

Számítógép architektúrák I.

2008

Órai jegyzet

mrjrm, Pogácsa, Pheenix és Quetzalcoatl
jegyzeteinek frissített változata

A jegyzet készítői az esetleges hibákért semmilyen felelősséget nem vállalnak, annak birtoklása az előadáson való részvételt nem pótolja!

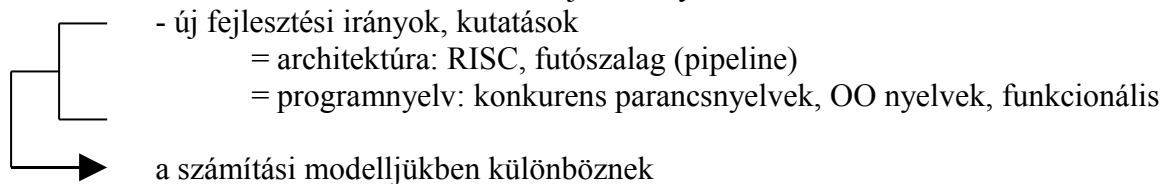
Tartalomjegyzék

1	SZÁMÍTÁSI MODELLEK	1
1.1	A számítási model	1
1.2	Számítási modell, programnyelvek, architektúra	1
1.3	Fejlesztési kronológia	1
1.4	A számítási modellek osztályozása	2
2	ADATALAPÚ SZÁMÍTÁSI MODELLEK	2
2.1	Neumann-féle számítási modell	2
2.1.1	Min hajtjuk végre a számítást?	2
2.1.2	Hogyan képezzük le a számítási feladatokat?	2
2.1.3	Mi vezérli a végrehajtást?	3
2.2	Adatfolyam számítási modell	3
2.2.1	Min hajtjuk végre a számítást?	3
2.2.2	Hogyan képezzük le a számítási feladatot?	3
2.2.3	Mi vezérli a végrehajtást?	4
2.3	Összehasonlítás	4
3	ARCHITEKTÚRA	5
3.1	Fogalmak	5
3.2	Logikai-fizikai architektúra	5
3.3	Egy korszerű számítógép szintjei	6
3.4	<i>Processzor szintű logikai architektúra</i>	6
3.4.1	Adattér	6
3.4.1.1	Memóriatér	6
3.4.1.2	Regisztertér	7
3.4.1.2.1	Egyszerű	7
3.4.1.2.2	Adattípusonként különböző regisztertér	9
3.4.1.2.3	Többszörös regiszterkészlet	10
3.4.1.2.3.1	Háttér információ	10
3.4.1.2.3.2	Több egymástól független regiszterkészlet	10
3.4.1.2.3.3	Átfedő regiszterkészlet	10
3.4.1.2.3.4	Stack-cache	11
3.4.2	Adatmanipulációs fa	12
3.4.2.1	Részei	12
3.4.2.2	Adattípusok	13
3.4.2.3	Műveletek	14
3.4.2.4	Operandus-típusok: bevezetés	15
3.4.2.4.1	Az utasítás-feldolgozás menete	15
3.4.2.4.1.1	Az utasítás-feldolgozás nagyvonalú folyamatábrája	15
3.4.2.4.1.2	I. Utasítás-lehívás	16
3.4.2.4.1.3	II. Utasítás-végrehajtás	16
3.4.2.4.2	Az utasítások fajtái (utasítás-típusok, Címzési módok)	17
3.4.2.4.2.1	4 címes utasítás	17
3.4.2.4.2.2	3 címes utasítás	17
3.4.2.4.2.3	2 címes utasítás	17
3.4.2.4.2.4	1 címes utasítás	18
3.4.2.4.2.5	0 címes utasítás	18
3.4.2.4.2.6	Napjaink trendje	18
3.4.2.5	Operandusok	18
3.4.2.5.1	Operandus-típusok	18
3.4.2.5.2	Architektúrák	18
3.4.2.5.2.1	Akkumulátor	19
3.4.2.5.2.2	Memória	19
3.4.2.5.2.3	Regiszter	19
3.4.2.5.2.4	Verem	19
3.4.2.6	Gépi kód	20
3.4.3	Állapottér	20
3.4.4	Állapotműveletek	20
3.5	<i>A processzorszintű fizikai architektúra</i>	21
3.5.1	Szinkron-aszinkron	21
3.5.2	Műveletvégző egység	21
3.5.2.1	Részei	21
3.5.2.2	Regiszterek	21
3.5.2.3	Adat utak	22
3.5.2.4	Kapcsolópontok	22
3.5.2.5	Szűk értelemben vett ALU	23
3.5.2.5.1	Fixpontos műveletek	23
3.5.2.5.1.1	Fixpontos összeadás	23
3.5.2.5.1.1.1	Egybites fél-összeadó	23
3.5.2.5.1.1.2	Egybites teljes összeadó	24
3.5.2.5.1.1.3	N - bites soros összeadó	25
3.5.2.5.1.1.4	Átvitel előrejelzés (Carry-Look-Ahead = CLA)	26
3.5.2.5.1.2	Fixpontos kivonás	28
3.5.2.5.1.2.1	Kettes komplement	29
3.5.2.5.1.3	Szorzás/Osztás	29
3.5.2.5.1.3.1	Architekturális megvalósítása a szorzás/osztásnak	30
3.5.2.5.1.3.2	Bináris szorzás sajátosságai	30
3.5.2.5.1.3.3	Gyorsítás	31
3.5.2.5.1.3.4	Booth - féle algoritmus	31

3.5.2.5.1.4	Fixpontos osztás	32
3.5.2.5.2	Lebegőpontos műveletek	33
3.5.2.5.2.1	Története	33
3.5.2.5.2.2	Jellemzői	33
3.5.2.5.3	IEEE 754-es lebegőpontos szabvány	36
3.5.2.5.3.1	Szabványos - kiterjesztett összehasonlítása	36
3.5.2.5.3.2	Műveletek	37
3.5.2.5.3.3	Kerekítések	37
3.5.2.5.3.4	Kivételkezelés	37
3.5.2.5.4	Esettanulmányok	38
3.5.2.5.5	Műveletek lebegőpontos számokkal	39
3.5.2.5.6	BCD	40
3.5.2.5.6.1	Összeadás BCD számokkal	41
3.5.2.5.7	Fixpontos, lebegőpontos, BCD összehasonlítása	43
3.5.2.5.8	Az ALU egyéb műveletei	43
3.5.3	Vezérlőrész	43
3.5.3.1	Huzalozott vezérlés	44
3.5.3.2	Mikroprogramozott vezérlés	46
3.5.3.2.1	Célja	46
3.5.3.2.2	A Wilkes-féle modell	46
3.5.3.2.3	Egy korszerű mikroutasítás felépítése	47
3.5.3.2.3.1	Egy korszerű mikro-vezérlő megvalósítása	47
3.5.3.2.3.2	A mikro-utasítás hosszát meghatározó tényezők	48
3.5.3.2.3.2.1	Cím mező hossza	48
3.5.3.2.3.2.2	Vezérlőrész hossza	49
3.5.3.2.3.2.2.1	Horizontális mikroutasítás	49
3.5.3.2.3.2.2.2	Vertikális mikroutasítás	49
3.5.3.2.3.2.3	Napjaink gyakorlata	49
3.5.3.2.3.3	A mikroprogramok időzítése	50
3.5.3.2.3.3.1	A mikroutasítások fajtái	50
3.5.3.2.3.3.2	A mikroprogramozott vezérlés gyorsítása	50
3.5.3.2.3.4	Mikroprogramozás	51
3.5.4	Sínrendszer (Buszrendszer)	51
3.5.4.1	Bevezetés	51
3.5.4.2	Kommunikáció fajtái	52
3.5.4.3	A külső sínrendszer fogalma	53
3.5.4.4	Jellemzői	53
3.5.4.5	Adatsín fajtái	53
3.5.4.5.1	Adatátvitel iránya szerint	53
3.5.4.5.2	Átvitel jellege szerint	53
3.5.4.5.2.1	Dedikált	53
3.5.4.5.2.2	Megosztott (shared) sín	54
3.5.4.5.3	Átvitt tartalom alapján	54
3.5.4.5.3.1	Címsín	54
3.5.4.5.3.2	Adatsín	55
3.5.4.5.3.3	Vezérlő vezetékek	55
3.5.4.5.4	Összekapcsolt területek alapján	55
3.5.4.5.4.1	Rendszersín (processzor-sín)	56
3.5.4.5.4.2	Bővítő-sín	56
3.5.4.6	A Nyitott sínrendszer - az IBM PC példáján	56
3.5.4.7	A sínrendszer működése	57
3.5.4.8	A sínfoglalás	57
3.5.4.8.1	Soros sínfoglalás	57
3.5.4.8.2	Párhuzamos sínfoglaltság	58
3.5.4.9	Az adatátvitel (bus timing)	59
3.5.4.9.1	Szinkron adatátvitel	59
3.5.4.9.2	Aszinkron adatátvitel	60
3.5.5	Az I/O rendszer	62
3.5.5.1	Fogalma	62
3.5.5.2	Határai	62
3.5.5.3	Tervezési tere	63
3.5.5.4	Programozott I/O	63
3.5.5.4.1	Fajtái címtér szerint	63
3.5.5.4.1.1	Különálló I/O címtér	63
3.5.5.4.1.2	A Memóriára leképzett I/O	65
3.5.5.4.2	A programozott I/O működése	66
3.5.5.4.2.1	Feltétlen átvitel	66
3.5.5.4.2.2	Feltételes átvitel	66
3.5.5.5	DMA - Direct Memory Access	66
3.5.5.5.1	Fogalma	66
3.5.5.5.2	Megvalósítása	67
3.5.5.5.3	Csatorna	68
3.6	Egy hipotetikus számítógép tervezése	70

Számítási modellek

70-es évek: - a Neumann-féle architektúra teljesítménye határaihoz ért



A számítási modell

- fogalma:

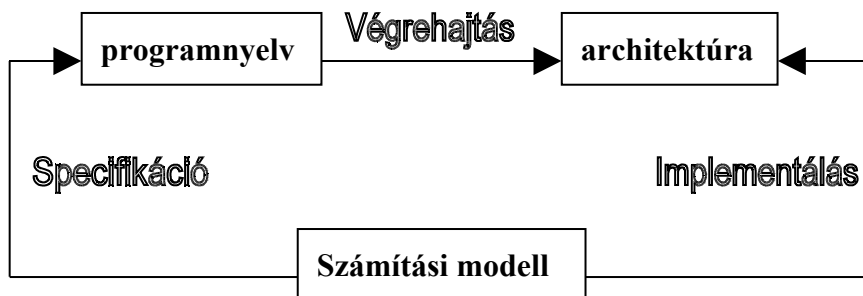
A számításra vonatkozó alapelvek absztrakciója.

(Korábban a jelentése: soros vagy párhuzamos végrehajtás?)

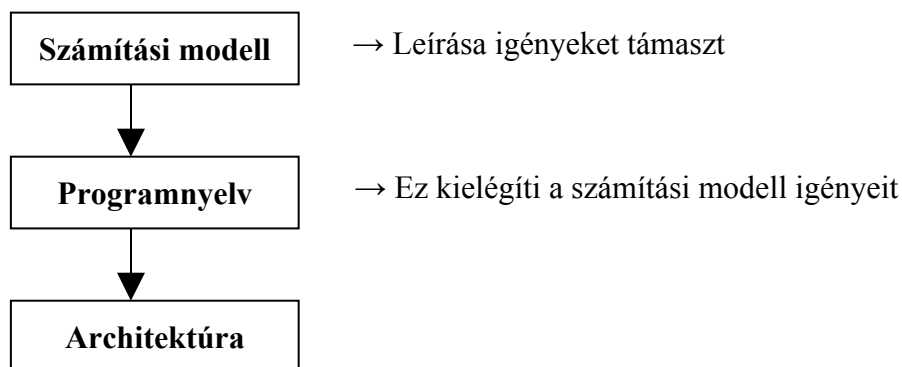
- jellemzői:

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



Neumann -féle számítási modell igénye:

- Programnyelv iránt: változó deklaráció, adatmanipuláló utasítások, vezérlésátadási utasítás

- Architektúrával szemben:

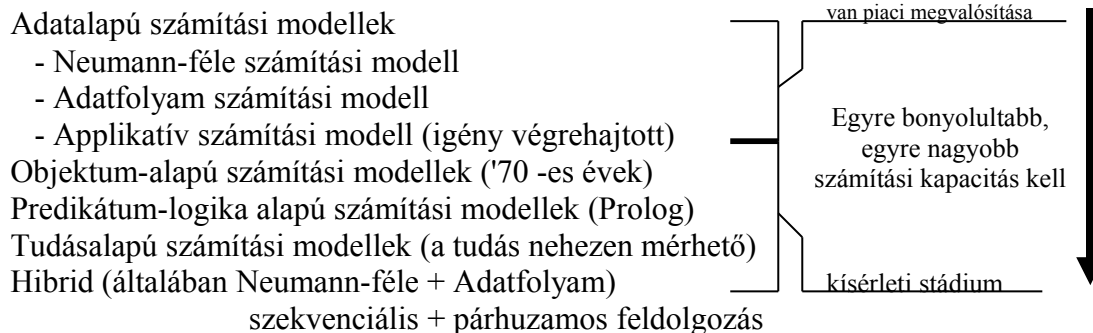
= változók kezelésére memória

= a következő utasítás címét tartalmazza egy speciális vas:

PC (Program Counter), amely inkrementál

A számítási modellek csoportosítása:

„Min hajtjuk végre a számítást?” elv szerint csoportosítunk:



Adataalapú számítási modellek

Adat jellemzői:

- az adatokat típusokhoz rendeljük
- az elemi adattípus esetében a típus meghatározza:
 - = az adat értelmezési tartományát
 - = az adat értékkészletét és
 - = a rajta értelmezett műveletek halmazát

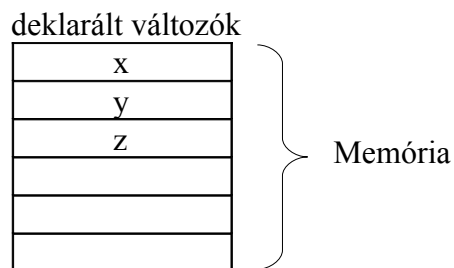
Pl. integer típus (16 bites előjeles):

- értelmezési tartomány: - 32768 ÷ + 32767 (0 is, ezért szimmetrikus)
- értékkészlet: kizárólag egész érték
- műveletek: +, -, *, /

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



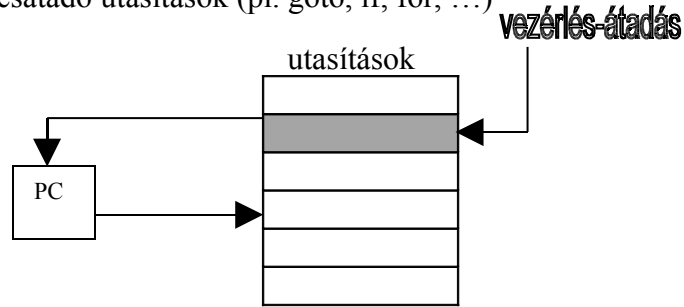
2. Hogyan képezzük le a számítási feladatokat?

- 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

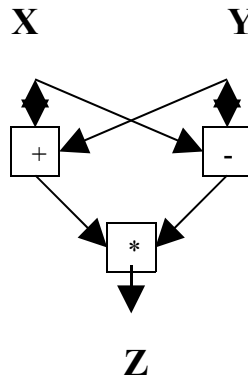
1. Min hajtjuk végre a számítást?

- adatokon, az adatokat bemenő adathalmaz képviseli, egyszeres értékadás!

2. 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

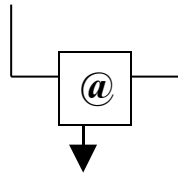
- addig az adatfolyam modellnél az összeadás és kivonás párhuzamosan végezhető, tehát példánkban 33%-os időmegtakarítást értünk el (3 időegység helyett csak 2).

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

3. Mi vezérli a végrehajtást?

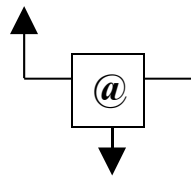
- adatvezérelt (más néven: stréber modell)

1. adat még nincs

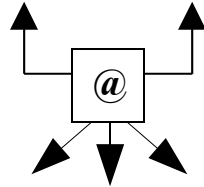


@: tetszőleges művelet

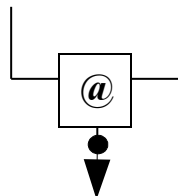
2. az egyik operandus rendelkezésre áll – adat megjelenik egy vezetéken



3. ha mindkét operandus biztosított, műveletvégzés, azonnal



4. az eredmény előállt



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

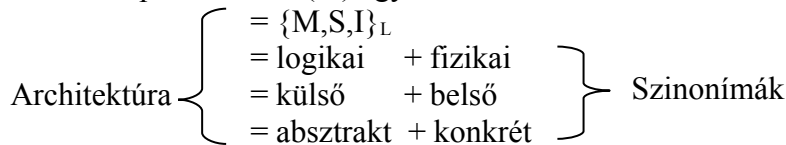
Programnyelvek: pl. Sigal

Architektúra: The Manchester Dataflow Machine

Számítógép architektúra

Architektúra fogalma:

1. 1964 Amdahl:
Mindazon ismeretek összessége, amit egy alacsony szintű nyelven programozónak ismerni kell ahhoz, hogy hatékony programot írjon.
Pl.: regiszterek, címzési módok, memória, utasításkészlet, utasítások végrehajtása
2. 1970: Bell:
Szinteket rendel az architektúra fogalmához.
 - PMS (Processor, Memory, Switches)
 - Programozói szint
 - o Magas szintű
 - o Alacsony szintű
 - Logikai áramköri szint – Rendszermérnöki szint
 - Áramköri szint
3. Egyéb:
 - külső jellemző
 - a belső felépítés
 - és működés együttese
4. Adott absztrakciós szinten (L) a számítási modell (M), a specifikáció (S) és az implementáció (I) együttese. A továbbiakban ezt a megfogalmazást fogjuk használni.



Logikai architektúra

- adott absztrakciós szinten a fizikai architektúra elvonatkoztatása
- logikai architektúra az adott absztrakciós szinten $= \{M, S\}_L$
- adott absztrakciós szinten a fekete doboz külseje

A processzor-szintű logikai architektúra (Instruction Set Architecture) részei:

- adattér
- adatmanipulációs fa
- állapottér
- állapotműveletek

A rendszerszintű logikai architektúra: Operációs rendszerrel interakciók

Fizikai architektúra

- adott absztrakciós szinten a logikai architektúra megvalósítása
- fizikai architektúra az adott absztrakciós szinten $= \{M, I\}_L$
(nem vagyunk kíváncsiak a specifikációra)
- adott absztrakciós szinten a fekete doboz belseje

A processzor-szintű fizikai architektúra (Microarchitecture) részei:

- műveletvégző
- vezérlő
- I/O rendszer
- Megszakítási rendszer

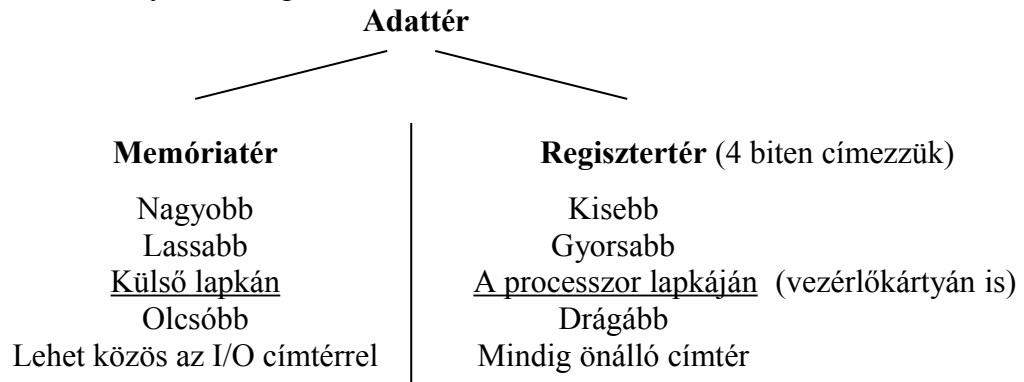
A rendszerszintű fizikai architektúra: Memória, sínrendszer

Egy korszerű számítógép szintjei:

- | | |
|---------------------------------|------------------------|
| - Alkalmazások | (Pl. Word, Excel) |
| ----- | |
| - Problémaorientált nyelv | (Pl. Pascal, C) |
| - Assembly szintű nyelvek | (Architektúra labor) |
| - Operációs rendszer gépi része | (Operációs rendszerek) |
| - Logikai architektúra | (Architektúra) |
| - Fizikai architektúra | (Architektúra) |
| - Digitális rendszer szintje | (Digitális technika) |
| ----- | |
| - Áramkörü szint | (Elektronika) |

Logikai architektúra (Processzor szintű)

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



Memóriatér

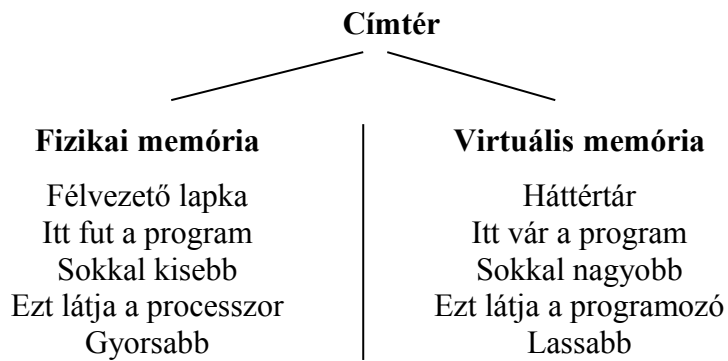
- Tárolási kapacitása az egyik legfontosabb tulajdonsága
- Kétféle címtér
 - Modell címtere: a címsín szélessége határozza meg a kapacitást (32 bit: 4GB)
 - Implementáció címtere: alkalmazás igénye, ill. anyagi lehetőségek határozzák meg
- A valós memória tárolási kapacitásának fejlődése:
 - 40-es évek: pár száz memóriarekesz
 - 1950 IAS 10 bites címsín, vagyis 1024 memóriarekeszt címezhetett meg
 - 1964 IBM360 16Mbyte címsín

