

# Számítógép architektúrák

A processzor teljesítmény növelése

# A mai témák

- CISC és RISC
- Párhuzamosságok
- Utasítás szintű párhuzamosságok
- Futószalag feldolgozás
- Többszörözés (szuperskalaritás)
- A függőségek kezelése
- A soros konzisztencia fenntartás

# A teljesítmény fokozás

- **Nem strukturális módszerek**
  - Órajel frekvencia növelés,
  - Instrukciók számának csökkentése (optimálás)
- **Strukturális módszerek**
  - Ciklusszám csökkentés: RISC architektúrákkal ...
  - Ciklusszám csökkentés párhuzamosításokkal

# CISC és RISC

- **CISC: Complex Instruction Set Computer**
- **RISC: Reduced Instruction Set Computer**
  - (Ezek CPU jellemzők)
- **Történelmileg előbb a CISC-ek**
  - minél többet bízz a hardverre,
  - bonyolult instrukciókat mikro-programokkal,
  - bonyolult instrukciókkal egyszerűbb a programozás (pl. PUSHALL),
  - bonyolult címzési módokat biztosítanak.
  - Az elgondolás nagyon jó, de ...

# A RISC gondolat

- Statisztikákból kiderült: gyakoribbak az egyszerű instrukciók.
- Akkor azokra “hegyezzük ki“ a CPU-t! (Ez az új gondolat!)
- Az egyszerű instrukciók azonos logikájúak:
  - egyszerűbb áramkörök, ezek gyorsabbak,
  - egyszerűbb, egységes dekódolás, ez is gyorsabb,
  - több regiszter lehet, ez is gyorsít,
  - egyszerűek, egyformák a címzési módok is.
- A bonyolultabb feladatokat viszont több instrukcióval. Lehet, hosszabb lesz a program.

# További előnyök

- Egyforma a ciklusidő (többnyire 1 utasítás/1 ciklus). Ez segíti a szupercsatornázást (lásd később).
- Az egyszerű áramkörök (nagyobb frekvenciát engednek) engedik a belső egységek többszörözését. Lehetséges a superskalaritás.
- Könnyebb a “spekulatív végrehajtás” is.
- Befér a tokba a gyorsítótár is, egyre nagyobb.
- Illesztés operációs rendszerhez, fordítóprogramhoz.

# Párhuzamosítások

- CPU-n belül:
  - Futószalag (pipe-line, csatorna) alkalmazása,
  - Többszörözésekkel: több instrukciót párhuzamosan dolgoznak fel (2-3 way: 2-3 utas)
- CPU-n kívül:
  - Fix feladat szétosztással (társprocesszorok)
    - lebegőpontos aritmetikára,
    - grafikára, képfeldolgozásra stb.
  - Változó feladat szétosztású multiprocesszoros rendszerek (dual/quad systems).

# A rendelkezésre álló és a hasznosított párhuzamosság

- A párhuzamosság egyik legjobb teljesítménynövelő technika
- A **rendelkezésre álló párhuzamosság**: ami a feladatból, a megoldásukból adódik, a probléma megoldásában benne van
- A **hasznosított párhuzamosság** : amit a végrehajtás során érvényesíteni tudunk.



# A rendelkezésre álló párhuzamosság

- Kétféle lehet: **funkcionális párhuzamosság** és **adat párhuzamosság**.
- A funkcionális ~ a **feladatmegoldás logikájából** jön. Belátható, hogy akár egy imperatív programban egyes szálak futhatnának párhuzamosan.  
A funkcionális ~ rendszerint szabálytalan (kivéve a ciklusszintű ~-ot).  
A párhuzamosság mértéke nem nagy (gyenge párhuzamosság).
- Az adat párhuzamosság olyan **adatszerkezetek használatából származik**, melyek elemein párhuzamosan lehet operációkat végezni.  
Többnyire szabályos ~.
- A párhuzamosság erős lehet (mértéke nagy, több-számjegyű).

# Az adatpárhuzamosság

- Adatpárhuzamos architektúra kell.
- Vektorprocesszorok.

