + sum(n-1);
   }
   ```
   Acest procesor nu are sloturi de întârziere. Stiva creşte spre adrese mai mici de memorie. Următorii regiştrii sunt utilizaţi în convenţiile de apel:

Numele registrului	Numărul registrului	Utilizare
$zero	0	Constanta 0
$at	1	Rezervat pentru asamblor

$v0, $v1	2, 3	Valorile de revenire ale funcţiei
$a0 - $a3	4 – 7	Valorile argumentelor funcţiei
$t0 - $t7	8 – 15	Temporar (salvate de apelant)
$s0 - $s7	16 – 23	Temporar (salvate de apelat)
$t8, $t9	24, 25	Temporar (salvate de apelant)
$k0, $k1	26, 27	Rezervate pentru ncleul OS
$gp	28	Pointer la aria globală
$sp	29	Pointerul de stivă - Stack Pointer
$fp	30	Pointerul de cadru Frame Pointer
$ra	31	Adresa de revenire

8. În secvenţa de cod MIPS ce urmează, de câte ori este accesată instrucţiunea de acces la memorie? De câte ori este accesată memoria de date? (Număraţi doar accesele la memorie nu şi la regiştrii).

```
lw   $v1, 0($a0)
addi $v0, $v0, 1
sw   $v1, 0($a1)
addi $a0, $a0, 1
```

9. Utilizaţi valorile din regiştrii şi memorie din tabelul următor pentru următoarea întrebare. Presupuneţi un procesor pe 32 de biţi. Presupuneţi că fiecare din următoarele întrebări porneşte de la valorile din tabel; nu utilizaţi valorile modificate de o întrebare ca propagate în viitoarea parte a întrebării.

Registru	Valoare	Locaţie de memorie	Valoare
R1	12	12	16
R2	16	16	20
R3	20	20	24
R4	24	24	28

   a. Daţi valorile pentru R1, R2 şi R3 după execuţia instrucţiunii add R3, R2, R1.
   b. Ce valori vor fi în R1 şi R3 după execuţia instrucţiunii load R3, 12 (r1).
   c. Ce valori vor fi în regiştrii după execuţia instrucţiunii addi R2,R3,#16.

10. Loop unrolling and Fibonacci: fie următorul pseudo C-cod pentru a calcula numărul 5 Fibonacci (F(5)).

```
1 int a,b,i,t;
2 a=b=1;              /* Set a and b to F(2) and F(1) respectively */
3 for(i=0;i<2;i++)
4 {
5 t=a;                /* save F(n-1) to a temporary location */
6 a+=b;               /* F(n) = F(n-1) + F(n-2) */
7 b=t;                /* set b to F(n-1) */
8 }
```

Codul mai poate fi scris şi astfel:

```
1 int a,b,t;
2 a=b=1;
3 t=a;
4 a+=b;
5 b=t;
6 t=a;
7 a+=b;
8 b=t;
```

   a. Convertiţi pseudo C-codul pentru ambele secvenţe de mai sus într-un cod MIPS rezonabil de eficient. Reprezentaţi fiecare variabilă din pseudo C-cod cu un registru. Încercaţi să urmăriţi pseudocodul cât mai aproape posibil (prima secvenţă trebuie să conţină o buclă, în timp ce a doua nu).

b. Presupuneţi acum că în loc de numărul 5 Fibonacci decideţi să îl calculaţi pe al 20-lea. Cât de multe instrucţiuni statice vor fi în prima versiune şi cât de multe vor fi în versiunea unrolled? Ce puteţi spune despre instrucţiunile dinamice? Aici nu trebuie scris codul în asamblare.

11. Scrieţi în limbajul de asamblare MIPS următorul segment de cod C:
```
for (i = 0; i < 98; i ++) {
  C[i] = A[i + 1] - A[i] * B[i + 2]
}
```
Ariile A,B şi C sunt plasate la adresele de memorie 0xA000, 0xB000 şi respectiv 0xC000. Încercaţi să deduceţi numărul total de instrucţiuni şi numărul de instrucţiuni scumpe precum înmulţirea.

12. Să presupunem că se implementează o nouă instrucţiune MIPS, denumită bcp, pentru a copia un bloc de cuvinte de la o adresă la alta. Presupunem că această instrucţiune cere ca adresa de start a blocului sursă să fie în registrul $t1, iar adresa blocului destinaţie în registrul $t2. Instrucţiunea necesită şi numărul de cuvinte ce urmează a fi copiate şi pentru aceasta se foloseşte registrul $t3 (>0). Mai mult presupunem că valorile acestor regiştrii precum şi a registrului $t4 sunt distruse de această instrucţiune. Realizaţi următoarele: Scrieţi cod în limbajul de asamblare MIPS pentru a implementa copierea unui bloc fără această instrucţiune. Scrieţi cod în limbajul de asamblare MIPS pentru a implementa copierea unui bloc cu această instrucţiune. Estimaţi numărul total de cicli necesari pentru fiecare realizare pentru a copia 100 de cuvinte pe un procesor multiciclu.

13. Care este codul în limbajul de asamblare MIPS pentru următorul program în C? Presupuneţi că variabilele f, g, h, şi i sunt date şi pot fi considerate întregi pe 32 de biţi aşa cum se declară în programul în C. Utilizaţi un număr minim de instrucţiuni în assembler MIPS.
f = g + (h – 5).

14. Fiind dat următorul cod în limbajul de asamblare MIPS, care este codul corespunzător în C?
```
add f, g, h
add f, i, f
```

15. Care este codul în limbajul de asamblare MIPS pentru următorul program în C? Presupuneţi că variabilele f, g, h, i, şi j sunt asignate regiştrilor $s0, $s1, $s2, $s3, şi respectiv $s4. Presupuneţi că adresele de bază ale ariilor A şi B sunt în regiştrii $s6 şi respectiv $s7.
B[8] = A[i–j];

16. Care sunt liniile C corespunzătoare codului în limbaj de asamblare MIPS de mai jos? Presupuneţi că variabilele f, g, h, i, şi j sunt asignate regiştrilor $s0, $s1, $s2, $s3, şi respectiv $s4. Presupuneţi că adresa de bază a ariilor A şi B sunt în regiştrii $s6 şi respectiv $s7.

```
sll  $t0, $s0, 2     # $t0 = f * 4
add  $t0, $s6, $t0   # $t0 = &A[f]
sll  $t1, $s1, 2     # $t1 = g * 4
add  $t1, $s7, $t1   # $t1 = &B[g]
lw   $s0, 0($t0)     # f = A[f]
addi $t2, $t0, 4
lw   $t0, 0($t2)
add  $t0, $t0, $s0
sw   $t0, 0($t1)
```

17. Pentru codul în limbaj de asamblare din exerciţiul 16, rescrieţi codul pentru a minimiza numărul de instrucţiuni MIPS (dacă este posibil) necesare pentru a realiza aceeaşi funcţie.

18. Tabelul următor prezintă valorile pe 32 de biţi a unei arii stocate în memorie.

Adresa	Data
24	2
38	4
32	3
36	6
40	1

a. Pentru locaţiile de memorie din tabelul de mai sus scrieţi cod în C pentru a sorta data de la valoarea cea mai mică spre valoarea cea mai mare, plasând valoarea cea mai mică la adresa de memorie cea mai mică. Presupuneţi că datele prezentate în tabel reprezintă o variabilă C denumită Array, care este o arie de tip int, şi că primul număr din arie este primul element din aceasta. Presupuneţi că procesorul este cu adresare la nivel de octet şi că un cuvânt constă din patru octeţi.
b. Pentru locaţiile de memorie din tabelul de mai sus scrieţi cod în limbajul de asamblare MIPS pentru a sorta data de la valoarea cea mai mică spre valoarea cea mai mare, plasând valoarea cea mai mică la adresa de memorie cea mai mică. Utilizaţi un număr minim de instrucţiuni MIPS. Presupuneţi că adresa de bază a Array este memorată în registrul $s6.

19. Prezentaţi modul în care valoarea 0xabcdef12 va fi aranjată în memorie pe un procesor little-endian respectiv big-endian. Presupuneţi că data este aranjată începând cu adresa 0.

20. Translataţi în zecimal valoarea 0xabcdef12.

21. Translataţi următorul cod în limbaj de asamblare MIPS. Presupuneţi că variabilele f, g, h, i, şi j sunt asignate regiştrilor $s0, $s1, $s2, $s3, şi respectiv $s4. Presupuneţi că adresele de bază a ariilor A şi B sunt în regiştrii $s6 şi respectiv $s7. Presupuneţi cî elementele ariilor A şi B sunt cuvinte pe 4 octeţi: B[8] = A[i] + A[j].

22. Translataţi următorul cod scris în limbaj de asamblare MIPS în cod C. Presupuneţi că variabilele f, g, h, i, şi j sunt asignate regiştrilor $s0, $s1, $s2, $s3, şi respectiv $s4. Presupuneţi că adresele de bază pentru ariile A şi B sunt în regiştrii $s6 şi respectiv $s7.
```
addi  $t0, $s6, 4
add  $t1, $s6, $0
sw   $t1, 0($t0)
lw   $t0, 0($t0)
add $s0, $t1, $t0
```

23. Pentru fiecare instrucţiune MIPS precizaţi valoarea opcode-ului (OP), şi câmpurile pentru registrul sursă (RS), şi registrul destinaţie (RT). Pentru instrucţiunile I-type precizaţi valoarea câmpului imediat, şi pentru instrucţiunile R-type precizaţi valoarea câmpului pentru registrul destinaţie (RD).

24. Presupuneţi că registrii $s0 ţi $s1 memorează valorile 0x80000000 şi respectiv 0xD0000000.

a. Care este valoarea din registrul $t0 după execuţia următorului cod în limbaj de asamblare MIPS?
    a. add $t0, $s0, $s1
b. Rezultatul din $t0 este rezultatul dorit sau a apărut un overflow?
c. Pentru valorile din regiştrii $s0 şi $s1 specificate anterior care este valoarea din registrul $t0 după execuţia următorului cod în limbaj de asamblare MIPS?
    a. sub $t0, $s0, $s1
d. Rezultatul din $t0 este rezultatul dorit sau a apărut un overflow?
e. Pentru valorile din regiştrii $s0 şi $s1 specificate anterior care este valoarea din registrul $t0 după execuţia următorului cod în limbaj de asamblare MIPS?
    a. add $t0, $s0, $s1
    b. add $t0, $t0, $s0
f. Rezultatul din $t0 este rezultatul dorit sau a apărut un overflow?

25. Presupunem că $s0 conţine valoarea holds the value 128_{10}.
a. Pentru instrucţiunea $t0, $s0, $s1, care este gama (-ele) valorilor din $s1 pentru care va rezulta overflow?
b. Pentru instrucţiunea sub $t0, $s0, $s1, care este gama (-ele) valorilor din $s1 pentru care va rezulta overflow?
c. Pentru instrucţiunea sub $t0, $s1, $s0, care este gama (-ele) valorilor din $s1 pentru care va rezulta overflow?

26. Precizaţi tipul şi instrucţiunea în limbaj de asamblare MIPS pentru următoarea valoare binară 0000 0010 0001 0000 1000 0000 0010 0000_2

27. Precizaţi tipul şi reprezentarea hexazecimală a următoarei instrucţiuni: sw $t1, 32($t2).

28. Precizaţi tipul, instrucţiunea în limbaj de asamblare, şi reprezentarea binară a instrucţiunii descrisă de următoarele câmpuri MIPS: op=0, rs=3, rt=2, rd=3, shamt=0, funct=34.

29. Precizaţi tipul, instrucţiunea în limbaj de asamblare, şi reprezentarea binară a instrucţiunii descrisă de următoarele câmpuri MIPS: op=0x23, rs=1, rt=2, const=0x4.

30. Presupuneţi că am dori să expandăm fişierul de regiştrii MIPS la 128 de regiştrii, precum şi setul de instrucţiuni pentru a se permite de patru ori mai multe instrucţiuni.

a. Cum va afecta aceasta mărimea fiecărui câmp de biţi pentru instrucţiunile de tip R?
b. Cum va afecta aceasta mărimea fiecărui câmp de biţi pentru instrucţiunile de tip I?
c. Cum ar putea fiecare din cele două schimbări propuse să reducă mărimea unui program scris în limbaj de asamblare MIPS?

31. Presupuneţi următorul conţinut pentru regiştrii: $t0 = 0xAAAAAAAA, $t1 = 0x12345678

a. Pentru valorile de mai sus din regiştrii care este valoarea din registrul $t2 pentru următoarea secvenţă de instrucţiuni?
```
sll $t2, $t0, 44
or $t2, $t2, $t1
```
b. Pentru valorile de mai sus din regiştrii care este valoarea din registrul $t2 pentru următoarea secvenţă de instrucţiuni?
```
sll $t2, $t0, 4
andi $t2, $t2, –1
```
c. Pentru valorile de mai sus din regiştrii care este valoarea din registrul $t2 pentru următoarea secvenţă de instrucţiuni?
```
srl $t2, $t0, 3
andi $t2, $t2, 0xFFEF
```

32. Găsiţi cea mai scurtă secvenţă de instrucţiuni MIPS care extrage biţii de la 16 la 11 din registrul $t0 şi utilizează valorile acestor biţi pentru a înlocui biţii de la 31 la 26 din registrul $t1 fără a schimba ceilalţi 26 de biţi din registrul $t1.

33. Furnizaţi un set minimal de instrucţiuni MIPS care poate fi utilizat pentru a implementa următoarea pseudoinstrucţiune: not $t1, $t2 // bit-wise invert.

34. Pentru următoarea secvenţă de linii scrise în C, scrieţi o secvenţă minimală de instrucţiuni MIPS în limbaj de asamblare care realizează operaţii identice. Presupuneți că $t1 = A, $t2 = B, şi $s1 este adresa de bază pentru C. A = C[0] << 4.

35. Presupuneţi că $t0 memorează valoarea 0x00101000. Care este valoarea lui $t2 după execuţia următoarelor instrucţiuni?
```
slt $t2, $0, $t0
bne $t2, $0, ELSE
j DONE
ELSE: addi $t2, $t2, 2
DONE:
```

36. Presupuneţi că program counter (PC) este setat cu valoarea 0x2000 0000. Este posibil să utilizaţi instrucţiunea MIPS în limbaj de asamblare jump (j) pentru a seta PC-ul cu adresa 0x4000 0000? Estet posibil ca să utilizaţi instrucţiunea MIPS în limbaj de asamblare branch-on-equal (beq) pentru a seta PC-ul cu aceeaşi adresă?

37. Următoarea instrucţiune nu este inclusă în setul de instrucţiuni MIPS:
```
rpt $t2, loop     # if(R[rs]>0) R[rs]=R[rs]–1, PC=PC+4+BranchAddr
```

a. Dacă această instrucţiune ar trebui implementată în setul de instrucţiuni MIPS care ar fi formatul cel mai corespunzător?
b. Care este cea mai scurtă secvenţă de instrucţiuni MIPS care ar realiza aceleaşi operaţii?

38. Fie următoarea buclă MIPS:

```
LOOP:  slt $t2, $0, $t1
       beq $t2, $0, DONE
       subi $t1, $t1, 1
       addi $s2, $s2, 2
       j LOOP
DONE:
```

a. Presupuneţi că registrul $t1 este iniţializat cu valoarea 10. Care este valoarea din $s2 presupunând că $s2 are valoarea iniţială zero?
b. Pentru fiecare buclă de mai sus, scrieţi codul echivalent al unei rutine C. Presupuneţi că regiştrii $s1, $s2, $t1, şi $t2 conţin întregii A, B, i, şi respectiv temp.
c. Presupuneţi, pentru buclele de mai sus scrise în limbaj de asamblare MIPS2, că registrul $t1 este iniţializat cu valoarea N. Câte instrucţiuni MIPS sunt executate?

39. Translataţi următorul cod C în cod în limbaj de asamblare MIPS. Utilizaţi un număr minim de instrucţiuni. Presupuneţi că valorile a, b, i, şi j sun în regiştrii $s0, $s1, $t0, li respectiv $t1.De asemenea presupuneţi că registrul $s2 păstrează adresa de bază pentru aria D.

```
for(i=0; i<a; i++)
 for(j=0; j<b; j++)
  D[4*j] = i + j;
```

40. Câte instrucţiuni MIPS sunt necesare pentru a implementa codul C din exerciţiul anterior (39)? Dacă variabilele a şi b sunt iniţializate cu 10 şi 1 şi toate elementele din D sunt iniţial 0, care este numărul total de instrucţiuni MIPS care se execută pentru a completa bucla?

41. Translataţi următoarea buclă în C. Presupuneţi că întregul i de la nivelul C este păstrat în registrul $t1, $s2 păstrează întregul de la nivelul C denumit result, iar $s0 păstrează adresa de bază al întregului.

```
       addi $t1, $0, 0
LOOP:  lw $s1, 0($s0)
       add $s2, $s2, $s1
       addi $s0, $s0, 4
       addi $t1, $t1, 1
       slti $t2, $t1, 100
       bne $t2, $s0, LOOP
```

42. Rescrieţi bucle din exerciţiul anterior reducând numărul de instrucţiuni MIPS executate.

43. Implementaţi următorul cod scris în C în cod scris în limbaj de asamblare MIPS. Care este numărul total de instrucţiuni MIPS necesare pentru a executa funcţia?

```
int fib(int n){
    if (n==0)
       return 0;
    else if (n == 1)
       return 1;
    else
       return fib(n-1) + fib(n-2);
```

44. Funcţiile pot fi adesea scrise de compilatoare “in-line.” O funcţie in-line este atunci când corpul funcţiei este copiat în spaţiul programului, permiţând eliminarea supracontrolului dat de apelul funcţiei. Implementaţi o versiune “in-line” version of the the C code in the table in MIPS assembly. Care este reducerea instrucţiunilor în asamblare MIPS necesare pentru a realiza funcţia? Presupuneţi că variabila C n este iniţializată cu 5.

45. Pentru fiecare apel de funcţie, prezentaţi conţinutul stivei după ce este realizat apelul de funcţie. Presupuneţi că poiterul de stivă are adresa originală 0x7ffffffc, şi urmează convenţia specificată în figura 2.11.

Preserved	Not preserved
Saved registers: $s0–$s7	Temporary registers: $t0–$t9
Stack pointer register: $sp	Argument registers: $a0–$a3
Return address register: $ra	Return value registers: $v0–$v1
Stack above the stack pointer	Stack below the stack pointer

46. Translataţi funcţia f în cod scris în limbaj de asamblare MIPS. Dacă utilizaţi regiştrii de la $t0 la $t7, utilizaţi prima dată regiştrii cu numerele mici. Presupuneţi că declaraţia funcţiei pentru func este “int f(int a, int b);”. Codul funcţiei f: int f(int a, int b, int c, int d){ return func(func(a,b),c+d);}

47. Se poate utiliza optimizarea tail-call pentru această funcţie? Dacă nu, explicaţi de ce. Dacă da, care este diferenţa privind numărul de instrucţiuni executate în f cu şi fără optimizare?

48. Chiar înainte ca fucţia f (două exerciţii mai sus -46) să revină, ce ştim despre regiştrii $t5, $s3, $ra, şi $sp? Amintiţi-vă faptul că se ştie cum arată toată funcţia f, dar pentru func ştim doar declaraţia sa.

49. Scrieţi un program în limbajul de asamblare MIPS pentru a converti un şir ASCII ce reprezintă întregi zecimali pozitivi sau negativi la un întreg. Programul trebuie să se aştepte că registrul $a0 conţine adresa unui şir terminat cu 0 şi care conţine combinaţii de digiţi de la 0 la 9. Programul vostru trebuie să calculeze vloarea întreagă echivalentă asctui şir de digiţi, după care să plaseze numărul în registrul $v0. Dacă apare un caracter non-digit oriunde în şir programul se va opri şi va returna valoarea -1 în registrul $v0. De exemplu dacă registrul $a0 pointează o secvenţă de trei octeţi 50_{10}, 52_{10}n, 0_{10} (şirul terminat cu zero este “24”), Când programul se va opri, registrul $v0 trebuie să conţină valoarea 24_{10}.

50. Pentru următorul cod:
    lbu $t0, 0($t1)
    sw $t0, 0($t2)
    Presupuneţi că registrul $t1 conţine adresa 0x1000 0000, iar registrul $t2 conţine adresa 0x1000 0010. Amintim că arhitectura MIPS utilizează adresarea big-endian. Presupuneţi că data (în hexazecimal) la adresa 0x1000 0000 este: 0x11223344. Care este valoarea memorată la adresa pointată de registrul $t2?

51. Scrieţi cod în limbaj de asamblare MIPS care creează constanta 0010 0000 0000 0001 0100 1001 0010 0100_2 pe 32 de biţi şi memoraţi această valoare în registrul $t1.

52. Dacă valoarea curentă a PC-ului este 0x00000000, se poate utiliza o singură instrucţiune de salt pentru a prelua în PC adresa din exerciţiul pecedent? Exercise 2.39 (51)?

53. Dacă valoarea curentă a PC-ului este 0x00000600, se poate utiliza o singură instrucţiune branch pentru a prelua în PC adresa din exerciţiul - Exercise 2.39 (51)?

54. Dacă valoarea curentă a PC-ului este 0x1FFFf000, se poate utiliza o singură instrucţiune branch pentru a prelua în PC adresa din exerciţiul - Exercise 2.39 (51)?

55. Scrieţi cod în limbaj de asamblare MIPS pentru a implementa următorul cod C:
    lock(lk);
    shvar=max(shvar,x);
    unlock(lk);

    Presupuneţi că adresa variabilei lk este în $a0, adresa variabilei shvar este în $a1, şi valoarea variabilei x este în $a2. Secţiunea voastră critică nu poate conţine nici un apel de funcţie Utilizaţi instrucţiunile ll/sc pentru a implementa operaţia lock(), iar operaţia unlock() este simplu o instrucţiune ordinară de memorare.

56. Repetă exerciţiul anterior (Exercise 2.43-55), dar de data aceast utilizează ll/sc pentru a realiza o actualizare atomică direct a variabilei shvar, fără a utiliza lock() şi unlock(). Amintim cî în această problemă nu există variabila lk

57. Utilizând codul din exerciţiul (Exercise 2.43 – 55) ca un exemplu, explicaţi ce se întâmplă când două procesoare încep să execute această secţiune critică în acelaşi timp, presupunând că fiecare procesor execută exact o instrucţiune pe ciclu.

58. Presupuneţi că pentru un procesor dat CPI pentru instrucţiunile aritmetice este 1, CPI pentru instrucţiunile load/store este 10, şi CPI pentru instrucţiunile branch este 3. Presupuneţi că programul are următoarele instrucţiuni executate: 500 milioane de instrucţiuni aritmetice, 300 de milioane de instrucţiuni load/store, 100 milioane de instrucţiuni branch.

a. Presupuneți că s-au adăugat la setul de instrucţiuni, instrucţiuni aritmetice noi mult mai puternice. Pe medie, prin utilizarea acestor instrucţiuni aritmetice mai puternice, putem reduce cu 25% numărul de instrucţiuni aritmetice executate de program, iar costul creşterii ciclului de ceas este de numai 10%. Este aceasta o bună alegere de proiectare? De ce?
b. Presupuneți că am găsit o cale pentru a dubla performanţele instrucţiunilor aritmetice. Care este creşterea generală de viteză la nivelul procesorului? Care este creşterea generală de viteză la nivelul procesorului dacă găsim o cale de a mări performanţele instrucţiunilor aritmetice de 10 ori?

59. Presupuneți că pentru un program dat 70% din instrucţiunile executate sunt aritmetice, 10% sunt load/store, şi 20% sunt branch.

a. Fiind dat amestecul de instrucţiuni şi presupunând că instrucţiunile aritmetice necesită 2 cicli, o instrucțiune load/store necesita 6 cicli, şi o instrucțiune branch necesită 3 cicli, găsiți CPI-ul mediu.
b. Pentru o îmbunătățire în performanță de 25%, cât de mulți cicli, în medie, poate necesita o instrucțiune aritmetică dacă instrucțiunile load/store şi branch nu sunt îmbunătăţite?
c. Pentru o îmbunătățire în performanță de 50%, cât de mulți cicli, în medie, poate necesita o instrucțiune aritmetică dacă instrucțiunile load/store şi branch nu sunt îmbunătăţite?

Probleme propuse pentru examenul de la disciplina Structura şi Organizarea Calculatoarelor

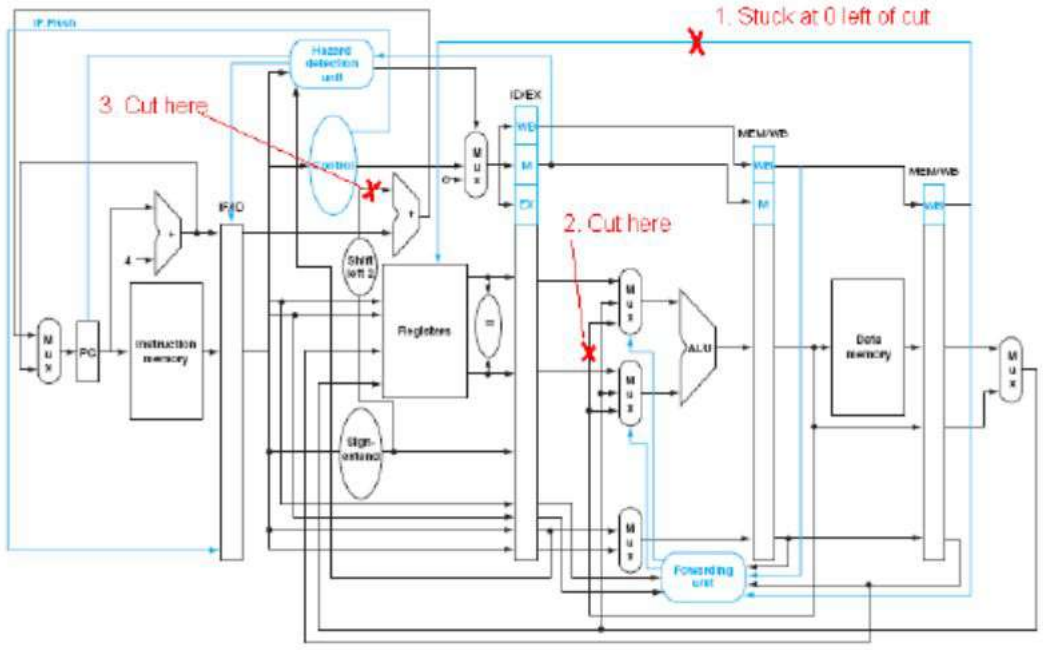
== MOODLE TYPE ==

Prof. dr. ing. Vasile Gheorghiță GĂITAN

Ş. l. dr. ing. Ionel ZAGAN

Cap. 3 Procesorul

1. Pentru calea de date MIPS din figură, unele linii sunt marcate cu X. pentru fiecare:
   a. Descrieţi în cuvinte consecinţele negative ale tăierii acestora relativ la procesorul care lucrează fără modificări.
   b. Furnizaţi un fragment de cod care va eşua.
   c. Furnizaţi un fragment de cod care totuşi va funcţiona.



2. Consideraţi următorul cod în limbajul de asamblare:

I0: ADD R4 = R1 + R0;
I1: SUB R9 = R3 - R4;
I2: ADD R4 = R5 + R6;
I3: LDW R2 = MEM[R3 + 100];
I4: LDW R2 = MEM[R2 + 0];
I5: STW MEM[R4 + 100] = R2;
I6: AND R2 = R2 & R1;
I7: BEQ R9 == R1, Target;
I8: AND R9 = R9 & R1;

Luaţi în considerare o pipeline cu avansare (forwarding), detecţia hazardului şi un slot de întârziere pentru ramificaţii. Piprline-ul este unul tipic MIPS cu 5 etaje IF, ID, EX, MEM, WB. Pentru codul anterior, completaţi diagrama care urmează (instrucţiunile în stânga, ciclii în partea de sus). Înseraţi caracterele IF, ID, EX, MEM, WB pentru fiecare instrucţiune. Presupuneţi că există două niveluri de bypassing, conform cărora a doua jumătate a etajului de decodificare realizează o citire a regiştrilor sursă, iar prima jumătate a etajului write-back scrie în fişierul de regiştrii. Etichetaţi în tabel cu X toate încetinirile (stalls). Etichetaţi toate avansările (forwarding) pe care unitatea de avansare le detectează (săgeţi între etajele care produc datele şi etajele care recepţionează datele).Care este timpul final de execuţie a codului?

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17
I0																		
I1																		
I2																		
I3																		
I4																		
I5																		
I6																		
I7																		
I8																		

3. Hazardul structural, de date şi de control în mod tipic necesită ca pipeline-ul procesorului să încetinească (stall). În continuare sunt listate o serie de tehnici de optimizare implementate în compilator sau într-un pipeline al unui procesor proiectat pentru a reduce sau a elimina încetinirile (stalls) datorate acestor hazarduri. Pentru fiecare dintre următoarele tehnici de optimizare, precizaţi care tip de hazard este adresat şi cum este adresat. Unele tehnici de optimizare pot adresa mai mult decât un singur tip de hazard, ca urmare fiţi sigur ca aţi inclus explicaţii pentru toate hazardurile adresate.
    a) Predicţia salturilor (Branch prediction)
    b) Planificarea instrucţiunilor (Instruction scheduling)
    c) Sloturi de întârziere (delay slots)
    d) Creşterea disponibilităţii unor unităţi funcţionale (increasing availability of functional units (ALUs, adders, etc))
    e) Caches

4. Predicţia salturilor. Consideraţi următoarea secvenţă a rezultatelor pentru o singură ramificaţie statică. T înseamnă că ramificaţia este efectuată. N înseamnă că ramificaţia nu este efectuată. Pentru această întrebare, presupuneţi că este singura ramificaţie din program.
   TTTNTNTTTNTNTTTNTN
    a) Presupuneţi că încercăm să prezicem această secvenţă cu un BHT utilizând un numărător cu un singur bit. Numărătorul din BHT este

iniţializat cu starea N. Care dintre salturile din această secvenţă va prezice greşit?

Predictor state before prediction	Branch outcome	Misprediction?
N	T	
	T	
	T	
	N	
	T	
	N	
	T	
	T	
	T	
	N	
	T	
	N	
	T	
	T	
	T	
	N	
	T	
	N	

b) Acum presupuneţi un predictor cu două niveluri care utilizează un bit pentru istoria ramificaţiei – un BHR cu un bit. Deşi există o singură ramificaţie în program, nu contează cum este BHR concatenat cu PC-ul ramificaţiei ca index în BHT. Presupuneţi că BHT utilizează un numărător de un bit şi ca urmare, din nou, toate intrările sunt iniţializate cu N. Care dintre salturile din secvenţă vor prezice greşit?

Predictor state before prediction	BHR	Branch outcome	Misprediction?
N	N	T	
		T	
		T	
		N	
		T	
		N	
		T	
		T	
		T	
		N	
		T	
		N	
		T	
		T	
		T	
		N	
		T	
		N	

c) Care este adresa de revenire de pe stivă? Când este actualizată adresa de pe stivă?

5. Pipeline-ul clasic are 5 etaje IF, ID, EX, MEM, WB, Acest pipeline este proiectat în mod specific pentru a executa setul de instrucţiuni MIPS. MIPS este o arhitectură load-store care realizează o operaţie cu memoria per instrucţiune, ca urmare este suficient un singur etaj MEM în pipeline. De asemenea, modul de adresare cel mai comun este modul de adresare prin registru cu deplasament. Etajul EX este plasat înainte de MEM pentru a permite ca EX să fie utilizat pentru calculul adresei. În această întrebare se va lua în considerare o variaţie în setul de instrucţiuni MIPS şi interacţiunea acestei variaţii în structura pipeline.
   Variaţia care se ia în calcul implică comutarea etajelor MEM şi EX, creându-se o nouă pipeline IF, ID, MEM, EX, WB. Această schimbare are două efecte asupra setului de instrucţiuni. Primul, ne previne asupra utilizării adresării pe bază de registru cu deplasament (nu există un EX înainte de MEM care să realizeze adunarea cu semn). Totuşi, putem utiliza instrucţiuni cu un operand de intrare în memorie, cum ar fi instrucţiuni cu adresare prin registru. De exemplu multf_m f0,f2,(r2) înmulţeşte conţinutul registrului f2 cu valoarea locaţiei de meorie pointată de r2 şi pune rezultatul în f0.

   a) Eliminarea modului de adresare prin registru cu deplasament este o mare pierdere, pentru că este modul cel mai utilizat în MIPS. De ce este aşa de frecvent? Enumeraţi două construcţii populare care utilizează modul de adresare prin registru cu deplasament (utilizaţi adresarea cu deplasament cu deplasament diferit de zero).
   b) Care este diferenţa între dependenţe şi un hazard?
   c) În această întrebare se lucrează cu bucla SAXPY.

