

I. Subiecte examen CN - 21.01.2017

Teorie

1. Care este tipul de cache cel mai potrivit pentru bus watching si care este folosit si in cadrul sistemelor multiprocesor?

a) Write-through (sigur?foarte)

b) write-back

E write-back, de ce ar fi write-through. Daca s-ar folosi write-through s-ar misca foarte greu procesoarele cand ar fi vorba de scris in memorie, plus ca bus-ul ar fi mult mai ocupat.

Protocoalele MSI si MESI sunt pentru cache-uri write-back cu snooping(adica bus watching).

si eu zic b: "Automat FSM pentru un protocol de coerență cache ce folosește cache write-back"

// Poate cineva sa explice de ce e asa ?

// Eu am vazut ca se foloseste mult wb in protocoalele MSI si MESI :D

<http://electronics.stackexchange.com/questions/285434/which-cache-type-is-better-for-bus-watching>

// cnv a intrebat asta azi :))

2. Pe perioada RAM refresh, CPU nu poate scrie in RAM.

a) Adevarat

b) Fals

Nu cred, pentru ca daca memoria avea valoarea 1 si noi voiam sa scriem 0 fix cand se facea refresh faceam chiar scurt la nivelul circuitelor (aveam 0 de o parte si 1 de cealalta).

//nu ar astepta in cazul asta pana termina refresh-ul ? si dupa sa il actualizeze ?

Pai atunci raspunsul ar fi fals la intrebare, adica CPU trebuie sa astepte RAM sa termine.

Si nu cred ca ai putea sa faci refresh la o coloana si dupa sa scrii valoarea si dupa sa continui refresh-ul pentru ca ai risca sa pierzi valorile din memorie.

- **Extended data output (EDO) DRAM** is the leading type of DRAM used in mid-1990s. It is faster than FPM DRAM because it allows the CPU to access data while the refresh cycle is being set up. The typical access time for EDO DRAM is 60 ns. As the CPU speed increases beyond 200 MHz, however, the popularity of EDO DRAM gives way to the faster SDRAM.

Sursa:

<https://books.google.ro/books?id=qhHsSlazGrQC&pg=PA325&lpg=PA325&dq=cpu+can+write+while+ram+is+refreshing&source=bl&ots=TiSr3ErgGu&sig=iz7zBluMIJId16YXGifUU0AcfV4&hl=ro&sa=X&ved=0ahUKEwjW1PyA1YPSAhVlcRQKHU4wCS4Q6AEIVTAI#v=onepage&q=CPU%20refreshing>

20write%20while%20refresh&f=false

!!!!!! While a refresh cycle is occurring the memory is not available for normal read and write operations, but in modern memory this "overhead" time is not large enough to significantly slow down memory operation. https://en.wikipedia.org/wiki/Memory_refresh

3. Daca avem un procesor care necesita 10ns pentru un acces la memorie in cazul unui hit si 40ns pentru un acces la memorie in cazul unui miss, iar hit rate-ul este de 75%, care este timpul mediu de acces la memorie?

- a) 7.5ns
- b) 40ns
- c) 17.5ns**
- d) 32.5ns

// cum se calculeaza?

// $10 \times 0.75 + 40 \times 0.25$

Average Memory Access Time = Hit time + miss_rate * miss_penalty

Dar miss penalty din problema cred ca este $40-10=30\text{ns}$

Deci ar veni $10+30 \times 0.25=17.5$ (ceea ce e echivalent cu ce e calculat sus pentru ca AMAT este si $\text{Hit_time} \times \text{hit_rate} + \text{miss_rate} \times (\text{hit_time} + \text{miss_penalty})$)

// Deci timpul de acces la memorie in cazul unui miss e (hit time + miss penalty)? De unde?

4. Care tip de cache e cel mai putin predispus la thrashing (mai multe blocuri din memorie sunt mapate in aceiasi linie)?

- a) complet asociativ
- b) set asociativ**
- c) mapat direct

// nu e A?

// de ce?

// <https://www.quora.com/What-is-cache-thrashing> - al doilea raspuns

// Eu din raspuns am inteles ca cel mai putin predispus e set-asociativ

// Citisem gresit intrebarea, scz :)

5. Presupunem un sistem multiprocesor care implementeaza coerenta cache-ului folosind protocolul MESI. Procesorul A are in cache o linie care se afla in starea E(exclusive). Daca procesorul B aduce aceiasi linie in cache, in ce stare va trece **procesorul A**: //(Probabil voia sa spuna in ce stare trece linia)

- a) Exclusive (cred. Ar fi fost Shared daca procesorul B ar fi pus linia in alt cache)**
- b) Shared (B cere linia, A trece pe shared si ii paseaza lui B linia - care tot shared va fi)
- c) Invalid
- d) Modified because writing to main memory is much slower than writing to cpu cache, and the data might be short during (means might change again sooner, and no need to put

the old version into memory). It's complex, but more sophisticated, most memory in modern cpu use this policy. – Eric Wang Aug 8 '16 at 4:32
add a comment

e) nu se poate preciza

// Pentru ca linia sa fie luata de B din A, nu ar trebui ca si A sa fie in **Shared**?