# A rendelkezésre álló funkcionális párhuzamosság szintjei

- A „szemcsézettség” (granuláció) különböző lehet
  - Utasítás szintű párhuzamosság (finom szemcsézettség);
    - Instrukciókat párhuzamosan hajtunk végre
  - Ciklus szintű párhuzamosság (közepes szemcsézettség);
    - Egymást különben követő iterációkat párhuzamosan ...
  - Eljárás szintű párhuzamosság (közepes szemcsézettség);
    - Eljárásokat, függvényhívásokat párhuzamosan ... Szálak ...
  - Program szintű párhuzamosság (durva szemcsézettség).
    - Felhasználói szint. Processzek (taszkok) párhuzamosan.
    - Hasznosításukhoz az operációs rendszer segítsége kell. Több processzoros HW is.

# Az eljárás szintű párhuzamosság hasznosulása

- Eljárások párhuzamosan.
  - A szálak (threads) kezelése kell
  - Lehet fejlesztő rendszer segítségével,
  - lehet az operációs rendszer segítségével hasznosítani

## A ciklusszintű párhuzamosság hasznosulása

- Iterációk párhuzamosan.
  - A fordítóprogram segíti felfedezni ezt

# Utasítás szintű párhuzamosság hasznosulása

- Instrukciókat párhuzamosan hajtunk végre utasításszinten párhuzamos architektúrákkal (Instruction-Level Paralell, ILP processzorokkal)
  - Hagyományos „soros” programoknál ez rejtett (transzparens) marad: a processzor fedezi fel a programban rejlő párhuzamosítási lehetőséget.
- **Futószalag feldolgozással és**
- **processzoron belüli funkcionális elemek többszörözésével.**

# A futószalag (pipe-line, csővezeték, csatorna) feldolgozás

- Egyetlen instrukció feldolgozása is több fokozaton (stage) megy keresztül. Legalább:
  - instrukció felhozatal (fetch),
  - dekódolás (decode) (és utasítás “kibocsátás”),
  - a tényleges végrehajtás,
  - Az eredmény beírás.
- Az egyes fokozatokat más-más egységek végzik, párhuzamosan dolgozhatnak:
  - az  $i$ . instrukció végrehajtása során
  - dekódolható az  $i+1$ . instrukció,
  - felhozható az  $i+2$ . sít.

# A RISC előnyök itt érvényesülnek

- Egyforma instrukciók - egyforma fokozat idők.
- Egy ciklusra valóban kijöhet egy instrukció!

## Vannak gondok is

- Időzítési kockázat: egy instrukcióhoz kell az előző eredménye. Várakozni kell rá. Függőség.