```
do I = 0,N
Z[I] = A*X[I] + Y[I]
```

   Aici este noul cod în asamblare.

```
0: slli     r2,r1,#3 // I is in r1
1: addi     r3,r2,#X
2: multf_m  f2,f0,(r3) // A is in f0
3: addi     r4,r2,#Y
4: addf_m   f4,f2,(r4)
5: addi     r4,r2,#Z
6: sf       f4,(r4)
7: addi     r1,r1,#1
8: slei     r6,r1,r5 // N is in r5
9: bnez     r6,#0
```

   Utilizând numerele pentru instrucţiuni etichetaţi datele şi dependenţele de control

   d) Umpleţi diagrama pipeline pentru codul nou al buclei SAXPY. Etichetaţi încetinirea (stall) ca d* pentru hazardul de date şi s* pentru hazardul structural. Care este întârzierea pentru o singură iteraţie? (numărul de cicli între completarea a două #0 instrucţiuni succesive). Pentru această întrebare, presupuneţi că adunarea FP ia 2 cicli, înmulţirea FP ia trei cicli şi că toate celelalte operaţii iau un singur ciclu. Unităţile funcţionale nu sunt pipeline. Sumatorul FP, multiplicatorul FP şi ALU pentru întregi sunt unităţi funcţionale separate, astfel încât nu există hazard structural între ele. Fişierul

de regiştrii este scris de etajul WB în prima jumătate a ciclului de ceas şi este citit de etajul ID în a doua jumătate a ciclului de ceas. În plus, procesorul are o unitate de avansare (forwarding) completă. Procesorul încetineşte (stalls) ramificaţiile până când rezultatul este disponibil la sfârşitul etajului EX. Procesorul nu are resurse pentru a menţine o stare precisă.

instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0: slli r2,r1,#3	F	D	M	X	W																			
1: addi r3,r2,#X		F	D	M	X	W																		
2: mulf_m f2,f0,(r3)			F	D		M	X	X	X	W														
3: addi r4,r2,#Y				F		D	M	X	W															
4: addf_m f4,f2,(r4)						F	D		M	X	X	W												
5: addi r4,r2,#Z							F		D	M	X		W											
6: sf f4,(r4)									F	D		M	X	W										
7: addi r1,r1,#1										F		D	M	X	W									
8: slei r6,r1,r5												F	D	M	X	W								
9: bnez r6, #0													F	D	M	X	W							
0: slli r2,r1,#3																	F	D	M	X	W			

e) În pipeline-ul pentru MIPS menţionat în text, care este motivul pentru a forţa ca operaţiile non-memorie să treacă prin etaju8l MEM în loc să meargă direct în etajul WB?
f) Pe lângă pierderea adresării prin registru cu deplasament şi a instrucţiunilor necesare pentru a suplini acest neajuns, care sunt alte două dezavantaje ale acestui tip de pipeline?
g) Reduceţi încetinirea prin planificarea pipeline-ului cu o singură iteraţie a buclei. Prezentaţi codul rezultat şi umpleţi diagrama pipeline. Pentru un răspuns corect nu trebuie să prezentaţi o planificare optimală.

instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0: slli r2,r1,#3	F	D	M	X	W																			
1: addi r3,r2,#X		F	D	M	X	W																		
2: mulf_m f2,f0,(r3)			F	D		M	X	X	X	W														
3: addi r4,r2,#Y				F		D	M	X	W															
4: addf_m f4,f2,(r4)						F	D		M	X	X	W												
5: addi r4,r2,#Z							F		D	M	X		W											
6: sf f4,(r4)									F	D		M	X	W										
7: addi r1,r1,#1										F		D	M	X	W									
8: slei r6,r1,r5												F	D	M	X	W								
9: bnez r6, #0													F	D	M	X	W							
0: slli r2,r1,#3																	F	D	M	X	W			

6. O întrebare cu două părţi

a) Detecţia dependenţelor
Această întrebare se referă la modul în care aţi înţeles dependenţele între instrucţiuni, Utilizând codul care urmează, listaţi toate dependenţele de tipul RAW, WAR, WAW. Listaţi dependenţele în tabelul următor (exemplu INST-X la INST-Y) prin scrierea numărului instrucţiunii implicate în dependenţă.

I0: A = B + C;
I1: C = A - B;
I2: D = A + C;
I3: A = B * C * D;
I4: C = F / D;
I5: F = A ^ G;
I6: G = F + D;

RAW Dependence		WAR Dependence		WAW Dependence	
From Instr	To Instr	From Instr	To Instr	From Instr	To Instr

b) Analiza dependenţelor
Fiind date patru instrucţiuni, câte comparaţii unice (între registrul sursă şi destinaţie) sunt necesare pentru a găsi toate dependenţele de tip RAW, WAR şi WAW. Răspundeţi pentru cazul cu patru instrucţiuni, şi apoi derivaţi o ecuaţie generală pentru N instrucţiuni. Presupuneţi că toate instrucţiunile au un registru destinaţie şi doi regiştrii sursă.

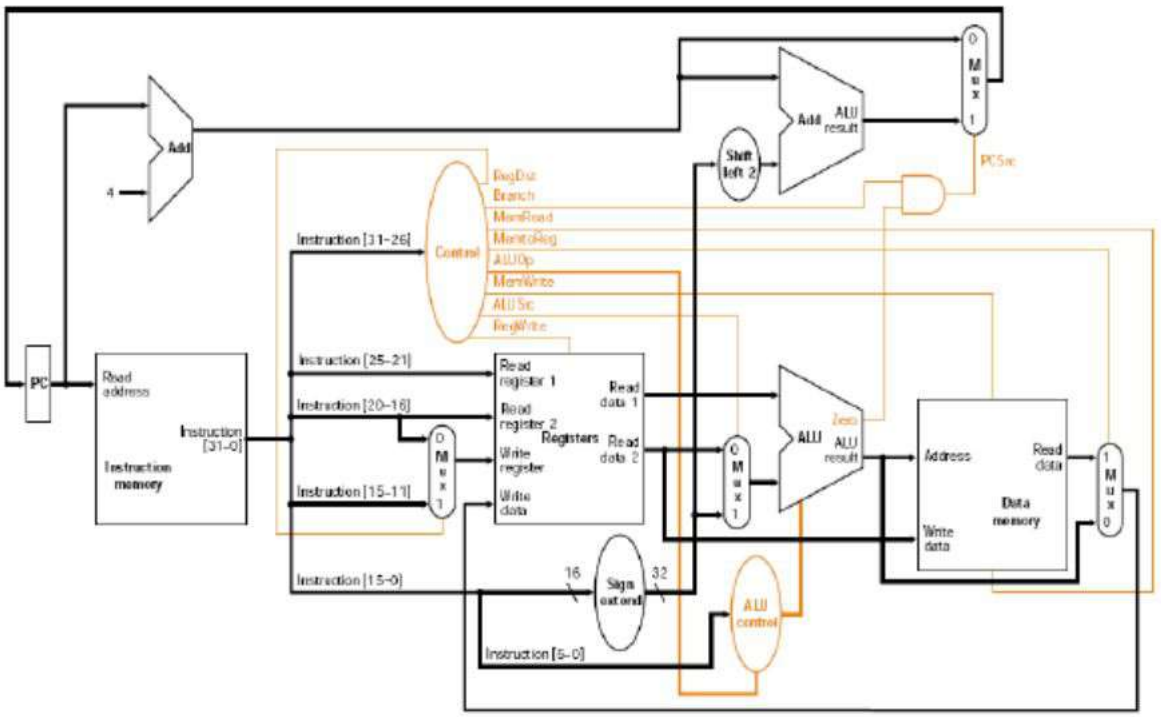
7. Se consideră o implementare cu un singur ciclu a ISA MIPS. Timpul de operare pentru componentele funcţionale majore sunt prezentate în tabelul următor:

Component	Latency
ALU	10 ns
Adder	8 ns
ALU Control Unit	2 ns
Shifter	3 ns
Control Unit/ROM	4 ns
Sign/zero extender	3 ns
2-1 MUX	2 ns
Memory (read/write) (instruction or data)	15 ns
PC Register (read action)	1 ns
PC Register (write action)	1 ns
Register file (read action)	7 ns
Register file (write action)	5 ns
Logic (1 or more levels of gates)	1 ns

În figura următoare este o copie a proiectului pentru o cale de date cu un singur ciclu. În această implementare ciclul de ceas este determinat de calea cea mai lungă posibilă. Căile critice pentru diferite tipuri de instrucţiuni care trebuie luate în considerare sunt: Format-R, Load-word, şi store-word. Toate instrucţiunile au aceeaşi paşi pentru extragere şi decodificare. Transferul de bază între regiştrii instrucţiunilor este:

Fetch/Decode: Instruction <- IMEM[PC];
R-type: R[rd] <- R[rs] op R[rt]; PC <- PC + 4;
load: R[rt] <- DMEM[R[rs] + signext(offset)]; PC <- PC +4;

store: DMEM[R[rs] + signext(offset)] <- R[Rt]; PC <- PC +4;



a) În tabelul care urmează, indicaţi componentele care determină calea critică pentru respectiva instrucţiune, în cazul în care apare calea critică. Dacă este utilizată o componentă, dar nu este parte a căii critice a instrucţiunii (lucrează în paralel cu altă componentă), nu trebuie pusă în tabel. Fişierul de regiştrii este utilizat pentru citire sau scriere; poate să apară de două ori pentru aceeaşi instrucţiune. Toate instrucţiunile încep prin a citi registrul PC cu o întârziere de 2ns.

Instruction Type	Hardware Elements Used By Instruction											
R-Format												
Load												
Store												

b) Plasaţi întârzierile componentelor pe care aţi decis că intră în calea critică a fiecărei instrucţiuni în tabelul care urmează. Calculaţi suma întârzierilor fiecărei componente pentru fiecare instrucţiune.

Instruction Type	Hardware Latencies For Respective Elements											Total
R-Format	2 ns											
Load	2 ns											
Store	2 ns											

c) Utilizaţi coloana cu întârzierea totală pentru a obţine următoarele informaţii privind calea critică:
   - Fiind dată calea de date, care instrucţiune determină calea critică generală?

- Care va fi ciclul de ceas rezultat pe baza analizei căii critice?
- La ce frecvență va merge procesorul?

8. Figura următoare ilustrează trei tipuri de predictori,
   - Ultimul luat - predicţia este luată pe 1
   - Sus-jos (Up-Down – numărător cu saturaţie) predicţia este luată pentru 11 şi 10
   - Automatul A3 – este luatî pentru 11 şi 10
   - Umpleţi tabelul care urmează pentru fiecare predictor de ramificaţie. Şablonul de execuţie pentru ramificaţia este NTNNTTTN.

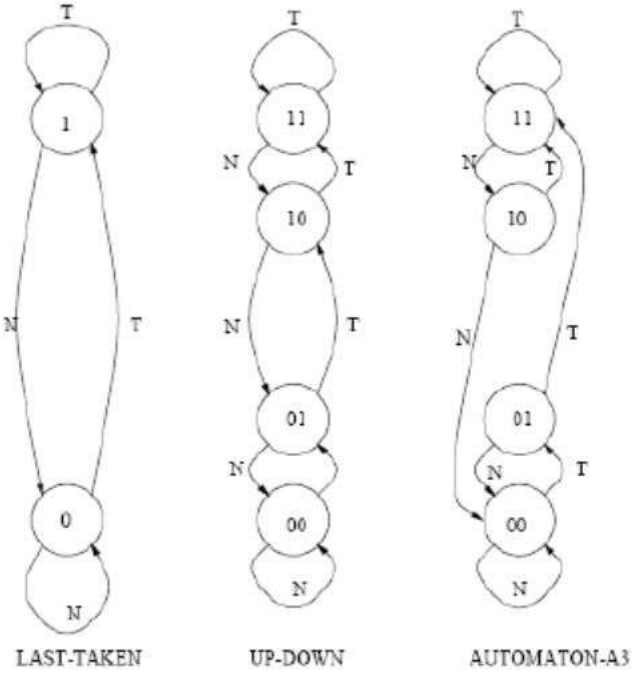


Table 1: Table for last-taken branch predictor

Execution Time	Branch Execution	State Before	Prediction	Correct or Incorrect	State After
0	N	0			
1	T				
2	N				
3	N				
4	T				
5	T				
6	T				
7	N				

Table 2: Table for saturating counter (up-down) branch predictor.

Execution Time	Branch Execution	State Before	Prediction	Correct or Incorrect	State After
0	N	01			
1	T				
2	N				
3	N				
4	T				
5	T				
6	T				
7	N				

Table 3: Table for Automata-A3 branch predictor.

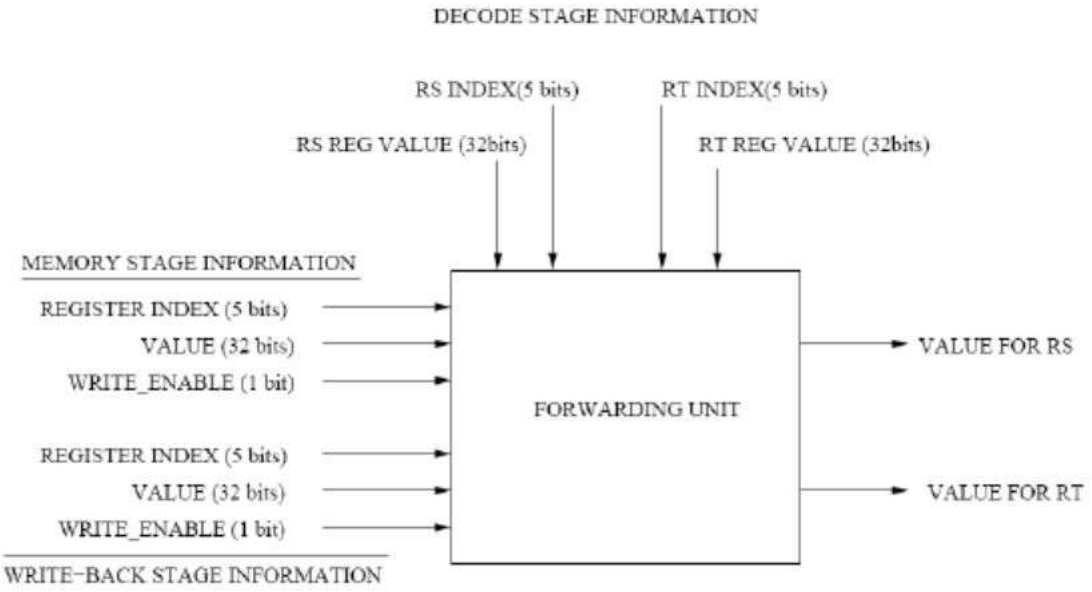
Execution Time	Branch Execution	State Before	Prediction	Correct or Incorrect	State After
0	N	01			
1	T				
2	N				
3	N				
4	T				
5	T				
6	T				
7	N				

Calculate the prediction rates of the three branch predictors:

Predictor	Prediction Accuracy
Last-taken	
Up-Down	
Automata-A3	

9. Pipeline este utilizat deoarece îmbunătăţeşte fluxul instrucţiunilor. Creştere nivelului de pipeline descreşte cantitatea de muncă la fiecare etaj pipeline, permiţând ca în procesor să fie executate mai multe instrucţiuni în acelaşi timp, iar o anumită instrucţiune să fie complet executată la o viteză mai mare. Totuşi, fluxul nu se va îmbunătăţii dacă pipeline-ul creşte nedefinit. Daţi două motive pentru această observaţie.

10. Proiectarea logicii de avansare. Pentru această problemă trebuie să proiectaţi unitatea de avansare pentru un procesor pipeline cu 5 etaje. Unitatea de avansare returnează valoare care trebuie avansată pentru instrucţiunea curentă. Există trei locuri de unde poate să vină valoarea pentru regiştrii RS şi RT: etalul de decodificare (fişierul de regiştrii), etajul de memorie şi etajul write-back (scrie înapoi).



Informaţia de la etajele write-back şi memorie constă din: _INDEX – precizează indexul registrului în lucru care trebuie scris, _VALUE – valoarea care trebuie scrisă, _ENABLE – dacă este sau nu instrucţiunea din etaj una care scrie.
Etajul de decodificare memorează indexul regiştrilor (pentru RS şi RT) şi valorile corespunzătoare din fişierul de regiştrii.
În general pot exista trei valori, una din care unitatea de avansare poate alege una dintre cererile pentru regiştrii RS şi RT. Etajul de memorie are valoarea MEM, etajul write-back are valoarea WB, şi fişierul de regiştrii are valorile RS-REG sau RT-REG.
Utilizând tabelul următor care conţine informaţii despre toate etajele instrucţiunilor, indicaţi care valoare trebuie avansată (forwarding) pentru instrucţiunea curentă: MEM, WB,RS-REG sau RT-REG. Fiecare linie reprezintî o evaluare a unităţii de avansare; nu există conexiuni între liniile de evaluare din tabel. Nu trebuie să vă îngrijoraţi asupra detecţiei hazardului, ci numai de bypass-area valorii.

	Mem Stage		Write-Back Stage		Register Stage		RS Value	RT Value
Evaluation	Index	Write	Index	Write	RS-Index	RT-Index		
0	5	1	23	0	6	7		
1	7	0	16	1	16	8		
2	10	1	10	1	11	10		
3	17	0	12	1	12	12		
4	19	0	19	0	19	25		

11. Consideraţi un procesor MIPS cu 5 etaje pipeline cu durata unui ciclu de 10ns, Consideraţi că executaţi un program unde o fracţie, f, din toate instrucţiunile urmează imediat un load de care devin dependente.
    a) Care este timpul total de execuţie pentru N instrucţiuni, din f, cu unitatea de avansare validată?
    b) Consideraţi un scenariu pentru care etajul MEM, împreună cu registrul ei de pipeline, necesită 12 ns. Există două opţiuni: se adaugă încă un etaj MEM astfel încât să rezulte etajele MEM1 şi MEM2, sau să se mărească durata ciclului maşină la 12 ns astfel încât etajul MEM să se încadreze în timpul noului ciclu şi numărul de etaje pipeline să nu fie afectat. Pentru un program care mixează caracteristicile anterior descrise, când este prima opţiune mai bună decât a doua? Răspunsul trebuie să se bazeze pe valoarea lui f.

c) Procesoarele înglobate (embedded) au două regiuni diferite de memorie – o memorie rapidă apropiată de procesor şi o memorie normală mai înceată. Presupuneţi că într-un procesor cu 6 etaje (cu MEM1 şi MEM2), există o regiune de memorie care este mai rapidă şi pentru care valoarea corectă se obţine la sfârşitul etajului MEM1 în timp ce restul memoriei are nevoie de ambele etaje MEM1 şi MEM2. Din motive de simplitate presupuneţi că există două tipuri de instrucţiuni load şi anume load.fast şi load.slow care indică ce regiune de memorie este utilizată. Cum se schimbă răspunsul la întrebarea precedentă dacă 40% din fracţia f menţionată anterior îşi preia valorile din memoria rapidă?

12. Fie un procesor cu 4 etaje pipeline descris în continuare:
    - IF: extragerea instrucţiunii
    - IDE: decodificarea instrucţiunii, extragerea regiştrilor, evaluarea ALU, instrucţiunea de ramificaţie modifică PC, calculul adresei pentru accesul la memorie,
    - MEM: accesul la memorie pentru instrucţiunile load şi store,
    - WB: scrie rezultatul execuţiei înapoi în fişierul de regiştrii. Scrierea înapoi apare în a doua jumătate a ciclului.

    Presupuneţi metoda de ramificaţie întârziată. Pentru programul următor, presupuneţi că bucla va itera de 15 ori. Presupuneţi că pipeline-ul termină o instrucţiune în fiecare ciclu cu excepţia cazului când saltul este luat sau când are loc o interblocare. O interblocare previne instrucţiunile de a fi executate într-o secvenţă greşită pentru a conserva dependenţele originale de date. Presupuneţi că regiştrii pot bypass-aţi atât de la ieşirea IDE cât şi de la ieşirea MEM. Presupuneţi de asemenea că r2 nu va mai fi folosit după întoarcerea din execuţie.

A		ADD	r3 ← 0
B		LOAD	r1 ← M(0200)
C	Loop:	LOAD	r2 ← M(0208+r1)
D		ADD	r3 ← r2 + r3
E		SUB	r1 ← r1 -4
F		BRANCH! =	PC ← Loop if r1 ! = 0
G		NOP	a delay slot to be filled
H		STORE	M(0204) ← r3
I		RETURN	return from function
J		NOP	a delay slot to be filled

a) Există o interblocare în program? Dacă da, realizaţi reordonarea codului din program şi prezentaţi noul program fără cicli de interblocare.
b) Calculaţi numărul total de ciclii necesari pentru a executa toate instrucţiunile înainte şi după ce s-a eliminat ciclii de interblocare.
c) Filaţi sloturile de întârziere. Descrieţi reordonarea codului şi/sau duplicatele realizate. Prezentaţi acelaşi program după filarea sloturilor de întârziere. Reamintim că RETURN este tot o instrucţiune de ramificare, Utilizaţi un apostrof (‘) pentru na marca noua copie a instrucţiuni duplicate. De exemplu dacă aţi duplicat D, numele copiei este D’.
d) Calculaţi numărul total de cicli necesari pentru a se executa toate instrucţiunile după ce aţi filat sloturile de întârziere.

13. Utilizând o optimizare ILP, dublaţi performanţele următoarei bucle, sau explicaţi de ce nu este posibil acest lucru. Procesorul poate realiza doar un salt per ciclu, dar altfel are resurse infinite.