Wikipedia:

Exclusive (E)

The cache line is present only in the current cache, but is *clean* - it matches main memory. It may be changed to the **Shared** state at any time, in response to a read request. Alternatively, it may be changed to the Modified state when writing to it.

▪ Dacă o linie este în starea **M** atunci nici un alt cache nu poate să aibă o copie a linei!

– Memoria rămâne coerentă, nu pot exista copii diferite

// Presupun ca se aplica si pentru starea E (amandoua sunt exclusive)

Eu cred ca este Shared, pentru ca atunci cand se face request de catre B, A vede cererea, si (pentru ca nu e o valoare noua) va trece in shared. De asemenea, memoria ii va da valoarea cred lui B, nu cache A. Cache-urile isi dau valorile doar daca era vorba de starea M (din ce am inteles din ppt-ul pus pe cs.curs)

Adevarat ce scrie mai sus. De pe wikipedia: In this step, a BusRd is posted on the bus and the snoopers on P1 senses this. It then flushes the data and changes its state to shared. The block on P3 also changes its state to shared as it has received data from another cache. There is no main memory access here. (Unde P1 = A si P3 = B)

6. Care este cel mai lent tip de cache (necesita cel mai lung timp de cautare)?

a) **complet asociativ**

b) set asociativ

c) mapat direct

7. O problema care poate aparea in cazul cache-urilor write-through este:

- a) poate duce la perioade lungi in care memori e indisponibila //I guess
- b) nu poate fi folosit in cadrul sistemelor multiprocesor, chiar daca se face snooping pentru update-uri
- c) **mareste timpul de**
- d) **scriere in memorie** ← **curs 2 slide 41**
- e) nicio varianta de mai sus

// Prin memorie se refera la cache sau la DRAM?

// Eu am intuitia ca raspunsul e **a)**, pentru ca timpul de scriere nu e afectat, ci doar DRAM-ul devine indisponibil => cache indisponibil pt ca asteapta (de aia se apeleaza la buffering)

//Nu cred ca e c), tot atat dureaza sa scrii in memorie, de ce ar dura mai mult daca folosesti Write through, in loc de write back, sa zicem? Doar ca la write back nu scrii si in RAM de fiecare data cand modifici in Cache. Dar timpul de scriere ramane acelasi. Prin urmare, merg pe **a)**.

// clar nu e c). in cursul 2, slide 41, scrie "Write-through încetinește memoria cache pentru a permite memoriei principale să scrie datele", ceea ce nu coincide cu raspunsul asta.

ESTE c). Daca faci write-through, trebuie sa astepti ca valoarea sa se scrie in memorie (ceea ce dureaza mult mai mult decat daca ai scrie in cache direct). Plus ca la a), memoria e "indisponibila" doar cat se face un write (ceea ce nu e o perioada lunga). E adevarat, ca va creste numarul de transferuri pe bus catre memorie.

^ I don't think that's true. Zice timpul de scriere in MEMORIE, nu in cache. Cred ca e d).

8. Care este avantajul primar al DMA in comparatie cu Interrupt-driven I/O?

Nu blocheaza CPU (si nu polueaza cache-ul), pentru ca transfera direct datele de la I/O la memorie.

// Nici interrupt driven I/O nu blocheaza CPU:

An alternative scheme for dealing with I/O is the interrupt-driven method. Here the CPU works on its given tasks continuously. When an input is available, such as when someone types a key on the keyboard, then the CPU is interrupted from its work to take care of the input data. In our example, the basketball player would

<https://www.calvin.edu/academic/rit/webBook/chapter2/design/hardware/io/interrupts.htm>

Pai il blocheaza mai putin decat pooling-ul(unde procesorul verifica mereu daca sunt date disponibile), dar aici tot procesorul trebuie sa ia datele. DMA le scrie singur.

Din curs7 am mai gasit: salveaza latime de banda si are performanta marita pt ca permite CPU si I/O in paralel.