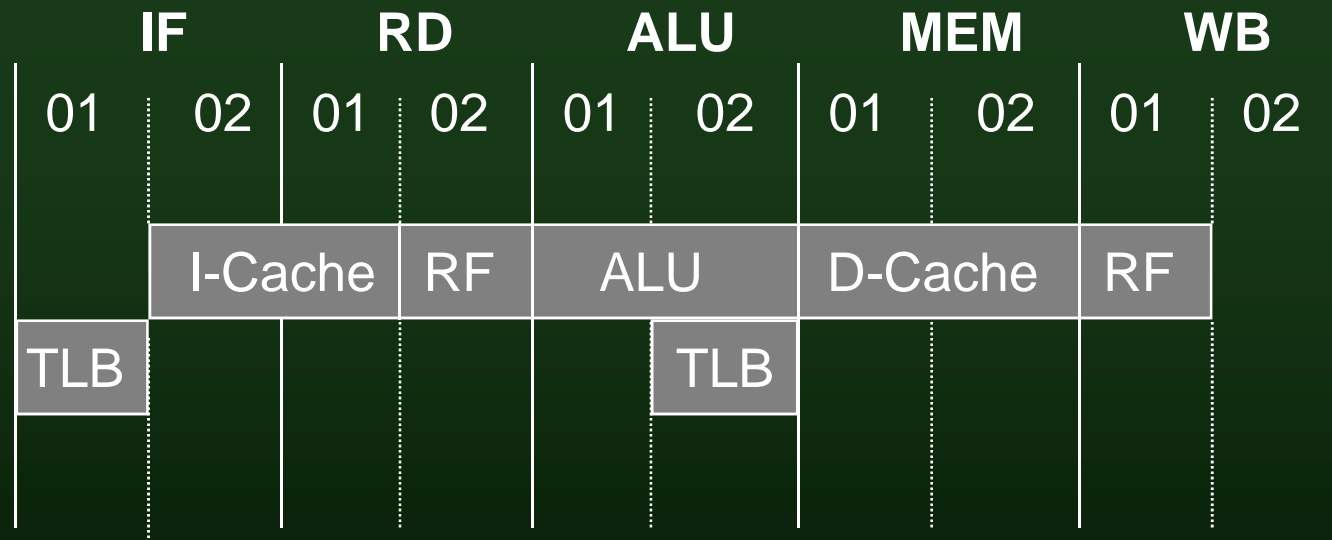
# Az R3000 szupercsöve

- Az instrukciók végrehajtását 5 fokozatra (stage) osztja. Minden fokozatot még 2 fázisra.
- 1 fokozat/1 ciklus
- A fokozatok:
  - Instrukció felhozatal IF
  - Olvasások, ellenőrzés RD
  - ALU operációk ALU
  - Adatmemória elérés MEM
  - Regiszter visszairás WB



# Az instrukció végrehajtása során használja

- a címleképzést segítő asszociatív tárat (TLB, Translation Lookaside Buffer),
- az instrukció gyorsítótárat (I-Cache),
- az adat gyorsítótárat (D-Cache),
- a regiszterfájlt (RF).



# A fokozatok, fázisok tevékenysége

IF	01	TLB-t használva virtuális címet fizikaira képez		
	02	Leképzett címet küldi az I-Cache-nek		
RD	01	Felhoz az I-Cache-ből, dekódol, ellenőriz		
	02	Regiszterfájl olvasás		Címszámítás
ALU	01	Aritmetikai számítás	Adat címszámítás	Döntés
	02		Adat cím leképzés	
MEM	01		Cím küldés D-Cache-nek	
	02		Adat mozgatása	
WB	01	Regiszterfájl írás	Regiszterfájl írás	

