Kidolgozott államvizsgatételek
Számítógép Architektúrák I-II-III.
tárgyakhoz

2010. június

A sikeres államvizsgához kizárólag ennek a dokumentumnak az ismerete nem elégséges, a témaköröket a Számítógép Architektúrák I-II-III órai jegyzetek részletesebben tárgyalják (de azok sem helyettesítik az órai részvételt!). A kidolgozott tételsorban és az órai jegyzetben előforduló hibákért a jegyzetek készítői semmilyen felelősséget nem vállalnak!

A jegyzeteket írta, elkészülésüket segítette és frissítette:
mrjrm, Pogácsa, Pheenix, Quetzalcoatl, AnNo, Broadcast és Félix.
Ne szakítsátok félbe a hagyományt ;)

Ezúton is köszönöm az oktatóknak, hogy türelmükkel és szakértelmükkel hozzájárultak a jegyzetek elkészüléséhez!

Tartalomjegyzék

SzA1-8: Számítógép Architektúrák I. (Dr. Broczkó Péter)

SzA9-10: Számítógép Architektúrák II. (Koschek Vilmos)

SzA11-14: Számítógép Architektúrák I. (Dr. Broczkó Péter)

SzA15-32: Számítógép Architektúrák II. (Dr. Broczkó Péter)

SzA33-64: Számítógép Architektúrák III. (Dr. Sima Dezső)

I. Az utasításszintű (logikai) architektúra

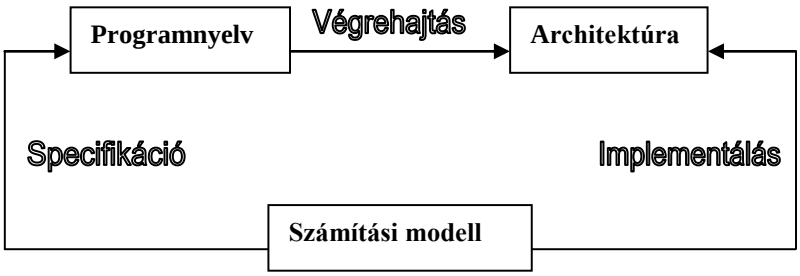
SzA1. Számítási modell

(fogalma; kapcsolatai, fajtái, a Neumann-féle és az adatfolyam számítási modell)

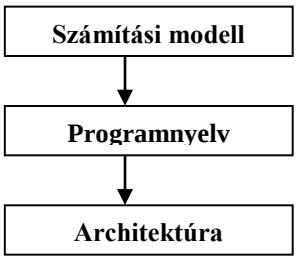
A számítási modell

- fogalma: A számításra vonatkozó alapelvek absztrakciója (Korábban a jelentése: soros vagy párhuzamos végrehajtás?).
- jellemzői:
  1. Min hajtjuk végre a számítást?
  2. Hogyan képezzük le a számítási feladatot?
  3. Mi vezérli a végrehajtás sorrendjét?

A számítási modell, a programnyelvek és az architektúra kapcsolata



Fejlesztési kronológia



→ Leírása igényeket támaszt

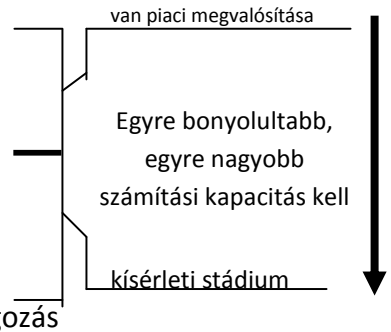
→ Ez kielégíti a számítási modell igényeit

A számítási modellek csoportosítása:

„Min hajtjuk végre a számítást?" elv szerint:

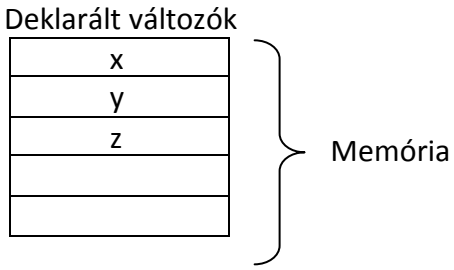
Adatalapú számítási modellek
- Neumann-féle számítási modell
- Adatfolyam számítási modell
- Applikatív számítási modell (igény végrehajtott)

Objektum-alapú számítási modellek ('70 -es évek)
Predikátum-logika alapú számítási modellek (Prolog)
Tudásalapú számítási modellek (a tudás nehezen mérhető)
Hibrid (általában Neumann-féle + Adatfolyam) szekvenciális + párhuzamos feldolgozás

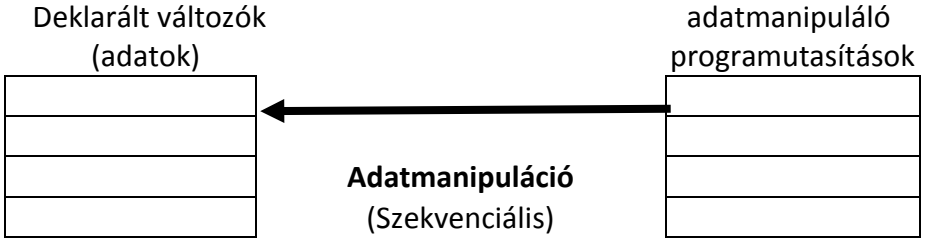


Neumann-féle számítási modell

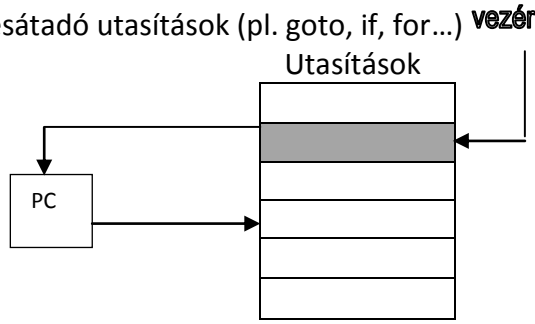
1. **Min hajtjuk végre a számítást?**
   - adatokon
   - az adatokat változók képviselik
   - biztosított, hogy a változók korlátlan számban változtathassák értékeiket (többszörös értékadás engedélyezett)
   - adatok és utasítások azonos memóriaterületen helyezkednek el



2. **Hogyan képezzük le a számítási feladatokat?**
   - adatmanipuláló utasítások sorozatával



3. **Mi vezérli a végrehajtást?**
   - az adatmanipulálások szekvenciája (implicit: természetes sorrend; add, mul, sub...)
   - az explicit vezérlésátadó utasítások (pl. goto, if, for…)

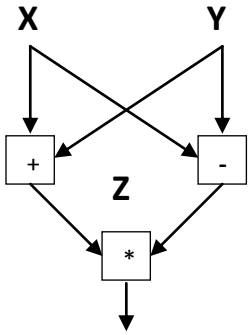


   - vezérlés-meghajtó (control-driver)
   - programnyelvek: Basic, Pascal, C, parancsnyelvek
   - architektúra: Neumann-féle architektúra

Adatfolyam számítási modell

- **Min hajtjuk végre a számítást?**
  - adatokon, az adatokat bemenő adathalmaz képviseli, egyszeres értékadás!
- **Hogyan képezzük le a számítási feladatot?**
  - adatfolyam gráffal
    = a csomópontok jelentik a műveletet (műveletvégzők)
    = élek az adat input-outputot, azaz adat-utakat, ahol az adat közlekedik (I/O vez.)

pl. z = (x + y) * (x – y)



- míg a Neumann-féle modellben a példa 3 db utasítást igényel:
  - összeadás (add)
  - kivonás (sub)
  - szorzás (mul)

  } 3 időegység

- addig az adatfolyam modellnél az összeadás és kivonás párhuzamosan végezhető, tehát példánkban 33%-os időmegtakarítást értünk el (3 időegység helyett csak 2).

Ez a számítási modell 1998 óta van a processzorokban (Pentium Pro megjelenése)

- **Mi vezérli a végrehajtást?**
  - adatvezérelt (más néven: stréber modell)

1. adat még nincs
2. az egyik operandus rendelkezésre áll – adat megjelenik egy vezetéken
3. ha mindkét operandus biztosított, műveletvégzés, azonnal
4. az eredmény előállt (ábra a korábbi jegyzetben)

Neumann-féle modell ⇔ Adatfolyam számítási modell

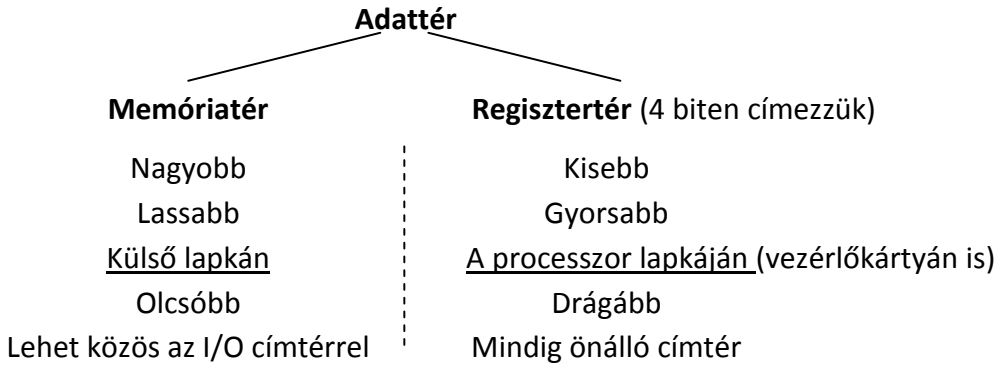
1.	Változók: adatokon hajtjuk végre közös operatív tár (=> program+adat)	Adatokon, egyszeres értékadás, bemenő adathalmazon. Az adattárolást az élek végzik
2.	Adatmanipuláló utasítások; szekvenciális, 1 processzor használata	Adatfolyamgráf Sok műveletvégző, párhuzamos működés
3.	Implicit szekvencia (adatman.utas.), explicit vezérlésátadás, vezérlésmeghajtott	Adatvezérelt, az összes bejövő adat megjelenésekor, azonnal

Programnyelvek: pl. Sigal
Architektúra: The Manchester Dataflow Machine

SzA2. Az adattér

(fogalma; a memória-tér; a regisztertér és fejlődése: egyszerű, adattípusonként különböző, többszörös regisztertér)

Adattér: A processzor által manipulálható tér, amíg a processzor „elér”.
pl. vezérlőkártyán I/O regiszter is idetartozik



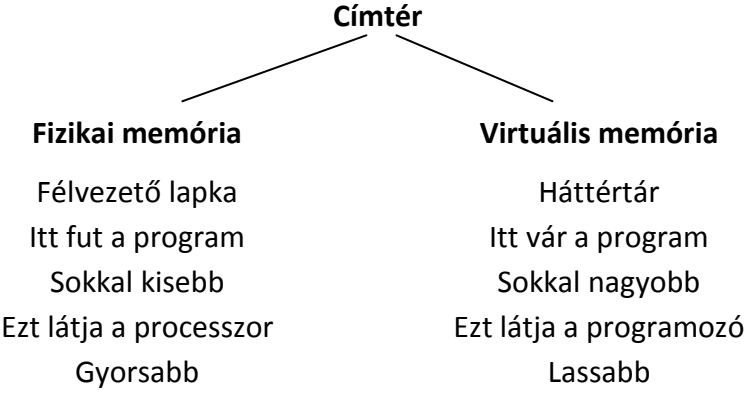
Memóriatér

- Tárolási kapacitása az egyik legfontosabb tulajdonsága
- Kétféle címtér
  - Modell címtere: a címsín szélessége határozza meg a kapacitást (32 bit: 4GB)
  - Implementáció címtere: alkalmazás igénye, ill. anyagi lehetőségek határozzák meg

Virtuális memória

Megjelenése: '50-es évek, elterjedése az IBM370-es gépcsaládhoz köthető. Jellemzői:

- kétféle memória: fizikai és virtuális memória
- Létezik olyan, a felhasználó számára transzparens mechanizmus, mely az éppen futó program számára nem szükséges **program- és adatrészeket** kiviszi a valós memóriatérből a virtuális memóriatérbe, majd amikor ezen adatok szükségessé válnak, visszaviszi a valós memóriatérbe.
  Kétirányú adatforgalom: [valós memória] ⟷ [virtuális memória]
- Létezik egy olyan, a felhasználó számára transzparens mechanizmus, mely a felhasználó által használt **virtuális címeket** a futási (execution) fázisban lefordítja valós címekké. Ez egyirányú, virtuális cím => valós cím



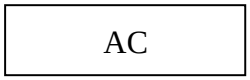
Regisztertér

Regiszterek osztályzása:

1. egyszerű regisztertér
2. adattípusonként különböző regisztertér
3. többszörös regisztertér

1. Egyszerű regisztertér

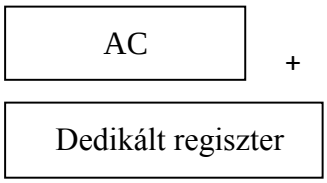
a) 40-es évek: egyetlen akkumulátor



Hátrányai:

- Az eredményt rendszeresen ki kell menteni
- Bizonyos műveleteknek két eredménye van (pl. osztás esetén hányados és maradék), az egyik az operatív tárba (memória) szorul ⇒ lassú

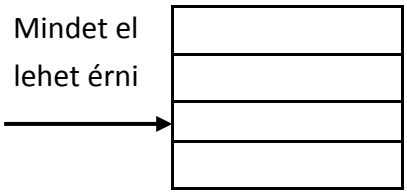
b) 50-es évek: egyetlen akkumulátor + dedikált regiszter



Előnye: Gyorsította a műveletet

Hátrányai: Drága, továbbá az esetek többségében üresen áll

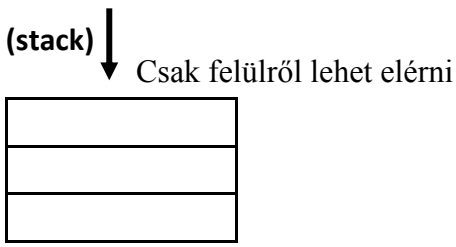
c) 60-as évek: univerzális regiszterkészlet



Előnyei:

- Általános célú
- Igyekeztek minden változót csak addig bent tartani a regiszterben, amíg szükség van rá
- Megfelelő gazdálkodással jelentős programgyorsítás érhető el

d) Veremregiszter (stack)

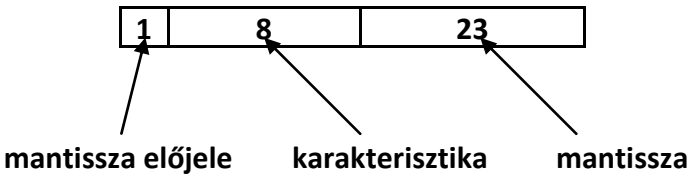


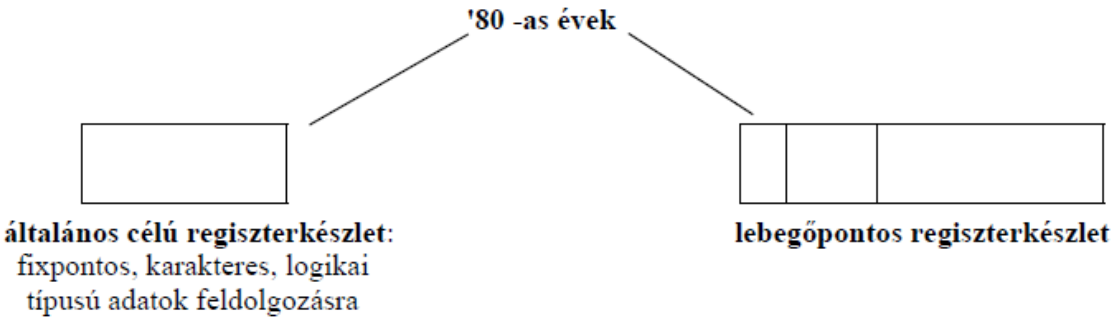
Előnye: Igen gyors

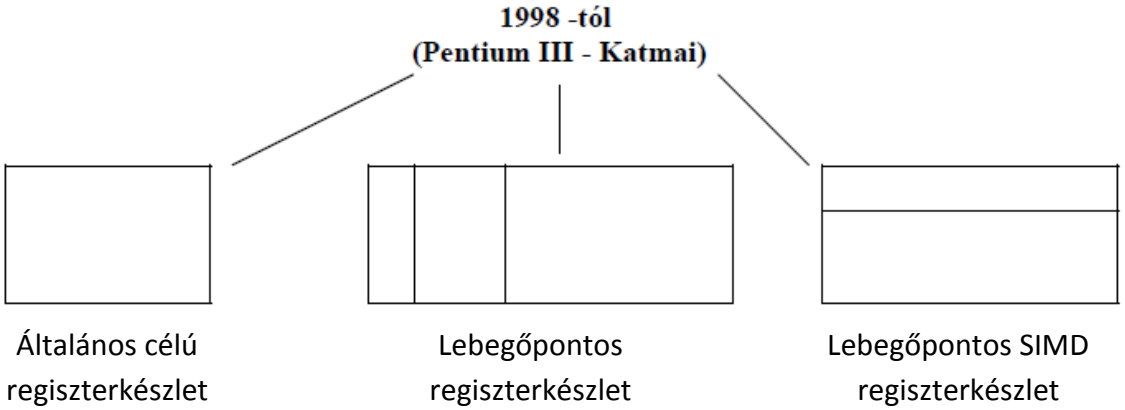
Hátrányai: Mivel csak a verem tetejét látjuk, ezért szűk keresztmetszetet ad

2. Adattípusonként különböző regisztertér

Célja: az adatfeldolgozás gyorsítása - különös tekintettel a lebegőpontos adatábrázolásra.
Szorzás esetén karakterisztika összeadódik, mantissza összeszorzódik.





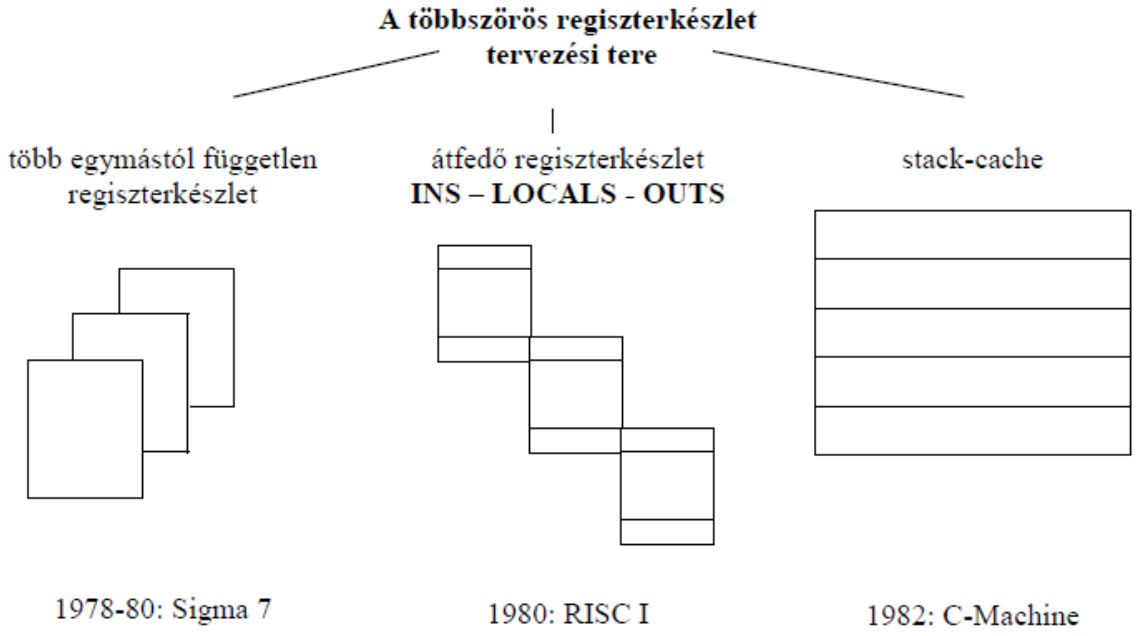


SIMD (Single Instruction Multiple Data): azon utasításoknak összefoglaló neve, amelyek egyszerre több adaton végzik el ugyanazt a műveletet

3. Többszörös regiszterkészlet

Háttér információ:

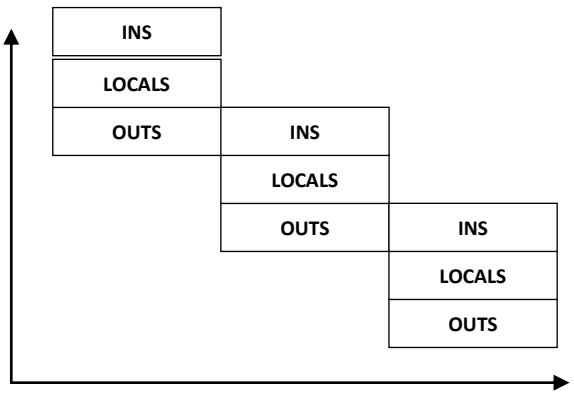
- Részei:
  - = a regiszterek aktuális tartalma
  - = az állapot-információk (flag) } Kontextus
- megszakítás esetén le kell mentenünk az éppen futó program kontextusát, annak érdekében, hogy majd a programot folytatni lehessen ugyanonnan
- a többfeladatos és több felhasználós feldolgozásnál igen sok a megszakítás.
  Amennyiben a kontextust az operatív tárba mentjük ⇒ <u>lassú</u>
  A memória helyett többszörös regiszterkészletet használjunk



Több egymástól független regiszterkészlet:

- <u>független</u> folyamatoknál ideális, pl. megszakítások
- paraméter-átadásos eljárásnál nem gyorsít, mivel a paraméterátadás a memórián keresztül történik → lassú, az átadást gyorsítsuk valahogy:

Átfedő regiszterkészlet:



y tengely: regiszterek száma

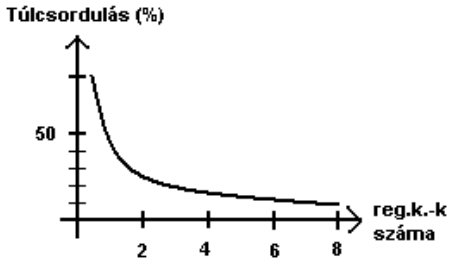
x tengely: regiszterkészletek száma

A paraméter-átadás problémájára megoldás.