9. Dati doua sau trei exemple de probleme care pot aparea atunci cand conectam mai multe dispozitive la o magistrala.

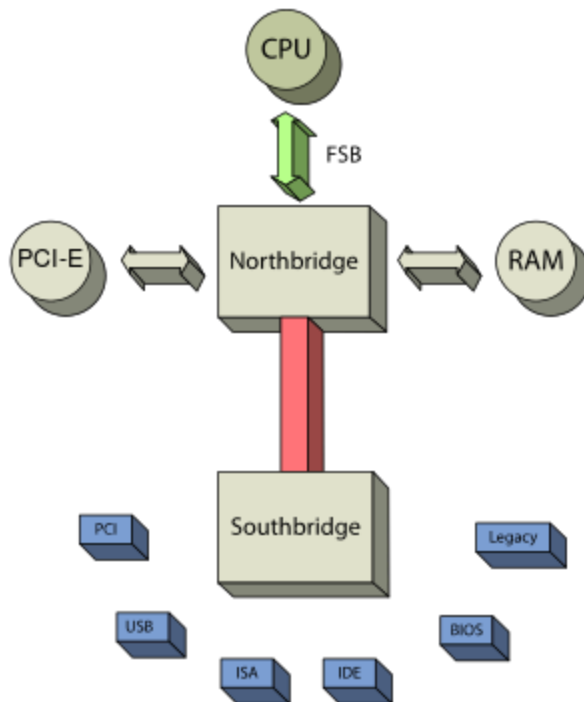
1. **Latimea de banda a magistralei poate limita I/O throughput**

2. **Magistrala trebuie sa permita functionarea unor dispozitive cu latente si viteze de transfer diferite**
3. **Coliziuni**

10. Dati exemplu de un modul conectat la Northbridge si un modul conectat la Southbridge in calculatorul vostru.

Northbridge: RAM, PCIe

Southbridge: USB, PCI, SATA (hard diskurile)



11.

```
for(j=0; j < N; j++) {
    for(i=0; i < M; i++) {
        x[i][j] = 2 * x[i][j];
    }
}
```



```
for(i=0; i < M; i++) {
    for(j=0; j < N; j++) {
        x[i][j] = 2 * x[i][j];
    }
}
```

Ce fel de localitate exploatează schimbarea de sus?

S-a prins cineva si poate da o explicatie?

Localitate spatiala -> **daca folosesti mai intai elementele de pe linii, care sunt contigue, se vor aduce in memoria cache (cel mai probabil) si blocurile urmatoare de memorie => timp mai mic pentru fetch (proprietate: cand se acceseaza o locatie de memorie este foarte probabil ca dupa sa se acceseze si locatiile vecine -> in cazul parcurgerii mai intai pe coloane, nu se indeplineste proprietatea)**

La fel si al asta:

```
for(i=0; i < N; i++)
    a[i] = b[i] * c[i];

for(i=0; i < N; i++)
    d[i] = a[i] * c[i];
```



```
for(i=0; i < N; i++)
{
    a[i] = b[i] * c[i];
    d[i] = a[i] * c[i];
}
```

Dar schimbarea de sus?

Localitate temporală -> **daca o locatie de memorie e accesata, e foarte probabil ca aceeaasi locatie de memorie sa fie accesata si in viitorul apropiat**

punem ca avem un procesor

Probleme

1. Presupunem ca avem un procesor cu frecventa de 2GHz care are CPI = 1.0 in cazul in care avem doar hit-uri. Avem un singur nivel de cache, care are miss rate de 3% si un timp de acces la memoria RAM de 200ns.

De cate ori devine mai rapid procesorul daca introducem si un cache de nivel 2, cu miss rate de 0.2% si timp de acces la date de 2ns?

$$\text{amat} = \text{hit_time} + \text{miss_rate} * \text{miss_penalty}$$

$$\text{hit_time1} = 1 / (2 * 10^9) = 0.5 \text{ ns}$$

$$\text{miss_rate1} = 3\%$$

$$\text{miss_penalty1} = 200 \text{ ns} \quad // \text{miss_penalty nu ar trb sa fie } 199.5 \text{ ns?}$$

$$\Rightarrow \text{amat1} = \text{hit_time1} + \text{miss_rate1} * \text{miss_penalty1} = 6.5 \text{ ns}$$

$$\text{miss_rate12} = 0.2\% \text{ din } 3\%$$

$$\text{miss_penalty12} = 200 \text{ ns (dureaza la fel de mult sa acceseze memoria RAM)}$$

$\text{hit_time2} = 2 \text{ ns}$ //de unde le scoti?(din enunt :) xD //nu e miss penalty pt cache2?(cache2 la miss cauta tot in ram ca si cache1(fara 2) deci penalty-ul e acelasi..nici eu nu sunt f sigur dar cred ca prin timp de access se ref la hit_time2, altfel nici nu vad cum l-ai putea calcula din ce iti da pe acolo)

$$\text{hit_time12} = 97\% \text{ hit_time1} + 99.8\% \text{ din } 3\% \text{ hit_time2} // 97\% \text{ L1, } 99.8\% \text{ din rest L2}$$

$$\Rightarrow \text{amat12} = \text{hit_time12} + \text{miss_rate12} * \text{miss_penalty12}$$

$$\text{Speedup} = \text{amat12} / \text{amat1} * 100$$

2. Avem un benchmark care ruleaza in 100ns, din care 90ns sunt operatii pe CPU, iar in celelalte 10ns sunt operatii de I/O. In urmatoorii 5 ani, performantele CPU s-au imbunatatit cu 50% la fiecare an, iar performantele I/O raman neschimbate.

