HÁLÓZATI RENDSZEREK ÉS SZOLGÁLTATÁSOK TANSZÉK

MŰEGYETEM 1782

Budapest,
2025.04.29.

SZÁMÍTÓGÉP ARCHITEKTÚRÁK

Az utasítás-pipeline szélesítése

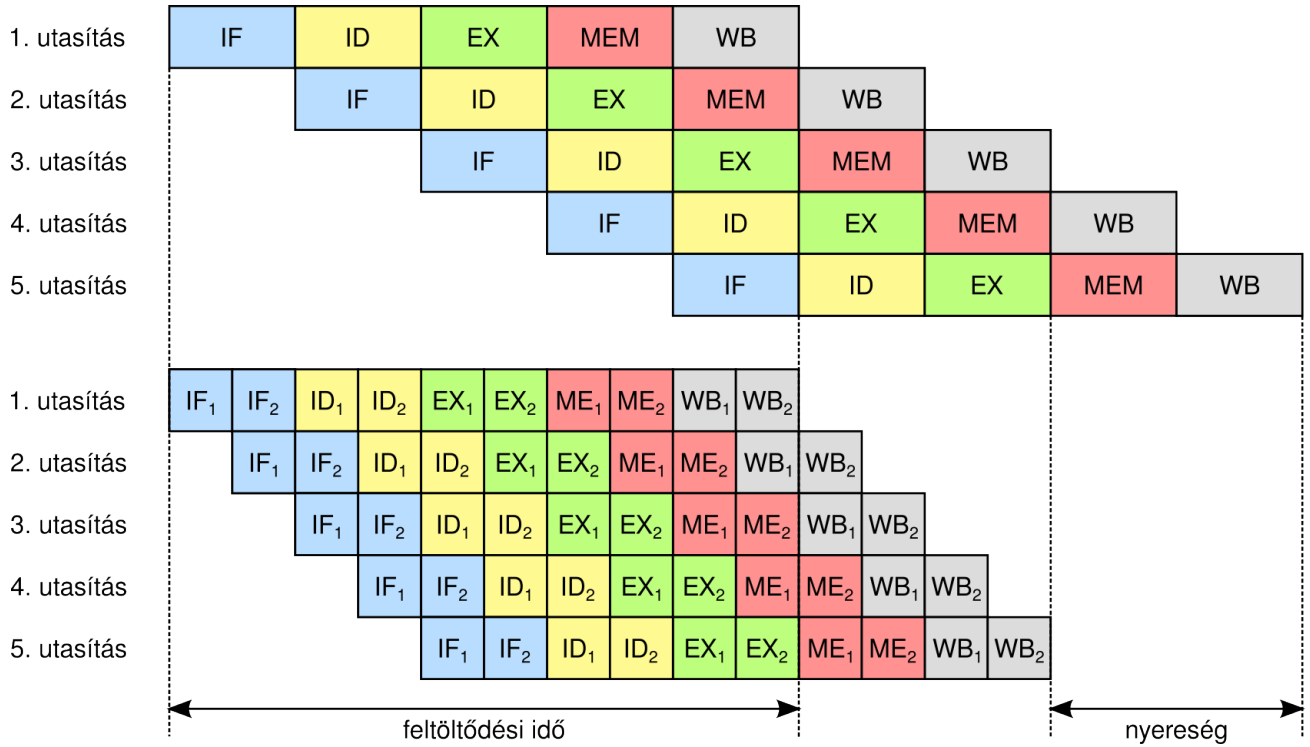
Horváth Gábor, Belső Zoltán

BME Hálózati Rendszerek és Szolgáltatások Tanszék
ghorvath@hit.bme.hu, belso@hit.bme.hu