Jellemzői:

- a hívó eljárás OUTS része fizikailag megegyezik a hívott eljárás INS részével, nem kell a regiszterek közötti műveleteket végrehajtani.
- a regiszterek száma fix, merev, viszonylag üres regiszterkészlet mellett is előfordulhat a túlcsordulás: ez a memória igénybevételével kerül feldolgozásra ⇒ lassul a feldolgozás

A regiszterkészletek száma:



= az ábrán látható, hogy 6-8 regiszterkészlet esetén már igen csekély %-os a túlcsordulás

= a programozás módszertan sem ajánlja, hogy 8 -nál több eljárást ágyazzunk egymásba, emberek számára nehezen követhetővé válik a programozás.

RISC I -nél 8 db regiszterkészlet szükséges (4-5% túlcsordulás)

Probléma: fix és merev

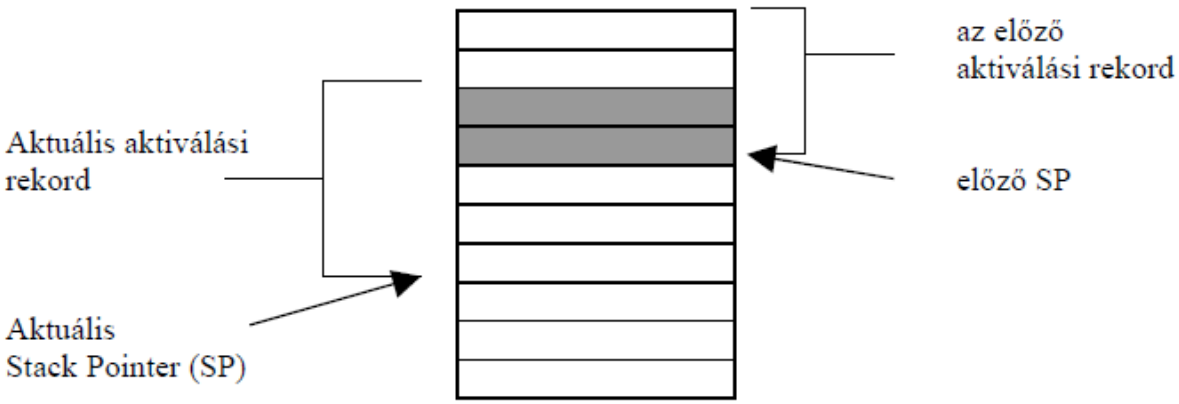
Stack-cache:

Ötvözi - a cache gyorsaságát és a veremregiszter szervezését

- a regiszterek közvetlen címezhetőségét

Működése:

- a compiler minden eljáráshoz hozzárendel egy-egy változó hosszúságú aktiválási rekordot (regiszterkészletet)



- a hívó eljárás OUTS része fizikailag megegyezik a hívott eljárás INS részével
- az aktiválási rekordok számának csak a stack-cache fizikai mérete szab határt (a túlcsordulást kiküszöböljük)

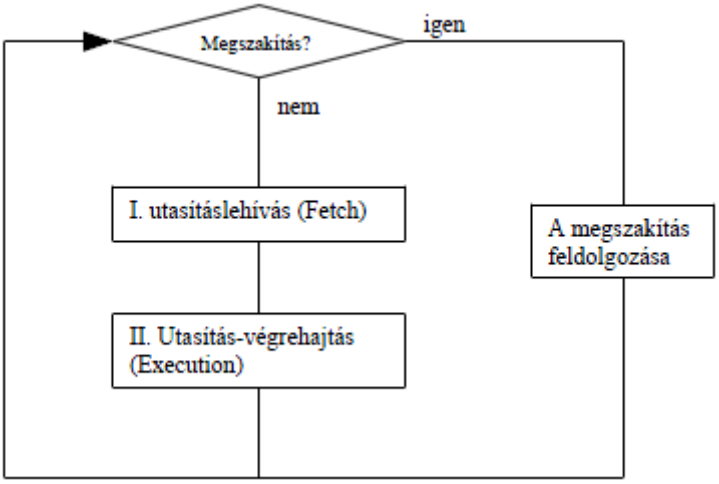
Ezekről nem volt szó órán, de egy korábbi jegyzetben igen:

- az adott aktiválási rekordot az SP segítségével közvetlenül is elérhetjük
- egy adatot is elérhetünk közvetlenül az SP és a relatív távolság megadásával

SzA3. A szekvenciális utasításvégrehajtás menete

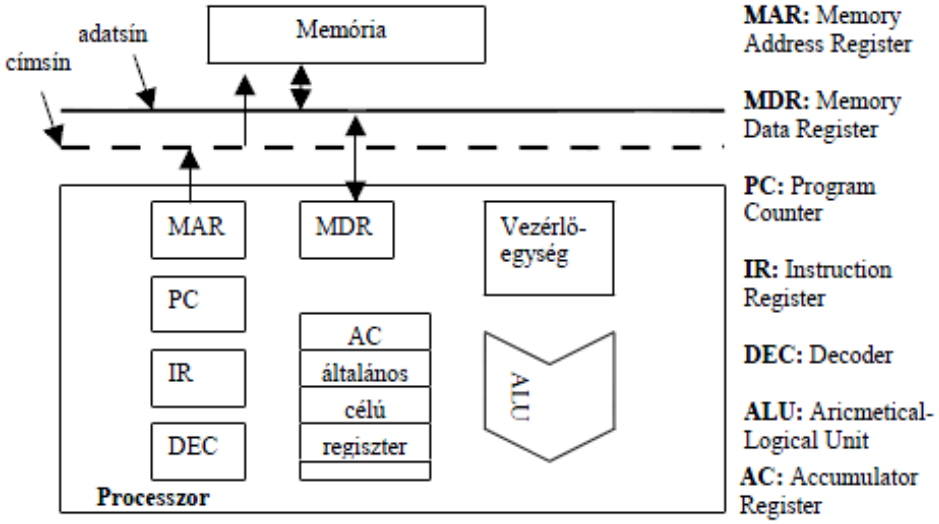
(az aritmetikai utasítások és a feltétlen vezérlés-átadási utasítás végrehajtásának sémája)

Az utasításfeldolgozás általános folyamatábrája:



Megszakítást csak két utasítás feldolgozása között lehet elfogadni!

A processzor regiszterei:



Utasítás-végrehajtás (minden utasítás esetén eltérő)

- aritmetikai-logikai utasítások (MK -tól függ)

DEC ⟵ IR

MAR ⟵ DEC címrész

MDR ⟵ (MAR) és

AC ⟵ AC + MDR vagy AC - MDR vagy AC * MDR vagy AC / MDR

- a feltétlen vezérlésátadás (ez a PC felülírása a gyakorlatban)

DEC ⟵ IR

PC ⟵ DEC címrész

SzA4. Az utasítás- és operandus típusok

(utasítás- és operandus típusok; szabályos architektúrák)

Az utasítások fajtái (utasítás-típusok):

MK	Címrész

op – operandus, s – Source (forrás), d – Destination (cél), @ - tetszőleges művelet

4 címes utasítás:

- $op_d := op_{s1}$ @ op_{s2} , op_4
- a 4. operandus a következő végrehajtandó utasítás címét tartalmazta

  Hátránya:

  = memóriapazarlás

  = további adatrögzítési hibák

  = merev program-struktúra ⇒ nehéz a program karbantartása
- Pl. ENIAC

3 címes utasítás: a következő utasítás címét egy speciális hardverben tároljuk: PC

- $op_d := op_{s1}$ @ op_{s2}
- az eredmény helyének explicit deklarálása
- Előnye: az előző utasítás eredményének mentésével párhuzamosan tölthetjük az aktuális utasítás két bemenő operandusát.
- Hátránya: Neumann szerint tipikusan az előző művelet eredménye a következő művelet egyik bemenő operandusa, akkumulálódjon az eredmény a gyűjtőben!
- Olyan helyeken használják ahol nagy mennyiségű adat van és nincs elágazás.

2 címes utasítás

- $op_{s1} := o_{s1}$ @ op_{s2} vagy $op_{s2} := op_{s1}$ @ o_{s2}
- pl. ADD[100],[102]: a gyakorlatban nem használjuk
- előnye:

  = kevesebb tárhelyet igényel (regiszter v. memória), de az egyiket felülírjuk.

  = kielégíti a Neumann féle követelményt
- Pl. IBM 360/370, Intel processzorokban (mai CISC architektúra)

1 címes utasítás

- be kell tölteni az egyik operandust az akkumulátorba LOAD[100]
- az összeadó utasításban lévő operandust hozzáadjuk az AC -hoz és az eredmény az AC -ban keletkezik: ADD[102]
- az AC tartalmát kimentjük: STORE[100]

Az utasítások maguk rövidebbek, de több utasításra van szükség

0 címes utasítás

- fajtái

  = NOP – No OPeration

  = a műveleti kód tartalmazza az operandust is, pl. CLEARD ⇒ a Dflag törlése

  = verem-műveletek PUSH, POP

Napjaink trendje:

3 címes utasítások a RISC gépekben

- mindhárom cím regiszterbeli operandusra mutat (r r r)
- csak a Load/Store utasítás engedélyezett

2 címes utasítások a mai CISC gépeket jellemzik:

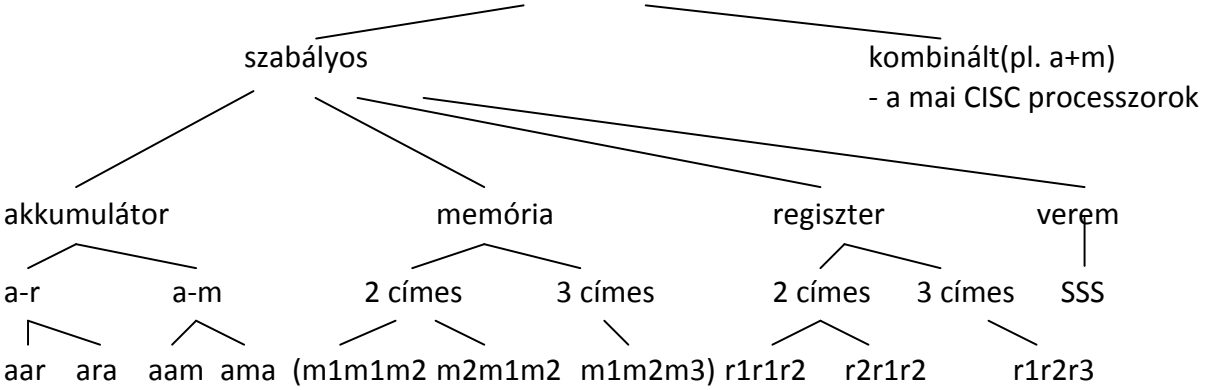
- általában az első operandus helyén keletkezik az eredmény
- az első operandus kizárólag regiszter lehet

Operandus-típusok

akkumulátor (a) memória (m) regiszter (r) verem (stack - s) immediate (i)

immediate: magában a programban adunk értéket a változónak ⇒
a gyakorlatban ez bemenő operandus (runtime)

Architektúrák osztályozása



A: Akkumulátor M: Memória R: Regiszter S: Stack

Akkumulátor (1 db)

- előny: gyors, rövid cím
- hátrány: szűk keresztmetszett
- napjainkban nem aktuális, ritkán alkalmazzák

Regiszter

- előny: igen gyors (kevés a regiszterek száma), rövid cím
- napjainkban RISC gépekben (3 címes)

Memória

- előny: nagy címtér
- hátrány: lassú, hosszú cím ⇒ hosszú utasítás
- napjainkban nem aktuális

Verem

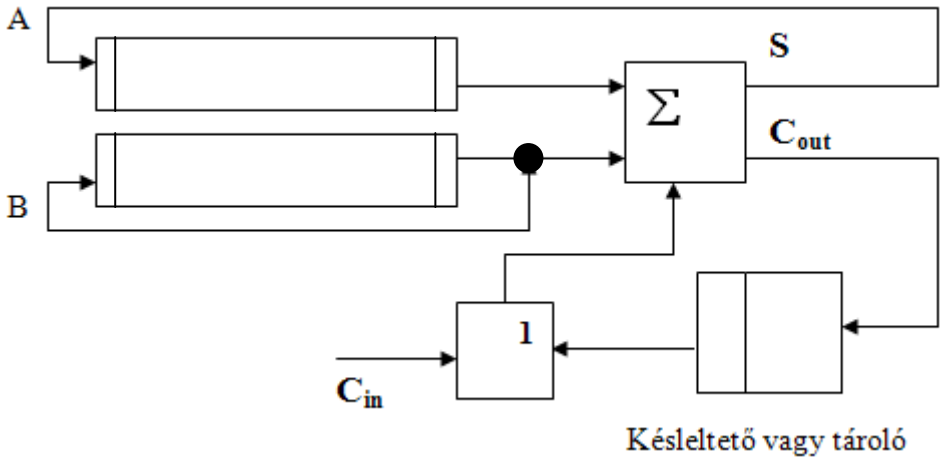
- előny: gyors: 0 hosszúságú cím
- hátrány: szűk keresztmetszett
- Pl. HP 3000, VT 1005 (VT: VideoTon)

II. Hagyományos mikroarchitektúra (fizikai architektúra)

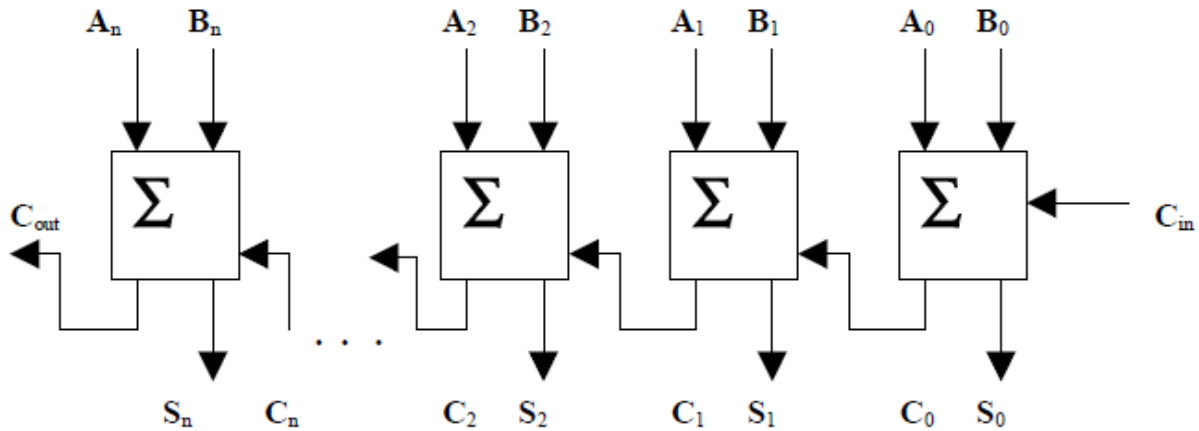
SzA5. Az aritmetikai egységek felépítése I.

(az n-bites soros és párhuzamos összeadó, valamint az előrejelzett átvitellel felépített n-bites összeadó)

N-bites soros összeadó

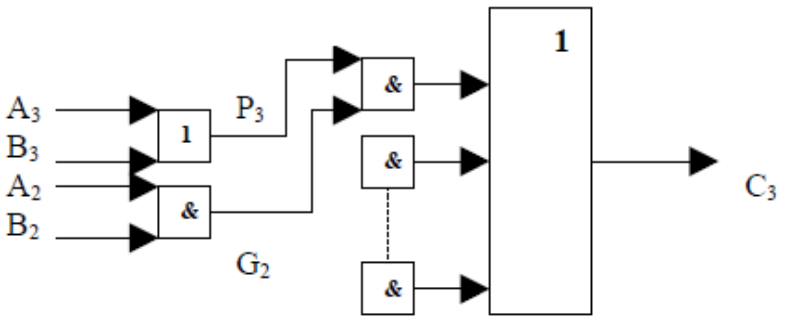


N-bites párhuzamos összeadó

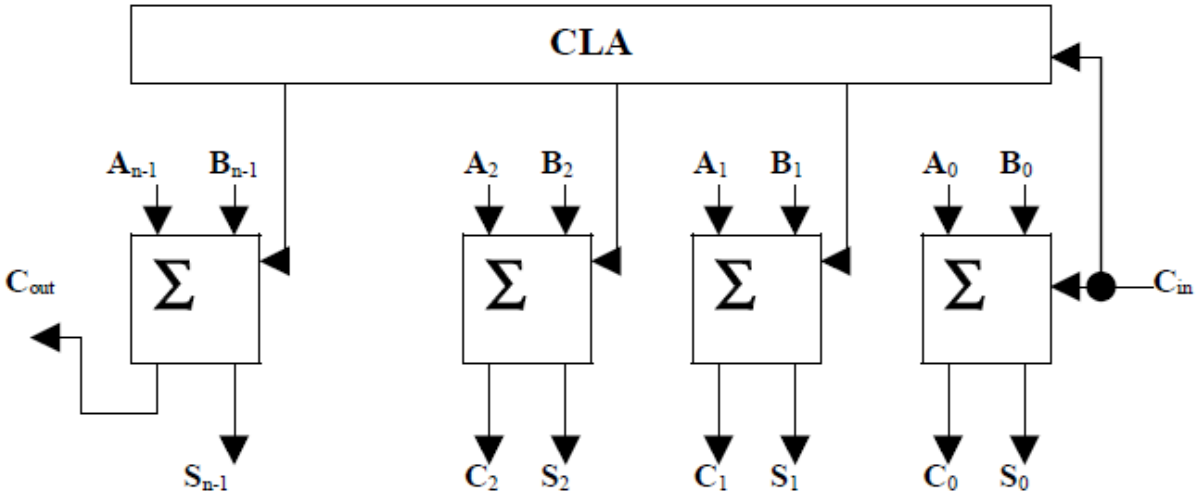


Előrejelzett átvitellel felépített n-bites összeadó (Carry-Look-Ahead = CLA)

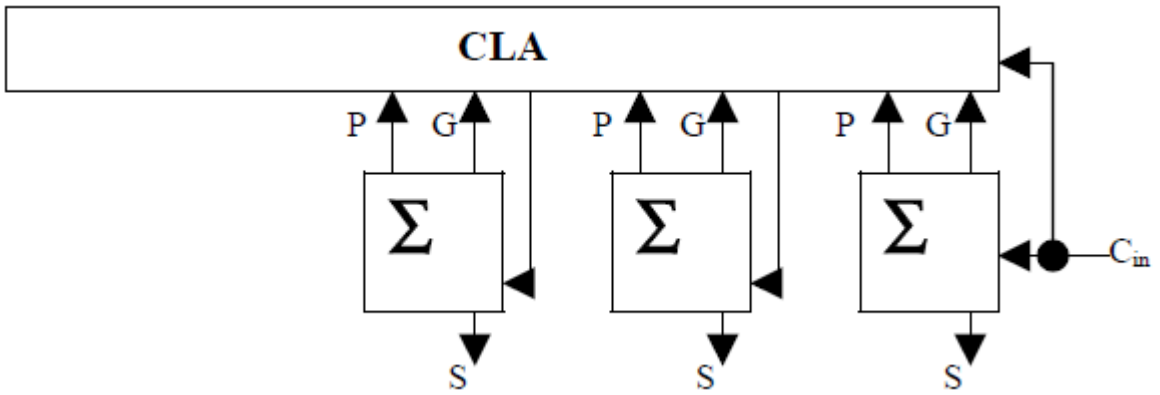
Pl. C_3 meghatározása előre (részletesebben lásd: órai jegyzet)



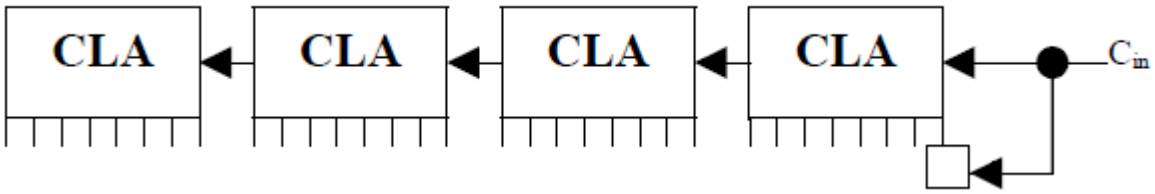
1. Katalógus áramkörökkel egybites teljes összeadó + CLA:



2. Az egybites teljes összeadót kiegészítjük 2 új áramkörrel (egy ÉS, és egy VAGY kapuval), hogy meghatározzuk a P és a G értékét:

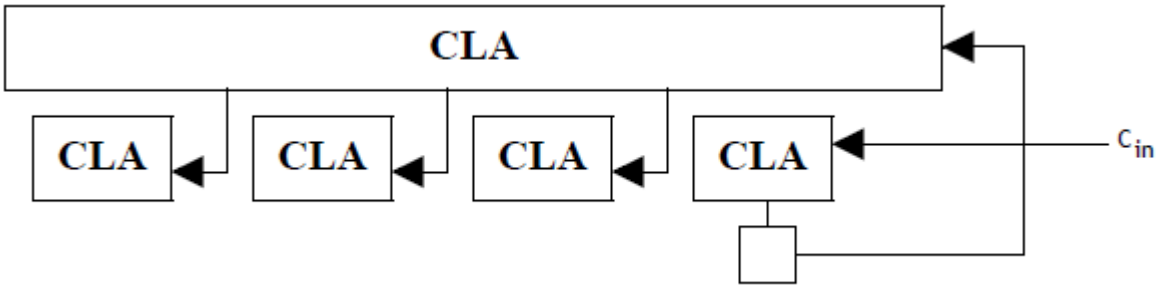


3. A VAGY kapu bemeneteinek száma technológiai korlátokba ütközik, ezért maximum 8 bit vonatkozásában építhető meg a CLA.32 bit megvalósítási lehetősége:



Hátrány: a CLA egységek között az átvitel sorosan terjed, ami lassú.

4. CLA a CLA-k számára:



SzA6. Az aritmetikai egységek felépítése II.

(a fixpontos szorzás algoritmusai és gyorsítási lehetőségek)

A fixpontos szorzás algoritmusai:

- Hagyományos módszer: a részeredményeket összegezzük
- Algoritmikus módszer: egy gyűjtőben összegezzük az aktuális részeredményt az előzőkkel
- Léptetéses módszer: az aktuális részeredményt balra léptetve adjuk hozzá az előzőekhez

Példákat lásd: órai jegyzet.

Bináris szorzás sajátosságai

- A bináris szám hossza: a bináris számok 3-4x hosszabbak, mint a decimális számok → az összeadási ciklusok száma magasabb lesz
- A szorzat hossza: mivel a szorzandó és a szorzó is egy-egy regiszterben helyezkedik el, így a szorzat kettő regiszterben keletkezik. A szorzat kisebb helyiértékű része a szorzó helyén képződik (Ezek az órai jegyzetben bizonyítva vannak)

A szorzás gyorsítása

- bitcsoportokkal való szorzás

= a léptetés nem egyesével, hanem csoportonként hajthatjuk végre ⇒ gyorsabb

= Pl. 2-es bitcsoportok:

- 00 – kettőt léptetek balra (00 -val szorzunk)
- 01 – a gyűjtőhöz hozzáadom az egyszeresét, majd léptetek kettővel balra
- 10 – a gyűjtőhöz hozzáadom a kétszeresét, majd léptetek kettővel balra
- 11 – a gyűjtőhöz hozzáadom a háromszorosát, majd léptetek kettővel balra (Példa az órai jegyzetben)

  Az utóbbi (11) csak ELVBEN létezik, ennek továbbfejlesztése:

Booth –féle algoritmus (ma is használja ezt mindegyik processzor)

= A bináris szorzáson belül az összeadási ciklus annyiszor fut, amíg egyes van a szorzóban

= Az algoritmus csökkenti a szorzóban lévő egyesek számát.

Akkor hatékony igazán, ha sok egymás utáni egyes van benne.

Példa:

Eredeti szorzó:

x*62	111110	5 db összeadás

Helyette:

x*64	1000000	1 db összeadás
x*2	000010	1 db összeadás (ezt vonjuk majd le az előzőből)
-		1 db kivonás (megvalósítása: összeadás)
		= 3 db összeadás. 40% -os gyorsulás.

SzA7. Az aritmetikai egységek felépítése III.

(a fixpontos osztás algoritmusai; a lebegőpontos algebrai műveletek és megvalósításuk)

A fixpontos osztás algoritmusai:

- Hagyományos osztás
  Ha már nem tudja kivonni, kiírja a számjegyet, szubrutin: csökkenti a szorzót 1/10 -re. Hátrány: minden kivonás előtt kell komparálnunk => lassú
- Visszatérés a nullán át:
  Előjel flag változása esetén szubrutin: kiírja a számjegyet, osztót hozzáadja az eredményhez, szoroz tízzel, majd folytatja a kivonást. Hátrány: felesleges munkát végzünk a visszatérés miatt.
- Visszatérés nélküli osztás: Amint a 0 -t átlépjük, az osztót csökkentjük a tizedére, majd ezzel közelítünk 0 -hoz, amíg azt át nem lépjük, utána folytatjuk. Flaget használ, a felesleges átlépkedést kihasználja. Ez a leggyorsabb módszer.

Geometriai értelmezéseket is tudni kell, lásd: órai jegyzet.

A lebegőpontos algebrai műveletek és megvalósításuk

Összeadás

X=A+B, ahol $A=\pm m_A * r^{\pm kA}$ és $B=\pm m_B * r^{\pm kB}$

Algoritmus

- A kitevőket megvizsgáljuk: csak azonos kitevőjű számok adhatók össze.
- Amennyiben a kitevők nem egyenlők, akkor
  - A kisebb kitevőjű szám mantisszájának törtpontját balra léptetjük, és közben inkrementáljuk a karakterisztika értékét
  - A ciklus addig fut, amíg a kitevők meg nem egyeznek.
- Mantisszákat összeadjuk, karakterisztikákat változatlanul hagyjuk
- Normalizálás szükség esetén (első értékes jegy elé tesszük a pontot)

Szorzás

$X = A*B = (m_A)*(m_B)*r^{kA+kB}$

Algoritmus: A mantisszákat összeszorozzuk, karakterisztikákat összeadjuk

Osztás

$X = A/B = (m_A)/(m_B)*r^{kA-kB}$

Algoritmus: Mantisszákat elosztjuk, karakterisztikákat kivonjuk egymásból

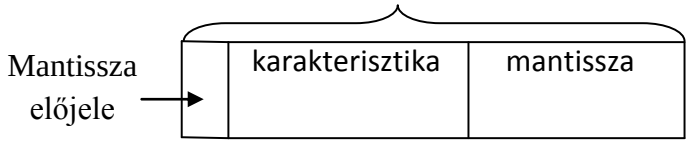
Konkrét megvalósítás

Lehetőségek:

- Univerzális végrehajtó egység, műveletvégző
- Dedikált végrehajtó egységek

- Univerzális végrehajtó egység, műveletvégző

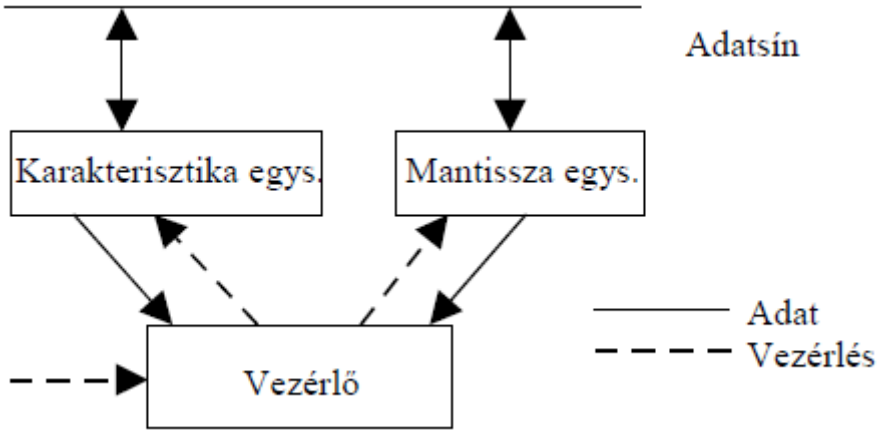
1. Általános célú **ALU** parciálásával (részekre bontásával)



A vezérlőrész bonyolítását eredményezi

2. Szervezési megoldás:
   Külön, egymás után elvégezzük a műveleteket a mantisszán és a karakterisztikán => külön regiszterekben tárolódnak műveletvégzés közben, majd a művelet után közös regiszterekben egyesítjük.

- Dedikált:



Dedikált megoldás jellemzői:

- Míg a mantissza egységnek szorozni/osztani is kell tudni, a karakterisztika egységnek elég összeadni/kivonni, ezért az utóbbi egyszerűbb
- A mantissza és a karakterisztika egység párhuzamosan is működhet (ekkor a mantissza egység jelenti a szűk keresztmetszetet a szorzás/osztás miatt, tehát azt kell igen gyors végrehajtásúra tervezni).

SzA8. Vezérlőegység

(az áramköri vezérlőegység és a mikrovezérlő jellemzőinek szembeállítása. Az áramköri vezérlőegység megvalósítása és működése)

Processzor = Műveletvégző egység + Vezérlőegység.
Centralizált (szekvenciális) vezérlés: Áramköri (huzalozott) vagy Mikroprogramozott vezérlés
Decentralizált (párhuzamos) vezérlés: Futószalagos vagy Szuperskalár vezérlés
A decentralizált vezérlés centralizált vezérlési elemekből épül fel.

Az áramköri vezérlőegység és a mikrovezérlő jellemzői:

- **Áramköri vezérlés**
  Hátrányai:
  - ember számára nehezen áttekinthető
  - merev, nehezen módosítható

  Előnye: igen gyors
  Elve: Egy forrásregiszter tartalmát a módosító áramkörön keresztül egy célregiszterbe vezetjük.
  Módosító áramkörök: inkrementálás, léptetés, invertálás, összeadás, komparálás, stb.
- **Mikrovezérlés:**
  Célja:
  - az ember számára áttekinthetővé tegye a vezérlést
    = mikroutasítások definiáltak, melyek meghatározott vezérlővonalat vagy vezérlővonalakat aktiválnak
    = mikroutasítások sorozata szolgál egy-egy gépi kódú utasítás végrehajtásának elemi szintű vezérlésére
    = a hagyományos Neumann-féle „makroszámítógépen” belül értelmezünk egy „mikroszámítógépet”, mely saját mikroutasításkészlettel rendelkezik
  - a vezérlőrész rugalmassá, könnyen módosíthatóvá alakítása
    = a mikroprogram a mikroprogramtárban helyezkedik el, így az módosítható
  - mikroutasítás például: MDR ← [MAR]

<u>A jellemzők szembeállítása:</u>

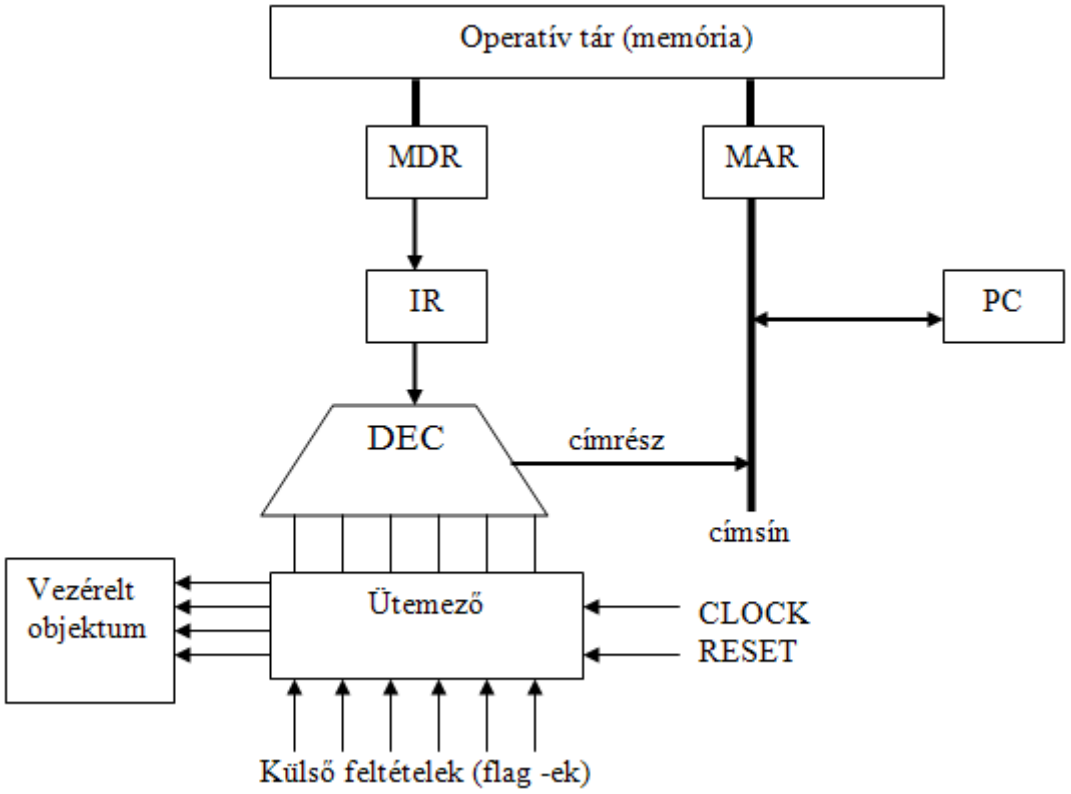
	Áramköri (huzalozott)	Mikroprogramozott
Áttekinthetőség	Ember számára nem áttekinthető	Ember számára áttekinthető
Sebesség	Mindig gyorsabb	Mindig lassabb
Módosíthatóság	Merev, nehézkesen módosítható	Mikroprogramcsere lévén cserélhető

Az áramköri (huzalozott) vezérlőegység megvalósítása és működése:

Működés:
- a vezérlő a forrásregiszter kimenő kapuját és a módosító áramkör bemenő kapuját megnyitja
- a forrásregiszter tartalma átmásolódik
- előírja a módosító áramkör számára, hogy milyen módosítást hajtson végre
  (pl. inkrementálás, léptetés, stb.)
- a módosított áramkör kimen kapuját és a célregiszter bemenő kapuját megnyitja
- az eredmény átmásolódik

Egy mai tipikus processzorban több száz vezérlési pont van (kapuk, stb.. amit vezérelni kell)

Megvalósítása:



Forrás- és célregiszterek, amelyek a vezérlésben részt vesznek:
- ALU ⇒ pl. AC, általános célú regiszterek
- Vezérlősín ⇒ IR, PC (vezérlő regiszterei)
- Memóriával kapcsolatosak ⇒ MAR, MDR
- I/O regiszterek ⇒ vezérlőkártyában (parancsregiszterek, adatregiszterek, állapotregiszterek)

SzA9. Félvezetős tárak

(jellemzőik; csoportosításuk)

Operatív tárként a félvezetős tárak előtt ferrit tárakat alkalmaztak

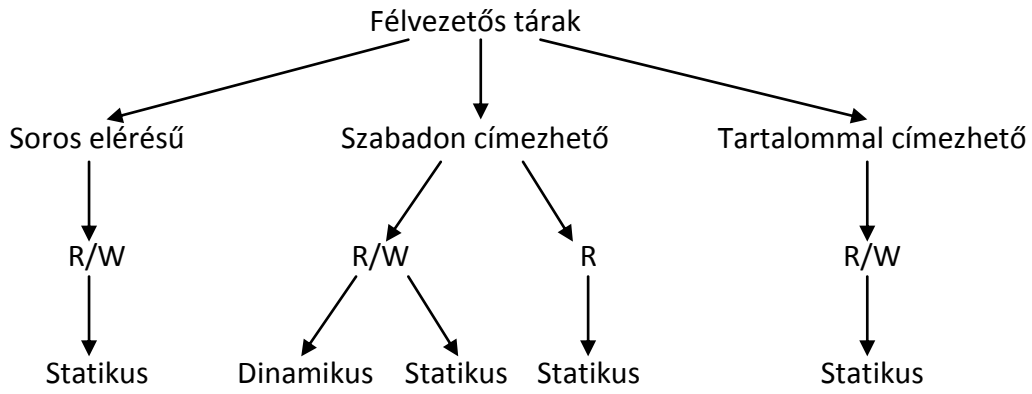
Jellemzők:

Félvezetős memóriák előnyei a többi tárral szemben:

- Működési sebesség: nagy, ~10 ns nagyságrendű (winchesteré: ms)
- Kapacitás: manapság 4 GB. A működési sebesség és a kapacitás szükséges, de nem elégséges feltétel, hogy egy eszköz elterjedjen.
- Tömeggyárthatóság: gazdaságos előállítás => kedvező ár. Pl. mágnes buborékos tárak => 60-as években IBM fejlesztette ki, gyors volt, nem volt benne mozgó alkatrész. Azonban nem lehetett gazdaságosan gyártani.
- Megbízhatóság: manapság természetesnek vesszük.
- Energiaigény: nagyságrendekkel kisebb, mint a korai típusoknál.
- Helyigény
- Modul rendszerű => bővíthető
- Az operatív tár és a CPU azonos technológiával készül => nem kell illesztés
- Táp
- Kikapcsoláskor tartalmát elveszíti.

Információ kérés igénye lép fel a tár felé, valamennyi idő múlva megjelenik a kért információ, azonban ezek nem egymás után következnek. Az információkérés kiadása után van egy bizonyos idő, miután megkapjuk a kért információt. Ezután pedig van egy bizonyos idő, miután kiadhatjuk újból az információkérést.
A **hozzáférési idő**: a kérés és a megjelenés közötti idő
A **ciklusidő**: a két kérés, tárhozzáférés közötti idő

Csoportosításuk:



RAM = Random Access Memory
ROM = Read Only Memory

Hozzáférés szerinti csoportosítás:

- Soros elérésű => FIFO és LIFO elrendezésű tárak. Az adatokhoz szekvenciálisan férünk hozzá, nem pedig tetszőleges sorrendben.
  Olvashatók/Írhatók. Az információtárolás statikus.
- Szabadon címezhető – címtől függetlenül el lehet érni az adatokat.
  Mindegy, hogy a tár melyik részén található az adat, az elérési jellemzői ugyanazok.
  Vannak Olvashatók/Írhatók => ezek információtárolása lehet dinamikus és statikus (RAM), és vannak csak Olvashatók => statikus (ROM)
- Tartalommal címezhető tár – asszociatív tár. Olvashatók/Írhatók.
  Statikus információtárolás.

Statikus RAM => SRAM.
Flip-Flop tárolja el az információt. Sebességük 1-3 ns.

Dinamikus RAM => DRAM.

- Az információt kondenzátorban tárolják el. Lassabb az SRAM –nál.
- Kisebb a fogyasztása.
- Kevesebb elemből áll
  => nagyobb az elemsűrűség: ugyanannyi tranzisztorból nagyobb tárat lehet elkészíteni
  => kisebb lesz az előállítási költség.
- A kondenzátor miatt kisebb lesz a sebesség a másik eszközhöz képest

DRAM értékelése:
Ez a technika lassabb, olcsóbb, nagyobb kapacitású eszköz kialakítását teszi lehetővé, szemben az SRAM -al, amiben több tranzisztor van, gyorsabb, relatíve drágább és kisebb kapacitású.

SRAM-DRAM összehasonlítás

Jellemző	SRAM	DRAM
Elemsűrűség		
Fogyasztás		
Sebesség		
Ár		

Az elemsűrűség-, fogyasztás-, sebesség- és árjellemzőknek a <u>következménye</u>:

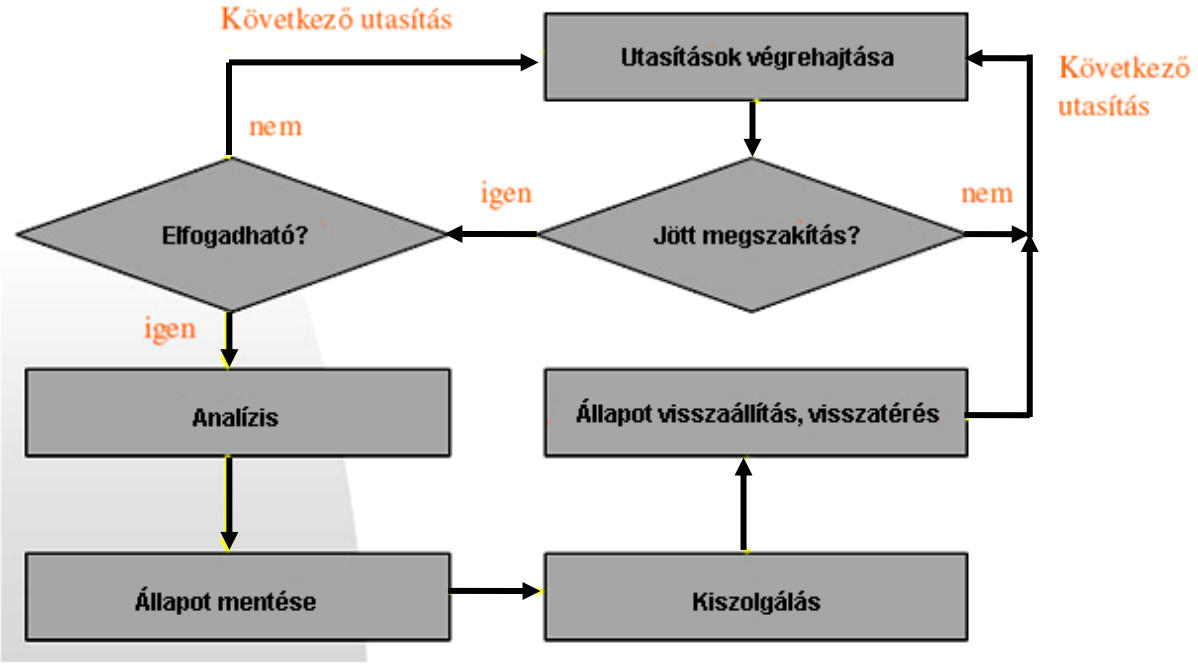
- az SRAM-ot speciális, kis kapacitás igényű helyeken használják (videokártya, processzor, cache tárak, regisztertár, winchesterben buffer tárnak =>
  nagy és alacsony sebességű eszköz összeköttetése, asszociatív tár)
- a DRAM -ot az operatív tár kialakítására használják.

SzA10. Megszakítási rendszer

(fogalma; megszakítási okok; a megszakítás folyamata; az egy- és a többszintű megszakítási rendszer)

Fogalma: A számítógép működése közben igen gyakran következnek be olyan események, amelyek a feldolgozás szempontjából váratlanak tekinthetők. Ezek a váratlan események a megszakítások (interrupt). Ezen megszakítások kezelésére szolgál a megszakítási rendszer.

Folyamata:



Az adott utasítások végrehajtása után a processzor megvizsgálja, hogy jött e megszakítás. Ha nem jött, folytatja a következő utasítást. Ha jött, meg kell vizsgálni, hogy elfogadható-e. Ha például magasabb prioritású megszakítás van folyamatban, akkor nem fogadjuk el. Ha elfogadható, következik az analízis. Meg kell nézni, mi okozta a megszakítást, és mik azok a feltételek, hogy ki lehessen szolgálni. Ezután következik az állapot mentése. Célja, hogy a megszakított programhoz vissza lehessen térni. Ezt követi a kiszolgálás, a megszakítás okának a megszüntetése. Végül az állapot visszaállítása és a visszatérés.

Az állapotok részletezése az órai jegyzetben.

Megszakítási okok prioritási sorendben:

1. Géphibák – azonnal le kell kezelni: az egész rendszerre hatással van!

- Jellemzően HW hibák
- Az egyes eszközök maguk ismerik fel a hibákat valamilyen hibajelző (pl. paritás) vagy hibajavító kóddal.
  - CPU regiszterei
  - Operatív tár - OPT
  - Adatátvitel
- Energiaellátás
- Klimatizációs hiba (hűtés)

2. I/O források:

Perifériák megszakítás kérése (CPU-n belüli), pl.: Nyomtató:

CPU => átvitel igény => nyomtató
CPU <= megszakítás <= nyomtató
CPU => adatátvitel => nyomtató

A CPU jelez a nyomtatónak, hogy szeretne nyomtatni. A nyomtató, amikor tud nyomtatni, megszakítást küld a CPU-nak, hogy jöhet a megfelelő adat. Akkor nem tud, a például nyomtat. Végül a CPU átküldi a megfelelő adatot a nyomtatónak. Ilyen esetekben a megszakítás sebességbeli különbségekből eredő problémákra is megoldást jelent.

3. Külső források:

Külső eszközök által generált megszakítások.

- Pl.: hálózati kommunikáció

4. Programozási források: utasítás végrehajtásakor

- Szándékos – rendszerhívás. Az operációs rendszer funkcióit, vagy a BIOS funkciókat rendszerhívásokon keresztül lehet elérni, ezek is megszakítási mechanizmuson keresztül működnek. Pl: INT 21h, a ténylegeshez hasonló.
- Hiba
  - Memóriavédelem megsértése: Minden program, ami betöltődik, saját memória címteret kap, ami védve van a többi programtól.
    Ha egy alkalmazás olyan címtérre szeretne hivatkozni, ami nem hozzá tartozik, akkor ilyen hiba keletkezik.
  - Tényleges tárkapacitás túlcímzése
  - Címzési előírások megsértése: Egyes esetekben meg van határozva, hogy bizonyos adatok címhatárai illeszkedjenek e 2, 4, 8… -al osztható címhatárra. Pl. C nyelvekben lehet ezt megmondani, hogy egy adatstruktúra milyen címhatárokra illeszkedjen. Ennek előnye, hogy a processzor az adott címhatárra illeszkedő memória részeket gyorsabban elérhet a memóriából. Abban az esetben, ha valamilyen előírás van, hogy az utasítások pl. 4 byte -os címen kezdődhetnek és a processzor egy páratlan címre szeretne hivatkozni, ott szeretne utasítást végrehajtani, akkor ebből hiba keletkezik.
  - Aritmetikai, logikai műveletek

Megszakítási rendszer szintek szerint

- Egyszintű – egyszerű, nem nagyon használják.
- Többszintű

Egyszintű

Két szint van: a normál és a megszakítási. 3 féle prioritás: 0,1,2. Legmagasabb a 0.

Példa:

1. fut a program a normál szinten
2. jön egy 1-es, 2-es szintű megszakítás. A megszakítási rendszer eldönti melyik a magasabb prioritású, és azt engedi érvényre jutni.
3. Közben jön egy 0-s prioritású.
   Azonban mivel két szint van, ezért az 1-est nem tudja megszakítani.
4. Befejeződik az 1-es kiszolgálása, majd a megszakítási rendszer kiértékeli milyen megszakítások várnak: 0 és 2. A 2-es csak a végén kerül lekezelésre.

A probléma, hogy a 0-s megszakítás nem jut szóhoz azonnal, hiába magasabb a prioritása. Tw időt kell várnia, mivel az 1-es megszakítást kiszolgáló utasításokat nem tudja megszakítani.

Többszintű

Több szint és több prioritás van.

Példa:

1. Jön egy 1-es, 2-es megszakítás, elkezdődik az 1-es kiszolgálása.
2. Jön a 0-ás szintű megszakítás, ami megszakítja az 1-es szintűt.
3. Amikor befejeződik, akkor folytatódik az 1-es megszakítás kiszolgálása.
4. Ennek végeztével következik a 2-es lekezelése.

Valóságban sokkal több megszakítás van. Ezzel a szisztémával akkor tudnánk őket lekezelni, ha annyi szint lenne, ahány megszakítás. Ez drága és bonyolult lenne.

Többszintű, több vonalú

Vannak prioritások, osztályok, az osztályokon belül alprioritások.

Példa:

1. Jön egy 1/a, 2/a, 2/b megszakítás, ebben az esetben az 1-es fog szóhoz jutni.
2. Közben jön egy 0/a, ami magasabb, az megszakítja 1/a-t.
3. Befejeződik a kiszolgálása 0/a-nak.
4. Jön az 1/b ami szinten belül van 1/a-val. Ő nem tudja megszakítani 1/a-t, így csak akkor jut szóhoz, ha az befejeződött.
5. Ezután jön az 1/b. Ha ez is befejeződik akkor jut szóhoz a 2/a és 2/b.

Maga a szisztéma egy több szintű megszakítás, de szinteken belül úgy viselkedik, mint egy egyszintű. Ez a módszer az előző kettő ötvözete.
Adott osztályon belüli megszakítás nem tudja félbeszakítani az eredetit.
(A magyarázó ábrák megtalálhatóak az órai jegyzetben)

SzA11. Külső sínrendszer

(fogalma; jellemzői; csoportosítása, a sínfoglalás (bus arbitration) módjai; az adatátvitel és felügyelete (szinkron, aszinkron))

Fogalma:

- Fizikai értelemben:

olyan vezetékköteg, melynek minden egyes erén egyidejűleg

= vagy a logikai nullának megfelelő ~0V

= vagy a logikai egyesnek megfelelő ~ 12V; 5V; 3,3V; 2,8V, vagy változó feszültség

jelenhet meg (felhasznált programtól függhet, pl. gépelés vagy filmnézés).

- Funkcionális értelemben:

olyan vezetékköteg, mely biztosítja, hogy a forrásból a célba egyidejűleg, azaz párhuzamosan n (16, 32, 64) db. bit juthasson el. Ebben a kontextusban a sín alatt nem csupán a vezetékeket értjük, hanem a sínfoglalást, valamint az adatátvitelt biztosító intelligenciát is.

Jellemzői:

- sínszélesség (vezetékek száma)
- tipikusan megosztott (shared) eszköz:
- minden vezetéke egy időpillanatban csak egy bitnyi információt továbbít
- regiszter-tulajdonsággal rendelkezik. Értelmezett a következő: databus ← r_0

r_1 ← databus

Csoportosítása: többféle módon lehetséges**.**

Adatátvitel iránya szerint: szimplex, félduplex, duplex.

Az átvitel jellege szerint: dedikált vagy megosztott

Az átvitt tartalom alapján: címsín vagy adatsín

Az összekapcsolt területek alapján: processzorsín (rendszersín) vagy bővítősín

(Részletesen kifejtve az órai jegyzetben)

Sínfoglalás:

Soros sínfoglalás:

- Hardver polling (daisy chain – gyermekláncfű) → hardveres lekérdezéses

Megvalósítása: órai jegyzetben.

Előnyei:

Kevés vezérlővonal → olcsó (egyszerű a megvalósítása)

Elvben végtelen számú egységet csatlakoztathatunk

Hátrányai:

A prioritás hardver úton szabályozott

Az előrébb lévő egységek elnyomhatják a hátrább lévőket

A bus grant meghibásodására érzékeny

1. bus request aktiválása (ha bus busy aktív, akkor vár, ha nem, bus grant)
2. U_1 átengedi, U_2 aktiválja a bus busy -t
3. adatot küld, bus busy deaktiválása

- Szoftver polling

A számláló algoritmus alapján szólítja meg az egységeket (pl. ott folytatja, ahol abbahagyta)

Megvalósítása: órai jegyzetben.

Előnyei:

A prioritás szoftver úton szabályozott → rugalmas

Kevésbé érzékeny a bus grant vonal meghibásodására (címvonalak)

Hátrányai:

Sok vezérlővonal → drága

A csatlakoztatható egységek számát a bus grant vonalak száma határozza meg pl. 3 vezeték esetén 23 = 8 db egység

Párhuzamos sínfoglalás

Megvalósítása: órai jegyzetben.

Előny: igen gyors

Hátrány: több vezeték kell hozzá → drágább (bonyolultabb vezérlés)

Rejtett sínfoglalás:

- két független hardver végzi a sínfoglalást és az adatátvitel vezérlését
- lehetőség van arra, hogy amíg az előző átvitel zajlik, a következő egység kiválasztása megtörténhessen. Amint az adatsín felszabadul, azt átadja.

Pl. PCI bus

Az adatátvitel lehet szinkron és aszinkron működésű.

Szinkron adatátvitel:

Fogalma: az adatátvitel mind az adó(forrás), mind pedig a vevő (cél) számára egy előre ismert időintervallumban történik, ezt az órajel biztosítja

Óraütem-adó:

- mind az adó, mind a vevő egy közös órajel-adótól kapja az órajelet: Akkor alkalmazzák, ha kicsi a távolság az adó és a vevő között
- mind az adó, mind a vevő saját óraütem-adóval rendelkezik, melyek azonos frekvenciájúak. Ekkor meghatározott időközönként a működésüket szinkronizálni kell → szinkronjel

Értékelése:

Előnye:

Olcsó az előállítása, mert egyszerű

Hátrány:

az átvitel során előre ismert intervallum hosszát mindig a leglassabb egység határozza meg → ez visszafogja a gyors egységeket (HDD, monitor, CPU).
Ez kiküszöbölhető többszintű sínrendszer alkalmazásával, ahol átviteli sebesség tartományonként csoportosítják az egységeket

A PC-n belül a szinkron átvitel dominál (ISA - 16 bit, EISA, PCI – 32/64 bit, AGP)

Aszinkron átvitel

Fogalma: az adott elemi művelet befejeződése egyben jelzés a következő elemi művelet kezdetére.

Fajtái:

- egyvezetékes (egy vezérlővezetékes)
  - adó oldali vezérlés: Felteszi a sínre, jelzi, vár, leveszi.
    Értékelés: az adónak nincs visszacsatolása az adat célba érkezéséről.
    (lehet, hogy a vevő évek óta rossz)
  - vevő oldali vezérlés → vevő küldi a vezérlőjelet
    Vevő kérésére történik az adatküldés.
    Értékelés: az előzőnél megbízhatóbb, hiszen a vevő a kérés pillanatában aktív

  Az egy vezetékes átvitel hátránya:
  Az adó nem kap visszacsatolást az adat célba érkezéséről.

- kétvezetékes vagy handshake (kézfogásos) átvitel
  - adó oldali vezérlés
    Átvitel folyamata: Az adó felteszi az adatot az adatsínre, aktiválja a Data ready vonalat, a vevő az adat elolvasása után aktiválja a Data ack vonalat, majd az adó visszaveszi az adatot. Ezután az adó a Data ready -t, a vevő pedig a Data Ack -t deaktiválja (a jegyzetben hibásan szerepel: Request helyett Ack vonal!)
  - vevő oldali vezérlés

  Az adó visszacsatolást kap az adat célba érkezéséről →
  megbízható átvitelt biztosít különböző sebességű eszközök esetén is

A magyarázó ábrák az órai jegyzetben szerepelnek.

SzA12. A processzor részvételével zajló I/O rendszer

(a programozott I/O, a különálló I/O címtér és az I/O port; a memóriában leképezett I/O címtér; működése (feltétlen és feltételes))

Az I/O rendszer fogalma:

A processzor-memória együttest a külvilággal összekapcsoló rendszer.

A programozott I/O fogalma:

Minden egyes I/O művelethez a processzor egy-egy utasítást hajt végre.

1. Különálló I/O címtér
   Elve: A CPU két különálló címteret lát: RAM és I/O címtér.
   Jellemzői:
   - ugyanazon a címsínen keresztül haladnak a memóriacímek és az I/O címek is (rendszersín)
   - létezik egy M/I/O vezérlővezeték, mely megmondja, hogy az adott időpillanatban memória- vagy I/O cím van a címsínen.
   - mivel két különálló címtérről van szó, ugyanaz a cím szerepelhet memóriacímként és I/O címként is.
   - Intel esetében a 32 bites címsínnek az I/O egység címzésére szolgáló része 16 bit hosszú => 2^16=65536 féle I/O cím adható ki.

I/O port:
Azok regiszterek, amelyeken keresztül a processzor a perifériákkal kommunikálhat.
Az I/O port fizikailag az I/O vezérlőegységben helyezkedik el.

Az I/O port regiszterei:

- Parancs (Command) regiszter: ebbe írja a processzor a kívánságait
- Adatregiszterek: bemeneti és kimeneti (a CPU szemszögéből, a másik fél a periféria)
- Állapot (status) regiszter: innen olvassa a processzor a periféria üzeneteit
  A mai gyakorlat: egy-egy közös regiszterben valósítják meg a parancs- es állapot regisztert, valamint a két adatregisztert.

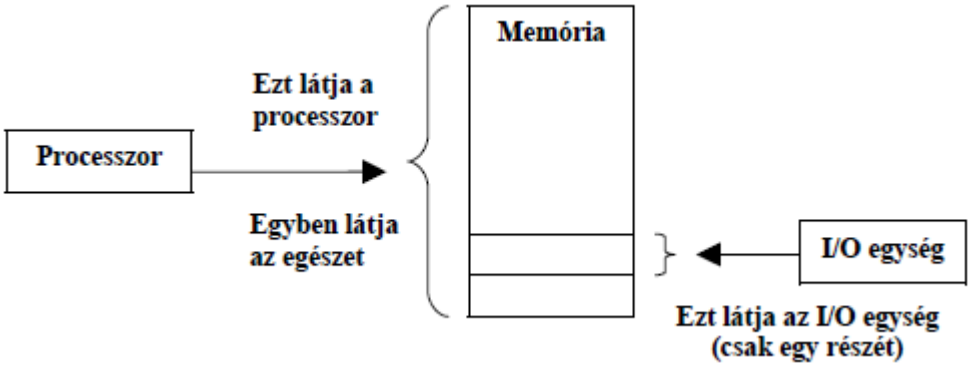
Napjaink további regiszterei az I/O porton belül:

- Az eszköz jelenlétét jelző regiszter
- Az eszköz tulajdonságait tartalmazó regiszter (plug&play)
- Lehet több regiszterkészlet is

A különálló I/O címtér értékelése:
Előny: egyszerű, olcsó a megvalósítása
Hátrány:
A processzor reszt vesz a kommunikációban
Az AC szűk keresztmetszetet jelent nagy tömegű I/O számára

Ezt az eljárást minden mai architektúra alkalmazza (pl. billentyűzet, soros és párhuzamos port). Az IBM PC -nél különálló I/O címtérrel csatlakozik a hálókártya.

2. A memóriára leképzett I/O
Elve: CPU → RAM → Periféria



Jellemzői:

- Megosztás: a processzor memóriakezelő utasítással (load/store) éri el azt a közös memóriaterületet, amit a periféria is kezelhet
- A perifériának hozzá kell férnie a rendszersínhez → igen gyors átviteli sebesség

Értékelése:
Előny: igen gyors (a különálló I/O címtérnél sokkal gyorsabb)
Hátrány: továbbra is a CPU -nak kell utasításokat végrehajtani az I/O műveletnél

Minden mai architektúrában megtalálható.
Példa: a PC környezetben képernyő (video) kezelés

A programozott I/O működése:

Feltétlen átvitel:

- a vevő mindig vételre kész állapotban van
- nem ellenőrizzük az átvitel sikerességét
- nincs szinkronizálás az adó és a vevő között

➡ Nem egészen biztonságos átvitel
Például: LED

Feltételes átvitel:

- lekérdezéses vagy „wait for flag”
  1. az első lépés során a processzor beírja kívánságát az I/O port parancsregiszterébe
  2. a processzor kiolvassa az I/O vezérlő állapotregiszterének tartalmát
  3. amennyiben nem „ready”, akkor vissza a 2. pontba
  4. amennyiben „ready”, akkor a processzor kiolvassa az I/O vezérlő adatregiszterének tartalmát, és azt eljuttatja az AC -ba

  Mivel a processzor és a periféria sebessége között igen nagy különbség lehet, a 2. és 3. pont olvasási ciklusa akár több milliószor is feleslegesen fut, pazarolja a CPU időt

- megszakításos átvitel
  1. a processzor beírja kívánságát az I/O parancsregiszterébe és elkezd mást csinálni
  2. az I/O egység gondoskodik arról, hogy a kívánt perifériáról a kívánt adat beírásra kerüljön az I/O port adatregiszterébe, és ekkor az állapotregiszter „ready” bitjét beállítja, továbbá megszakításjelzést küld a processzor felé
  3. a processzor a következő utasítás-töréspontban észleli a megszakítást és forrását:
     - kiolvassa a megszakító I/O port állapotregiszterének tartalmát
     - mivel ott a „ready” bit be van állítva, ennek megfelelő megszakítás-feldolgozó programot indít el; ez kiolvassa az I/O port adatregiszterét és tartalmát átviszi az AC –ba.

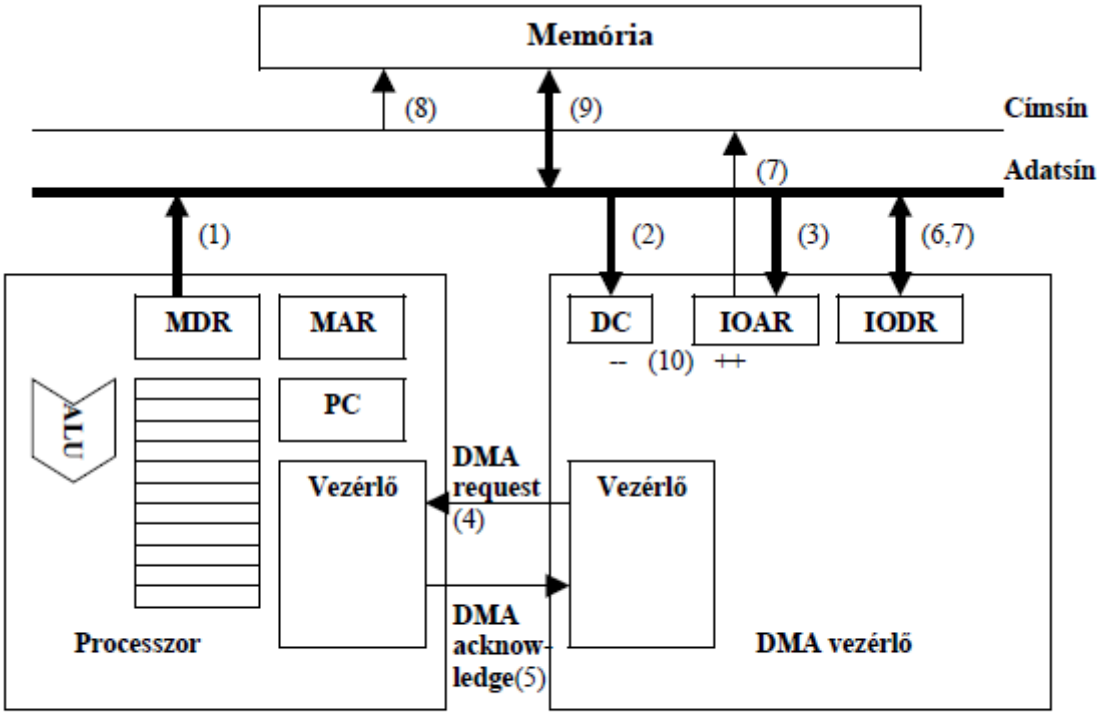
A különálló I/O címteres és a memóriára leképzett I/O címteres módszerek megvalósítását az órai jegyzetben található ábrák mutatják be.

SzA13. A közvetlen memória-hozzáférés (DMA)

(fogalma; megvalósítása; működése: blokkos és ciklusloposos üzemmód)

Fogalma: nagy tömegű adat gyors periféria alkalmazásával történő átvitele, a processzor közreműködése nélkül

Megvalósítása:



DC – Decrementer vagy Data Counter (átvivendő adat mennyisége)
IOAR – I/O Address Register
IODR – I/O Data Register
DMA request – igény benyújtása a rendszersínre

Működése:

Előkészítés:

A DMA vezérlő „felprogramozása”: programozott I/O-val átvisszük a processzorból a DMA vezérlőbe az átvitelhez szükséges információkat (mit, honnan, hová kell vinni):

- a DC -be beírjuk az átviendő adategységek számát
- IOAR -be beírjuk az adat leendő memóriabeli kezdőcímét, továbbá:
  - az egységet (byte, félszó, szó)
  - az átvitel irányát
  - a periféria címét
  - az átvitel jellegét blokkos vagy ciklusloposos módon
  - a résztvevő egységeket
    (memória - I/O, memória - memória vagy I/O – I/O átvitel)

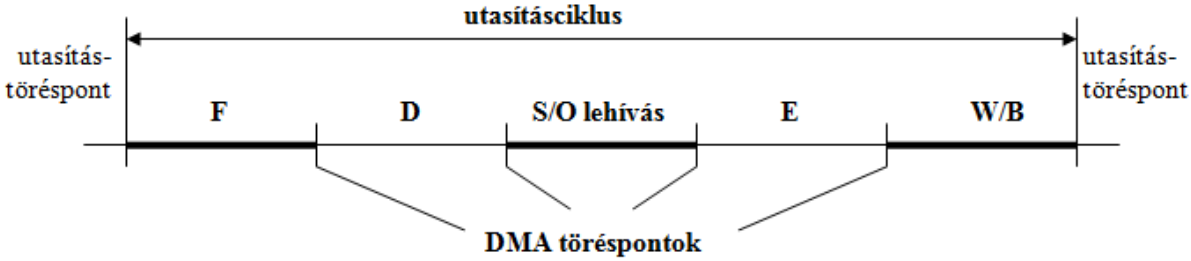
Működés blokkos (burst) átvitel esetén (pl. winchester esetén):

- A CPU felprogramozza a DMA vezérlőt (1-3)
- a DMA vezérlő DMA Request jelzést küld a processzornak:
  kéri a rendszersín használati jogot (4)
- a processzor DMA Acknowledge jelzéssel lemond a rendszersín használati jogáról (5)
- a DMA vezérlő a kapott adatok alapján a perifériáról beírja az első átviendő adatot
  az IODR -be (6)
- a DMA vezérlő az IODR -ben lévő adatot a rendszersínen keresztül beírja az IOAR által meghatározott memóriacímre (7-9)
- a DMA vezérlő dekrementálja a DC-ben tárolt értéket, és inkrementálja az IOAR -ben tárolt értéket (egy adategységgel növel – 1,2,4 byte) (10)
- DMA ellenőrzi a DC tartalmát. Ha nem 0, vissza a (6) -ra, ha igen, megszakításkéréssel jelzi a CPU felé, hogy befejeződött egy blokk átvitele (pl 3200 byte lehet egy HDD nél)

(Előfordulhat, hogy az órán más számozással szerepeltek a lépések, de a sorrend elvileg helyes)

Működés ciklusloposos (cycle stealing) átvitel esetén:

pl. gyorsnyomtató: karakteres szervezésű adat kezelése esetén



F - fetch
D – dekóder
E – executing
S/O – forrás operandus letöltés
W/B – visszaírja az eredményt
a processzor használja a rendszersínt
nem használja

- Nincs értelmezve a címgenerálás
- Míg az utasítás-töréspontban a megszakítás feldolgozással a CPU –ra további munka várhat, addig a DMA töréspontban a DMA vezérlő a processzor helyett végezhet munkát
- Ez a processzor és a DMA vezérlő általi időosztásos rendszersín használat
- Elve: utasítás-feldolgozás felbontása pl. a következő lépésekre:
  lehívás, dekódolás, operandusok lehívása, végrehajtás, visszaírás a memóriába

SzA14. Az egyes alkotóelemek összerakása

(egy hipotetikus számítógép tervezése és működése)

Egy hipotetikus számítógép tervezése

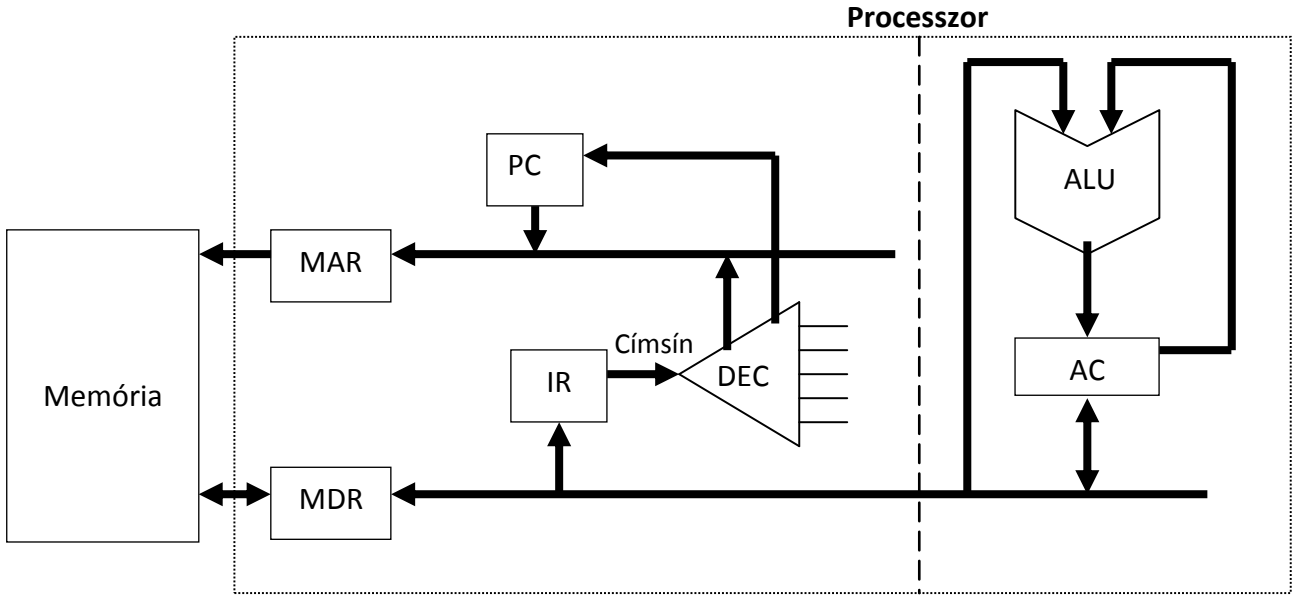
Jellemzői – korlátai:

- Minden utasítás két byte hosszú, ebből a címrész 1 byte → a címtér: 256 cím

Műveleti Kód	Címrész
1 byte	1 byte

- Csak processzorból és memóriából áll (nincs kapcsolat a külvilággal, az adatok valahogy bejutottak a memóriába)
- Utasításkészlet:

ADD 100	AC:=AC+100
ADD[100]	AC:=AC+MEM[100]
INC	AC:=AC+1
NUL	AC:=0
LOAD[100]	AC:=MEM[100]
STORE[100]	MEM[100]:=AC
JUMP120	PC:=120



Utasítás lehívás (Fetch)

MAR ← PC
MDR ← (MAR)
IR ← MDR
PC ← PC+1

Ehhez hasonlóan a többi műveletet is ki kell tudni fejteni, pl. Load, ADD, feltétlen vezérlésátadás

PC tartalma: 2-vel inkrementálódik itt a PC

100	LOAD[200]	
102	ADD[201]	
104	STORE[202]	
106	NUL	(Ac ← 0)
108	JUMP 150	(108 -ról 150 -re átírjuk a PC -t)

III. Korszerű mikroarchitektúrák (fizikai architektúra)

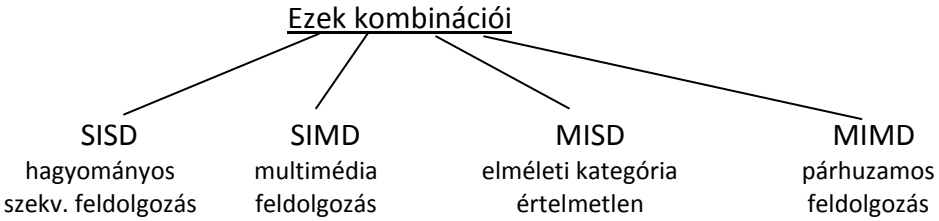
SzA15. Számítógép architektúrák osztályozása

(Flynn-féle, illetve korszerű osztályozás)

Architektúrák osztályozása

Flynn -féle osztályozás:

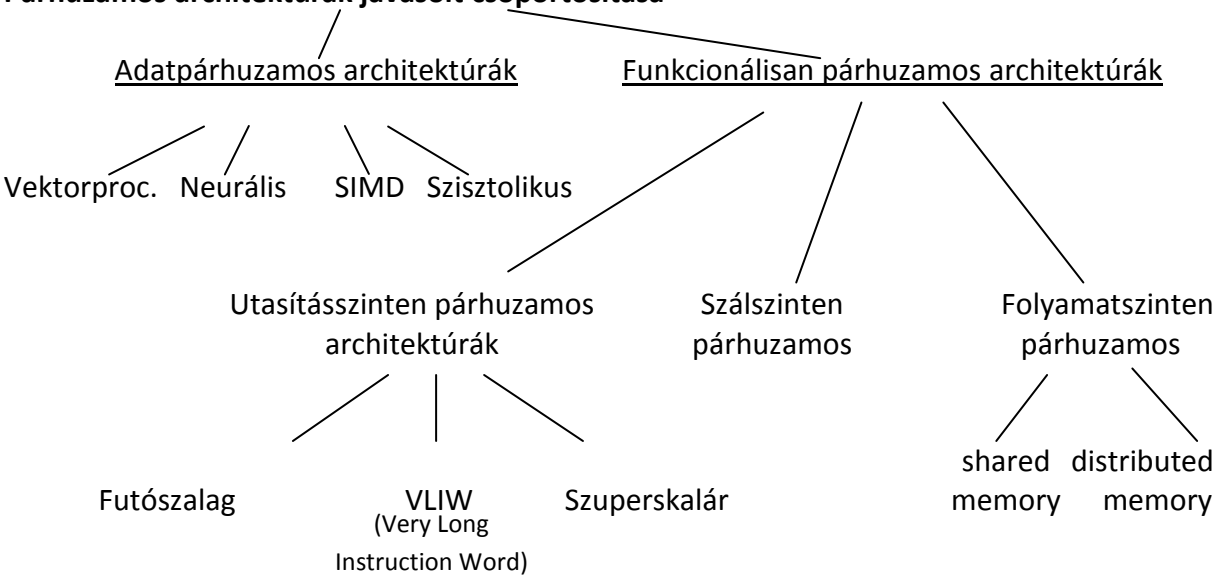
- Fogalmak:
  - SI – Single Instructions: a vezérlő egyetlen utasításfolyamot bocsát ki
  - MI – Multiple Instructions: a vezérlő több független utasításfolyamot bocsát ki
  - SD – Single Data: egyetlen utasításfolyamot dolgoz fel
  - MD – Multiple Data: A végrehajtó egység több utasításfolyamot dolgoz fel



Értékelés:

- előnye: egyszerű és jól átlátható
- hátránya: nem mutatja meg a párhuzamosság fajtáját, szintjét és a kihasználásának módját

Párhuzamos architektúrák javasolt csoportosítása



2000 –ig szuperskalár, utána szálszinten párhuzamosítottak.

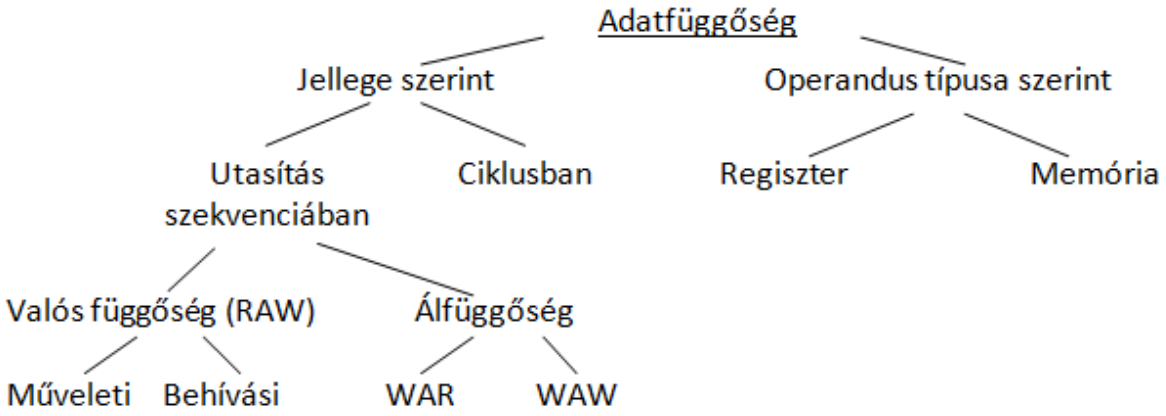
SzA16. Adatfüggőségek

(fogalma, főbb fajtái, teljesítmény-korlátozó hatása)

Fogalma:

Az utasítás kibocsátása nem történhet meg addig, amíg nem áll rendelkezésre az összes operandus. Például: az egymást követő utasítások ugyanazt az adatot használják.

Főbb fajtái:



(Read After Write, Write After Read, Write After Write)

Műveleti adatfüggőség:

Ha például egy művelet eredményét akarjuk felhasználni: I_1: ADD $r_3r_2r_1$ és I_2: SHL r_3
A probléma kezelése: operandus előrehozással, extra HW (regiszter) segítségével.
Ehhez a CPU –nak érzékelnie kell, hogy az adatra még szükség lesz, így az ALU –ba is visszavezeti azt, az eredeti r_3 cím mellett. Ez valós adatfüggőségnek tekinthető, csak mérsékelni tudjuk, kiküszöbölni nem. Minden mai processzor használja.

Lehívási adatfüggőség: A CPU a gyorsítótárból a regisztertárba szállítja az adatot. A CPU érzékeli, hogy erre neki is szüksége lesz, ezért extra HW -vel operandus előrehozás.
Ez is valós adatfüggőség, az előzőhöz hasonló módon csak mérsékelni lehet a hatását.

WAR: Egy lassabb művelet forrásoperandusát felülírhatja egy későbbi gyorsabb utasítás eredménye, ha ugyanaz a regiszter szerepel mindkét helyen. Megoldás: regiszterek átnevezése. A valóságban a CPU minden eredményregisztert átnevez (átmeneti reg. készlet)

WAW: Két egymást követő utasításnak ugyanaz az eredményregiszter van kijelölve.
Megoldás: szintén regiszterátnevezéssel. Történhet statikusan és dinamikusan is.

Ciklusbeli adatfüggőség: Egy ciklusmagon belül pl. a x(i):=A(i)*x(i-1)+B(i) utasítás esetén (ahol i a ciklusváltozó) nem tudunk továbblépni az x(i-1) kiszámolása nélkül. Erős a függés, ezt az algoritmus átalakításával lehet feloldani.

SzA17. Vezérlésfüggőségek és teljesítmény korlátozó hatásuk csökkentése

(vezérlésfüggőségek fogalma, teljesítmény korlátozó hatása és annak csökkentése, a feltétlen vezérlésátadás, a statikus és dinamikus elágazásbecslés, valamint a spekulatív elágazáskezelés elve)

A vezérlésfüggőség fogalma:

Az utasítások végrehajtása függ annak környezetétől, akár feltétlen, akár feltételes vezérlésátadás előzi meg azokat.

Feltételes vezérlésátadás esetén a feltétel kiértékeléséig nem lehet megmondani, melyik irányban halad tovább a végrehajtás; a következő utasítás címe függ a feltételtől.

A feltétlen vezérlésátadásnál az a probléma, hogy feleslegesen hívunk le utasításokat, hiszen biztosan nem ott folytatjuk a végrehajtást, emellett a regiszterek tartalmát is veszélyeztetjük

A vezérlésfüggőségek teljesítménykorlátozó hatása és annak csökkentése:

Teljesítménykorlátozó hatás:

- Feltétlen elágazás esetén a feleslegesen végrehajtott utasítások száma a futószalag fokozatainak száma -1, tehát n fokozat esetén n-1. Ezt nevezzük ugrási résnek (buborék).
- Feltételes elágazás esetén, ha a feltételben szereplő adat még nem áll rendelkezésre, a feleslegesen végrehajtott utasításokon kívül további időt is igénybe vehet a kiértékelés.

A teljesítménykorlátozó hatás csökkentése:

- Feltétlen elágazás esetén fel kell tölteni az ugrási rést.
  Ennek **statikus** kezelése abból áll, hogy a buborékot feltöltik NOP –okkal. Ezzel szintén felesleges műveleteket végzünk, de a regisztertartalmakat nem veszélyeztetjük.
  A **dinamikus** módszert az optimalizáló compiler –ek alkalmazzák, itt az utasításszekvenciát módosítják olyan módon, hogy az adatmanipuláló utasítások ne az elágazási utasítás előtt, hanem azt kövesse, így nem válik feleslegessé a vezérlésátadó utasítások végrehajtása.
- Feltételes elágazási utasítások esetén beszélhetünk ugrási irány becslésről, az ugrási cím elérésének gyorsításáról, illetve feloldatlan elágazások kezeléséről (spekulatív elágazásbecslés)

A statikus elágazásbecslés: a becslés a programkód alapján történik, a compiler által. Az első generációs szuperskalárok alkalmazzák. A ciklusokat jól ismeri fel, ezáltal hatékony.

A dinamikus elágazásbecslés: a hardver feladata az elágazásbecslés megoldása. Ugrástörténet alapján történik a becslés extra eszközök segítségével. Példa: Pentium I (egyszintű), Pentium Pro, Pentium MMX (kétszintűek).

A spekulatív elágazásbecslés: a feloldatlan elágazások kezelése

Feloldatlan (vagy függő) elágazás fogalma: ha a feltétel kiértékelése valamilyen nagy késleltetésű művelet eredményétől függ. Futószalag és első generációs szuperskalárok esetén blokkol.

A függő elágazások kezelése: az alapblokkokat kiterjesztjük virtuális alapblokkokká. Egy virtuális alapblokk több alapblokkból áll. A folytatási irány megbecsülése (lásd korábban!) után lehívjuk az utasításokat, majd a feltétel kiértékelésekor vagy eldobjuk a lehívott és végrehajtott utasításokat, vagy pedig folytatjuk a végrehajtást. Ez akár több tucat utasítás feldolgozását is jelentheti, azok „véglegesítése" a ROB segítségével történik meg, amennyiben a feltétel kiértékelése megerősíti a becslést.

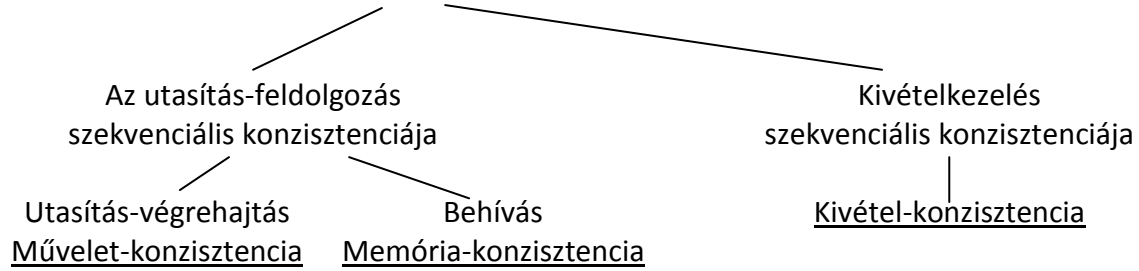
Alapvető mutató: a spekuláció mélysége. Annál több utasítás lehet a várakoztató állomásban, minél mélyebb a spekuláció, ugyanakkor, ha mégis rossz a becslés, annál több utasítást kell eldobnunk. A mélység általában 2 és 7 között van.

A spekulatív elágazásbecslés infrastrukturális feltételei: átnevezési regisztertár, ReOrder Buffer (a rollback –hez) => második generációs szuperskalárok.

SzA18. Szekvenciális konzisztencia

(az utasítás-feldolgozás és a kivételkezelés soros konzisztenciája, a precíz megszakítás-kezelés)

Szekvenciális (soros) konzisztencia (ILP –kkel szembeni elvárások egyike)



Művelet-konzisztencia

A probléma felvetése: több egymást követő utasítás esetén a JZ feltételes utasítás kiértékelése nem egyértelmű az utasítások eltérő végrehajtási ideje miatt.

Szekvenciális végrehajtás:

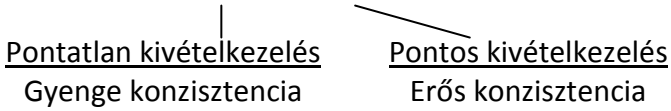
Először az első (pl. MUL), aztán a második (pl. ADD) utasítás hajtódik végre, a címkére ugrás pedig akkor következik be, ha a második (ADD) eredménye 0.

Párhuzamos feldolgozás:

A lassabb (MUL) utasítás fejeződik be később, tehát biztosítani kell, hogy a címkére ugrás csak akkor következzen be, ha a második (ADD) eredménye 0.

Memória-konzisztencia: nem foglalkoztunk vele.

Kivételkezelés soros konzisztenciája



- Pontatlan kivételkezelés
  Probléma felvetése: lásd az előző eset.

  Ha párhuzamos esetben először az ADD fejeződik be, s tegyük fel, hogy túlcsordul: amennyiben a processzor a megszakítást soron kívül elfogadja, gyenge konzisztenciát eredményez. A MUL utasítás ekkor definiálatlan állapotba kerül, nem lehet tudni, befejeződik-e, vagy sem. A megfelelő állapot helyreállítása csak bonyolult eljárásokkal lehetséges. Alkalmazása: korai szuperskalároknál.

- Pontos kivételkezelés
  A korszerű processzorok ezt alkalmazzák, azaz a megszakítást kizárólag az utasítások eredeti sorrendjében fogadja el.
  Megvalósítása: átrendező-puffer (ReOrder Buffer, ROB) használatával.
  A processzor csak akkor fogadja el a megszakításkérést, amikor az adott utasítást kiírjuk az átrendező-pufferből.
  Pl.: Intel processzorcsalád, és a mai korszerű processzorok.

SzA19. Az elágazások vizsgálata

(Az elágazások csoportosítása, a feltételes utasítások használata, a műveletek eredményének vizsgálata az állapottér módszerrel és közvetlen adatvizsgálattal, az elágazási utasítások aránya az utasítás-mixben, a feltétlen és feltételes elágazási utasítások arányai, az elágazások teljesülési és nem teljesülési arányai)

Elágazások fajtái: Feltétlen és feltételes elágazás.
Feltétlen: Egyszerű feltétlen (GOTO, JMP), eljáráshívás, visszatérés az eljárásból
− Egyszerű feltétlen elágazások: mindig ugrik
− Eljáráshívás: az ugrás előtt elmentik a visszatérési címet (LIFO elvű tárolás)
− Visszatérés az eljárásból: felhasználva a visszatérési címet
Feltételes elágazások: Cikluszáró illetve egyéb feltételes elágazások (utóbbi pl: CASE, IF)

Elágazások használata: Két okból használjuk:
- Az utasításban meghatározott művelet eredményét vizsgáljuk, pl.: nulla, negatív, átvitel
- Két operandus összehasonlítása: ha megegyeznek, ugrunk, ha nem, soros folytatás
  Visszavezethetjük a műveletek eredményének vizsgálatára, kivonjuk egymásból őket és az eredményt vizsgáljuk, tehát gyakorlatilag visszavezettük az előző esetre

A műveletek eredményének vizsgálata az állapottér módszerrel:
- Az állapotvizsgálatot a CPU automatikusan végzi, és az eredményt beírja
  - feltétel kódba (CC) vagy
  - jelzőbitekbe (flag)
- Működése
  - aktualizálása: minden művelet után a CPU karbantartja az állapotteret, a megfelelő biteket bebillenti a jelzőbitek közül (pl.: nulla az eredmény, túlcsordulás)
  - ezt követően a programozó tesztelheti a számára szükséges jelzőbitet, és annak értékétől függően ugrás, vagy soros folytatás
- Példák: IBM 360/370, Intel, Motorola, PowerPC, SPARC

- Értékelés
  - Hátrányok
    - Ebben az esetben a tesztelés további időt igényel (párhuzamosítható)
    - Kiegészítő hardvert is igényel
    - Megszakítás esetén az adattér mellett az állapotteret is le kell menteni, és ehhez az operációs rendszerre is szükség van
    - Alapvetően szekvenciális szemléletű
    - Párhuzamosítás esetén:
      = Több jelzőbit-készletet alakítanak ki
      = Minden utasításhoz hozzá kell rendelni a megfelelő jelzőbit készletet
      = Tesztelésnél nekünk a megfelelő jelzőbit-készletet kell tesztelni
    - Az állapottér aktualizálása és tesztelése szekvenciális, ami valós függőséget eredményez. Ez visszafogja a teljesítményt, lassít.
    - (A közvetlen adatvizsgálat kiválóan alkalmas a párhuzamos feldolgozásra)
  - Előnyök
    - A tárgykód 1-7% -kal rövidebb (nem jelentős manapság)
    - A jelzőbit (flag) vizsgálata rendkívül gyors
- Következtetés: A hátrányok súlyosabbak, mint az előnyök, tehát a jövőben inkább a közvetlen adatvizsgálat elterjedése várható

A műveletek eredményének vizsgálata közvetlen adatvizsgálattal:

Nem értelmezett az állapottér

- Fajtái:
  - Kétutasításos megoldás:
    - Az egyik utasítással tesztelünk, és az eredményt egy regiszterben tároljuk
    - A másik utasítással a regiszter tartalmát vizsgáljuk
  - Egy utasításos megoldás
    - Egyetlen utasításban hajtódik végre a tesztelés és a feltételvizsgálat
- Értékelés: párhuzamos feldolgozásra is kiválóan alkalmas, jövőbeli fejlesztésekben várható az alkalmazása

Példák az órai jegyzetben.

Az elágazási utasítások aránya az utasítás-mixben:

Az általános célú programokban az utasítások 20% -a feltételes vezérlésátadás, tudományos-műszaki célú programok esetén ugyanez 5-10%

A feltétlen és feltételes elágazási utasítások arányai:

Gromovski szerint:Feltétlen elágazások 1/3, Cikluszáró feltételes 1/3, Egyéb feltételes 1/3

⇨ **Feltétlen: 1/3 és feltételes: 2/3**

Az elágazások teljesülési és nem teljesülési arányai:

Forrás	**Fajta**	**Teljesült**	**Nem teljesült**
Gromovski	Feltétlen	Mind	
	Feltételes cikluszáró	n-1 (~ mind)	
	Egyéb feltételes	1/6	1/6
Összeg		**5/6**	**1/6**
Egyéb I.		**75%**	**25%**
Egyéb II.		**57-99%**	**1-43%**
Végeredmény		***75%***	***25%***

SzA20. Az utasítások időben párhuzamos feldolgozásának alapvető lehetőségei

(prefetching, átlapolt utasítás végrehajtás, futószalag elvű feldolgozás, ebből adódó szűk keresztmetszetek (memóriasávszélesség 15.o és elágazások) feloldása)

A felsorolt fogalmak a futószalagos feldolgozás előzményei. Az utasítás feldolgozását feloszthatjuk fokozatokra, erre egy példa a következő:

Fetch: Az utasítás lehívása
Decode/Source Op.: Az utasítás dekódolása és a forrásoperandusok lehívása
Execute: Az utasítás végrehajtása
Write Back (W/B): az eredmény visszaírása.

Prefetching:
Előlehívás. A Fetch és a Write Back fokozatot egymástól függetlenül el lehet végezni, azaz az aktuális utasítás eredményének kiírása és a következő utasítás lehívása párhuzamosítva van. Maximum 1 óraciklust nyerünk vele, amennyiben nincsenek függőségek. Intel 80286.

Átlapolt utasítás végrehajtás (Pipelined EUs):
Az utasítások végrehajtási fokozatait párhuzamosítjuk. Például a 60-as években az IBM 360. A lebegőpontos számolásoknál például párhuzamosítani lehet a különböző részeit a műveletnek, a vektorprocesszoroknál használták. Mikroprocesszoroknál ez a fejlődési lépés kimaradt.

Futószalag elvű feldolgozás (Pipelined Processors):
A teljes utasítás feldolgozási ciklus párhuzamosítva van. Elvben 4 utasítás feldolgozása ciklusonként, ezt a függőségek akadályozzák. Példa: Intel 80386.

Szűk keresztmetszetek:
Memória sávszélesség:
A szekvenciális utasítás feldolgozásánál 4 óraciklusonként történt memóriához fordulás, a futószalagok esetén akár ciklusonként is történhet. A DRAM nem képes ekkora sebességgel történő kiszolgálásra. **A megoldás**: a cache megjelenése, 80-as évek végétől, pl. Intel 80486.

Elágazábecslés:
Féltetlen elágazásnál az ugrási rés (buborék) méretének megfelelő számú utasítás lehívását blokkolja az elágazási utasítás. Ha „n” a futószalag fokozatainak száma, akkor az ugrási rés mérete n-1. Feltételes elágazás esetén ehhez hozzáadódik a feltétel kiértékelése és az ugrási cím számítása is (1-2 óraciklustól akár 50ig is tarthat). **Kezelése**: korai RISC –eknél késleltetett ugrás, korai CISC –nél hardveresen (dekódolási fokozatban történik meg a feltétel kiértékelése és az ugrási cím számítása), késői CISC –eknél fix előrejelzés, pl 80486. A másfeledik generációs futószalagoknál van cache, de nincs még (fix) elágazásbecslés, a második generációsoknál (pl 80486) már van mindkettő. Fix elágazásbecslés: mindig ugrik.

SzA21. A futószalag (pipeline) elvű utasítás-végrehajtás

(a futószalag elve; jellemzői; logikai és fizikai futószalagok)

A futószalag elve:

Az utasításfeldolgozási ciklust felbonthatjuk fokozatokra, pl: F D/So E és WB (lásd korábban).
Az utasításokat átlapolva dolgozzuk fel, így bár egy utasítást ugyanannyi idő alatt lesz feldolgozva, de óraciklusonként 1 utasítás kerül végrehajtásra.
A futószalagos feldolgozás előfeltételei: a fokozatok időigénye egyezzen meg, egymástól független hardverek alkossák, az egyik szakasz kimenete legyen a következő bemenete, valamint mindegyik fokozat órajelre lépjen működésbe.
Nagy utasításszám esetén elvben olyan mértékben gyorsul a feldolgozási sebesség, amennyi fokozatból áll a futószalag. Ezt hátráltatják a függőségek

A futószalagos feldolgozás jellemzői:

A fokozatok száma: a fokozatok számának növelésével egy idő után csökkenni fog a teljesítmény a függőségek miatt: sok lehívott utasítást kell eldobni, ha rossz irányba tettük.
80-as években: 2-3 fokozat, 2000-ben kb. 10, napjainkban 15-25. Nem nő tovább.

Operandus előlehívás: FX szorzás/osztás esetén a részeredményeket az Execute fokozat bemenetére visszavezethetjük annak kimenetéről. Extra hardverrel, lásd korábban.

A processzorok napjainkban mind **szinkron feldolgozással** (ütemezéssel) működnek: Aszinkron esetben a feldolgozás folyamatos, amit úgy valósítanak meg, hogy az egymást követő fokozatok jelzik, amikor elkészültek egy feladat végrehajtásával és képesek egy újabb feladat fogadására. Szinkron esetben órajel hatására, azonos időben kezdenek hozzá a fokozatok a feladatok végrehajtásához, az ütemet a legtöbb időt igénylő egység szabja meg.

A futószalagok logikai felépítése:

I. szint: a futószalagok funkcionális kialakítása: lehetőleg mindegyik utasításkategóriához 1-1 cél-futószalag kialakítása, pl. FP, FX, L/S (Load/Store), B (Branch).
II. szint: egyes fokozatok által végrehajtandó elemi műveletek (mikroutasítások) specifikálása
Példa: lásd órai jegyzet.

A futószalagok általános fizikai felépítése:

60-80-as évek, futószalagok: a fokozatok végrehajtó egységei között elválasztó (rejtett) regiszterek használata.
90-es évek, szuperskalárok: több futószalagnak azonos fokozatai közös regiszterekből veszi az adatot (elválasztó regiszter és multiplexer).

A futószalagok fizikai felépítése:

Univerzális futószalag: a hardver elemek számát minimalizálják, de lassú. 1 fizikai futószalaghoz több logikai futószalagot rendelnek hozzá. pl RISC I.
Master futószalag: a master futószalag minden utasítás feldolgozására alkalmas, a másik futószalag csak egyszerű utasításokra. Pl: Pentium I.
Dedikált futószalag: minden logikai futószalagot igyekeznek külön fizikai futószalagon megvalósítani. Pl: PowerPC 604. Egy lehetséges megvalósítást lásd: órai jegyzetben.

SzA22. Első generációs (keskeny) szuperskalár processzorok

(közvetlen kibocsátás, végrehajtási modelljük, kibocsátási szűk keresztmetszetük)

Első generációs szuperskalárok jellemzői:

- Közvetlen (nem pufferelt) utasítás-kibocsátás
- Statikus elágazás-becslés: a programkód alapján
- Gyorsítótárak (szűk keresztmetszet miatt)
  - Külön adat- és utasítás-gyorsítótár
  - Kétszintű gyorsítótár:
    - L1 gyorsítótár: a processzor lapkáján helyezkedik el
    - L2 gyorsítótár: különálló lapkán helyezkedik el

Közvetlen kibocsátás fogalma:

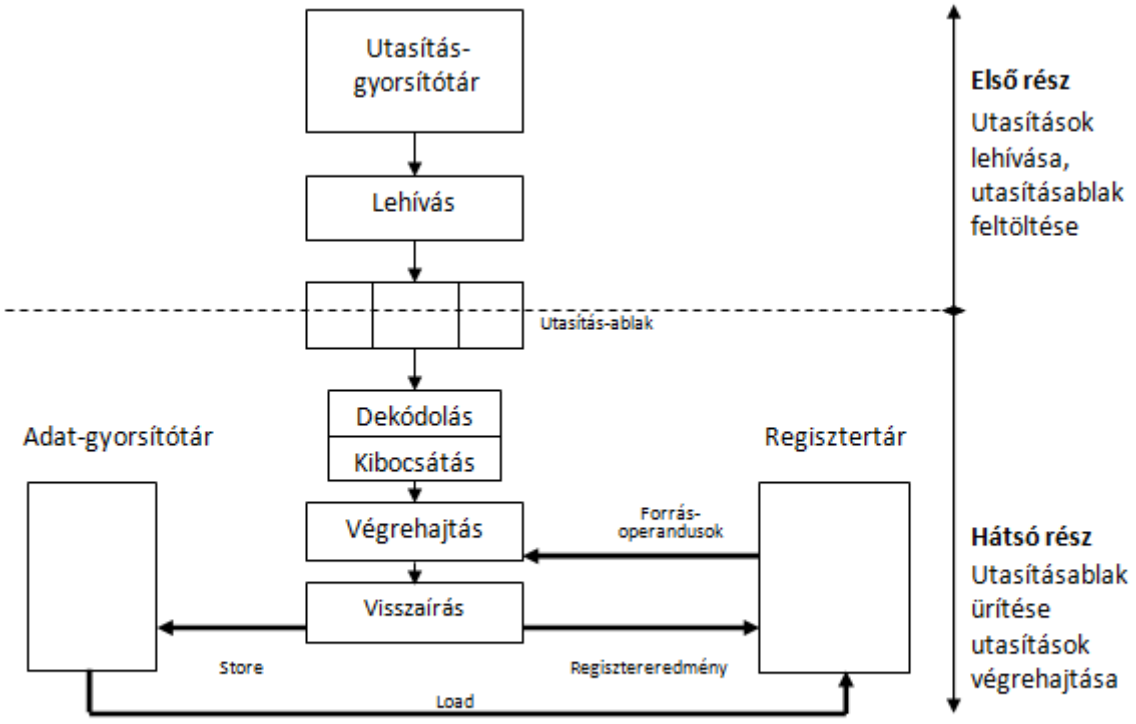
Puffer nélkül történik az utasítások kibocsátása az utasítás gyorsítótárból.

A kibocsátási szűk keresztmetszet:

Mivel a kibocsátás az első függő utasításig történik, ezért függőség esetén nem mindig tudunk annyit garantálni, amekkora az utasításablak mérete. Bár az utasításablak mérete RISC esetében 3, CISC esetében 2, de a feldolgozási ráta 1 utasítás / óraciklus alatt van a függőségek okozta blokkolások miatt.

Ennek feloldása a második generációs szuperskalárok esetén fog megtörténni pufferek, úgynevezett várakoztató állomások beiktatásával és regiszter-átnevezéssel.

Végrehajtási modell:

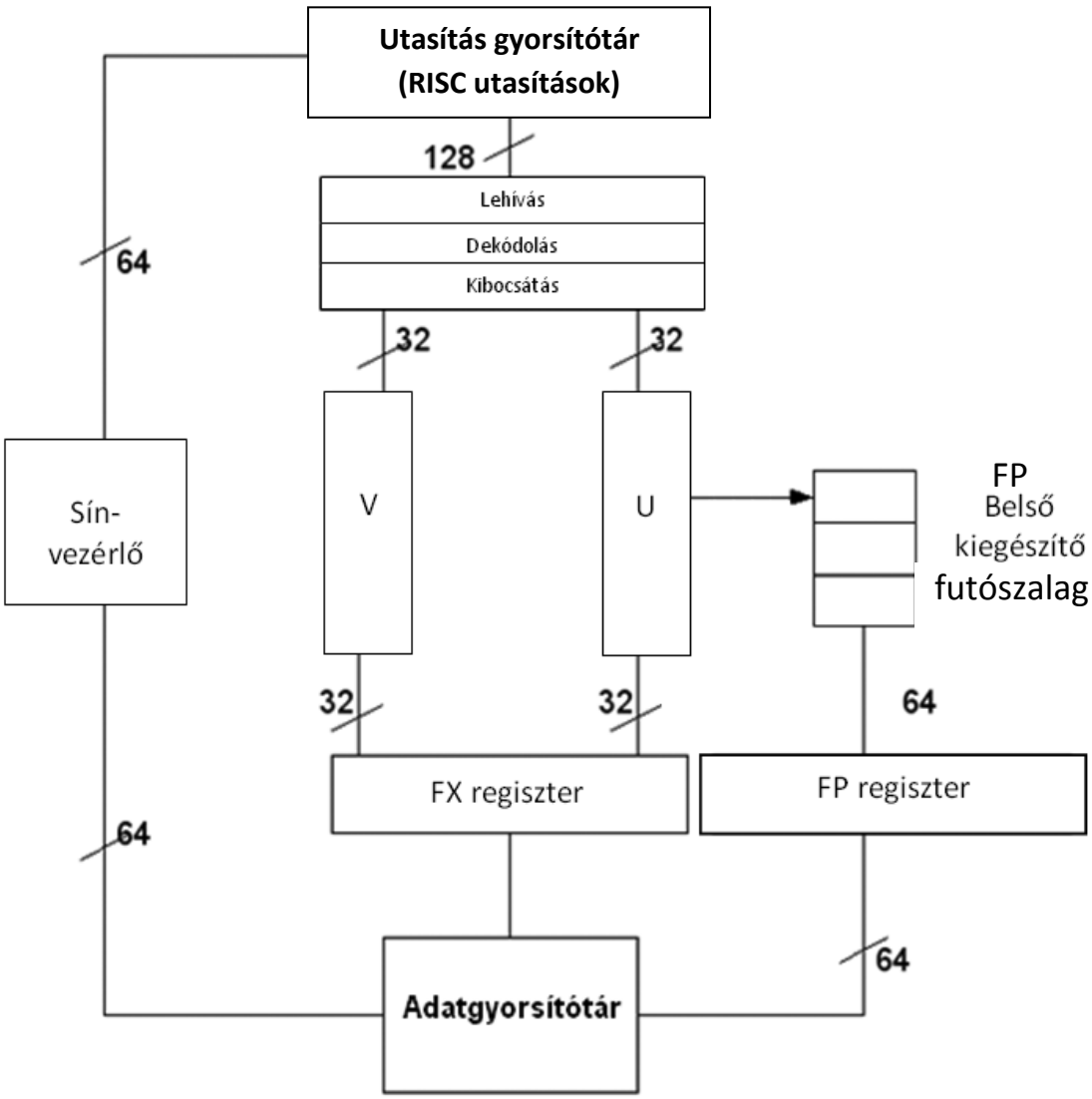


Az ábra alrendszerek sorozatát tartalmazza.
Az egész rendszer teljesítményét a legszűkebb keresztmetszet határozza meg.

SzA23. Első generációs (keskeny) szuperskalár processzorokra esettanulmány

(Pentium I.: megvalósítás és jellemzők)

Megvalósítás:



Jellemzői:

- A belső sínrendszer 64 bites
- Processzor:
  - Két futószalag:
    - U (master) futószalag: minden utasítás feldolgozására alkalmas (univerzális)
    - V futószalag: csak az Intel által egyszerűnek titulált utasítások feldolgozására alkalmas
      pl. FX, L/S, B
  - Mindkét futószalag 5 fokozatú
    - F, D, gyorsítótár elérés, E, W/B
  - Csak akkor működik párhuzamosan, ha mindkét futószalag egyszerű utasítást dolgoz fel
  - Az FP utasításokat az U futószalag előfeldolgozza, s magát a számítást egy háromfokozatú kiegészítő lebegőpontos futószalag végzi
- Gyorsítótár: 2 db, egyenként 8KB méretű van belőle, utasítás-, ill. adatgyorsítótár.

SzA24. Második generációs (széles) szuperskalár processzorok

(a kibocsátási szűk keresztmetszet kiküszöbölése: dinamikus utasítás-ütemezés és regiszter-átnevezés, végrehajtási modelljük, értékelésük)

Kibocsátási szűk keresztmetszet kiküszöbölése

1. Dinamikus utasítás-ütemezés
2. Regiszter-átnevezés

1. Dinamikus utasítás-ütemezés:

- Pufferelt utasítás-kibocsátás
- Sorrenden kívüli kiküldés

Elvi felépítés: lásd órai jegyzet. Várakoztató állomások beszúrása a cache és a VE közé.

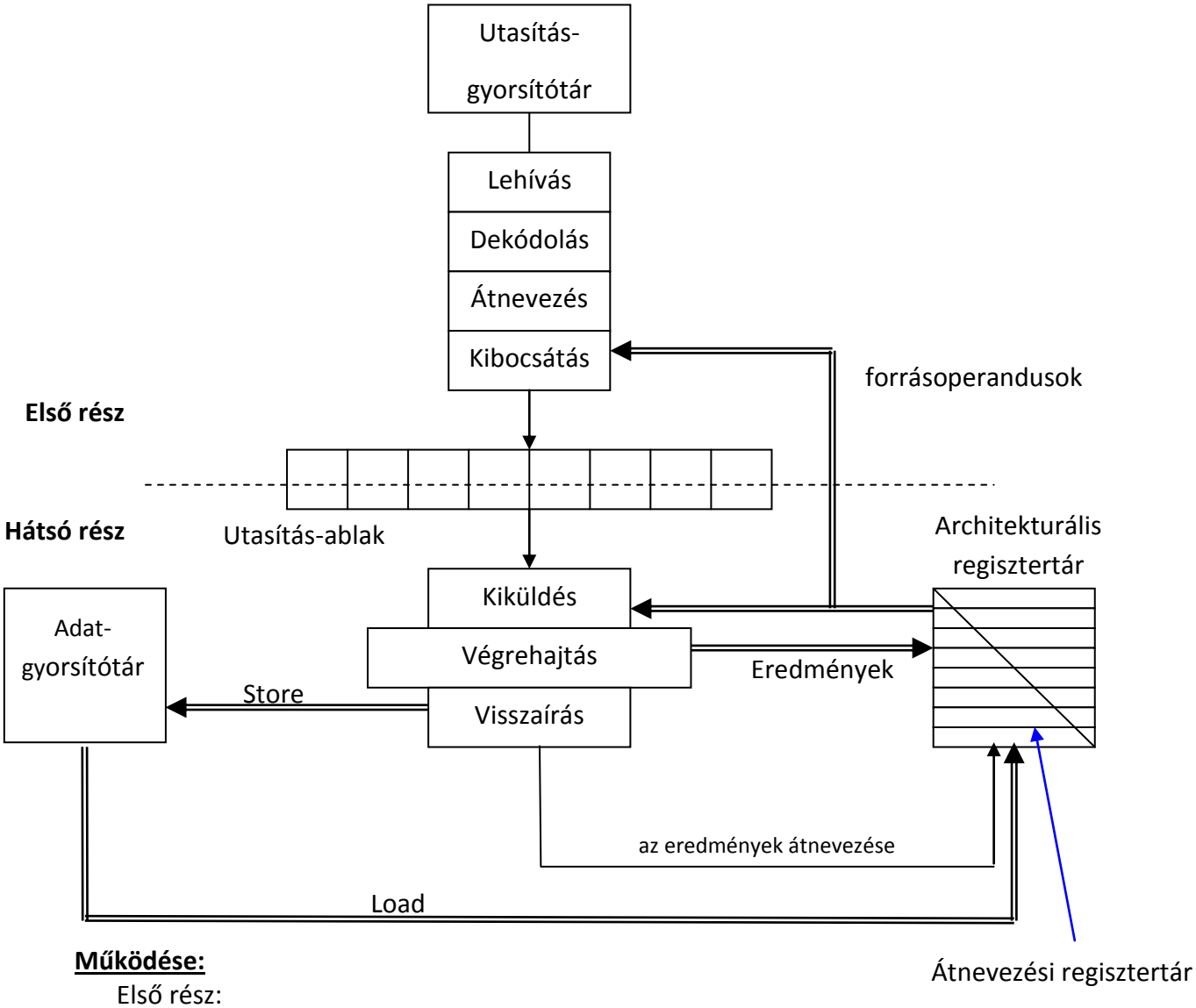
Működése:

- Első rész:
  Mivel a kibocsátás során nincs függőségvizsgálat, ezért a lehívás- dekódolás-kibocsátás a nominális rátával működhet (RISC: 4 db utasítás / óraciklus)
- Hátsó rész:
  Óraciklusonként akár több tucat utasítás közül is kiválaszthatja a CPU a függetleneket, és azokat kiküldi a megfelelő végrehajtó egységek felé.
  A kiküldés sorrenden kívül történik, a konzisztencia biztosításáról külön kell gondoskodnunk. Várakoztató állomás: adatfolyam végrehajtási modell.

2. Regiszter-átnevezés:

- 2. generációs szuperskalár processzoroknál jelent meg
- A CPU minden eredményregiszterhez hozzárendel egy átnevezési regisztert
- A CPU az adott regiszterre forrásregiszterként történő hivatkozásokat is kezeli
- Az utasítás-feldolgozás végén a processzor az átnevezési regiszterek tartalmát átmásolja az architekturális regiszterekbe, és az átnevezési regisztert felszabadítja
- Regiszter átnevezések eredményeképpen az ál-adatfüggőségek teljesen kiküszöbölődnek, ennek eredményeképpen a várakoztató állomásokban lévő független regiszterek száma nő

A második generációs szuperskalárok végrehajtási modellje:



Működése:

Első rész:

- Az utasításablak feltöltése
- Minden alrendszer a nominális rátával működik

Hátsó rész:

- A várakoztató állomásból a független utasítások kiküldésre kerülnek VE -khez
- A visszaírás során
  - az eredményeket az átnevezési regiszterbe írják
  - Az átnevezési regiszterből pedig az architekturális regiszterbe

Jellemző átbocsátó képesség:

- A kiküldési ráta tipikusan 5-8 utasítás/ciklus
- A végrehajtási alrendszer rátája még nagyobb, mivel minden esetben győznie kell a feldolgozást.

CISC feldolgozás

- A CISC processzoron belül kialakítunk egy RISC magot
- A CPU a beérkező CISC utasításokat lefordítja RISC utasításokká, majd ezeket dolgozza fel. Egy CISC utasítás ~1,2-1,5 RISC utasítássá alakítható át átlagosan
- A CISC kibocsátási ráta 3 utasítás/ciklus, a RISC magon belül ez ~ 4 RISC utasítás/ciklussá alakul

SzA25. Utasításlehívás I.

(a lokális elágazás előrejelzés fogalma, a lokális egyszintű és kétszintű dinamikus elágazás előrejelzés)

A lokális elágazás előrejelzés fogalma:

Az utasításlehívási alrendszer gyorsítható, amennyiben a feltétel kiértékelése előtt sikerül kellőképp megjósolni, hogy a feltételes elágazásnál merre fog folytatódni a vezérlés.
A „lokális" jelentése, hogy az elágazási utasítást önmagában vizsgáljuk, a környezetétől függetlenül, ellentétben a globálissal.

A statikus és dinamikus előrejelzés: az ugrási irány megbecslése

- **Lokális egyszintű fix**: Tipikusan mindig ugrik. Futószalagoknál használták.
- **Lokális egyszintű statikus**: a programkód alapján történik a becslés (compiler által). Jól felismerhető a ciklus, ezáltal jó a becslés. Első generációs szuperskalárok alkalmazzák.
- **Lokális egyszintű dinamikus**: Bimodal Branch Prediction, ugrástörténeten alapuló becslés. Megvalósítása: az órai jegyzetben. Minden elágazási utasításhoz létrehozunk egy-egy bejegyzést, amelyek 2 bites telített (nem-átforduló) számlálók. Az értékei:
  11 - erős ugrás
  10 - gyenge ugrás
  01 - gyenge folytatás
  00 - erős folytatás.
  Az első bithelyiérték alapján történik az ugrási irány becslése.
  Működés: Alapállapot: 11. Ha a feltétel kiértékelése után soros folytatás, csökkentjük.
  A bejegyzéseket tartalmazó tábla (BHT) kellően nagy, azaz minden elágazási utasításhoz hozzárendelhetünk egy-egy külön bejegyzést, a becslés igen pontos lesz (~93%). Amennyiben erre nincs lehetőség, azaz több elágazási utasításnak kell osztoznia 1 bejegyzésen, a becslés romlik.
  Példa: Pentium I
- **Lokális kétszintű dinamikus**: A BHT ebben az esetben egy 4 bites léptetőregiszter, amely az előző 4 kiértékelés eredményét tartalmazza a korábbiak alapján (0: soros folytatás, 1: ugrás). Minden BHT bejegyzéshez (1. szint) tartozik egy-egy 16 db telített számlálót tartalmazó tábla (2. szint), amely tartalmát a BHT tartalma címzi meg. A telített számláló működése a korábbiakkal azonos. A becslés a számlálók tartalma alapján történik, ezt és a léptetőregisztert karban kell tartani a feltétel kiértékelése után. Szabályos minták esetén pontosabb a becslés (~97%), ugyanakkor lassabb és több hardver szükséges hozzá. Példa: Pentium Pro, Pentium MMX.

SzA26. Utasításlehívás II.

(az ugrási cél lehető leggyorsabb elérésének probléma-felvetése, a kiszámítási /behívási, a BTAC és a BTIC módszer)

A probléma felvetése:

Az elágazási utasítások blokkolnak. Ennek kiküszöbölésére alkalmazzák az elágazásbecslést. Az ugrási cél mielőbbi elérése azért fontos, mert a vezérlés kb 75%-ban az ugrási cím irányában folytatódik tovább.

Kezelés:

A lehető legkevesebb ciklussal próbáljuk elérni az ugrási célt, sokféle megoldás létezik erre.

Az utasításgyorsítótár (I-cache) felépítése:

Address => I, I+1, I+2, I+3. 4-4 byte hosszú utasítások (16 byte 1 bejegyzés).
Igazított utasítások:
Az első (I) pozícióban az ugrási cél utasítás (BTI), a többiben az őt követő további utasítások.
Gyors i-cache:
Az ugrási cím (A) kiszámítása utáni óraciklusban kiolvasásra kerül az ugrási cél utasítás (I).
Lassú i-cache: 2-3 óraciklus alatt kerül csak kiolvasásra.

Kiszámítási/behívási módszer:

Lényege: Az ugrási cím elérésének a természetes, hagyományos módszere.

Megvalósítása: az órai jegyzetben. IFAR lehetséges inputjai: IIFA, inkrementálás, BTA.

Működése:

= A CPU meghatározza induló program kezdőcímét (IIFA), és betölti az IFAR -ba
= IFAR inkrementálódik, az adatmanipuláló utasítások feldolgozásra kerülnek
= Elágazási utasítás esetén futószalag fokozattal vagy címszámítóval meghatározásra kerül az ugrási cím
= A kapott címmel felülírjuk az IFAR tartalmát
= A vezérlés átadásra kerül az IFAR –ban lévő ugrási címre

Értékelés

- előnye:
  = Egyszerű
  = Nem igényel extra hardvert
- hátrány:
  = A címszámítás és az ugrás szekvenciális
  = Például ciklus estén, ha az n-szer fut akkor n-szer kell kiszámítani ugyan azt a címet felesleges munka, időveszteség, mert ezt előre is meg tudnánk mondani

BTAC módszer:

Lényege:

Az ugrási címet csak egyszer számoljuk ki, és azt tároljuk a BTAC –ben, a következőkben az ott eltárolt címet használjuk fel

Megvalósítása: órai jegyzetben.

Plusz hardver: BTA Cache. Azt az ugrási célcímet, amit már egyszer kiszámoltunk, ebben a gyorsítótárban helyezzük el. Ennek a tartalmával írhatjuk felül az IFAR -t.

Működése:

= Az ugrási utasítások címe és a hozzájuk tartozó ugrási célcím beírásra kerül a BTAC -be

= Amennyiben az IFAR –ban olyan ugrási utasítás címe van, amelyhez tartozik bejegyzés a BTAC –ben, akkor

- az utasítás-gyorsítótár elérésével egyidejűleg a BTAC -ből kiolvasásra kerül az ugrási cím.
- a következő óraciklusban pedig már az ugrási címre adódik át a vezérlés

Értékelés:

Előny:

- Nem képez szekvenciát az ugrási cím meghatározása és az ugrás.
- Csak egyszer kell meghatározni az ugrási címet

Hátrány

- Extra hardvert igényel: BTAC

Példa: Pentium I

BTIC módszer:

Lényege:

Ha lassú az utasítás gyorsítótár, 2-3 óraciklus késéssel tudja csak szállítani az ugrási cél utasítást. Ezt a késést hasznosítjuk ezzel a módszerrel.

Megvalósítása: órai jegyzetben. BTIC: kicsi és gyors cache. BA, BTI és (BTA+ vagy BTI+).

Működése:

= BTIC-be behívásra kerülnek a legutóbbi néhány

= Elágazási utasítás címe, vagy

= Az elágazási utasítások közül azok címe, melyek az ugrási irányban folytatódtak

= A BTIC további részében az ugrási utasítás (BTI) ÉS az azt követő néhány utasítás címe (BTA+) VAGY az azt követő néhány utasítás (BTI+)

= Amennyiben az IFAR –ban olyan elágazási utasítás címe kerül, melyhez tartozik bejegyzés a BTIC –ben, akkor **az ugrási címen lévő utasítás a BTIC –ből kerül lehívásra**, sokkal gyorsabban, mint az utasítás-gyorsítótárból megtörténhetne.

Példa: AMD AM29000

SzA27. Elődekódolás

(szükségessége, megvalósítása, feladatai, azok leképezése az előkódoló által)

Az előkódolás szükségessége: A dekódolási alrendszer gyorsítása.

- A szuperskalár feldolgozás esetén többletfeladatok hárulnak a processzorokra:
  - = CISC esetén RISC –szerű utasításra fordítás
  - = Átnevezés
  - = Szabad helyek meglétének ellenőrzése
    - az átnevezési pufferben (ROB)
    - a várakoztató állomáson
  - = Megfelelő várakoztató állomás kiválasztása
  - = Egy várakoztató állomásba egy óraciklusban több utasítás kibocsátása
- Mennyiségileg is többet kell elvégezni
  - = CISC esetében 3 utasítás / óraciklus
  - = RISC esetében 4 utasítás / óraciklus

A nagy mennyiségű kiküldhető utasítás kezelésére a megoldás az előkódolás.

A megvalósítás: L2 cache -> Előkódoló -> L1 cache. 128 és 148 bit/óraciklus sávszélesség.

Feladatai:

- CISC esetben az utasítás kezdetének és a végének felismerése
  (a CISC változó hosszúságú utasítás, a RISC hossza fix)
- Az utasítás típusának meghatározása (Load/Store, aritmetikai, stb)
- Ugrási utasítások felismerése
- A műveleti kód behatárolása
- Akár az elágazási címet is kiszámíthatja

A feladatok leképzése az előkódoló által:

Példa:

az AMD K5 CISC processzor minden byte -hoz hosszá tesz 3 bitet => 60% -kal több

RISC: tipikusan 4-7 bitet tesznek hozzá => 10-12% -os növekedés

SzA28. Kibocsátáshoz kötött operandus-lehívás

(megvalósítása, a kibocsátáshoz és a kiküldéshez kapcsolódó feladatok, frissítés, értékelés)

op_s – source (forrás) operandus

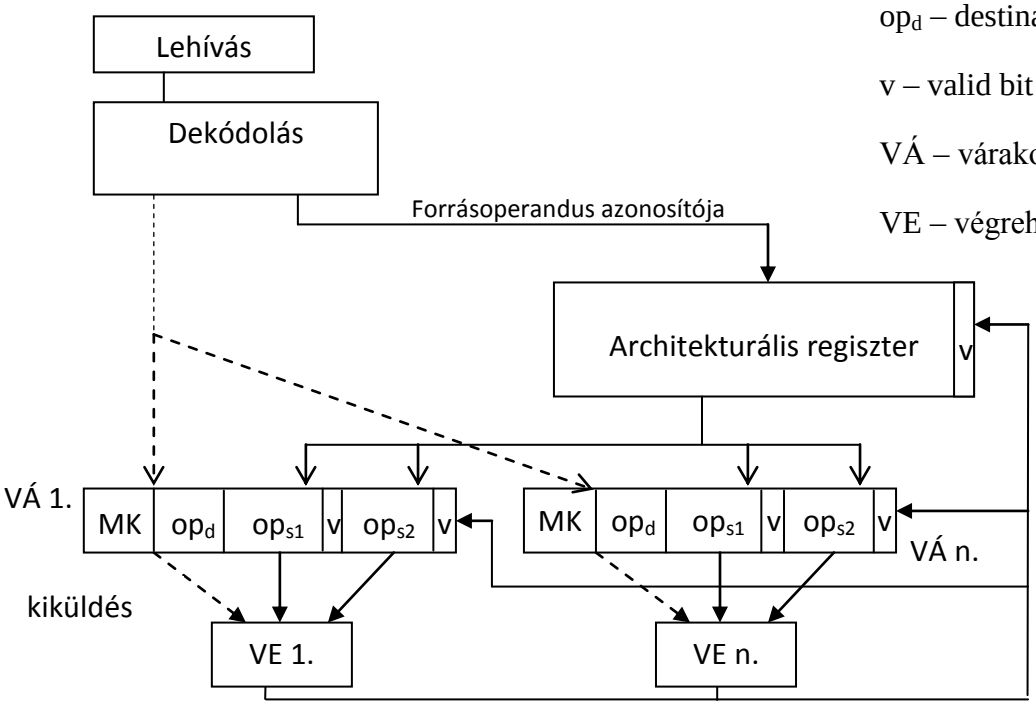
op_d – destination operandus

v – valid bit

VÁ – várakoztató állomás

VE – végrehajtó egység

Megvalósítása



Infrastruktúra az érvényesség-jelzéshez: az adatregiszterek után 1 bites kiegészítés, az állapotbit jelzi az érvényességet. 1, ha érvényes, 0, ha nem. Ez megtalálható mind az adatregiszterben, mind pedig a várakoztató állomás forásoperandusainál.

Feladat: A forrás-operandusok érvényességének ellenőrzése.

A kibocsátás során a várakoztató állomásba kerül a műveleti kód és az eredmény regiszterazonosítója. A forrás-operandusok regiszter-azonosítói eljutnak az architektúrális regiszterbe. A lehetséges esetek:

- amennyiben az adott regiszterben lévő adat valid-bitje 1, akkor maga az érték bemásolásra kerül a várakoztató állomásba, és a valid-bitet 1 -re állítja
- amennyiben az adott regiszterben lévő adat valid-bitje 0, akkor az adott regiszter-azonosító kerül be a várakoztató állomásba, és a valid-bitet 0 -ra állítja.

Az eredmények frissítése két helyen is szükséges:

- Az architektúrális regiszterben: az eredményt beírjuk, és a valid-bitet 1-re állítjuk
- A várakoztató állomásokban: az eredménysínre fel kell helyezni mind az eredményt, mind pedig a regiszter-azonosítót; asszociatív keresést kell végrehajtani a várakoztató állomásban az adott regiszter-azonosítóval, s ahol megtaláltuk, ott a regiszter-azonosítót felülírjuk az eredménnyel, és a valid-bitet 1-re állítjuk. Ezt a keresést el kell végezni minden olyan várakoztató állomásban, ahol ez az eredmény előfordulhat (például fixpontos eredmény esetén az összes fixpontos várakoztató állomásban)

Értékelés

Hátrányai:

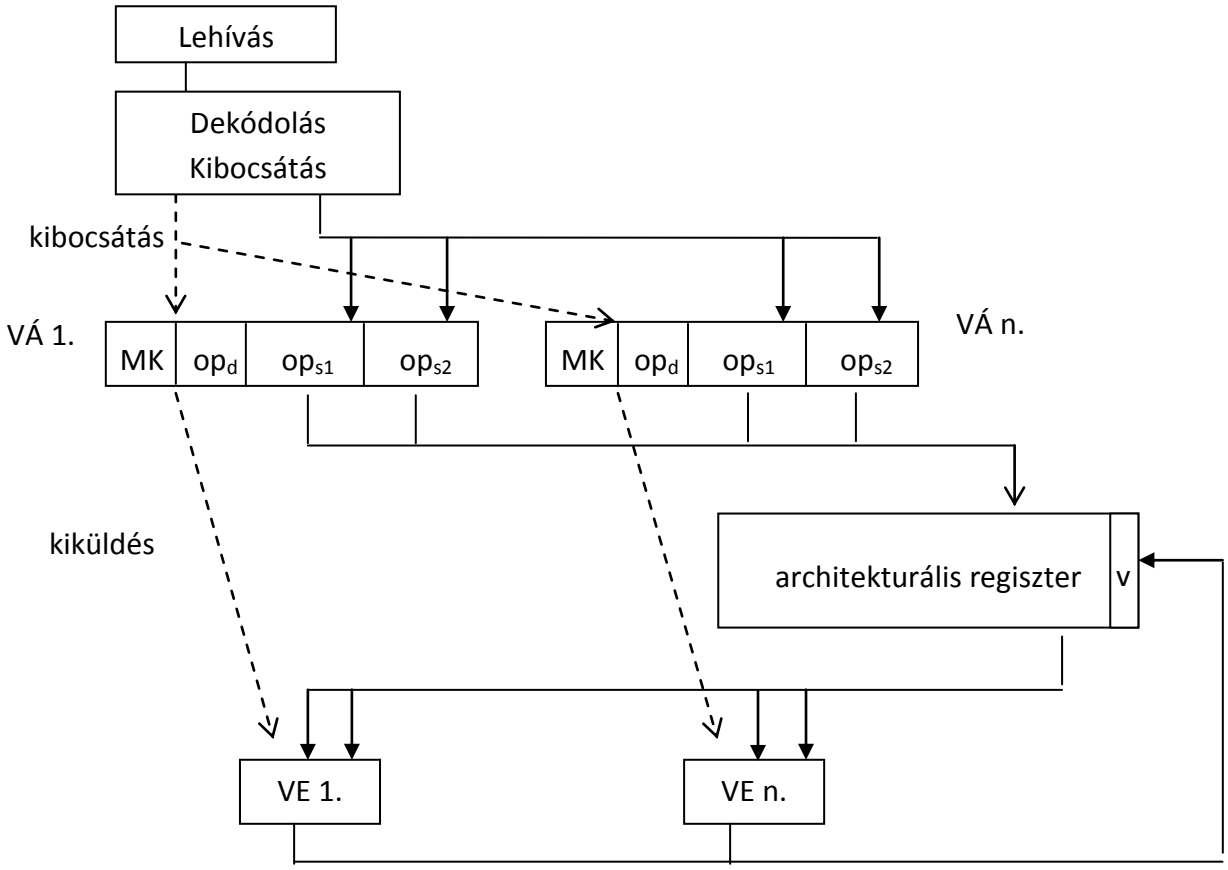
- A szűk keresztmetszetet jelentő dekódolási/kibocsátási fázist terheli, lassít
- A várakoztató állomásokban maguk az értékek vannak, ezért azok hosszúak
- Két helyen kell az eredményekkel karbantartást végezni

Előnye: igen gyors a valid bitek kiértékelése

SzA29. Kiküldéshez kötött operandus-lehívás

(megvalósítása, a kibocsátáshoz és a kiküldéshez kapcsolódó feladatok, frissítés, értékelés)

Megvalósítása:



Feladat:

A forrás-operandusok érvényességének ellenőrzése

= Kibocsátáskor a megfelelő várakoztató állomásba juttatjuk a műveleti kódot és a cél valamint a forrásoperandusok azonosítóit

= A CPU ellenőrzi a forrás-operandusokhoz tartozó regiszterek valid-bitjét az architekturális regiszterben

Eredmények frissítése:

= Amennyiben az utasítás mindkét forrás-operandusának valid-bitje 1, akkor

- Az adott utasítás műveleti kódja és eredmény azonosítója kiküldésre kerül a végrehajtóegységbe, továbbá
- A forrás-operandusok azonosítói kiküldésre kerülnek az architekturális regiszterbe, ahonnan az értékek kis késleltetéssel elindulnak a végrehajtó egységbe

= Az eredményekkel kizárólag az architektúrális regisztert kell karbantartani

Értékelés

Előnyei

- Felszabadítja a szűk keresztmetszetet jelentő dekódolási/kibocsátási fázist => gyorsít
- A várakoztató állomások csak regiszter azonosítót tartalmaznak => rövid
- Az eredményekkel csak az architektúrális regisztert kell frissíteni

Hátránya: A forrás-operandusok érvényességének ellenőrzése bonyolultabb

SzA30. Átrendezési puffer (ROB)

(megvalósítása, működése, az átnevezés, a spekulatív végrehajtás és a pontos megszakítás-kezelés támogatása)

A ROB bejegyzések tartalma: utasítás azonosítói, operandusok, állapotbitek

Megvalósítása: körpuffer segítségével.
A kezdeti mutató az első üres helyre mutat, a végmutató a kiírandó bejegyzésre.

Működése:

- A kibocsátás sorrendben történik, így a kibocsátás sorrendjében folyamatosan töltjük fel a körpuffert a kezdeti mutatónak megfelelően
- A processzor követi a kiküldést: végrehajtás alatt van-e, vagy már befejeződött
- A kiírási feltétel (W/B, az eredményt az architekturális regiszterbe vagy memóriába)
    - Az adott utasítás végrehajtása befejeződött
    - Minden, ezt megelőző utasítás kiírásra került
- Megjegyzés: a RISC –szerű utasítások eredményét nem lehet kiírni, csak az eredeti CISC utasítás eredményét
- A kiírt bejegyzés helyét a CPU felszabadítja

Átnevezés:
A második generációs szuperskalár feldolgozásnál az átnevezéseket a ROB felügyeli. A ROB hozzárendel az utasítások célregiszteréhez egy-egy regisztert az átnevezési regisztertárból, ezzel minimalizálja az adatfüggőségek hatását.

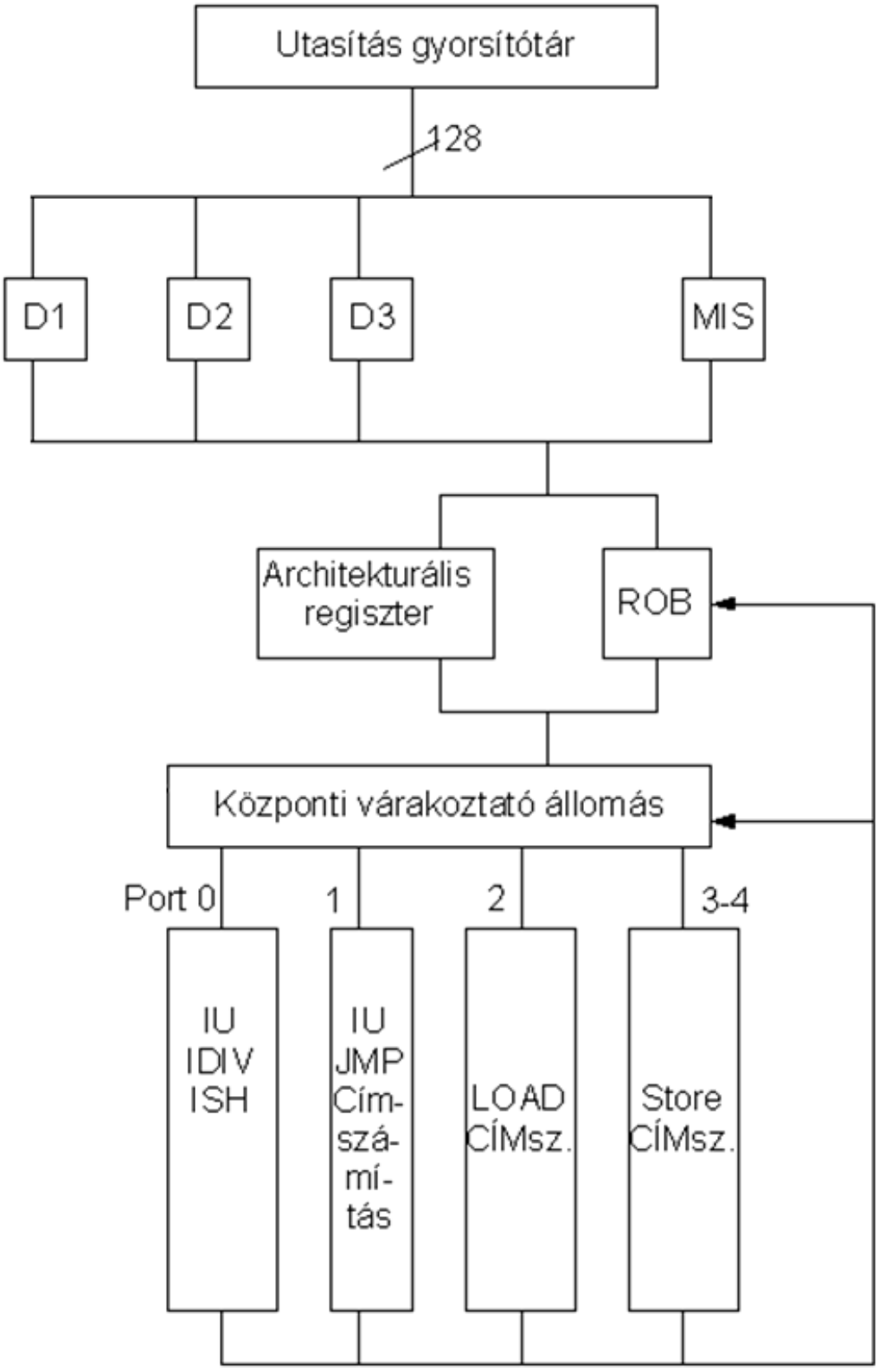
Spekulatív elágazás-kezelés
- Infrastruktúra:
    - a ROB minden bejegyzéséhez hozzárendelünk egy állapotbitet, mely jelzi, hogy az adott bejegyzés spekulatív-e, vagy sem
- További kiírási feltétel:
    - Spekulatív utasítást nem szabad kiírni

A pontos megszakítás-kezelés támogatása:
Megszakítást csak akkor lehet elfogadnia a processzornak, amikor az adott bejegyzés a ROB -ból kiírásra kerül: kivétel-konzisztencia.

SzA31. Második generációs (széles) szuperskalár processzorokra esettanulmány

(Pentium Pro: megvalósítás és jellemzők)

Pentium Pro megvalósítása:



Dekóderek: 2 egyszerű, 1 összetett és 1 MIS (Microcode Instruction Sequencer).
Egyszerű: 1 db, összetett: 1-4 db, MIS: 4 feletti RISC utasítás dekódolása.

Jellemzői:

- 1995: eredetileg 133Mhz, 14 fokozatú fixpontos futószalag
- Központi várakoztató állomás 20 db RISC –szerű utasítást képes befogadni, fixpontost és lebegőpontost egyaránt.
- Az átnevezéseket a ROB kezeli, a szekvenciális konzisztencia szintén ROB által biztosított
- operandus lehívás, átnevezés, kibocsátás:
  - kibocsátáshoz kötött operandus lehívás
  - minden célregisztert átnevez, erre a ROB –ot használja fel
  - várakoztató állomásban RISC utasítás-szerkezet: r1 <= r2@r3

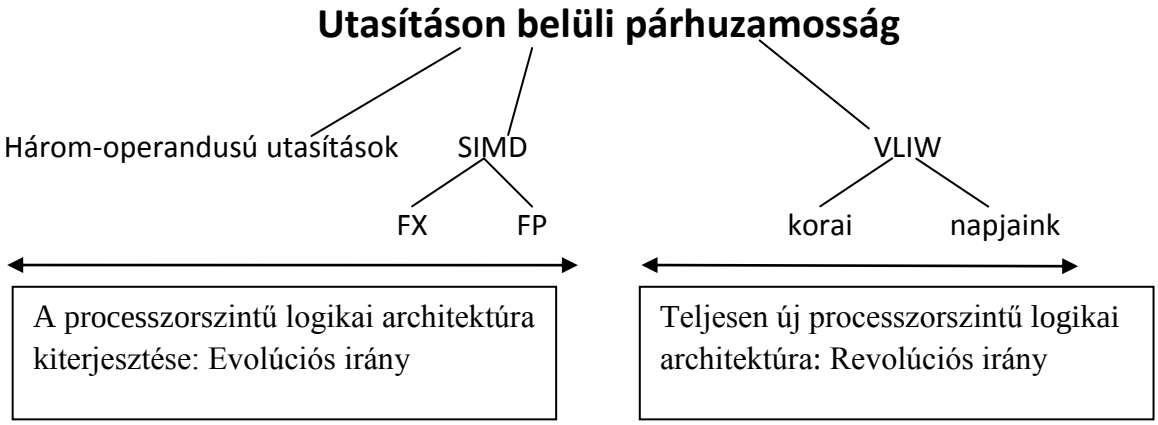
MK	R_1	R_2	A_2	R_3	A_3

A: állapotjelzőbit

  - Amennyiben a forrásoperandus értéke még nem áll rendelkezésre, akkor a regiszterazonosítót írja be a várakoztató állomásba, és az állapotbitet 0 –ra állítja
  - Ha a forrásoperandus a ROB –ban is megtalálható, akkor az architekturális regiszterrel szemben a ROB –beli értéket helyezi előnybe
- kiküldés: az idősebb utasításokat részesíti előnyben
- végrehajtás
  - Azért van 6 db VE egy porton, hogy helyet takarítson meg a lapkán
  - IU – integer unit, IDIV – egész osztás, ISH – léptetés
- Visszaírás: 2 helyen kell visszaírni:
  - Várakoztató állomásba
    - Kiveszi azt a regiszter azonosítót, amelynek az eredménye elkészült
    - Ezt felülírjuk az eredménnyel
    - Állapotbitet 1 –re állítjuk
  - ROB –ba
    - A későbbi utasítások számára, melyek ugyanerre a regiszterre hivatkoznak
    - A ROB –ba való kiírás szabályai:
      - = Az összes megelőző utasítás eredménye már kiírásra került
      - = RISC utasítások eredménye csak akkor írhatók ki, ha az adott CISC –hez tartozó összes RISC elkészült
      - = Amikor a CISC eredményét kiírta a ROB –ból az architekturális regiszterbe, akkor felszabadítja az adott utasításnak a ROB –ban elfoglalt helyét

SzA32. Harmadik generációs szuperskalár processzorok: az utasításon belüli párhuzamos végrehajtás

(három-operandusú utasítások, SIMD-utasítások, VLIW-architektúrák)



Három-operandusú utasítások

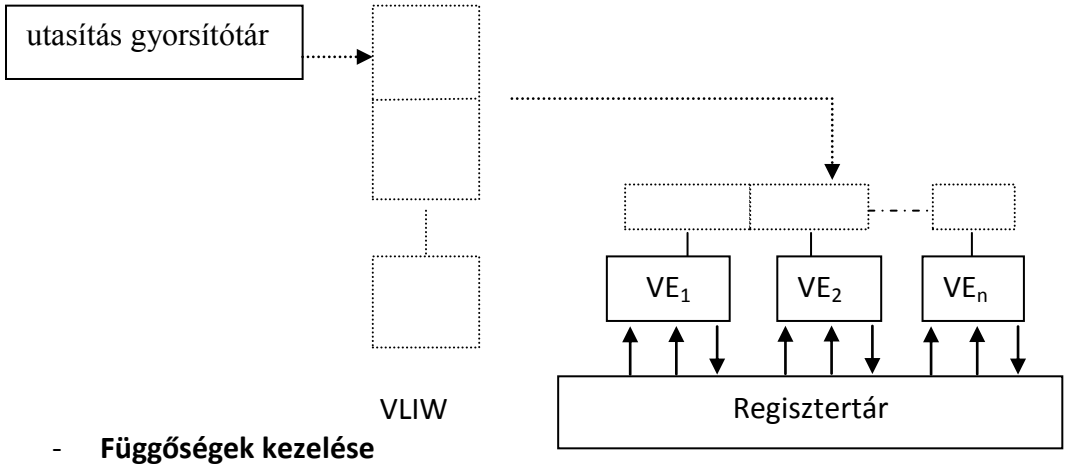
- **Fogalma:** egy utasításon belül két különböző művelet
- Például: multiply-add: x= a*b+c; load-op: utasítás betöltés és végrehajtás; shift-add
- Numerikus feldolgozást gyorsítják, főleg RISC, ritkábban CISC környezetben alkalmazzák
- 70-es években jelentek meg

SIMD – Simple Instruction Multiple Data:

- **Fogalma:** egyetlen utasításon belül több operanduson ugyanazt a műveletet hajtjuk végre
- A SIMD megjelenése jelenti a 3. generációs szuperskalárok megjelenését!
- Alapvető jellemzői:
    - A processzorszintű logikai architektúra kiterjesztése
    - Az L2 gyorsítótár felkerült a processzor lapkájára (a több operandus miatt)
    - A rendszerarchitektúra szűk keresztmetszetének bővítésére megjelent az AGP sín, kifejezetten a videokártya számára
- A SIMD utasítások kizárólag a multimédia műveleteket gyorsítják, az általános célú feldolgozásra nincsenek hatással.
- fajtái:
    - fixpontos multimédia:
        - hang és a pixeles képfeldolgozás
        - 2-8 operandus egy utasításban
    - Lebegőpontos multimédia
        - vektoros képfeldolgozásban
        - 2-4 operandus egy utasításban

VLIW – Very Long Instruction Word

- **Működési elve:**



- **Függőségek kezelése**
  - Statikusan, compiler által
  - Hátránya:
    - A compiler erősen technológiafüggő, azaz ismernie kell a fizikai architektúrát pl.:
      - a végrehajtó egységek számát
      - a végrehajtó egységek késleltetését
      - a gyorsítótár késleltetését
  - Előnye:
    - Egyanolyan fokú párhuzamosítás mellett a VLIW sokkal egyszerűbben megvalósítható
    - Ennek köszönhetően jelenhetett meg csaknem 10 évvel korábban, mint a szuperskalár processzorok
- **Utasítások:**
  teljesen új logikai architektúra: teljesen új, hosszú utasításszavú utasítások
- **Fajtái, fejlődése**
  - Széles vagy korai VLIW-ek
    - 80 -as években jelentek meg
    - Pl.: TRACE processzor:
      - 256-1024 bit hosszú utasításszavak
      - 7-28 utasítást tartalmaztak
      - Igen hamar leállították a forgalmazását, ennek oka: a compiler nagyfokú technológiai függősége
  - Keskeny VLIW-ek
    - 90 –es évek 2. felében jelentek meg, közben sokat fejlődtek a compilerek
    - Fajtái:
      - Digitális jelfeldolgozás, multimédia
        Az ok ugyanaz, mint SIMD esetén: nincs benne feltételes elágazás
      - Általános célú feldolgozás
        - Szerverpiac: Intel Itanium (2004 után: többmagos CPUk)
        - Hordozható gépek: Transmeta Crusoe
          (Változó tápfeszültség-szint; a VLIW alacsonyabb fogyasztású)