- a) Cu cat s-a imbunatatit performanta CPU dupa cei 5 ani?
- b) Cu cat s-a imbunatatit timpul de rulare al benchmark-ului?

a) $(3/2)^5$

b) $1 / (0.9 / (3/2)^5 + 0.1)$

//de unde $(3/2)^5$?

Vezi problema din curs 09. Era bine cu $3/2$ la a 5-a. In curs e cu 10 la a 4-a sau ceva de genul.

$$(3/2)^5 + 0.1$$

//de unde $(3/2)^5 \rightarrow (150\% \text{ de } 5 \text{ ori} \rightarrow (150/100) * (150/100) * \dots \rightarrow (150/100)^5 \rightarrow (3/2)^5$

// si atunci nu ar trebuii modificat si a) ? gata :)

De unde $150\%?!?!?!?$

3. Avem urmatoarele 2 secvente de cod:

Loop A:

```
int sum = 0;
```

```
for(int i = 0; i < 128; i++)
```

```
    for(int j = 0; j < 64; j++)
```

```
        sum += a[i][j];
```

Loop B:

```
int sum = 0;
for(int j = 0; j < 64; j++)
    for(int i = 0; i < 128; i++)
        sum += a[i][j];
```

Consideram ca avem un sistem cu cache pe 4KB si linii care ocupa 32 de octeti si au 8 cuvinte. De asemenea, vom considera ca procesorul citeste din memorie doar valorile tabloului **a**, celelalte variabile fiind permanent stocate in registri.

Tabloul **a** este format din intregi (pe 4 octeti) si este stocat continuu in memorie, adica:

- > la adresa 0, A[0][0],
- > la adresa 4, A[0][1],
- > la adresa 8, A[0][2],
- >
- > la adresa 256, A[1][0] // sigur e ok?
- > ...
- > la adresa (4 * 64 * 128), A[63][127] // aici e A[127][63] mai mult ca sigur :)

a) Calculati numarul de miss-uri pentru Loop A si Loop B daca cache-ul este mapat direct.

Considerand ca matricea A are 128 linii si 64 coloane si stocam in memorie linie cu linie din matrice. Vom avea:

Loop A:

For-ul Interior: 8 miss-uri ~~2 miss-uri (in cache avem 16 linii a cate 32 octeti)~~

For-ul Exterior: 128 miss-uri (la fiecare linie noua dam request + la 16 iteratii incepem sa suprascriem -- not that it would matter now)

Miss-uri Totale: 1024 ($128 * 64 / 8$, pentru ca la fiecare 8 accesari avem un miss)

***** Linia are 32 de bytes, dar un element din matrice ocupa 4 bytes, nu 1 byte**

Cred ca e asa:

i = 0

j = 0 miss aduce linia cu elementele (0,1,2,3,4,5,6,7) - adresele (0-31) // linie de 32B

j = 1-7 => 7 hits

j = 8 iar miss

j = 9-15 => 7 hits

deci pentru tot forul cu j face $64/8$ cicluri => 8 miss + $8 * 7$ hits

pentru tot forul cu i, face $128(8\text{miss} + 8*7\text{ hits})$

Loop B:

For-ul Interior: 128 miss-uri

For-ul Exterior: 64 miss-uri

Miss-uri Totale: 128 x 64

Nu cred ca e corect la LoopB. Pt ca in cache incap 128 de linii a cate 8 elem.

Cand am $j=0$ pun in cache primele 8 elem de pe fiecare linie.

Cand $j=1$, eu am deja in cache $a[0][1]$, $a[1][1]$, ..., $a[127][1]$

Si o sa fie hit-uri de la $j=1$ pana la $j=7$ inclusiv.

La $j=8$ iar o sa am 128 de miss-uri. Dar care ma asigura pana la $j=16$. ETC...

Missuri Totale = $128 * (64/8) = 128 * 8$

E ciudat ca da la fel ca LoopA si ma asteptam sa fie mai multe.

// Nu asa functioneaza, cache-ul incarca o linie din matrice ca e zona contigua de memorie. Nu se aplica si la coloane.

In momentul in care ceri $a[0][0]$, cache incarca linia $a[0][0] \rightarrow a[0][7]$ (este zona contigua de mem)

Apoi cand ceri $a[1][0]$, incarca linia $a[1][0] \rightarrow a[1][7]$

Deci pt $j=0$, cache o sa incarce 128 de linii cu elementele $a[\text{linie}][0] \rightarrow a[\text{linie}][7]$

- b) Care este numarul minim de linii pentru cache astfel incat sa nu avem niciun miss atunci cand se executa Loop A (exceptand miss-urile obligatorii de la inceput) ? Dar pentru Loop B?