UTASÍTÁSFELDOLGOZÁS GYORSÍTÁSA

1. Lehetőség: Mélyebb pipeline

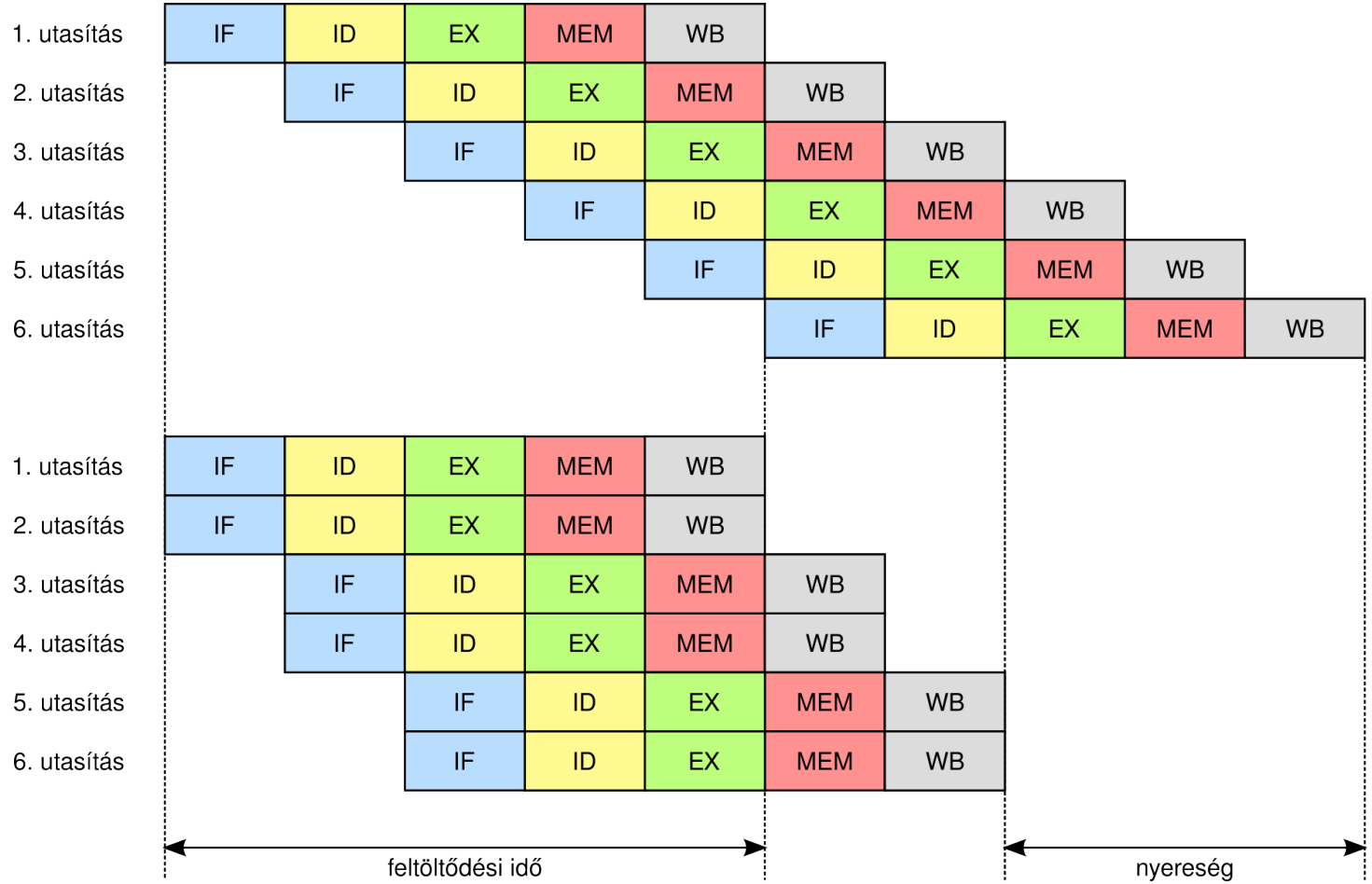
- Ciklusidőt felére csökkentjük (órajel 2x)
- Ehhez minden fázist két részre kell bontanunk
- Megdupláztuk az átviteli sebességet!



