II.2. Memoria virtuală

Ideea de pornire

Problema

- aplicaţiile - 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

Soluţia - 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 informaţii globale
- sistemul de operare

Fişierul de paginare

- conţine zonele de memorie evacuate pe disc
- informaţii pentru regăsirea unei zone stocate
  - adresele din memorie
  - programul căruia îi aparţine
  - dimensiunea
  - etc.

Politica de înlocuire (1)

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

Politica de înlocuire (2)

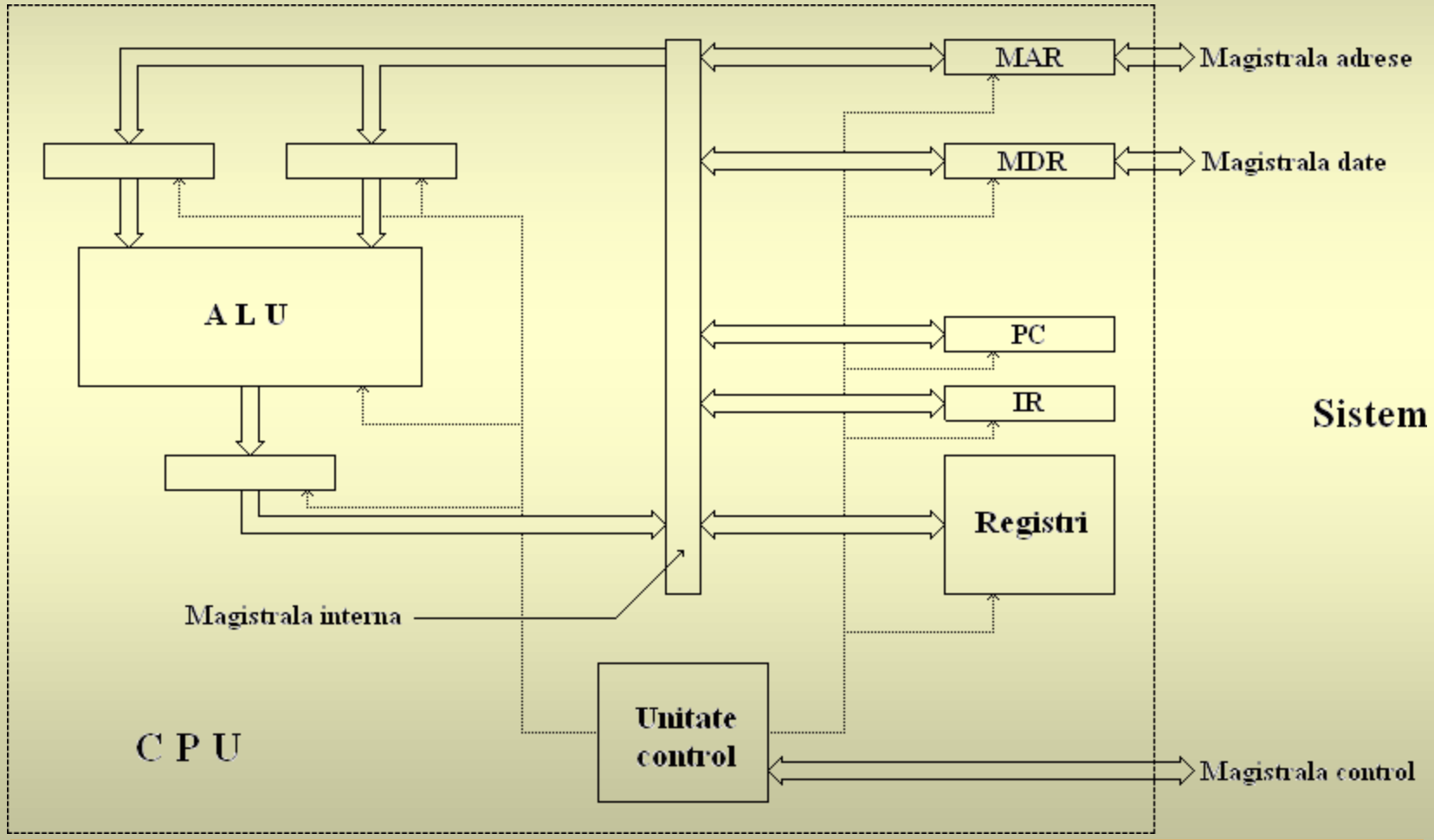
- set de lucru (*working set*) - zonele de memorie necesare programului la un moment dat
- uzual mult mai mic decât totalitatea zonelor folosite de program
- dacă încape în memorie - puţine accese la disc

Politica de înlocuire (3)

- se va selecta pentru evacuare zona care nu va fi necesară în viitorul apropiat
- nu se poate şti 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 şi logică (ALU)
  - efectuează calculele propriu-zise
- regiştrii de uz general
- unitatea de control
  - comandă celelalte componente
  - stabileşte ordonarea temporală a operaţiilor
- magistrala internă

Structura CPU (3)

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

Structura CPU (4)

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

IV. Îmbunătăţirea performanţei CPU

Cum putem creşte performanţa?