Loop A:

Cache: Trebuie sa avem 8 linii, fiecare a cate 64 octeti pentru a evita miss-uri pe for-ul interior.

Nu cumva doar o linie? (adica nu iti mai pasa de datele vechi) // ideea e sa ai linii de minim 64 de octeti. Cum dimensiunea cache-ului e data si fixa, nu poti controla si nr de linii din cache.

Latimea e fixa. Nu poti sa modifici dimensiunea unei linii.

Loop B:

Cache: Trebuie sa avem 128 linii, fiecare a cate 1 octet pentru a evita un numar maxim de miss-uri pe for-ul interior. Pe for-ul exterior vom avea in continuare 64 miss-uri.

Nu cumva $128 * 64 * 4 / 32$? (ca sa nu incarci de doua ori aceeasi valoare) (numarul total de octeti necesari / numarul de octeti pe linie)

- c) Care este numarul de miss-uri pentru Loop A si pentru Loop B daca cache-ul este complet asociativ, iar politica de inlocuire este FIFO?

Loop A:

Din anii trecuti

II. Teorie

0. Avem o memorie cache cu cuvinte de 1 octet, linii de 8 octeti lungime organizata in 32 de seturi de cate 4 linii. Care este dimensiunea utila a memoriei cache, in octeti? Daca adresarea se face pe 32 de biti, care este dimensiunea tag-ului pentru fiecare linie de cache?

1KB // Aici zic ceva de dimensiunea utila, nu ar trb sa scadem si partea de tag-uri sau ceva?

5 biti set index

3 biti de offset

$32 - (5 + 3) = 24$

1. daca ai linii de 2 ori mai lungi in cache spune ce se intampla cu miss obligatoriu, miss de conflict si miss de capacitate

Miss obligatoriu - scade

Miss de capacitate - scade // Ramane aceeasi

Miss de conflict - creste (daca numarul de linii se injumatatesteste)

<https://courses.cs.washington.edu/courses/cse378/02sp/sections/section9-2.html>

Aici zice ca miss de capacitate ramane la fel, practic sansa de miss fiindca nu mai ai spatiu e aceeasi, fiindca chiar daca ai mai mult spatiu, acel spatiu este ocupat de liniile care intra in cache.

2. daca ai linii de 2 ori mai scurte (cache e la jumate din capacitate pt ca se pastreaza asociativitatea si inca ceva) ce se intampla cu hit time, miss rate si miss penalty

Hit time - scade

Miss rate - creste

Miss penalty - scade // Miss penalty = RAM_time - hit_time, iar daca hit_time scade, miss_penalty nu creste?

3. Se da urmatoarea secventa de cod:

add.d f0, f1, f8

add.d f2, f3, f8

add.d f4, f5, f8

add.d f6, f7, f8

Care din urmatoarele tehnici va aduce o imbunatatire pentru codul de mai sus:

-branch prediction (n-ai instructiuni de branch)

-out of order execution cu register renaming (register renaming nu trebuie si nici out-of order nu aduce vreun beneficiu pentru ca e fix aceeasi instructiune)

-superscalar (da, pentru ca se pot executa simultan toate 4)

4. daca ai pagini de 2 ori mai mari ce se intampla cu rata de miss TLB si **cu TLB reach??**

4. Avem un sistem cu memorie virtuala si TLB. Daca dublam numarul de intrari in TLB, cum vor fi afectate rata de miss a TLB si capacitatea acestuia de adresare (TLB reach)?

TLB reach creste. Miss la ce? // la TLB

The TLB stores the recent translations of **virtual memory** to **physical memory** and can be called an address-translation cache.

The amount of memory accessible from the **TLB**.

TLB Reach = (TLB Size) X (Page Size).

The working set (= un set de pagini virtuale active) of each process is stored in the **TLB**, Otherwise there is a high degree of page faults.

Increase the Page Size : This may lead to an increase in fragmentation as not all applications require a large page size

Provide Multiple Page Sizes : This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation

5.Care sunt componentele unei adrese de 16 biti a unui cache mapat direct, cu datele accesibile la nivel de octeti? Linia de cache are o latime de $W = 4$ octeti. Dimensiunea cache $L=1024$ linii(4kb).

Pozitia unui octet in linie este codificata pe $\log_2(4) = 2$ biti.

Adresa de index a unei linii de cache este $\log_2(1024) = 10$ biti.

Eticheta: $16 - (2 + 10) = 4$ bit.