UTASÍTÁSFELDOLGOZÁS GYORSÍTÁSA

2. Lehetőség: Szélesebb pipeline

- Minden fázisban több utasítást dolgozunk fel



ÉRTÉKELÉS

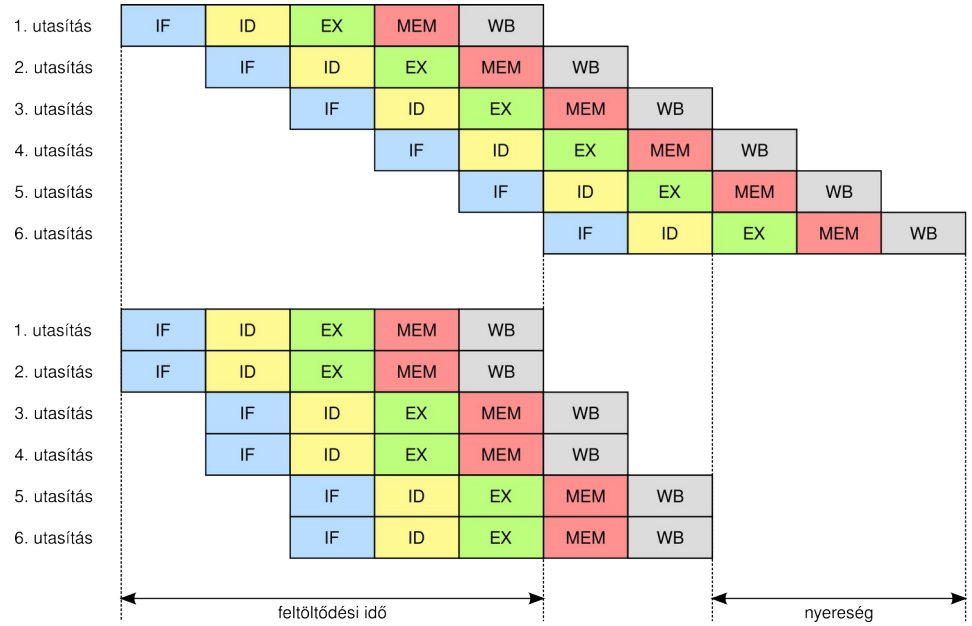
- Mélység: k-szoros, szélesség: m-szeres
  → Elméletileg m*k-szoros gyorsulás
- Gyakorlatilag több korlát:
  - m*k nagy: sok utasítás fut egyszerre → sok az egymásrahatás
    → sokszor kell megállni a feloldáshoz → romló hatékonyság
  - széles pipeline → m-el négyzetesen növő forwarding utak
  - k hiába nagy → órajel nem lehet tetszőlegesen nagy
    - pl. pipeline regiszter írás/olvasás bele kell férjen
  - k nagy: rossz spekulatív döntések drasztikus hatása
    - pl. rossz elágazásbecsléskor sor tévesen elkezdett utasítás
- Tipikus értékek:
  - Mélység: 5-30
  - Szélesség: 1-6

PÉLDÁK

	Mélység	Szélesség
Pentium	5	1
Pentium Pro	14	1-3 (1 bármilyen + 2 egyszerű)
Pentium 4 Prescott	31	3
Intel Core	14	4
Intel Core i7 Nehalem	16	4
Intel Core i7 Kaby Lake	14-19	4
Intel Atom	16	2
Alpha 21264	7	4
ARM Cortex A55	8-10	2
ARM Cortex A75	11-12	3
APPLE A10	?	6
POWER9	12-16	6

A PIPELINE SZÉLESÍTÉSE

- Hatékonyság kulcsa:
  - Elég sok független utasítás
- Ki válogatja össze?
  - Fordító:
    - Statikus ütemezés
    - VLIW/EPIC
  - Processzor:
    - Dinamikus ütemezés
    - Szuperskalár processzor



UTASÍTÁSOK ÜTEMEZÉSE

- A tisztán dinamikus és tisztán statikus megoldás között vannak köztes lehetőségek is

	Párhuzamosan végrehajtható csoportok kiválasztása	Hozzárendelés a műveleti egységekhez	Végrehajtás idejének meghatározása
Szuperskalár	Hardver	Hardver	Hardver
EPIC	Fordító	Hardver	Hardver
Dinamikus VLIW	Fordító	Fordító	Hardver
VLIW	Fordító	Fordító	Fordító

Széles pipeline dinamikus ütemezéssel

SZUPERSKALÁR PROCESSZOROK

- A processzor több utasítást is le tud hívni egyszerre
- A processzor végzi a
  - Párhuzamosan végrehajtható utasítások kiválogatását
  - Az utasítások végrehajtó egységhez rendelését
  - A függőségi analízist (mikor hajtható végre egy utasítás)
- Ha minden fázisban m új utasítás végrehajtása kezdődik meg

  → ***m-utas szuperskalár processzor***
- Kétféle megoldás:
  - In-order:
    - végrehajtási sorrend: program követése
  - Out-of-order:
    - átrendezi az utasításokat, hogy gyorsabb legyen
    - program szemantikáját megtartja

IN-ORDER VS. OUT-OF-ORDER

- Példa:

```
i1: R1 ← R2 + R3
i2: R4 ← R1 - R5
i3: R7 ← R8 - R9
i4: R0 ← R2 - R3
```

- 2-utas esetben:

In-order:

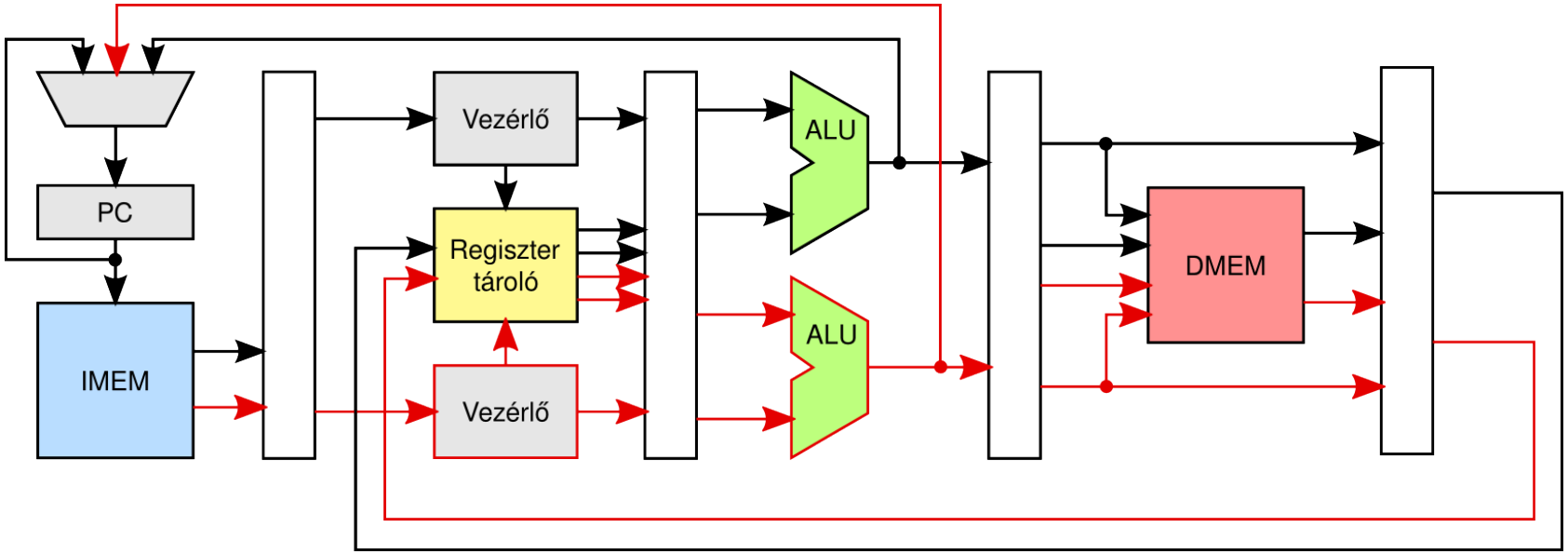
Órajel	Utasítások
1:	i1
2:	i2, i3
3:	i4

Out-of-order:

Órajel	Utasítások
1:	i1, i3
2:	i2, i4

IN-ORDER SZUPERSKALÁR PIPELINE

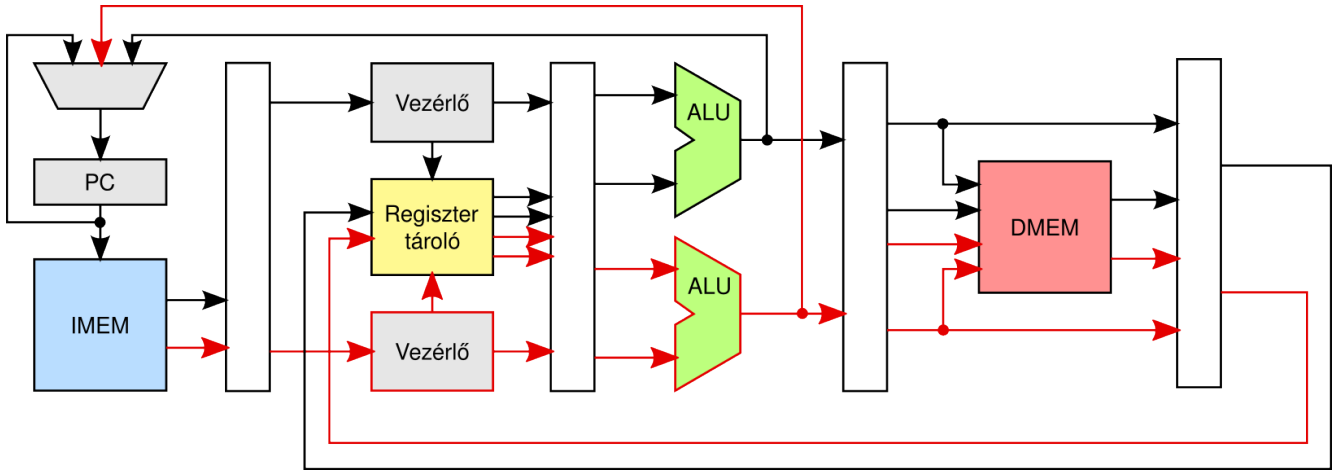
Hagyományos egy-utas nem szuperskalár processzor