```
r1 = ... ; r1 is head pointer to a linked list
r3 = 0
LOOP:
r2 = M[r1 + 8]
r3 = r3 + r2
r1 = M[r1]
branch r1 != 0, LOOP
... = r3 ; r3 este utilizat când bucla este completă
```

14. Consideraţi calea de date de dedesubt. Acest procesor nu suportă cod cu sloturi de întârziere pentru ramificaţii (prezice că saltul nu se realizează cu o penalizare de 1 ciclu). Pentru fiecare semnal de control listat în tabelul de mai jos, determinaţi valşoarea acestuia în cicli 3 până la 9, inclusiv. De asemenea, arătaţi instrucţiunile care ocupă fiecare etaj din pipeline în toţi cicli (presupuneţi că linia de validare a scrierii în IF/ID este setată cu inversul semnalului Stall).

Starea iniţială a procesorului este:

PC = 0

Toţi regişstrii pipeline conţin 0

Toţi regişstrii din fişierul de regişstrii conţin 0

Memoria de date conţine 0 în toate locaţiile

Memoria de instrucţiuni conţine:

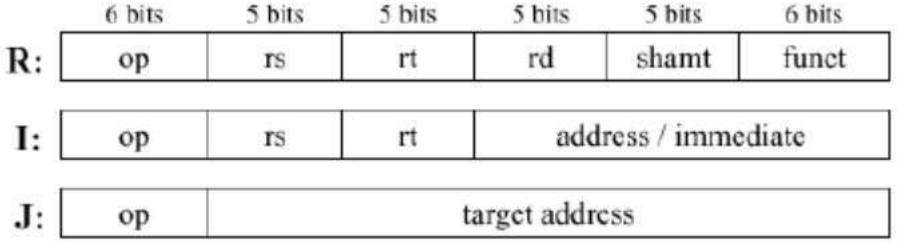
```
00: addiu $3, $zero, 4
04: lw $4, 100($3)
08: addu $2, $4, $3
0C: beq $4, $zero, 0x14
10: addiu $3, $3, 1
14: addu $2, $2, $3
```

Toate celelalte locaţii conţin 0.

Utilizaţi avansarea datelor unde este posibil. Toate intrările multiplexoarelor sunt numerotate vertical de sus în jos, începând cu 0 dacă priviţi la calea de date orientată pe orizontală. De asemenea valorile pentru ALUOp sunt:

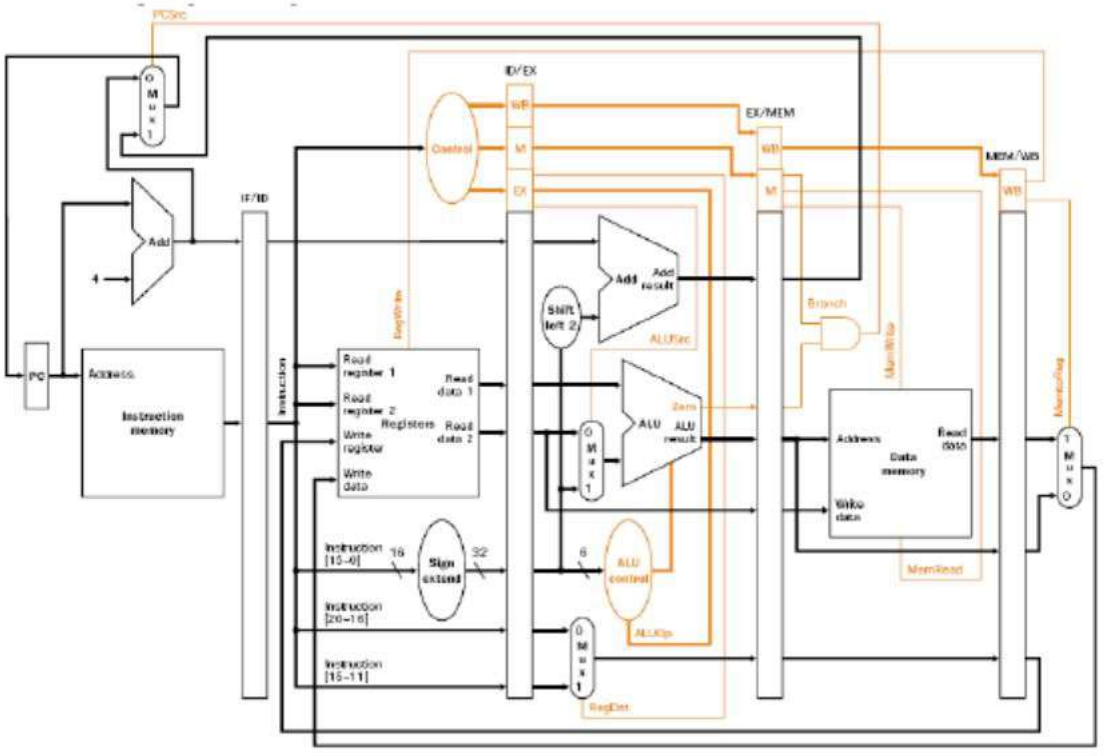
Value	Desired ALU Action
00	Add
01	Subtract
10	Determine by decoding funct field

Instruction formats:



Time	PCWrite	IF.flush	Branch	Stall	ALUSrc	ALUOp	RegDst	ForwardA	ForwardB	MemRead	MemWrite	MemtoReg	RegWrite
	IF		ID		EX					M		WB	
0	00: addiu		--		-					--		-	
	1	0	0	0	0	00	0	00	00	0	0	0	0
1	04: lw		00: addiu		-					--		-	
	1	0	0	0	0	00	0	00	00	0	0	0	0
2	08: addu		04: lw		00: addiu					--		-	
	1	0	0	0	1	00	0	00	00	0	0	0	0
3													
4													
5													
6													
7													
8													
9													

15. Consideraţi următoarea cale de date:



a) Discutaţi funcţionalitatea semnalelor de control RegDst şi ALUSrc.

b) Modificaţi, diagrama pentru a indica schimbările căii de date (şi orice multiplexare adiţională) necesară pentru a furniza bypass-ul din EX îm EX pentru toate hazardurile RAW posibile pentru operaţiile aritmetice. Cum se schimbă ALUSrc când se adaugă bypass-ul?

16. Imaginaţi o instrucţiune a cărei funcţie este de a citi din memorie patru cuvinte adiacente pe 32 de biţi în patru regiştrii arhitecturali pe 32 de biţi specificaţi. Presupuneţi că pipeline-ul cu cicni etaje este filat cu aceste instrucţiuni şi NUMAI cu aceste instrucţiuni, care este numărul minim de citiri şi scrieri care pot fi cerute la porturile fişierului de regiştrii?

17. Pipelining şi bypass. În această întrebare se va explora modul cum bypass-ingul afectează performanţele de execuţie a programului. Pentru început se consideră un MIPS clasic cu 5 etaje. Pentru referinţă luaţi în considerare figura care urmează. Pentru această întrebare, se va utiliza următorul cod pentru a evalua performaţele pipeline-ului.

```
1 add   $t2, $s1, $sp
2 lw    $t1, $t1, 0
3 addi  $t2, $t1, 7
4 add   $t1, $s2, $sp
5 lw    $t1, $t1, 0
6 addi  $t1, $t1, 9
7 sub   $t1, $t1, $t2
```

Care este întârzierea dată de utilizarea lui load (load-use) pentru MIPS-ul standard cu 5 etaje?
Din nou, utilizând pipeline-ul MIPS standard, identificaţi unde valoarea pentru fiecare operand din registru vine din bypass sau din fişierul de regiştrii. Pentru claritate, scrieţi REG sau BYPASS în fiecare celulă.

Instruction	Src Operand 1	Src Operand 2
1		
2		N/A
3		N/A
4		
5		N/A
6		N/A
7		

Câţi cicli va lua programul pentru a se executa pe un pipeline MIPS standard? Presupuneţi, că datorită constrângerilor de circuit, că firele pentru bypass de la etajul de memorie înapoi către etajul de execuţie sunt omise din pipeline. Care întârzierea la utilizarea lui load pentru acest pipeline modificat?
Identificaţi dacă valoarea pentru fiecare operand din registru vine din bypass sau din fişierul de regiştrii pentru pipeline-ul modificat. Pentru claritate scieţi REG sau BYPASS în fiecare celulă.

Instruction	Src Operand 1	Src Operand 2
1		
2		N/A
3		N/A
4		
5		N/A
6		N/A
7		

Cât timp ia execuţia programului pe pipeline-ul modificat?

18. Fie pseudo-codul în asembler:

```
1: mov r1 = 0 ;; iteration count i
2: mov r2 = 0 ;; initialize x
loop:
3: sll r3 = r1, 2 ;; r3 = r1*4
4: lw r4 = A(r3) ;; load A[i]
5: add r2 = r2, r4 ;; x += A[i]
6: add r1 = r1, 1 ;; increment counter
7: bne r1, 1024, loop ;; backwards branch
```

Presupuneţi că aveţi un procesor cu resurse infinite (lărgime infinită, fereastră infinită pentru instrucţiuni, un număr infinit de unităţi funcţionale, resurse infinite de redenumire, un număr infinit de încărcări întârziate/suprapuse), şi care este pe deplin pipeline, astfel încât toate instrucţiunile normale iau un ciclu, dar încărcările care lipsesc iau 10 cicli. Presupuneţi o predicţie perfectă a salturilor.

a) Prezentaţi o planificare dinamică a instrucţiunilor pentru primele două iteraţii ale buclei în tabelul ce urmează, presupunând că lipsa încărcărilor (load) în cache apare iteraţiile pare (r1 = 0,2,…) şi potrivirea încărcărilor (load) în iteraţiile impare (r1 = 1,3,…). Fiţi siguri că aţi indicat iteraţia buclei la care este asociată fiecare instrucţiune. De exemplu, dacă instrucţiunea 5 din iteraţia 7 poate rula în ciclul 3, atunci se poate scrie “5.7” în una din celulele din rândul ciclului 3. DE notat că primul ciclu a fost filat şi celulele din table sunt mai multe decât aveţi nevoie.

Cycle	Instructions					
1	1	2				
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						

19. Predicţie şi predicaţie.
    a) Consideraţi următoarea secvenţă de cod:

```
for (i=0; i < 1000000; i++{
 a= random(100);
 if (a >= 50){
 ….
 }
}
```

    Presupuneţi că random(N) returnează un număr aleatoriu uniform distribuit între 0 şi N – 1 inclusiv. Consideraţi instrucţiunea de ramificaţie asociată cu linia if. Există acolo un predictor de ramificaţie care prezice corect? Explicaţi răspunsul.

    b) Predicarea poate elimina toate ramificaţiile condiţionale de avansare în program. Salturile înapoi în program sunt, în mod tipic, asociate cu buclele şi ca urmare sunt în majoritate luate (se sare la adresa destinaţie). Ca urmare este posibil de a elimina salturile de avansare şi de a prezice static salturile înapoi ca luate şi aceasta va elimina complexitatea predictorilor într-un procesor. Dar, procesorul Itanium care are suport hardware pentru predicaţie totuşi păstrează un predictor complex cu două niveluri pentru ramificaţie. Explicaţi aceasta în acest caz.

20. Ocazional compilatoarele sunt utile. O buclă comună în programele ştiinţifice este aşa numita SAXPY:

```
for (i = 0; i < N; i++)
{
Y[i] = A*X[i] + Y[i];
}
```

    a) Pentru A = 3 şi N=300 convertiţi codul C în asamblare MIPS. Puteţi presupune că X şi Y sunt arii cu întregi pe 32 de niţi. Încercaţi să evitaţi pseudo-codul datorită părţii a doua a acestei întrebări. De asemenea, evitaţi să introduceţi dependenţe false. Utilizaţi vă rog $s0 pentru a păstra valoarea lui i. Puteţi, de asemenea, presupune că $s2 şi $s3 păstrează iniţial adresa de start a lui X, respectiv Y. Sugestie: deoarece A=3=2+1 puteţi realiza uşor înmulţirea fără a utiliza instrucţiunea mult.
    b) Cât de multe instrucţiuni statice (numărul total de instrucţiuni în memorie) are fragmentul de program? Câte instrucţiuni dinamice (numărul total de instrucţiuni executate) are fragmentul de program?
    c) Presupuneţi că dorim să realizăm mai mult de o buclă SAXPY:

```
for (i = 0; i < N; i++)
{
Y[i] = A*X[i] + Y[i];
}
for (i = 0; i < N; i++)
{
Z[i] = A*X[i] + Z[i];
}
```

    Utilizând calculele de la întrebările a şi b, câte instrucţiuni dinamice şi câte statice are acest fragment de program?

d) Există două optimizări care pot fi realizate cu acest cod. Prima buclă este fuzionată, astfel cele două bucle sunt combinate într-o singură buclă:

```
for (i = 0; i < N; i++)
{
Y[i] = A*X[i] + Y[i];
Z[i] = A*X[i] + Z[i];
}
```

Altă optimizare este eliminarea subexpresiilor comune. Astfel putem scoate în afară calculul repetat:

```
for (i = 0; i < N; i++)
{
T = A*X[i];
Y[i] = T + Y[i];
Z[i] = T + Z[i];
}
```

Convertiţi fiecare din aceste fragmente de program în asamblare MIPS. $s4 va conţine adresa de start pentru Z, Utilizaţi $s1 pentru a memora valoarea lui T. Câte instrucţiuni dinamice şi câte statice are fiecare fragment? Cât de mult s-a salvat cu fuziunea buclelor. Cât de mult s-a salvat cu eliminarea sub-expresiei?

e) Adnotaţi codul în asamblare de la punctul d) cu arcuri pentru dependenţele de date (flux, anti, şi ieşire – fără control

21. Consideraţi următoarea instrucţiune:
    Instrucţiune: and Rd,Rs,Rt
    Interpretare: Reg[Rd] = Reg[Rs] AND Reg[Rt]
    a) Care sunt valorile semnalelor de control generate de controlul din figura 4.2 pentru instrucţiunea următoare sw rt,offset(rs)?

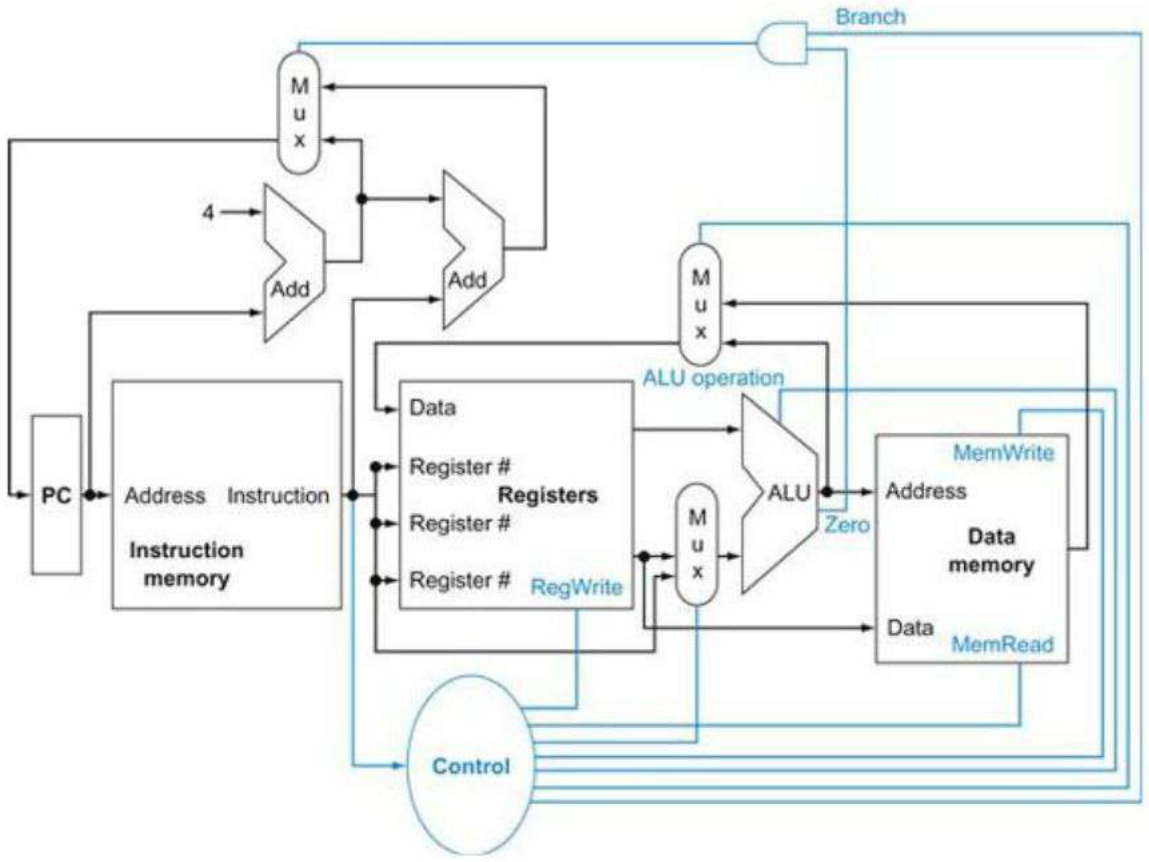


Figura 4.2

b) Ce resurse (blocuri) realizează o funcţie utilă pentru această instrucţiune?
c) Care resurse (blocuri) produc ieşiri dar ieşirile lor nu sunt folosite pentru această instrucţiune. Care resurse nu produc ieşiri pentru această instrucţiune?

22. Implementarea MIPS de bază din figura 4.2 poate să implementeze doar câteva instrucţiuni. Se pot adăuga noi instrucţiuni la o arhitectură a setului de instrucţiuni (ISA), dar decizia dacă se face sau nu această adăugare depinde, printre alte lucruri, de costul şi complexitatea pe care adăugarea propusă o introduce în calea de date şi control. Fie instrucţiunea: LWI Rt, Rd(Rs). Interpretare: Reg[Rt] = Mem[Reg[Rd] + Reg[Rs]].

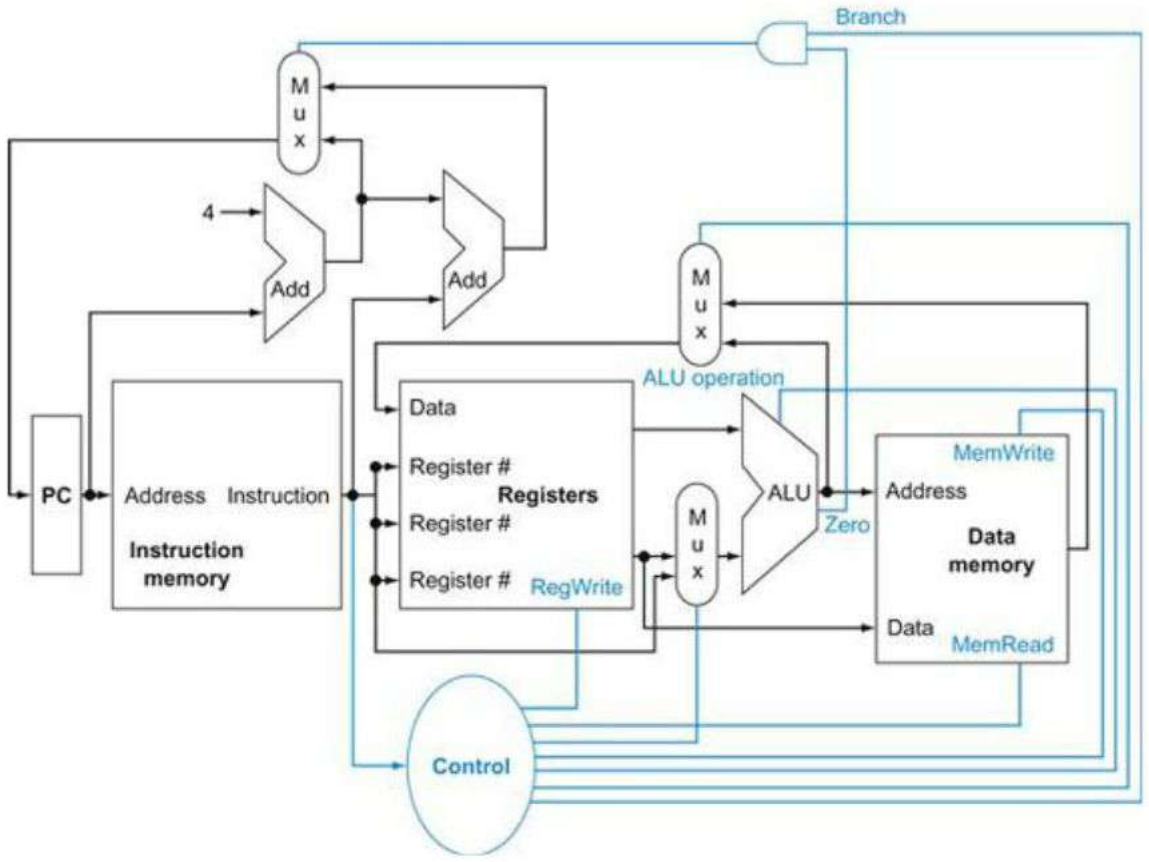


Figura 4.2

a) Care dintre blocurile existente (dacă există unul) poate fi utilizat pentru această instrucţiune?
b) Ce blocuri funcţionale noi (dacă există unul) sunt necesare pentru această instrucţiune?
c) Ce semnale noi (dacă există unul) sunt necesare pentru unitatea de control pentru a suporta această instrucţiune?

23. Atunci când un proiectant de procesoare consideră o posibilă îmbunătăţire a căii de date, decizia depinde în mod uzual de raportul cost/performanţă. Presupunem că se începe cu calea de date din figura 4.2 unde I-Mem, Add, Mux, ALU, Regs, D-Mem şi blocurile de control introduc întârzieri de 400 ps, 100ps, 30ps, 120ps, 200ps, 350ps şi respectiv 100ps şi un cost de 1000, 30, 10, 100, 200, 2000 şi respectiv 500. Consideraţi adăugarea unui multiplicator la ALU. Această adăugare va aduce o întârziere de 300 ps la ALU şi un cost de 600 tot la ALU. Rezultatul va fi 5% mai puţine instrucţiuni executate deoarece nu se va mai emula instrucţiune MUL de înmulţire.

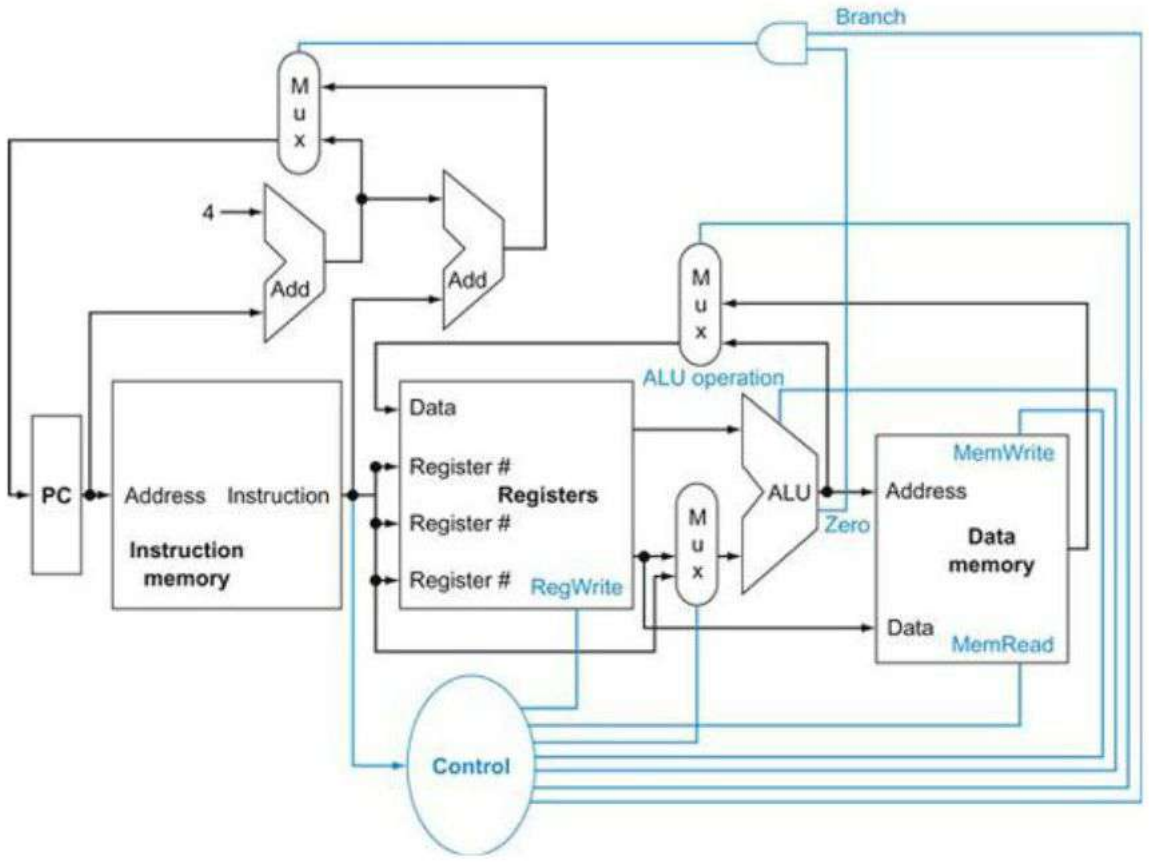


Figura 4.2

a) Care este durata unui ciclu de ceas cu şi fără această îmbunătăţire?
b) Care este creşterea de viteză prin adăugarea acestei îmbunătăţiri?
c) Comparaţi raportul cost/performanţă cu şi fără această îmbunătăţire.

24. Să presupunem că blocurile logice necesare pentru implementarea unei căi de date au următoarele întârzieri:

I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-Extend	Shift-Left-2
200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps

a) Dacă singurul lucru pe care trebuie să-l facem în procesor este extragerea de instrucţiuni consecutive (figura 4.6), care va fi durata unui ciclu?

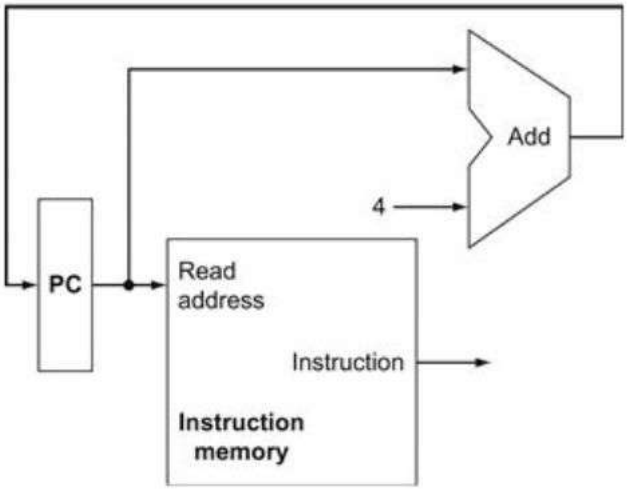


Figura 4.6

b) Consideraţi o cale de date similară cu aceea din figura 4.11, dar pentru un procesor care are numai un singur tip de instrucţiune: ramificaţie necondiţională relativă la PC. Care va fi durata ciclului pentru această cale de date?

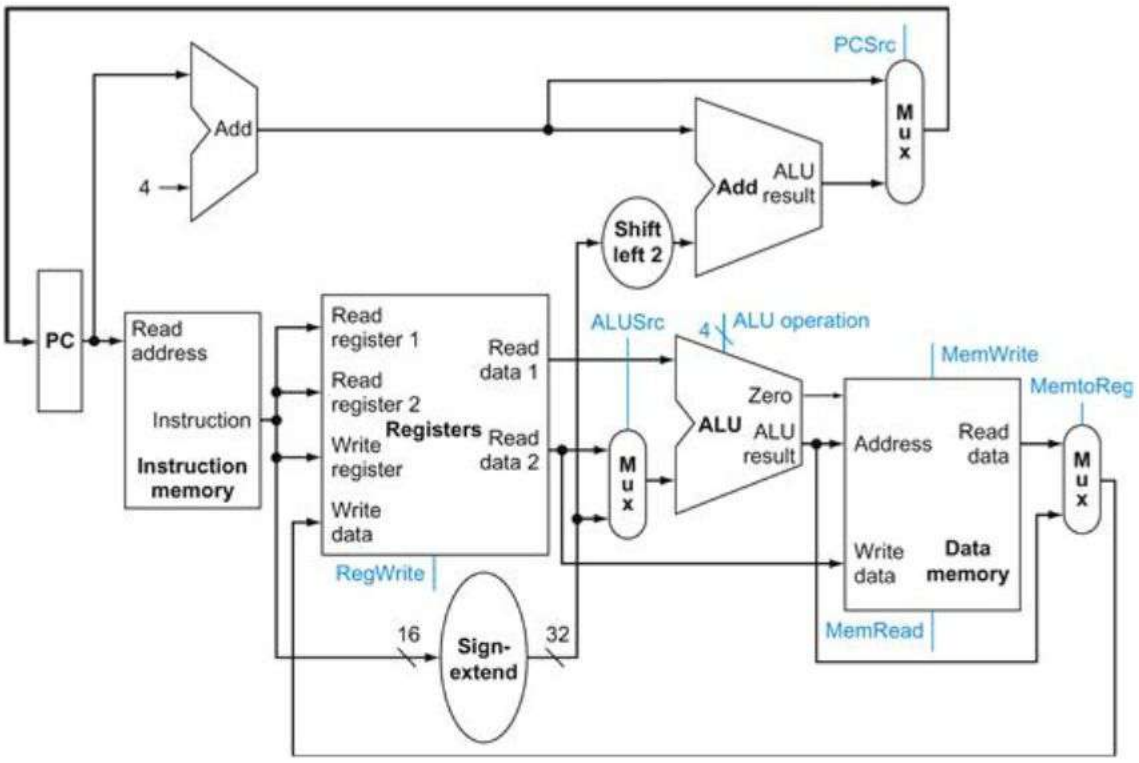


Figura 4.11

c) Repetaţi punctul anterior (b) dar de data aceasta trebuie suportată doar ramificaţie condiţionată relativă la PC.

d) Fie deplasarea la stânga cu 2. Care tipuri de instrucţiuni necesită această resursă?
e) Fie deplasarea la stânga cu 2. Pentru care tipuri de instrucţiuni (dacă există unul) este această resursă pe calea critică?
f) Fie deplasarea la stânga cu 2. Presupunând că sunt suportate doar instrucţiunile beq şi add, discutaţi cum schimbările în întârzierile acestei resurse afectează durata unui ciclu al procesorului. Presupuneţi că întârzierile altor resurse nu se schimbă.

25. Presupunem că nu există încetiniri (stalls) ale pipeline-ului şi că defalcarea instrucţiunilor dată de execuţia instrucţiunilor este:

add	addi	not	beq	lw	sw
20%	20%	0%	25%	25%	10%

a) În câte fracţiuni din toţi cicli sunt utilizate datele din memorie?
b) În câte fracţiuni din toţi cicli este utilizat circuitul pentru extensia de semn? Ce face circuitul atunci când intrările sale nu sunt necesare?

26. Atunci când se fabrică cipurile pe siliciu, defectele din materiale (siliciu) şi erorile de fabricaţie pot genera circuite defecte. Un defect foarte des întâlnit este ca un fir să afecteze semnalul din alt fir. Acest defect este denumit convorbire încrucişată (cross-talk). O clasă specială de eroare cross-talk este atunci când un semnal este conectat la un fir care are o valoare logică constantă (de exemplu un fir de alimentare). În acest caz avem o eroare de tip blocat pe 0 sau blocat pe 1, şi semnalul afectat are întotdeauna valoarea m0 respectiv 1. Se va lua în considerare în continuare bitul 0 de la intrarea Write Register din fişierul de regiştrii din figura 4.24.

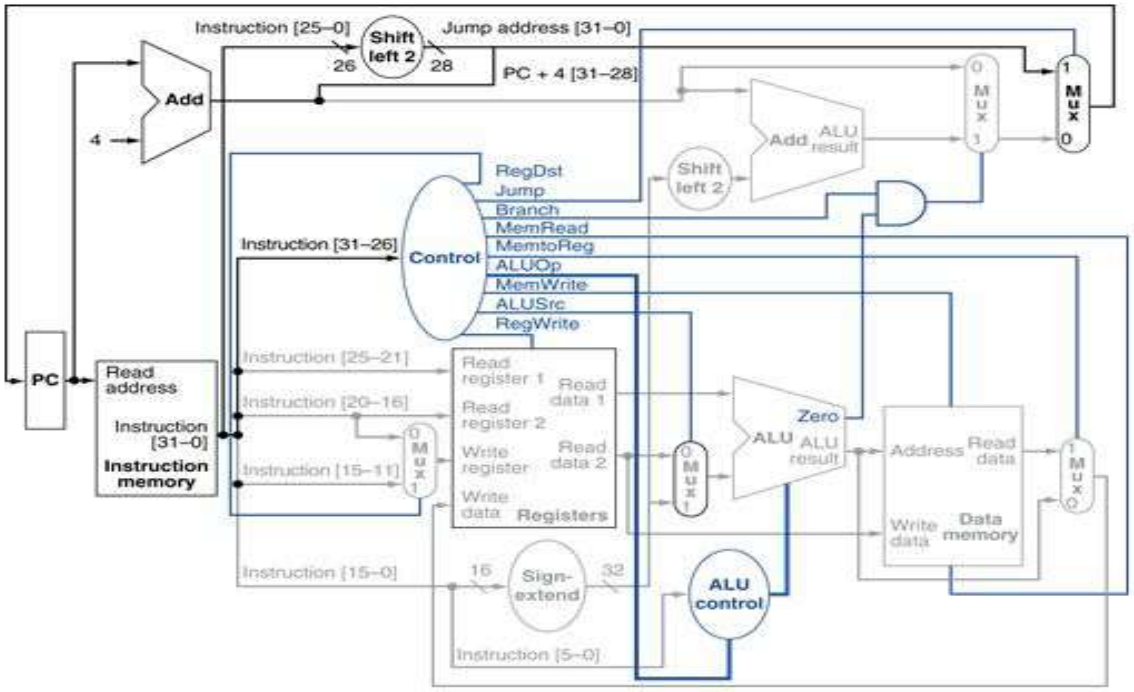


Figura 4.24

a) Presupuneţi că testarea procesorului este realizată prin umplerea PC-ului, regiştrilor, şi a memoriilor de date şi instrucţiuni cu unele valori (puteţi alege care valori), se permite execuţia unei singure instrucţiuni, după care se citesc PC, memoriile şi regiştrii. Aceste valori sunt apoi examinate pentru a determina dacă apare o anumită eroare. Puteţi proiecta un test (valorile pentru PC, memorii şi regiştrii) care va determina o eroare de tip blocare pe 0 pe acest semnal?
b) Repetaţi punctul anterior pentru eroare de tipul blcat pe 1. Puteţi utiliza un singur test atât pentru blocat pe 0 cât şi pentru blocat pe 1? Dacă da, explicaţi cum; dacă nu explicaţi de ce nu.
c) Dacă ştim că procesorul are un defect de tipul blocat pe 1 pe acest semnal, mai este procesorul utilizabil? Pentru a fi utilizabil, trebuie să fim capabil să convertim orice program care se execută pe un procesor MIPS normal într-un program care să ruleze pe acest procesor. Puteţi presupune că există suficiente memorie de instrucţiuni şi date liberă care să vă permită să faceţi programul mai lung şi să memoraţi date adiţionale. Sugestie: procesorul este utilizabil dacă fiecare instrucţiune afectată de acest defect poate fi înlocuită cu o secvenţă de instrucţiuni care au acelaşi efect.

d) Repetă punctul a, dar acum eroarea de testat este aceea că semnalul MemRead devine 0 dacă semnalul de control RegDst devine 0. Altfel nu este nici un defect.
e) Repetă punctul d, dar acum eroarea de testat este aceea că semnalul Jump devine 0 dacă semnalul de control RegDst devine 0. Altfel nu este nici un defect.

27. În acest exerciţiu vom examina în detaliu cum se execută o instrucţiune într-o cale de date cu un singur ciclu. Problema se referă la ciclul de ceas în care procesorul extrage următorul cuvânt instrucţiune:
10101100011000100000000000010100
Presupuneţi că memoria de date este toată filată cu 0 şi că regiştrii procesor au următoarele valori la începutul ciclului când este extrasă instrucţiunea anterioară:

r0	r1	r2	r3	r4	r5	r6	r8	r12	r31
0	–1	2	–3	–4	10	6	8	2	–16

a) Care sunt ieşirile unităţilor de extensie de semn şi de deplasare la stânga cu 2 pentru jump (vezi figura 4.24 în partea de sus) pentru acest cuvânt instrucţiune?
b) Care sunt valorile intrărilor unităţii de control pentru ALU pentru această instrucţiune?
c) Care este noua adresă din PC după ce este executată această instrucţiune? Accentuaţi calea pe care este determinată această valoare.
d) Pentru fiecare Muz arătaţi valorile ieşirilor de date pe durata execuţiei acestei instrucţiuni şi a acestor valori din regiştrii.
e) Pentru ALU şi cele două unităţi de adunare, care sunt valorile datelor de intrare?
f) Care sunt valorile pentru toate intrările unităţii “Registers”?

28. În acest exerciţiu, se va examina cum afectează pipline-ul durata ciclului de ceas a procesorului. Se presupune că etajele individuale ale căii de date au următoarele întârzieri:

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

De asemenea, presupuneţi că instrucţiunile executate de procesor sunt defalcate după cum urmează:

alu	beq	lw	sw
45%	20%	20%	15%

a) Care este durata ciclului de ceas într-un procesor pipelined şi în unul fără pipeline?
b) Care este întârzierea totală a unei instrucţiuni LW într-un procesor pipelined şi în unul fără pipeline?
c) Dacă putem împărţi un etaj din calea de date pipeline în două noi etaje, fiecare cu jumătate de întârziere faţă de etajul original, care etaj l-aţi înjumătăţii şi care este noua durată a ciclului de ceas a al procesorului?
d) Presupunând că nu există hazard sau încetiniri (stalls) care este utilizarea memoriei de date?

e) Presupunând că nu există hazard sau încetiniri (stalls), care este utilizarea portului write-register din unitatea “Registers”?

29. În acest exerciţiu, se va examina cum afectează dependenţele de date execuţia într-un pipeline de bază cu 5 etaje. Întrebările se referă la următoarea secvenţă de instrucţiuni:

```
or      r1,r2,r3
or      r2,r1,r4
or      r1,r1,r2
```

De asemenea, presupuneţi următorii timpi pentru ciclul de ceas pentru fiecare opţiune legată de avansare (forwarding):

Without Forwarding	With Full Forwarding	With ALU-ALU Forwarding Only
250 ps	300 ps	290 ps

a) Indicaţi dependenţele şi tipul lor.
b) Presupuneţi că nu există avansare în acest procesor. Indicaţi hazardul şi adăugaţi instrucţiuni nop necesare pentru a-l elimina.
c) Presupuneţi că există avansare completă. Indicaţi hazardul şi adăugaţi instrucţiuni nop necesare pentru a-l elimina.
d) Care este timpul total de execuţie pentru această secvenţă de instrucţiuni fără avansare (forwarding) şi cu avansare completă? Care este creşterea de viteză obţinută prin adăugarea unei avansări complete la o pipeline care nu are avansare?
e) Adăugaţi instrucţiuni nop la acest cod pentru a elimina hazardul dacă există numai o avansare ALU-ALU (nu există avansare de la etajul MEM la etajul EX).
f) Care este timpul total de execuţie a acestui secvenţe de instrucţiuni cu avansare doar de tipul ALU-ALU? Care este creşterea de viteză faţă de cazul fără avansare?

30. În acest exerciţiu, se va examina cum hazardul de resurse, cel de control şi proiectarea arhitecturii setului de instrucţiuni (ISA) poate afecta execuţia pipeline. Întrebările din această problemă se referă la următoarea secvenţă de cod MIPS:

```
sw      r16, 12(r6)
lw      r16, 8(r6)
beq     r5, r4, label    # se presupune că r5!=r4
add     r5, r1, r4
slt     r5, r15, r4
```

Presupuneţi că etajele pipeline individuale au următoarele întârzieri:

IF	ID	EX	MEM	WB
200ps	120ps	150ps	190ps	100ps

a) Pentru această întrebare, presupuneţi că toate ramificaţiile sunt perfect predictibile (acest fapt elimină hazardul de control) şi că nu se utilizează nici un slot de întârziere. Dacă avem o singură memorie (atât pentru instrucţiuni

şi date), există un hazard structural de fiecare dată când dorim să extragem o instrucţiune în acelaşi ciclu cu accesul la date al altei instrucţiuni. Pentru a garanta progresul avansării, acest hazard trebuie întotdeauna rezolvat în favoarea instrucţiunii care realizează accesul la date. Care este timpul total de execuţie a acestei secvenţe de instrucţiuni într-un pipeline cu 5 etaje care are o singură memorie? S-a văzut că hazardul de date poate fi eliminat prin adăugarea de cod de tip nop. Se poate realiza la fel şi cu hazardul structural? De ce?

b) Pentru această întrebare, presupuneţi că toate ramificaţiile sunt perfect predictibile (acest fapt elimină hazardul de control) şi că nu se utilizează nici un slot de întârziere. Dacă schimbăm instrucţiunile load/store astfel încât să utilizeze un registru (fără offset) ca adresă, aceste instrucţiuni nu mai au nevoie de ALU. Ca rezultat, etajele MEM şi EX pot fi suprapuse şi pipeline-ul are numai 4 etaje. Schimbaţi codul pentru acomodarea la această schimbare a ISA. Presupuneţi că această schimbare nu afectează durata ciclului de ceas, care este creşterea de viteză în secvenţa de instrucţiuni?

c) Presupuneţi că avem încetiniri la ramificaţii (stall-on-branch) şi nu avem sloturi de întârziere, care este creşterea de viteză a acestui cod dacă ieşirile ramificaţiilor sunt determinate în etajul ID, relativ la execuţia unde ieşirile ramificaţiilor sunt determinate în etajul EX?

d) Fiind date aceste întârzieri în etajele pipeline, repetaţi creşterea de viteză calculată la punctul b, dar luaţi în calcul (posibila) modificarea duratei ciclului de ceas. Când etajele MEM şi EX sunt executate într-un singur etaj, majoritatea execuţiei lor poate fi realizată în paralel. Ca rezultat, noul etaj EX/MEM are o întârziere care este mai mare ca cea originală a celor două etaje, plus 20ps necesare pentru execuţie care nu poate fi realizată în paralel.

e) Fiind date aceste întârzieri în etajele pipeline, repetaţi creşterea de viteză calculată la punctul c, dar luaţi în calcul (posibila) modificarea duratei ciclului de ceas. Presupuneţi că întârzierea etajului ID creşte cu 50% şi întârzierea etajului EX descreşte cu 10ps când ieşirea ramificației este mutată di EX în ID.

f) Presupuneţi că avem încetiniri la ramificaţii (stall-on-branch) şi nu avem sloturi de întârziere, care este noua durată a cilului de ceas şi timpul de execuție a acestei secvențe de instrucțiuni dacă se calculează adresa pentru beq în etajul MEM? Care este creşterea de viteză pentru această modificare? Presupuneţi că întârzierea pentru etajul EX se reduce cu 20ps şi întârzierea pentru etajul MEM este neschimbată atunci când ieşirea pentru ramificaţie este mutată din EX în MEM.

31. Presupuneţi următoarea buclă:

```
Loop:
lw      r1,0(r1)
and     r1,r1,r2
lw      r1,0(r1)
lw      r1,0(r1)
beq     r1,r0,loop
```

Presupuneţi că se utilizează o predicţie perfectă a salturilor (nici o încetinire datorită hazardului de control), că nu există sloturi de întârziere, şi pipeline-ul are suport complet pentru avansare. Presupuneţi că sunt executate multe iteraţii ale acestei bucle înainte de ieşire.

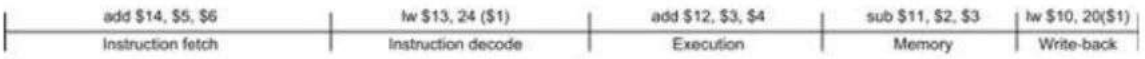
a) Prezentaţi diagrama de execuţie pentru a treia iteraţie a acestei bucle, de la ciclul în care se va extrage prima instrucţiune a acestei iteraţii până la (fără a include) ciclul în care se va extrage prima instrucţiune a următoarei iteraţii. Prezentaţi toate instrucţiunile care sunt în pipeline pe durata acestor cicli (nu doar cei din iteraţia a treia).
b) Cât de des (ca procent din toţi cicli) există un ciclu în care toate cele cinci etaje pipepine realizează o muncă (execuţie) utilă?

32. Acest exerciţiu încearcă să vă ajute să înţelegeţi costul/ complexitatea/ performanţele avansării (forwarding) într-un procesor pipelined. Exerciţiul se referă la calea de date din figura 4.5. Întrebările din problemă presupun că, toate instrucţiunile au un tip particular de dependenţe de date de tip RAW (Read After Write – citire după scriere). Tipul de dependenţă de date RAW este identificat prin etajul care produce rezultatul (EX sau MEM) şi instrucţiunea care consumă rezultatul (prima instrucţiune care urmează unei care produce rezultatul, sau a doua instrucţiune, sau ambele). Presupunem că scrierea în registru este realizată în prima parte a ciclului maşină în timp ce citirea se realizează în a doua jumătate a ciclului maşină, astfel încât dependenţele de date de tipul EX cu a treia instrucţiune care urmează şi MEM cu a treia instrucţiune care urmează nu sunt luate în calcul deoarece nu conduc la hazard de date. De asemena presupuneţi cî CPI-ul procesorului este 1 dacă nu există nici un hazard de date.

EX to 1st Only	MEM to 1st Only	EX to 2nd Only	MEM to 2nd Only	EX to 1st and MEM to 2nd	Other RAW Dependences
5%	20%	5%	10%	10%	10%

Presupuneţi următoarele întârzieri pentru etajele individuale ale pipeline-ului. Pentru etajul EX, întârzierile sunt date separat pentru un procesor fără avansare şi pentru un procesor cu diferite tipuri de avansare.

IF	ID	EX (no FW)	EX (full FW)	EX (FW from EX/MEM only)	EX (FW from MEM/ WB only)	MEM	WB
150 ps	100 ps	120 ps	150 ps	140 ps	130 ps	120 ps	100 ps



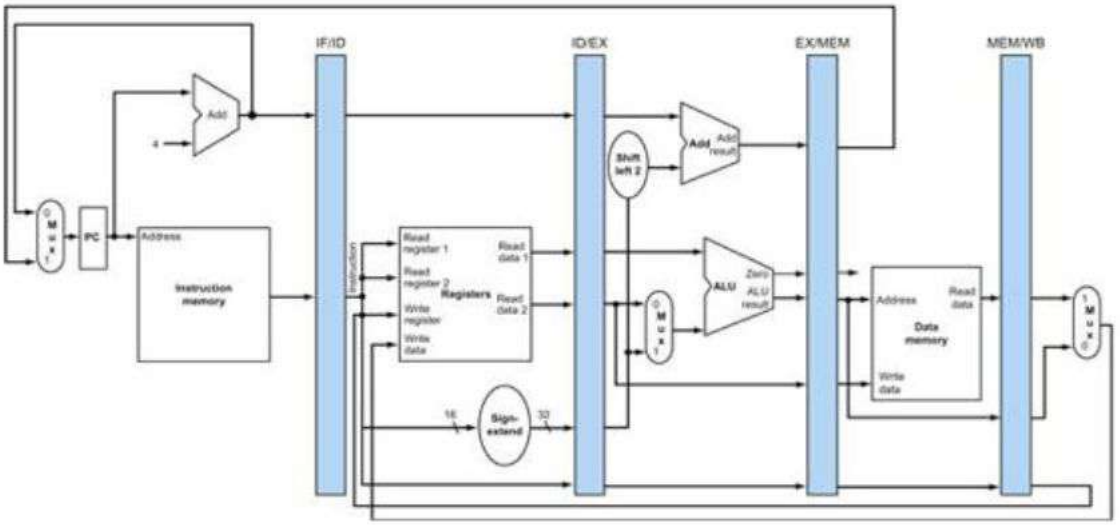


Figura 4.45

a) Dacă nu se utilizează avansarea, ce fracţiune din ciclii sunt încetiniţi datorită hazardului?
b) Dacă utilizăm avansarea completă (se avansează toate rezultatele care pot fi avansate), ce fracţiune din ciclii sunt încetiniţi datorită hazardului?
c) Să presupunem că nu putem avea multiplexoare cu trei intrări necesare pentru avansarea completă. Trebuie să decidem dacă este mai bine să avansăm numai din registrul pipeline EX/MEM (avansare din ciclul următor), sau numai din registrul pipeline MEM/WB (avansare cu doi cicli). Care dintre aceste două soluţii generează mai puţine încetiniri pentru cicli de date?
d) Pentru o probabilitate dată a hazardului şi întârzieri ale etajelor pipeline, care este câştigul în viteză dat de adăugarea la pipeline unei unităţi de avansare complete faţă de un pipeline fără avansare?
e) Care va fi sporul adiţional de viteză (relativ la un pipeline cu avansare) dacă se utilizează o avansare prin călătorie în timp (time-trevel forwarding) care elimină toate hazardurile? Presupuneţi că acest circuit (neinventat încă) adaugă o întârziere de 100ps la unitatea de avansare completă pentru etajul EX.
f) Repetaţi c) dar de data aceasta determinaţi care dintre cele două opţiuni conduce la un timp mai scurt pe instrucţiune.

33. Acest exerciţiu încearcă să vă ajute să înţelegeţi relaţiile între avansare, detecţia hazardului şi proiectarea ISA (arhitectura setului de instrucţiuni). Întrebările din această problemă se referă la următoarea secvenţă de instrucţiuni, şi presupuneţi că se execută pe o cale de date pipeline cu 5 etaje:

```
add   r5,r2,r1
lw    r3,4(r5)
lw    r2,0(r2)
or    r3,r5,r3
sw    r3,0(r5)
```

a) Dacă nu există avansare sau detecţia hazardului, inseraţi nops pentru a asigura execuţia corectă.
b) Repetaţi punctul a) dar acum utilizaţi nops numai dacă hazardul nu poate fi evitat prin schimbarea sau rearanjarea acestor instrucţiuni. Puteţi presupune că, în codul vostru modificat, registrul R7 poate fi utilizat pentru a păstra valori temporare.
c) Dacă procesorul are avansare, dar am uitat să implementăm unitatea de detecţie a hazardului, ce se întâmplă când se execută acest cod?
d) Dacă există avansare, pentru primii cinci ciclii din execuţia acestui cod, specificaţi care semnale sunt activate în fiecare ciclu de detecţia hazardului şi unitatea de avansare din figura 4.60.
e) Dacă nu există avansare, ce noi semnale de intrare şi ieşire sunt necesare pentru unitatea de detecţie a hazardului din figura 4.60? Utilizând această secvenţă de instrucţiuni ca un exemplu, explicaţi de ce este necesar fiecare semnal.
f) Pentru noua unitate de detecţie a hazardului de la punctul e) specificaţi care semnale de ieşire sunt activate în fiecare dintre primii cinci cicli procesor pe durata execuţiei acestui cod.

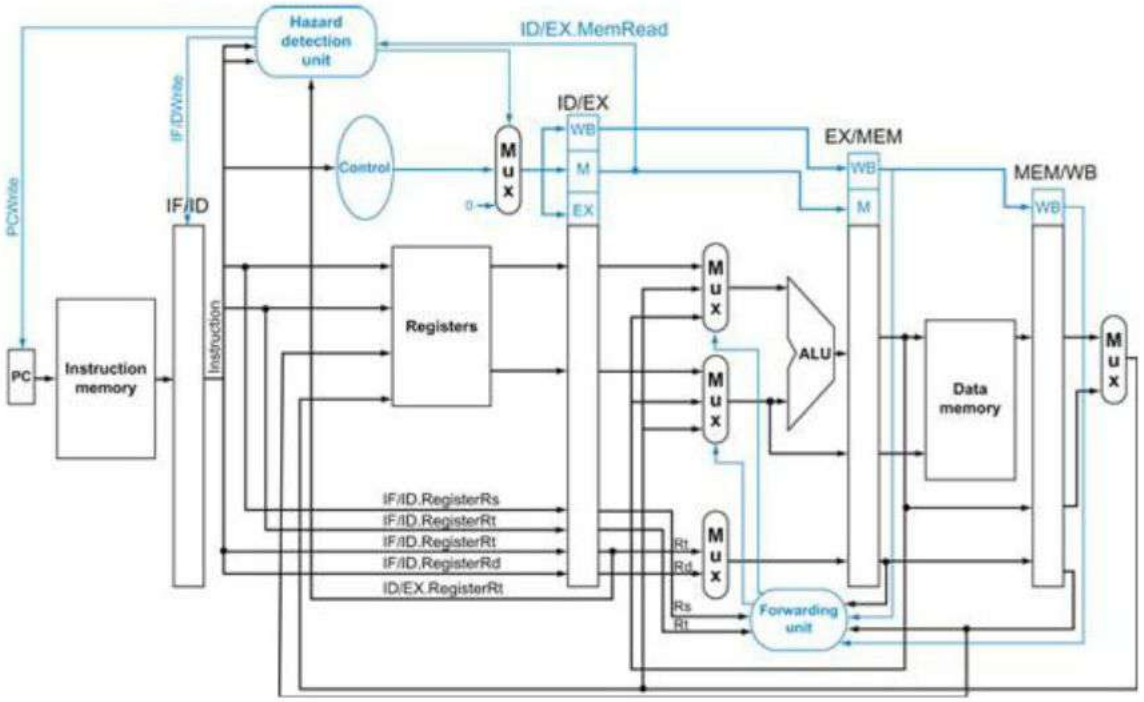


Figura 4.60

34. Acest exerciţiu încearcă să vă ajute să înţelegeţi relaţia între sloturile de întârziere, hazardul de control şi execuţia ramificaţiilor într-un procesor cu pipeline. În acest execrciţiu, presupunem că se execută următorul cod MIPS pe un procesor cu pipeline cu 5 etaje, avansare completă, şi un predictor cu predicţia conform căreia ramificaţia are loc (predict-taken).