6. Descrieti, pe scurt, modul de functionare al unui controller DMA.

CN2-lecture8.pdf — EECS 252 Graduate Computer Architecture Lec 01 - Introduction

16 of 37


101,719

Index

- Calculatoare Nu... 1
- Comic of the day 2
- Din episodul ante... 3
- Configurații de bus 4
- Exemplu de calcu... 5
- Exemplu de calcu... 6
- Creșterea lățimii ... 7
- Bus Master 8
- Arbitrare 9
- Tehnici de arbitrare 10
- Tehnici de arbitra... 11
- Rolul sistemului ... 12
- Slide Number 13 13
- Slide Number 14 14
- Direct Memory A... 15
- Funcționarea DMA 16**
- Direct Memory A... 17
- Transfer DMA ol... 18
- Avantajul de baz... 19
- DMA și memoria ... 20
- Probleme DMA: ... 21
- Probleme DMA: ... 22
- DMA și Memoria ... 23
- Unde sunt toate ... 24
- PCI este... 25
- Slide Number 26 26
- Conexiuni fizice: ... 27
- Spațiul de adresă... 28
- Dispozitivele PCI ... 29
- Localizarea tutur... 30
- Alocarea adreselor 31
- Întreruperi PCI 32

Funcționarea DMA

* Figure is courtesy of Dave Patterson



- CPU inițializează device ID, adresele din memorie și numărul de octeți de transferat
- DMA controller (DMAC) inițiază accesul și devine bus master
- Pentru transferul mutiplu de date, DMAC incrementează adresele
- DMAC întrerupe CPU la terminarea transferului

(Context: Direct memory access (DMA) is a feature of computer systems that allows certain hardware subsystems to access main system **memory (RAM)**, independent of the **central processing unit (CPU)**. Without DMA, when the CPU is using **programmed input/output**, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an **interrupt** from the DMA controller when the operation is done.)

Standard DMA, also called third-party DMA, uses a DMA controller. A DMA controller can generate **memory addresses** and initiate memory read or write cycles. It contains several **hardware registers** that can be written and read by the CPU. These include a memory address register, a byte count register, and one or more control registers. The control registers could specify the source, the destination, the direction of the transfer (reading from the I/O device or writing to the I/O device), the size of the transfer unit, and/or the number of bytes to transfer in one burst depending on what features the DMA controller provides.^[1]

To carry out an input, output or memory-to-memory operation, the host processor initializes the DMA controller with a count of the number of **words** to transfer, and the memory address to use. The CPU then sends commands to a peripheral device to initiate transfer of data. The DMA controller then provides addresses and read/write control lines to the system memory. Each time a byte of data is ready to be transferred between the peripheral device and memory, the DMA controller increments its internal address register until the full block of data is transferred.

7. enumera 3 magistrale si la ce folosesc

Magistrala Procesor-Memorie : trebuie doar sa conecteze sistemul de memorie

Magistrala I/O : acomodeaza o gama larga de dispozitive I/O

Backplane Bus : permite procesoarelor, memoriei si dispozitivelor I/O sa coexiste

8. in ce consta consistenta secventiala (asta era la multiprocesoare)

APD ftw

^ didn't you mean *fm*? ← INDEED!

9. Explicati cum sunt afectate cele 3 tipuri de miss(obligatoriu, de conflict si de capacitate) daca adaugam capacitatea de prefetching unui cache?

Obligatoriul scade, capacitatea ramane la fel, conflict (s-ar putea sa scada sau ramane la fel)

^ nu creste missul de conflict? e posibil sa poluezi cache-ul.

// si capacitatea si conflicti sunt afectate negativ din cauza poluarii

10. Cum sunt afectati parametrii de Hit Time, Miss Rate si Miss Penalty daca dublam asociativitatea unui cache set-asociativ(dar pastream capacitatea si dimensiunea liniilor constante)?

Hit_time creste, miss rate scade, miss_penalty creste

11. Care dintre cele 3 abordari furnizeaza cea mai mare imbunatatire a performantei la rularea urmatorului cod:

LD R1 0(R2) #cache miss

ADD R2 R1 R1

LD R1 0(R3) #cache hit

LD R3 0(R4) #cache hit

ADD R3 R1 R3

ADD R1 R2 R3

Variante:

-branch prediction// (care-i faza de e recurenta treaba cu branch prediction?)

-out of order execution cu register renaming (nu prea cred ca ar imbunatati mult)

-superscalar (https://en.wikipedia.org/wiki/Superscalar_processor)

12. Avem un sistem cu memorie virtuala si TLB. Daca dublam numarul de intrari in TLB, cum vor fi afectate rata de miss a TLB si capacitatea acestuia de adresare(TLB reach)?

(vezi si ex 4 de mai sus tot cu TLB)

TLB Reach = (TLB Size) X (Page Size).

TLB Size creste => TLB Reach va creste => mai multa memorie va fi accesibila din TLB

Cum mai multa memorie e accesibila => sunt sanse mai mari ca adresa pe care o cautam sa existe in TLB, deci rata de miss ar trebui sa scada.

13.