- eliminarea factorilor care frânează CPU
  - exemplu - folosirea memoriei cache
- structuri cât mai simple
  - nu mai este posibil la procesoarele actuale
- creşterea frecvenţei ceasului
  - limitată de tehnologie
- execuţia instrucţiunilor în paralel

Creşterea performanţei - tehnici

- Structura de tip *pipeline*
- Multiplicarea unităţilor de execuţie
- Predicţia salturilor
- Execuţia speculativă
- Predicaţia
- Execuţia *out-of-order*
- Redenumirea regiştrilor
- *Hyperthreading*
- Arhitectura RISC

IV.1. Pipeline

Ideea de pornire

- execuţia unei instrucţiuni - număr mare de paşi
- în paşi diferiţi se folosesc resurse diferite ale CPU
- execuţia unei instrucţiuni poate începe înainte de terminarea celei anterioare
- instrucţiunile se execută (parţial) în paralel

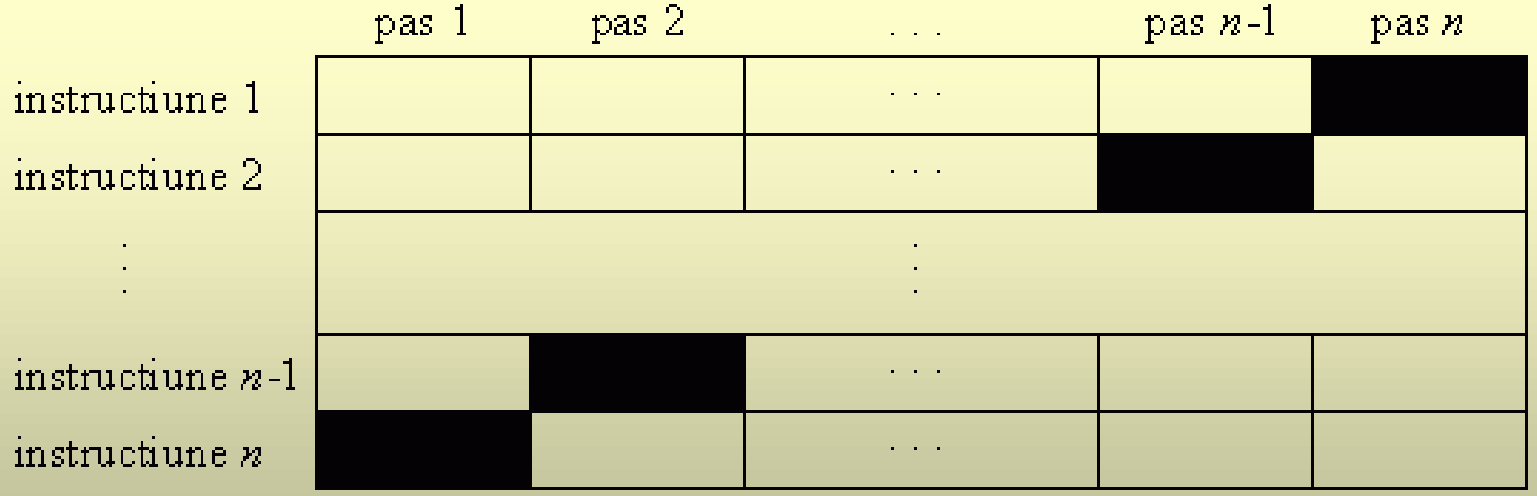
O primă implementare

Procesorul Intel 8086

- format din două unităţi
  - unitatea de interfaţă cu magistrala (BIU)
    - comunicarea cu exteriorul
  - unitatea de execuţie (EU)
    - execuţia propriu-zisă a operaţiilor
- BIU şi EU pot lucra în paralel

Principiul benzii de asamblare

- execuţia unei instrucţiuni - *n* paşi
- la un moment dat - *n* instrucţiuni în execuţie
- fiecare instrucţiune - în alt pas



Pipeline

- secvenţa paşilor (*stagii*) prin care trece execuţia unei instrucţiuni
- trecerea între două stagii - la fiecare ciclu de ceas
- cât durează până la terminarea unei instrucţiuni?
  - prima instrucţiune - *n* cicluri de ceas
  - următoarele instrucţiuni - câte 1 ciclu de ceas !

