

Cache parțial asociativ (1)

- numit și cache asociativ pe seturi
- derivat din cache-ul cu adresare directă
- fiecare linie de cache conține mai multe seturi de date (4, 8, 16, ...)
- structura unui set
 - bit de validare
 - etichetă
 - date din memoria principală

Cache parțial asociativ (2)

set ₁			set ₂			set _n		
bit validare ₁			bit validare ₂			bit validare _n		
↙	eticheta ₁	date ₁	↙	eticheta ₂	date ₂	...	↙	eticheta _n date _n
						...		
⋮								
						...		
⋮								
						...		

Cache parțial asociativ (3)

Timpul de acces (T_c)

- puțin mai mare decât la cache-ul cu adresare directă
 - trebuie verificate toate cele n seturi

Rata de succes (H)

- ridicată
 - elimină problema suprapunerilor

Scrierea în cache (1)

- scriere într-o locație care nu se află în cache
- unde se face scrierea?
- variante
 - doar în memoria principală - nu se poate
 - de ce?
 - doar în cache (*write-back*)
 - atât în cache, cât și în memoria principală (*write-through*)

Scrierea în cache (2)

Cache de tip *write-back*

- scrierea se face doar în cache
- datele ajung în memoria principală doar la evacuarea din cache
- viteză mare
- probleme în sistemele multiprocesor

Scrierea în cache (3)

Cache de tip *write-through*

- scrierea se face atât în cache, cât și în memoria principală
- mai lent
 - datorită accesului la memoria principală
- ambele tipuri de cache sunt larg folosite

Conceptul de cache - extindere

- aplicabil nu doar la procesoare
- tipul de problemă: comunicarea cu o entitate lentă, de mare capacitate
- soluția: interpunerea unei entități cu capacitate mai mică și viteză mai mare
 - reține ultimele date vehiculate

Unde mai putem folosi ideea?

Aplicabilitate

- oriunde funcționează legile localizării
- hardware
- software

Example (1)

Cache-uri de disc

- 2 direcții de aplicare
 - hardware - circuit de memorie integrat în controller
 - software - o zonă din memoria sistem
- entitatea mai mare și lentă - discul
- entitatea mai mică și rapidă - memoria

Example (2)

Browserul web

- ultimele pagini accesate sunt reținute pe disc
 - numai localizare temporală - de ce?
- entitatea mai mare și lentă - rețeaua (Internet)
- entitatea mai mică și rapidă - discul

II.2. Memoria virtuală

Ideea de pornire

Problema

- aplicațiile - consum mare de memorie
- memoria disponibilă - insuficientă

Cum se poate rezolva?

- capacitatea discului hard - foarte mare
- nu toate zonele de memorie ocupate sunt accesate la un moment dat

Memoria virtuală

Soluția - memoria virtuală (*swap*)

- unele zone de memorie - evacuate pe disc
- când este nevoie de ele, sunt aduse înapoi în memorie

Cine gestionează memoria virtuală?

- sunt necesare informații globale
- sistemul de operare

Fișierul de paginare

- conține zonele de memorie evacuate pe disc
- informații pentru regăsirea unei zone stocate
 - adresele din memorie
 - programul căruia îi aparține
 - dimensiunea
 - etc.

Politica de înlocuire (1)

- problema - aceeași ca la memoria cache
- aducerea unei zone de memorie din fișierul de paginare implică evacuarea alteia
 - care?
- scop - minimizarea acceselor la disc
- politică inefficientă → număr mare de accese la disc → scăderea vitezei

Politica de înlocuire (2)