5. Monica este studenta in anul 1 la Calculatoare si doreste sa imbunatateasca viteza de rulare a codului ei. Ea modifica in felul urmatoar programul ei:

Before

```
for (i=0; i<N; i++)  
  for (j=0; j<N; j++)  
    B[j*N+i] = A[i*N+j];
```

After

```
int BS=8;  
for (i=0; i<N; i+=BS)  
  for (j=0; j<N; j++)  
    for (k=0; k<BS; k++)  
      B[j*N+(i+k)] = A[(i+k)*N+j];
```

A procedat Monica corect?

Monica: Pare ca Monica a facut cache tiling, deci a procedat corect.

14.

6. Prietenul Monicai, Irinel, are aceeași problema:

Before

```
for (i=0; i<N; i++)  
  c[i] = a[i] + b[i];
```

After

```
int P=1;  
for (i=0; i<N; i++) {  
  prefetch(&a[i+P]);  
  prefetch(&b[i+P]);  
  prefetch(&c[i+P]);  
  c[i] = a[i] + b[i];  
}
```

Se executa mai rapid codul dupa optimizarea lui Irinel?

Da, adica face prefetch.

7. Avem un sistem de calcul cu un singur nivel de cache (L1) Daca adaugam un al doilea nivel de cache (L2), cum se vor modifica urmatoorii parametri de functionare: L1 Hit Time, L1 Miss Rate, L1 Miss Penalty?

15.

L1 hit time scade, L1 miss penalty scade, L1 miss rate... (ramane la fel!?) // miss rate creste, deoarece L1 este un cache mai mic

16.

8. Descrieti protocolul MESI.

17.

9. Explicati modul de functionare al unui TLB.

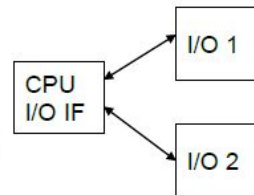
18.

10. In sistemele de calcul moderne se folosesc din ce in ce mai mult magistrale de comunicare seriala (SATA, USB etc). De ce sunt acestea preferate magistrelor paralele (dati cel putin trei motive)?

Linii seriale punct-la-punct dedicate



- Legăturile punct-la-punct funcționează la viteze multi-gigabit folosind codificări avansate pentru ceas/semnale de date (necesită multă electronică la ambele capete ale comunicației)
- Low power – avem o singură legătură de date cu un singur periferic
- SATA, USB, Firewire, etc.
- Transferuri simultane multiple
- Conectori și cabluri ieftine (mai puțin cupru pentru cabluri/ mai mult siliciu pentru logica de comunicație)
- Fiecare dispozitiv are legătura lui de date, cu parametri proprii de viteză și comunicație
- Exemple: Ethernet, Infiniband, PCI Express,



19. Alex este student in anul I la Calculatoare si doreste sa imbunatateasca viteza de rulare a codului lui. El modifica in felul urmator programul lui:

Before :

```
for(i = 0; i < M; i++)
    for(j = 0; j < N; j++)
        x[i*N+j] = 2*x[i*N+j];
```

After

```
for(j = 0; j < N; j++)
    for(i = 0; i < M; i++)
        x[i*N+j] = 2*x[i*N+j];
```

Unde x este un vector de intregi de 32 de biti. A procedat Alex corect?

Nu prea, ca nu se mai foloseste de localitatea spatiala

20. Prietena lui Alex, Antonia are aceeași problemă:

Before:

```
for(i = 0; i < N; i++)  
    a[i] = b[i]*c[i];  
for(i = 0; i < M; i++)  
    d[i] = a[i]*c[i];
```

After:

```
for(i = 0; i < N; i++){  
    a[i] = b[i]*c[i];  
    d[i] = b[i]*c[i];  
}
```

Se execută mai rapid codul după îmbunătățirea Antoniei?

- **Loop fusion - se execută mai rapid**

II. Probleme

1. 2014

1. Un procesor are două niveluri de cache (L1 și L2) și o memorie RAM. ~~Sistemul are un CPI = 1 dacă~~ Programul care rulează pe această arhitectură are 10000 de operații load și store care au 10% hit în L1, 30% hit în L2 iar restul rezidă în RAM. Timpii de acces pentru diferitele memorii sunt: $t_{L1} = 2$ cicli, $t_{L2} = 5$ cicli și $t_{RAM} = 100$ cicli. Care este valoarea timpului mediu de acces la memorie? (1p)

2. Un bus sincron are o perioadă de ceas de 10ns și fiecare transmisie durează un ciclu de ceas. Alt bus asincron are nevoie de 40ns pentru fiecare handshake. Fiecare bus are 32 de biți de date. Care este lățimea de bandă a fiecărui bus pentru citiri de un cuvânt dintr-o memorie cu 500ns per read? (1p)