Performanţa unui pipeline

- rezultatul obţinut la fiecare stagiu trebuie reţinut
- regiştri de separaţie - plasaţi între stagii
- frecvenţa ceasului - dată de stagiul cel mai lung
- paşi mai simpli
  - număr de stagii mai mare
  - frecvenţă mai mare a ceasului

Execuţia unei instrucţiuni

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

Execuţia unei instrucţiuni

6. decodificarea instrucţiunii de către unitatea de control
7. citire operand din memorie
   - depunere adresă operand în MAR
   - comandă citire
   - preluare operand în MDR

7'. selecţie registru care conţine operandul

Execuţia unei instrucţiuni

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

Execuţia unei instrucţiuni

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 destinaţie

Evoluţie

- 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 instrucţiunile se pot executa în paralel
- dependenţă - o instrucţiune trebuie să aştepte terminarea alteia
- conflict în accesul la aceeaşi resursă

Parametri de performanţă

- latenţa (*latency*) - numărul de cicluri de ceas necesar pentru execuţia unei instrucţiuni
  - dat de numărul de stagii
- rata de execuţie (*throughput*) - numărul de instrucţiuni terminate pe ciclu de ceas
  - teoretic - egală cu 1
  - practic - mai mică (din cauza dependenţelor)

Tipuri de dependenţe

- structurale
- de date
- de control

Dependenţe structurale

- instrucţiuni aflate în stagii diferite au nevoie de aceeaşi componentă
- o singură instrucţiune poate folosi componenta la un moment dat
- celelate instrucţiuni care au nevoie de ea sunt blocate

Dependenţe structurale - exemple

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

Dependenţe de date

- o instrucţiune calculează un rezultat, alta îl foloseşte
- a doua instrucţiune are nevoie de rezultat înainte ca prima să-l obţină
- a doua instrucţiune este blocată

Dependenţe de date - exemplu

```
mov eax,7
sub eax,3
```

- prima instrucţiune: scrierea în `eax` - în ultimul stagiu
- a doua instrucţiune: utilizarea `eax` - în primele stagii (decodificare)
  - aşteaptă până când prima instrucţiune depune rezultatul în `eax`

Dependenţe de control (1)

Actualizarea valorii PC (uzual)

- adunarea la vechea valoare a dimensiunii codului instrucţiunii anterioare
- încărcarea unei valori noi - instrucţiuni de salt

Dependenţe de control (2)

Tipuri de instrucţiuni de salt

- necondiţionat
  - se face saltul întotdeauna
- condiţionat
  - se face saltul numai dacă este îndeplinită o anumită condiţie
  - altfel se continuă cu instrucţiunea următoare

Dependenţe de control (3)

Adresa de salt - moduri de exprimare

- valoare constantă
  - absolută
  - deplasament faţă de adresa instrucţiunii curente
- valoarea dintr-un registru
- valoarea dintr-o locaţie de memorie

Dependenţe de control (4)

Adresa de salt - exemple:

```
jmp 1594
jmp short -23
jmp eax
jmp dword ptr [esi]
```

Dependenţe de control (5)

Probleme

- calculul adresei de salt - în ultimele stagii de execuţie
- instrucţiunile următoare (multe!) au început deja execuţia
- dacă se face salt - efectele lor trebuie anulate

Dependenţe de control (6)

Probleme

- "golirea" pipeline-ului → pierdere de performanţă
  - operaţii complicate
  - durează mult până la terminarea primei instrucţiuni → scade rata de execuţie
- o instrucţiune din 7 (în medie) este de salt !

Tratarea dependenţelor

Soluţii

- staţionarea (*stall*)
- avansarea (*forwarding*)

Staţionarea (1)

- atunci când o instrucţiune foloseşte un rezultat care încă nu a fost calculat
- instrucţiunea "stă" (nu trece la etapa următoare)
- echivalent cu inserarea unei instrucţiuni care nu face nimic (*nop*)
- spunem că în pipeline a fost inserată o bulă (*bubble*)

Staţionarea (2)

- instrucţiunea trece mai departe când devine disponibil rezultatul de care are nevoie
- sunt necesare circuite de detecţie
- nu e o soluţie propriu-zisă
  - nu elimină efectiv dependenţa
  - asigură doar execuţia corectă a instrucţiunilor
  - dacă o instrucţiune staţionează, vor staţiona şi cele de după ea