Két-utas in-order szuperskalár processzor

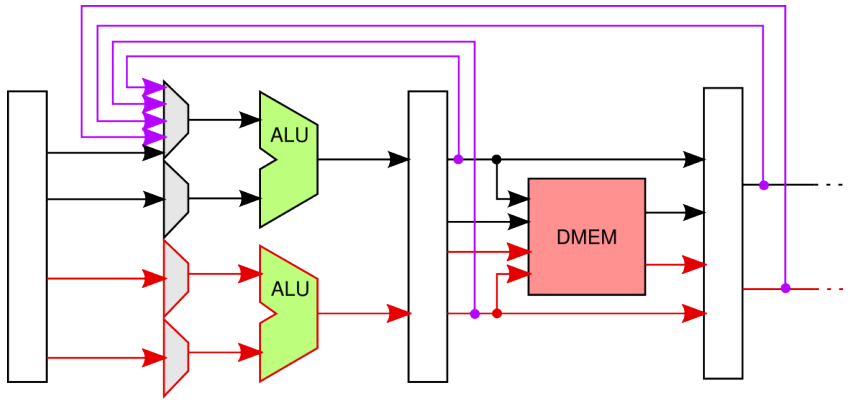
FÁZISOK

- **IF fázis:**
  - Most *m* utasítást kell lehívnia
    - Ha az *m* utasítás sorban jön, semmi gond
    - Ugrás esetén hatékonyságcsökkenés
- **ID fázis:**
  - ALU vezérlőjelek előállítása: nem nehezebb *m*-re mint 1-re
  - Operandusok kiolvasása: több portos regiszter tároló kell
  - Adatfüggőségek detektálása: m-el négyzetesen bonyolódik



FÁZISOK

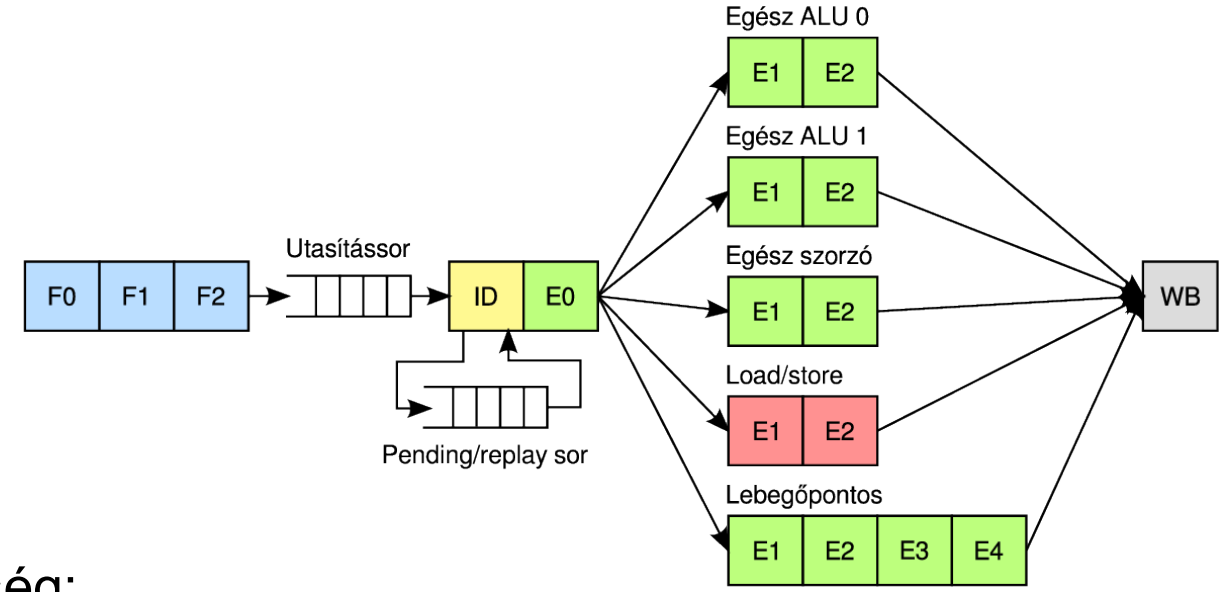
- **EX fázis:**
  - *m* db ALU beépítése nem gond
  - … de forwarding utak: m^2 van belőle
  - Egyenként 32 – 64 bites sínek → nem OK!