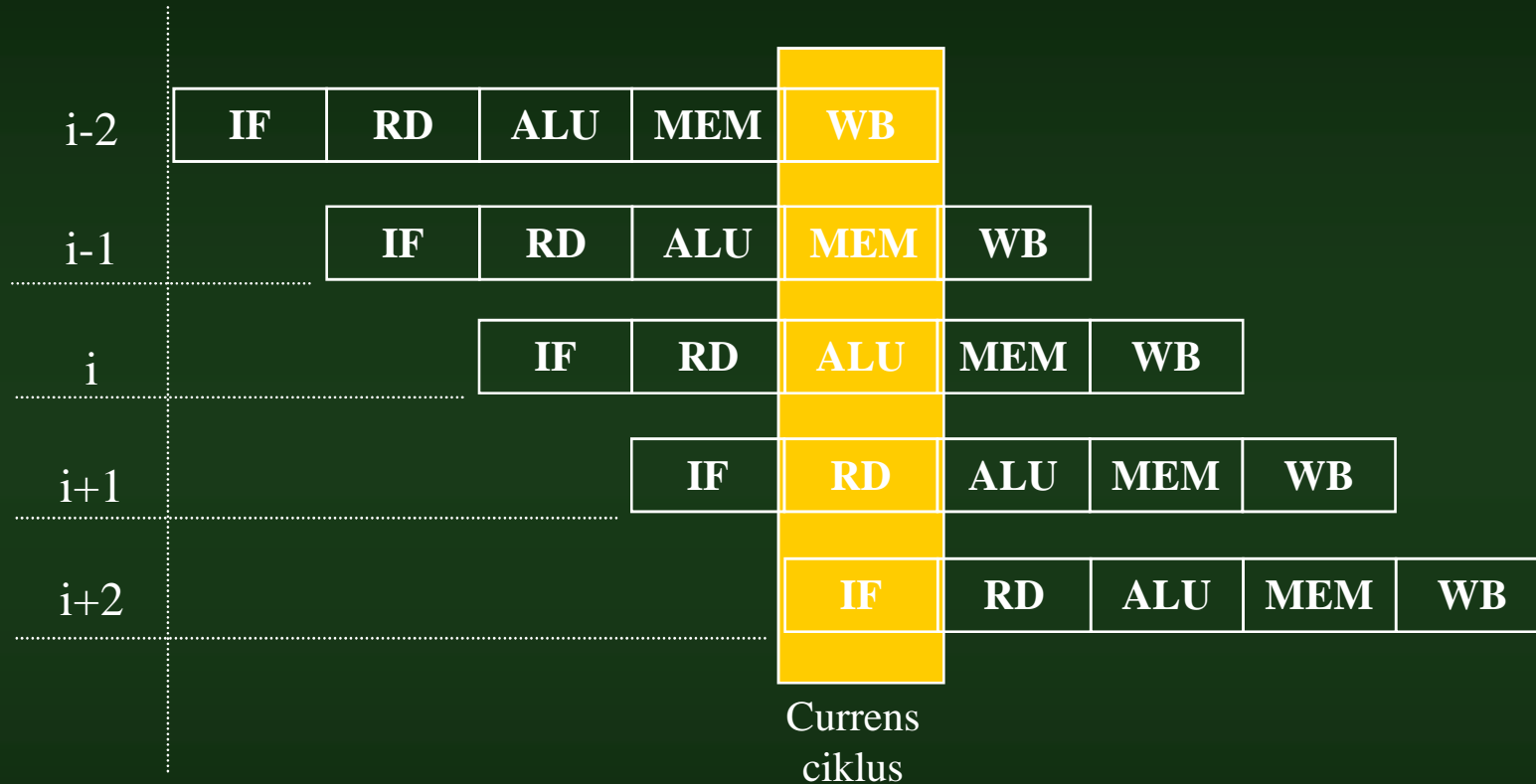
Aritmetikai  
instrukció

Tároló (Store)  
instrukció

Betöltő (Load)  
instrukció

Ugró  
instrukció

# Az 5 mélységű futószalag



# A PowerPC 601 futószalagjai

- **Elágazások**
  - Felhozatal + Dekódolás-kibocsátás-végrehajtás-becslés (2 fokozat)
- **Fixpontos aritmetika**
  - Felhozatal + Dekódolás-kibocsátás + Végrehajtás + Beírás (4 fokozat)
- **Load/Store instrukciók**
  - Felhozatal + Dekódolás-kibocsátás + Címképzés + Gyorsítótár + Beírás (5 fokozat)
- **Lebegőpontos aritmetika**
  - Felhozatal + Kibocsátás + Dekódolás + Végrehajtás1 + Végrahajtás2 + Beírás (6 fokozat)

# Megjegyzés

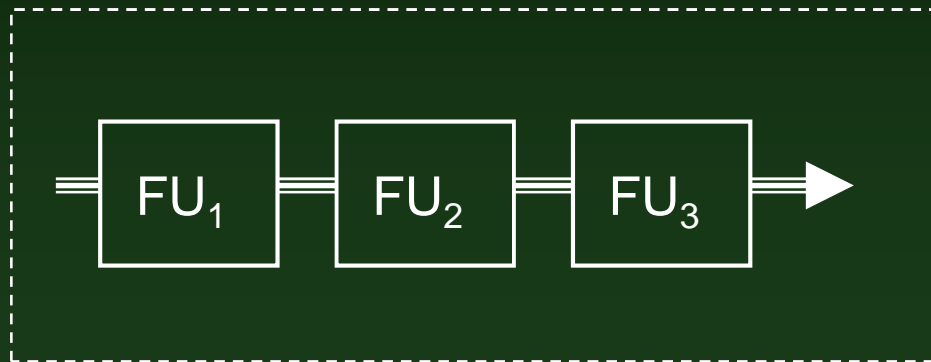
- **A supercsatornás CPU: nagyon sok fokozat**
- **A futószalag technika nemcsak processzoron belül (mikroszinten) alkalmazható**
- **Makroszinten (több processzor képezi csövet) is alkalmazzák**
- **Logikai szinten is (figyelj a burok csővezetékére)**
- **Az adatfolyam gépek is – ha úgy tetszik – futószalagot használnak**