Avansarea (1)

```
add dword ptr [eax],5
sub ecx,[eax]
```

- rezultatul adunării - calculat de ALU
- durează până când este scris la destinaţie
- instrucţiunea de scădere poate prelua rezultatul adunării direct de la ALU

Avansarea (2)

Avantaj

- reduce timpii de aşteptare

Dezavantaje

- necesită circuite suplimentare complexe
- trebuie considerate relaţiile între toate instrucţiunile aflate în execuţie (în pipeline)

IV.2. Multiplicarea unităţilor de execuţie

Unităţi superscalare

- ideea de bază - mai multe ALU
- se pot efectua mai multe calcule în paralel
- folosită împreună cu tehnica pipeline
- MAR şi MDR nu pot fi multiplicate
- cât de mult se pot multiplica ALU?
  - depinde de structura şi eficienţa pipeline

Unităţi superpipeline

- mai multe pipeline în acelaşi procesor
  - de obicei 2
- 2 (sau mai multe) instrucţiuni executate complet în paralel
- restricţii
  - accesele la memorie şi periferice - secvenţial
  - unele instrucţiuni pot fi executate de un singur pipeline

IV.3. Predicţia salturilor

Predicţie (1)

- rezolvarea dependenţelor de control
- ideea de bază - a "prezice" dacă un salt se execută sau nu
  - nu se aşteaptă terminarea instrucţiunii de salt
- predicţie corectă - fără blocaje în pipeline
- predicţie eronată - se execută instrucţiuni care nu trebuiau executate
  - efectul acestora trebuie anulat

Predicţie (2)

- spor de performanţă - cât mai multe predicţii corecte (nu neapărat 100%)
- o instrucţiune executată eronat produce efecte doar când rezultatul este scris la destinaţie
- rezultatele instrucţiunilor - memorate intern de procesor până când se verifică dacă predicţia a fost corectă

Scheme de predicţie

Tipuri de scheme

- statice
  - întotdeauna aceeaşi decizie
- dinamice
  - se adaptează în funcţie de comportarea programului

Scheme statice de predicţie (1)

1. Saltul nu se execută niciodată

- rata predicţiilor corecte ≈ 40%
- ciclurile de instrucţiuni
  - apar des în programe
  - salturi frecvente

Scheme statice de predicţie (2)

2. Saltul se execută întotdeauna

- rata predicţiilor corecte ≈ 60%
- ratări dese - structuri de tip *if*

Scheme statice de predicţie (3)

3. Salturile înapoi se execută întotdeauna, cele înainte niciodată

- combină variantele anterioare
- rată superioară a predicţiilor

Scheme dinamice de predicţie (1)

- procesorul reţine într-un tabel comportarea la salturile anterioare
  - salt executat/neexecutat
- un singur element pentru mai multe instrucţiuni de salt
  - tabel mai mic → economie de spaţiu

Scheme dinamice de predicţie (2)

Tipuri de predictori

- locali
  - reţin informaţii despre salturile individuale
- globali
  - iau în considerare corelaţiile dintre instrucţiunile de salt din acelaşi program
- micşti

Intel Pentium

- *Branch Target Buffer* (BTB)
  - cache asociativ pe 4 căi
  - 256 intrări
- Stările unei intrări
  - puternic lovit - se face salt
  - slab lovit - se face salt
  - slab nelovit - nu se face salt
  - puternic nelovit - nu se face salt

Implementarea BTB (1)

Memorarea şi evoluţia unei stări

- contor cu saturaţie pe 2 biţi
  - poate număra crescător şi descrescător
  - gama de valori - între 0 (00) şi 3 (11)
  - din stările extreme nu se poate trece mai departe (doar înapoi)
- la fiecare acces, starea se poate schimba
  - condiţia de salt este adevărată - incrementare
  - condiţia de salt este falsă - decrementare

Implementarea BTB (2)

- codificarea stărilor
  - puternic lovit - 11 (se face salt)
  - slab lovit - 10 (se face salt)
  - slab nelovit - 01 (nu se face salt)
  - puternic nelovit - 00 (nu se face salt)
- de ce 4 stări?
  - a doua şansă - comportament pe termen lung

Implementarea BTB (3)

stare curentă	stare următoare	
	condiţie salt adevărată	condiţie salt falsă
00	01	00
01	10	00
10	11	01
11	11	10

Utilizarea cache-ului în predicţie

Cache-ul de instrucţiuni

- reţine vechea comportare a unui salt
  - condiţie
  - adresă destinaţie
- *trace cache*
  - memorează instrucţiunile în ordinea în care sunt executate
  - nu în ordinea fizică