```
lw      r2,0(r1)
labell1:
beq     r2,r0,label2    #nu se ramifică (not taken) o dată, apoi se ramifică (taken)
lw      r3,0(r2)
beq     r3,r0,label1    # ramificaţia se execută (taken)
add     r1,r3,r1
label2:
sw      r1,0(r2)
```

a) Desenaţi diagrama pipeline pentru acest cod, presupunând că nu există sloturi de întârziere şi că ramificaţiile se execută în etajul EX.
b) Repetaţi punctul a), dar presupuneţi că sunt utilizate sloturile de întârziere. În codul dat, instrucţiunea care urmează ramificaţiei este acum instrucţiunea din slotul de întârziere pentru ramificaţie.
c) O cale de a muta rezoluţia ramificaţiei un etaj mai târziu este aceea de a nu avea nevoie de o operaţie ALU în ramificaţiile condiţionate. Instrucţiunea de ramificaţia va fi *bez rd, label* şi *bnez rd, label*, şi se va ramifica dacă registrul are sau respectiv nu are valoarea zero. Schimbaţi codul pentru a utiliza aceste instrucţiuni de ramificaţie în loc de *beq.* Puteţi presupune că registrul R8 este disponibil pentru ca să-l utilizaţi ca registru temporar, şi că poate fi utilizată o instrucţiune *seq* (set dacă este egal) de tip R.
d) Utilizând prima instrucţiune de ramificaţie din codul dat ca un exemplu, descrieţi logica de detecţie a hazardului necesară pentru a suporta execuţia ramificaţiei în etajul ID ca în figura 4.62. Care tip de hazard este presupus a fi detectat cu această nouă logică?

e) Pentru codul dat care este creşterea de viteză datorată mutării execuţiei ramificaţiei în etajul ID? Explicaţi răspunsul. În calculul creşterii de viteză, presupuneţi că această comparaţie adiţională în etajul ID nu afectează durata ciclului de ceas.

f) Utilizând prima instrucţiune de ramificaţie din codul dat ca un exemplu, descrieţi suportul pentru avansare care trebuie adăugat pentru a realiza execuţia ramificaţiei în etajul ID. Comparaţi complexitatea acestei noi unităţi de avansare cu complexitatea unităţii de avansare existente în figura 4.62.

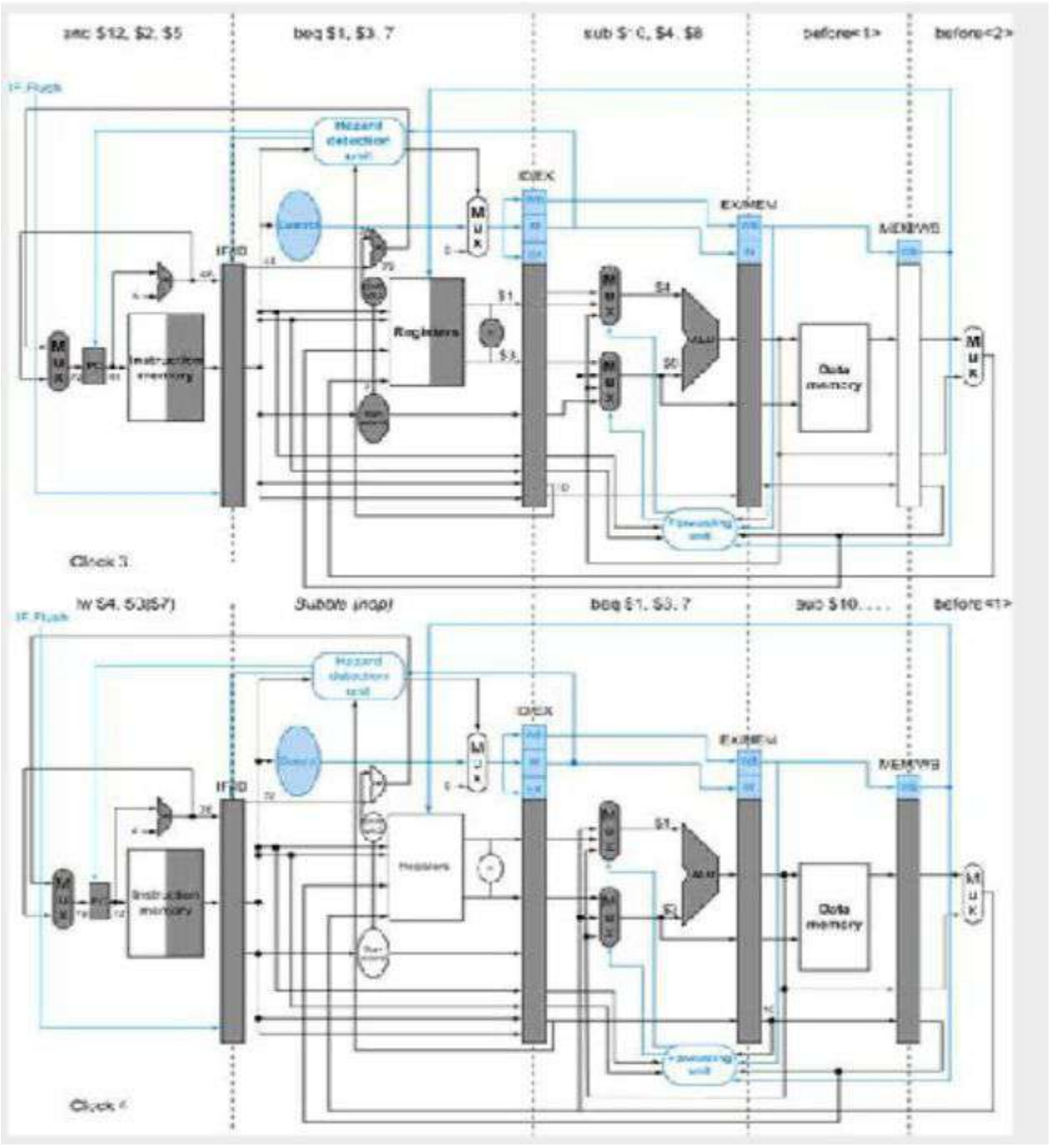


Figura 4.62

35. Importanţa de a avea un predictor bun depinde adesea de modul cum sunt executate ramificaţiile condiţionate. Pe lângă acurateţea predicţiei ramificaţiilor, aceasta va determina cât de mult timp se petrece cu încetinirile datorită greşelilor de predicţie. În acest exerciţiu, presupuneţi că eşecurile execuţiei dinamice a instrucţiunilor pe diferite categorii de instrucţiuni sunt următoarele:

R-Type	BEQ	JMP	LW	SW
40%	25%	5%	25%	5%

Presupuneţi de asemenea următoarea acurateţe a predictorului pentru ramificaţii:

Always-Taken	Always-Not-Taken	2-Bit
45%	55%	85%

a) Cicli de încetinire datorită greşelilor de predicţie cresc CPI-ul (cicli pe instrucţiune). Care este creşterea CPI dată de greşelile de predicţie date de un predictor de tipul ramificaţia este întotdeauna luată (realizată)? Presupuneţi că ieşirea ramificaţiei este determinată în etajul EX, şi ca urmare nu apare hazard de date, şi nu se utilizează nici un slot de întârziere.
b) Repetaţi punctul a) pentru un predictor de tipul "ramificaţia întotdeauna nu este luată"
c) Repetaţi punctul a) pentru un predictor cu 2 biţi.
d) Cu un predictor cu 2 biţi, care este creşterea în viteză dacă am putea converti o jumătate din instrucţiunea de ramificaţie într-un mod care să înlocuiască o instrucţiune de ramificaţie cu o instrucţiune ALU? Presupuneţi există aceeaşi şansă de înlocuire atât pentru instrucţiune prezisă corect cât şi pentru aceea prezisă incorect.
e) Cu un predictor cu 2 biţi, care este creşterea în viteză dacă am putea converti o jumătate din instrucţiunea de ramificaţie într-un mod care să înlocuiască fiecare instrucţiune de ramificaţie cu două instrucţiuni ALU? Presupuneţi există aceeaşi şansă de înlocuire atât pentru instrucţiune prezisă corect cât şi pentru aceea prezisă incorect.
f) Unele instrucţiuni de ramificaţie sunt mult mai predictibile decât altele. Dacă cunoaştem 80% din toate instrucţiunile de ramificaţie executate, este uşor să prezicem ramificaţiile de întoarcere din buclă care sunt întotdeauna prezise corect, care este acurateţea unui predictor pe 2 biţi pentru cele 20% de instrucţiuni de ramificaţie?

36. Acest exerciţiu examinează acurateţea diferiţilor predictori pentru ramificaţii pentru următorul şablon repetitiv (de exemplu într-o buclă) al ieşirilor ramificaţiei T, NT, T, T, NT.

a) Care este acurateţea unor predictori de tipul ramificaţia este întotdeauna luată sau întotdeauna nu este luată pentru această secvenţă de ieşire a ramificaţiei?
b) Care este acurateţea unui predictor cu 2 biţi pentru primele 4 ramificaţii din acest şablon, presupunând că predictorul porneşte în starea din stânga jos din figura 4.63 (predicţia nu este luată)?

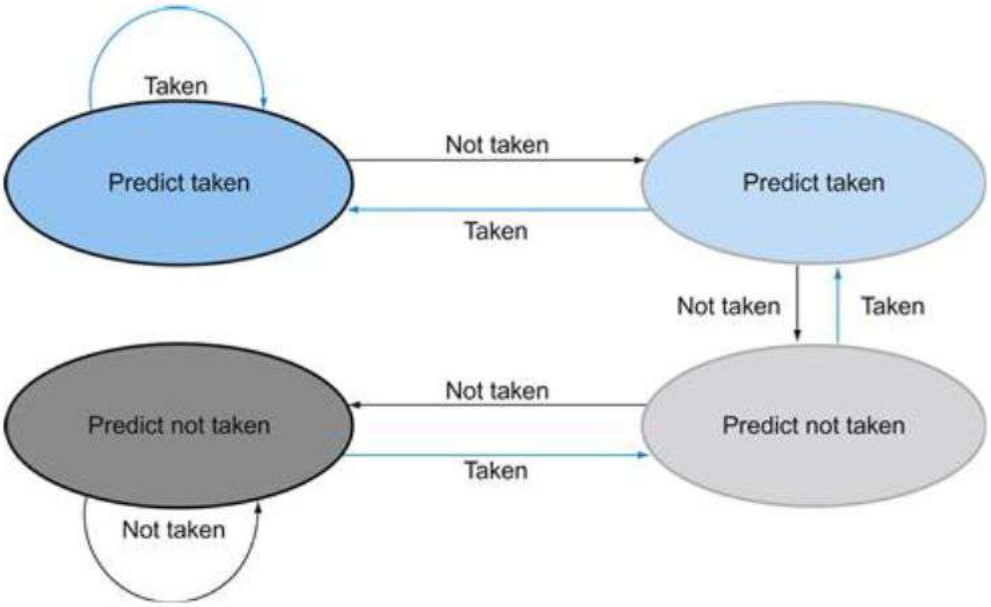


Figura 4.63

c) Care este acurateţea unui predictor cu 2 biţi dacă acest şablon se repetă la infinit?
d) Proiectaţi un predictor care va atinge o acurateţe perfectă dacă acest şablon se va repeta la infinit. Predictorul vostru trebuie să fie un circuit secvanţial cu o ieşire care furnizează predicţia (1 pentru ramificaţie luată şi 0 pentru ramificaţie care nu este luată).
e) Care este acurateţea pentru predictorul de la punctul d) dacă este dat un şablon de repetare exact în opoziţie cu cel din problemă?
f) Repetaţi punctul d), dar acum predictorul va trebui să fie capabil ca eventual (după o perioadă de încălzire pe durata căreia poate face predicţii greşite) să înceapă o predicţie perfectă atât pentru şablonul din problemă cât şi pentru opusul acestuia. Predictorul vostru trebuie o intrarea care să-i spună care a fost ieşirea reală. Sugestie: această intrare permite predictorului vostru să determine care din cele două şabloane este activ.

37. Acest execrciţiu explorează modul în care excepţiile afectează proiectarea pipeline-ului. Primele trei întrebări se referă la următoarele două instrucţiuni:

Instruction 1	Instruction 2
BNE R1, R2, Label	LW R1, 0(R1)

a) Care excepţie poate fi generată de fiecare dintre aceste instrucţiuni? Pentru fiecare dintre aceste excepţii, specificaţi etajul pipeline în care este detectată.
b) Dacă există o adresă separată pentru handler-ul fiecărei excepţii, prezentaţi cum trebuie schimbată organizarea pipeline-ul pentru a fi capabilă să gestioneze această execuţie. Puteţi presupune cp adresele acestor handler-e sunt cunoscute atunci când este proiectat procesorul.
c) Dacă a doua instrucţiune este extrasă imediat după prima instrucţiune, descrieţi ce se întâmplă în pipeline dacă prima instrucţiune cauzează prima excepţie pe care aţi listat-o la punctul a). Prezentaţi diagrama de execuţie pipeline de la

momentul în care este extrasă prima instrucţiune până când este completată prima instrucţiune din handler-ul de tratare a excepţiei.

d) Atunci când se utilizează gestiunea vectorizată a excepţiilor, tabelul cu adresele handler-elor de tratare a excepţiilor este în memorie la o adresă (fixă) cunoscută. Modificaţi pipeline-ul pentru a implementa acest mecanism de gestiune a excepţiilor. Repetaţi punctul c) utilizând acest pipeline modificat şi gestiunea vectorizată a excepţiilor.

e) Dorim să emulăm gestiunea vectorizată a excepţiilor pe un procesor care are doar o adresă fixă pentru un handler. Scrieţi codul care trebuie să fie la această adresă.

38. În acest exerciţiu se compară performanţele unui procesor care execută o instrucţiune pe cilu şi unul care execută două instrucţiuni pe ciclu, luând în calcul transformările din program care pot fi realizate pentru a optimiza execuţia cu două căi (două instrucţiuni pe ciclu). Întrebările din acest exerciţiu se referă la următoarea buclă (programul este scris în C):

```
for (i=0; i!=j; i+=2)
b[i] = a[i] – a[i+1]
```

Când scrieţi codul MIPS presupuneţi că variabilele sunt păstrate în regiştrii după cum urmează, şi că toţi regiştrii cu excepţia celor indicaţi ca fiind liberi (Free) sunt utilizaţi pentru a păstra diferite variabile, astfel încât nu trebuie folosiţi în oricare altă parte.

i	j	a	b	c	Free
R5	R6	R1	R2	R3	R10, R11, R12

a) Translataţi codul C în instrucţiuni MIPS. Translatarea trebuie să fie directă, fără rearanjarea instrucţiunilor pentru a obţine performanţe mai bune.

b) Dacă există bucle după execuţia a doar două iteraţii, desenaţi o diagramă pipeline pentru codul MIPS al vostru ca parte a execuţiei pe procesorul cu două căi prezentat în figura 4.69. Presupuneţi că procesorul are un predictor perfect pentru predicţia ramificaţiilor şi poate extrage oricare două instrucţiuni (nu doar instrucţiuni consecutive) în acelaşi ciclu.

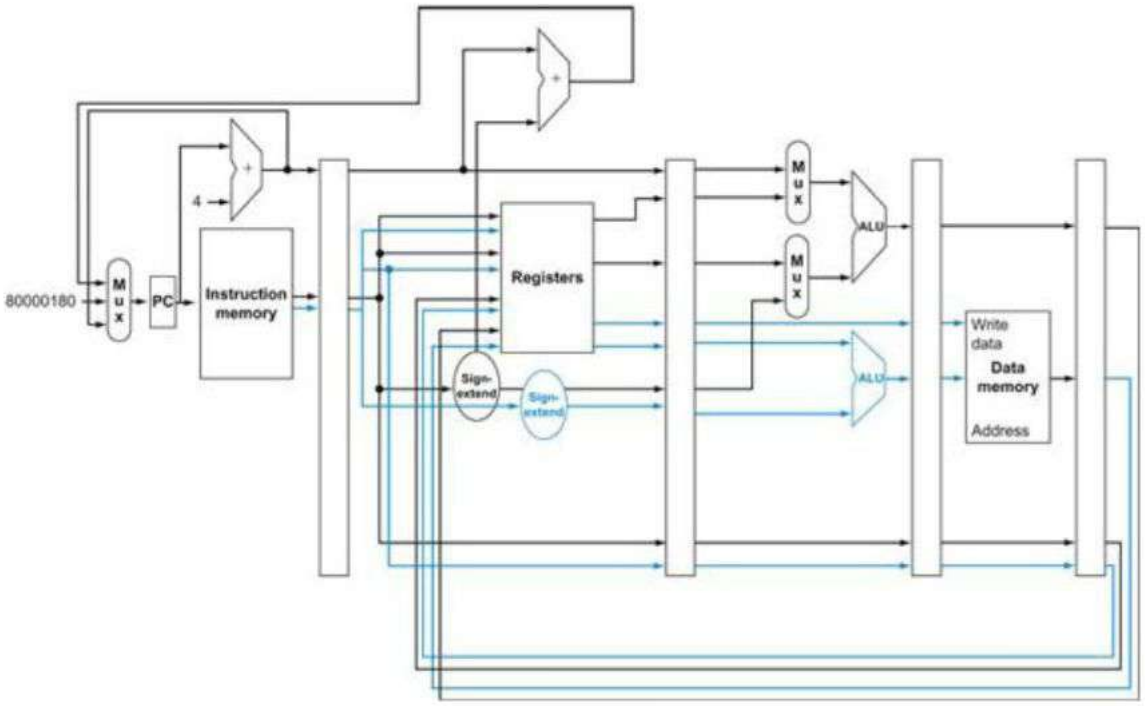


Figura 4.69

c) Rearanjaţi codul de la punctul a) pentru a atinge performanţe mai bune pe procesorul cu planificare statică cu două căi din figura 4.69.
d) Repetaţi punctul b) dar de data aceasta utilizaţi codul MIPS de la punctul c).
e) Care este creşterea de viteză obţinută prin trecerea de la un procesor cu o cale la procesorul cu două căi din figura 4.69? Utilizaţi codul de la punctul a) pentru ambele tipuri de procesoare şi presupuneţi că sunt executate 1000000 de iteraţii ale buclei. Ca la punctul b) presupuneţi că procesorul are un predictor perfect pentru ramificaţii, şi că procesorul cu 2 căi poate extrage oricare două instrucţiuni în acelaşi ciclu.
f) Repetaţi punctul e) dar presupuneţi de data aceasta că pentru procesorul cu două căi una din instrucţiunile care se pot într-un ciclu poate fi de orice fel dar a doua trebuie să fie o instrucţiune care nu lucrează cu memoria.

39. Acest exerciţiu explorează eficienţa energetică şi relaţia acesteia cu performanţele. Întrebările din acest exerciţiu presupun următoarele consumuri energetice pentru activităţile din memoria de instrucţiuni, regiştrii şi memoria de date. Puteţi presupune că celelalte componente din calea de date consumă o cantitate neglijabilă d energie.

I-Mem	1 Register Read	Register Write	D-Mem Read	D-Mem Write
140pJ	70pJ	60pJ	140pJ	120pJ

Presupuneţi că, componentele din calea de date au următoarele întârzieri. Puteţi presupune că celelalte componente din calea de date au întârzieri neglijabile.

I-Mem	Control	Register Read or Write	ALU	D-Mem Read or Write
200ps	150ps	90ps	90ps	250ps

a) Cât de multă energie este consumată pentru a executa o instrucţiune ADD pentru o proiectare cu un singur ciclu şi pentru o proiectare pipeline cu 5 etaje?
b) Care este cazul cel mai defavorabil de instrucţiune MIPS privind consumul de energie, şi care este consumul de energie necesar pentru a o executa?
c) Dacă reducerea consumului de energie este criteriul de bază al proiectării, cum se va schimba aceasta pentru pipeline? Care este procentul de reducere al consumului energetic pentru o instrucţiune LW după această schimbare?
d) Care este impactul schimbării pe care aţi realizat-o la punctul c) privind performanţele?
e) Putem elimina semnalul de control MemRead şi putem citi datele din memoria de date în fiecare ciclu, adică putem avea semnalul MemRead permanent pe 1. Explicaţi de ce procesorul totuşi mai lucrează corect după această modificare. Care este efectul acestei modificări asupra frecvenţei de ceas şi a consumului energetic?
f) Dacă o unitate în aşteptare consumă 10% din puterea pe care o consumă dacă este activă, care este energia consumată de memoria de instrucţiuni în fiecare ciclu? Ce procent din energia totală consumată de memoria de instrucţiuni reprezintă această energie când blocul este în aşteptare?

Probleme propuse pentru examenul de la disciplina Structura și Organizarea Calculatoarelor

== MOODLE TYPE ==

Prof. dr. ing. Vasile Gheorghiță GĂITAN

Ș. I. dr. ing. Ionel ZAGAN

Cap. 4

Studiul de caz 1: optimizarea performanțelor memoriei cache via tehnici avansate.

Conceptele ilustrate:

- Cache fără blocare,
- Optimizările oferite de compilator pentru cache,
- Pre-extragere software și hardware,
- Calculul impactului performanțelor cache-ului în procesoare din ce în ce mai complexe.

1 Transpusa unei matrice schimbă între ele rândurile și coloanele așa cu se prezintă în continuare:

$A11\ A12\ A13\ A14\ A21\ A22\ A23\ A24\ A31\ A32\ A33\ A34\ A41\ A42\ A43\ A44$ =>
$A11\ A21\ A31\ A41\ A12\ A22\ A32\ A42\ A13\ A23\ A33\ A43\ A14\ A24\ A34\ A44$

Fie următorul cod C simplu care realizează transpusa:

```
for (i = 1; i <5; i++) {
  for (j = 1; j <5; j++) {
    ouput [j][i] = input [j][i];
  }
}
```

Presupuneți că atât matricea de intrare cât și cea de ieșire sunt memorate în ordinea bazată pe rândul major (row major order – ceea ce înseamnă că indicele rândului se modifică cel mai repede). Presupuneți că executați o transpunere dublă precizie 256 x 256 pe un procesor cu memorie cache L1 de 16 KB, pe deplin asociativă cu algoritmul LRU pentru politica de înlocuire a paginilor, organizată pe blocuri de 64 de octeți. Presupuneți că lipsurile sau pre-extragerile (misses) la nivelul L1 necesită 16 cicli de ceas și există întotdeauna potrivire la nivelul memorie cache L2, și că L2 poate procesa o cerere la fiecare doi cicli procesor. Presupuneți că fiecare iterație din bucla interioară necesită patru cicli dacă data este prezentă în L1. Presupuneți că cache-ul are pentru lipsurile la scriere o politică de tipul write-allocate (alocă line cache) fetch-on-write (extrage blocul din nivelul imediat inferior). Deși nerealist, presupuneți că scrierea înapoi a paginilor modificate necesită 0 cicli. Pentru implementarea simplă dată anterior, ordinea de execuție nu va fi ideală pentru matricea de intrare; totuși aplicând o optimizare prin interschimbarea buclelor se va crea o ordine neideală pentru matricea de ieșire. Deoarece interschimbarea buclelor nu este suficientă pentru îmbunătățirea performanțelor , în loc acestea trebuie blocată execuția.

a. Care trebuie să fie mărimea minimă a cache-ului pentru a profita de avantajul blocării execuției?
b. Care va fi numărul relativ de lipsuri pentru versiunile cu blocare sau fără blocare în comparație cu mărimea minimă aleasă anterior?
c. Scrieți cod pentru a realiza transpusa cu mărimea blocului B care utilizează B x B blocuri?
d. Care este minimul de asociativitate necesară pentru cache-ul L1 pentru a obține performanțe consistente independent de poziția ambelor matrice în memorie?

e. Încercați transpoziția cu blocare sau fără blocare a unei matrice de 256 x 256 pe un calculator. Cât de aproape se potrivesc rezultatele cu așteptările voastre bazate pe ceea ce cunoașteți despre sistemul de memorie a calculatoarelor? Explicați, dacă este posibil, orice discrepanță.

2 Presupuneți că proiectați un prefetcher (un dispozitiv de pre extragere) pentru codul de transpunere fără blocare a unei matrice din problema 1. Cel mai simplu tip de prefetcher hardware pre-extrage numai secvențial blocuri cache la apariția unei miss (lipse). Prefetcher-ele hardware mai complicate de tip "non-unit stride" (pas non-unitate) [1] pot analiza șirul de referințe care creează lipsuri (miss) și să detecteze și să pre-extragă cu pași non-unitate. În contrast, pre extragerea software poate determina pași non-unitate la fel de ușor cum poate determina și pașii unitate. Presupuneți că pre extragerea scrie direct în cache și că nu există "poluare" (suprascrierea datelor care trebuie utilizate înaintea ca datele să fie pre extrase). Pentru a obține cele mai bune performanțe utilizând un prefetcher non-unitate, câte **pre extrageri trebuie pot fi în aștepare (outstanding la un moment dat,** dacă bucla internă este în starea de echilibru?

3

Studiul de caz 2: Să le adunăm pe toate la un loc: sisteme de memorie cu un grad ridicat de paralelism.

Conceptele ilustrate:

- Facilități transversale,
- Proiectarea ierarhiei de memorie .

4 Programul ce urmează poate fi utilizat pentru a evalua comportarea unui sistem de memorie. Cheia este aceea de a avea o evaluarea precisă privind timpul și pasul programului prin memorie invocând diferite niveluri ale ierarhiei. Programul este scris în C. Prima parte este o procedură care utilizează un utilitar standard pentru a prelua o măsură cu acuratețe mare privind timpul CPU al utilizatorului; această procedură poate fi schimbată pentru a lucra pe diferite sisteme. Partea a doua, este o buclă cuibărite pentru a citi și a scrie memoria cu diferiți pași (strides) și diferite mărimi ale memorie cache. Pentru a prelua cu acuratețe timpul, acest cod este repetat de multe ori. Partea a treia măsoară, doar pentru a fi extras din timpul total măsurat, timpul pentru supracontrolul din buclele cuibărite pentru a vedea cât de lung este accesul. Rezultatele sunt memorate într-un fișier de ieșire de format .CSV pentru a facilita importul în programe de calcul tabelar. Se poate modifica CACHE_MAX în funcție de întrebarea la care trebuie să răspundeți și mărimea memoriei din sistemul pe care l-ați măsurat. Execuția programului în modul singur utilizator sau cel puțin fără a activa alte

---

[1] "Stride"(pas) reprezintă numărul de octeți pe care cineva trebuie să-i mute pentru a trece de la un element la altul (la următorul sau la precedentul) . Pasul nu poate fi mai mic decât mărimea elementului, dar poate fi mai mare, indicând un extra spațiu între elemente. O arie cu pasul de aceeași mărime cu mărimea fiecărui element este continuă în memorie. O astfel de arie se spune că are **pasul unitate** (unit stride). Non-unit stride arrays (ariile cu pasul **non-unitate**) sunt uneori mai eficiente în particular pentru matrici (2D) sau pentru arii multidimensionale, depinzând de efectul asupra memoriei cache și de șablonul de acces ales).

programe va da rezultate mult mai consistente. Presupuneți că mărimea tuturor componentelor din ierarhia de memorie sunt puteri ale lui 2. Presupuneți că mărimea unei pagini este mult mai mare ca mărimea unui bloc din al doilea nivel de memorie cache, și că mărimea unui bloc din al doilea nivel de cache este mai mare sau egal cu m[rimea unui bloc de pe primul nivel de memorie cache. În figura 2.30 este trasat un exemplu de program; cheile indică mărimea ariilor testate.

```
#include "stdafx.h"
#include <stdio.h>
#include <time.h>
#define ARRAY_MIN (1024) /* 1/4 smallest cache */
#define ARRAY_MAX (4096*4096) /* 1/4 largest cache */
int x[ARRAY_MAX]; /* array going to stride through */

double get_seconds() { /* routine to read time in seconds */
  __time64_t ltime;
  _time64( &ltime );
  return (double) ltime;
}
int label(int i) {/* generate text labels */
  if (i<1e3) printf("%1dB,",i);
  else if (i<1e6) printf("%1dK,",i/1024);
  else if (i<1e9) printf("%1dM,",i/1048576);
  else printf("%1dG,",i/1073741824);
  return 0;
}
int _tmain(int argc, _TCHAR* argv[]) {
int register nextstep, i, index, stride;
int csize;
double steps, tsteps;
double loadtime, lastsec, sec0, sec1, sec; /* timing variables */

/* Initialize output */
printf(" ,");
for (stride=1; stride <= ARRAY_MAX/2; stride=stride*2)
  label(stride*sizeof(int));
printf("\n");

/* Main loop for each configuration */
for (csize=ARRAY_MIN; csize <= ARRAY_MAX; csize=csize*2) {
  label(csize*sizeof(int)); /* print cache size this loop */
  for (stride=1; stride <= csize/2; stride=stride*2) {

    /* Lay out path of memory references in array */
    for (index=0; index < csize; index=index+stride)
      x[index] = index + stride; /* pointer to next */
    x[index-stride] = 0; /* loop back to beginning */

    /* Wait for timer to roll over */
    lastsec = get_seconds();
    do sec0 = get_seconds(); while (sec0 == lastsec);
```

```
    /* Walk through path in array for twenty seconds */
    /* This gives 5% accuracy with second resolution */
    steps = 0.0; /* number of steps taken */
    nextstep = 0; /* start at beginning of path */
    sec0 = get_seconds(); /* start timer */
    do { /* repeat until collect 20 seconds */
      for (i=stride;i!=0;i=i-1) { /* keep samples same */
        nextstep = 0;
        do nextstep = x[nextstep]; /* dependency */
        while (nextstep != 0);
      }
      steps = steps + 1.0; /* count loop iterations */
      sec1 = get_seconds(); /* end timer */
    } while ((sec1 - sec0) < 20.0); /* collect 20 seconds */
    sec = sec1 - sec0;

