Paralelism la nivel de instrucțiune

- Din 1985 toate procesoarele utilizează pipeline pentru creșterea performanțelor

- Suprapunerea de instrucțiuni – așa cum este și cazul pentru pipeline – se numește paralelism la nivel de instrucțiune

- Pentru a crea un paralelism cât mai eficient avem în esență două metode:
  - Hardware-ul poate determina și exploata paralelismul în mod dinamic;
  - Folosim tehnologia software pentru a determina paralelismul în mod static la momentul compilării
- CPI (cycles per instruction)
  - $Pipeline_{CPI} = CPI\ ideal\ pipe + stall - uri\ structurale + stall - uri\ hazarde\ de\ date + stall - uri\ control$

Tehnica	Efect de reducere pentru
Forwarding	Potențialele stall-uri pt hazardele de date
Salturi întârziate	Stall-urile pt hazardele de control
Programare dinamică de bază	Stall-urile pt hazardele de date din dependințe
Salturi întârziate	Stall-urile pt hazardele de control
Programare dinamică de bază	Stall-urile pt hazardele de date din dependințe

- Pentru MIPS tipic avem: media aparițiilor salturilor dinamice este de 15% - 25%

- Putem executa între 3 și 6 instrucțiuni MIPS până a avea un salt; pre puține instrucțiuni pentru a ne gândi la un paralelism → paralelism la nivel de blocuri

- ILP (Instruction Level Parallelism) poate fi crescut prin exploatarea parelismului din interiorul unui ciclu

- Adunarea a două array-uri de 1000 elemente în paralel:

```
for (i=1; i< 1000; i=i+1)
        x[i] = x[i] + y[i]
```

- Fiecare iterație poate fi suprapusă cu orice altă iterație – va exista o conversie de la acest tip de ciclu la un paralelism la nivelul instrucțiunii

- Se desface ciclul static de către compilator sau dinamic de către hardware. O altă metodă este aceea de a utiliza instrucțiuni vectori

- Codul prezentat poate fi executat în 4 intrucțiuni dacă avem procesoare care lucrează pe vectori: 2 instr de LOAD; 1 instr de ADD și 1 instr STORE BACK. Aceste instrucțiuni pot fi executate în pipe.

Dependințe de date și hazarde

- Este critic să determinăm cum o instrucțiune depinde de alta pentru determinarea paralelismului existent într-un program. Practic trebuie determinat câte instrucțiuni pot fi executate în paralel.

- Dacă 2 instrucțiuni sunt paralele => ele pot fi executate într-un pipe de adâncime arbitrară fără a cauza stall-uri

- Există 3 tipuri de dependințe:
  - Dependințe de date
  - Dependințe de nume
  - Dependințe de control

- O instrucțiune j prezintă o dependință de date față de instrucțiunea i dacă:
  - Instr i produce un rezultat care va fi utilizat de instr j **SAU**
  - Instr j prezintă de depință de date de instrucțiunea k, iar instrucțiunea k prezintă o depință de date față de instrucțiunea i

OBS !!!! ADD R1, R1, R1 nu se consideră dependință

- Dacă 2 instrucțiuni prezintă dependințe de date, ele nu pot fi executate simultan, sau nu pot fi complet suprapuse => hazarde de date

- Execuția paralelă pe un procesor cu interlocks

- Procesor fără interlocks, compilatorul nu poate programa instrucțiuni dependente astfel încât ele să fie complet suprapuse => execuție incorectă a programului

- Dependința de date implică :
  - Posibilitatea unui hazard
  - Ordinea de execuție
  - Cât de mult paralelism poate fi exploatat

- O dependință poate fi eliminată astfel:
  - Menținem dependința dar eliminăm hazardul
  - Eliminam dependința prin transformarea codului

Dependințe de nume

- O dependință de nume apare când 2 instrucțiuni utilizează același registru sau aceeași locație de memorie (nume), dar nu există un flux de date între instrucțiunile asociate cu acest nume

- O **antidependență** între instrucțiunile i și j apare când instrucțiunea j scrie un registru sau o locație de memorie care este citită de instr i => trebuie menținută ordinea

- O **dependință de ieșire** apare instr i și j scriu același registru sau aceeași locație de memorie =>trebuie menținută ordinea

- Soluția cea mai utilizată de soluționare este redenumirea. Redenumirea poate fi făcută static de către compilator sau dinamic de către hardware

Hazarde posibile

- Considerăm 2 instr i și j și instr i precede instr j
- RAW – instr j incearcă să citească o sursă înainte ca instr i să scrie în ea (dependință de date adeverată)
- WAW – instr j încearcă să scrie un operand înainte ca el să fie scris de instr i (dependință de ieșire)
- WAR – instr j încearcă să scrie o destinație înainte ca ea să fie citită de i (antidependință)
- RAR – nu este hazard