- **MEM fázis:**
  - Több portos adat cache kell
  - … vagy csak 1 memóriaműveletet engedünk egyszerre
- **WB fázis:**
  - Több portos regiszter tároló kell

AZ ARM CORTEX A53 PIPELINE-JA

- 2-utas in-order 8 fázisú futószalag



- IF egység:
  - F0: utasításszámláló inkrementálás, ugráskor számolás
  - F1: cím kiadása az utasításcache-nek.
  - F2: Megjön az adat az utasításcache-től → utasítássorba
- Cortex A55: ugyanez, külön Load és Store egységgel

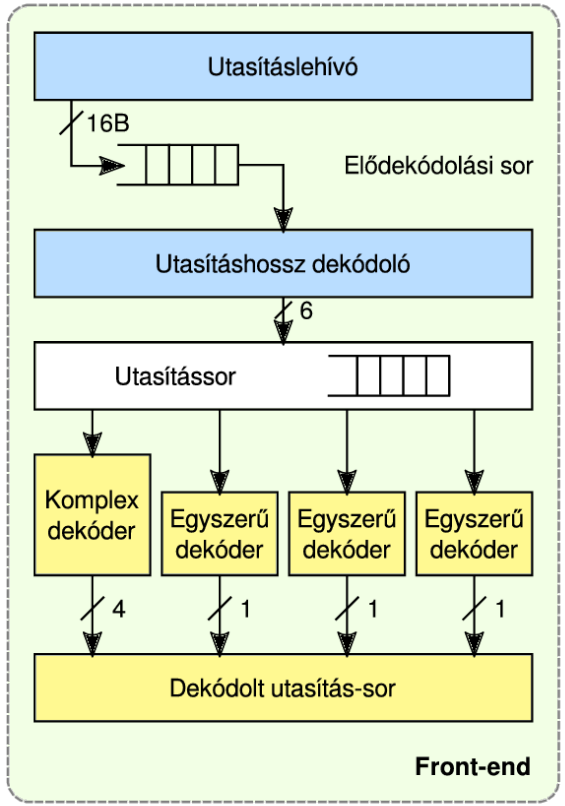
OUT-OF-ORDER SZUPERSKALÁR PIPELINE

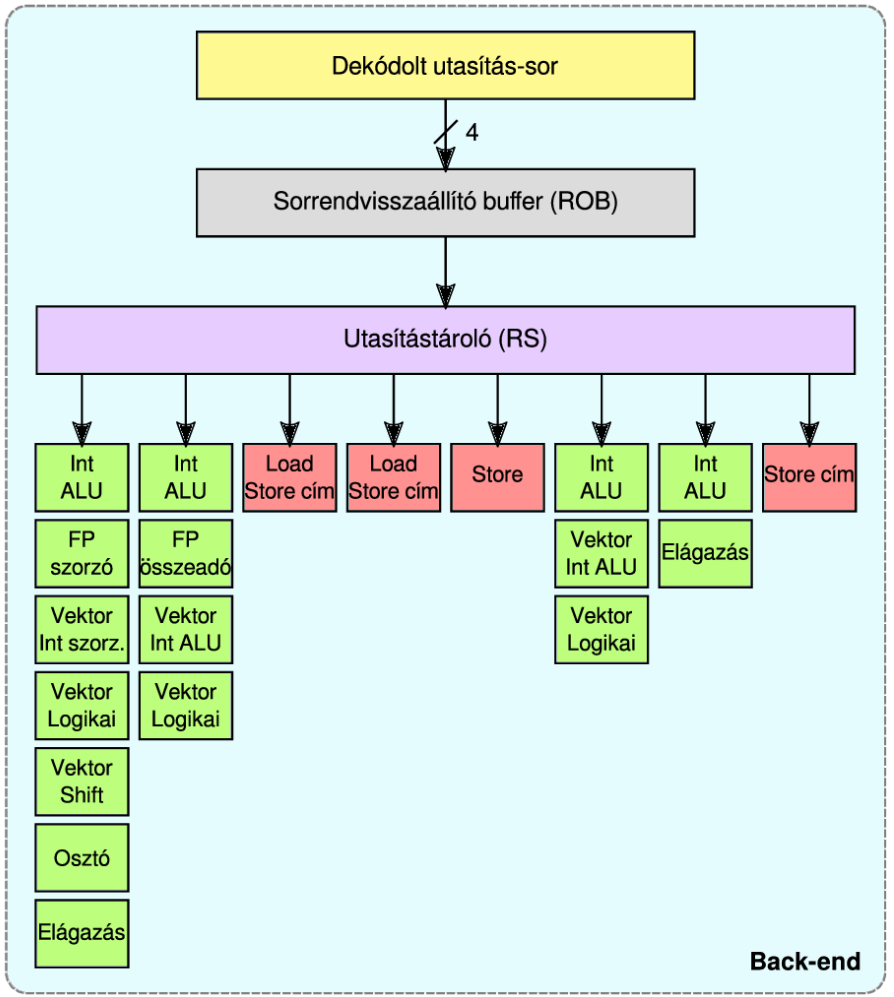
- Egy utas esetet már láttuk
  - Nyilvántartásokat kell vezetni:
    - Az utasítások végrehajtási állapotáról
    - Műveleti egységek foglaltságáról
  - Döntéseket kell hozni
    - Utasítások műveleti egységekhez rendelése