    /* Repeat empty loop to loop subtract overhead */
    tsteps = 0.0; /* used to match no. while iterations */
    sec0 = get_seconds(); /* start timer */
    do { /* repeat until same no. iterations as above */
      for (i=stride;i!=0;i=i-1) { /* keep samples same */
        index = 0;
        do index = index + stride;
        while (index < csize);
      }
      tsteps = tsteps + 1.0;
      sec1 = get_seconds(); /* - overhead */
    } while (tsteps<steps); /* until = no. iterations */
    sec = sec - (sec1 - sec0);
    loadtime = (sec*1e9)/(steps*csize);
    /* write out results in .csv format for Excel */
    printf("%4.1f,", (loadtime<0.1) ? 0.1 : loadtime);
    }; /* end of inner for loop */
    printf("\n");
  }; /* end of outer for loop */
  return 0;
}
```

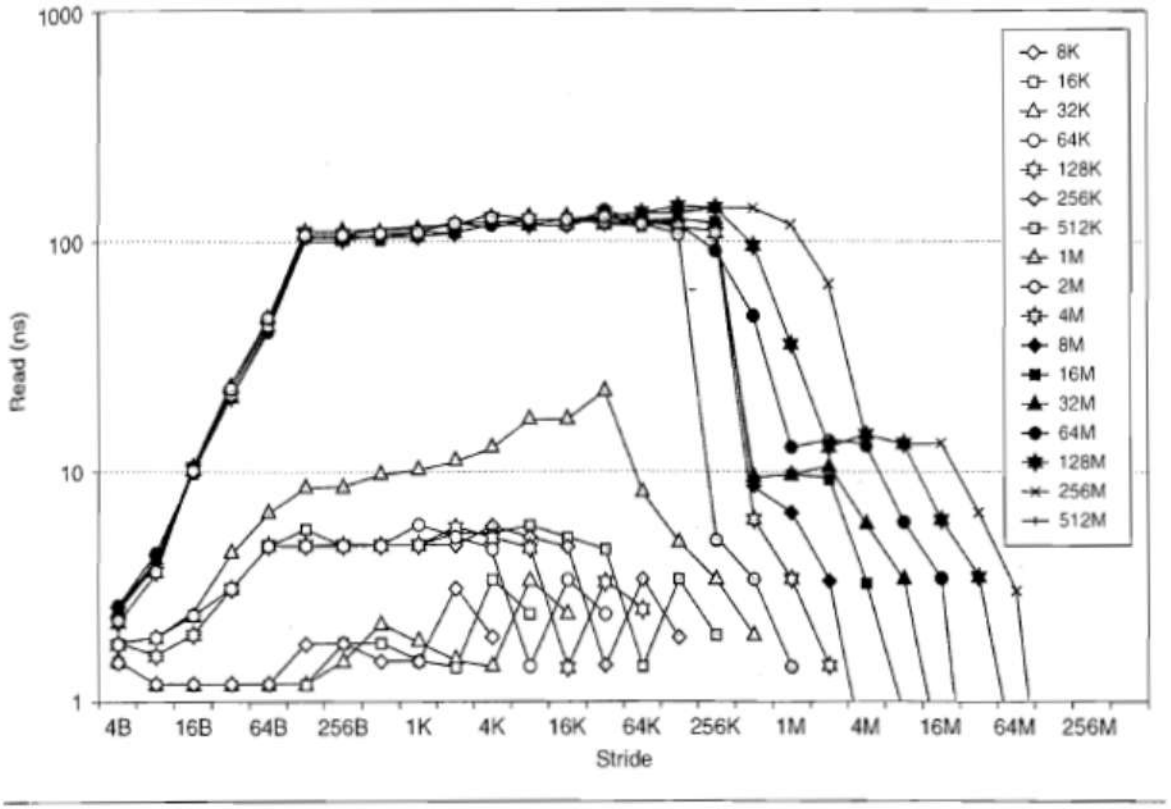


Figure 2.30 Sample results from program in Figure 2.29.

Utilizând exemplu de program prezentat anterior:

a. Care este mărimea totală și mărimea unui bloc pentru memoria cache de pe nivelul 2?
b. Care este penalitatea dată de lipsuri (miss) de pe nivelul 2 al memoriei cache?
c. Care este asociativitatea pe nivelul 2 al memorie cache?
d. Care este mărimea memoriei principale?
e. Care este timpul unei pagini dacă mărimea unei pagini este de 4KB?

5 Dacă este necesar modificați codul de la problema 4 pentru a măsura următoarele caracteristici sistem. Trasați rezultatele experimentale cu timpul consumat pe axa y și pasul de memorie pe axa x. Utilizați scalarea logaritmică pentru ambele axe, și desenați o linie pentru fiecare mărime a cache-ului.

a. Ce este mărimea unei pagini sistem?
b. Cât de multe intrări are buffer-ul de translatare cu anticipare (translation lookaside buffer - TLB)?
c. Ce este penalitate pentru lipsuri pentru TLB?
d. Ce este asociativitatea TLB-ului ?

6 În sistemele multi – microprocesor, nivelurile inferioare ale ierarhiei memoriei pot să nu fie saturate de un singur procesor dar trebuie să fie capabile să fie saturate de procesoare multiple care lucrează împreună. Modificați codul de la problema 4, și rulați copii multiple în același moment. Puteți determina:

a. Câte procesoare actuale sunt în calculatorul vostru și câte procesoare sistem sunt doar contexte adiționale pentru fire multiple de execuție (multithreading)?
b. Câte controlere de memorie nare sistemul vostru?

7 Puteți să vă gândiți la o cale pentru a testa unele caracteristici a unui cache pentru instrucțiuni utilizând un program? Indicație: compilatorul poate genera un număr larg de instrucțiuni care nu sunt evidente pentru un anumit cod. Încercați să utilizați instrucțiuni aritmetice simple de lungime cunoscută din arhitectura setului vostru de instrucțiuni (Instruction Set Architecture -ISA).

8 Această problemă investighează impactul unei memorii cache mici și simple utilizând CACTI și luând în calcul o tehnologie de 65 nm (0,065 μm). (CACTI este disponibil online la http://quid.hpl.hp.com:9081/cacti/.)

a. Comparați timpul de acces a unui cache de 64KB cu blocuri de 64 de octeți și cu un singur banc. Care este timpul relativ de acces al memoriilor cache asociative cu două căi și cu patru căi în raport cu o organizare cu mapare directă?
b. Comparați timpul de acces al unei memorii cache asociative cu patru căi cu blocuri de 64 de octeți și un singur banc. Care este timpul relativ de acces a unor meorii cache de 32 și respectiv 64 de KB în comparație cu un cache de 16KB?
c. Pentru un cache de 64 KB, găsiți asociativitatea între 1 și 8 cu cel mai mic timp de acces mediu la memorie, cunoscând faptul că lipsurile per instrucțiune pentru o anumită suită de încărcări este de 0,00664 pentru maparea directă, 0,00366 pentru asociativitatea cu două căi, 0,000987 pentru asociativitatea cu patru căi, și 0,000266 pentru asociativitatea cu 8 căi. Per total sunt 0,3 referințe per instrucțiune. Presupuneți că lipsa cache ia 10ns. Pentru a calcula potrivirea în cicli, preluați ieșirea CACTI care calculează timpul unui ciclu, care corespunde frecvenței maxime la care poate opera un cache fără bubble (goluri) în pipeline .

9 În această problemă se investighează posibilele beneficii ale unei căi de a prezice (way-predicted) pentru memoria cache L1. Presupuneți că intr-un sistem memoria cache L1 având mărimea de 64 KB organizată asociativ cu 4 căi este cea care limitează timpul unui ciclu mașină. Ca o alternativă la organizarea memoriei cache, trebuie să considerați o memorie cache modelată cu o cale de predicție, având 64KB, organizată cu mapare directă și cu o acuratețe a predicției de 80%. Până când nu se precizează altceva, presupuneți că accesul la calea de predicție care asigură potrivirile (hits) în cache presupune un ciclu mașină în plus. Presupuneți că rata pentru penalitățile de lipsuri (miss) și rata de potrivire (hit) sunt cele din problema 5, întrebarea c.

a. Care este media timpului de acces la memorie pentru cache-ul curent (în cicli) versus cache-ul care are o cale de predicție?
b. Dacă toate celelalte componente pot opera cu memoria cache cu o cale de predicție care este mai rapidă (incluzând memoria principală), care va fi impactul asupra performanțelor atunci când se utilizează memoria cu o cale de predicție?
c. Memoria cache cu cale de predicție se utilizează în mod uzual pentru memoria cache pentru instrucțiuni care utilizează o coadă sau un buffer pentru instrucțiuni. Imaginați-vă că doriți să încercați calea de predicție pentru o memorie cache pentru date. Presupuneți că aveți o acuratețe de 80% și că operația care urmează (de exemplu operații dependente de accesul la cache al altor instrucțiuni) este lansată presupunând o cale de predicție corectă. Astfel, o cale de predicție greșită presupune golirea benzii și o capcană pentru răspuns (replay trap), care necesită 15 cicli. Este pozitivă sau negativă schimbarea timpului mediu de acces la memorie per instrucțiunea de încărcare (load instruction) cu memorie cache de date cu cale de predicție, și cât de mare este schimbarea?
d. Ca o alternativă la calea de predicție, multe memorii cache nivel L2 cu asociativitate mare serializează accesul la etichetă (tag) și dată, astfel încât trebuie activată doar

aria cerută care conține setul de date. Acest fapt micșorează consumul de putere dar crește timpul de acces. Utilizați interfața WEB CACTI pentru: un proces de 0,065 μm, capacitatea memorie cache de 1MB, organizarea memoriei cu mapare asociativă pe seturi cu 4 căi, mărimea unui bloc de 64B, citirea la ieșire a 144biți, un banc, un singur port de citire/scriere, etichete pe 30 de biți, și tehnologia ITRS-HP cu fire globale. Care este raportul între timpul de acces pentru serializarea etichetelor și a datelor, și accesul paralel?

10 Ați fost rugat să investigați pentru un procesor nou performanțele relative a unei memorii cache de date L1 organizată pe bancuri sau sub formă de bandă de asamblare (pipelined). Presupuneți o memorie cache cu mărimea de 64KB, cu mapare asociativă pe seturi cu două căi având mărimea unui bloc de 64 de octeți (64O-octeți sau 64B - byte). Banda de asamblare pentru cache constă din trei etaje, similar în capacitate cu memorie cache de date de la Alpha 21264. O implementare pe bancuri va consta din două bancuri de 32KB cu mapare asociativă pe seturi cu două căi. Pentru a răspunde la următoarele întrebări utilizați CACTI și presupuneți utilizarea unei tehnologii pe 0,065 μm. Durata unui ciclu în versiunea WEB arată la ce frecvență poate lucra o memorie cache fără bule (bubble) în banda de asamblare.
   a. Care este durata unui ciclu pentru memoria cache în comparație cu timpul său de acces, și câte etaje pipeline va avea cache-ul (până la două zecimale) ?
   b. Comparați aria și energia dinamică totală pentru citire per proiectarea de tip pipelined versus proiectarea utilizând bancuri. Precizați care va necesita o arie mai mică și care va necesita putere mai multă, și explicați de ce.

11 Considerați pentru lipsurile memoriei cache de nivel 2 (L2) mecanismele denumite cuvântul critic primul (critica word first)[2] și repornirea timpurie (early restart)[3]. Presupuneți o memorie cache L2 de 1MB, cu blocuri de 64 de octeți și o cale de reumplere (refill) de 16 octeți. Presupuneți că L2 poate fi scrisă cu 16 octeți la fiecare 4 cicli procesor, că timpul de recepție a primului bloc de 16 octeți din controlerul de memorie este de 120 de cicli, că fiecare bloc adițional de 16 octeți din memorie principală necesită 16 cicli, și că data poate fi direct trimisă (direct bypass) la portul de citire al memoriei cache L2. Ignorați orice ciclu de transfer a unei cereri de tip lipsă (miss) la L2 și datele cerute pentru memoria cache L1.
   a. Câți cicli vor fi necesari pentru o lipsă L2 cu și fără mecanismele denumite cuvântul critic primul (critica word first) și repornirea timpurie (early restart)?
   b. Credeți că mecanismele denumite cuvântul critic primul (critica word first) și repornirea timpurie (early restart) vor fi mult mai importante pentru memorie cache L1 sau pentru L2, și care sunt factorii care contribui la importanța lor relativă?

12 Proiectați un buffer de scriere între memoria cache L1 cu scrierea de tip scriere imediată (write through) și memoria cache L2 cu scrierea de tip scriere întârziată (write-back). Magistrala de date pentru L2 este de 16B și poate realiza o scriere la o adresă cache independentă în 4 cicli procesor.
   a. Câți octeți trebuie să aibă fiecare intrare în buffer-ul de scriere?

---

[2] Cuvântul critic primul (critica word first) – cere prima dată de la memorie cuvântul lipsă și trimite-l către UCP imediat ce sosește; lasă UCP să-și continue execuția în timp ce se umple restul de cuvinte din bloc. Se mai numește extragere adaptată (wrapped fetch)

[3] Repornire timpurie (early restart) – cât de curând (imediat ce) sosește cuvântul cerut din bloc, aceste este trimis de urgență la UCP (procesor) lăsundu-l astfel pe acesta să-și continue execuția.

"

b. La ce creștere de viteză ne putem aștepta în starea de echilibru utilizând un buffer de scriere cu fuzionare(merging write buffer)[4] în loc de un buffer de scriere fără fuzionare atunci când se filează cu 0 memorie prin execuția unei memorări pe 64 de biți dacă toate celelalte instrucțiuni pot fi lansate în paralel cu memorarea și blocurile sunt prezente în cache-ul L2.
c. Care poate fi efectul unei posibile lipse (miss) la nivelul memoriei cache L1 relativ la numărul de intrări cerute de la buffer-ul de scriere, pentru sisteme cu memorie cache cu blocare și fără blocare?

13 Considerați un sistem de tip desktop cu un procesor conectat la un RAM de 2GB cu cod de corectare a erorilor (error-corecting code - ECC). Presupuneți că există un singur canal de memorie cu 72 de biți, dintre care 64 sunt pentru date și 6 pentru ECC.
a. Câte chipuri DRAM are DIMM dacă sunt utilizate chipuri DRAM de 1 GB, și câte operații I/O trebuie executate pentru fiecare DRAM dacaă doar un DRAM este conectat la fiecare pin de date al DIMM-ului?
b. Ce lungime de salvă (burst) este necesară pentru a suporta blocuri de memorie cache L2 de mărime 32B.
c. Calculați vârful de lărgime de bandă pentru DIMM-urile DDR2-667 și DDR2-533 pentru citiri dintr-o pagină activă excluzând supracontrolul dat de ECC.

14 În figura 2/31 este prezentat un exemplu de diagramă de timp pentru DDR2 SDRAM.

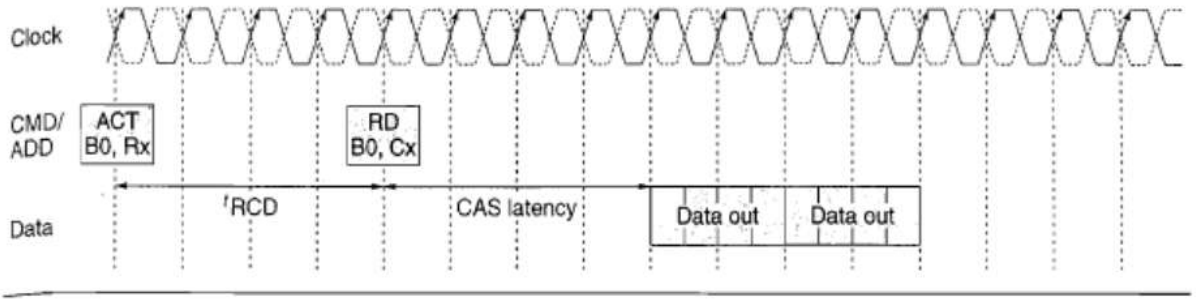


Figure 2.31 DDR2 SDRAM timing diagram.

tRCD este timpul necesar pentru a activa un rând într-un banc, și latența (CL) pentru strobul de adresă pentru coloană (CAS) este numărul de cicli necesari pentru a citi la ieșire o coloană din rând. Presupuneți că RAM-ul este pe DIMM-ul DDR2 cu ECC standard, având 72 de linii de date. Presupuneți de asemenea că lungimea unei salve de 8 citește la ieșire 8 biți cea ce înseamnă un total de 64B de la DIMM. Presupuneți tRCD = CAS (or CL) * clock_frecuency, și că clock_frecuency = transfer_per_seconds/2. Latența pe chip la o lipsă cache de la nivelul 1 la nivelul 2 și înapoi, nu include accesul la DRAM, în cele 20ns.
a. Cât timp este necesar de la prezentarea unei comenzi active până la ultimul bit de date cerut de la tranziția DRAM de la valid la invalid pentru DIMM-ul DDR2-667 1GB cu CL = 5. Presupuneți că pentru fiecare cerere se va prextrage automat o altă linie cache adiacentă din aceeași pagină.
b. Care este latența relativă, atunci când se utilizează DIM-ul DDR2-667, a unei cereri de citire a unui banc activ versus una pentru o pagină deja deschisă, incluzând și timpul necesar pentru a procesa o lipsă în interiorul procesorului?

[4] Bufferul de scriere cu fuzionare (Merging write buffer) combină scrierile care au ca destinație adrese consecutive într-o singură intrare în buffer. Altfel, acestea vor ocupa intrări separate care vor crește șansa de oprire a benzii de asamblare a procesorului.

15 Presupuneți că pentru 130$ de dolari puteți achiziționa un DIMM de tip DDR2-667 de 2 GB cu CL=5 și la 100$ un DIMM DDR2-533 de 2 GB cu un CL =4. Presupuneți că cele două DIMM-uri sunt utilizate în sistem, și restul sistemului costă 800$. Considerați performanțele sistemului utilizând DIMM-urile DDR2-667 și DDR2-533 în condițiile unei execuții cu 3,33 lipsuri la 1K de instrucțiuni, și presupuneți că 80% din toate citirile din DRAM necesită o activare. Presupunând că este restantă la un moment o lipsă în L2 şi că avem un nucleu care lucrează în ordine (in-order) cu un CPI de 1,5 fără a include timpul de acces dat de lipsurile L2, care este cost-performanța întregului sistem când se utilizează DIMM-uri diferite?

16 Ați achiziționat un server cu 8 nuclee cu frecvența de 8GHz CMO, care poate executa o încărcare cu un CPI de 2 (presupunând că nu sunt întârziate lipsurile de reumplere la memoria cache L2). Linia cache L2 are 32 de octeți. Presupuneți că sistemul utilizează DIMM-uri DDR2-667, câte canale de memorie independente de memorie trebuie furnizate astfel ca sistemul să nu fie limitat de lărgimea de bandă a memoriei dacă lărgimea de bandă necesară este uneori de două ori mai mare ca media? Încărcarea implică, per media 6,67 lipsuri L2 la 1K instrucțiuni.

17 Activarea unei pagini poate necesita o cantitate mare de energie (mai mult de trei ori) pentru DRAM (vezi http://download.micron.com/pdf/technotes/ddr2/TN4704.pdf și www.micron.com/systemcalc) .Presupuneți că veți construi un sistem cu 2GB de memorie utilizând fie 8 bancuri de tip 2Gb x 8 DDR2 DRAM sau 16 bancuri de 1 Gb x 16 DRAM, ambele cu aceeași viteză. Ambele utilizează pagini de 1KB și ultimul nivel pentru liniile cache au 64B. Presupuneți că DRAM-urile care nu sunt active sunt în așteptare cu consum redus pentru preîncărcare (precharged standby) și disipă o putere neglijabilă. Presupuneți că timpul de tranziție din starea de așteptare cu consum redus în starea activă este nesemnificativ.

a. De la care tip de DRAM se va aștepta să furnizeze performanțe sistem mai mari? Explicați de ce?
b. Din punctul de vedere al puterii, cum se compară un DIMM de 2 GB care se construiește cu un DRAM de 1GB x 8 DDR2 cu un DIMM de capacitate similară care se construiește cu 1Gb x 4 DDR2?

18 Pentru a accesa date de la un DRAM tipic, trebuie să activăm prima dată rândul corespunzător. Presupuneți ca această activare aduce o întreagă pagină de mărime 8KB în bufferr-ul pentru rânduri. Apoi, selectăm o coloană din buffer-ul pentru rânduri. Dacă accesele imediat următoare la DRAM sunt la aceeași pagină, atunci se poate sări pasul de activare; altfel trebuie să închidem pagina curentă și să pre-încărcăm linia de bit pentru următoarea activare. O altă politică populară pentru DRAM este de a închide pro-activ pagina și de a pre-încărca linia de bit imediat ce apare un acces. Presupuneți că fiecare citire sau scriere din/ în DRAMeste de 64B și latența DDR (Data aut în figura 2.31) pentru a trimite 512 biți este Tddr.

a. Presupunând că DDR2-667 ia 5 cicli pentru preîncărcare, cinci cili pentru activare, și patru cicli pentru a citi o coloană, pentru ce valoare a fracției de potrivire la bufferul de rând (r) alegeți o politică în loc de cealaltă pentru a obține cel mai bun timp de acces? Presupuneți că fiecare acces la DRAM este separat de timp suficient pentru a termina un nou acces aleatoriu.
b. Dacă 10% din accesul total la DRAM se întâmplă unul după altul sau continuu fără nici un interval de timp pierdut, cum se va schimba decizia?
c. Calculați diferența mediei energiei consumate de DRAM per acces între cele două politici utilizând fracția de potrivire pentru buffer-ul de rând calculată anterior.

Presupuneți că preîncărcarea necesită 2 nJ, activarea 4 nJ și 100pJ sunt necesari pentru a citi sau scrie în buffer-ul de rând.

19 Ori de câte ori un calculator este în așteptare (idle), fie îl putem trece în starea cu consum redus (stanby), când DRAM încă este activ, sau îl putem lăsa să hiberneze. Presupuneți că, pentru hibernare, trebuie doar să copiem conținutul DRAM-ului pe un mediu nevolatil precum un Flash. Dacă citire sau scrierea unei linii Flash de 64 B necesită 2,56 μJ, iar DRAM-ul necesită 0,5 nJ, iar dacă consumul în idle al DRAM-ului este de 1,6W (pentru 8 GB), cât timp trebuie un sistem trecut în idle pentru a beneficia de hibernare? Presupuneți că memoria principală are 8GB.

20 Mașinile virtuale (VM) au potențial pentru a aduce multe facilități benefice unui sistem de calcul, precum îmbunătățirea costului total al utilizării (TCO) sau a disponibilității. Pot fi VM-urile utilizate pentru a furniza următoarele facilități? Dacă da, cum pot acestea oferii aceste facilități?

a. Testarea aplicațiilor în mediile de producție utilizând sisteme de dezvoltare?
b. Repornirea rapidă a aplicațiilor în caz de dezastru sau cădere?
c. Performanțe înalte în aplicații cu operații intensive de I/O?
d. Izolarea defectelor între diferite aplicații, rezultând o disponibilitate crescută pentru servicii?
e. Permite mentenanța software-ului pe sisteme în timp ce aplicațiile rulează fără întreruperi semnificative?

21 Mașinile virtuale pot pierde performanța datorită unor evenimente cum ar fi: execuția instrucțiunilor privilegiate, lipsuri în TLB, capcane și operații I/O. Aceste evenimente sunt uzual gestionate de codul sistem. Astfel, o cale de a estima încetinirea atunci când se lucrează pe VM-uri este procentul de timp când aplicația se execută în modul sistem versus modul utilizator. De exemplu, o aplicația care petrece 10% din execuția sa în modul sistem poate încetinii cu 60% când se execută pe o VM, Figura 2.32 listează performanțele timpurii pentru diferitele apeluri de sistem sun execuție nativă, virtualizare pură și para virtualizare pentru LMbench utilizând sisteme Xen pe Itanium cu măsurarea timpului în microsecunde.

Benchmark	Native	Pure	Para
Null call	0.04	0.96	0.50
Null I/O	0.27	6.32	2.91
Stat	1.10	10.69	4.14
Open/close	1.99	20.43	7.71
Install sighandler	0.33	7.34	2.89
Handle signal	1.69	19.26	2.36
Fork	56.00	513.00	164.00
Exec	316.00	2084.00	578.00
Fork + exec sh	1451.00	7790.00	2360.00

Figure 2.32 Early performance of various system calls under native execution, pure virtualization, and paravirtualization.

a. La ce tipuri de programe vă așteptați să aibă încetiniri mai mici când aceste rulează sub VM?

b. Dacă încetinirea este liniară ca funcție de timpul sistem, fiind dată încetinirea de la punctul anterior, cât de mult încetinește un program ce petrece 20% din timpul său de execuție în modul sistem ?
c. Care este încetinirea mediană a apelurilor sistem din tabel sub virtualizarea pură și paravirtualizare?
d. Care funcție din tabel are cea mai mare încetinire? Care credeți că poate fi cauza acesteia?

22 Popek și Goldberg au definit mașina virtuală spunând că nu poate fi distinsă față de o mașină reală decât prin performanțele sale. În această întrebare, se va utiliza acea definiție care să ne permită să știm dacă avem acces la execuția nativă pe un procesor sau execuția este pe o mașină virtuală. Tehnologia Intel VT-x oferă efectiv un al doilea set de niveluri privilegiate pentru a fi utilizate de mașinile virtuale. Ce trebuie să facă o mașină virtuală ce rulează pe o altă mașină virtuală, dacă se utilizează tehnologia VT-x?

23 O dată cu adoptarea suportului pentru virtualizare pe arhitecturile x86, mașinile virtuale au evoluat în mod activ și au devenit "de masă". Comparați și scoateți în evidență diferențele între tehnologiile Intel VT-x și AMD – V. (Informațiile despre AMD-V pot fi găsite la http://sites.amd.com/us/business/it-solutions/virtualization/Pages/resources.aspx.)
a. Care dintre ele poate furniza performanțe superioare pentru aplicații cu acces intensiv la memorie în cazul unor memorii mari?
b. Informațiile despre AMD IOMMU pot fi găsite la http://devoloper.amd.com/us/documentation/articles/pages/892006101.aspx.) Ce face tehnologia de virtualizare și unitatea de gestiune a memoriei (IOMMU) pentru a îmbunătății performanțele de virtualizare a operațiilor I/O?

24 Deoarece paralelismul la nivelul instrucțiunilor poate fi poate fi efectiv exploatat atât în cazul procesoarelor superscalare cu execuție în-ordine, cât și pentru procesoarele de tip VLIW (Very large Instruction Word – cu instrucțiuni cu cuvânt foarte mare) cu execuție speculativă, un motiv foarte important pentru construirea unui procesor cu execuție în afara ordinii (Out-of-Order - OOO) este acela de a avea abilitatea de a tolera întârzierea nepredictibilă a memorie cauzată de lipsurile cache. Ca urmare, puteți gândii că hardware-ul care suportă OOO este parte a sistemului de memorie! Priviți amprenta pe siliciu a procesorului Apha 21264 din figura 2.33 pentru a găsi aria relativă a cozilor și a dispozitivelor de mapare pentru întregi și virgulă flotantă versus memoriile cache. Cozile planifică lansarea instrucțiunilor în execuție, iar dispozitivele de mapare redenumesc specificatorii de regiștrii. Ca urmare acestea sunt elemente adiționale pentru a suporta execuția de tip OOO. Procesorul 21264 are pe chip doar cache de nivel L1 pentru date și instrucțiuni, ambele având mărimea de 64KB cu mapare asociativă pe seturi cu două căi. Utilizați un simulator superscalar precum SimpleScalar (www.cs.wisc.edu/"mscalar/simplescalar.html) cu un test (benchmark) intensiv pe accese la memorie pentru a găsii câtă performanță se pierde dacă aria pentru cozi și dispozitive de mapare este utilizată pentru memoria de date cache L1 într-un procesor superscalar cu execuția în ordine, în loc să fie utilizată pentru execuția OOO așa cum este modelată această în 21264. Asigurați-vă că celelalte aspecte ale procesoarelor sunt cât mai similare posibil pentru a avea o comparație cinstită. Ignorați orice creștere în acces sau timpul unui ciclu dată de memoriile cache mari, și efectul memoriilor cache de date mari asupra repartiției pe zone a chipului (de notat că această comparație nu va fi în totalitate corectă, deoarece codul nu va fi planificat de compilator pentru un procesor cu execuție în ordine).

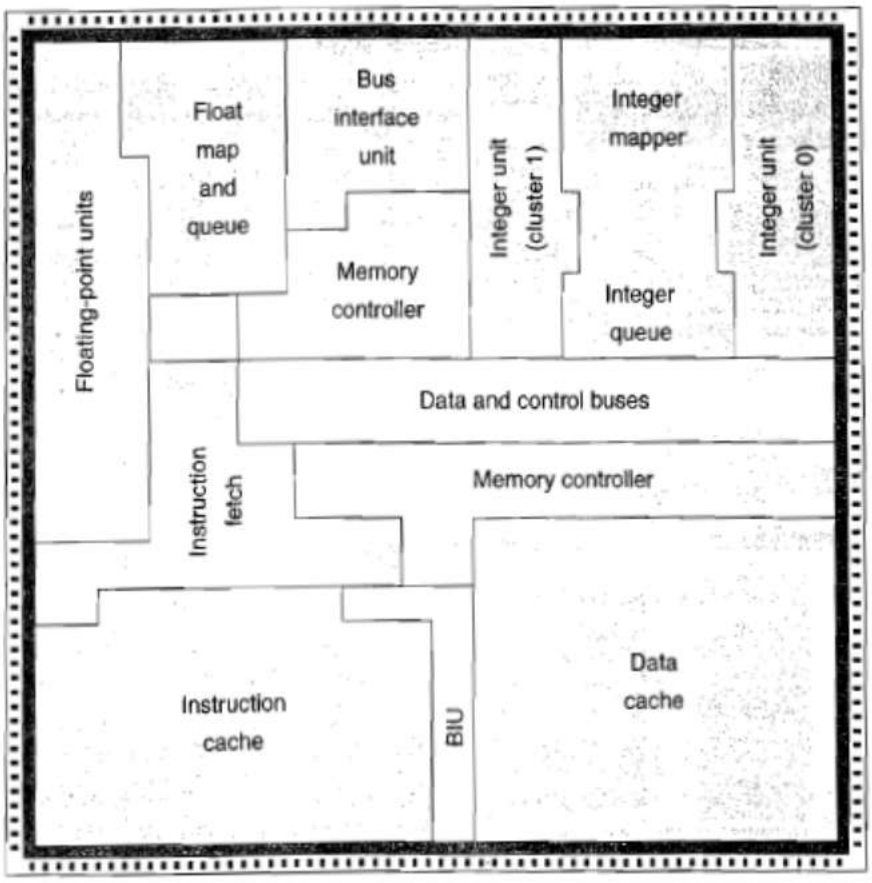


Figure 2.33 Floorplan of the Alpha 21264 [Kessler 1999].

25 Analizorul Intel VTune pentru performanțe poate fi utilizat pentru a realiza multe măsurători privind performanțele memorie cache. O versiune de evaluare liberă atât pentru Windows cât și pentru Linux poate fi descărcată de la http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/. Programul (aca_ch2_cs2.c) utilizat în studiul 2 de caz a fost modificat astfel încât să poată lucra cu ieșirea din VTune din Microsoft Visual C++. Programul poate fi descărcat de la www.hpl.hp.com/researc/cacti/aca_ch2_cs2_vtune.c. Au fost înserate funcții VTune speciale pentru a exclude supracontrolul dat de inițializare și bucle pe durata procesului de analizare a proceselor. Direcțiile de setare detaliate ale VTune sunt date în secțiunea README din program. Programul păstrează bucla pentru 20 de secunde pentru fiecare configurație. În experimentul care urmează puteți găsii efectul mărimii datelor în cache și performanțele generale ale procesorului. Rulați programul VTune pe un procesor Intel cu mărimea datelor de intrare de 8KB, 128KB, 4MB și 32MB, și păstrați un pas de 64 de octeți (pasul unei linii cache pe procesoarele i7). Colectați statisticile privind performanțele generale și performanțele memoriilor cache de date L1, L2 și L3.

a. Listați numărul de lipsuri per 1K instrucțiuni în cache-ul de date L1, L2 și L3 pentru fiecare mărime a setului de date pentru viteza și modelul procesorului vostru. Pe baza acestor rezultate, ce puteți spune despre mărimea memoriilor cache L1, L2 și L3 de pe procesorul vostru? Explicați observațiile.
b. Listați instrucțiunile per ceas (instructions per clock - IPC) pentru fiecare mărime a setului de date, modelul vostru de procesor și viteza acestuia. Pe baza acestor rezultate, ce puteți spune despre penalitățile referitoare la lipsurile memoriilor cache L1, L2 și L3 de pe procesorul vostru?

c. Rulați programul VTune cu mărimi ale setului de date de 8KB și 128KB pe un procesor Intel cu OOO. Pentru ambele configurații listați numărul de lipsuri cache L1 și L2 per 1K instrucțiuni și CPI-ul. Ce puteți spune despre eficacitatea tennicilor de ascundere a latenței memoriei în procesoarele OOO de mare performanță? Sfat: trebuie să găsiți întârzierea dată de latența de la nivelul memorie cache L1 pentru procesorul vostru. Pentru i7 recente aceasta este de 11 cicli.

Probleme propuse pentru examenul de la disciplina Structura şi Organizarea Calculatoarelor

== MOODLE TYPE ==

Prof. dr. ing. Vasile Gheorghiță GĂITAN

Ş. l. dr. ing. Ionel ZAGAN

Cap. 5. Instruction-Level Parallelism and Its Exploitation

1. Care vor fi performanţele de referinţă (în cicli, per o iteraţie a buclei) a secvenţei de cod din figura 3.48 dacă nu se poate iniţia nici o instrucţiune nouă până când nu s-a completat execuţia vechii instrucţiuni. Ignoraţi extragerea şi decodificarea instrucţiunii. Presupuneţi acum că execuţia nu este încetinită dacă lipseşte următoarea instrucţiune, dar pot fi lansată doar o instrucţiune/ciclu. Presupuneţi că saltul este luat, şi că există un slot de întârziere pentru ramificaţii.

			Latencies beyond single cycle	
Loop:	LD	F2,0(RX)	Memory LD	+4
I0:	DIVD	F8,F2,F0	Memory SD	+1
I1:	MULTD	F2,F6,F2	Integer ADD, SUB	+0
I2:	LD	F4,0(Ry)	Branches	+1
I3:	ADDD	F4,F0,F4	ADDD	+1
I4:	ADDD	F10,F8,F2	MULTD	+5
I5:	ADDI	Rx,Rx,#8	DIVD	+12
I6:	ADDI	Ry,Ry,#8		
I7:	SD	F4,0(Ry)		
I8:	SUB	R20,R4,Rx		
I9:	BNZ	R20,Loop		

Figure 3.48 Code and latencies for Exercises 3.1 through 3.6.

2. Gândiţi-vă ce înseamnă în realitate numerele date pentru latenţă – ele indică numărul de cicli pe care o funcţie dată îi cere pentru a produce ieşirea şi nimic mai mult. Dacă se cunosc la nivelul global al liniei piepeline cicli de latenţă pentru fiecare unitate funcţională, atunci cel puţin se poate garanta că orice pereche de instrucţiuni una după alta (un producător urmat de un consumator) se va executa corect. Dar, nu toate perechile de instrucţiuni au o relaţie producător/consumator. Uneori două instrucţiuni adiacente nu au nimic de a face una cu alta. Câţi cicli va necesita corpul buclei din secvenţa de cod din figura 2.48 dacă linia pipeline detectează dependenţele reale de date şi produce încetinirea pentru acestea, mai degrabă decât să încetinească orbeşte orice doar pentru că o unitate funcţională este ocupată? Prezentaşi codul cu <stall> înserat acolo unde este necesar pentru a realiza latenţele stabilite.

3. Luaţi în considerare o proiectare cu lansări multiple în execuţie. Presupuneţi că aveţi două linii pipeline pentru execuţie, fiecare fiind capabilă să înceapă execuţia unei instrucţiuni per ciclu şi, în faţă, o lărgime de bandă suficientă pentru fetch/decode astfel încât să nu fie încetinită execuţia. Presupuneţi că rezultatul poate fi imediat avansat de la o unitate de execuţie la alta, sau de la ea însăşi. Mai departe presupuneţi că singurul motiv pentru care pipeline-ul încetineşte este acela de a observa o dependenţă adevărată de date. Ca urmare câţi cicli necesită bucla?

4. Pentru proiectarea cu lansări multiple în execuţie de la exerciţiul 3, se pot recunoaşte unele subtilităţi. Chiar dacă cele două pipeline au exact acelaşi repertoriu, ele nu sunt nici identice, nici interschimbabile, deoarece există o ordine implicită între ele care trebuie să reflecte ordinea instrucţiunilor din programul original. Dacă instrucţiunea N+1

îşi începe execuţia în Pipe 1 în acelaşi timp cu instrucţiunea N din Pipe 0, şi dacă N+1 necesită o latenţă în execuţie mai mică decât N, atunci N+1 se va completa înainte de N (deşi ordinea programului necesită completarea inversă). Descrieţi cel puţin două motive de ce această situaţie poate fi hazard şi necesită o considerare specială în micro-arhitectură. Daţi un exemplu de două instrucţiuni din codul din figura 3.48 care demonstrează acest hazard.

5. Reordonaţi instrucţiunile pentru a îmbunătăţii performanţele codului din figura 3.48. Presupuneţi o maşină cu două pipeline-uri ca aceea din exerciţiul 3 şi că execuţia în afara ordinii din exerciţiul 4 a fost distribuită cu succes. Fiţi doar îngrijoraţi pe moment doar de a observa dependenţele reale de date şi latenţele unităţilor funcţionale. Câţi cicli ia codul reordonat?

6. Fiecare ciclu care nu iniţiază o nouă operaţie în pipeline este o oportunitate pierdută, în sensul în care hardware-ul nu funcţionează la întreg potenţialul său.

   a) Ce fracţie din toţi cicli codului reorganizat la exerciţiul 5, luând în considerare ambele pipeline-uri, a fost pierdută?
   b) Desfăşurarea buclelor este una dintre tehnicile standard ale compilatoarelor pentru a găsi mai mult paralelelism la nivelul codului, cu scopul de a minimiza pierderile privind oportunităţile pentru performanţă. Desfăşuraţi manual două iteraţii ale buclei din codul reordonat la execrciţiul 5.
   c) Ce creştere de viteză aţi obţinut (pentru acest exerciţiu doar coloraţi cu verde instrucţiunea iteraţiei a (N+1) –a, pentru a o distinge de iteraţia a N-aş Dacă desfăşuraţi bucla trebuie să reasignaţi regiştrii pentru a preveni coliziunile între iteraţii).

7. Calculatoarele petrec o parte însemnată din timpul lor în bucle, astfel buclele cu iteraţii multiple sunt locurile cele mai adecvate ude se poate găsi execuţie speculativă care să ţină ocupate resursele CPU. Nimic nu este vreodată uşor; compilatorul emite doar o singură copie a codului buclei, astfel chiar dacă iteraţiile multiple gestionează date distincte, ele apar ca utilizând aceeaşi regiştrii. Pentru a evita coliziunile datorate utilizării regiştrilor de către iteraţiile multiple, se vor redenumii regiştrii. Figura 4.49 prezintă un exemplu de cod care pe dorim să o redenumească hardware-ul. Compilatorul poate să desfăşoare simplu bucla şi să utilizeze regiştrii diferiţi pentru a evita conflictele, dar dacă ne aşteptăm ca hardware-ul nostru se desfăşoare bucle, şi acesta trebuie să facă redenumirea regiştrilor. Cum? Presupuneţi că hardware-ul nostru are o arie de regitrii temporari (denumiţi registrii T, şi presupuneţi că 64 dintre ei, T0 la T63) care pot substitui regiştri desemnaţi de compilator. Acest hardware de redenumire este indexat de denumirea src (sursă) a registrului, şi valoarea în tabel este registrul T al ultimei destinaţii care avea ca ţintă acel registru. (Gândiţi-vă la aceste valori din tabel ca la producători, iar regiştrii src sunt consumatorii; nu contează pre mult unde pun producătorii rezultatelor lor atât timp cât consumatorii le pot găsi). Consideraţi secvenţa de cod din figura 3.49. De fiecare dată când vedeţi în cod un registru destinaţie, substituiţi-l cu următorul T disponibil, începând cu T9. Apoi actualizaţi corespunzător toţi registrii src, astfel încât dependenţele reale de date să fie menţinute. Prezentaţi codul rezultat.