Virtuális memória

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

Kulcsjellemezői:

- 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



Az Intel processzorcsalád címterei (nem lesz számon kérve):

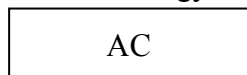
Típus	Dátum	Sínszél.	Valós (Mbyte)	Virtuális
8086	1978	20	1	-
80286	1982	24	16	1Gbyte
80386	1985	32	4096	64Tbyte

Regiszterek osztályozása

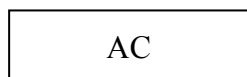
1. egyszerű regisztertér
2. adattípusonként különböző regisztertér
3. többszörös regisztertér

1. Egyszerű regisztertér

- 40-es évek: egyetlen akkumulátor



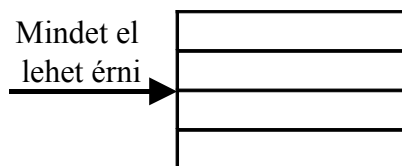
- 50-es évek: egyetlen akkumulátor + dedikált regiszter



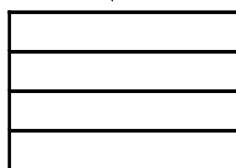
+



- 60-as évek: univerzális regiszterkészlet



- Veremregiszter (stack) ↓ Csak felülről lehet elérni



Értékelések:

Egyetlen akkumulátor – Neumann-féle architektúra

Hátránya

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

Egyetlen akkumulátor + dedikált regiszter (pl. hányados regiszter)

Előnye

- Gyorsította a műveletet

Hátránya

- Drága, továbbá az esetek többségében üresen áll

Univerzális regiszter

Előnye

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

Veremregiszter

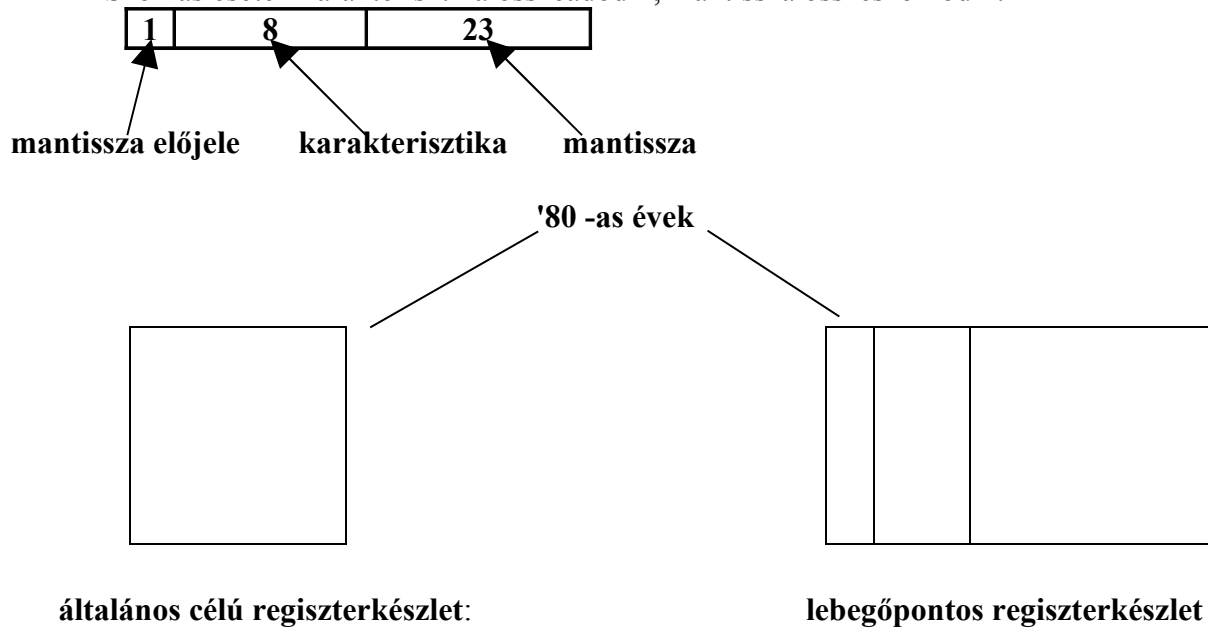
- Igen gyors
- Verem szervezésű

Mivel csak a verem tetejét látjuk, ezért szűk keresztmetszetet ad

2. Adattípusonként különböző regisztertér

Célja: az adatfeldolgozás gyorsítása - különös tekintettel a lebegőpontos adatábrázolásra.

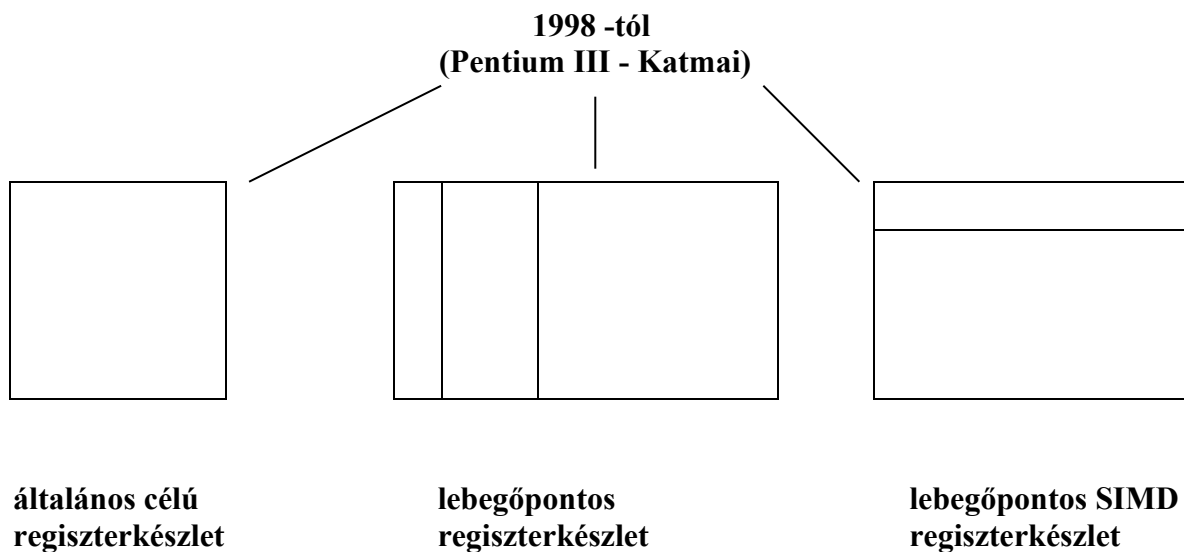
Szorzás esetén karakterisztika összeadódik, mantissza összeszorozódik.



általános célú regiszterkészlet:

fixpontos, karakteres, logikai
típusú adatok feldolgozásra

lebegőpontos regiszterkészlet



**általános célú
regiszterkészlet**

**lebegőpontos
regiszterkészlet**

**lebegőpontos SIMD
regiszterkészlet**

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

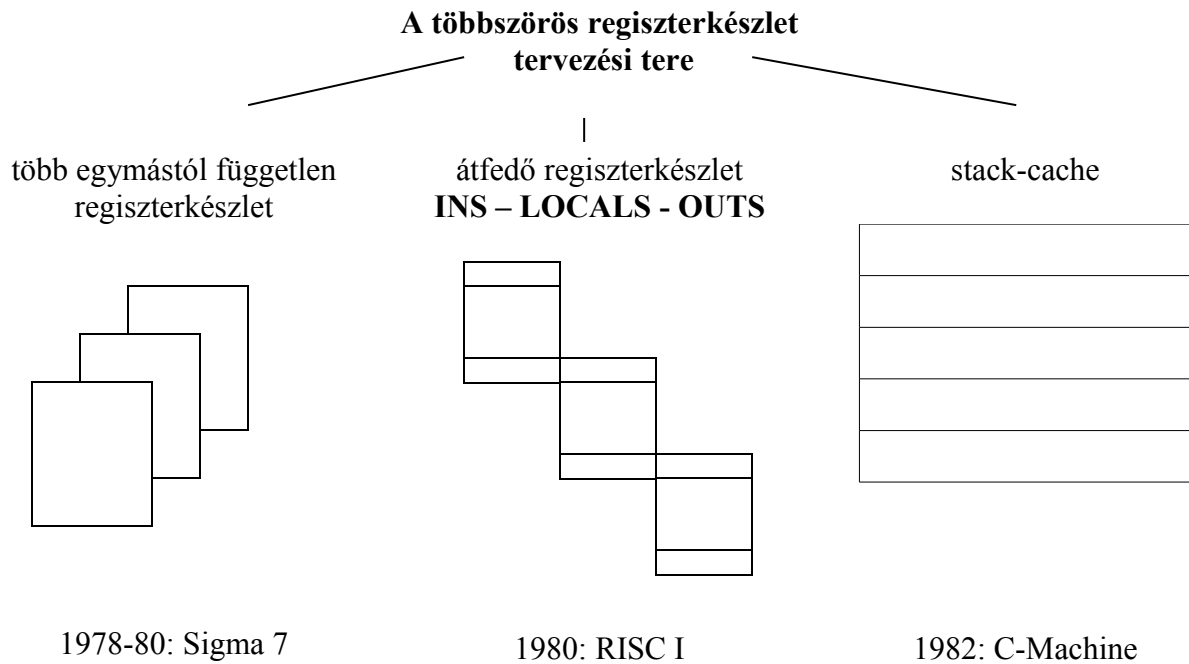
Típus	Megjelenés éve	Fixpontos regiszterkészlet	Lebegőpontos regiszterkészlet	Lebegőp. SIMD regiszterkészlet
IBM 360	1964	16*32	4*64	-
Intel i860	1989	32*32	16*64	-
IBM RISC 6000	1990	32*32	32*64	-
Intel 80386	1985	8*32	8*80	-
Intel Pentium III	1998	8*32	8*80	8*128

Ez a táblázat nem lesz számon kérve!

3. Többszörös regiszterkészlet

Háttér információ:

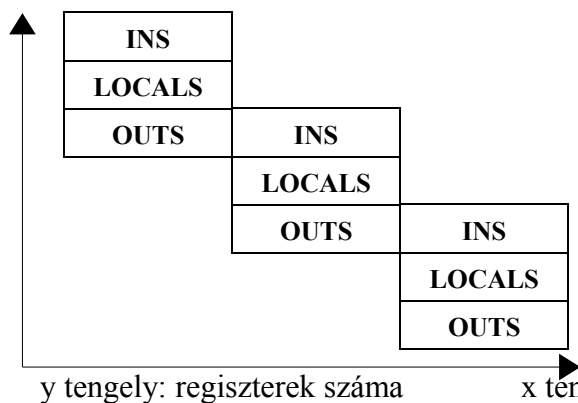
- Részei:
 - = a regiszterek aktuális tartalma
 - = az állapot-információk (flag)
- } Kontextus
- megszakítás esetén le kell mentenünk az éppen futó program kontextusát, annak érdekében, hogy majd a programot folytatni lehessen ugyanonnan
 - a többfeladatos és több felhasználós feldolgozásnál igen sok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



Több egymástól független regiszterkészlet:

- független folyamatoknál ideális, pl. megszakítások
- paraméter-átadásos eljárásnál nem gyorsít, mivel a paraméterátadás a memórián keresztül történik → lassú, az átadást gyorsítsukvalahogy:

Átfedő regiszterkészlet:

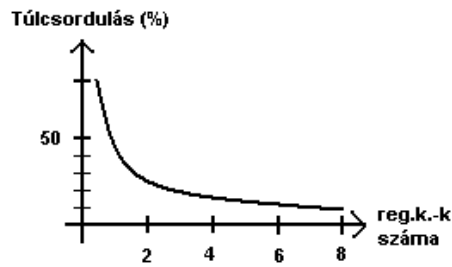


Jellemzői:

- 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 túlcsoordulás: ez a memória igénybevételével kerül feldolgozásra ⇒ lassul a feldolgozás

A paraméter-átadás problémájára megoldás.

- a regiszterkészletek száma:



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

= 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% túlsordulás)

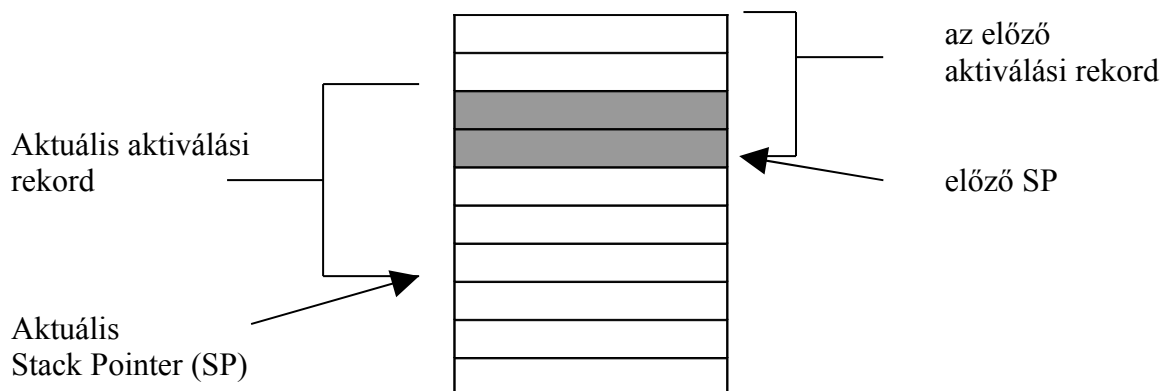
Probléma: fix és merev

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 túlsordulást kiküszöböljük)

Ezekről nem volt szó órán, de a 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

Adatmanipulációs fa

(Processzorszintű logikai architektúra 2. része)

1. Adattípusok: FX1 FX2 FX3 ... FP6 ...

2. Műveletek:



3. Operandusok típusai: rrr rmr ... mmm

4. Címzési módok (memória): R+D PC+D RI+D

Gépi kód:

01110100

r: regiszter
m: memória
D: displacement – eltolás
FX: Fixpontos
FP: Lebegőpontos (Float)
PC: Program Counter
RI: Indexregiszter

Az adatmanipulációs fa megmutatja,

- az összes adatmanipulációs lehetőséget
- másrészt egy al-fája megmutatja egy konkrét implementáció adatmanipulációs lehetőségeit

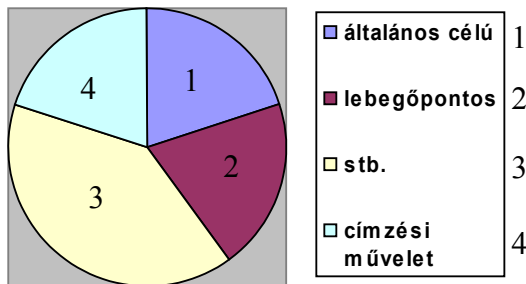
Az ideális az lenne, ha minden processzor képes lenne az összes adattípus, művelet, operandustípus, és címzési mód kezelésére. Ennek azonban akadályai vannak.

Adattípusok megvalósítása

- a 80-as évekig:

= technológiai korlát:

a lapka mérete korlátozta a lehetőségeket (segédprocesszor készlet)



= gazdasági korlát: igen drága a hardver → célkonfigurációk:

gazdasági célú: karakteres és BCD műveletek

műszaki-tudományos célú: lebegőpontos számítás

- napjainkban:

a technológiai fejlődés és az árak csökkenése eredményeképpen univerzális processzorok különféle műveletvégzőkkel.

Egy lapkán belül az általános és lebegőpontos is (miniatürizálás), de különálló feldolgozó egységek maradtak.

Adattípusok

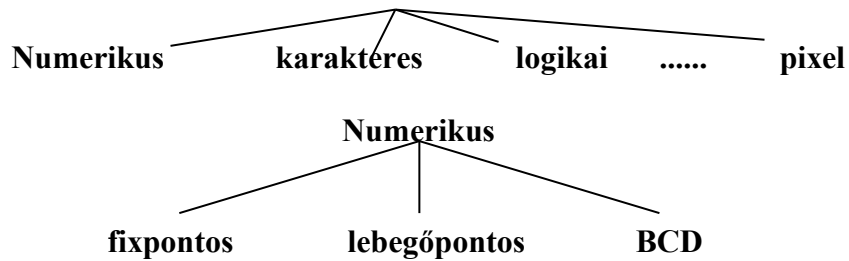
Elemi

verem
sor (FIFO)
fa
tömb
lista

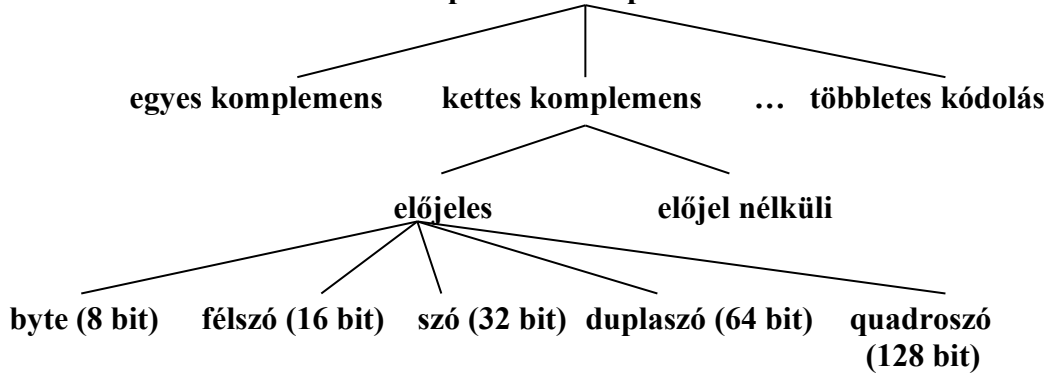
Összetett

- Memóriában, háttértáron helyezkednek el
- Nem képezik tárgyát az architektúrának
- Az összetett adattípusok az elemi adattípusokból épülnek fel.

