IV.9. Arhitectura RISC

Structura clasică a CPU

CISC (*Complex Instruction Set Computer*)

- număr mare de instrucţiuni
- complexitate mare a instrucţiunilor → timp mare de execuţie
- număr mic de regiştri → acces intensiv la memorie

Observaţii practice

- multe instrucţiuni sunt rar folosite
- 20% din instrucţiuni sunt executate 80% din timp
  - sau 10% sunt executate 90% din timp
  - depinde de sursa de documentare...
- instrucţiunile complexe pot fi simulate prin instrucţiuni simple

Structura alternativă

RISC (*Reduced Instruction Set Computer*)

- set de instrucţiuni simplificat
  - instrucţiuni mai puţine (relativ)
  - şi mai simple funcţional (elementare)
- număr mare de regiştri (zeci)
- mai puţine moduri de adresare a memoriei

Accesele la memorie

- format fix al instrucţiunilor
  - acelaşi număr de octeţi, chiar dacă nu toţi sunt necesari în toate cazurile
  - mai simplu de decodificat
- arhitectură de tip *load*/*store*
  - accesul la memorie - doar prin instrucţiuni de transfer între memorie şi regiştri
  - restul instrucţiunilor lucrează doar cu regiştri

Avantajele structurii RISC

- instrucţiuni mai rapide
- reduce numărul de accese la memorie
  - depinde de capacitatea compilatoarelor de a folosi regiştrii
- accesele la memorie - mai simple
  - mai puţine blocaje în pipeline
- necesar de siliciu mai mic - se pot integra circuite suplimentare (ex. cache)

IV.10. Arhitecturi paralele de calcul

Calcul paralel - utilizare

- comunicare între aplicaţii
  - poate fi folosită şi procesarea concurentă
- performanţe superioare
  - calcule ştiinţifice
  - volume foarte mari de date
  - modelare/simulare
    - meteorologie, astronomie etc.

Cum se obţine paralelismul?

- structuri pipeline
  - secvenţial/paralel
- sisteme multiprocesor
  - unităţile de calcul - procesoare
- sisteme distribuite
  - unităţile de calcul - calculatoare

Performanţa

- ideal - viteza variază liniar cu numărul de procesoare
- real - un program nu poate fi paralelizat în întregime
- exemple
  - operaţii de I/O
  - sortare-interclasare

Scalabilitatea (1)

- creşterea performanţei o dată cu numărul procesoarelor
- probleme - sisteme cu foarte multe procesoare
- factori de limitare
  - complexitatea conexiunilor
  - timpul pierdut pentru comunicare
  - natura secvenţială a aplicaţiilor

Scalabilitatea (2)

- funcţională pentru un număr relativ mic de procesoare
- număr relativ mare
  - creşterea de performanţă nu urmează creşterea numărului de procesoare
- număr foarte mare
  - performanţa se plafonează sau poate scădea

Sisteme de memorie

- după organizarea fizică a memoriei
  - centralizată
  - distribuită
- după tipul de acces la memorie
  - comună (partajată)
  - locală

Tipuri de sisteme multiprocesor

- sisteme cu memorie partajată centralizată
- sisteme cu memorie partajată distribuită
- sisteme cu schimb de mesaje

Memorie partajată centralizată

- denumiri
  - UMA (*Uniform Memory Access*)
  - SMP (*Symmetrical Multiprocessing*)
- memorie comună
- accesibilă tuturor procesoarelor
- timpul de acces la memorie
  - acelaşi pentru orice procesor şi orice locaţie

Memorie partajată distribuită

- denumiri
  - DSM (*Distributed Shared Memory*)
  - NUMA (*Non-Uniform Memory Access*)
- memoria este distribuită fizic
- spaţiu de adrese unic
  - văzut de toate procesoarele
- accesul la memorie - neuniform

Sisteme cu schimb de mesaje

- multicalculatoare
- fiecare procesor are propria memorie locală
  - nu este accesibilă celorlalte procesoare
- comunicarea între procesoare
  - transmiterea de mesaje explicite
  - similar reţelelor de calculatoare

Comunicare

- conectarea
  - între procesoare
  - între procesoare şi memorie
- variante
  - sisteme cu memorie partajată centralizată - magistrală
  - sisteme cu memorie partajată distribuită - reţele de interconectare

Comunicare pe magistrală

- economic
- proiectare simplă
- performanţă mai redusă
- nu scalează bine cu numărul de procesoare
- rol important - memoriile cache
  - probleme de coerenţă

Reţele de interconectare (1)

- între
  - procesoare
  - procesoare şi memorie
- scop
  - flexibilitate
  - performanţă
    - mai multe accese în paralel

Reţele de interconectare (2)

- tipuri de conectare
  - totală - fiecare cu fiecare
  - parţială - unele perechi de componente nu sunt conectate direct
- conectare procesoare-memorii
  - mai multe circuite de memorie
  - pot fi accesate în paralel de procesoare diferite

Coerenţa memoriei

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

Coerenţa - cache *write-back*

`x` - variabilă partajată

Procesor	Acţiune	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

Coerenţa - cache *write-through*

`x` - variabilă partajată

Procesor	Acţiune	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 coerenţa? (1)

1. ordinea execuţiei
   a) procesorul P scrie în variabila X
   b) apoi procesorul P citeşte X
   – între a) şi b) nu există alte scrieri în X

   → citirea b) returnează valoarea scrisă de a)

Ce înseamnă coerenţa? (2)

2. viziune coerentă a memoriei
   a) procesorul P scrie în variabila X
   b) apoi procesorul Q (Q≠P) citeşte X
   – între a) şi b) nu există alte scrieri in X
   – între a) şi b) a trecut suficient timp

   → citirea b) returnează valoarea scrisă de a)

Ce înseamnă coerenţa? (3)

3. serializarea scrierilor
   a) procesorul P scrie în variabila X
   b) procesorul Q (Q=P sau Q≠P) scrie în variabila X

→ toate procesoarele văd cele două scrieri în aceeaşi ordine

– nu neapărat a) înaintea b)

Menţinerea coerenţei cache-urilor

- protocoale de menţinere a coerenţei
- se bazează pe informaţiile despre liniile de cache
  - *invalid* - datele nu sunt valide
  - *dirty* - doar cache-ul curent deţine valoarea actualizată
  - *shared* - cache-ul curent deţine valoarea actualizată, la fel memoria principală şi eventual alte cache-uri

Tipuri de protocoale

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

Actualizarea cache-urilor

- fiecare cache anunţă modificările făcute
- celelalte cache-uri reacţionează
- contează doar operaţiile 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 acţiune
  - au o copie a datei modificate - îşi invalidează linia corespunzătoare
  - valoarea corectă va fi preluată când va fi nevoie

Scriere cu invalidare (2)

`x` - variabilă partajată

Proc.	Acţiune	Reacţie 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 acţiune
  - au o copie a datei modificate - preiau noua valoare

Scriere cu actualizare (2)

`x` - variabilă partajată

Proc.	Acţiune	Reacţie 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 aceeaşi locaţie
  - *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 aceeaşi linie de cache
  - modificarea unei locaţii 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 şi 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 şi dezavantaje
- scrierea cu invalidare - ocupare (mult) mai redusă a memoriei şi magistralelor
- scrierea cu actualizare - rata de succes a cache-urilor mai mare
- mai des folosită - invalidarea

V. Dispozitivele periferice

Magistrale (1)

- căi de comunicaţie a informaţiei
- 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 părţile implicate
  - mod de conectare

Magistrale (2)

Accesul la magistrală

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

Arbitrarea magistralelor

Tipuri de arbitrare a magistralelor

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

Conectarea la magistrală

- probleme de natură electrică
- mai multe circuite conectate împreună
  - în intrare şi ieşire
- nu se pot conecta mai multe ieşiri
  - nivelele diferite de tensiune ar distruge circuitele
- o soluţie - multiplexarea
  - toate ieşirile sunt conectate la un multiplexor

Circuite *tri-state*

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

Circuite *open-collector*

- în unele cazuri se numesc *open-drain*
  - în funcţie de tehnologia utilizată
- este posibilă conectarea mai multor ieşiri simultan
- valoarea rezultată - funcţia AND între ieşirile conectate

Magistrale - privire generală

Avantaje

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

Dezavantaj

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