# A funkcionális egységek többszörözése

- A funkcionális egységek többszörözése általános párhuzamosítási technika
- Az utasításszintű párhuzamosításban is lehet többszörözés:
  - Több dekódoló
  - Több végrehajtó egység (ALU/VE) stb.
- A többszörözés – természetes – makroszinten is
  - vö. MIMD párhuzamosság

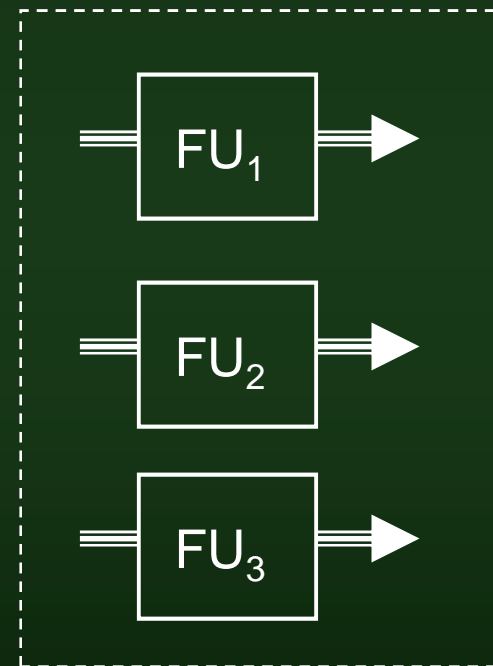
# Pipe-line versus többszörözés

Pipe-lined CPU



FU - funkcionális egység

Szuperskalár CPU



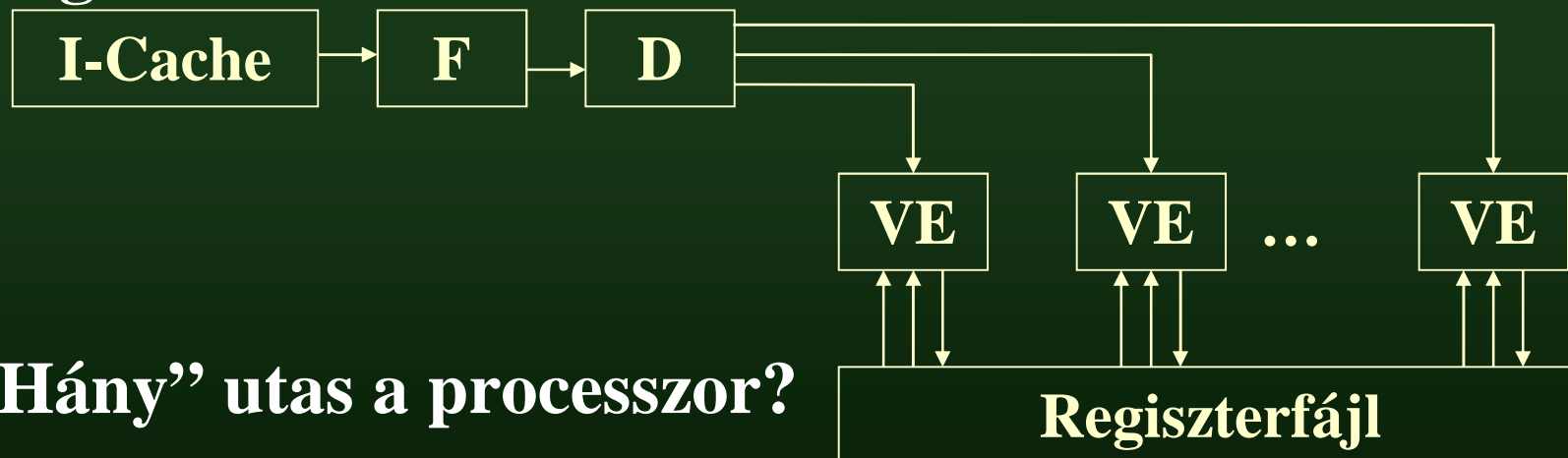
# Többszörözés processzoron belül

- **Egyik típus: VLIW (Very Long Instruction Word) architektúrák**
  - Pl. Trace, Intel IA64
  - Speciális fordító állítja elő a hosszú utasítást (pl. egy hosszú utasításban egy lebegőpontos és egy fixpontos ADD vagy MUL, esetleg még „szélesebb”)
  - Több ALU, ezeknek párhuzamosan a hosszú utasítást „szétbontva” bocsátja ki a dekódoló
  - Statikus a függőség feloldás (függőséget lásd később)
- **Másik: szuperskalár processzorok**



# Többszörözés processzoron belül, szuperskalár processzorok

- Egy ütemben (időablakban) több hagyományos instrukciót visznek be (hoznak be)
- A több hagyományos instrukciót a (esetleg több) dekódoló elemzi, és többszörös az utasítás kibocsátás
- Párhuzamosan több ALU (végrehajtó egység, VE) dolgozik



- „Hány” utas a processzor?

# A szuperskalár processzorok

- **Jellemző a dinamikus függőség-feloldás**
- **Természetes a futószalag technika is**
- **A jellegzetes feladatok a szuperskalár feldolgozásban**
  - **Párhuzamos dekódolás**
  - **Szuperskalár (többutas) utasítás kibocsátás**
  - **Párhuzamos végrehajtás**