```
Loop: LD      F4,0(Rx)
I0:   MULTD   F2,F0,F2
I1:   DIVD    F8,F4,F2
I2:   LD      F4,0(Ry)
I3:   ADDD    F6,F0,F4
I4:   SUBD    F8,F8,F6
I5:   SD      F8,0(Ry)
```

Figure 3.49 Sample code for register renaming practice.

```
I0:   LD      T9,0(Rx)
I1:   MULTD   T10,F0,T9
...
```

Figure 3.50 *Hint:* Expected output of register renaming.

8. Exerciţiul 3.7 a explorat redenumirea simplă a regiştrilor: atunci când un bloc hardware de redenumire a regiştrilor vede un registru sursă, substituie regsitrul destinaţie T al ultimei instrucţiuni care avea ca ţintă acel registru sursă. Atunci când un tabel de redenumire vede un registru destinaţie, acesta substituie pentru acesta următorul T disponibil, dar proiectarea superscalarilor trebuie să gestioneze instrucţiuni multiple pe un singur ceas la nivelul fiecărui etaj, incluzând cel pentru redenumirea regiştrilor. Prin urmare, un procesor scalar simplu trebuie să caute atât maparea pentru src pentru fiecare instrucţiune şi să aloce o nouă mapare destinaţie per ciclu de ceas. Procesoare superscalare trebuie să fie capabile să facă acelaşi lucru, dar trebuie să asigure că orice relaţie dest-to-src între două instrucţiuni concurente este gestionată corect. Consideraţi exemplu de cod din figura 3.51. Presupuneţi că dorim să redenumim simultan primele două instrucţiuni. Mai mult presupuneţi că următorii doi T regiştrii disponibili pentru a fi utilizaţi sunt cunoscuţi la începutul ciclului de ceas în care cele două instrucţiuni vor fi redenunite. Conceptual, ceea ce dorim este ca pentru prima instrucţiune să se facă căutarea în tabelul de redenumire şi apoi să realizăm actualizarea tabelului destinaţie cu registrul T. Apoi a doua instrucţiune va face exact acelaşi lucru, şi orice dependenţă între instrucţiuni va fi gestionată corect. Dar nu este suficient timp pentru a scrie registrul T desemnat în tabelul de redenumire şi apoi să căutăm din nou pentru a doua instrucţiune totul în acelaşi ciclu de ceas. Ca urmare substituţia regiştrilor trebuie realizată imediat (în paralel cu actualizarea tabelului cu redenumirea regiştrilor). Figura 3.52 prezintă diagrama circuitului, care utilizează multiplexoare şi comparatoare, care vor asigura redenumirea “din zbor”. Sarcina voastră este să prezentaţi starea ciclu cu ciclu a tabelului de redenumire pentru fiecare instrucţiune din codul din figura 3.51. Presupuneţi că tabelul porneşte cu fiecare intrare egală cu indexul său (T0=0, T1 = 1, …)

```
I0:   SUBD    F1,F2,F3
I1:   ADDD    F4,F1,F2
I2:   MULTD   F6,F4,F1
I3:   DIVD    F0,F2,F6
```

Figure 3.51 Sample code for superscalar register renaming.

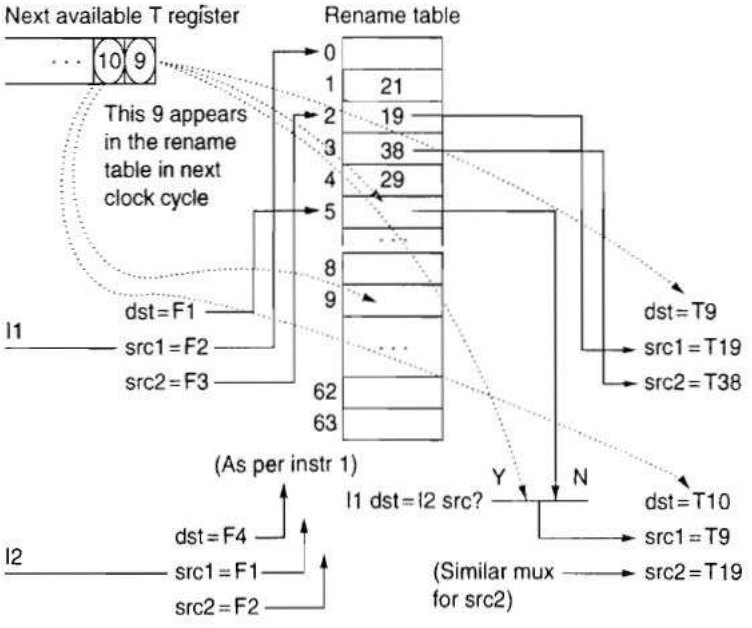


Figure 3.52 Rename table and on-the-fly register substitution logic for superscalar machines. (Note that src is source, and dest is destination.)

9. Dacă nu înţelegeţi ce face redenumirea regiştrilor, mergeţi înapoi la codul în asamblare pe care l-aţi executat, şi întrebaţi-vă ce urmează să se întâmple pentru a obţine rezultatul corect. De exemplu, consideraţi un bloc de redenumire pentru un superscalar cu trei căi care să redenumească concurent aceste trei instrucţiuni.

   ADDI R1,R1,R1
   ADDI R1,R1,R1
   ADDI R1,R1,R1

   Dacă valoarea din R1 este 5, care va fi valoarea atunci când va fi executată această instrucțiune?

10. Proiectanții pentru Very Long Instruction Word - VLIW au câteva opţiuni de bază pentru a face reguli arhitecturale de utilizare a regiştrilor. Presupuneţi că un VLIW este proiectat cu pipeline-uri cu auto-drenare: odată ce este iniţiată o operaţie, rezultatul acesteia va apărea în registrul destinaţie la cel mult L cicli mai târziu (unde L este latenţa operaţiei). Nu există niciodată suficienţi regiştrii, ca urmare există tentaţia de a stoarce maximum din regiştrii care există. Consideraţi figura 3.53. Dacă load are latebţa de 1 + 2 cicli, desfăşuraţi această buclă o dată şi arătaţi cum un VLIW capabil de a executa două load şi două adunări per ciclu poate utiliza numărul minim de regiştrii, în absenţa oricărei întreruperi sau încetiniri a pipeline-ului. Daţi un exemplu de

un eveniment care, în prezența pipeline-ului cu autodrenare, poate deranja pipeline-ul și conduce la rezultate eronate.

```
Loop: LW      R4,0(R0) ;  ADDI    R11,R3,#1
      LW      R5,8(R1) ;  ADDI    R20,R0,#1
      <stall>
      ADDI    R10,R4,#1;
      SW      R7,0(R6) ;  SW      R9,8(R8)
      ADDI    R2,R2,#8
      SUB     R4,R3,R2
      BNZ     R4,Loop
```

Figure 3.53 Sample VLIW code with two adds, two loads, and two stalls.

11. Presupuneți o micro-arhitectură cu un singur pipeline și cinci etaje (fetch, decode execute, memory, and write-back) și codul din figura 3.54. Toate operațiile sunt de 1 ciclu cu excepția LW și SW care sunt de 1 + 2 cicli, și ramificațiile care sunt de 1 + 1 ciclu. Nu există avansare. Prezentați fazele pentru fiecare instrucțiune per ciclu de ceas pentru o iterație în buclă.

```
Loop:  LW      R3,0(R0)
       LW      R1,0(R3)
       ADDI    R1,R1,#1
       SUB     R4,R3,R2
       SW      R1,0(R3)
       BNZ     R4, Loop
```

Figure 3.54 Code loop for Exercise 3.11.

a) Câți cicli de ceas per o iterație în buclă sunt pierduți datorită supracontrolului introdus de ramificație?
b) Presupuneți prezența unui predictor static capabil să recunoască salturile înapoi în etajul Decode. Acum, câți cicli de ceas sunt pierduți de supracontrolul dat de ramificații?
c) Presupuneți prezența unui predictor dinamic. Câți cicli sunt pierduți la o predicție corectă?
d)

12. Să considerăm ce poate realiza planificarea dinamică. Presupunem că avem arhitectura din figura 3.55. Presupuneți că ALU poate face toate operațiile aritmetice (MULTD, DIVD, ADDD, ADDI, SUB) și ramificațiile, și că stația de rezervare RS poate dispeceriza cel puțin câte o operație la fiecare unitate funcțională per ciclu (o operație la fiecare ALU plus o operație cu memorie la LD/ST).

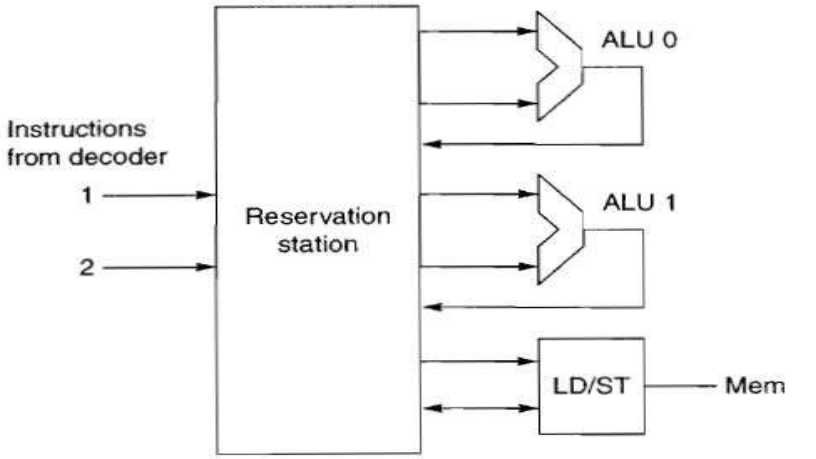


Figure 3.55 An out-of-order microarchitecure.

a) Presupuneţi că toate instrucţiunile din secvenţa din figura 3.48 sunt prezente în RS, fără execuţia redenumirii. Marcaţi oricare instrucţiune din cod pentru care redenumirea registrului conduce la îmbunătăţirea performanţelor. Presupuneţi aceleaşi latenţe pentru unităţile funcţionale ca în figura 3.48.
b) Presupuneţi că versiunea cu redenumirea regiştrilor de la punctul a este rezidentă în RS în ciclul N, cu latenţele date în figura 3.48. Arătaţi cum trebuie RS să dispecerizeze aceste instrucţiuni în afara ordinii, ceas cu ceas, pentru a obţine performanţe optimale pentru acest cod (presupuneţi aceleaşi restricţii pentru RS ca la punctul a, presupuneţi că rezultatele trebuie scrise în RS înainte ca acestea să fie disponibile pentru utilizare – fără bypassing). Câţi cicli de ceas ia execuţia acestei secvenţe de cod?
c) Punctul b lasă staţiei de rezervare RS să încerce să planifice optimal aceste instrucţiuni. Dar în realitate, întreaga secvenţă de interes nu este uzual prezentă în RS. În loc, diferite evenimente şterg RS, şi din noua secvenţă de instrucţiuni venită de la decodificator, RS trebuie să aleagă să dispecerizeze ceea ce are. Presupuneţi că RS este goală. În ciclul 0, primele două instrucţiuni cu redenumirea regiştrilor din această secvenţă apar în RS. Presupuneţi că este necesar un singur ciclu de ceas pentru a dispeceriza orice operaţie, şi presupuneţi că latenţele unităţilor funcţionale sunt cele din exerciţiul 3.2. Mai mult presupuneţi că în faţă (decodificator/ blocul de redenumire a regiştrilor) va continua să alimenteze continuu cu două noi instrucţiuni per ciclu de ceas. Câţi cicli de ceas necesită acum secvenţa de cod?
d) Dacă se doreşte îmbunătăţirea performanţelor de la punctul (c) ce ne va ajuta mai mult: (1) încă un ALU? (2) Altă unitate LD/ST? (3) un bypassing complet a rezultatelor ALU către operaţiile subsecvenţiale? Sau (4) Împărţirea în două a celei mai mari latenţe? Care este creştere de viteză?
e) Să considerăm acum speculaţia, actul de extragere, decodificare şi execuţie dincolo de una sau mai multe ramificaţii condiţionate. Motivaţia noastră de a face acest lucru este dublă: Planificarea dispecerizării de la punctul (c) are o mulţime de nop-uri, şi cunoaştem calculatoare care îşi petrec majoritatea din timp executând bucle (care implică salturi înapoi la vârful buclei şi care sunt uşor de prezis). Buclele ne spun unde să găsim mai multă treabă de făcut: planificarea împrăştiată a dispecerizării ne sugerează că avem oportunităţi să facem o parte din această muncă mai timpuriu decât după. La punctul (d) s-a găsit calea critică prin buclă. Imaginaţi o pliere a unei a doua copie a acelei căi obţinută prin planificarea de la punctul (b). Câţi cicli suplimentari sunt necesari pentru a executa aceste două bucle

cu valoare în execuţie (Presupuneţi că toate instrucţiunile sunt rezidente în RS)? (Presupuneţi că toate unităţile funcţionale sunt complet implementate sub formă de pipeline).

13. În acest exerciţiu, veţi explora performanţele de compromis între trei procesoarecare implică diferite tipuri de multithreading (execuţie de fire multiple). Fiecare dintre aceste procesoare este superscalar, utilizează pipeline-uri în ordine, necesită o încetinire fixă de trei cicli ce urmează pentru toate încărcările şi ramificaţiile, şi au cache-ul L1 identic. Instrucţiunile din acelaşi fir lansate în execuţie în acelaşi ciclu sunt citite în ordinea programului şi trebuie să nu conţină nicio dependenţă de date sau control.
    - ✔ Procesorul A este o arhitectură SMT (Multithreading simultam) superscalară, capabilă să lanseze două instrucţiuni per ciclu de la două fire.
    - ✔ Procesorul B este o arhitectură MT fină, capabilă să lanseze 4 instrucţiuni per ciclu pentru un singur fir şi să comute firele atunci când pipeline-ul este încetinit.
    - ✔ Procesorul C este o arhitectură MT brută, capabilă să lanseze până la 8 instrucţiuni per ciclu dintr-un singur fir şi să comute firele la lipsurile de la cache-ul de nivel 1 L1.

    Aplicaţia avută în vedere este una de căutare în liste, care scanează o regiune de memorie pentru o valoarea specifică memorată în R9 în gama de adrese specificată de R16 şi R17. Este paralelizată fie prin divizarea în 4 blocuri ocupând spaţii continue de memorie şi asignând câte un fir de căutare pentru fiecare bloc (implicând 4 fire). Majoritatea timului de execuţie a fiecărui fir este petrecut în corpul următoarei bucle desfăşurate:

```
loop:  LD R1,0(R16)
       LD R2,8(R16)
       LD R3,16(R16)
       LD R4,24(R16)
       LD R5,32(R16)
       LD R6,40(R16)
       LD R7,48(R16)
       LD R8,56(R16)
       BEQAL R9,R1,match0
       BEQAL R9,R2,match1
       BEQAL R9,R3,match2
       BEQAL R9,R4,match3
       BEQAL R9,R5,match4
       BEQAL R9,R6,match5
       BEQAL R9,R7,match6
       BEQAL R9,R8,match7
       DADDIU R16,R16,#64
       BLT R16,R17,loop
```

    Presupuneţi următoarele:

    - ✔ Se utilizează o barieră pentru a se asigura că toate firele încep simultan.
    - ✔ Prima lipsă la nivelul cache-ului L1 apare după două iteraţii ale buclei.
    - ✔ Nici este luată (Taken) nici una dintre ramificaţiile BEQAL.

✔ BLT este întotdeauna luat.
✔ Toate trei procesoarele planifică firele de execuţie într-o manieră roud-robin.
Determinaţi câţi cicli sunt necesari pentru fiecare procesor pentru a completa primele două operaţii ale buclei.

14. În acest exerciţiu, vom examina modul în care o tehnică software poate extrage paralelismul la nivel instrucţiunilor (ILP) într-o buclă vector uzuală. Următoarea buclă este aşa numita buclă DAXPY (calculul în dublă precizie aX plus Y) care este operaţia centrală în eliminarea Gaussian. Următorul cod implementează operaţia DAXPY, Y = aX + Y, pentru un vector cu lungimea 100. Iniţial, R1 este setat cu adresa de bază al ariei X şi R2 cu adresa de bază a ariei Y:

```
        DADDIU   R4,R1,#800   ; R1 = upper bound for X
foo:    L.D      F2,0(R1)     ; (F2) = X(i)
        MUL.D    F4,F2,F0     ; (F4) = a*X(i)
        L.D      F6,0(R2)     ; (F6) = Y(i)
        ADD.D    F6,F4,F6     ; (F6) = a*X(i) + Y(i)
        S.D      F6,0(R2)     ; Y(i) = a*X(i) + Y(i)
        DADDIU   R1,R1,#8     ; increment X index
        DADDIU   R2,R2,#8     ; increment Y index
        DSLTU    R3,R1,R4     ; test: continue loop?
        BNEZ     R3,foo       ; loop if needed
```

Presupuneţi că latenţele unităţilor funcţionale sunt prezentate în tabelul următor. Presupuneţi că întârzierea de un ciclu a ramificaţiilor se rezolvă în etajul ID. Presupuneţi că rezultatele sun complet byapassing.

Instruction producing result	Instruction using result	Latency in clock cycles
FP multiply	FP ALU op	6
FP add	FP ALU op	4
FP multiply	FP store	5
FP add	FP store	4
Integer operations and all loads	Any	2

a) Presupuneţi un pipeline cu o singură lansare în execuţie. Prezentaţi cum va arăta bucla atât neplanificată de compilator şi după ce compilatorul o planifică atât pentru operaţiile în virgulă flotantă şi întârzierile ramificaţiilor, incluzând încetinirile sau cicli de ceas de aşteptare. Care este timpul de execuţie (în cicli) per element al vectorului rezultat, Y, neplanificat şi planificat? Cu cât trebuie să fie mai rapid ceasul pentru hardware-ul procesorului pentru ca acesta singur să atingă îmbunătăţirea performanţelor dată de planificarea compilatorului? (neglijaţi orice efect al creşterii vitezei ceasului asupra performanţelor memoriei).
b) Presupuneţi un pipeline cu o singură lansare în execuţie. Desfăşuraţi bucla de atâtea ori cât este necesar pentru a o planifica fără nici o încetinire, reducând instrucţiunile care dau supracontrol buclei. De câte ori trebuie desfăşurată bucla?

Prezentaţi planificarea instrucţiunilor, Care este timpul de execuţie per element al rezultatului?

c) Presupuneţi utilizarea unui procesor VLIW care are instrucţiuni care conţin cinci operaţii aşa cum se arată în figura 3.16. Se vor compara două grade de desfăşurare a buclei. Primul, desfăşoară bucla de 6 ori pentru a extrage ILP şi o planifică fără nici o încetinire (de ex. ciclu de lansare complet gol), reducând instrucţiunile care dau supracontrol buclei, şi apoi se repetă procesul dar se desfăşoară bucla de 10 ori. Se ignoră slot-urile de întârziere pentru ramificaţii. Prezentaţi cele două planificări. Care este timpul de execuţie per element al vectorului rezultat pentru fiecare planificare? Ce procentaj pe slot-ul de operare s-a utilizat pentru fiecare planificare? Cât de mult diferă mărimea codului pentru cele două planificări? Care este cererea totală de regiştrii pentru cele două planificări?

Memory reference 1	Memory reference 2	FP operation 1	FP operation 2	Integer operation/branch
L.D F0,0(R1)	L.D F6,-8(R1)			
L.D F10,-16(R1)	L.D F14,-24(R1)			
L.D F18,-32(R1)	L.D F22,-40(R1)	ADD.D F4,F0,F2	ADD.D F8,F6,F2	
L.D F26,-48(R1)		ADD.D F12,F10,F2	ADD.D F16,F14,F2	
		ADD.D F20,F18,F2	ADD.D F24,F22,F2	
S.D F4,0(R1)	S.D F8,-8(R1)	ADD.D F28,F26,F2		
S.D F12,-16(R1)	S.D F16,-24(R1)			DADDUI R1,R1,#-56
S.D F20,24(R1)	S.D F24,16(R1)			
S.D F28,8(R1)				BNE R1,R2,Loop

Figure 3.16 VLIW instructions that occupy the inner loop and replace the unrolled sequence. This code takes 9 cycles assuming no branch delay; normally the branch delay would also need to be scheduled. The issue rate is 23 operations in 9 clock cycles, or 2.5 operations per cycle. The efficiency, the percentage of available slots that contained an operation, is about 60%. To achieve this issue rate requires a larger number of registers than MIPS would normally use in this loop. The VLIW code sequence above requires at least eight FP registers, while the same code sequence for the base MIPS processor can use as few as two FP registers or as many as five when unrolled and scheduled.

15. În acest exerciţiu, vom analiza modul în care performează algoritmul lui Tomasulo atunci când execută bucla din exerciţiul 3.14.

Status	Wait until	Action or bookkeeping
Issue all instructions		`if (RegisterStat[rs].Busy)/*in-flight instr. writes rs*/ {h ← RegisterStat[rs].Reorder; if (ROB[h].Ready)/* Instr completed already */ {RS[r].Vj ← ROB[h].Value; RS[r].Qj ← 0;} else {RS[r].Qj ← h;} /* wait for instruction */ } else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0;}; RS[r].Busy ← yes; RS[r].Dest ← b; ROB[b].Instruction ← opcode; ROB[b].Dest ← rd;ROB[b].Ready ← no;`
FP operations and stores	Reservation station (r) and ROB (b) both available	`if (RegisterStat[rt].Busy) /*in-flight instr writes rt*/ {h ← RegisterStat[rt].Reorder; if (ROB[h].Ready)/* Instr completed already */ {RS[r].Vk ← ROB[h].Value; RS[r].Qk ← 0;} else {RS[r].Qk ← h;} /* wait for instruction */ } else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0;};`
FP operations		`RegisterStat[rd].Reorder ← b; RegisterStat[rd].Busy ← yes; ROB[b].Dest ← rd;`
Loads		`RS[r].A ← imm; RegisterStat[rt].Reorder ← b; RegisterStat[rt].Busy ← yes; ROB[b].Dest ← rt;`
Stores		`RS[r].A ← imm;`
Execute FP op	(RS[r].Qj == 0) and (RS[r].Qk == 0)	Compute results—operands are in Vj and Vk
Load step 1	(RS[r].Qj == 0) and there are no stores earlier in the queue	`RS[r].A ← RS[r].Vj + RS[r].A;`
Load step 2	Load step 1 done and all stores earlier in ROB have different address	Read from `Mem[RS[r].A]`
Store	(RS[r].Qj == 0) and store at queue head	`ROB[h].Address ← RS[r].Vj + RS[r].A;`
Write result all but store	Execution done at r and CDB available	`b ← RS[r].Dest; RS[r].Busy ← no; ∀x(if (RS[x].Qj==b) {RS[x].Vj ← result; RS[x].Qj ← 0}); ∀x(if (RS[x].Qk==b) {RS[x].Vk ← result; RS[x].Qk ← 0}); ROB[b].Value ← result; ROB[b].Ready ← yes;`
Store	Execution done at r and (RS[r].Qk == 0)	`ROB[h].Value ← RS[r].Vk;`
Commit	Instruction is at the head of the ROB (entry h) and ROB[h].ready == yes	`d ← ROB[h].Dest; /* register dest, if exists */ if (ROB[h].Instruction==Branch) {if (branch is mispredicted) {clear ROB[h], RegisterStat; fetch branch dest;};} else if (ROB[h].Instruction==Store) {Mem[ROB[h].Destination] ← ROB[h].Value;} else /* put the result in the register destination */ {Regs[d] ← ROB[h].Value;}; ROB[h].Busy ← no; /* free up ROB entry */ /* free up dest register if no one else writing it */ if (RegisterStat[d].Reorder==h) {RegisterStat[d].Busy ← no;};`

Figure 3.14 Steps in the algorithm and what is required for each step. For the issuing instruction, rd is the destination, rs and rt are the sources, r is the reservation station allocated, b is the assigned ROB entry, and h is the head entry of the ROB. RS is the reservation station data structure. The value returned by a reservation station is called the result. RegisterStat is the register data structure, Regs represents the actual registers, and ROB is the reorder buffer data structure.

Unităţile funcţionale (FU) sunt descrise în tabelul următor.

FU Type	Cycles in EX	Number of FUs	Number of reservation stations
Integer	1	1	5
FP adder	10	1	3
FP multiplier	15	1	2

Presupuneţi următoarele:

- ✔ Unităţile funcţionale nu sunt de tip pipeline.
- ✔ Nu există forwarding între unităţile funcţionale; resultatele sun comunicate de către common data bus (CDB).
- ✔ Etajul de execuţie EX realizează atât calculul efectiv al adresei cât şi accesul la memorie pentru loads şi stores. Astfel pipeline-ul este IF/ID/IS/EX/WB.
- ✔ Loads necesită un ciclu de ceas.
- ✔ Etajele de lansare în execuţie IS şi WB necesită fiecare un ciclu de ceas.
- ✔ Există 5 slot-uri în buffer-ul de încărcare (load) şi 5 slot-uri în buffer-ul de memorare (store).
- ✔ Presupuneţi că instrucţiunea BNEZ necesită un ciclu de ceas.

a) Pentru această problemă utilizaţi un pipeline MIPS Tomasulo cu o singură lansare în execuţie ca aceea din figura 3.6 cu latenţele definite în tabelul anterior. Prezentaţi numărul de cili de încetinire pentru fiecare instrucţiune şi în care ciclu de ceas a început execuţia fiecărei instrucţiuni (intră în primul ciclu de EX) pentru trei iteraţii ale buclei. Câţi cicli durează fiecare iteraţie a buclei? Raportaţi răspunsul sub forma unui tabel cu următoarele capete de tabel:
   - ✔ Iteraţia (numărul iteraţiei buclei)
   - ✔ Instrucţiunea
   - ✔ Lansarea (ciclu când a fost lansată instrucţiunea)
   - ✔ Execuţia (cicli când instrucţiunea se execută)
   - ✔ Accesul la memorie (cicli când este accesată memoria)
   - ✔ Scrierea pe CDB (cicli când rezultatul este scris pe CDB)
   - ✔ Comentarii (descrierea oricărui eveniment la care aşteaptă instrucţiunea)

   Prezentaţi în tabel trei iteraţii ale buclei.
b) Repetaţi punctul (a) dar presupuneţi se utilizează algoritmul lui Tomasulo cu două lansări iar FPU este complet pipeline.

16. Algoritmul lui Tomasulo are un dezavantaj: se poate calcula per clock, per CDB un singur rezultat. Utilizaţi configuraţia hardware şi latenţele de la întrebarea precedentă şi găsiţi o secvenţă de cod nu mai mare de 10 instrucţiuni unde algoritmul lui Tomasulo trebuie să încetinească datorită disputei pentru CDB. Indicaţi unde apar acestea în secvenţa voastră.

17. Un predictor de ramificaţie cu corelaţie (m,n) utilizează comportarea celor mai recente m ramificaţii executate pentru a alege din 2^m predictori, fiecare dintre aceştia fiind un predictor pe n biţi. Un predictor local cu două niveluri lucrează într-o manieră asemănătoare, dar ţine doar urma comportării trecute a fiecărei ramificaţii individuale pentru a prezice comportarea viitoare. Există un compromis în proiectare implicat de un astfel de predictor: predictorii cu corelare necesită mai puţină memorie pentru istorie ceea ce le permite să menţină predictori pe 2 biţi pentru un număr mare de ramificaţii individuale (reducând probabilitatea ca o instrucţiune de ramificaţie să reutilizeaze un acelaşi predictor), în timp ce predictorii locali necesită substanţial mai multă memorie pentru a păstra istoria şi sunt astfel limitaţi la urmărirea unui număr relativ mic de instrucţiuni de ramificaţie. Pentru acest exerciţiu, consideraţi un predictor de corelaţie (1,2) care poate urmării patru ramificaţii (necesită 16 biţi) versus un predictor local (1,2) care poate urmării două ramificaţii utilizând aceeaşi cantitate de memorie. Pentru fiecare ieşire următoarei ramificaţii, furnizaţi fiecare predicţie, tabela de intrare utilizată pentru a realiza predicţia, orice actualizare a tabelului ca rezultat al predicţiei, şi rat predicţiei finale greşite pentru fiecare predictor. Presupuneţi că toate ramificaţiile până la acest punct au fost luate. Iniţializaţi fiecare predictor după cum urmează:

Correlating predictor			
Entry	Branch	Last outcome	Prediction
0	0	T	T with one misprediction
1	0	NT	NT
2	1	T	NT
3	1	NT	T
4	2	T	T
5	2	NT	T
6	3	T	NT with one misprediction
7	3	NT	NT

Local predictor			
Entry	Branch	Last 2 outcomes (right is most recent)	Prediction
0	0	T,T	T with one misprediction
1	0	T,NT	NT
2	0	NT,T	NT
3	0	NT	T
4	1	T,T	T
5	1	T,NT	T with one misprediction
6	1	NT,T	NT
7	1	NT,NT	NT

Branch PC (word address)	Outcome
454	T
543	NT
777	NT
543	NT
777	NT
454	T
777	NT
454	T
543	T

18. Presupuneţi că aveţi un procesor cu un pipeline adânc, pentru care se implementează un buffer pentru ţintele ramificaţiilor (branch-taget buffer) numai pentru ramificaţiile condiţionate. Presupuneţi că penalitatea pentru predicţie greşită este întotdeauna de 4 cicli şi penalitatea pentru lipsa din buffer este întotdeauna de 3 cicli. Presupuneţi că 90% este rata de potrivire, 90% acurateţea şi 15 % este frecvenţa ramificaţiilor. Cât de rapid este un procesor cu un buffer pentru ţintele ramificaţiilor (BTB) versus un procesor care are o penalitate fixă de doi cicli per ramificaţie? Presupuneţi un ciclu de ceas de bază per instrucţiune (CPI) fără încetinire la ramificaţie.

19. Consideraţi un buffer pentru ţintele ramificaţiilor (BTB) care are penalităţi de zero, doi, şi doi cicli de ceas pentru predicţia corectă a ramificaţiilor condiţionate, predicţie incorectă şi respectiv o lipsă din buffer. Consideraţi o proiectarea a unui BTB care distinge ramificaţiile condiţionate de cele necondiţionate, şi care memorează adresa ţintă pentru ramificaţiile condiţionate şi instrucţiunea ţintă pentru ramificaţiile necondiţionate.
    a) Care este penalitatea în cicli de ceas atunci când o ramificaţie necondiţionată este găsită în buffer?
    b) Determinaţi îmbunătăţirea dată de plierea ramificaţiilor pentru ramificaţiile necondiţionate. Presupuneţi o rată de potrivire de 90%, o frecvenţă a salturilor necondiţionate de 5%, şi o penalitatea de 2 cicli pentru o lipsă din buffer. Cât de multă îmbunătăţire se câştigă cu această sporire? Cât de mare trebuie să fie rata de potrivire pentru această sporire pentru a furniza câştig de performanţă?

Probleme propuse pentru examenul de la disciplina Structura şi Organizarea Calculatoarelor

== MOODLE TYPE ==

Prof. dr. ing. Vasile Gheorghiță GĂITAN

Ş. l. dr. ing. Ionel ZAGAN

Cap. 6 Data-Level Parallelism in Vector, SIMD, and GPU Architectures

1) Presupuneţi constantele prezentate în figura 4.2. Prezentaţi codul pentru MIPS şi VMIPS. Presupuneţi că nu putem utiliza încărcările şi memorările împrăştiate-adunate. Presupuneţi că adresele de start pentru tiPL, tiPR,clL, cşR şi clP sunt în RtiPL, RtiPR, RclL, RclR şi respectiv RclP. Presupuneţi că lungimea registrului VMIPS este programabilă de către utilizator şi poate fi asignată prin setarea registrului special VL (de exemplu li VL4). Pentru a facilita reducţia pentru adunarea vectorilor, presupuneţi că am adăugat următoarele instrucţiuni la VMIPS:
   SUMR Fd, Vs Vector Summation Reduction Single Precision:
   Această instrucţiune realizeqază reducţia sumării pe un registru vector Vs, scriind suma într-un registru scalar Fd.

Constants	Values
AA,AC,AG,AT	0,1,2,3
CA,CC,CG,CT	4,5,6,7
GA,GC,GG,GT	8,9,10,11
TA,TC,TG,TT	12,13,14,15
A,C,G,T	0,1,2,3

Figure 4.32 Constants and values for the case study.

2) Presupuneţi seq_lenght == 500, care este contorul pentru execuţia dinamică pentru ambele implementări?

3) Presupuneţi că instrucţiunea de reducere a vectorilor este executată în unitatea funcţională pentru vectori, similar cu instrucţiunea de adunare a vectorilor. Prezentaţi modul în care secvenţa de cod stabilită în convoi presupune o singură instanţă a fiecărei unităţi funcţionale. Cât de multe “chimes” necesită codul? Cât de mulţi cicli per FLOP sunt necesari, ignorând supracontrolul dat de lansarea în execuţie a instrucţiunii pe vector?

4) Presupuneţi acum că putem utiliza încărcări şi memorări împrăştiate-adunate LVI şi SVI. Presupuneţi că tiPL, tiPR, clR, şi clP sunt ranjate consecutiv în memorie. De exemplu, dacă seq_lenght == 500, aria tiPR va începe la 500 * 4 octeţi în memorie după aria tiPL. Cum va afecta aceasta modul în care puteţi scrie codul VMIPS pentru acest nucleu? Presupuneţi că puteţi iniţializa regiştrii vector cu întregi utilizând următoarele tehnici care pot, de exemplu, iniţializa registrul vector V1 cu valorile (0, 0, 2000, 2000):

```
LI R2,0
SW R2,vec
SW R2,vec+4
LI R2,2000
SW R2,vec+8
SW R2,vec+12
LV V1,vec
```

Presupuneţi că lungimea maximă a vectorului este 64. Există vreun mod în care pot fi îmbunătăţite performanţele utilizând încărcări împrăştiate-adunate? Dacă da, cu cât?

5) Presupuneţi că dorim să implementăm nucleul MrBayes pe un GPU utilizând un singur bloc de tip fir. Rescrieţi codul în C al kernel-ului utilizând CUDA. Presupuneţi că pointerul la tabelul cu tranziţiile de probabilitate şi cu probabilităţile sunt specificaţi ca parametrii pentru kernel. Invocaţi un fir pentru fiecare interacţiune din buclă. Încărcaţi şi reutilizaţi valorile în memoria partajată înainte de a realiza operaţii pe ea.

6) Cu CUDA se poate utiliza paralelismul grosier – fir la nivelul bloc pentru a calcula probabilităţile condiţionale a mai multor noduri care lucrează în paralel. Presupuneţi că vom calcula probabilităţile condiţionale de la baza spre vârful unui arbore. Presupuneţi că ariile cu probabilităţile condiţionale şi probalităţile de tranziţie sunt orgnizate în memorie , aşa cum se prezintă în gigura 4.33. Schimbaţi metoda prin care calculaţi indicele în răspunsul de la Exerciţiul 4.5. pentru a include numărul blocului.

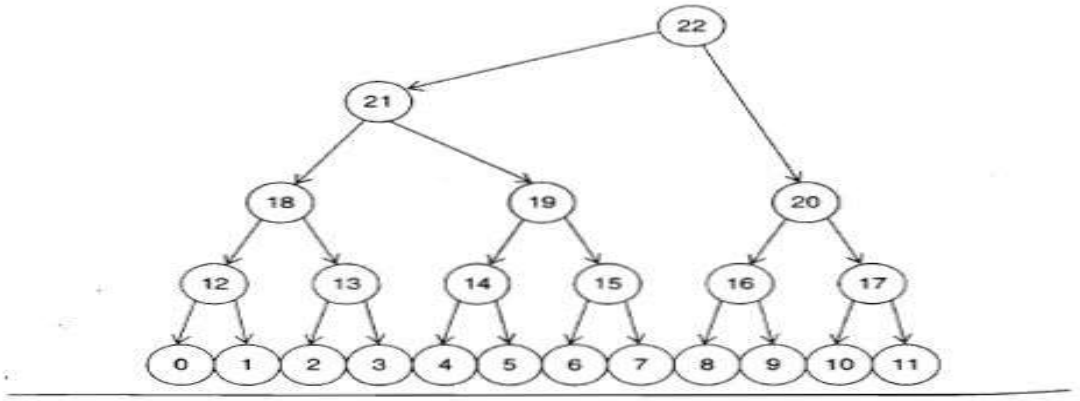


Figure 4.33 Sample tree.

7) Convertiţi codul de la exerciţiul 4.6 în cod PTX. Cât de multe instrucţiuni sunt necesare pentru kernel?

8) Cât de bine vă aşteptaţi ca acest cod să lucreze pe un GPU? Explicaţi răspunsul.

9) Consideraţi următorul cod, care multiplică doi vectori care conţin valori complexe în virgulă flotantă simplă precizie:

```
for (i = 0; i < 300; i++) {
  c_re[i] = a_re[i] * b_re[i] – a_im[i] * b_im[i];
  c_im[i] = a_re[i] * b_im[i] – a_im[i] * b_re[i];
}
```

Presupuneţi că procesorul rulează la 700 Mhz şi are lungimea maximă a vectorului de 64. Unitatea load/store are un supracontrol de start de 15 cicli; unitatea de înmulţire, 8 cicli; şi unitatea de adunare/ scădere de 5 cicli.

a) Care este intensitatea aritmetică a kernel-ului? Justificaţi răspunsul.
b) Convertiţi această buclă în asamblare VMIPS utilizând mineritul datelor pe benzi (strip-mining)
c) Presupunând o singură memorie pipeline înlănţuită, cât de mulţi “chimes” sunt necesari? Cât de mulţi cicli de ceas sunt necesari pentru rezultate complexe, incluzând supracontrolul la pornire (start-up)?
d) Dacă secvenţa de vectori este înlănţuită, cât de mulţi cicli de ceas sunt necesari pentru valorile rezultat complexe, incluzând supracontrolul?
e) Presupuneţi acum că procesorul are trei pipeline-uri şi înlănţuiri. Dacă nu există conflict de banc în accesele din buclă, cât de mulţi cicli de ceas sunt necesari per rezultat?

10) În această problemă, vom compara performanţele unui procesor vectorial cu un sistem hibrid care conţine un procesor scalar şi un coprocesor GPU. În sistemul hibrid, procesorul gazdă are performanţe scalare superioare faţă de GPU, astfel în caest caz tot codul scalar este executat pe procesorul gazdă în timp ce tot codul vectorial este executat pe GPU. Ne vom referi la primul sistem ca la calculatorul vectorial şi la al doilea sistem ca la sistemul hibrid. Presupuneţi că aplicaţia ţintă conţine un kernel vector cu o intensitate aritmetică de 0,5 FLOP per octet DRAM accesat; totuşi, aplicaţia are o componentă scalară care trebuie realizată înainte şi după kernel cu scopul de a pregătii vectorii de intrare, respectiv pe cei de ieşire. Pentru un eşantion de date de intrare, porţiunea scalară de cod necesită 400 ms de execuţie atât pe procesorul vectorial cât şi pe procesorul gazdă în sistemul hibrid. Kernel-ul citeşte vectorii de intrare constând din 200 Mo de date şi are date de ieşire constând din 100 Mo de date. Procesorul vectorial are un flux de vârf al lărgimii de bandă cu lucrul cu memoria de 30 GB/sec şi GPU are un flux de vârf al lărgimii de bandă cu lucrul cu memoria de 150 GB/sec. Sistemul hibrid are un supracontrol adiţional care necesită ca toţi vectorii de intrare să fie transferaţi între memoria gazdă şi memoria locală a GPU înainte şi după ce a fost invocat kernel-ul. Sistemul hibrid are o interfaţă cu acces direct la memorie (DMA) cu lărgimea de bandă de 10 GB/sec şi o latenţă medie de 10 ms. Presupuneţi că atât procesorul vectorial cât şi GPU sunt mărginiţi ca performanţă de lărgimera de bandă a memoriei. Calculaţi timpul de execuţie cerut de ambele calculatoare utilizate pentru această aplicaţie.

11) Secţiunea 4.5 discută despre operaţia de reducţie care reduce un vector la un scalar prin aplicarea repetată a unei operaţii. O reducţie este un tip special de recurenţă a unei bucle. Un exemplu este prezentat în continuare:

```
dot 0.0;
for (i == 0; i < 64; i++) dot = dot + a[i] * b[i];
```

Un compilator cu vectorizare poate aplica o transformare denumită *scalar expansion*, care expandează dot într-un vector şi împarte bucla astfel încât poate fi realizată înmulţirea cu o operaţie pe vector, lăsând reducţia ca o operaţie scalară separată:

```
for (i == 0; i < 64; i++) dot[i] = a[i] * b[i];
```

for (i == 0; i < 64; i++) dot[0] = dot[0] + dot[i];

Aşa cum s-a menţionat în secţiunea 4.5 dacă permitem ca adunarea în virgulă flotantă asociativă, există câteva tehnici disponibile pentru reducţia paralelizării.

a) Una din tehnici este denumită dublarea recurenţei, care adaugă secvenţe pentru progresul vectorilor scurţi (de exemplu doi vectorii cu 32 de elemente, apoi doi vectori cu 16 elemente, etc). Arătaţi cum va arăta codul C pentru execuţia celeie de a doua bucle în acest caz.
b) În unele procesoare vectoriale, elementele individuale di regiştrii vector sunt adresabili. În acest caz, operanzii pentru o operaţie pe vectori pot fi două părţi diferite ale aceluiaşi registru vector. Aceasta permite o altă soluţie pentru reducţie denumită sumă parţială (*partial sum).* Ideea este de a reduce vectorul la m sume unde m este latenţa totală prin unitatea funcţională pentru vectori, incluzând timpul de citire şi de scriere a operandului. Presupuneţi că regiştrii vector ai VMIPS sunt adresabili (de exemplu poţi iniţia o operaţie pe vector cu operandul V(16), indicând că operandul de intrare începe cu elementul 16). De asemenea, presupuneţi că latenţa locală pentru adunări, incluzând citirea şi scrierea rezultatului este de 8 cicli. Scrieţi o secvenţă de cod VMIPS care reduce conţinutul lui V1 la 8 sume parţiale.
c) Când se realizează o reducţie pe un GPU, un fir este asociat cu fiecare element al vectorului de intrare. Primul pas este, pentru fiecare fir să scrie valoarea corespunzătoare în memorie partajată. În următorul pas, fiecare fir intră într-o buclă în care adună fiecare pereche de valori de intrare. Aceasta reduce numărul de elemente la jumătate după fiecare iteraţie, ceea ce însemnă că numărul de fire active de asemenea se vor reduce la jumătate după fiecare iteraţie. Cu scopul de a minimiza performanţele reducţiei, numărul de locaţii deplin populate (warps) trebuie maximizate în timpul rulării buclei. Cu alte cuvinte, firele active trebuie să fie continue. De asemenea, fiecare fir trebuie să indexeze o arie partajată într-un astfel de mod pentru a evita conflictele de banc în memoria partajată. Următoarea buclă violează numai prima opţiune din acest ghid şi utilizează de asemenea operatorul modulo care este foarte scump pentru GPU-uri:

```
unsigned int tid = threadIdx.x;
 for(unsigned int s=1; s < blockDim.x; s *= 2) {
 if ((tid % (2*s)) == 0) {
 sdata[tid] += sdata[tid + s];
 }
 __syncthreads();
 }
```

Rescrieţi bucle pentru a îndeplini aceste ghiduri şi a elimina utilizarea operatorului modulo. Presupuneţi că există 32 de fire per warp şi un conflict de banc apare atunci când două sau mai multe fire de la acelaşi warp referenţiază un index a cărui modulo cu 32 este egal.

12) Următorul kernel realizează o porţiune din metoda cu diferenţe finite în domeniul timp (FDTD) pentru calculul ecuaţiilor lui Maxwell într-un spaţiu tridimensional, partea 1 a benchmark-ului SPEC06fp:

```
for (int x=0; x<NX-1; x++) {
 for (int y=0; y<NY-1; y++) {
 for (int z=0; z<NZ-1; z++) {
 int index = x*NY*NZ + y*NZ + z;
 if (y>0 && x >0) {
 material = IDx[index];
 dH1 = (Hz[index] – Hz[index-incrementY])/dy[y];
 dH2 = (Hy[index] – Hy[index-incrementZ])/dz[z];
 Ex[index] = Ca[material]*Ex[index]+Cb[material]*(dH2-dH1);
}}}}
```

Presupuneţi că dH1, dH2, Hy, Hz, dy, dz, Ca, Cb, şi Ex sunt arii de valori în virgulă în virgulă simplă. Presupuneţi că IDx este o arie de întregi fără semn.

a) Care este intensitatea aritmetică a acestui kernel?
b) Este kernel-ul maleabil cu execuţia SIMD a vectorului?
c) Presupuneţi că acest vector se execută pe un procesor care are lărgimea de bandă pentru memorie de 30 GB/sec. Va fi acest kernel mărginit de memorie sau de calcul?
d) Dezvoltaţi un model de tip roofline pentru acest procesor, presupunând că are un vârf de putere de calcul de 85 FFLOP/sec.

13) Presupuneţi o arhitectură GPU care conţine 10 procesoare SIMD. Fiecare SIMD instrucţiune are o lărgime de 32 şi fiecare procesor SIMD conţine 8 plane pentru aritmetica în virgulă simplă şi instrucţiunile load/store ceea ce înseamnă că o instrucţiune SIMD nedivergentă poate produce 32 de rezultate la fiecare 4 cicli. Presupuneţi un kernel care are ramificaţii divergente care cauzează o medie de 80% de fire ca să fie active. Presupuneţi că 70 % din toate instrucţiunile SIMD executate sunt în virgulă flotată singură precizie şi 20 % sunt load/store. Deşi nu sunt acoperite toate latenţele memoriei, presupuneţi că lansarea unei instrucţiuni SIMD se face cu o rată de 0,85. Presupuneţi că GPU are un ceas de 1,5 GHz.

a) Calculaţi fluxul, în GFLOP/sec, pentru acest kernel pe acest GPU.
b) Presupuneţi că aveţi următoarele alegeri:
    1) Creşterea numărului de plane pentru simplă precizie la 16
    2) Creşterea numărului de procesoare SIMD la 15 (presupuneţi că această creştere nu afectează alte metrici privind performanţele şi că se scalează codul cu procesoarele suplimentare)
    3) Adăugaţi un cache care efectiv va reduce latenţa memoriei cu 40%, care va creşte viteza de lansare a instrucţiunilor în execuţie cu 0,95.

   Care este creşterea de viteză a fluxului de calcul pentru fiecare dintre aceste îmbunătăţiri?

14) În acest exerciţiu, vom examina câteva bucle şi potenţialul lor pentru paralelizare.

a) Are următoarea buclă o dependenţă de tipul transport din buclă (loop-carried)?

```
for (i=0;i<100;i++) {
 A[i] = B[2*i+4];
 B[4*i+5] = A[i];
 }
```

b) În bucla următoare, găsiţi toate dependenţele reale, dependenţele de ieşire şi antidependenţele. Eliminaţi prin redenumire dependenţele de ieşire şi anti dependenţele.

```
for (i=0;i<100;i++) {
 A[i] = A[i] * B[i]; /* S1 */
 B[i] = A[i] + c; /* S2 */
 A[i] = C[i] * c; /* S3 */
 C[i] = D[i] * A[i]; /* S4 */
```

c) Consideraţi următoarea buclă:

```
for (i=0;i < 100;i++) {
 A[i] = A[i] + B[i]; /* S1 */
 B[i+1] = C[i] + D[i]; /* S2 */
}
```

Există dependenţe între S1 şi S2? Este această buclă paralelă? Dacă nu, arătaţi cum o facem paralelă?

15) Listează şi descrie cel puţin patru factori care influenţează performanţele unui nucleu GPU. Cu alte cuvinte, care este comportarea în timpul execuţiei care este cauzată de execuţia codului kernel care cauzează reducţia în utilizarea resurselor pe durata execuţiei kernel –ului.

16) Presupuneţi un GPU ipotetic cu următoarele caracteristici:

- ✔ Viteza ceasului de 1,5 GHz
- ✔ Conţine 16 SIMD procesoare, fiecare conţinând 16 unităţi în virgulă flotantă simplă precizie.
- ✔ Are 100 GB/sec lărgimea de bandă off-chip.

Fără a considera lărgimea de bandă a memoriei, care este vârful fluxului pentru operaţiile în virgulă flotantă simplă precizie pentru acest GPU în GFLOP/sec, presupunând că toată latenţa memoriei poate fi ascunsă? Este acest flux sustenabil fiind dată limitarea privind lărgimea de bandă a memoriei?