  → Bonyolult processzor
- Nem nehéz több utasra kiterjeszteni!
  - Több IF és DS egység kell
  - Egy ciklusban több utasítás lép be a tárolóba
  - Egy ciklusban több utasítás végrehajtása is elindulhat

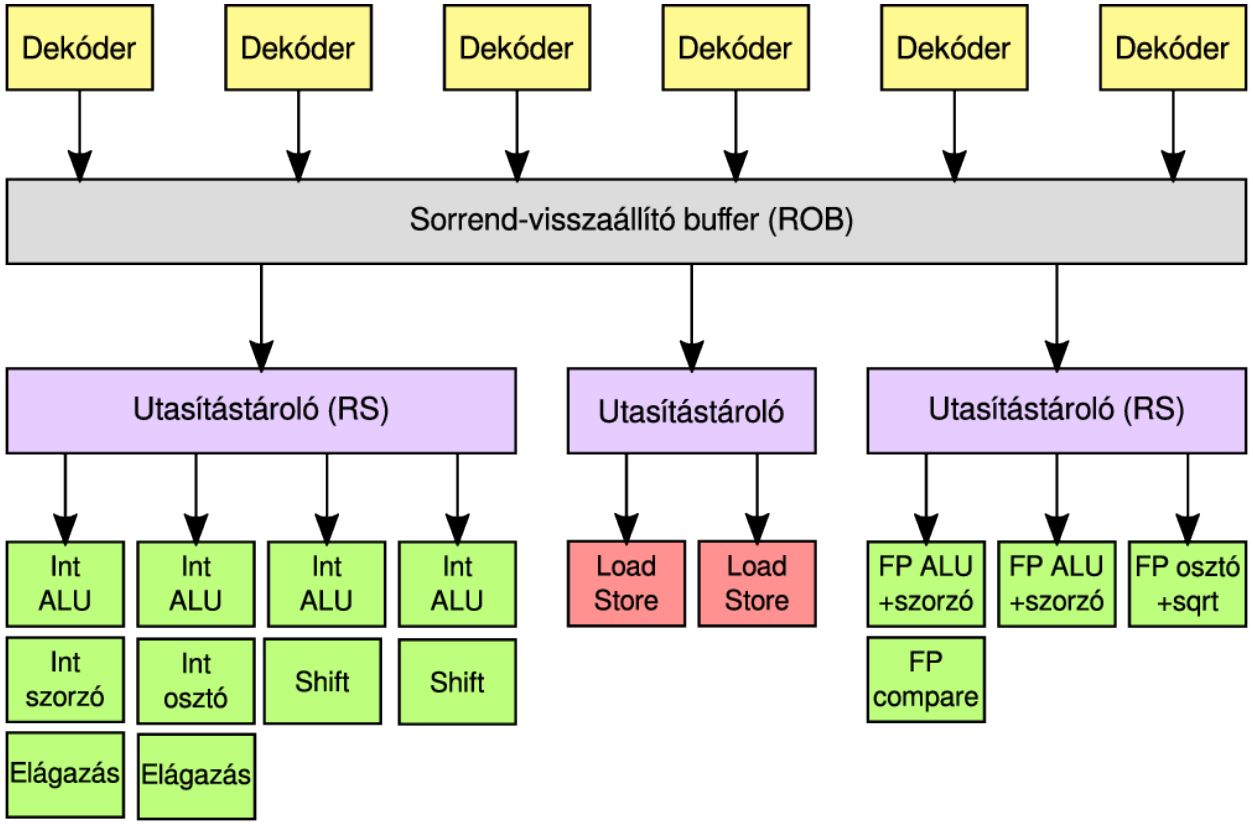
AZ INTEL HASWELL PIPELINE-JA

- Széles, sorrenden kívüli szuperskalár





AZ APPLE CYCLONE PIPELINE-JA



SZUPERSKALÁR ÖSSZEFOGLALÓ

- In-order: egyszerűbb
- Out-of-order: bonyolultabb
  - Adatáramlásos elven történő utasításvégrehajtás
  - Táblázatok tartják nyilván a műveleti egységek, utasítások és regiszterek állapotát
- Miért szereti a programozó az out-of-order processzort?
  - Mert nem kell kézzel optimalizálni az utasítás sorrendet (lásd: gyakorlat)
  - A CPU automatikusan összeszedi és végrehajtja, amit végre lehet