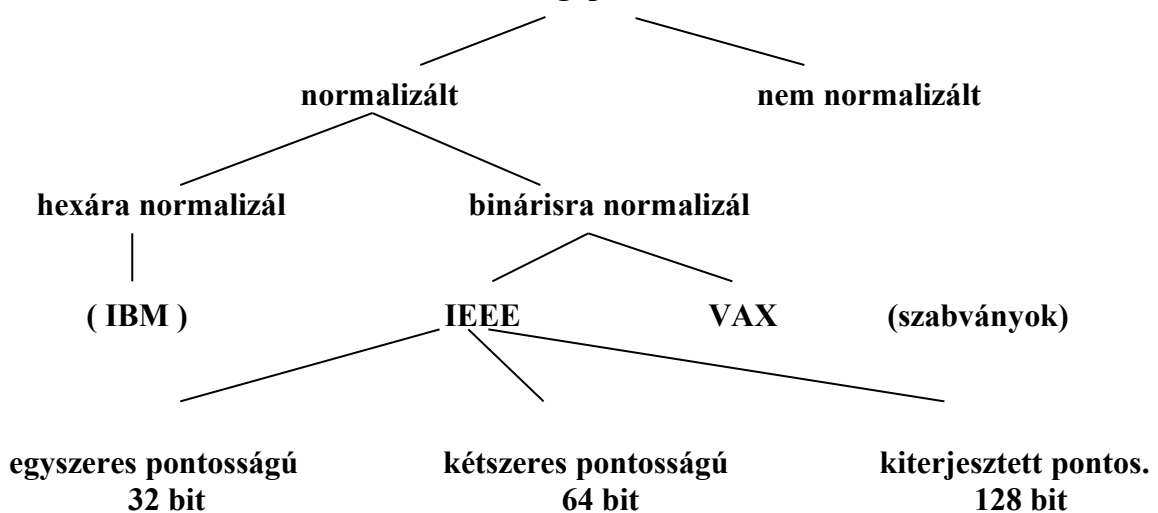
Elemi adattípusok

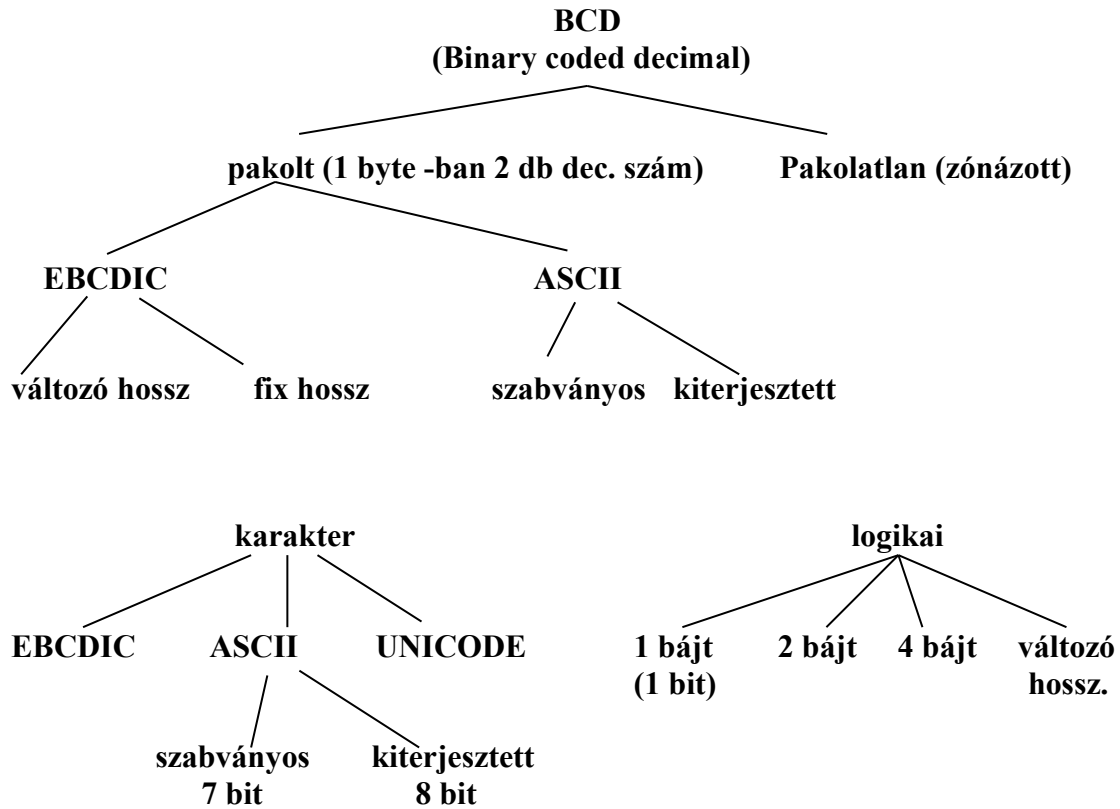


Fixpontos adattípusok



Lebegőpontos





Műveletek

Pontosan kell a műveleteket deklarálni, beleértve a kivételek kezelését is

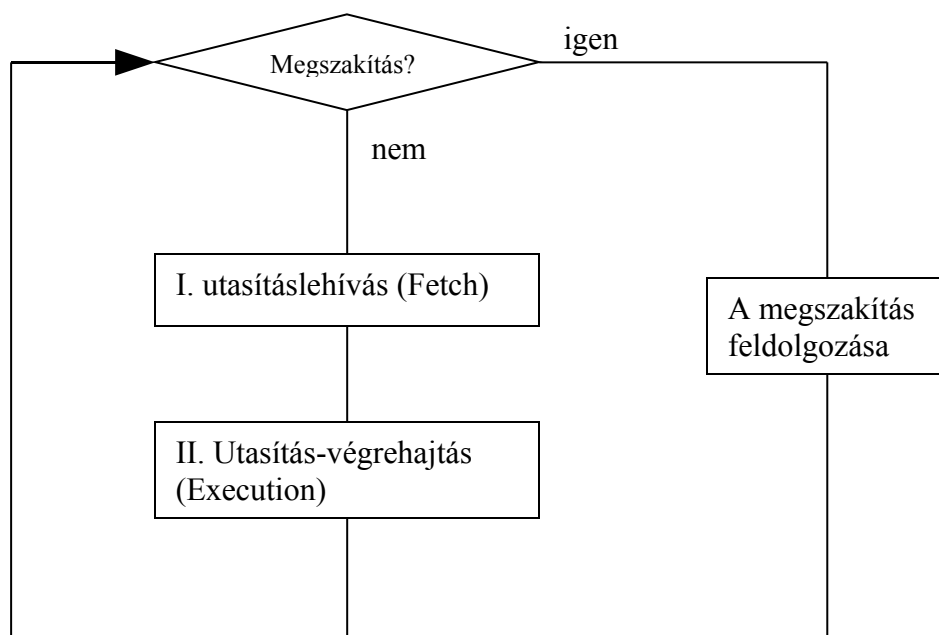
Operandus típusok

Bevezetés: Az utasítás-feldolgozás menete

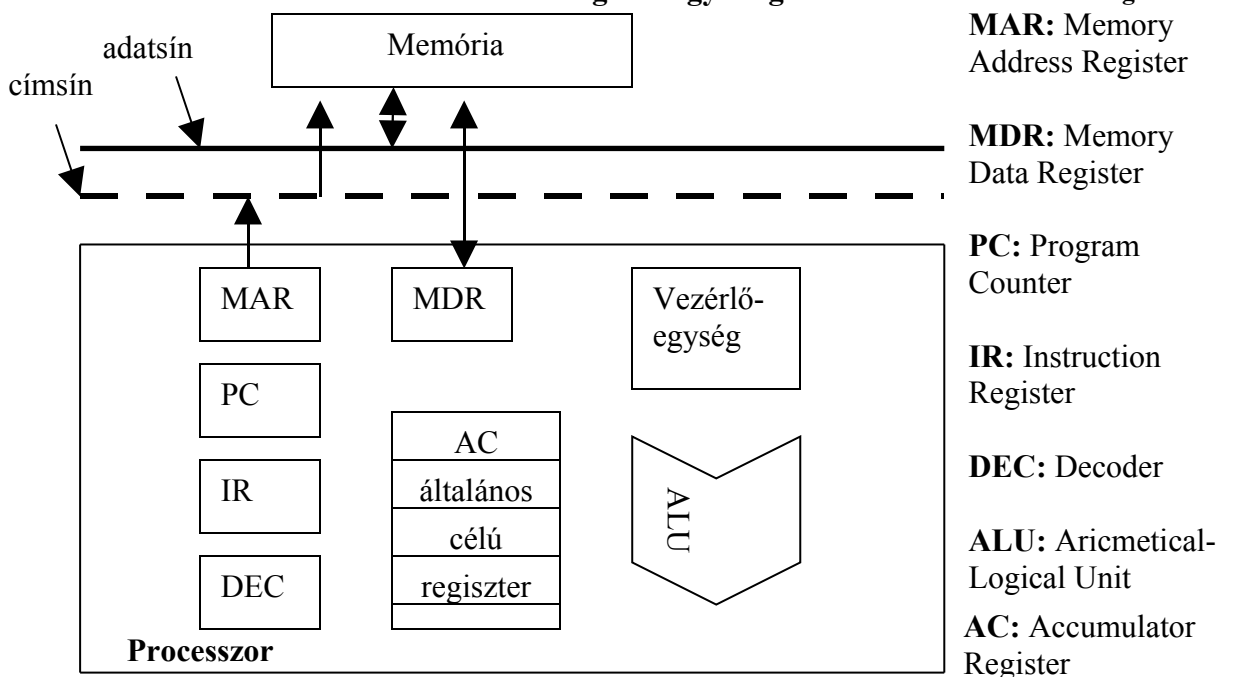
Egy gépi kódú utasítás általános formája:



Az utasítás-feldolgozás általános folyamatábrája:



2 utasítás között lehet csak feldolgozni egy megszakítást!



I. Utasítás-lehívás

A PC mindig a következő végrehajtandó utasítás címét tartalmazza

MAR ← PC
 MDR ← (MAR)
 IR ← MDR
 PC ← PC+1 (Következő utasítás címét letárolja a PC)

Az utasítás lehívás minden utasítás esetén megegyezik.

II. Utasítás-végrehajtás (minden utasítás esetén eltérő)

- adatbehívás (load)

DEC ← IR
 MAR ← DEC címrész
 MDR ← (MAR)
 AC ← MDR

- aritmetikai-logikai utasítás, pl. összeadás (MK -tól függ)

DEC ← IR
 MAR ← DEC címrész
 MDR ← (MAR)
 AC ← AC + MDR vagy
 AC ← AC - MDR vagy
 AC ← AC * MDR vagy
 AC ← AC / MDR

- adattárolás (store) [cím] = érték

DEC ← IR
 MAR ← DEC címrész
 MDR ← AC
 (MAR) ← MDR

- a feltétlen vezérlésátadás

DEC ← IR
 PC ← DEC címrész
 Ez a PC felülírása a gyakorlatban

Operandus típusok

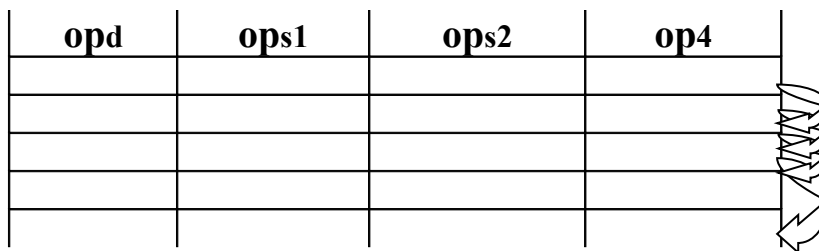
Bevezetés 2: Az utasítások fajtái (utasítás-típusok)

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: következő utasítás címét speciális hardverben tároljuk: PC

- $op_d := op_{s1} @ op_{s2}$
- az eredmény helyének explicit deklarációja
- 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, akkumulálódik az eredmény a gyűjtőben!
- Olyan helyeken használják ahol nagy mennyiségű adat van és nincs elágazás.

2 címes utasítás

- $op_{s1} := op_{s1} @ op_{s2}$ vagy
- $op_{s2} := op_{s1} @ op_{s2}$
- 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
 Pl. 1951 IAS – ötvenes-hatvanas években fordultak elő ilyen architektúrák
 '70 -es években már nem

0 címes utasítás

- fajtái
 - = NOP – no operation
 - = a műveleti kód tartalmazza az operandust is, pl. CLEAR ⇒ a Dflag törlése
 - = verem-műveletek PUSH, POP

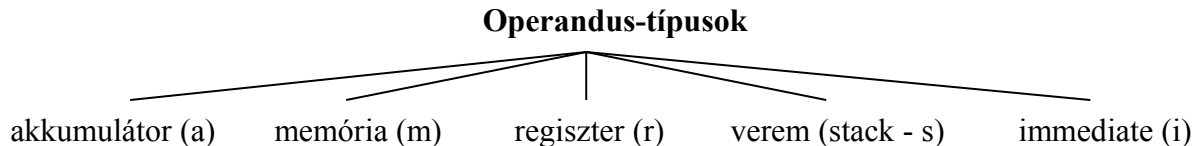
Napjaink trendje

3 címes utasítások a RISC gépekben

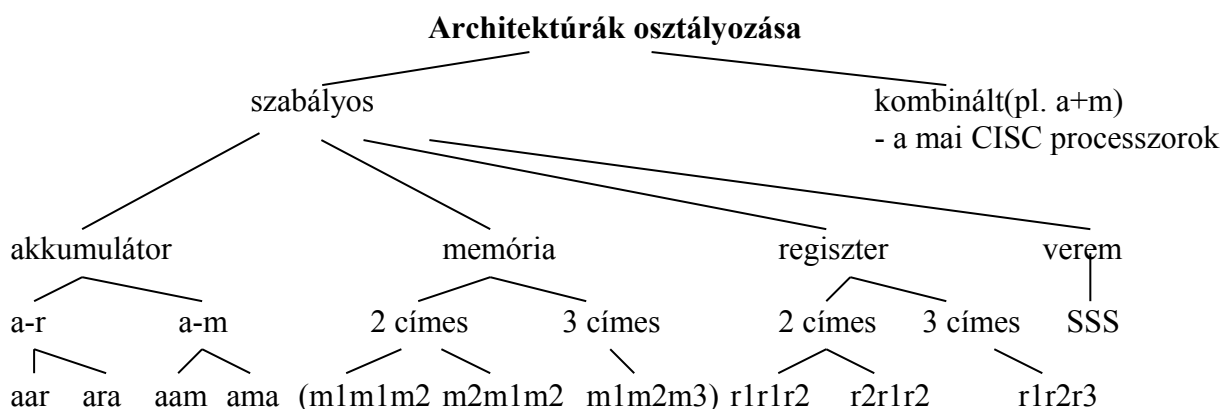
- mindhárom cím regiszterbeli operandusra mutat (r r r)
- csak a Load/Store utasítás engedélyezett

2 címes utasítások a mai CISC gépeket jellemzik:

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



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



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

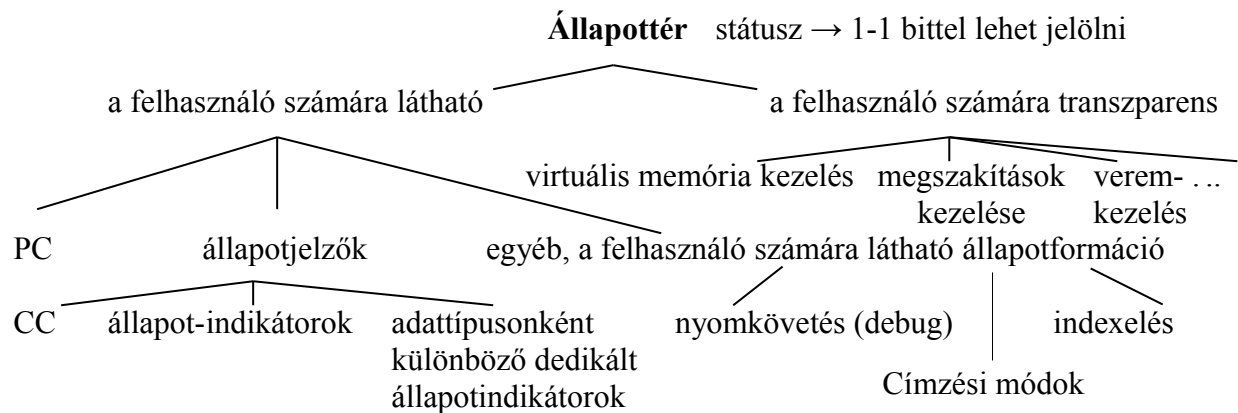
- előny:
 - nagy címtér
- hátrány:
 - lassú
 - hosszú cím \Rightarrow hosszú utasítás
- napjainkban nem aktuális

Verem

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

Gépi kód

Minden architektúra esetén más.



CC - Condition Code

- 2 bites ⇒ négy érték felvétele (más-más jelentéseket rendelnek hozzá)
- IBM 360

Állapotjelzők (indikátorok, flag):

- minden helyi érték saját jelentéssel bír, pl. negatív, nulla, túlsordulás

Adattípusonként különböző állapotjelzők:

- minden regiszterkészlet típushoz hozzárendelnek külön állapot-indikátor készletet, pl. az általános célú processzorhoz tartozó flageken kívül a lebegőpontos processzorhoz is hozzárendelnek külön flagkészletet.
- a lebegőpontos flagkészlet eseményei pl. alul-, túlsordulás, denormalizált szám, 0

Megszakítás esetén történő mentés:

- IBM 370 ⇒ PSW ⇒ Program Status Word ⇒ megszakításkor a PSW -t mentették
- kontextus:
 - = a regiszterek aktuális értékei
 - = az állapottér aktuális értékei (flag, PC) – állapotinformáció
- megszakítás esetén a kontextus kerül lementésre

Állaptműveletek

PC

- inkrementálás
- felülírás egy utasításból vett címmel (elágazásnál)

flag

- save (mentés)
- load (visszatöltés)
- set (beállítás)
- clear (törlés)
- reset (kezdeti értékkel beállítás)

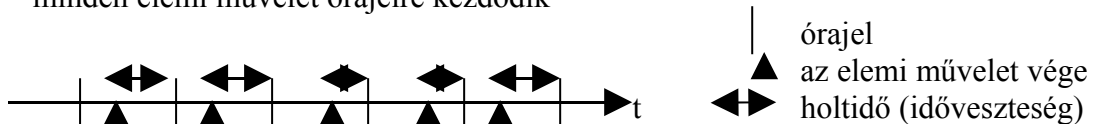
A processzorszintű fizikai architektúra

Processzor = műveletvégző + vezérlő

Szinkron-aszinkron processzorok

- Szinkron:

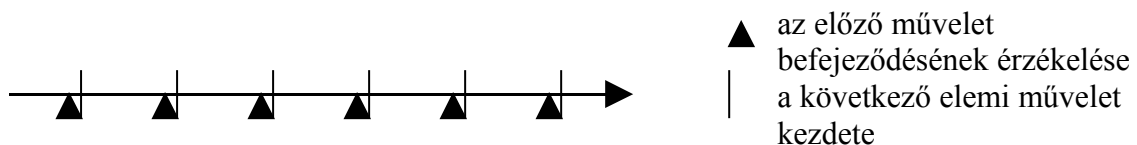
- = egy elektronikus óra meghatározott időnként órajelet generál
- = minden elemi művelet órajelelre kezdődik



- = hátránya: az elemi műveletek különböző ideig tartanak \Rightarrow holtidőt eredményez
- = előnye: az elektronikus óra és az órajel vezetése egyszerű és olcsó
- = A mai piacon jelenleg ez a domináns

- Aszinkron:

- = minden elemi művelet befejeződése egyben jelzés a következő elemi művelet elkezdésére



- = az elemi művelet befejeződésének érzékelése időt vesz igénybe \Rightarrow idővesztés
- = az elemi művelet befejeződésének érzékelője bonyolult, drága

A ma piacon lévő processzorok szinkron vezérlésűek \sim 3GHz frekvenciával

1. Műveletvégző egység

Részei:

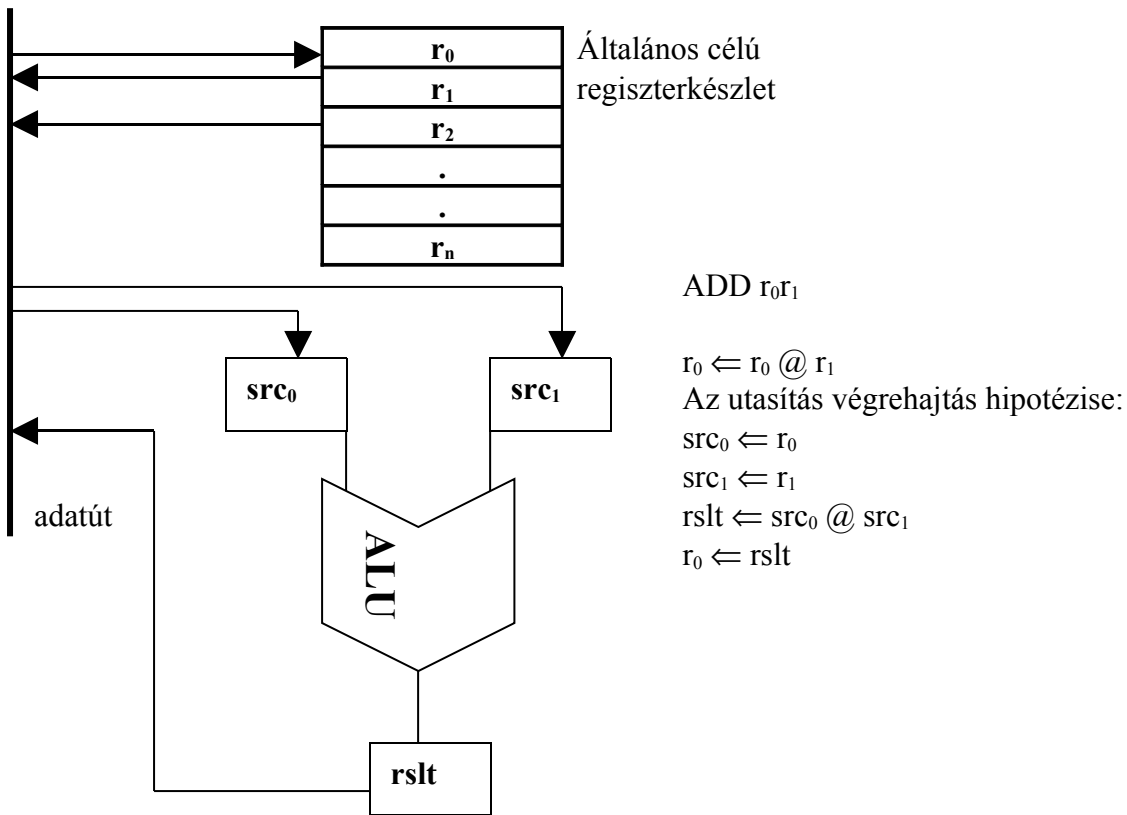
- Regiszterek
- Adat utak
- Kapcsolópontok
- Szűk értelemben vett ALU

Regiszterek

- a felhasználó számára látható (amire a programozó hivatkozni tud) ezt vizsgáltuk a logikai architektúrában
- rejtett regiszterek az adat feldolgozási technológiához szükséges puffer regiszterek ezekre nem lehet hivatkozni

Adat utak

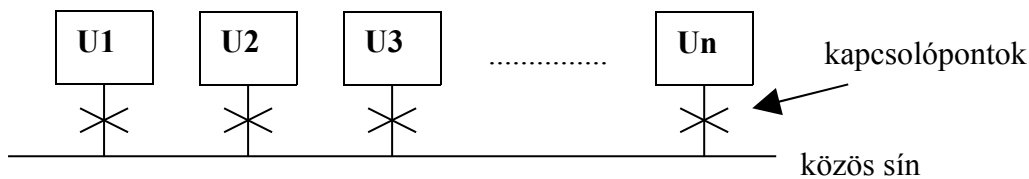
- a memóriához való hozzáférést a vezérlő rész kezeli. A műveletvégzőben tehát nem értelmezett a címek kezelése → ezért adatút, nem pedig adatsín



- az src_0 , az src_1 és a $rslt$ rejtett regiszterek
- az első operandus helyére kerül az eredmény

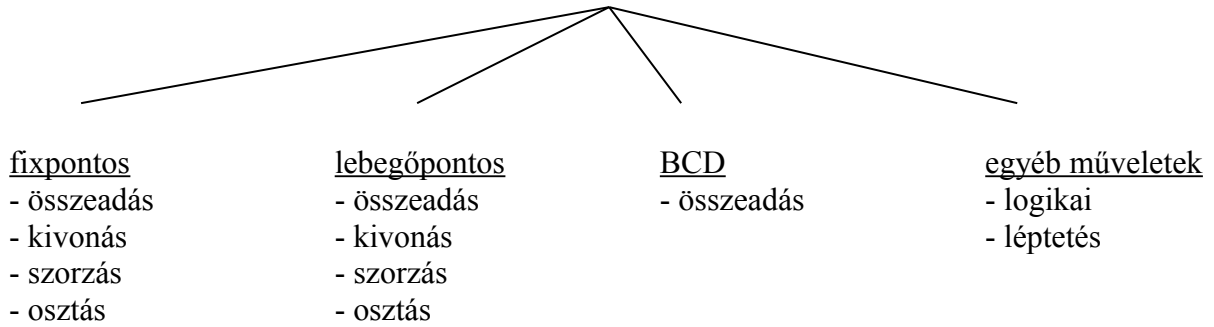
Kapcsolópontok

- a sín nem megosztott eszköz → egy időben csak egyetlen adó lehet



- a kimenő kapu 3 állású (kapcsolópontok állásai):
 = 0
 = 1
 = zárva (z)
- a bemenő kapu 2 állású:
 = nyitva
 = zárva
- a kapcsolópontok architektúráisan általában a regiszterek részét képezi