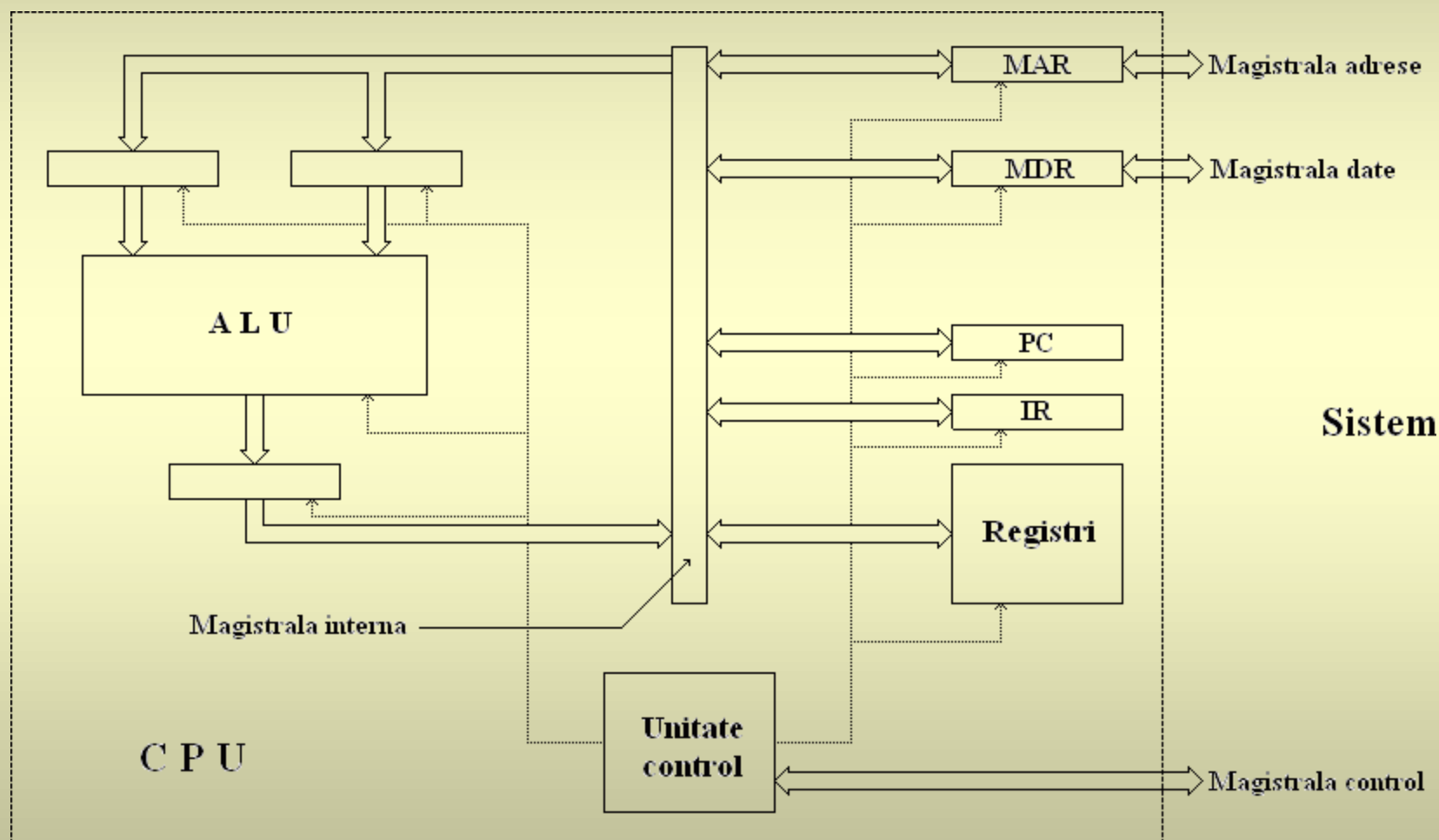
- set de lucru (*working set*) - zonele de memorie necesare programului la un moment dat
- uzual mult mai mic decât totalitatea zonelor folosite de program
- dacă încapă în memorie - puține accese la disc

Politica de înlocuire (3)

- se va selecta pentru evacuare zona care nu va fi necesară în viitorul apropiat
- nu se poate ști cu certitudine - estimare
 - pe baza comportării în trecutul apropiat
- paginare la cerere (*demand paging*) - evacuare pe disc numai dacă este strict necesar

III. Unitatea centrală de procesare (CPU)

Structura CPU (1)



Structura CPU (2)

- unitatea aritmetică și logică (ALU)
 - efectuează calculele propriu-zise
- regiștrii de uz general
- unitatea de control
 - comandă celelalte componente
 - stabilește ordonarea temporală a operațiilor
- magistrala internă

Structura CPU (3)

- contorul program (PC)
 - reține adresa următoarei instrucțiuni de executat
 - actualizat de procesor
 - uzual nu este accesibil prin program
- registrul de instrucțiuni (IR)
 - reține codul ultimei instrucțiuni aduse din memorie

Structura CPU (4)

- regiștrii de interfață
 - asigură comunicarea cu magistralele sistemului
 - de adrese: MAR (*Memory Address Register*)
 - de date: MDR (*Memory Data Register*)
- regiștrii temporari
 - intermediari între diverse componente
 - exemple: regiștrii operanzi ALU, registrul rezultat ALU

IV. Îmbunătățirea performanței CPU

Cum putem crește performanța?

- eliminarea factorilor care frânează CPU
 - exemplu - folosirea memoriei cache
- structuri cât mai simple
 - nu mai este posibil la procesoarele actuale
- creșterea frecvenței ceasului
 - limitată de tehnologie
- execuția instrucțiunilor în paralel

Creșterea performanței - tehnici

- Structura de tip *pipeline*
- Multiplicarea unităților de execuție
- Predicția salturilor
- Execuția speculativă
- Predicația
- Execuția *out-of-order*
- Redenumirea regiștrilor
- *Hyperthreading*
- Arhitectura RISC

IV.1. Pipeline

Ideea de pornire

- execuția unei instrucțiuni - număr mare de pași
- în pași diferiți se folosesc resurse diferite ale CPU
- execuția unei instrucțiuni poate începe înainte de terminarea celei anterioare
- instrucțiunile se execută (parțial) în paralel

O primă implementare

Procesorul Intel 8086

- format din două unități
 - unitatea de interfață cu magistrala (BIU)
 - comunicarea cu exteriorul
 - unitatea de execuție (EU)
 - execuția propriu-zisă a operațiilor
- BIU și EU pot lucra în paralel

Principiul benzii de asamblare

- execuția unei instrucțiuni - n pași
- la un moment dat - n instrucțiuni în execuție
- fiecare instrucțiune - în alt pas

	pas 1	pas 2	...	pas $n-1$	pas n
instrucțiune 1			...		
instrucțiune 2			...		
⋮	⋮				
instrucțiune $n-1$...		
instrucțiune n			...		