  - **A végrehajtás soros konzisztenciájának megőrzése**
  - **A kivételkezelés soros konzisztenciájának megőrzése**

(Az ötlet már 1970-ben felmerült [Tjaden és Flynn]. Az első szuperskalár processzor 1982-83-ban jelen meg [IBM]. A szuperskalár elnevezést 1989-től használjuk.)

# Az utasítások közötti függőségek

- A függőségek a párhuzamos végrehajtás alapvető korlátját jelentik
- **Adatfüggőség:** egy instrukció az előző eredményét használja
- **Vezérlésfüggőség:** feltételes ugró (vezérlésátadó) utasítástól függenek a vezérlési ágak
- **Erőforrás függőség:** instrukciók ugyanazt az erőforrást igénylik (pl. valamilyen végrehajtó egységet, ALU-t)

# Az adatfüggőségek

- **Valódi függőség** a RAW (Read after Write) függőség

i1: load r1, a //  $r1 \leftarrow (a)$

i2: add r2, r1, r1 //  $r2 \leftarrow (r1) + (r1)$

- **Hamis függőség** a WAR (Write after Read) és a WAW (Write after Write) függőség,

i1: mul r1, r2, r3 //  $r1 \leftarrow (r2) * (r3)$

i2: add r2, r4, r5 //  $r2 \leftarrow (r4) + (r5)$

ui. **regiszter-átnevezéssel fel lehet oldani**

i1: mul r1, r2, r3 //  $r1 \leftarrow (r2) * (r3)$

i2: add r36, r4, r5 //  $r36 \leftarrow (r4) + (r5)$

# További adatfüggőség, függőségi gráf

- A **ciklusfüggőség** (ismétlési függőség)
  - $k$ -ad rendű ismétlési függőség esetén a szóban forgó utasítás a megelőző  $k$ . ciklusban kiszámított értékre hivatkozik
- Az **adat- és vezérlésfüggőségek** felfedezhetők és
- ún. **függőségi gráfban** rögzíthetők.
  - Irányított gráf: csomópontok az instrukciók, élek a függőségek
- A függőségi gráf segítheti az utasításütemezést a valódi függőségek feloldására.

