

# Jumătatea a 2-a a semestrului

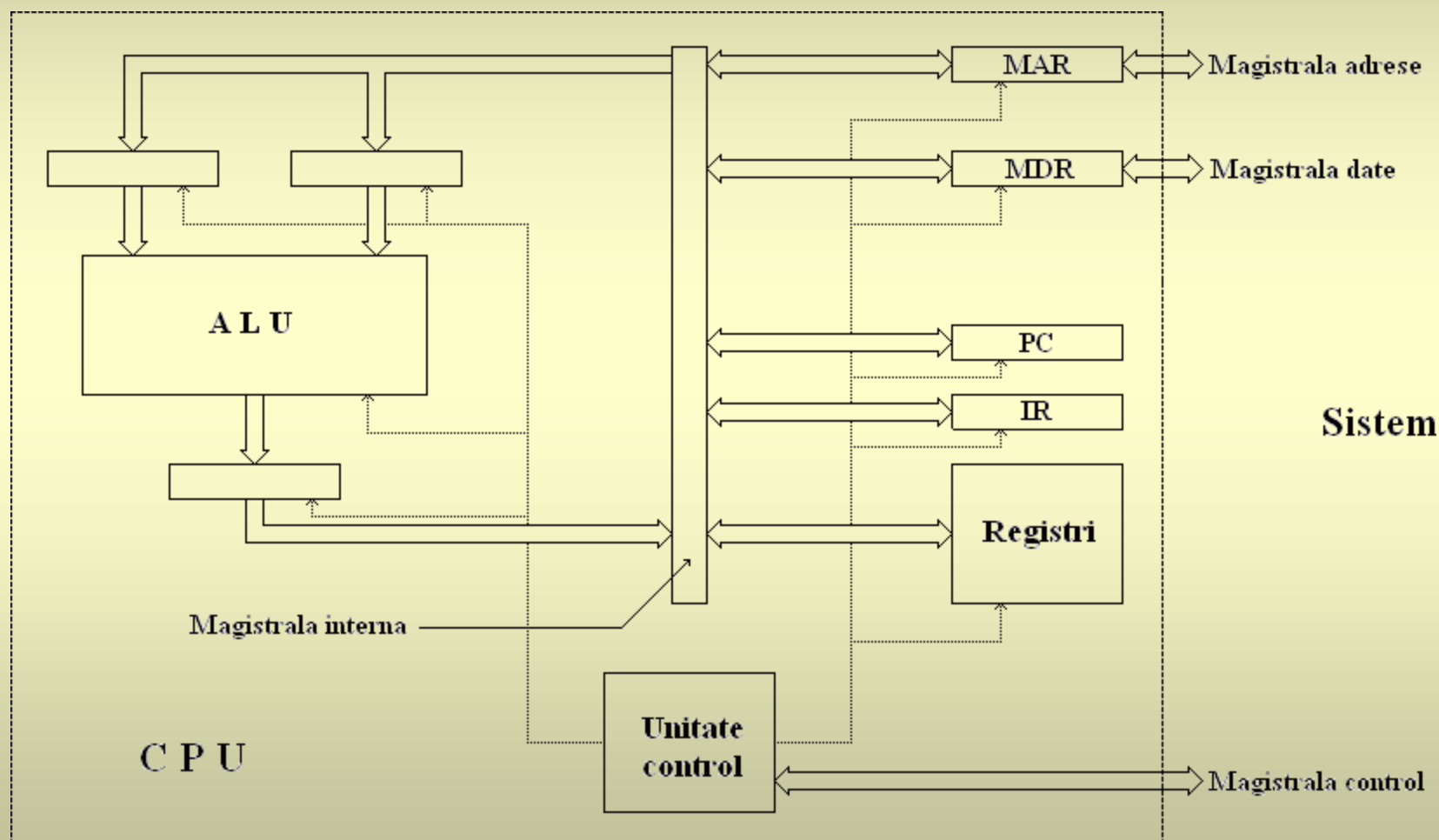
- curs
  - structura hardware a unui sistem de calcul
  - finalizare - test scris
    - condiție susținere test - cel mult 2 absențe la laborator
- laborator
  - limbaj de asamblare
  - finalizare - test practic

# Cuprins

- I. Unitatea centrală de procesare (CPU)
- II. Îmbunătățirea performanței CPU
- III. Dispozitivele periferice
- IV. Sistemul de întreruperi
- V. Sistemul de operare

# I. 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

# II. 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

# II.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 stagiou 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 !