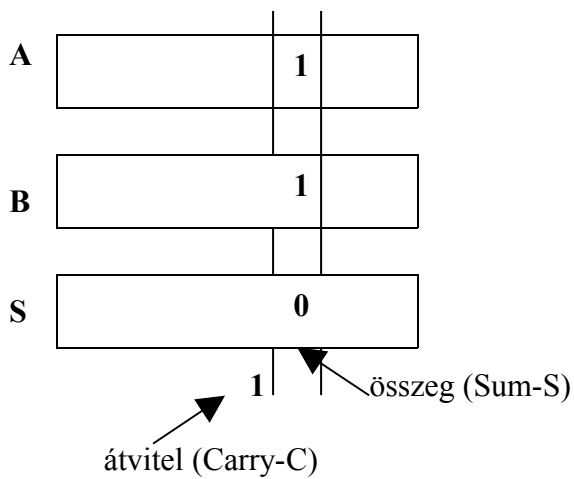
Szűk értelemben vett ALU



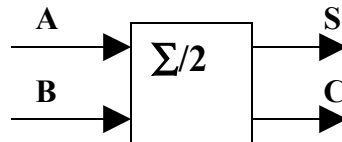
Fixpontos összeadás

- mivel a kivonást, a szorzást és az osztást az összeadásra vezetjük vissza, ezért a sebesség meghatározó

Egybites félösszeadó



Jelölése: (félösszeadó)



Igazságtáblázat:

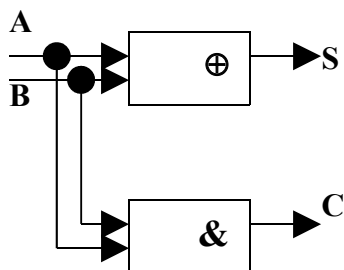
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Logikai függvények felírása:

$S = A \oplus B$ (XOR)

$C = AB$

Megvalósítás:

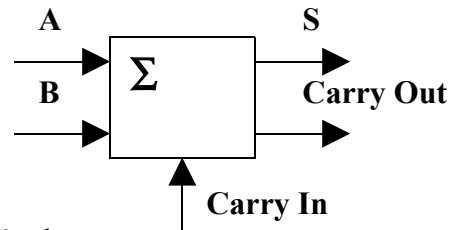


Hátránya:

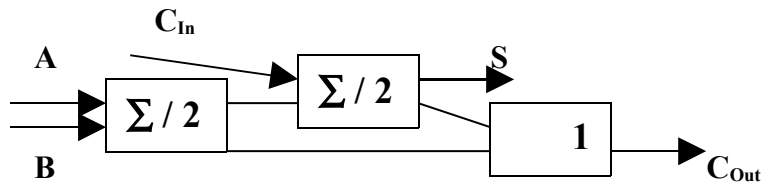
- nem kezeli a bejövő Carry -t

Egybites teljes összeadó

- kezeli a bejövő átvitelt (C)



- megvalósítása 2 db fél-összeadóval

Tervezése:

1, Az igazságtábla felírása

A	B	C _{in}	S	C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

2, logikai függvények felírása (C_{in} ⇒ C)

$$C_{out} = nABC + AnBC + ABnC + ABC$$

3, Azonos átalakítások a következő célfüggvényekkel

- az elemszám minimalizálásával
- a végrehajtási idő minimalizálásával

Felhasznált azonosságok: $ABC = ABC + ABC + ABC$
 $nA + A = 1$

$$C_{out} = nABC + AnBC + ABnC + ABC + ABC + ABC = (nA + A)BC + (nB + B)AC + (nC + C)AB = BC + AC + AB$$

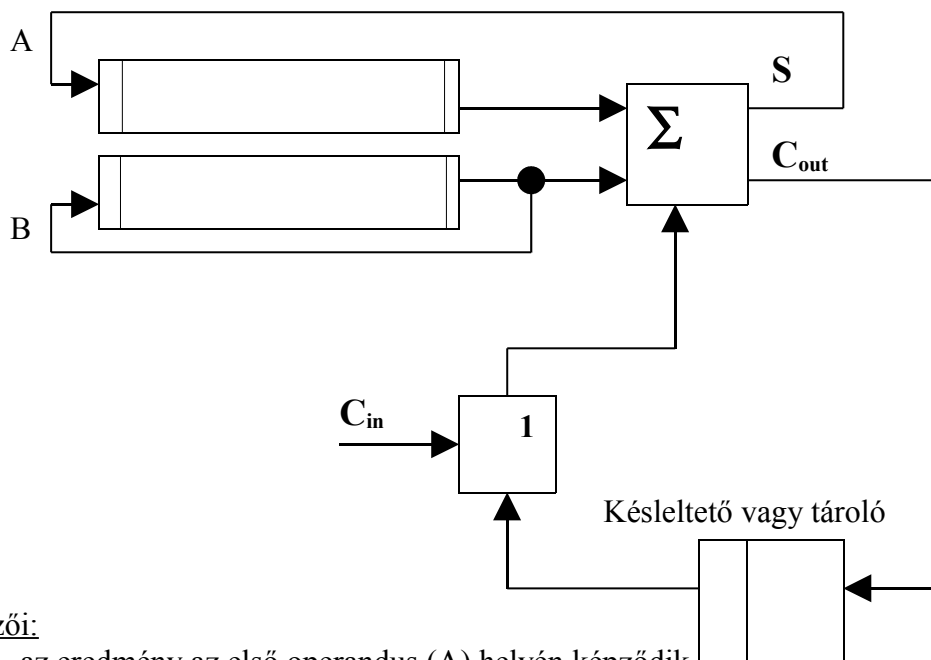
- i: - 4db elem
 - 20% megtakarítás
 - Ha 1 kapu késleltetése d, akkor T = 3d (időigény)
- ii: - 33% időmegtakarítás
 - T = 2d

4, Megvalósítás

N – bites soros összeadó

Megjelenésének oka: a számítógépben a számokat tipikusan több/változó bit hosszúságú regiszterben tárolják \Rightarrow ezek tartalmát kell összeadni.

Megvalósítása:



Jellemzői:

- az eredmény az első operandus (A) helyén képződik
- az A és a B léptető regiszter (vezérlésoptimalizálás szempontjából jó)
- a C_{out} esetén biztosítani kell, hogy az előző bithelyiértéken képződő átvitel az aktuális bithelyiérték összeadásakor érkezzen be. Erre szolgál a tároló (puffer) vagy késleltető elem.
- a bejövő átvitel (C_{in}) csak az első bithelyiérték esetén vesszük figyelembe.

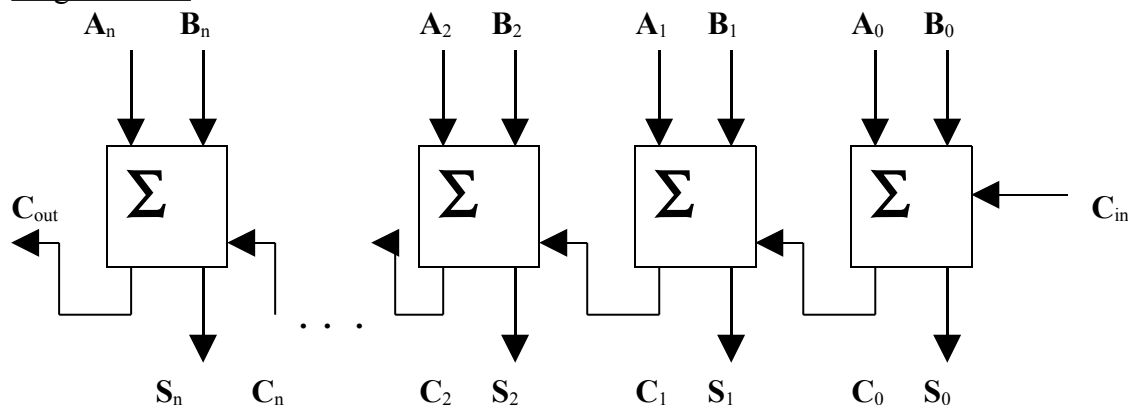
Értékelése:

Ha az egybites teljes összeadó végrehajtási idejét t , akkor az n - bit összeadási ideje: $T = n \cdot t$.

Gyorsítás:

- t csökkentése: igen gyors egybites teljes összeadó alkalmazása.
- csökkentjük az n értékét egyre oly módon, hogy minden bithelyiértékhez hozzárendelünk egy egybites teljes összeadót \Rightarrow n - bites párhuzamos összeadó.

Megvalósítás:



Működése:

I.

$$\begin{array}{r} \text{A } 0101 \\ + \text{B } 1010 \\ \hline 1111 \end{array}$$
 $C_{in}=0, T_{min}=t ;$

II.

$$\begin{array}{r} \text{A } 0101 \\ + \text{B } 1010 \\ \hline 1110 \\ 1100 \\ 1000 \\ 0000 \end{array}$$
 $C_{in}=1, T_{max} = n*t ;$
 1.lépés 1110 $C_0=1$
 2.lépés 1100 $C_1=1$
 3.lépés 1000 $C_2=1$
 4.lépés 0000 $C_3=1$

Értékelése:

- igen nagy befektetés árán (megtöbbszöröztük az egybites összeadókat számát)
- csupán hullámzó teljesítményt értünk el, mert meg kell várni az átvitel soros terjedését

Átvitel előrejelzés

(Carry-Look-Ahead = CLA)

Alapelve:

$$C_{out} = A * B + (A + B) * C_{in}$$

Az egyes bithelyiértékeken képződő átvitel függ

- a két bejövő operandustól és
 - a kívülről bejövő átviteltől,
- de nem függ az egyes bithelyiértékeken képződő átviteltől.

ez előre ismert, csak ezektől függ!!

Jelölések: $A * B \Rightarrow G$ (Generate)

$A + B \Rightarrow P$ (Propagate)

$$C_i = G_i + P_i * C_{i-1}$$

$$C_0 = G_0 + P_0 * C_{in}$$

$$C_1 = G_1 + P_1 * C_0 = G_1 + P_1 * G_0 + P_1 * P_0 * C_{in}$$

$$C_2 = G_2 + P_2 * C_1 = G_2 + P_2 * G_1 + P_2 * P_1 * G_0 + P_2 * P_1 * P_0 * C_{in}$$

$$C_3 = G_3 + P_3 * C_2 = G_3 + P_3 * G_2 + P_3 * P_2 * G_1 + P_3 * P_2 * P_1 * G_0 + P_3 * P_2 * P_1 * P_0 * C_{in}$$

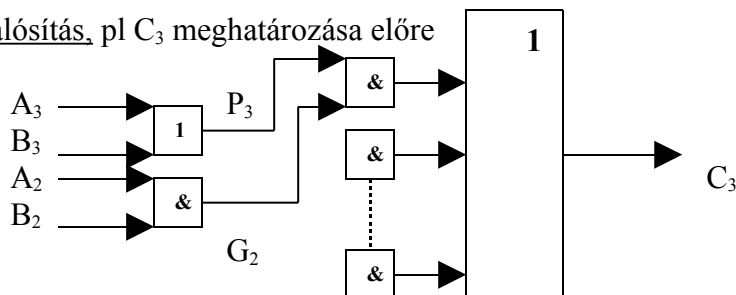
Értékelés:

- minden átvitel meghatározásához szükség van
 = ÉS kapuk sorozatára és
 = ezeket összekapcsoljuk VAGY kapuval
- a P és G meghatározása további

2 fokozat
 1 fokozat
 $\Sigma: 3$ fokozat

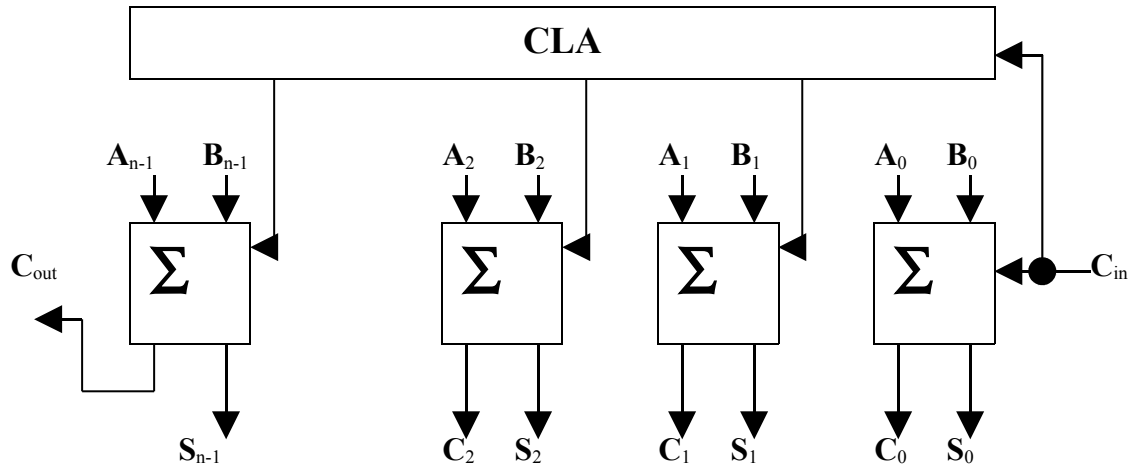
Amennyiben egy fokozat késleltetése **d**, akkor az egyes bithelyiértéken képződő átvitel meghatározásának ideje: **T=3d**.

Megvalósítás, pl C₃ meghatározása előre

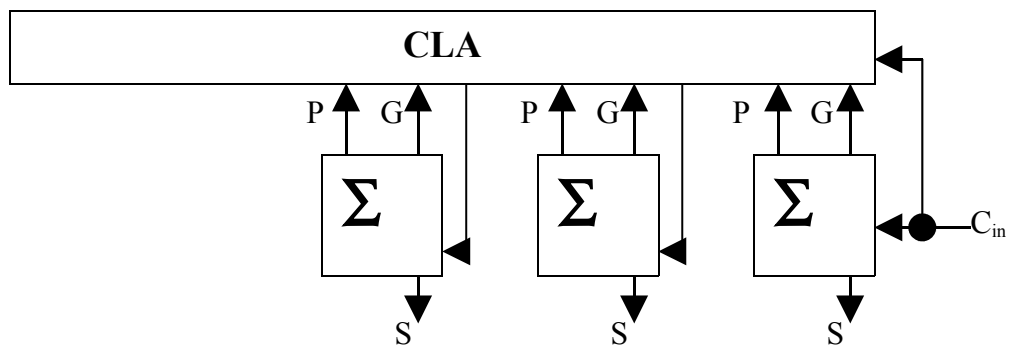


Megvalósítási alternatívák

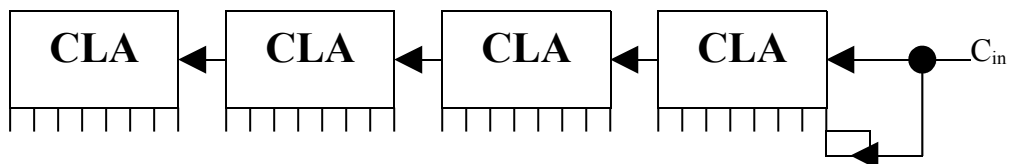
1. Katalógus áramkörökkel egybites teljes összeadó + CLA



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

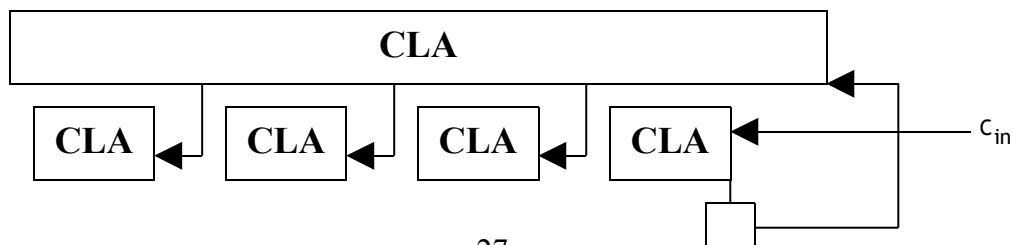


3. A VAGY kapu bemeneteinek száma technológiai korlátokba ütközik, ezért maximum 8 bit vonatkozásában építhető meg a CLA.
32 bit megvalósítási lehetősége:



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

4. CLA a CLA-k számára:



Fixpontos kivonás

Alapelvek:

- a fixpontos összeadáshoz hasonlóan megtervezhető
 - = az 1 bites fél-kivonó
 - = az 1 bites teljes kivonó
- az alkalmazásának hátrányai:
 - = a kivonó csak akkor ad bit helyes eredményt, ha a nagyobb számból vonjuk ki a kisebbet. Ehhez minden kivonás előtt komparálni kell \Rightarrow lassú
 - = a kivonó megvalósítása megduplázza a műveletvégző elemszámát \Rightarrow a lapka mérete viszont korlátozott \Rightarrow nem célszerű

Megvalósítása: a számok átkódolása, kivonás visszavezetése összeadásra.

Hagyományos kivonás:

A	13	1101
<u>-B</u>	-(+7)	0111
X		ezt másképpen végezzük

Egyes komplement

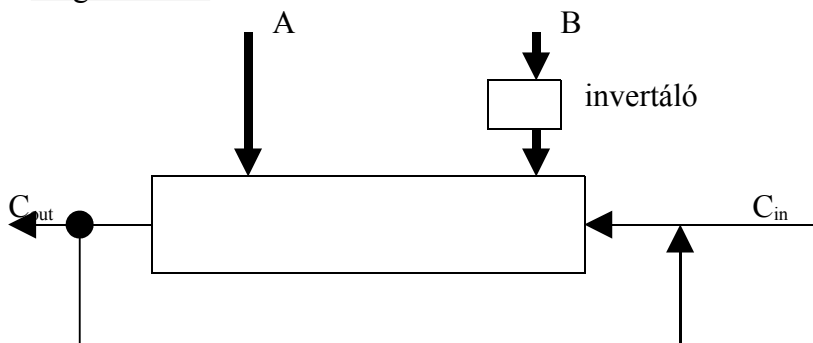
A	13	0 1101
<u>-B</u>	+(-7)	1 1000
X	6	10 0101 = +5

\swarrow
 \searrow
 0110 = +6

Egyes komplement meghatározása:

- pozitív számé maga a szám
- negatív szám esetén bitenként invertálunk

Megvalósítás:



Kettes komplement

Meghatározása:

- az egyes komplementhez hozzáadunk egyet (számítógépes megoldás)
- a bináris számból hátulról leírjuk az összes nullát és az első egyest, és utána bitenként invertáljuk.

$$\begin{array}{r}
 \text{Pl.} \quad A \quad A \quad 13 \quad 0|1101 \quad \text{segédszámítás: } 7=0111 \Rightarrow 1001\text{-kettes komplemente} \\
 - B \quad +(-B) \quad +(-7) \quad \underline{1|1001} \\
 \hline
 X \quad X \quad +6 \quad 00110 = +6
 \end{array}$$

Az ideális: tudjon összeadni is.

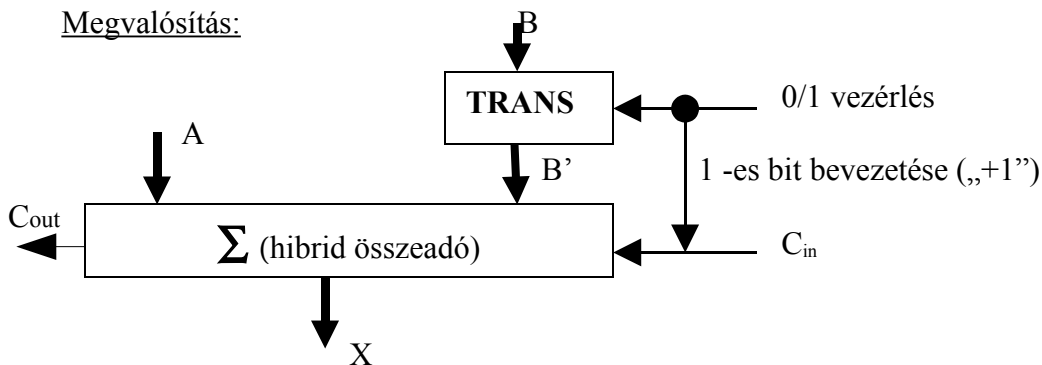
TRANS egység kialakítása (egyes komplementet képz, átkódol)

Igazságtábla:

Vezérlés	B	B'
0	0	0
0	1	1
1	0	1
1	1	0

- kizáró vagy (XOR)
- amennyiben a vezérlés nulla, akkor a B értékeit változatlanul átengedi, amennyiben 1, akkor bitenként invertál.

Megvalósítás:



Szorzás/Osztás

- minden műveletvégzőnek kell tudni
 - = összeadni
 - = invertálni
 - = léptetni,
 de nem kell tudni szorozni/osztani, mivel az felépíthető az említett három műveletből.
- nem szükséges hardveres úton megvalósítani
- az aritmetikai műveletekhez összeadás és invertálás szükséges

Architekturális megvalósítása a szorzás/osztásnak

Fejlődése:

50-80 -es években

Alkatrész	szorzás és osztás megvalósítása	Teljesítmény(sebesség)
olcsó processzor	gépi kódú programmal	lassú
közepes áru processzor	mikro programmal	közepes
drága processzor	hardveres úton	gyors

90 -es évektől:

- PowerPC 604

= kettő db műveletvégző az egyszerű fixpontos műveletekhez (+, -)

= egy db műveletvégző az összetett fixpontos műveletekhez (*, /)

- Pentium Pro fixpontos műveletvégzői

= általános célú (2 db)

= léptető (1 db)

= egész osztó (1db)

} Szakosodott műveletvégzők

Fixpontos szorzás

Hagyományos szorzás

$$\begin{array}{r}
 X=A*B=13*123 \\
 \quad 39 \\
 \quad 26 \\
 \underline{13} \\
 1599
 \end{array}$$

Algoritmikus módszer

$$\begin{array}{r}
 13*123 \\
 0000 \leftarrow \text{gyűjtő} \\
 \underline{39} \\
 0039 \\
 \underline{260} \\
 299 \\
 \underline{1300} \\
 1599
 \end{array}$$

Léptető módszer

$$\begin{array}{r}
 \underline{13*123} \\
 0000 \\
 \underline{39} \\
 0039 \\
 \underline{26} \quad (\text{balra léptetés}) \\
 299 \\
 \underline{13} \\
 1599
 \end{array}$$

felveszünk egy gyűjtőt és ki-nullázzuk, a korábbi eredményeket elveszítjük.

Az összeadás ciklus annyiszor fut, ahány helyiértékű a szorzó. Ez is tartalmaz szorzást!

Bináris szorzás sajátosságai

- A bináris szám hossza

Decimális		Bináris	
Helyiértékek száma	P1.:	Helyiértékek száma	
1	9	4	← Ennyivel lehet leírni
2	99	7	
3	999	10	

Konklúzió: a bináris számok 3-4x hosszabbak, mint a decimális számok

→ az összeadási ciklusok száma magasabb lesz

- A szorzat hossza

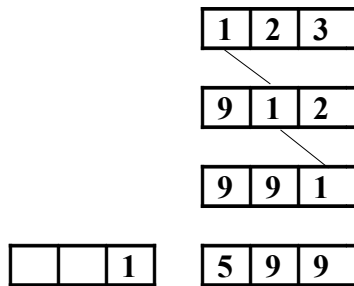
Decimális helyiértékek száma				Általános formában
A	1	2	2	m
B	1	1	2	n
X	2	3	4	m+n

$9*9 < 99, 99*9 < 999, 99*99 < 9999 \dots m*n \leq m+n$

Konklúzió: mivel a szorzandó és a szorzó is egy-egy regiszterben helyezkedik el, így a szorzat kettő regiszterben keletkezik.

Pl.: a szorzat kisebb helyiértékű része a szorzó helyén képződik.

Legyen 3 helyiértékes a léptetőregiszterünk. Részeredményeket nem tárol.



A szorzás gyorsítása

- bitsoportokkal való szorzás

= a léptetés nem egyesével, hanem csoportonként hajthatjuk végre \Rightarrow gyorsabb

= Pl. 2-es bitsoportok

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

$$\begin{array}{r}
 7*9 = 63 \\
 0111 * 1001 \\
 \underline{0000} \\
 0111 \\
 1110 \\
 \hline
 111111 = 63
 \end{array}$$

Segédszámítás: a szorzandó kétszeresének meghatározása

A, összeadással:

$$\begin{array}{r}
 0111 \\
 \underline{0111} \\
 1110
 \end{array}$$

B, léptetéssel: $0111 \rightarrow 1110$
kétszeresét kapjuk!

Az utóbbi (11) csak ELVBEN létezik, ennek kiváltása a gyakorlatban:

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

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

=Pl.

*62	111110	5 db összeadás
helyette:		
*64	100000	1 db összeadás
*2	000010	1 db összeadás
-		1 db kivonás
		3 db összeadás

40%-os gyorsítás

Csökkenti a szorzóban lévő egyesek számát.

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

Fixpontos osztás

Hagyományos osztás:
 $X=A/B=150/48$

150	I.	3,1
<u>-48</u>		
102	II.	
<u>-48</u>		
54	III.	
<u>-48</u>		
60	I.	
<u>-48</u>		
120		

Ha már nem tudja kivonni, kiírja a számjegyet, és szubrutin: **csökkenti a szorzót 1/10 -re.** (mintha az osztandót növelné)

Geometriai értelmezés:

0,48

Hátrány: minden kivonás előtt kell komparálnunk \Rightarrow lassú

Visszatérés a nullán át

150	I.	3,1
<u>-48</u>		
102	II.	
<u>-48</u>		
54	III.	
<u>-48</u>		
6	I.	
<u>-48</u>		
-42		
<u>+48</u>		
60	I.	
<u>-48</u>		
12	10 -el való	
<u>-48</u>	szorzás után	
-36		
<u>+48</u>		
120		

Geometriai értelmezés:

Előny: a kivonások előtti komparálások helyett csak előjel flag-et vizsgálunk \Rightarrow gyors
 előjel flag változása: szubrutin; kiírja a számjegyet, osztót hozzáadja az eredményhez, szoroz tízzel, majd folytatja a kivonást.
Hátrány: felesleges munkát végzünk a visszatérés miatt.

Visszatérés nélküli osztás

$X=11/6$

11	I.	1,83
<u>-6</u>		
5	10.lépés	
<u>-6</u>		
-10	9.lépés	
<u>+6</u>		
-4	8.lépés	
<u>+6</u>		
+20	I.	
<u>-6</u>		
14	II.	
<u>-6</u>		
8	III.	
<u>-6</u>		
2		
<u>-6</u>		
-40		

Amint 0 -t átlépjük, osztót csökkentjük a tizedére, majd ezzel közelítünk 0 -hoz, amíg azt át nem lépjük, utána folyt. flaget használ, a felesleges átlépkedést kihasználja. Ez a leggyorsabb módszer.

Geometriai értelmezés:

0,6 0,6

Lebegőpontos műveletek

A lebegőpontos ábrázolás kialakulásának oka:

A fixpontos ábrázolás hátrányai

- szűk értelmezési tartomány. Pl.: integer esetén: $-32768 \div +32767$
- a tört számok pontatlan ábrázolása
 - Ha a kettedes pontot a regiszter végébe tesszük, akkor pl. $7/4=1$

A lebegőpontos ábrázolás kiküszöböli ezeket a hátrányokat

- A számokat hatványkitevős formában ábrázolja

$$\boxed{\pm M * r^{-k}}$$

M : mantissza

r : radix (számrendszer alapja)

k: karakterisztika

Története

- 1933 Konrad Zusen a Zuse 3-ban alkalmazta
- Neumann: ellenezte, mivel a számok maguk hosszúak, → memóriaigényesek, a számítás velük bonyolult → összetett vezérlőrészt igényelt → nem javasolta használatát

A hetvenes évek főbb vonulatai

- UAX
- Cray
- IBM 370/390

1985-ben az IEEE szabványosította.

Jellemzői

radix

- A megállapodás szerinti architektúrán belül állandó
- Tipikusan kettő
- IBM 370/390 esetén 16

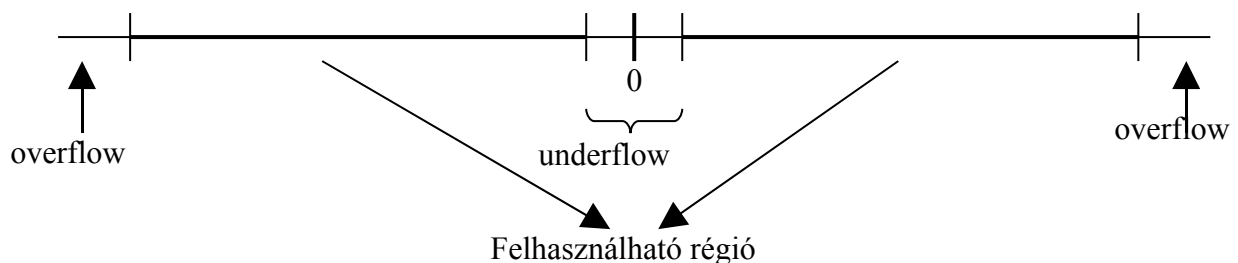
Nem normalizált

- $123,567 * 10^0 = 0,1234567 * 10^3 = 12345,67 * 10^{-1}$
ezt a formátumot az architektúrák tipikusan nem alkalmazták

A normalizált formátum

- képlettel $1/r \leq M < 1$
- szövegesen: a törtpontot az első értékes számjegy elé helyezzük
- pl.: $1/2 \leq M < 1$; $1/10 \leq M < 1$; + n fenn van tartva a végtelen jelölésére

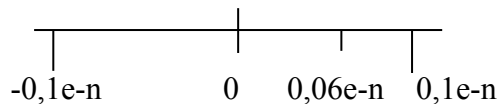
$$-0,999 * 10^{+n-1} \qquad -0,1 * 10^{-n} \qquad +0,1 * 10^{-n} \qquad +0,999 * 10^{+n-1}$$



over- underflow

- underflow esetén következmény
 - kijelzi
 - vagy nullát, vagy denormalizált számot ábrázol
- overflow
 - kijelzi
 - vagy a lehető legnagyobb számot, vagy előjeles végtelent ábrázol

denormalizált szám



a denormalizált számok ábrázolását a mai architektúrák többsége lehetővé teszi, a denormalizált flageit bebillentik

Értelmezési tartomány

A karakterisztika helyiértékeinek száma	Pl. Tízes számrendszer	Konklúzió
1	9^9 milliárd	Függ a karakterisztika
2	9^{99}	helyiértékeinek száma
1	Kettes számrendszer 2^1	Függ a számrendszer alapjától

Pontosság

= ha a regiszterünk mantissza része 3 helyiértékes, akkor 0,3456 mantisszát csak 0,345-ként tudjuk felírni. Pl. 10^6 karakterisztika esetén 600-zal torzul az eredmény.

= tehát a pontosság függ a mantissza helyiértékeinek számától

A 0 körüli számok

= amennyiben a mantissza értéke nulla, akkor az architektúrától elvárt, hogy a karakterisztika is nulla legyen

Rejtett bit

$$\frac{1}{2} \leq M < 1$$

0,111010

= mivel a kettedes pont utáni első értékbiteknek nincs információ tartalma, hanem kötelezően egyesnek kell lennie, ezért

- a memóriába vagy a háttértárra való kiírás előtt ezt a bitet balra léptetik, s jobbról be-
léptetünk egy értékes bitet \Rightarrow így módon egy helyiértékkal megnöveljük a mantissza
hosszát \Rightarrow növekszik a szám pontosság
- amikor egy lebegőpontos számot betöltünk az operatív tárból vagy a háttértárról, ak-
kor legelőször visszaállításra kerül a rejtett bit (biztosan „1”!), hiszen a számításnál
szükség van rá
 - ❖ a rejtett bitet már alkalmazta a Zuse is (1933)
 - ❖ a rejtett bitet minden mai piacon lévő processzor használja

Őrző bitek (a pontosságot „őrzik”)

= a processzoron belül, mind a lebegőpontos regiszterek mantissza része, mind a műveletvégző 4-6 bittel hosszabb, mint a tárolási formátum \Rightarrow ezek az őrző bitek (32+4, 32+6)

= az őrző bitek felhasználása:

- a rejtett bit helyén történő balra léptetésekor jobbról értékes bitet lehet beléptetni (helyreállítás)
- a tárolási formátum kérésekor kerekített értéket írhatunk ki \Rightarrow pontosabb (számolás)
- az eredmény normalizálásakor jobbról értékes bitet tudunk beléptetni (rejtés)

= minden mai piacon lévő architektúra alkalmazza az őrző biteket

mantissza kódolása

= a mantisszát az architektúrák mindig 2-es komplementes formában ábrázolják

karakterisztika kódolása

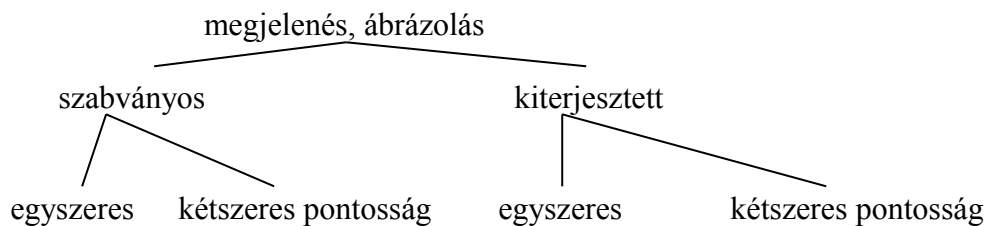
= a karakterisztika kódolása tipikusan többlétes kód

- a többlétes kód kialakítása gyorsabb, mint a 2-es komplementes
- mivel a karakterisztikával csak +, - és léptetés műveleteket kell végrehajtani – ez a többlétes kód esetében is elvégezhető – ezért alkalmazzák.

= minden ma piacon lévő architektúra esetén ezt a megoldást alkalmazzák

IEEE 754-es lebegőpontos szabvány

- 1977-ben kezdték a kidolgozását, 1985 -ben jelent meg
- célja: architektúrák között az adatszintű kompatibilitásnak megteremtése
- minden architektúrából összegyűjtötték a legjobb megoldásokat
- rendszerszinten gondolkoztak, azaz nem írták elő, hogy mit kell megvalósítani hardver- és mit szoftverúton.
- a szabvány fejezetei:
 - = formátumok
 - = műveletek
 - = kerekítések
 - = kivételek

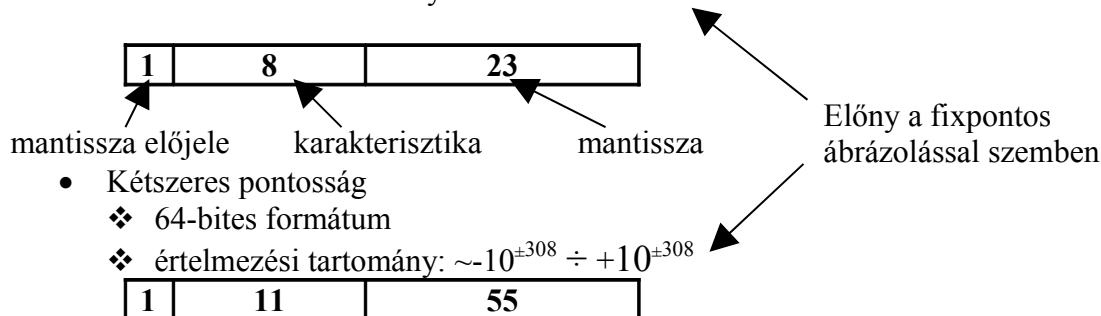


= szabványos - kiterjesztett:

- a **szabványos** az operatív tárban illetve a háttértáron használt **tárolási** formátum
- a **kiterjesztett** a processzoron belüli **feldolgozási** formátum
- a tárolási a rövidebb, a feldolgozási a hosszabb
- a szabványost szigorúbb szabályokkal rögzítették
- a kiterjesztett esetén maximális szabadságot biztosítottak a gyártóknak a CPU -n belül

= szabványos:

- Kizárólag az egyszeres pontosság a kötelező, a kétszeres pontosság opcionális
- Egyszeres pontosság:
 - ❖ Gyorsabb futás
 - ❖ Kisebb memóriaigény
 - ❖ Kevésbé pontos az eredmény
- Kétszeres pontosság:
 - ❖ Lassabb futás
 - ❖ Nagyobb memóriaigény
 - ❖ Pontosabb eredmény
- A kétféle formátum között a programozó dönt
- Egyszeres pontosság
 - ❖ 32-bites formátum
 - ❖ Értelmezési tartománya: $\sim -10^{\pm 38} \div +10^{\pm 38}$



= kiterjesztett formátum

= kizárólag a processzoron belül használják

= egyszeres pontosság \Rightarrow min 43 bit hosszú (32+11(őrzőbit))

= kétszeres pontosság \Rightarrow min 79 bit hosszú (64+15(őrzőbit))

- műveletek

- min. a 4 aritmetikai művelet

- maradékképzés

- négyzetgyökvonás

- bináris, decimális konverzió

= értelmezett a végtelennel való műveletvégzés is

$$\text{pl. } (3+(+\infty))=+\infty \quad (3+(-\infty))=-\infty$$

- kerekítések

= a legközelebbi felé történő kerekítés (mint a hagyományos kerekítés, pl. 83,4 \Rightarrow 83)

= a pozitív végtelen felé kerekítés

= a negatív végtelen felé kerekítés

- az utóbbi kettő az intervallumalgebrához kapcsolódik
- kétszer hajtják végre a számításokat:
 - ❖ egyszer a pozitív végtelen felé
 - ❖ egyszer a negatív végtelen felé
- az eredmény a két kapott eredmény között helyezkedik el
- amennyiben a két eredmény közti intervallum a számítási igényünk szempontjából túl nagy, akkor
 - ❖ elemezni kell a feladat megnevezését
 - ❖ illetve a számítási algoritmust

= levágja vagy trunc (pl. 83,9 \Rightarrow 83)

- kivételkezelés:

= nullával való osztás

= overflow

= underflow

= négyzetgyök negatív számból

= osztás

Esettanulmányok

Intel processzorcsalád:

= 1981-ben az Intel 8087-es processzorban valósul meg először, 1985-ben megjelent IEEE 754-es szabvány

- Logikai architektúra:

- = a radix 2
- = underflow esetén denormalizált számot ábrázol
- = overflow esetén előjeles végtelen ábrázolása
- = létezik a rejtett bit
- = léteznek az őrző bitek
- = a mantissza 2-es komplementum
- = karakterisztika többletes kódolású
- = a szabályos formátumok közül megvalósításra került, mind az egyszeres, mind a kétszeres pontosság
- = a kiterjesztett formátum 80 bit hosszú



- a programozó dönti el, hogy ebből a formátumból egyszeres vagy kétszeres formában írja ki

- Fizikai architektúra:

= az Intel 8087-es lebegőpontos segédprocesszor külön lapkán volt, mivel nem fért rá a 8088-as lapkára technikai okok miatt. Hasonló elvű 80287 és a 80387 is.
80486DX: a miniaturizálás eredményeképpen már közös lapkán helyezkedik el a lebegőpontos és az általános célú processzor.

= teljesítményjellemző:

Kétfajta teszt.

- Relatív mutató:

Processzor	Sebesség	Relatív MIPS
80386	25MHz	~17
80486	66MHz	~1700
Pentium	133MHz	~6000

- relatív MIPS: az eredeti 1981-ben megjelent IBM PC teljesítménye=1 (8086)
 - a 100-szoros növekedés annak köszönhető, hogy hardveres úton valósították meg a lebegőpontos műveleteket
 - a 3-szoros növekedés a futószalag bevezetésének eredménye
- „Checkit” lebegőpontos teszt (Windows alatt nem ajánlott a futtatása)

Mai megoldások:

- Pentium I: 2 db 5 fokozatos futószalag (egyszerű (v), és összetett (u) műveletekhez) +1 db 3 fokozatos a lebegőpontos számoláshoz. Az „u” készítette fel előre ennek a futószalagnak az adatokat.
- Pentium Pro: további szakosodás a lebegőpontos végrehajtó egységek terén: Összeadó/kivonó, szorzó, osztó.
- Power PC 604: 1 db szakosodott lebegőpontos futószalag.

Műveletek lebegőpontos számokkal

Összeadás

$$X=A+B$$

$$A=\pm m_A * r^{\pm k_A}$$

$$B=\pm m_B * r^{\pm k_B}$$

Példa

$$A=0,95 * 10^4 \rightarrow 0,95 * 10^4$$

$$B=0,90 * 10^3 \rightarrow \underline{0,09 * 10^4}$$

$$1,04 * 10^4 \rightarrow 0,104 * 10^5$$

Algoritmus

- A kitevőket megvizsgáljuk: csak azonos kitevőjű számok adhatók össze.
- Amennyiben a kitevők nem egyenlők, akkor
 - A kisebb kitevőjű szám mantisszájának törtponyját 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

$$X=A*B=(m_A)*(m_B)*r^{k_A+k_B}$$

Algoritmus

- A mantisszákat összeszorozzuk, karakterisztikákat összeadjuk

Osztás

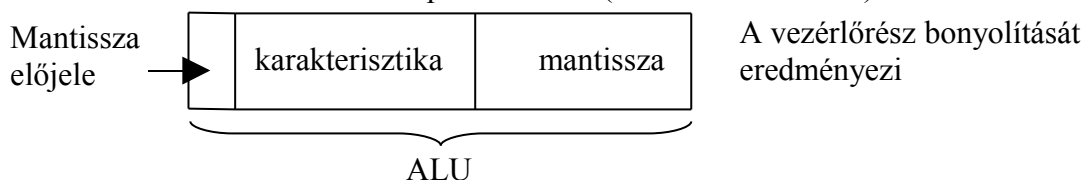
$$X=A/B=(m_A)/(m_B)*r^{k_A-k_B}$$

Algoritmus

- Mantisszákat elosztjuk, karakterisztikákat kivonjuk egymásból

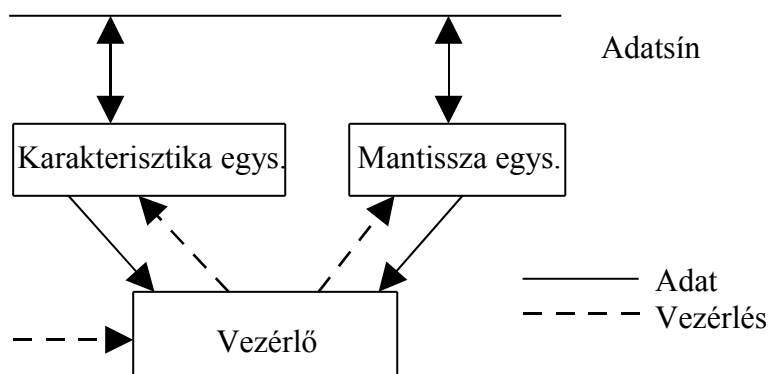
Konkrét megvalósítás

- Univerzális végrehajtó egység, műveletvégző
 - Általános célú ALU parciálásával (részekre bontásával)



- 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 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 végrehajtására tervezni).

BCD (Binary Coded Decimal)

$$1|5_D = 1111_B = 0001\ 0101_{BCD}$$

Megjelenésének oka:

- Fixpontos ábrázolással a törtszámok pontatlanok
- Lebegőpontos ábrázolással a mantissza-, karakterisztikaforma nem teljesen pontos, csak pontosabb.
- A kettes komplementes esetében: 10-es számrendszerű számokat átszámítjuk kettesbe, majd vissza...
- BCD esetében decimális számrendszerből átkódoljuk a számokat kettesbe és vissza. Kódolás = egyértelmű megfeleltetés → A BCD pontosabb ábrázolás
=> A számábrázolás pontosságának javítása

Jellemzői

- Ábrázolási forma

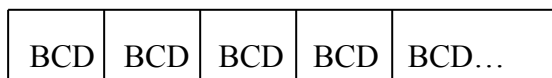
- Zónázott



Egy byte két részre oszlik.

Magas tetrád a zóna, kisebb tetrád a BCD. A zóna tipikusan nyomtatható karakterre egészíti ki a BCD számot. ASCII esetén tipikusan „3” és EBCDIC esetén tipikusan „F”. Általában nem lehet műveletet végezni a zónázott számokkal. (IBM, VAX), de az Intel kivétel, nála lehet!

- Pakolt



Pl.: Intelnél 10 db. bájttal hosszú, ahol

- Az első bájttal első bitje az előjel
- A bájttal többi bitje nem használatos
- A további kilenc bájttal tartalmazza a BCD számokat
- Az értelmezési tartomány: $-9.99 \cdot 10^{18} \div +9.99 \cdot 10^{18}$

$(10-1) \cdot 2 = 18$ BCD/szám

IBM, VAX: 31 BCD/szám

- Az előjel hossza
 - 4 bit – tipikusan a zónázott ábrázolásnál
 - 8 bit – tipikusan a pakolt ábrázolásnál
- Az előjel helye: a szám elején (nyomtatásnál) vagy a végén (számolásakor)
- Az előjel értéke
 - Érvénytelen tetrád
 - A, C, E, F a pozitív (de pl IBMnél F: előjellel nem rendelkező számok)
 - B, D a negatív
 - A + és a – az ASCII kódja!
- A szám hossza:
 - fix vagy változó (utóbbi esetben jelölni kell a hosszát is egy külön számban)

Összeadás BCD számokkal

Ugyanúgy adjuk össze a BCD számokat is, mint a binárisakat, csak

- fel kell ismernünk az érvénytelen tetrádokat és
- ezeknél korrekciót kell végrehajtani.

A, Az érvénytelen tetrádok felismerése

A BCD számok

Hexa	BCD kód
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Érvényes tetrádok

Érvénytelen tetrádok

Felismerésünk:

- az első bithelyiértéken 1-es áll ÉS

- a második VAGY a harmadik bithelyiértéken is 1-es áll.

B, Korrekció:

az érvénytelen tetrádok esetén

= kivonunk belőle 10_D -et és

= lépünk egy 10-es átvitelt

pl.:

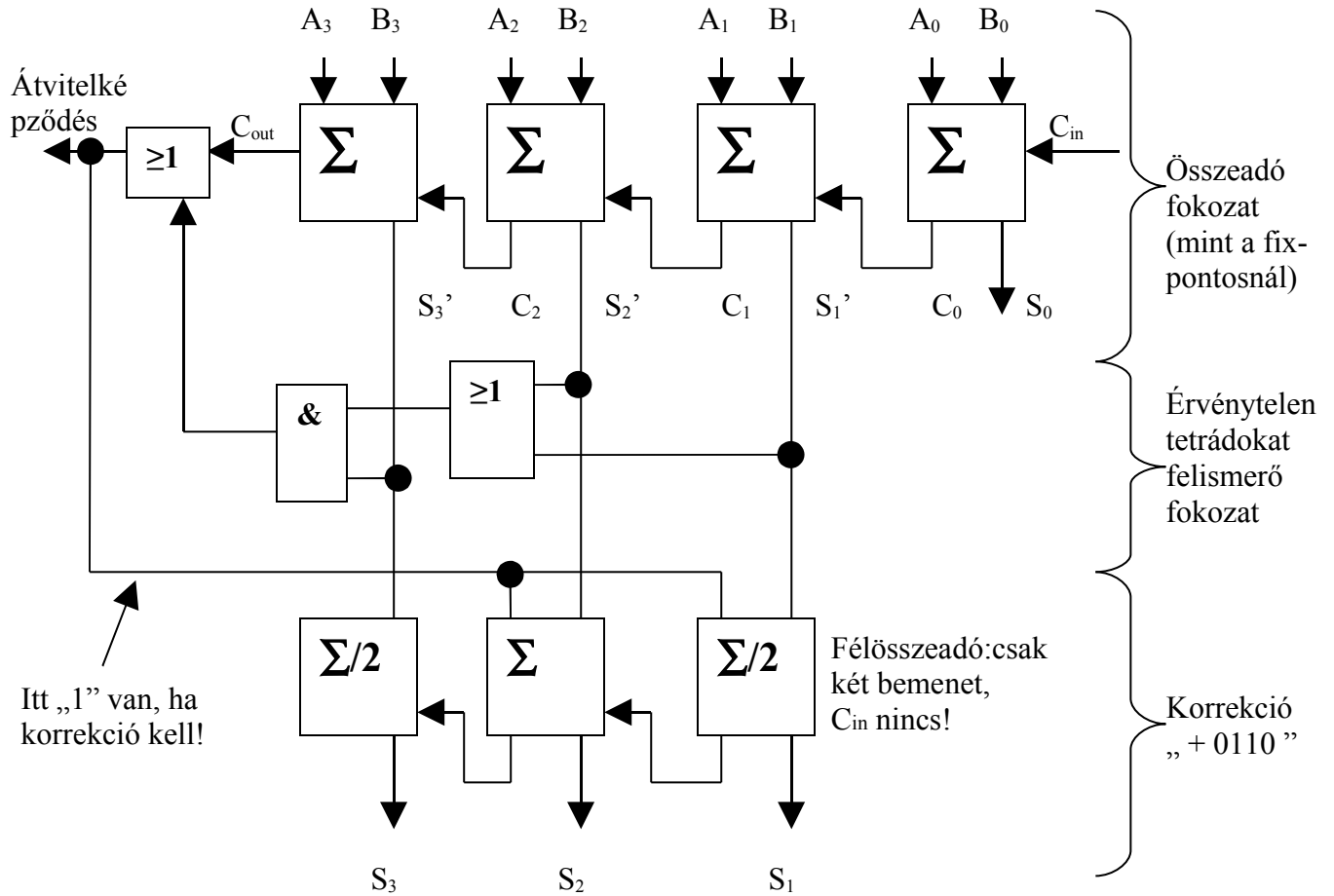
$$\begin{array}{r}
 A \quad 8 \quad 1000 \\
 +B \quad +7 \quad 0111 \\
 \hline
 X \quad 15 \quad 1111 \\
 +(-10) \quad 0110 \\
 \hline
 5 \quad 1|0101
 \end{array}$$

10-es átvitel

5_D

Segédszámítás:

$$10_D = 1010_B = 0110_{\text{kettes komplement}}$$

Megvalósítás:

2 dolog lehet átvitel oka: vagy MSB -ről átvitel, vagy érvénytelen tetrád.

Σ: egybites teljes összeadó (3 bemenet; van C_{in} is)

A BCD műveletvégzők megvalósítása

- univerzális műveletvégzőben, a BCD -nek megfelelő vezérléssel
- dedikált: külön BCD műveletvégző

A BCD jelentősége:

- előnye: teljesen pontos
- hátránya:

= bonyolultabb a műveletvégzés, több félvezetőt igényel, tovább tart

= a BCD számok átlagosan 40% -al hosszabbak, mint a binárisak (érvénytelen tetrád)

Pl.: $12_D = 1100_B = 0001|0010_{BCD}$

Pl: 9+9

```

  1001
+ 1001
-----
 1 0010
   0110
-----
 1 1000   = 18_D

```

Esettanulmány: Intel

Zónázott és pakolt ábrázolást is támogatja, a műveleteket a zónázott formában is el lehet végezni. Pakolt formátum: 80 bit hosszú; 1 bájt előjel, 9 bájt BCD szám. Ért. tart.: $-9^{18} \div +9^{18}$

A fixpontos, a lebegőpontos és a BCD számábrázolás összehasonlítása

Fixpontos (minimum 8, 16 bit):

- + igen gyors a fixpontos műveletvégzés (ciklusváltozónak ideális)
- + a memória igény kisebb, mert többféle formátumot alkalmaz: 8, 16, 32, 64, 128 bites.
A formátumok közül mindig a számunkra szükséges értelmezési tartománynak megfelelően válasszuk
- + egész számok esetén teljesen pontos az ábrázolás, pl. if $a=1$
csak fixpontos és BCD ábrázolás esetén adható ki, lebegőpontosnál nem
- törtnél teljesen pontatlan ($7/4=1$)

Lebegőpontos (minimum 32 bit):

- + akkor alkalmazzuk, amikor
 - = a számunk értelmezési tartománya túllép a fixpontos által biztosított, ill.
 - = törtszámok ábrázolási igénye esetén
- + a lebegőpontos ábrázolás a gyakorlatban kielégítő pontosságot biztosít
- csak indokolt esetben alkalmazzuk a kétszeres pontosságot, mert lassúbb (erőforrás igényes)

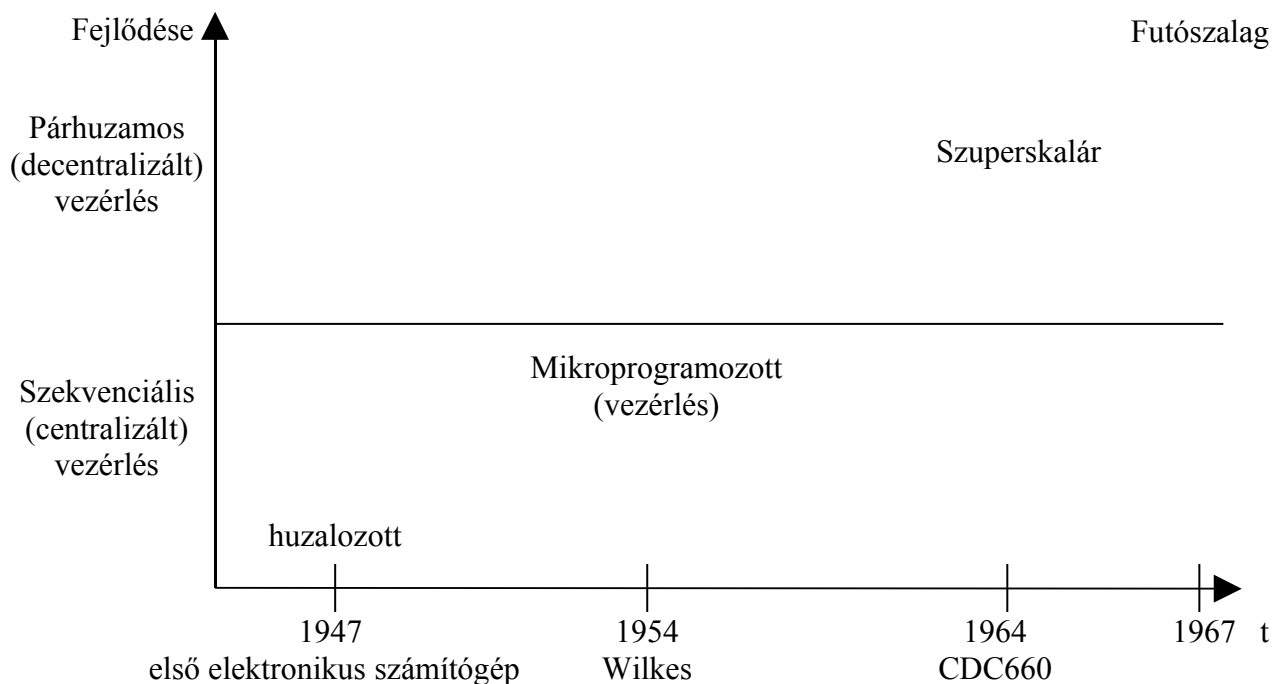
BCD:

- + a lebegőpontos ábrázolás egy alternatívája a pontosabb BCD ábrázolás
- + kifejezetten banki alkalmazásoknál használatos

Az ALU egyéb műveletei

- mind a 16, kétoperandusos logikai műveletre képes (Pl.: OR, AND, XOR, NOT)
- léptetés, invertálás, load/store, összehasonlítás
- címszámítás (végrehajtási állapotban):
 - = korai gépekben az általános célú műveletvégző végezte
 - = napjainkban tipikusan célhardver végzi
- a karakteres műveletek tipikusan az általános célú műveletvégzőben történnek

2. Vezérlőrész



- a decentralizált vezérlés a huzalozott ill. mikroprogramozott vezérlési elemekből épül fel

Huzalozott vagy áramköri vezérlés

Tervezése:

- igazságtábla tervezése
- logikai függvények felírása
- azonos átalakítások a következő célfüggvénnyel:
 - = az az áramköri elemek számának minimalizálása (a lapkaméret korlátozott volta)
 - = a végrehajtási idő minimalizálása
- megvalósítás

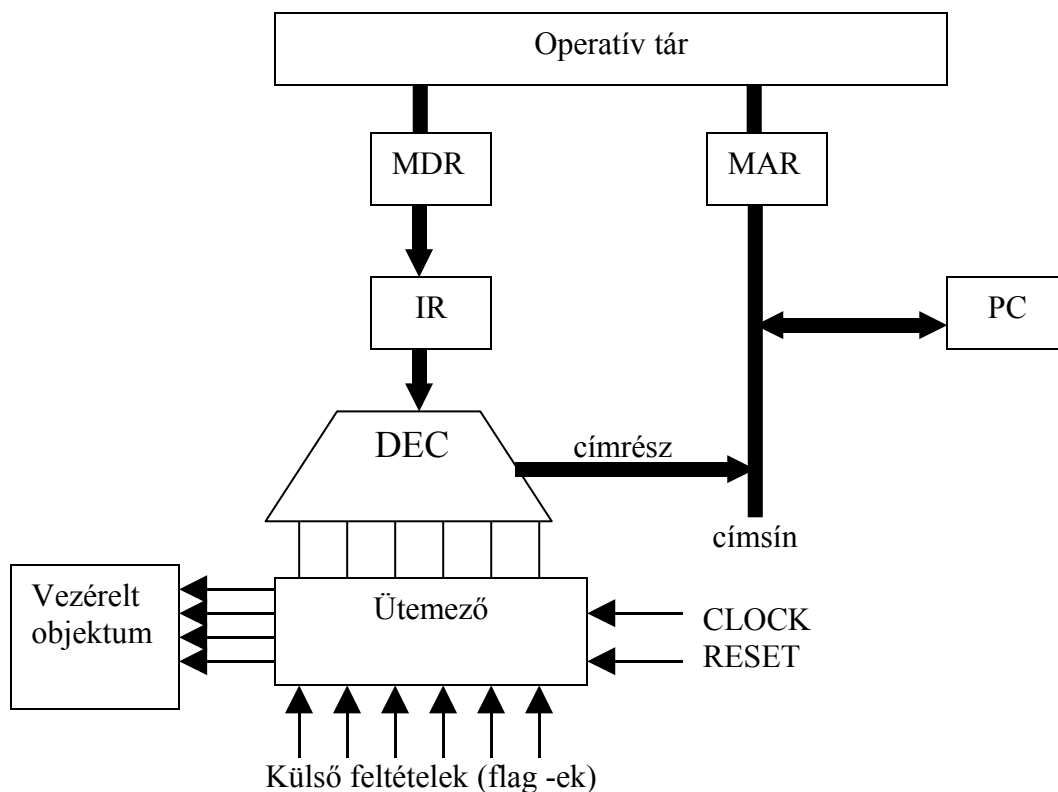
Hátrányai:

- ember számára nehezen áttekinthető
- merev, nehezen módosítható

Előnyei:

- igen gyors

A vezérlés megvalósítása



Elve:

Egy forrásregiszter tartalmát a módosító áramkörön keresztül egy célregiszterbe vezetjük.

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)

Módosító áramkörök:

- inkrementálás
- léptetés
- invertálás
- összeadás
- komparálás
- stb.

Működése:

- 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 kimenő kapuját és a célregiszter bemenő kapuját megnyitja
- az eredmény átmásolódik

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

Mikroprogramozott vezérlés

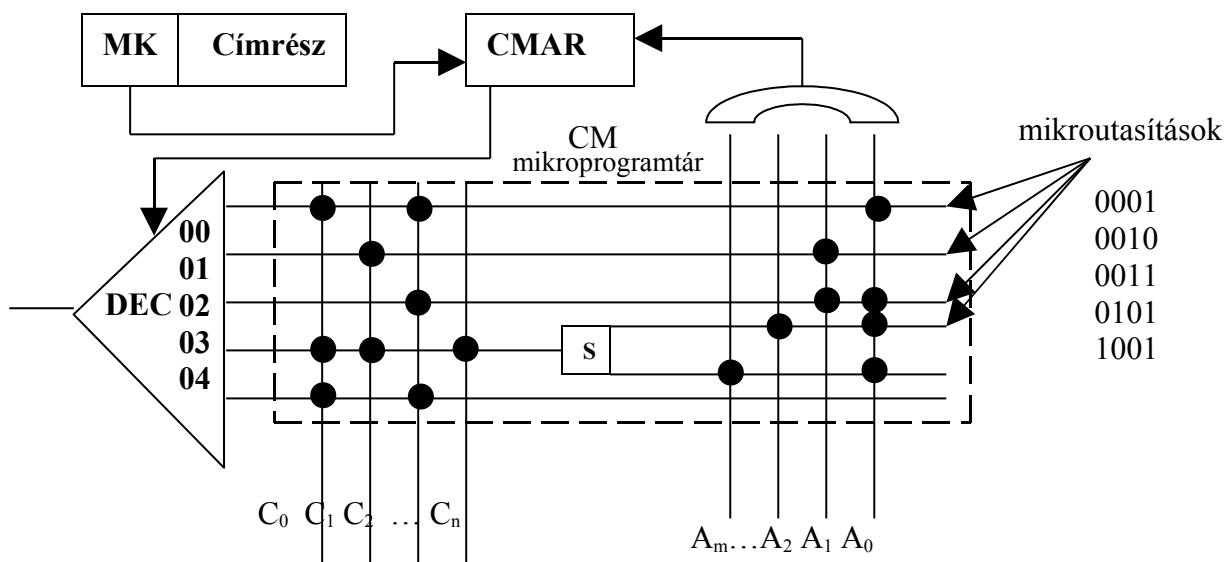
1954 Maurice Wilkes (University of Cambridge)

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 vezérlő rész rugalmassá, könnyen módosíthatóvá alakítása
 - = a mikroprogram a mikroprogramtárban helyezkedik el, így az módosítható

A Wilkes-féle modell

Pl. ADD (ennek nincs köze a Neumann -féle ADD -hoz)



Működése:

A. Mikroutasítás szekvencia:

1. A Neumann -féle program utasítás műveleti kód része - megfelelően kódolva - bemásolásra kerül a CMAR-ba. Megfelelő kódolás: a CMAR-ba az adott gépi kódú utasítás végrehajtását elemi műveleti szinten vezérlő mikroprogram kezdő címe kerül.
2. Ez a CMAR-ból a cím a DEC-be kerül, amely az adott című mikroutasítást érvényes állapotba helyezi.
3. Az érvényesített mikroutasítás végrehajtása során aktiválja a vezérlőmátrixban a kijelölt vezérlővonalakat, ezt bizonyos ideig kitarítja
4. Az adott mikroutasítás címrészében lévő következő végrehajtandó mikroutasítás címét beírja a CMAR-ba. Vissza a 2. pontra

Mikroutasítás például: $MDR \leftarrow (MAR)$ vagy $PC \leftarrow PC + 1$

B. Külső feltételes mikroutasítás végrehajtása („S”)

- amennyiben feltételes ugrás mikroutasítás végrehajtása következik, akkor:
 - = aktiválásra kerülnek a mikroutasítás által kijelölt vezérlőútvonalak, ezek egy bizonyos ideig kitartódnak, majd
 - = az adott mikroutasítás címrészében lévő kettő db. cím közül a feltétel igaz vagy hamis voltától függően vagy az egyik vagy a másik cím kerül a CMAR -ba letöltésre

Pl:

Az ADD utasítás végrehajtása

$DEC \leftarrow IR$

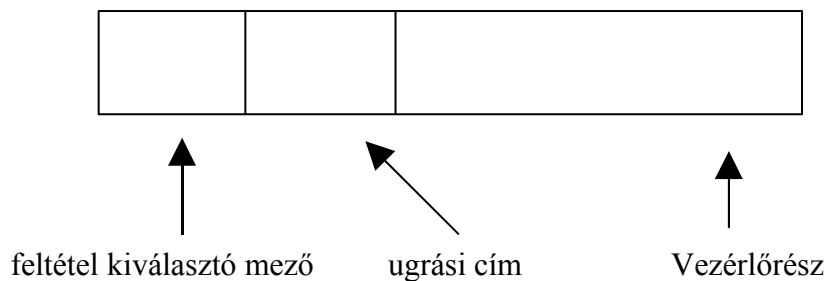
$MAR \leftarrow DEC \text{ címrész}$

$MDR \leftarrow (MAR)$

$AC \leftarrow AC + MDR$

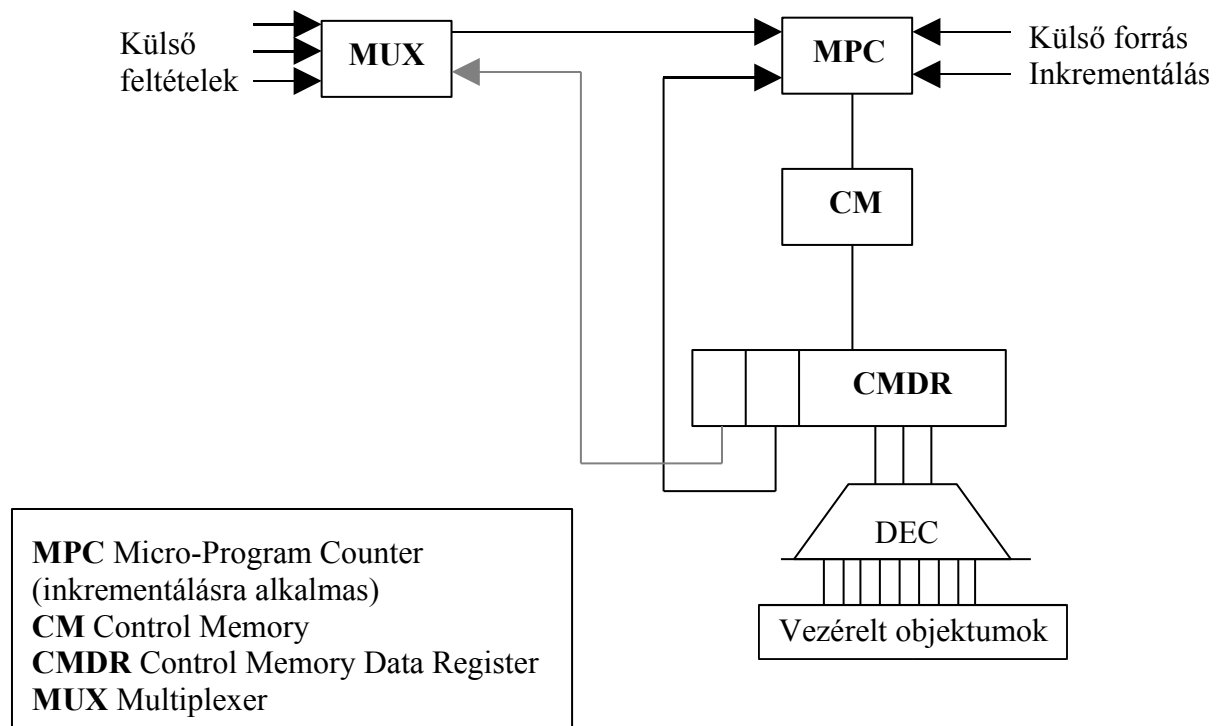
} Mikroutasítások

Egy korszerű mikro-utasítás felépítése (formátuma):



Feltétel kiválasztó mező: megmutatja, hogy a tesztelendő feltételek közül melyiket választjuk ki
Ugrási cím: ugró mikroutasítás esetén, amikor a feltétel igaz erre a címre adódik át a vezérlés
Vezérlőrész: kijelöli melyik vezérlővonalakat kell aktiválni

Egy korszerű mikro-vezérlő megvalósítása



Működése:**A. Mikro-utasítás szekvencia**

1. az MPC-ben lévő következő végrehajtandó mikroutasítás címe által meghatározott mikroutasítás a CM-ből eljut a CMDR-be MPC → CM → CMDR
2. a CMDR-ben lévő mikroutasítás vezérlő része aktiválja a kijelölt vezérlővonalakat, és azt meghatározott ideig kitartja
3. az MPC tartalma inkrementálódik és vissza az első pontra

B. feltételes ugrási mikroutasítás

- amennyiben a CMDR-ben feltételes ugrási mikro-utasítás van, akkor
 = a vezérlő része aktiválja a kijelölt vezérlővonalakat, és azt meghatározott ideig kitartja
 = a feltétel kiválasztó mező által meghatározott külső feltétel tesztelésre kerül, és annak igaz vagy hamis voltától függően

- vagy a mikroutasításban tárolt címmel felülíródik az MPC tartalma
- vagy pedig az MPC tartalma inkrementálódik

Megjegyzések:

- míg a Neumann-féle „makroszámítógép” operatív tárában együtt tároljuk az adatokat és az utasításokat, s ezért szükség van külön PC-re, MAR-re
 - addig a mikroprogramozott vezérlő CM-jében csak mikroprogramozott utasítások vannak, ezért

- * amennyiben nincs szükség, lehetőség inkrementálásra, akkor CMAR-t használunk,
- * amennyiben szükséges az inkrementálás, akkor MPC-t.