# A függőségek észlelése és feloldása

- A függőségek észlelése és feloldása lehet statikus, vagy dinamikus
- **Statikus:** a fordítóprogram észleli és oldja fel: átrendezett instrukciósorozatot generál (kódoptimalizálás)
  - A VLIW (Very Long Instruction Word) processzorok függőségmentes instrukciósorozatot várnak
  - Szuperskalár és futószalag processzorokra is lehet
- **Dinamikus:** a függőségek észlelése és kezelése a processzor feladata
  - A legtöbb szuperskalár processzor ilyen

# A dinamikus függőség-kezelés

- A processzor két csúszó ablakot alkalmaz
  - **Kibocsátási ablakot**, melyben
    - azok az instrukciók vannak, melyeket a következő ciklusban kibocsátana;
  - **Végrehajtási ablakot**, melyben
    - Az instrukciók még végrehajtás alatt vannak (eredményük még nincs meg).
- Minden ütemben vizsgálja, a kibocsátási ablakban van-e
  - a másik ablakbeli instrukciótól függő instrukció (még nincs kész),
  - Ill. a kibocsátási ablak instrukciói között van-e függőség.
  - Ezekről és a kibocsátási politikától is függ a kibocsátás

# Kibocsátási politikák

- **Blokkolós kibocsátás**
  - Addig blokkol instrukciót, míg a függősége megszűnik
- **Sorrenden kívüli kibocsátás**
  - blokkolt utáni függetleneket sorrenden kívül
- **Spekulatív kibocsátás**
  - Vezérlési függőség kezelésére mindkét ágat kibocsátja



# A spekulatív végrehajtás

- Egy-egy instrukciót (operációt, elemi instrukciót) a lehető leghamarabb végrehajtának, és függetlenül attól, hogy eredményére szükség lesz-e vagy sem... (amint lehet + függetlenül a szükségességétől. Szükségtelensége esetén gondoskodnak arról, hogy ne okozzon hibát!).
- A „betöltő” (load) instrukciókat (elég gyakoriak és elég költségesek) pl. célszerű spekulatívan végrehajtani (minél előbb és mindenképpen).

# A soros konzisztencia fenntartása

- **Konzisztencia itt: ellentmondás mentesség**
  - Ha „felborul” a sorrend a statikus v. dinamikus függőségkezelés, kódoptimalizálás miatt? A programozó szándéka? A logikai integritás?
- Párhuzamos végrehajtásnál is fenn kell tartani a soros végrehajtás logikáját!
- **A soros konzisztencia lehet**
  - Utasítás feldolgozás soros konzisztenciája,
    - **Processzor konzisztencia** (instrukciók sorrendje)
    - **Memória konzisztencia** (memória hozzáférések sorrendje)
  - Kivétel feldolgozás soros konzisztenciája

# A processzor konzisztencia

- Az instrukciók befejezésének a sorrendje a kérdés
- **Gyenge konzisztencia** estén a befejezési sorrend eltérhet a programozottól, de ez integritási hibát nem okozhat.
- **Erős konzisztencia** esetén a befejezési sorrend szigorúan a programozói sorrend
  - Legtöbbször ezt egy **átrendező puffer (ROB, ReOrder Buffer)** alkalmazásával történik

# A ROB

- **A ROB egy kezdet- és végmutatókkal rendelkező körpuffer.** Kezdetmutató jelzi a következő szabad bejegyzés helyét. Az egyes bejegyzésekben nyilvántartják a bejegyzéshez tartozó instrukció állapotát (kibocsátott, végrehajtás alatti, befejeződött). **Egy instrukció csak akkor írható ki, ha befejeződött, és minden előtte lévő már ki van írva.**
- A ROB segíti a spekulatív végrehajtásból adódó instrukciók érvényesítését (ha kiderül, hogy tényleg kell), és nem érvényesítését is (további állapotjelzővel)

# A megszakításkezelési sorrend

- Ezt a konzisztenciát is segíti a ROB
- A megszakítások, kivételek kérését akkor fogadja el a processzor, mikor a ROB-ból kiírják (érvényesítik) az instrukciót

# A memória konzisztencia

- Gyenge memória konzisztencia esetén eltérés lehet a programozott sorrendtől
  - ahol a programozói szándék nem sérül,
  - A be-ki mozgó (load-store) instrukcióknál is lehetséges a spekulatív végrehajtás: általában a load-ok előzhetnek store-okat

# Számítógép architektúrák

A processzor teljesítmény növelése  
VÉGE