# Tratarea dependențelor

## Soluții

- staționarea (*stall*)
- avansarea (*forwarding*)

## Staționarea (1)

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

## Staționarea (2)

- instrucțiunea trece mai departe când devine disponibil rezultatul de care are nevoie
- sunt necesare circuite de detecție
- nu e o soluție propriu-zisă
  - nu elimină efectiv dependența
  - asigură doar execuția corectă a instrucțiunilor
  - dacă o instrucțiune staționează, vor staționa și 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 destinație
- instrucțiunea de scădere poate prelua rezultatul adunării direct de la ALU

## Avansarea (2)

### Avantaj

- reduce timpii de așteptare

### Dezavantaje

- necesită circuite suplimentare complexe
- trebuie considerate relațiile între toate instrucțiunile aflate în execuție (în pipeline)

## II.2. Multiplicarea unităților de execuție



# Unități superscalare

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

# Unități superpipeline

- mai multe pipeline în același procesor
  - de obicei 2
- 2 (sau mai multe) instrucțiuni executate complet în paralel
- restricții
  - accesele la memorie și periferice - secvențial
  - unele instrucțiuni pot fi executate de un singur pipeline

## II.3. Predicția salturilor

## Predicție (1)

- rezolvarea dependențelor de control
- ideea de bază - a "prezice" dacă un salt se execută sau nu
  - nu se așteaptă terminarea instrucțiunii de salt
- predicție corectă - fără blocaje în pipeline
- predicție eronată - se execută instrucțiuni care nu trebuiau executate
  - efectul acestora trebuie anulat

## Predicție (2)

- spor de performanță - cât mai multe predicții corecte (nu neapărat 100%)
- o instrucțiune executată eronat produce efecte doar când rezultatul este scris la destinație
- rezultatele instrucțiunilor - memorate intern de procesor până când se verifică dacă predicția a fost corectă

# Scheme de predicție

## Tipuri de scheme

- statice
  - întotdeauna aceeași decizie
- dinamice
  - se adaptează în funcție de comportarea programului

# Scheme statice de predicție (1)

## 1. Saltul nu se execută niciodată

- rata predicțiilor corecte  $\approx 40\%$
- ciclurile de instrucțiuni
  - apar des în programe
  - salturi frecvente

# Scheme statice de predicție (2)

2. Saltul se execută întotdeauna

- rata predicțiilor corecte  $\approx 60\%$
- ratări dese - structuri de tip *if*



# Scheme statice de predicție (3)

3. Salturile înapoi se execută întotdeauna, cele înainte niciodată
- combină variantele anterioare
  - rată superioară a predicțiilor

# Scheme dinamice de predicție (1)

- procesorul reține într-un tabel comportarea la salturile anterioare
  - salt executat/neexecutat
- un singur element pentru mai multe instrucțiuni de salt
  - tabel mai mic → economie de spațiu

# Scheme dinamice de predicție (2)

## Tipuri de predictor

- locali
  - rețin informații despre salturile individuale
- globali
  - iau în considerare corelațiile dintre instrucțiunile de salt din același program
- micști

# 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 și evoluția unei stări

- contor cu saturație pe 2 biți
  - poate număra crescător și descrescător
  - gama de valori - între 0 (00) și 3 (11)
  - din stările extreme nu se poate trece mai departe (doar înapoi)
- la fiecare acces, starea se poate schimba
  - condiția de salt este adevărată - incrementare
  - condiția 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 șansă - comportament pe termen lung

## Implementarea BTB (3)

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

# Utilizarea cache-ului în predicție

## Cache-ul de instrucțiuni

- reține vechea comportare a unui salt
  - condiție
  - adresă destinație
- *trace cache*
  - memorează instrucțiunile în ordinea în care sunt executate
  - nu în ordinea fizică