3. Avem un procesor cu adrese pe 8 biți și o memorie cache de 16 octeți cu lungimea liniei de 2 octeți, datele fiind accesibile la nivel de octet. Timpul de acces în cazul unui hit este $T_{hit} = 20 \text{ ns}$ și timpul de acces în cazul unui miss este $T_{miss} = 100 \text{ ns}$. Cât va dura următoarea secvență de cod și ce date se vor afla în cache dacă:

- adresele sunt **mapate direct**; (1p)
 - adresele sunt mapate **set asociativ cu 2 cai** și politica de înlocuire a liniilor este **LRU**; (2p)
 - adresele sunt mapate **set asociativ cu 2 cai** și politica de înlocuire a liniilor este **FIFO**? (2p)
 - Ce date se vor afla în RAM la aceleași adrese (cele folosite în cod) după terminarea execuției în cele trei cazuri dacă politica de scriere în cache este **write-back**? (4p)
- Se știe: RAM[8] = 0; RAM[9] = 1; RAM[16] = 2; RAM[17] = 3; RAM[20] = 4; RAM[21] = 5; RAM[32] = 6; RAM[33] = 7;

```
load R0, 8
load R1, 9
load R2, 21
load R3, 16
store 8, R1
load R0, 32
store 20, R1
```

Pentru Problema 2 -> exemplu asemanator din curs

CN2-lecture8.pdf — EECS 252 Graduate Computer Architecture Lec 01 - Introduction

5 of 37

101,719

Index

- Calculatoare Nu...
- Comic of the day
- Din episodul ante...
- Configurații de bus
- Exemplu de calcu...
- Exemplu de calcu...
- Creșterea lățimii ...
- Bus Master
- Arbitrare
- Tehnici de arbitrare
- Tehnici de arbitra...
- Rolul sistemului ...
- Slide Number 13
- Slide Number 14
- Direct Memory A...
- Funcționarea DMA
- Direct Memory A...
- Transfer DMA ol...
- Avantajul de baz...
- DMA și memoria ...
- Probleme DMA: ...
- Probleme DMA: ...
- DMA și Memoria ...
- Unde sunt toate ...
- PCI este...
- Slide Number 26
- Conexiuni fizice: ...
- Spațiul de adresă...
- Dispozitivele PCI ...
- Localizarea tutur...
- Alocarea adreselor
- Întreruperi PCI

Exemplu de calcul performanță

Un bus sincron are o perioadă de ceas de 50ns și fiecare transmisie durează un ciclu de ceas. Alt bus asincron are nevoie de 40ns pentru fiecare handshake. Fiecare bus are 32 de biți de date. Care este lățimea de bandă a fiecărui bus pentru citiri de un cuvânt dintr-o memorie cu 200ns per read.

Răspuns:

Pașii pentru magistrala sincronă:

- Trimite adresa la memorie: 50 ns
- Citește memoria: 200 ns
- Trimite datele la dispozitiv: 50 ns

Lățimea maximă de bandă este 4 octeți la 300 ns $\Rightarrow \frac{4 \text{ bytes}}{300 \text{ ns}} = \frac{4 \text{ M B}}{0.3 \text{ sec}} = 13.3 \text{ MB/sec}$

Catedra de Calculatoare

Facultatea de Informatică și Calculatoare

Index

Calculatoare Nu...	1
Comic of the day	2
Din episodul ante...	3
Configurații de bus	4
Exemplu de calcu...	5
Exemplu de calcu...	6
Creșterea lățimii ...	7
Bus Master	8
Arbitrare	9
Tehnici de arbitrare	10
Tehnici de arbitra...	11
Rolul sistemului ...	12
Slide Number 13	13
Slide Number 14	14
Direct Memory A...	15
Funcționarea DMA	16
Direct Memory A...	17
Transfer DMA ol...	18
Avantajul de baz...	19
DMA și memoria ...	20
Probleme DMA: ...	21
Probleme DMA: ...	22
DMA și Memoria ...	23
Unde sunt toate ...	24
PCI este...	25
Slide Number 26	26
Conexiuni fizice: ...	27
Spațiul de adresă...	28
Dispozitivele PCI ...	29
Localizarea tutur...	30
Alocarea adreselor	31
Întreruperi PCI	32

Exemplu de calcul performanță



Secvența de pași pentru magistrala asincronă:

1. Memory read address atunci când apare ReadReq : 40 ns
 - 2,3,4. Data ready & handshake: $\max(3 \times 40 \text{ ns}, 200 \text{ ns}) = 200 \text{ ns}$
 - 5,6,7. Read & Ack. : $3 \times 40 \text{ ns} = 120 \text{ ns}$
- } 360 ns

Lățimea maximă de bandă este 4 octeți la 360 ns $\Rightarrow \frac{4 \text{ bytes}}{360 \text{ ns}} = \frac{4 \text{ MB}}{0.36 \text{ sec}} = 11.1 \text{ MB/sec}$