

Coerența memoriei

- trebuie ca toate procesoarele să folosească ultima valoare scrisă pentru o variabilă partajată
- problema - memoriile cache
- scopul - orice variabilă partajată să aibă aceeași valoare
 - în toate cache-urile
 - în memoria principală

Coerența - cache *write-back*

x - variabilă partajată

Procesor	Acțiune	Valoare cache A	Valoare cache B	Valoare memorie
				9
A	i = x ;	9		9
B	j = x ;	9	9	9
A	x = 5 ;	5	9	9
B	k = x ;	5	9	9

Coerența - *cache write-through*

x - variabilă partajată

Procesor	Acțiune	Valoare cache A	Valoare cache B	Valoare memorie
				9
A	i=x;	9		9
B	j=x;	9	9	9
A	x=5;	5	9	5
B	k=x;	5	9	5

Ce înseamnă coerența? (1)

1. ordinea execuției

- a) procesorul P scrie în variabila X
- b) apoi procesorul P citește X
- între a) și b) nu există alte scrieri în X

→ citirea b) returnează valoarea scrisă de a)

Ce înseamnă coerența? (2)

2. viziune coerentă a memoriei

- a) procesorul P scrie în variabila X
 - b) apoi procesorul Q ($Q \neq P$) citește X
 - între a) și b) nu există alte scrieri în X
 - între a) și b) a trecut suficient timp
- citirea b) returnează valoarea scrisă de a)

Ce înseamnă coerența? (3)

3. serializarea scrierilor

- a) procesorul P scrie în variabila X
- b) procesorul Q ($Q=P$ sau $Q \neq P$) scrie în variabila X

→ toate procesoarele văd cele două scrieri în aceeași ordine

– nu neapărat a) înaintea b)

Menținerea coerenței cache-urilor

- protocoale de menținere a coerenței
- se bazează pe informațiile despre liniile de cache
 - *invalid* - datele nu sunt valide
 - *dirty* - doar cache-ul curent deține valoarea actualizată
 - *shared* - cache-ul curent deține valoarea actualizată, la fel memoria principală și eventual alte cache-uri

Tipuri de protocoale

- *directory based*
 - informațiile despre fiecare linie de cache -
ținute într-un singur loc
- *snooping*
 - fiecare cache are o copie a liniei partajate
 - fără centralizarea informației
 - cache-urile monitorizează magistrala
 - detectează schimbările produse în liniile de cache

Actualizarea cache-urilor

- fiecare cache anunță modificările făcute
- celelalte cache-uri reacționează
- contează doar operațiile de scriere
- variante
 - scriere cu invalidare (*write invalidate*)
 - scriere cu actualizare (*write update*)

Scriere cu invalidare (1)

- un procesor modifică o dată
- modificarea se face în cache-ul propriu
 - toate celelalte cache-uri sunt notificate
- celelalte cache-uri
 - nu au o copie a datei modificate - nici o acțiune
 - au o copie a datei modificate - își invalidează linia corespunzătoare
 - valoarea corectă va fi preluată când va fi nevoie

Scriere cu invalidare (2)

x - variabilă partajată

Proc.	Acțiune	Reacție cache	Cache A	Cache B	Memorie
					9
A	i=x;	read miss	9		9
B	j=x;	read miss	9	9	9
A	x=5;	invalidation	5	inv.	9
B	k=x;	read miss	5	5	5

Scriere cu actualizare (1)

- un procesor modifică o dată
- modificarea se face în cache-ul propriu
 - toate celelalte cache-uri sunt notificate
 - se transmite noua valoare
- celelalte cache-uri
 - nu au o copie a datei modificate - nici o acțiune
 - au o copie a datei modificate - preiau noua valoare

Scriere cu actualizare (2)

x - variabilă partajată

Proc.	Acțiune	Reacție cache	Cache A	Cache B	Memorie
					9
A	i=x;	read miss	9		9
B	j=x;	read miss	9	9	9
A	x=5;	invalidation	5	5	5
B	k=x;	read hit	5	5	5

Invalidare vs. actualizare (1)

- mai multe scrieri succesive în aceeași locație
 - *write invalidation* - o singură invalidare (prima dată)
 - *write update* - câte o actualizare pentru fiecare scriere
 - mai avantajos - invalidare

Invalidare vs. actualizare (2)

- mai multe scrieri în aceeași linie de cache
 - modificarea unei locații necesită invalidarea/actualizarea întregii linii
 - *write invalidation* - o singură invalidare (prima dată)
 - *write update* - câte o actualizare pentru fiecare scriere
 - mai avantajos - invalidare

Invalidare vs. actualizare (3)

- "timpul de răspuns"
 - timpul între modificarea unei valori la un procesor și citirea noii valori la alt procesor
 - *write invalidation* - întâi invalidare, apoi citire (când este necesar)
 - *write update* - actualizare imediată
 - mai avantajos - actualizare

Invalidare vs. actualizare (4)

- ambele variante au avantaje și dezavantaje
- scrierea cu invalidare - ocupare (mult) mai redusă a memoriei și magistralelor
- scrierea cu actualizare - rata de succes a cache-urilor mai mare
- mai des folosită - invalidarea

III. Dispozitivele periferice

Dispozitiv periferic

- realizează o formă oarecare de comunicare între procesor și "lumea exterioară"
- pentru gestiunea comunicării cu procesorul, include un controller I/O
 - de obicei, acesta conține o serie de regiștri care rețin informații necesare pentru comunicare
 - date
 - informații de stare
 - comenzi (de la procesor)

Intrări-ieșiri (I/O)

- cum privește sistemul comunicarea
 - I/O proiectat pe memorie
 - citirile/scrierile sunt văzute ca și cum ar fi realizate în locații de memorie
 - adresele I/O - în spațiul de adrese de memorie
 - aceleași semnale de control ca pentru memorie
 - I/O izolate
 - adresele I/O - separate de adresele de memorie
 - semnale de control diferite de ale memoriei

Moduri de comunicare (1)

- intrări-ieșiri programate
 - *programmed I/O*
 - programul așteaptă într-o buclă până când perifericul inițiază un transfer
 - eficient dacă se știe dinainte momentul când perifericul va solicita comunicarea
 - consum inutil de timp de execuție al procesorului

Moduri de comunicare (2)

- acces direct la memorie
 - *Direct Memory Access* (DMA)
 - un controller specializat (DMA controller) efectuează transferurile
 - foarte rapid
 - preia controlul magistralelor și transferă datele direct între periferic și memorie
 - fără intervenția procesorului
 - util pentru transferul volumelor mari de date

Moduri de comunicare (3)

- intrări-ieșiri gestionate prin întreruperi
 - interrupt-driven I/O
 - când un periferic dorește să inițieze comunicarea, notifică procesorul
 - printr-o cerere de întrerupere
 - în restul timpului, procesorul poate îndeplini alte sarcini
 - metoda cea mai flexibilă

Magistrale (1)

- căi de comunicație a informației
- o magistrală leagă între ele mai mult de 2 componente printr-o cale unică
- descrierea unei magistrale
 - semnalele electrice folosite
 - reguli de comunicare - trebuie respectate de toate părțile implicate
 - mod de conectare

Magistrale (2)

Accesul la magistrală

- mai multe entități pot solicita simultan accesul
- este necesară o procedură de arbitraj
 - decide cine va primi accesul
 - celelalte trebuie să aștepte eliberarea magistralei

Arbitrarea magistralelor

Tipuri de arbitrare a magistralelor

- centralizat
 - decizia o ia un circuit specializat (arbitru)
- descentralizat
 - componentele se înțeleg între ele
 - pe baza regulilor care definesc funcționarea magistralei

Conectarea la magistrală

- probleme de natură electrică
- mai multe circuite conectate împreună
 - în intrare și ieșire
- nu se pot conecta mai multe ieșiri
 - nivelele diferite de tensiune ar distruge circuitele
- o soluție - multiplexarea
 - toate ieșirile sunt conectate la un multiplexor

Circuite *tri-state*

- ieșirea are 3 stări posibile
 - 0
 - 1
 - impedanță infinită (*High-Z*)
- primele două corespund valorilor obișnuite
- a treia implică decuplarea de pe magistrală
 - ca și cum ieșirea circuitului nici nu ar fi conectată la magistrală

Circuite *open-collector*

- în unele cazuri se numesc *open-drain*
 - în funcție de tehnologia utilizată
- este posibilă conectarea mai multor ieșiri simultan
- valoarea rezultată - funcția AND între ieșirile conectate

Magistrale - privire generală

Avantaje

- activitatea pe magistrală - ușor de controlat
- economic - structură relativ simplă

Dezavantaj

- performanțe mai scăzute
 - doar 2 componente pot comunica la un moment dat

IV. Sistemul de întreruperi

Ce este o întrerupere?

- procesorul poate suspenda execuția programului curent
- scop - tratarea unor situații neprevăzute
- după tratare se reia programul întrerupt
- scopul inițial - comunicarea cu perifericele
- procesorul nu "așteaptă" perifericele
 - acestea îl solicită când este necesar

Înteruperi hardware

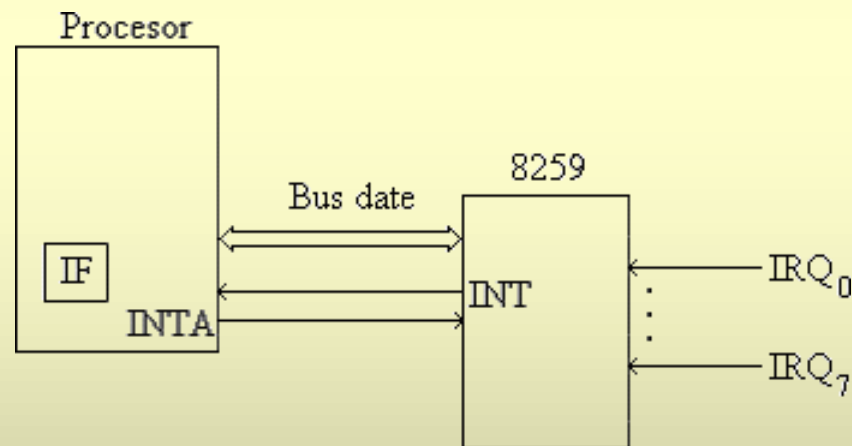
- dezactivabile (*maskable*)
 - procesorul poate refuza tratarea lor
 - depinde de valoarea bistabilului IF (*Interrupt Flag*): 1 - acceptare, 0 - refuz
 - IF poate fi modificat prin program
- nedezactivabile (*non-maskable*)
 - procesorul le tratează întotdeauna

Controllerul de întreruperi (1)

- circuit specializat
- preia cererile de întrerupere de la periferice
- le trimite procesorului
- rezolvă conflictele (mai multe cereri simultane) - arbitrare
 - fiecare periferic are o anumită prioritate

Controllerul de întreruperi (2)

- inițial - Intel 8259
 - se pot folosi mai multe (cascadare)



- astăzi - integrat în chipset

Tratarea întreruperilor - etape (1)

- dispozitivul periferic generează o cerere de întrerupere pe linia sa IRQ_i
- controllerul trimite un semnal pe linia INT
- procesorul verifică valoarea bistabilului IF
 - numai pentru întreruperi dezactivabile
 - dacă este 0 - refuză cererea; stop
 - dacă este 1 - răspunde cu un semnal INTA

Tratarea întreruperilor - etape (2)

- se întrerupe execuția programului curent
- se salvează în stivă regiștrii (inclusiv PC)
- se șterge bistabilul IF
 - se blochează execuția altei întreruperi în timpul execuției programului pentru întreruperea în curs
 - poate fi repus pe 1 prin program

Tratarea întreruperilor - etape (3)

- identificare periferic (sursa cererii)
 - controllerul depune pe magistrala de date un octet *type*
 - indică perifericul care a făcut cererea
 - maximum $2^8=256$ surse de întrerupere
 - fiecare sursă are rutina proprie de tratare
 - periferice diferite - tratări diferite

Tratarea întreruperilor - etape (4)

- determinarea adresei rutinei de tratare
 - adresa fizică 0 - tabelul vectorilor de întreruperi
 - conține adresele tuturor rutinelor de tratare
 - dimensiune: $256 \text{ adrese} \times 4 \text{ octeți} = 1 \text{ KB}$
 - octet type = $n \rightarrow$ adresa rutinei de tratare: la adresa $n \times 4$

Tratarea întreruperilor - etape (5)

- salt la rutina de tratare a întreruperii
- execuția rutinei de tratare
- revenire în programul întrerupt
 - restabilire valoare bistabil IF
 - restabilire valoare regiștri (din stivă)
 - reluarea execuției programului de unde a fost întrerupt

Extindere

- sistem puternic și flexibil
- poate fi extins - utilizare mai largă
- programele trebuie întrerupte și în alte situații (nu doar pentru comunicarea cu perifericele)
- util mai ales pentru sistemul de operare

Tipuri de întreruperi

- *hardware* - generate de periferice
- *excepții (traps)* - generate de procesor
 - semnalează o situație anormală
 - exemplu: împărțire la 0
- *software* - generate de programe
 - folosite pentru a solicita anumite servicii sistemului de operare