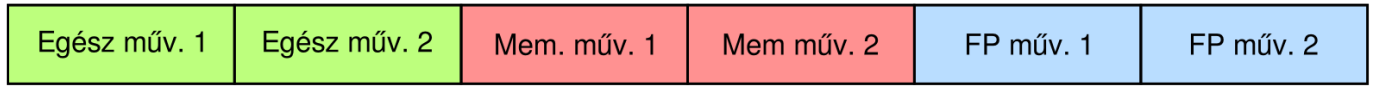
Széles pipeline statikus ütemezéssel

A MÁSIK VÉGLET

- Szuperskalár: a CPU keresgéli a független, párhuzamosan végrehajtható utasításokat
- A fordítóprogram is megteheti!
- Sőt, jobban! Több utasítást lát!
- Kapjunk vérszemet:
  - Csináljon mindent a fordító!
    - Párhuzamosan végrehajtható utasítások gyűjtése
    - Utasítások műveleti egységhez rendelése
    - Egymásrahatások detektálása és feloldása

A VLIW ARCHITEKTÚRA

- VLIW = Very Long Instruction Word
- 1 pipeline, ***utasításcsoport***okon dolgozik



- Csoportok szerepe:
  - Független utasítások kijelölése
  - Utasítások műveleti egységhez rendelése
- Nem használt pozíciókban: NOP
- Döbbenet:
  - ***A VLIW processzorok nem foglalkoznak egymásrahatásokkal!***
- Mi van, ha adat-egymásrahatás van, és szünetet kellene beiktatni?
  - Processzor fütyül rá
  - Vegye észre a fordító!
  - Iktasson be egy csupa NOP csoportot

A VLIW ARCHITEKTÚRA

- Példa:
- Késleltetések: egész – 1, memória – 3, lebegőpontos – 4

```
i1: R3 ← MEM[R1+0]
i2: R4 ← MEM[R1+4]
i3: D1 ← MEM[R1+8]
i4: R5 ← R3 + R4
i5: R6 ← R3 - R4
i6: D2 ← D1 * D1
i7: MEM[R2+0] ← R5
i8: MEM[R2+4] ← R6
i9: MEM[R2+8] ← D2
```

	Egész 1.	Egész 2.	Mem 1.	Mem 2.	FP 1.	FP 2.
1.	NOP	NOP	i1	i2	NOP	NOP
2.	NOP	NOP	i3	NOP	NOP	NOP
3.	NOP	NOP	NOP	NOP	NOP	NOP
4.	i4	i5	NOP	NOP	NOP	NOP
5.	NOP	NOP	i7	i8	i6	NOP
6.	NOP	NOP	NOP	NOP	NOP	NOP
7.	NOP	NOP	NOP	NOP	NOP	NOP
8.	NOP	NOP	NOP	NOP	NOP	NOP
9.	NOP	NOP	i9	NOP	NOP	NOP

A VLIW ARCHITEKTÚRA

- Tipikus utasításcsoport-méretek:
  - 3-4, extrém esetben akár 28 utasítás
- Minden nehezet a fordító csinál
  → processzornak alig marad dolga!
- VLIW processzorok tipikus alkalmazása:
  - Olcsó, kis fogyasztású beágyazott rendszerek
  - Ha fontos a kiszámítható, spekuláció és predikciómentes működés: DSP (pl. TMS320C6x – 8 utasítás/csoport)
  - Grafikus processzorok: sok egyszerű feldolgozóegység kell pl. AMD a Radeon HD néhány generációjában: VLIW3 ill. VLIW4
- Hátrányok:
  - A program csak azon a processzoron fut, amire lefordították
  - **Nagy probléma a transzparens cache megvalósítása!**
    - Memóriaműveletek sebessége nem lesz állandó
    - Fordítóprogram nem tudja, hogyan ütemezzen
      → VLIW processzorokban nincs cache
  - Programok mérete igen nagy a sok NOP miatt

A DINAMIKUS VLIW ÉS EPIC ARCHITEKTÚRÁK

- **Dinamikus VLIW architektúra**
  - A fordító csak csoportosít
  - A processzor ütemez
    - Észleli az egymásrahatásokat, szüneteket tud beiktatni
  - Előnyök:
    - Lehet cache-t csinálni!
- **EPIC**
  - VLIW-ben az utasítás csoportbeli helye a műveleti egységet is kijelölte:

  - EPIC-ben nem. Utasításcsoport: párhuzamosan végrehajtható utasítások halmaza

  - Lehet EPIC szuperskalár processzort készíteni, több műveleti egységgel
    → A CPU több csoport egyidejű végrehajtását végzi
  - Lehet csoportokat láncolni, így jelezheti a fordító, hogy nem 3, hanem 6, 9, stb. független utasítást talált

HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK