ÓBUDAI EGYETEM

SZÁMÍTÓGÉP ARCHITEKTÚRÁK ALAPJAI

13-14. előadás

Durczy Levente

2025/2026/1

Futószalag processzorok:

Bevezetés:

A futószalag (pipeline) végrehajtás lényege, hogy az utasítást több részre bontjuk (általában fetch, decode, execute, writeback), majd ezeket a részeket külön, egymással párhuzamosan hajtjuk végre.

Elég gondolni az autógyártásra, ahol különböző munkaállomásokon időben párhuzamosan más-más gyártási műveletet hajtanak végre (Henry Ford). Az effajta párhuzamosságot hívjuk időbeli párhuzamosságnak. Az utasítások n részre osztásával elméletileg a sebesség n-szeresére növekszik, amennyiben minden rész azonos idő alatt hajtódik végre.

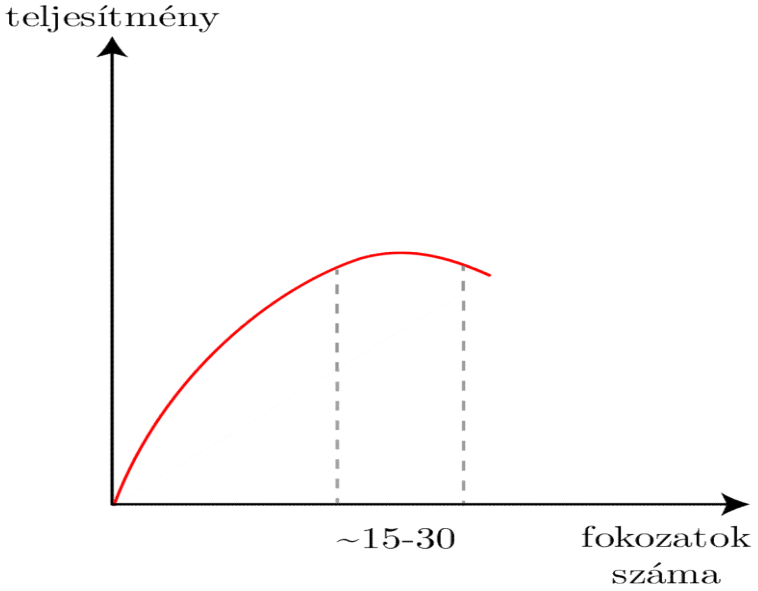
Instruction$_1$	**F (t_1)**	**D (t_2)**	**E (t_3)**	**W/B (t_4)**	
Instruction$_2$		F (t_1)	D (t_2)	E (t_3)	W/B (t_4)

ahol t1=t2=t3=t4

A végrehajtást gátolják a függőségek (adat, vezérlés, erőforrás), ezért a valóságban nem mindig arányos a fokozatok számának növelésével járó gyorsulás növekedés.

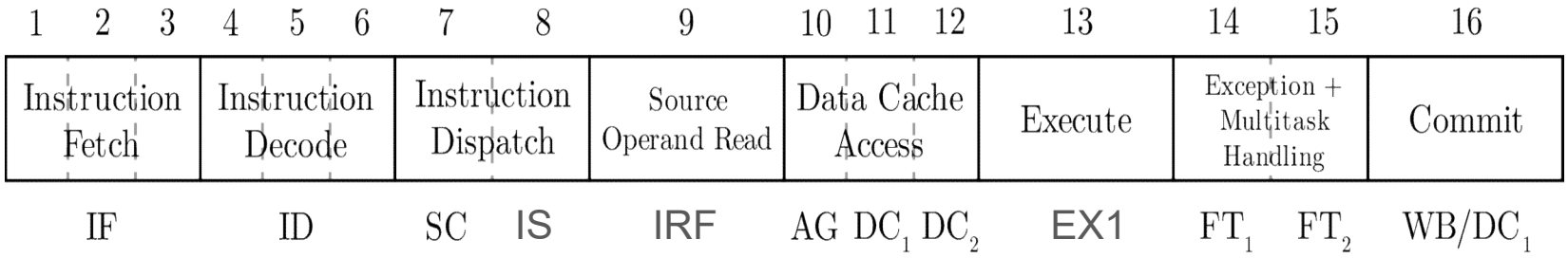
A maximális hatékonyság körülbelül 15-30 fokozatú futószalag esetén érhető el.
Általános célú alkalmazásoknál efölött a függőségek miatt már csökken a teljesítmény.
Speciális feladatok esetén, ahol kevés függőség van használható superpipeline CPU, ami akár 200 fokozatú is lehet.

A mai CPU-k jellemzően 15-25 fokozatú futószalagokkal rendelkeznek. Tipikusan az FP futószalagok 2-3 fokozattal hosszabbak az FX futószalagoknál (dedikált futószalagok!)



Gyakorlati példa: Intel Atom CPU

Az Intel Atom processzor a 2008-ban jelent meg.
16 fokozatú tiszta futószalag processzor volt.
CISC (Complex Instruction Set Computing) architektúrájú, azaz egy utasításon belül nem csak a regiszterekből, hanem a memóriából vagy a gyorsítótárból is képes adatot lehívni.



SC – Switch Context

IS – Instruction Schedule

IRF – Instruction register file

AG – Address Generation (CISC architektúrához szükséges)

DC1 – Data Cache 1 (CISC architektúrához szükséges)

DC2 – Data Cache 2 (CISC architektúrához szükséges)

FT1 – Fault Tolerant 1

FT2 - Fault Tolerant 2

WB/DC1 – visszaírás memóriába vagy gyorsítótárba

Értékelése: a tisztán futószalag használat miatt teljesítményben visszalépést jelentett a korábbi architektúrákhoz képest, de nagy előny volt az alacsony fogyasztás. Ezért főleg mobil eszközökben használták.

Futószalagos feldolgozás előfeltételei:

2 fokozatú ideális futószalag esetén:

- A számítógép 2 db egymástól teljesen független végrehajtó egységgel rendelkezik
- Az egyik fokozat kimenete a másik fokozat bemenete kell legyen (időbeli párhuzamosság)
- Mindkét fokozat végrehajtási ideje azonos
- A fokozatok órajelre szinkronizáltak: órajelre kapják az inputot és egyetlen óraciklus alatt elvégzik a feladatukat

A fenti feltételek mindegyikének teljesülése esetén: **t = T/2**

ahol T a szekvenciális, t a futószalagos végrehajtási idő.

Történeti áttekintés:

1989	**Intel 80486**	**5 fokozat, de már külön lebegőpontos futószalag**
1993	Intel Pentium (P5)	már 5 fokozat
1999	Intel Pentium III (P6)	11-17 fokozat
2000	Intel Pentium IV (Netburst)	20-31 fokozat
2006	Intel Core 2 (újratervezett P6)	általában 14 fokozat
2008	**Intel Core i**	**16-20 fokozat**

Függőségek kezelés:

- Operandus előrehozással (minden architektúránál használják)
- Újrafeldolgozással (ez leggyakrabban az Execute fokozat egymás utáni többszöri végrehajtását jelenti: például a szorzás összeadások sorozata, akkor ne töltsük be minden iterációban az eredményt, hanem használjuk újra fel ⟶ a futószalag feldolgozást lassítja, de összességében jobb teljesítmény biztosít!)

Futószalag típusok:

1. Előlehívás (overlapping):

   A visszaírással egyidőben történik a következő utasítás lehívása:

	t_1	t_2	t_3	t_4	t_5	t_6	t_7
I_1	F	D	E	W/B			
I_2				F	D	E	W/B

Előnye:

- 1 óraciklust nyerünk ⟶ 4 fokozat esetén 25%-kal nő a teljesítmény
- **nincs függőség (!!!)**, mivel a forrás operandus beolvasásakor (t5) már megvan az előző utasítás eredménye (t4)

Hátrány: - nem túl nagy gyorsítás

Vektor CPU:

Csak az Execute fokozat működött futószalagszerűen, a végrehajtási fázisok kerültek átfedésbe.

Teljes pipeline:

A futószalagos feldolgozás kiterjesztése a teljes folyamatra:

	t_1	t_2	t_3	t_4	t_5
I_1	F	D	E	W/B	
I_2		F	D	E	W/B

1.7. Logikai futószalagok

Az eltérő utasítások eltérő felépítésű futószalagokat igényelnek, ezért egy processzor több futószalagot is tartalmaz. A cél a funkcionális kialakítás. Példák különböző funkciókat ellátó futószalagokra:

- aritmetikai (FX, FP): fokozatai tipikusan F, DS/O, E, W/B
  (egyszerű FX/FP: +, -, léptetés összetett FX/FP: *, /)
- ugró (branch): fokozatai F, E
- LOAD/STORE

Az utasításokat két szinten értelmezhetjük:

Első szint: pl.: Fetch

Második szint: mikroutasítások szintjén

MAR ⟶ PC innen tudjuk honnan kell lehívni a következő utasítást

MDR ⟶ [MAR] MAR által mutatott címen lévő értéket töltjük be

IR ⟶ MDR

PC ⟶ PC+1

Futószalag fizikai megvalósítása (implementáció):

Cél a logikai futószalagok fizikai megvalósítása. Alkalmazásuk alapján megkülönböztetünk:

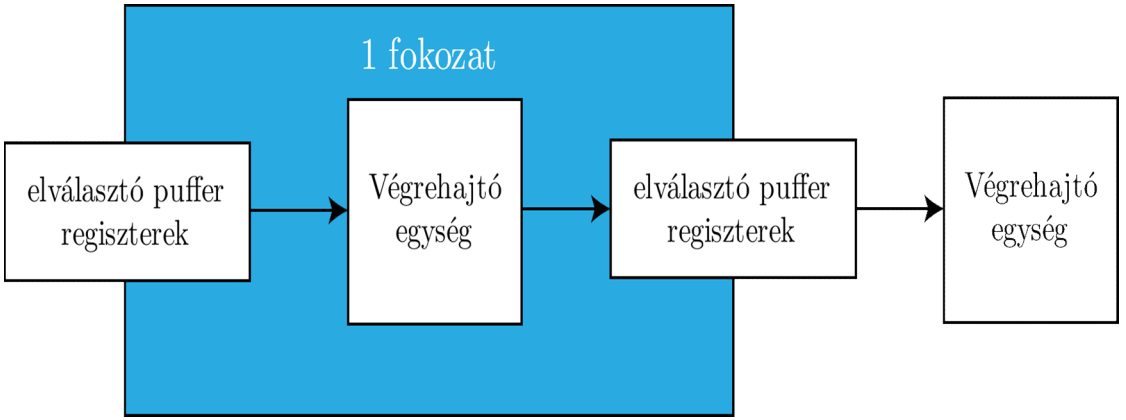
- univerzális: mindenre jó ⟶ sok tranzisztor, bonyolult, drága, lassú!
- dedikált futószalagokat: adott műveletre specializált

  Előny: Kevesebb logikai kaput igényelnek ⟶ gyorsul a végrehajtás (a bemenettől a kimenetig az elektronok gyorsabban átérnek). Pl.: Branch, LOAD/STORE, FX, FP

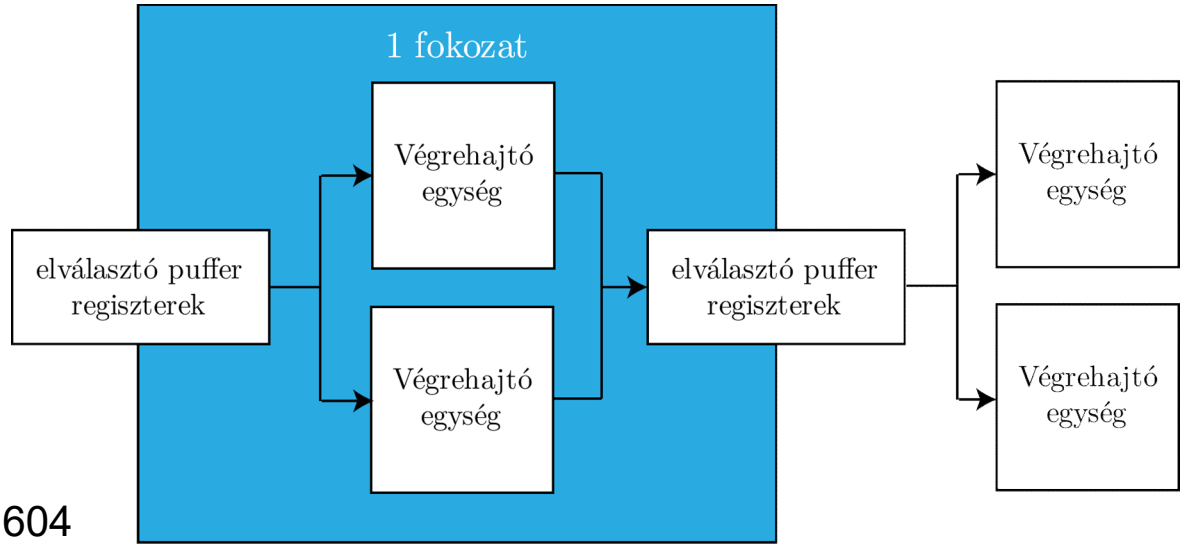
A futószalag sebességét általában a leglassabb fokozat sebessége határozza meg, tehát tervezési cél a megközelítőleg azonos sebességű fokozatok létrehozása!

Fokozatok kialakítása:

Minden fokozat előtt vannak elválasztó puffer regiszterek. Ezek a felhasználó számára nem látható regiszterek (transzparens), az adat először ezekbe, majd innen a végrehajtó egységbe töltődik. Az utasítás elvégzése után a kimenet szintén puffer regiszterekbe kerül. Mivel a gyakorlatban egy fokozat nem mindig végez egy óraciklus alatt és addig nem töltődhet be adat míg nem végez a végrehajtó egység, ezért az adat ezekben a puffer regiszterekben tárolódik, így orvosolva az eltérő végrehajtási időket.



Térbeli párhuzamosítás során, mikor egymás mellé több végrehajtó egységet helyeztek vezérlőjelek segítségével határozzák meg, hogy a puffer regiszterekből melyikbe töltődjön az adat:



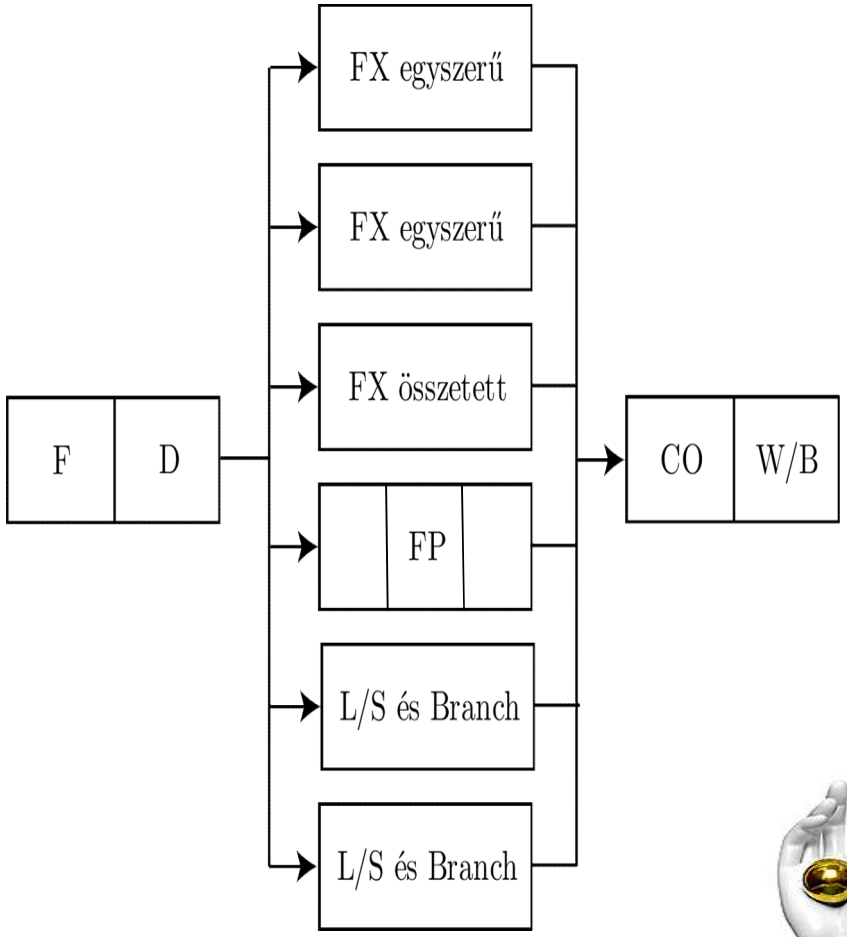
Példa: Power PC 604

A processzorban egymással párhuzamosan több dedikált futószalag is működött. A Fetch és Decode fokozatok minden óraciklusra lehívtak egy utasítást és a megfelelő futószalagba töltötték. Így a CPU képes volt egymással párhuzamosan akár 4 utasítást is végrehajtani. Mivel előfordulhatott, hogy a később lehívott és betöltött utasítás végzett hamarabb, szükséges volt egy konzisztencia fokozat (CO) bevezetése. Az utasítások címkézésre kerültek és a címkék szerint a CO biztosította, hogy az eredmények megfelelő sorrendben kerüljenek visszaírásra. A párhuzamosság miatt szükség volt Interlock funkcióra, ami biztosította, hogy az utasítások a megfelelő sorrendben hajtódjanak végre azáltal, hogy várakoztatott a fokozatok között.

Példa: Power PC 604

A processzorban egymással párhuzamosan több dedikált futószalag is működött (Execute fokozatban). A Fetch és Decode fokozatok minden óraciklusra lehívtak egy utasítást és a megfelelő futószalagba töltötték. Így a CPU képes volt egymással párhuzamosan akár 4 utasítást is végrehajtani. Mivel előfordulhatott, hogy a később lehívott és betöltött utasítás végzett hamarabb, szükséges volt egy konzisztencia fokozat (CO) bevezetése.

Az utasítások címkézésre kerültek és a címkék szerint a CO fokozat biztosította, hogy az eredmények megfelelő sorrendben kerüljenek visszaírásra. A párhuzamosság miatt szükség volt Interlock funkcióra, ami biztosította, hogy az utasítások a megfelelő sorrendben hajtódjanak végre azáltal, hogy várakoztatott a fokozatok között. FP több fokozat!



Történeti áttekintés:

Futószalag fokozatok száma (Intel):

Intel 486:	5 fokozat
P5 architektúra (Pentium):	5 fokozat
P6 architektúra (Pentium III.):	11-17 fokozat
Netburst architektúra (Pentium 4):	20-31 fokozat
Core 2 (újratervezett P6 arch.):	14 fokozat
Core i architektúra:	16-20 fokozat

A futószalagos feldolgozás igen nagymértékben növeli a rendszer teljesítményét és általános hatékonyságát a hardver optimális kihasználásával!

Jelentősen növeli a feldolgozási sebességet, de a megjelenő különböző típusú **függőségek** kezelése kritikus jelentőségű a hatékonyság fokozása érdekében.

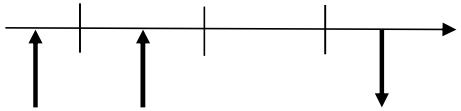
Futószalagos feldolgozás következményei:

- Jelentősen felgyorsult az utasítás lehívás és az operandus betöltés
- Sebességolló jelenség: amíg a processzorok feldolgozási sebessége nőtt, addig a memória sebessége kevésbé ➡ Cache bevezetése. Az óriási sebességkülönbséget próbálták kiküszöbölni a gyakran használt operandusok cache-be való töltésével.

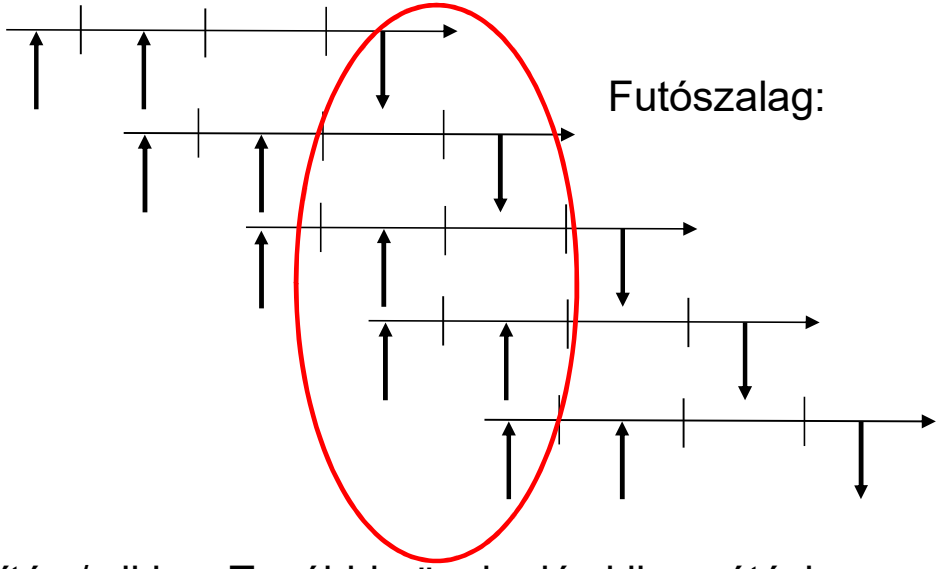
Probléma:

szekvenciális végrehajtás:

F D/so E WB



memóriához fordulások száma



Futószalag:

- Maximális végrehajtási sebesség: 1 utasítás / ciklus. További növekedés kibocsátási párhuzamossággal vagy utasításon belüli párhuzamossággal (multimédia) lehetséges.
- A vezérlés átadási utasítások kifinomult feldolgozási technikája szükséges a függőségek miatt

- Az elágazások kezelés bonyolódik:

  Korai RISC gépek:

  - Feltétlen elágazásnál ugrási buborék használata, ami blokkolja a következő utasítás lehívását n-1 óraciklusig ⟶ sok fokozat ⟶ nagy „büntetés” (lassulás)
  - Feltételes elágazásnál minimum +2 óraciklus (feltétel kiértékelés és ugrási címszámítás)

Kezelés:

Korai CISC gépek:

- Itt nem alkalmazták az ugrási buborékot. Helyette a dekódoló fokozatba építették a címszámító és a logikai komparáló egységet (feltétel kiértékelés), így a dekódolási ciklus végére előállt az ugrási cím.

Későbbi CISC architektúrák (futószalag és első generációs szuperskalár architektúrák)

- Megjelent a fix előrejelzés, például mindig ugrik. Ezzel a megoldással az ugrási cím előre kiszámításra kerül és megkezdődik az ugrási címen lévő utasítások lehívása. Ha mégse kell ugrani, az utasítások visszatörlésre kerülnek és az eredeti helyen folytatódik a végrehajtás. (például Intel 80486).

  Előnye, hogy a hivatkozási feltétel ismerté válásáig kiküszöbölte a kibocsátási blokkolást és növekedett a teljesítmény.

  Korlátja, hogy ha nagy látenciával rendelkező utasítást kell végrehajtani az ugrási feltételben, az blokkolja a kibocsátást.

Példa:

a := X / Y	Ez a futószalag CPU-knál blokkolja a további utasítások lehívását, mert
If a < 10 ….	nem értékelhető ki a végrehajtási fázisban!

II. generációs szuperskalárok-tól:

- Előredekódolás:

  A CPU a dekódolási feladatok egy részét már akkor elvégzi, amikor az utasításokat az L1 utasítás cache-be írja!

  Feladat:

  - Utasítás típus azonosítás
  - Ugrások felismerése
  - Utasításhossz meghatározása (CISC)

- Spekulatív elágazásbecslés (később)

RISC és CISC architektúrák:

Az utasításkészlet (tervezési stratégia) alapján kétféle architektúrát különböztetünk meg:

- RISC (Reduced Instruction Set Computing): csökkentett utasításkészletű architektúra
  például ARM (Advanced RISC Machine) processzorok
- CISC (Complex Instruction Set Computing): bővített utasításkészletű architektúra
  például x86 és x86-64 processzorok

A magas szintű programozási nyelveket jellemzően az utasításkészlet alapján tervezik, mivel a hardver és az utasítások megfelelő kialakítása határozza meg, hogy milyen gyors lesz a CPU .

Történeti összefoglaló:

Az első CPU-k kevés utasítással rendelkeztek (RISC), majd a 1970-es években az egyre több funkció és bonyolultabb utasítások miatt ez a szám növekedett (CISC). Az 1980-as éveben rájöttek, hogy a sok utasítás ugyan megkönnyíti a programozást, de a címzés bonyolultsága miatt káros hatással van a teljesítményre. Ez vezetett a RISC architektúrák újbóli megjelenéséhez.

RISC architektúrák főbb tulajdonságai:

- Kis számú (50-150) utasítással rendelkezik ⟶ címzési módok egyszerűsödése
- Nincs olyan utasítás, ami a LOAD/STORE-t aritmetikával kombinálja (nem lehet egyszerre betölteni az adatot és végrehajtani műveletet)
- Minden műveletvégző utasítás kizárólag regisztereket használ ⟶ memóriából és gyorsítótárból nem lehet dolgozni
- Memória és cache elérése csak LOAD/STORE utasításokkal történhet
- Nagyszámú regiszterkészlet emiatt
- Általában 3 operandusos utasítások ⟶ az eredmény nem írja felül a bementi regisztert
- Minden utasítás hossza egyforma (például 64 bit, 128 bit) ⟶ futószalagos feldolgozás könnyebb
- Bonyolultabb fordítóprogram a kevés utasítás miatt
- Általában huzalozott (hardveres) az utasítás dekódolás
- Utasítás végrehajtás általában egy óraciklust vesz igénybe ⟶ cél az egyforma ciklusidő!

Előnye: alacsonyabb energiafogyasztás, CISC architektúrákhoz képest gyorsabb végrehajtás utasítás szinten (pl.: szorzás 8086-nál ~70-80 ciklus, RISC: ~30-40 ciklus)

Hátrány: bonyolultabb feladatokat instrukció szekvenciákkal kell megoldani, ez növelheti a program méretét, kisebb kompatibilitás, kisebb teljesítmény ugyanazon a frekvencián (x86-64-hez képest)

CISC architektúrák főbb tulajdonságai:

- Nagy számú utasításkészlet (több száz)
- A sok utasítás nagy belső mikroprogramtárat igényel
- Sokféle utasítás és címzési mód
- Változó méretű (akár összetett) utasítások ⟶ a dekódolónak nem csak dekódolni kell az utasítást, hanem azonosítani is az utasítás végét. Ezt hívják utasítás határra illesztésnek, ami plusz hardvert és időt igényel (pl.: x86: 1-60 byte utasításhossz!)
- Közvetlen memória elérés lehetséges ⟶ a második operandus lehet memória vagy cache cím is
- Két operandusos utasítások ⟶ az első operandus felülíródik az eredménnyel. Ezáltal az első operandus nem lehet memória vagy cache cím, mivel a memóriába való írás nagyon lelassítaná a működést
- Utasítások feldolgozása több ciklusidő lehet ⟶ bonyolultabb feldolgozás
- Egyszerűbb a gépi kódú programozás a sokféle utasítás miatt (egyszerűbb fordítóprogramok)
- Egy utasítás több elemi műveletet is végre tud hajtani
- Kompatibilitás: mivel az utasítások folyamatosan bővültek, így a régebbi architektúrákra írt programok kompatibilisek maradtak
- Támogatja többek között a többszálúságot (HT) és a virtualizációt
- Futószalag fokozatok közt sebesség különbség lehet ⟶ feloldása Interlock funkcióval
- Általában +2 fokozat a RISC-hez képest: AG (címszámítás) és cache elérés

Előny: egyszerűbb compilerek, kompatibilitás, széles termékskála
Hátrány: komplexebb hardver, lassabb végrehajtás utasítás szinten, magasabb energiafogyasztás
(Pl.: az 1980-as években elterjedt Intel 80386 egy tisztán CISC architektúra volt)

Példa (elméleti):

CISC: ADD [100], [102] - egy utasításban a betöltés, a művelet, és a visszaírás!

RISC: LOAD AX, [100]
LOAD BX, [102]
ADD CX, AX, BX
STORE [100], CX

- sok utasítás, de egyszerű utasítások, ezért a dekódolás is egyszerűbb, gyorsabb és kevesebb tranzisztor kell hozzá!
⟶ több regisztert lehet kialakítani!

CISC: cél a teljesítmény maximalizálása (jellemzően magasabb energiafogyasztás) és a kompatibilitás. Ezeken felül egyszerűsíti a programozást (compiler), hiszen sok utasítást ismer és segíti jobb a memória kihasználást.
Viszont bonyolultabb az áramkör tervezés és lassabb a dekódolási folyamat!

Összességében a RISC megközelítés előnyösebb lehet ott, ahol prioritást élvez az energia-hatékonyság, a tervezés egyszerűsége, (mobil eszközök, a beágyazott rendszerek, cél-hardverek, edge computing, lásd: IoT, AI)

A CISC megközelítés a komplex és funkcióban gazdag utasításkészlettel továbbra is jó választás az általános célú számítástechnika, a szerverek és a munkaállomások számára, ahol a nyers teljesítmény és a szoftver kompatibilitása kiemelt jelentőségű.

Hibrid architektúrák:

Megfigyelték, hogy egy CISC CPU a működése során a rendelkezésre álló utasításoknak
~ 20%-át használja az idő ~80%-ában! ⟶ Célszerű ezeket gyorsítani ⟶ RISC mag!

Manapság a CPU-k jó része hibrid, azaz CISC CPU-k RISC maggal!

Szuperskalár architektúrák

Bevezetés:

A futószalag architektúráknál a dekódoló óraciklusonként egy utasítást tudott kibocsátani, ami korlátozta a teljes CPU sebességét (elméleti maximum: 1 utasítás / óraciklus)

További teljesítmény növekedés:

Időbeli és térbeli párhuzamosság megjelenése: ⟶ szuperskalár architektúra!

Itt már párhuzamos kibocsátást alkalmaznak.

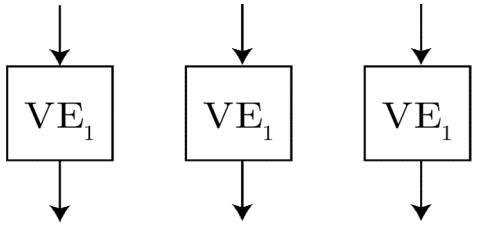
1990 környékén jelent meg és három generációra osztható.

A harmadik generációban már a multimédiás (utasításon belüli párhuzamosság) képességek is megjelentek.

Közös jellemzők

1. A dekódoló egységből képes óraciklusonként több utasítást kibocsátani (kibocsátási párhuzamosság!) ⟶ ennek mérőszáma a kibocsátási ráta
   - Első generáció 2-3 utasítás / ciklus
   - Második generációnál 4-6 utasítás / ciklus

2. Időbeli és térbeli párhuzamosság: ⟶ több futószalag párhuzamosan



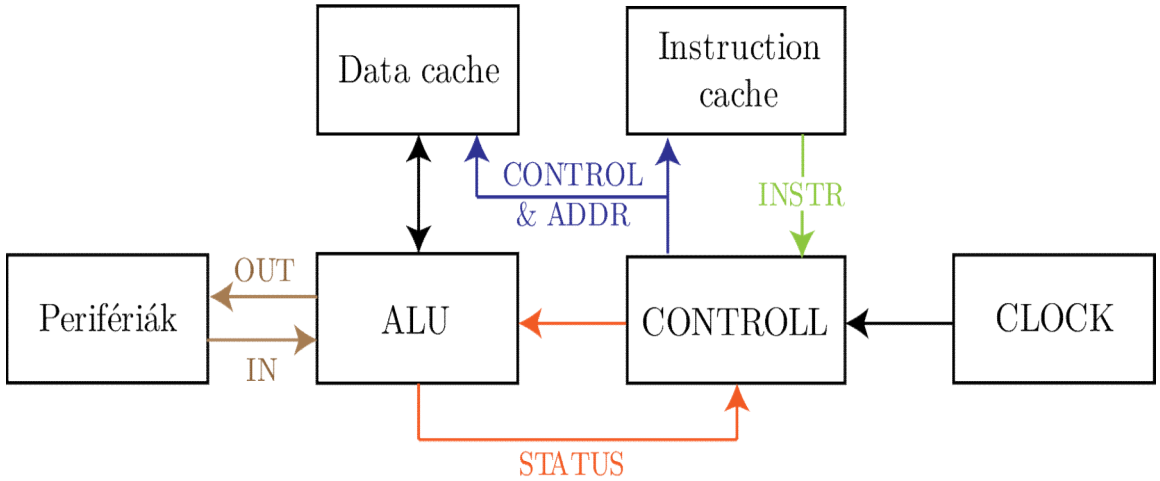
3. Maguk küzdenek meg a függőségekkel: dinamikusan, extra hardverek segítségével
   pl.: adattípusonként külön regisztertár
4. Kompatibilitás: evolúció a futószalagokhoz képest, tehát kompatibilis volt velük, így régi programok is futtathatóak maradtak.

Harvard architektúra:

1944-ben dolgozták ki az elvet, melynek lényege, hogy az adat és a programkód fizikailag elkülönített útvonalakon mozog.

Következménye, hogy párhuzamos adatutakat jönnek létre, utasítás és adat párhuzamosan tud mozogni, így növekszik a teljesítmény. A szuperskalár architektúráknál az L1 cache-nél használnak Harvard architektúrát, ahol az utasítás és az adat külön tárolódik. Manapság egy módosított Harvard architektúrát használnak, ami azt jelenti, hogy ezen felül képes programot adatként is betölteni. Mivel az L2 és L3 gyorsítótárban közösen tárolódnak az utasítások és adatok, látható, hogy a mai processzorok tervezésénél mind a Harvard, mind a Neumann elveket felhasználják.

Vezérlési vázlat:



A vezérlőegység (CONTROLL) hívja le az utasítást az instruction cache-ből (INSTR adatút), majd ez alapján jelet küld a data cache-nek, hogy az ALU-ba milyen címen lévő adat kerüljön (CONTROLL & ADDR). Ezzel egyidőben az instruction cache felé is küld jelet, tehát a következő órajelre az adat és a következő utasítás egyszerre hívódik le. A vezérlőegység felel az ALU irányításért is. Az ALU az IN és OUT adatutakon kommunikálhat a perifériákkal, a STATUS adatúton pedig visszacsatolást biztosít a vezérlőegység számra. Az egész működés órajelre szinkronizált.

Előnyei:

- Képes párhuzamosan adatot és utasítást olvasni vagy írni **cache nélkül is**
- Az adat és utasítás tárolók különálló címtartománnyal rendelkeznek, amikben a címek különböző hosszúak is lehetnek (például utasítás címek 32 bit, adat címek 64 bit)

Neumann architektúra: egy óraciklusban vagy utasítást olvas be, vagy ovassa / írja a memóriát!
(csak egy busz van!)

Első generációs szuperskalárok (keskeny szuperskalárok):

Kibocsátási ráta: RISC architektúráknál 3 utasítás/ciklus
CISC-nél 2 utasítás/ciklus

Ez 2-3x-os sebességnövekedés a futószalag elvű processzorokhoz képest.

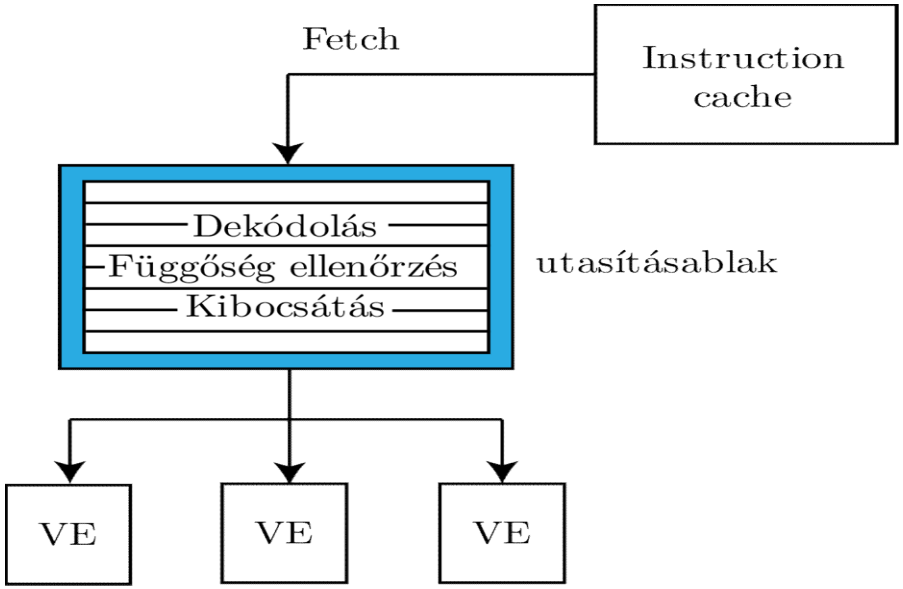
Jellemzői:

- Közvetlen, vagyis nem pufferelt kibocsátás: a CPU a dekódolt utasítást közvetlenül küldi a végrehajtó egységhez
- Statikus elágazásbecslés, amit a Fetch alrendszer végzett. Kétféleképpen valósult meg:
  - Fix elágazás: elágazásnál mindig ugrik, vagy
  - A programkód bizonyos tulajdonságai alapján hozott létre egy statikus elágazásbecslést
- Kétszintű gyorsítótár a memória lassúságának kiküszöbölésére: L1 cache a processzorlapkán, L2 különálló lapkán
- Különálló adat és utasítás az L1 gyorsítótárban ➡ Harvard architektúra, L1 elérése Harvard elvű működéssel!
- Operatív tár és L2 cache közös az adatok és utasítások számára ➡ Neumann architektúra, elérésük Neumann elvű működéssel!

ÓBUDAI EGYETEM

Közvetlen nem pufferelt kibocsátás:

A CPU a dekódolt utasítást küldi a végrehajtó egységhez. Ehhez új fogalmat vezettek be: utasításablak. Ez egy olyan puffer, amely az óraciklusonként kibocsátott utasításokat tartalmazza. Az utasításablak az utasítás pufferből kerül feltöltésre és utasítás kibocsátáskor kiürül. Itt történik a dekódolás és a függőség ellenőrzés. A végrehajtható utasítások kibocsátásra, vagyis egyből a végrehajtó egységbe kerülnek. A végrehajtható utasítás olyan utasítás, aminek nincs függősége. A függőséggel rendelkező utasítások addig maradnak az utasításablakban, amíg a függőség meg nem szűnik.



Utasításablak működése:

Utasítás pótlás lehetőségei:

- A kibocsátott utasításokat egyenként pótoljuk
- A kibocsátott utasításokat egyszerre pótoljuk (megvárjuk, hogy az összes utasítás kiürül)

Utasítás végrehajtás és kibocsátás lehetőségei:

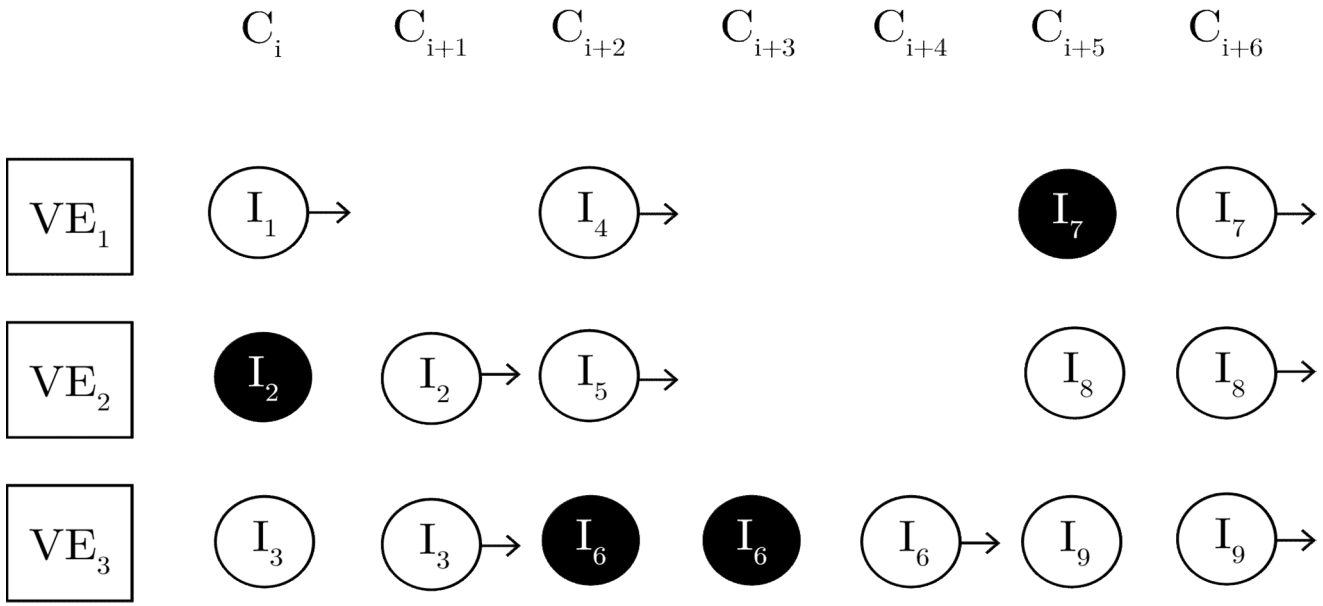
- Sorrendben
- Sorrenden kívül

Kezdeti megoldás az utasításablak egyszerre történő feltöltése és sorrendi végrehajtás volt. A sorrendi kibocsátás hátránya, hogy egy függő utasítás a függőség megszűnéséig blokkolja a kibocsátást, így a végrehajtóegységek kihasználatlanul álltak. Ez volt a legnagyobb probléma az első generációs szuperskalárokkal.

Példa: 3 darab végrehajtó egységünk van, az utasításablak kezdeti tartalma pedig két független és egy függőséggel rendelkező utasítás. Jelöljük körrel a független utasításokat és teli körrel a függőket. Az adott óraciklus kibocsátott utasításokat jobbra mutató nyíl jelzi.

ÓBUDAI EGYETEM

Az első óraciklusban (Ci) az I1 független ezért kibocsátható. Az I2 utasításnak függése van, így még nem bocsátható ki, a sorrendiség miatt az I3 sem. Az I2 és I3 a második óraciklusban (Ci+1) kerül kibocsátásra. Ezután új utasításokat hívunk le (I4, I5, I6). I4-et és I5-öt rögtön ki tudjuk bocsátani, mivel nincs függőségük és a sorrendiség miatt I6 függősége sem akadályozza őket. I6 függőségének feloldására valamiért két óraciklust kell várnunk, ezért csak a Ci+5 óraciklusban bocsátható ki. Mivel megint kiürült az utasításablak, újabb három utasítást hívunk le. Ebben az óraciklus viszont egyetlen utasítást sem tudunk kibocsátani, mivel a sorrendiség miatt I7 függősége I8-at és I9-et is akadályozza. Ezen utasítások kibocsátása csak a Ci+6 óraciklusban lehetséges.



3 végrehajtó egység esetén a kibocsátási ráta körülbelül 9/7.
A kibocsátási ráta a tapasztalatok alapján általában közelített az 1 utasítás/ciklushoz.

Rengeteg plusz tranzisztor, sokkal komlexebb architektúra ⟶ alig nőtt a teljesítmény!
„Sok hűhó semmiért!”
Ezért hívják keskeny szuperskalárnak!

Végrehajtási modell:

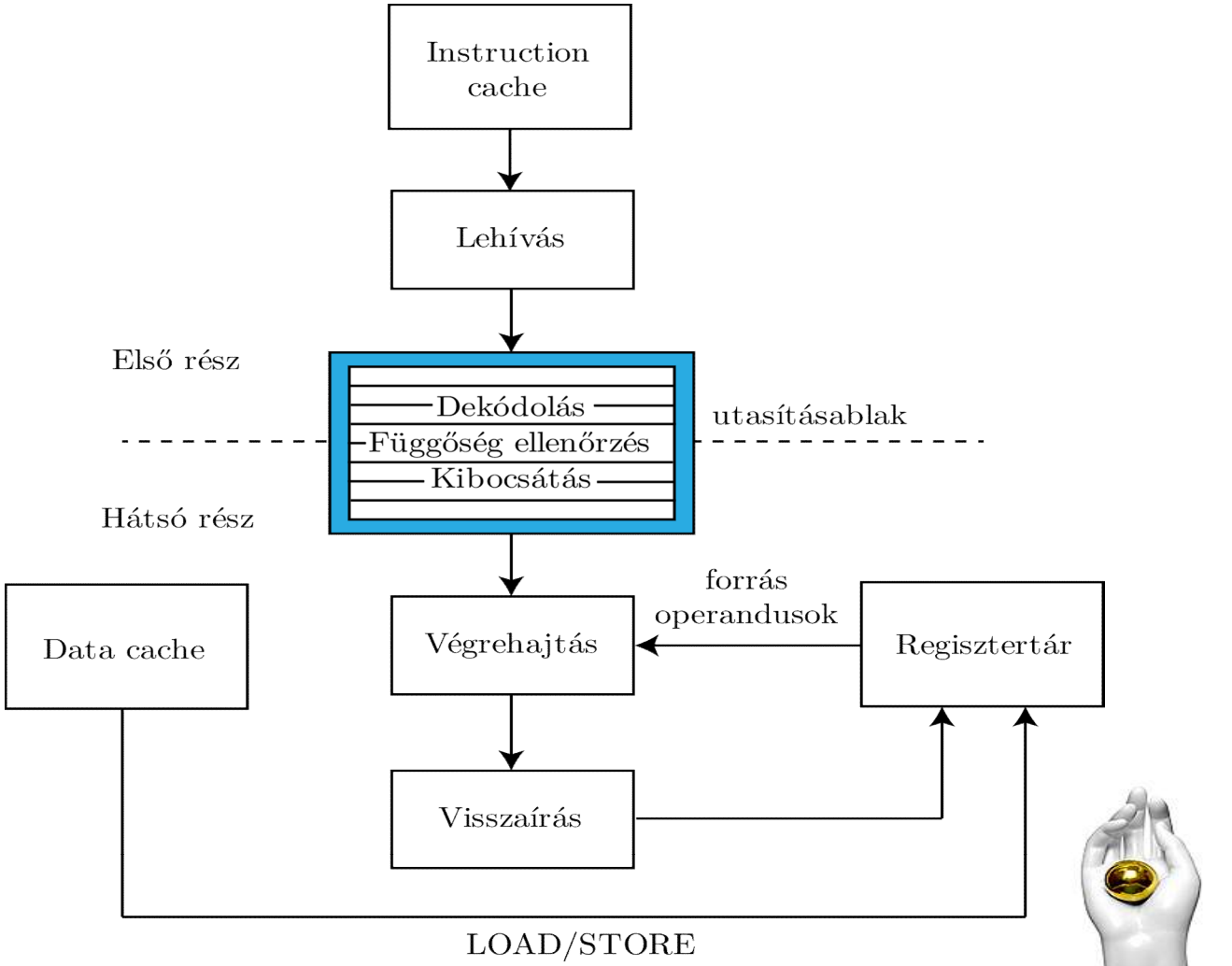
A teljes rendszer feldolgozási sebességét az alrendszerek átbocsátási képessége határozza meg.
Az alrendszerek jellege és száma az adott mikroarchitektúrától függ.
A végrehajtási modell RISC architketúra esetén:

- Első rész: feladata az utasítás lehívás és az utasításablak feltöltése
- Utasításablak
- Hátsó rész: feladata az utasításablak kiürítése (dekódolás, függőség ellenőrzése, kibocsátás), végrehajtás és a visszaírás

ÓBUDAI EGYETEM

RISC architektúra esetén az adat cache közvetlenül nem írható, csak regisztertárba lehet visszaírni. A regisztertár és az adat cache közötti adatmozgatást LOAD/STORE utasítások segítségével érhetjük el.

Szélességnek nevezzük az alrendszerek átbocsátási képességét. A teljes rendszer szélességét a legkeskenyebb alrendszer szélessége határozza meg.
(példa a következő lapon)



Szűk keresztmetszet:

Az első generációs szuperskalároknál a következő szűk keresztmetszetek lépnek fel:

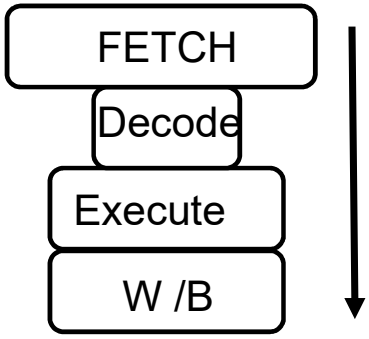
- Kibocsátás: oka a közvetlen kibocsátás, így kezelni nem lehetett, ezért következménye, hogy a végrehajtás általános célú alkalmazások esetén korlátozódik. Ez maximum 2 utasítás/ciklust jelentett CISC-nél, RISC-nél pedig 3 utasítás/ciklust (elméletben). Gyakorlatban viszont ~ 1 utasítás/ciklus volt a kibocsátás
- Memória: ⟶ csökkentése cache bevezetésével
- Elágazásfeldolgozás: ⟶ csökkentése statikus elágazásbecsléssel. Nem tudták viszont kezelni a RAW függőségek és az adatfüggőségek által létrehozott szűk keresztmetszetet. Még az ál függőségek is blokkolnak.

Esettanulmány: Intel Pentium I

A Pentium I-es CPU CISC architektúrájú keskeny szuperskalár volt.

Újdonság, hogy belül 64 bites, kívül pedig 32 bites buszokkal kapcsolódik.

(korábban is volt ilyen: Intel 8088: 2x8bit, 80286: 2x16 bit)

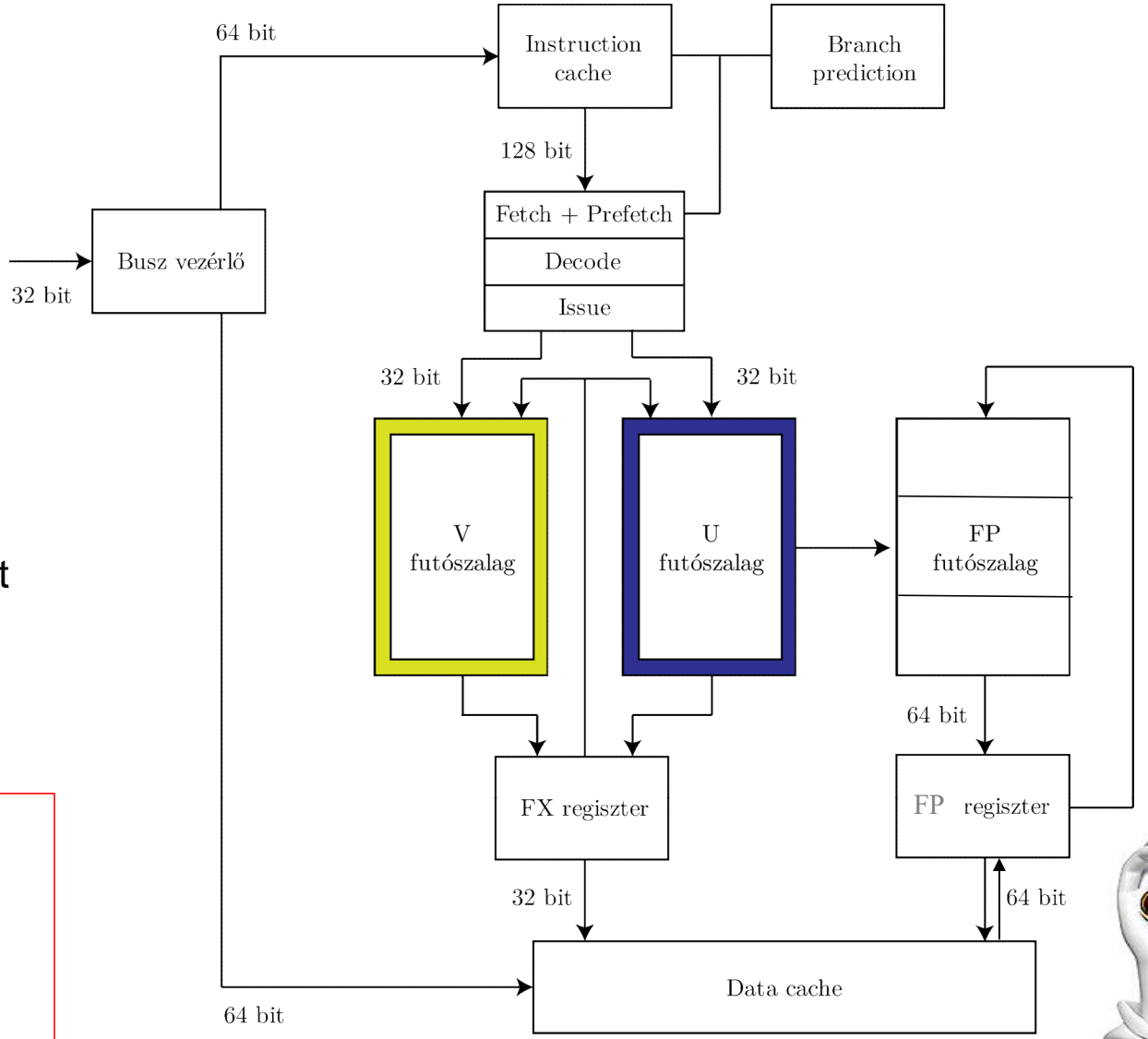


Futószalagok: 2 futószalaggal rendelkezett, ebből egy dedikált (V) és egy univerzális (U vagy master). A dedikált futószalag képes volt FX, L/S és Branch (elsősorban 1 ciklusos) utasítások végrehajtására.

A két futószalag csak akkor dolgozott egyszerre, ha mindkettő egyszerű utasításokat dolgozott fel. Mindkét futószalag 5 fokozatú volt (F, D, AG, E, W/B), + 3 fokozat FP-nek az U futószalagon!

Cache: adat és utasítás cache is 8 kbyte-os volt. Utasítás típus eloszlás (ált. célú alk.):

FX instructions	~ 40 %
Load instructions	~ 30 %
Store instructions	~ 10 %
Branches	~ 20 %
FP instructions	~ 1-5 %



A branch prediction modulban található a prefetch buffer.

Az utasítás cache-ből történik a lehívás, ami 128 bites szakaszokban történik.

CISC révén változó utasítás hosszal rendelkezett (1-17 byte-os utasítások).

⟶ Utasítás határra illesztés!

Dekódolás után a kibocsátás V és U futószalagra mehet 32 bites buszon.

A masterről van egy leágazás a lebegőpontos futószalagra, amiből 64 biten kerülnek az FP regiszterbe.

Operandusok beolvasása szintén ezekből a regiszterekből történik.

64 biten történik a buszvezérlőn keresztül az operatív tárból az utasítások és adatok olvasása.