Pipeline

- secvența pașilor (*stagii*) prin care trece execuția unei instrucțiuni
- trecerea între două stagii - la fiecare ciclu de ceas
- cât durează până la terminarea unei instrucțiuni?
 - prima instrucțiune - n cicluri de ceas
 - următoarele instrucțiuni - câte 1 ciclu de ceas !

Performanța unui pipeline

- rezultatul obținut la fiecare stadiu trebuie reținut
- regiștri de separație - plasați între stagii
- frecvența ceasului - dată de stagiul cel mai lung
- pași mai simpli
 - număr de stagii mai mare
 - frecvență mai mare a ceasului

Execuția unei instrucțiuni

1. depunerea valorii PC (adresa instrucțiunii) în MAR
2. citirea din memorie
3. preluarea codului instrucțiunii în MDR
4. depunerea codului instrucțiunii în IR
5. actualizarea valorii PC

Execuția unei instrucțiuni

6. decodificarea instrucțiunii de către unitatea de control
7. citire operand din memorie
 - depunere adresă operand în MAR
 - comandă citire
 - preluare operand în MDR
- 7'. selecție registru care conține operandul

Execuția unei instrucțiuni

8. depunere operand în registru operand ALU
9. repetare pași 7-8 pentru al doilea operand
10. transmiterea către ALU a codului operației dorite
11. preluare rezultat în registrul rezultat ALU
12. testare condiție salt

Execuția unei instrucțiuni

13. salt (dacă este cazul)

14. scriere rezultat în memorie

- depunere rezultat în MDR
- depunere adresă în MAR
- comandă scriere

14'. scriere rezultat în registrul destinație

Evoluție

- Intel Pentium III - 10 stagii
- Intel Pentium IV (Willamette, Northwood) - 20 stagii
- Intel Pentium IV (Prescott) - 32 stagii
- AMD Athlon - 17 stagii

Probleme

- nu toate instrucțiunile se pot executa în paralel
- dependență - o instrucțiune trebuie să aștepte terminarea alteia
- conflict în accesul la aceeași resursă

Parametri de performanță

- latența (*latency*) - numărul de cicluri de ceas necesar pentru execuția unei instrucțiuni
 - dat de numărul de stagii
- rata de execuție (*throughput*) - numărul de instrucțiuni terminate pe ciclu de ceas
 - teoretic - egală cu 1
 - practic - mai mică (din cauza dependențelor)

Tipuri de dependențe

- structurale
- de date
- de control

Dependențe structurale

- instrucțiuni aflate în stagii diferite au nevoie de aceeași componentă
- o singură instrucțiune poate folosi componenta la un moment dat
- celelalte instrucțiuni care au nevoie de ea sunt blocate

Dependențe structurale - exemple

- ALU
 - instrucțiuni aritmetice
 - calculul adreselor operanzilor
 - actualizarea valorii PC
- accesele la memorie
 - citire cod instrucțiune
 - citire operand
 - scriere rezultat

Dependențe de date

- o instrucțiune calculează un rezultat, alta îl folosește
- a doua instrucțiune are nevoie de rezultat înainte ca prima să-l obțină
- a doua instrucțiune este blocată

Dependențe de date - exemplu

```
mov  eax, 7
```

```
sub  eax, 3
```

- prima instrucțiune: scrierea în `eax` - în ultimul stadiu
- a doua instrucțiune: utilizarea `eax` - în primele stagii (decodificare)
 - așteaptă până când prima instrucțiune depune rezultatul în `eax`

Dependențe de control (1)

Actualizarea valorii PC (uzual)

- adunarea la vechea valoare a dimensiunii codului instrucțiunii anterioare
- încărcarea unei valori noi - instrucțiuni de salt

Dependențe de control (2)

Tipuri de instrucțiuni de salt

- necondiționat
 - se face saltul întotdeauna
- condiționat
 - se face saltul numai dacă este îndeplinită o anumită condiție
 - altfel se continuă cu instrucțiunea următoare

Dependențe de control (3)

Adresa de salt - moduri de exprimare

- valoare constantă
 - absolută
 - deplasament față de adresa instrucțiunii curente
- valoarea dintr-un registru
- valoarea dintr-o locație de memorie

Dependențe de control (4)

Adresa de salt - exemple:

`jmp 1594`

`jmp short -23`

`jmp eax`

`jmp dword ptr [esi]`

Dependențe de control (5)

Probleme

- calculul adresei de salt - în ultimele stagii de execuție
- instrucțiunile următoare (multe!) au început deja execuția
- dacă se face salt - efectele lor trebuie anulate

Dependențe de control (6)

Probleme

- "golirea" pipeline-ului → pierdere de performanță
 - operații complicate
 - durează mult până la terminarea primei instrucțiuni → scade rata de execuție
- o instrucțiune din 7 (în medie) este de salt !