A mikro-utasítás hosszát meghatározó tényezők

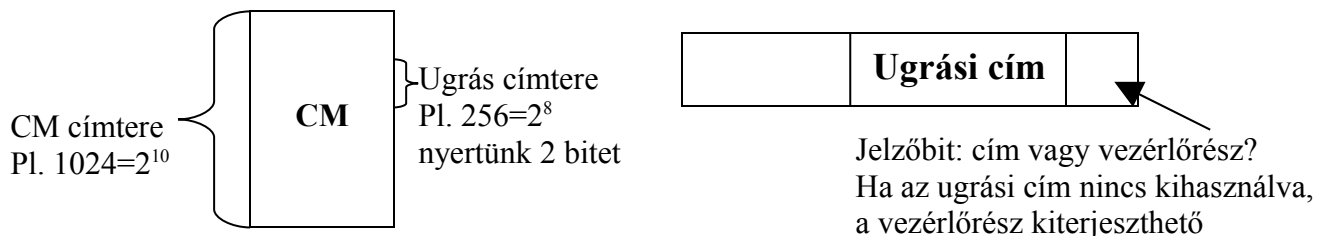
a feltétel kiválasztó mező rövid

I. Címrész:

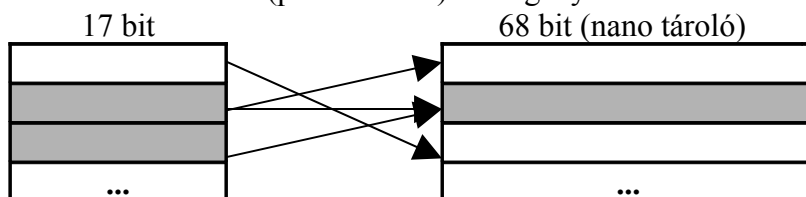
- A, a következő végrehajtandó utasítás címe
 - maga a mikroutasítás tartalmazza (Wilkes féle modell)
 - MPC alkalmazásával határozzuk meg (korszerű mikrovezérlő)

B, Ugrási cím

- kisebb az ugrási címtér, mint a CM címtere
 - az ugrási cím ritkán van kihasználva, utasítás szekvenciák esetén hasznosítható vezérlő részként
 - külső forrásból töltjük fel (puffert rendelünk hozzá)



- kétszintű mikroutasítás (pl. Motorola) - időigényes

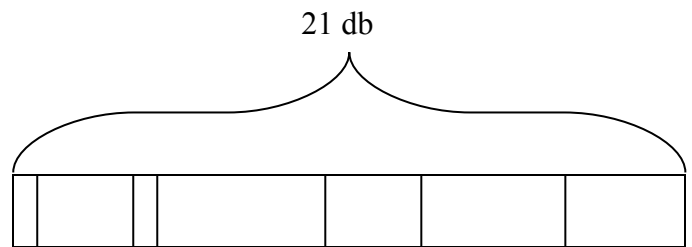
**Megtakarítás:**

Több mutató is mutathat ugyanarra a hosszú vezérlőrészre (ezt ugyan csak egyszer tároljuk)

II. vezérlőrész

= horizontális mikroutasítás

- hosszú mikroutasítások
- magas szintű párhuzamosság
- csekély mértékű kódolás
- pl. IBM 360, Motorola;
- Esettanulmány
 - IBM 360
 - 90 bit
 - 21 vezérlőmező



azaz 21 db, hardverileg egymástól független egységet kell vezérelni ⇒ maximum 21-szeres párhuzamosság érhető el

- 65.-67. bithelyiértékek vezérlik az ALU jobboldali bemenetét: azaz mely regisztereket kell rákapuzni
- 68.-71. bithelyiértékek írják elő az ALU számára, hogy milyen műveletet hajtson végre. A lehetőségek: bináris vagy BCD összeadás a bejövő és a kimenő átvitel különféle kezelésével.

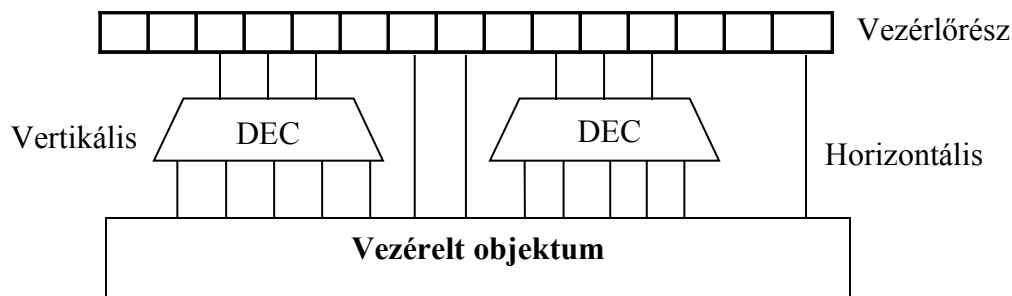
= vertikális mikroutasítás

- rövidek
- erős kódolás
- csekély mértékű párhuzamosság
- Pl. IBM 370, PDP 8, Intel processzorok
- PDP 8:
 - a vezérlőrész 128 bit hosszú
 - hipotézis: $2^7=128$, azaz 7 biten kódoljuk le a 128 féle értéket. Ez az elve a vertikális mikroutasításnak.
- Esettanulmány: IBM 370
 - a mikroutasítás a gépi kódú utasításra emlékeztet: MK+címrész

Opcod	op1 címe	op2 címe	CM, címképzési információ
-------	----------	----------	---------------------------

- négy bájtnál hosszabb
- a címek tipikusan processzor-regisztereket címeznek
- a címképzési információ emlékeztet a Wilkes -féle modellre

= napjaink gyakorlata (a kettő között)



- a gyakorlatban szükséges vezérlések horizontálisak, a ritkábban használtak tömörített formában helyezkednek el a vezérlőrészben, amit a dekóder old fel. Így kompenzálják a dekódolás miatt vesztett időt.

A mikroprogramok időzítése

A mikroutasítások fajtái:

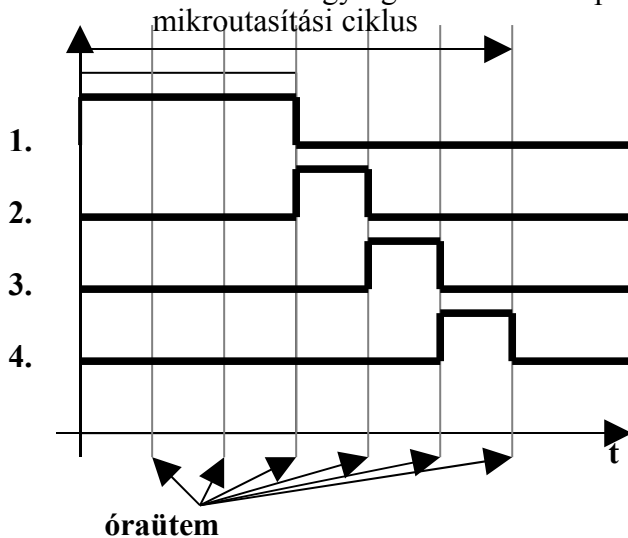
- monofázisú mikroutasítások: a mikroutasítás végrehajtási ciklusa megegyezik az óraciklussal
- polifázisú mikroutasítás: egy mikroutasítás több óraciklus alatt hajtódik végre \Rightarrow egy mikroutasításban egy egész elemi műveleti szekvenciasort írhatunk elő

= pl. $R_1 \leftarrow f(R_0)$ (R_1, R_0 : regiszterek)

- az elemi műveleti szekvencia

$R_0 \rightarrow f \rightarrow R_1$

1. a mikroutasítás lehívása a CM-ből
2. az R_0 tartalmának rákapuzása az „f” módosító áramkör bemenetére
3. előírjuk az „f” által végrehajtandó módosítást
4. az f módosító egység kimenetét rákapuzzuk az R_1 regiszter bemenetére



Következtetés: mivel a huzalozott vezérlésnél nincs értelmezve a mikroutasítás lehívása, ezért a huzalozott vezérlés mindig gyorsabb, mint a mikroprogramozott

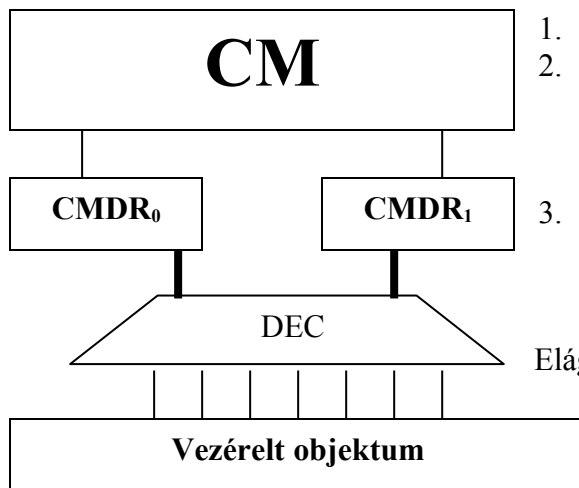


: huzalozott gép esetén nincs!

Előny: mikroutasítások megtakarítása \rightarrow mikroutasításban egész elemi művelet van definiálva
Hátrány:

A mikroprogramozott vezérlés gyorsítása

Amennyiben a mikroprogramozott vezérlés mellett döntöttünk, a lehető leggyorsabb vezérlés érdekében horizontális mikroutasítást és igen gyors CM-t kell alkalmazni (gyorsítani nehéz)
- szervezési gyorsítási lehetőségek: prefetch azaz elő lehívás alkalmazása.



1. lehívjuk a $CMDR_0$ -ba a következő utasítást
2. Amíg a vezérlés a $CMDR_0$ -ban lévő mikroutasítás alapján történik, azzal párhuzamosan lehívjuk a következő végrehajtandó mikroutasítást a $CMDR_1$ -be
3. A vezérlés most a $CMDR_1$ -ből fog történni, s ezzel párhuzamosan lehívjuk a következő végrehajtandó mikroutasítást a $CMDR_0$ -ba

Elágazási utasítás esetén teljesítményvesztés lép fel!

Mikroprogramozás:

- a mikroprogramozás igen mély hardverismeretet igényel. Ezt a szintet a gyártók tipikusan nem publikálják.
- a mikroprogram utasításai numerikus kódok, az assembly szintű nyelvekre emlékeztet
- létezik mikroassembler, azaz fordítóprogram, mely a forrás nyelvű mikroprogramot végrehajtható formátumra fordítja ⇒ az utóbbi tölthető be a CM -ba
- a mikroprogram hiba következményei:
pl a hardverműködés képtelenné válik (hibás reset állapot áll vissza)

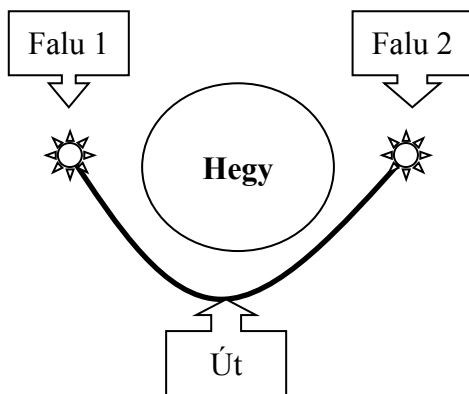
	Áramkörü (huzalozott)	Mikroprogramozott
Áttekinthetőség	Ember számára nem áttekinthető	Ember számára áttekinthető
Sebesség	Mindig gyorsabb	Mindig lassabb
Módosíthatóság	Merev, nehézkesen módosítható	Mikroprogramcsere lévén cserélhető

Sínrendszer

(Buszrendszer)

Bevezetés:

- a sínrendszer az egységek közötti kommunikációra szolgál (CPU, RAM, perifériák)
- maga a sínrendszer a kommunikáció infrastrukturális része, később önálló I/O rendszerként tárgyalni fogjuk a perifériáknak a processzorral, ill. a memóriával való kapcsolatának sajátosságait
- a sínrendszer egy történelmi fejlődés eredménye ⇒ ez bizonyult a legjobbnak
- az egységek egymással kizárólag a sínrendszeren keresztül kommunikálnak, mégpedig szervezett, egységes módon
- pl.



A sínrendszer a felhasználó számára transzparens, a következőkben felsorolt jellemzők közül a tervezők választhatnak csak

Kommunikáció fajtái:

- Egységen belüli kommunikáció (pl. a processzoron belül)

= kicsik a távolságok

- Közös vezérlés
- Közös órajel



viszonylag egyszerű megvalósítás

Belső sínrendszer
(processzoron belül)

- Egységek közötti kommunikáció (pl. a processzor és a perifériák között)

= viszonylag nagyok a távolságok, jelentős késleltetéssel jár →
nem célszerű a központi óraütemadó alkalmazása.

Szabványosítása:

- a processzor-memória közötti sínen szintén lehetetlen és értelmetlen
- a perifériák esetében már jelenhetnek meg szabványok (pl. PCI)

= jelentősek az optimális sebességek közti különbségek az egyes egységek vonatkozásában (pl. más az optimális sebessége a processzornak és más a billentyűzetnek)

= Eltérő architektúrák (szabványok)

Külső sr.
(alaplap)

- Távolsági kommunikáció

= számítógép hálózatok

A külső sínrendszer fogalma:

- fizikai: olyan vezetékköteg, melynek minden egyes erén egyidejűleg

= vagy csak a logikai nullának megfelelő $\sim 0V$

= vagy a logikai egyesnek megfelelő kb. 12V; 5V; 3,3V; 2,8V, vagy változó feszültség jelenhet meg (felhasznált programtól függhet, pl. gépelés vagy filmnézés)

- funkcionális: olyan vezetékköteg, mely biztosítja, hogy a forrásból a célba egyidejűleg, azaz párhuzamosan n (16, 32, 64) db. bit juthasson el. Ebben a kontextusban (értelmi összefüggésben) 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: $r_0 \leftarrow \text{databus}$
 $r_1 \leftarrow \text{databus}$

Fajtája

- Az adatátvitel iránya szerint:

= simplex: egyirányú

= félduplex: időben csak egyirányú (vagy az egyik vagy a másik irányba történhet az átvitel egyidejűleg)

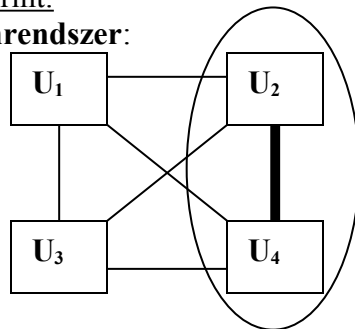
= duplex: egy időben kétirányú forgalom (gyakorlati megvalósítása: 2 vezetéken)

- pl. egyirányú: órajel, reset, címvonal

kétirányú: adatsín

- az átvitel jellege szerint:

= **dedikált sínrendszer:**



• Jellemzői:

- ❖ Minden egységet minden egységgel összekapcsoljuk:
 - egyirányú kapcsolat esetén: $n \cdot (n-1)$ vonal
 - kétirányú kapcsolat esetén: $n \cdot (n-1)/2$ vonal

• Előnyei:

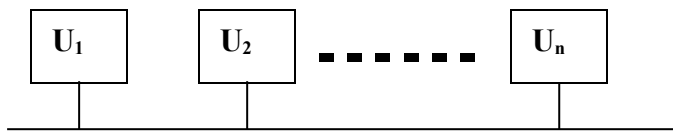
- ❖ Igen gyors: minden egység minden egységgel kommunikálhat
- ❖ Megbízható: architektúrális feltétele következtében pl. az U_2 és U_4 közötti kapcsolat megszakadása esetén a kommunikáció történhet az U_1 -en vagy U_3 -on keresztül is

• Hátrányai

- ❖ Drága: a vezetékek biztosítása
- ❖ Új egységek kialakítása bonyolult, mert ezek csatlakoztatása egyedi
- ❖ A miniatürizálás miatt sok csatlakozó láb kialakítása technológiai nehézségekbe ütközik (notebook)

= megosztott (shared) sínrendszer

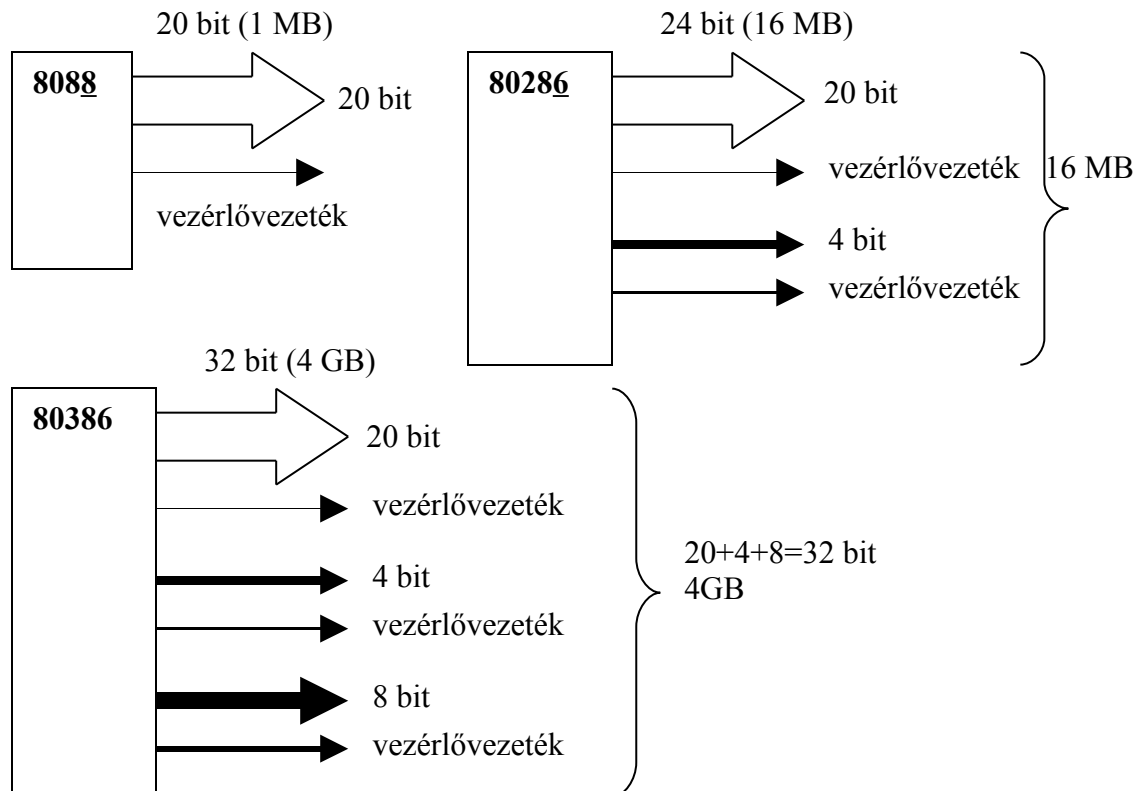
3. generációs számítógépek újdonsága. 1 vagy több vezérlő is lehet, sínvezérlő utasítások léteznek az ütközések ellen



- Jellemzői:
 - ❖ Minden egység a közös sínen keresztül kommunikál → egyidejűleg csak egy adó lehet
- Előnyei:
 - ❖ Viszonylag olcsó
 - ❖ Szabványos kialakítása lévén új egységek csatlakoztatása egyszerű
- Hátrányai:
 - ❖ Mivel egyidejűleg csak egyetlen adó használhatja a közös sínt, ezért viszonylag lassú
 - ❖ A közös sínhasználat vezérlése bonyolult, nem olcsó
 - ❖ Érzékenyebbé válik a rendszer a közös sín meghibásodására

- Átvitt tartalom alapján:

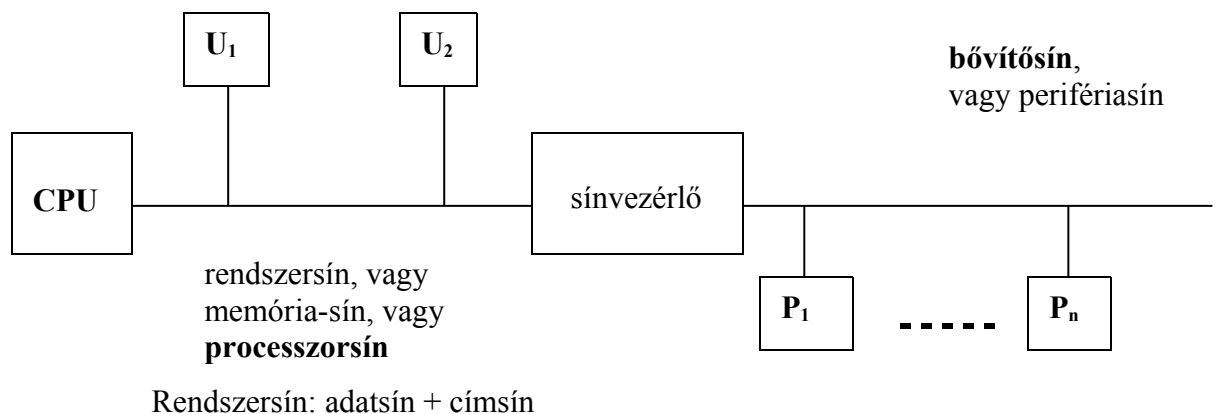
- Címsín
 - ❖ Feladata: az egységek (pl. hálózati kártya) valamint az egységek egy részének (pl. memóriacím) azonosítása
 - ❖ Fejlődése Intel processzorok esetén:



- a fokozatos bővítés nem eredményez tiszta tervet

- adatsín:
 - ❖ feladata: adatok továbbítása
 - ❖ fejlődése:
 - 8086: 8 vezeték (8 bit)
 - 80286: 16 vezeték
 - 80386: 32 vezeték
 - a fokozatos fejlesztés nem eredményezett tiszta tervet (lásd címsín)
 - ❖ elfogadott megoldás, hogy ugyanazon vezetékeken keresztül továbbítják a címeket és az adatokat:
 - ezzel egyrészt vezetékeket takarítanak meg
 - a csatlakozó lábak számát tudják csökkenteni
 - pl. PCI
 - = időbeli multiplexelés, azaz ugyanazon vezetékeken átviszi a blokk kezdőcímét, majd ciklikusan az adatokat, közben inkrementálással állapítva meg a címet
 - ez a megoldás előnyös blokkos átvitelnél
 - szükséges egy olyan vezérlővonal, mely megmutatja, hogy a közös vezetéken adat vagy cím van
- vezérlő vezetékek (nem sín! nem rakják össze sínne):
 - ❖ feladata: a vezérlési feladatok továbbítása
 - ❖ számuk tipikusan: 10-15
 - ❖ fajtái:
 - adatátvitellel kapcsolatos vezérlővezetékek
 - M/I/O** a címvezetéken memória-cím vagy I/O cím található
 - R/W** read/write: a processzor felől nézve az átvitel iránya
 - B/W** byte/word: párhuzamosan átvitt bitek száma
 - AS** address strobe: a cím a címsínre lett helyezve
 - DS** data strobe: az adat az adatsínre lett helyezve
 - A/D** address/data: a közös sínen most éppen cím vagy adat van
 - RDY** ready – kész
 - megszakítással kapcsolatos vezérlővezetékek:
 - megszakítás kérése és megszakítás engedélyezése/tiltása
 - sínfoglalással kapcsolatos vezérlővezetékek:
 - sínfoglalás kérése, jelzése és engedélyezése
 - egyéb
 - CLCK – órajel
 - Reset – reset-jel

- az összekapcsolt területek alapján:



- Processzorsín
 - ❖ feladata: a processzor-memória és a sínvezérlő forgalmának biztosítása
 - ❖ jellemzői:
 - az átviteli sebessége napjainkban már sokszorososa a bővítő-sínénél
 - processzor közeli lévén a teljesítmény-növelés érdekében célszerű figyelembe venni az architektúráis sajátosságokat: nem szabványosították sikerrel
 - ❖ elnevezések:
 - rendszersín: a rendszer-forgalmat továbbítja (cím, adat, vezérlősín)
 - memória-sín: a memória-elemeket köti össze
 - processzorsín napjainkban:
 - = az adatok blokkos átvitel esetén a DMA vezérlésével közvetlenül mennek a winchester és a memória között
 - = a processzor a másodlagos gyorsítótárral kommunikál
- bővítő-sín
 - ❖ feladata: a perifériák csatlakoztatása a processzor-memória kettőshöz
 - ❖ Fejlődése:
 - kezdetben összekábeleztek az egységeket (dedikált kapcsolatok)
 - DEC első gépei már sín-orientáltak voltak: csatlakozó helyeket alakítottak ki tesztelő készülékek csatlakoztatására. Szerzői joggal védték le.
 - 1976: az Altair tervezője kialakította az S-100-as bővítő-sín architektúrát, csatlakozóhelyenként 100 db érintkezővel. Ezt az IEEE szabványként fogadta el
 - 1981: az IBM PC megjelenése.

A nyitott sínrendszer az IBM PC példáján

Technológiai nyitott architektúra:

- ❖ Szabványosított aljzatok
 - = meghatározták az egyes érintkezők funkcióját (pl. tápellátás, vezérlés, adat, cím)
 - = sokféle egység csatlakoztatására volt lehetőség

Szerzőjogilag nyitott architektúra

- ❖ az IBM azonnal publikálta a csatlakozás műszaki jellemzőit

Következmények:

- ❖ a gyártók sora kezdte el a fejlesztést, hiszen igen nagy piaci szegmensben dolgozhattak
- ❖ a felhasználókat gyártók hada „kényeztette” a legkülönbözőbb megoldásokkal

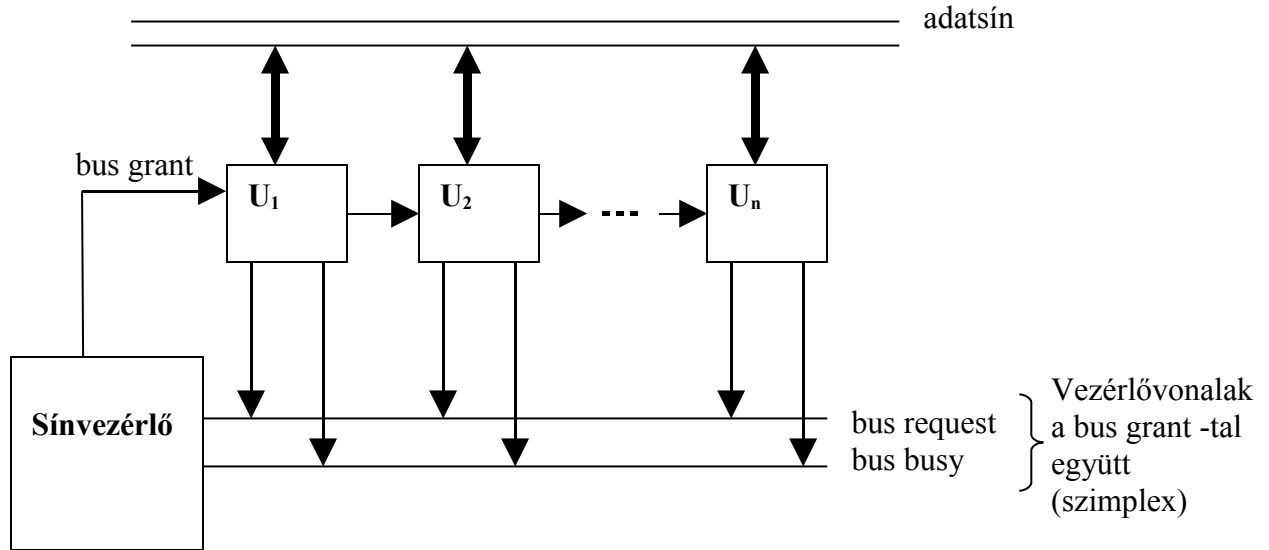
A sínrendszer működése

2 fázisból áll (megosztott slave sínrendszerek esetén):

- A. sínfoglalás (Bus arbitration)
- B. átvitel (Bus timing)

A. Sínfoglalás**Soros sínfoglalás:**

- hardver pulling (daisy chain – gyermekláncfü) → hardveres lekérdezéses = megvalósítása

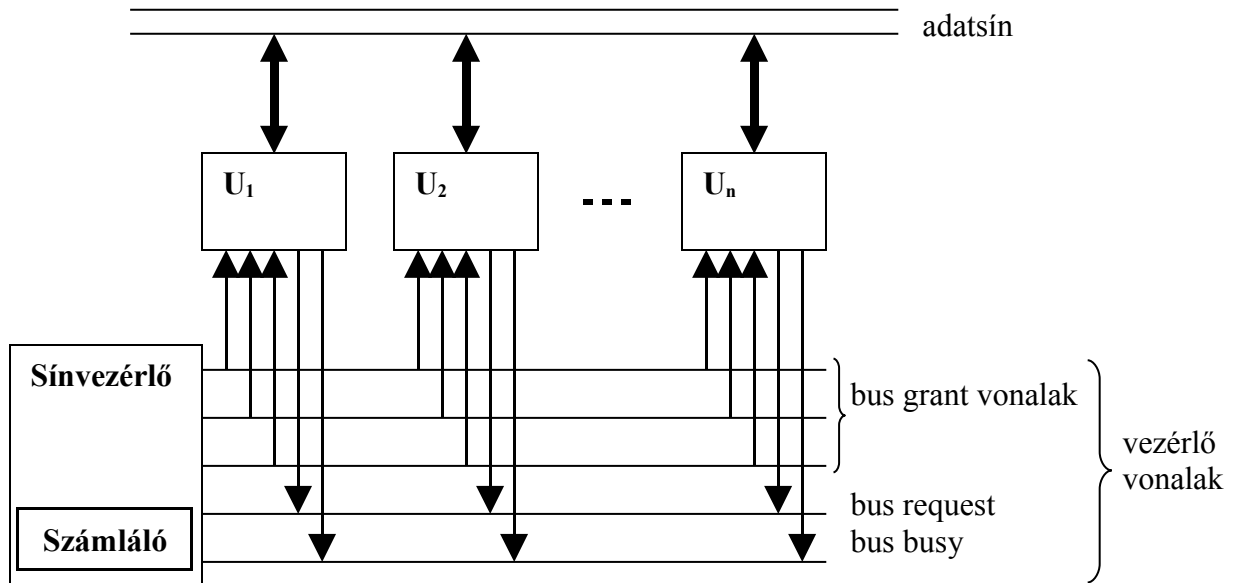


= értékelése:

- * 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 pulling
= megvalósítása

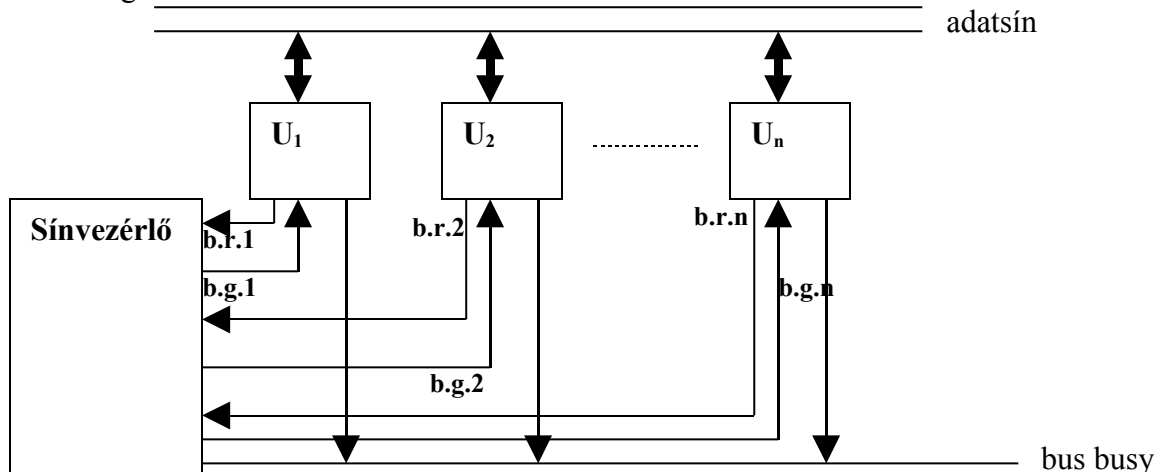


A számláló algoritmus alapján szólítja meg az egységeket (pl. ott folytatja, ahol abbahagyta)
= értékelése:

- * 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 $2^3 = 8$ db egység

Párhuzamos sínfoglalás

- megvalósítás:



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

B. Adatátvitel

(Bus timing)

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 előállítás, 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)

A bővítő sínek csoportosítása

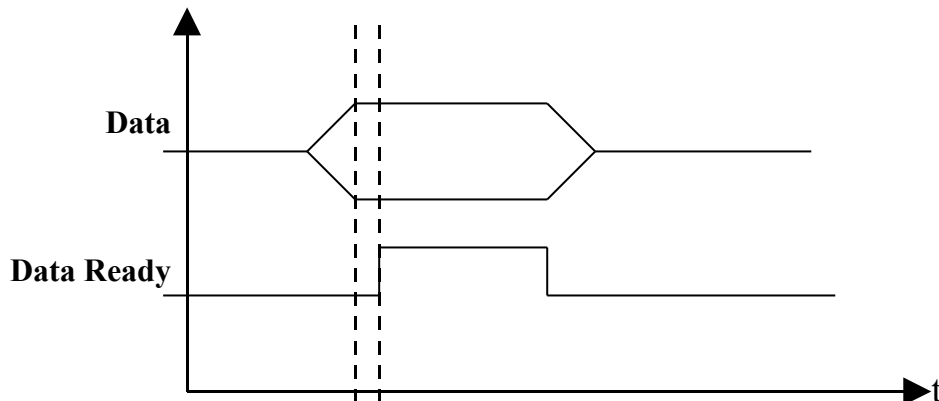
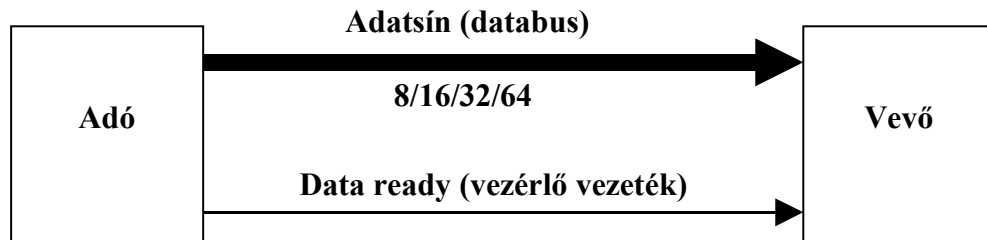
- a szinkron meghajtású sínek csoportosítása átviteli sebességük szerint:
 - hagyományos vagy kompatibilitási sín ~ 5 Mb/sec
 - PCI – 132 vagy 264 Mb/sec
 - AGP ~ 500 Mb/sec
- a tervezési szempontok szerint
 - platformfüggő bővítő sín. Pl.: ISA, EISA, MCA
 - platform független bővítő sín. Pl.: PCI, SCSI, soros & párhuzamos port

Aszinkron átvitel

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

Fajtái:

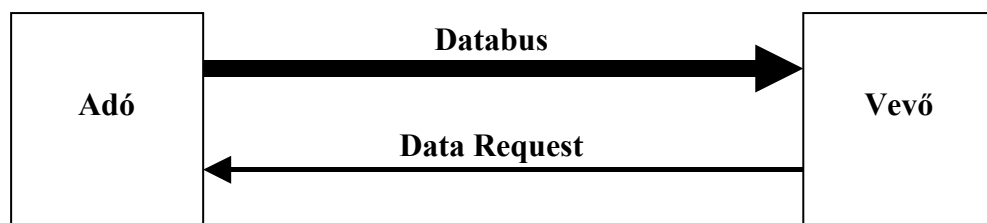
- egyvezetékes (egy vezérlővezetékes)
 - adó oldali vezérlés

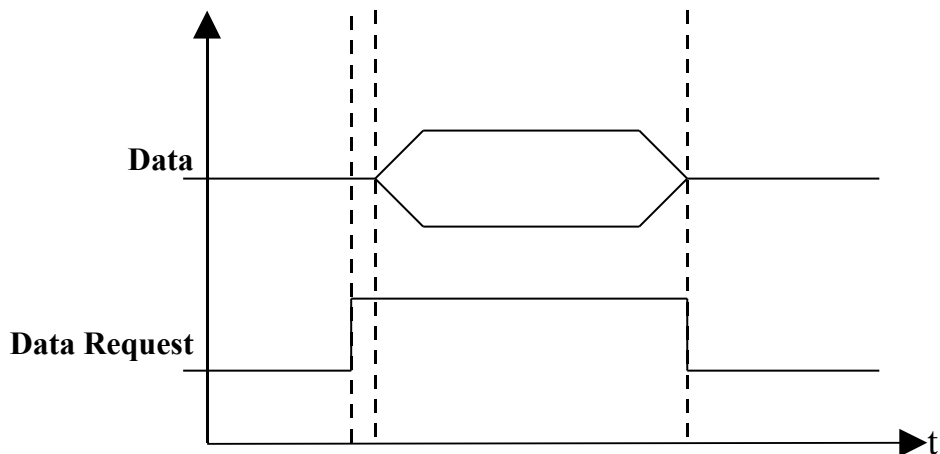


Felteszi a sínre, jelzi, vár, leveszi.

Értékelés: az adónak nincs visszacsatolása az adat célba érkezéséről. (lehet, hogy a vevő évek óta rossz)

- vevő oldali vezérlés → vevő küldi a vezérlőjelet



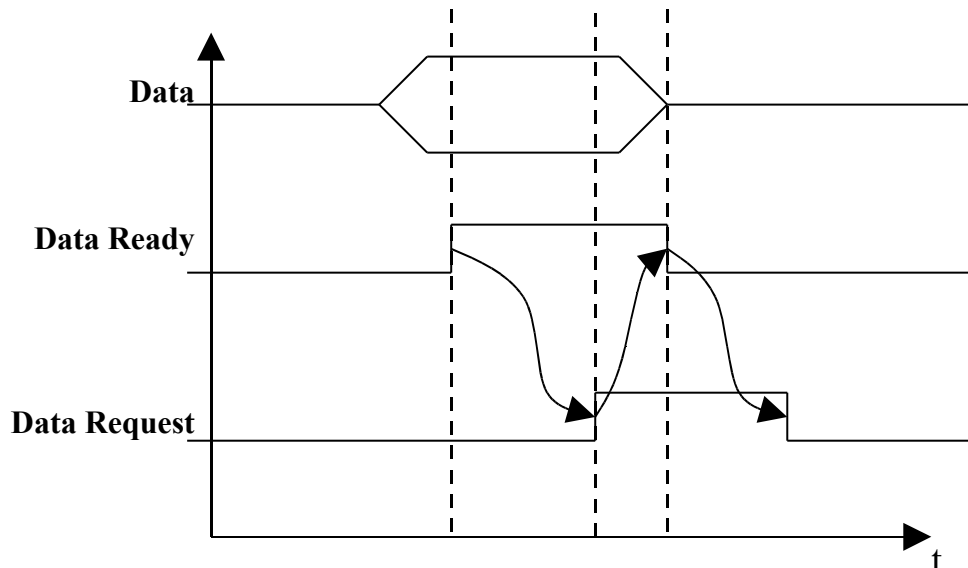
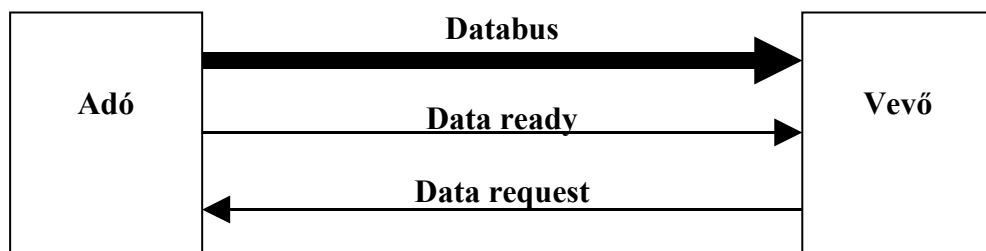


Vevő kérésére történik az adatküldés.

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

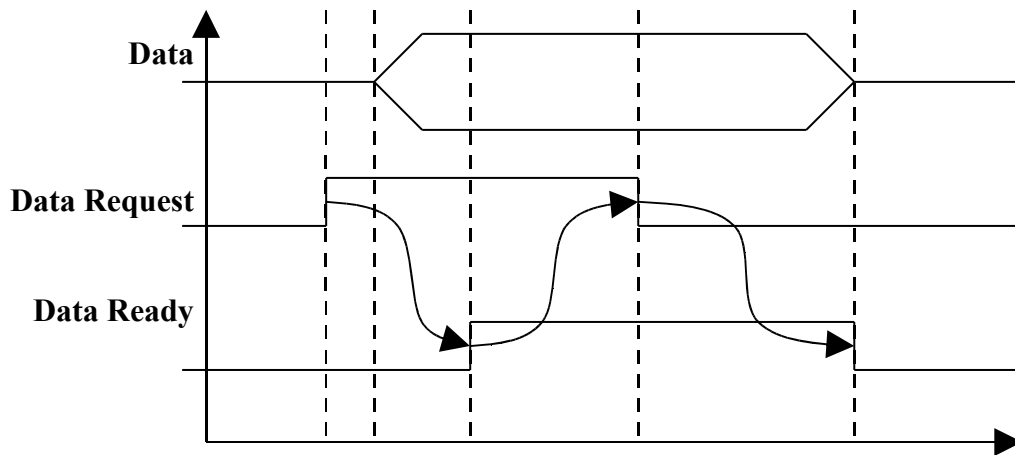
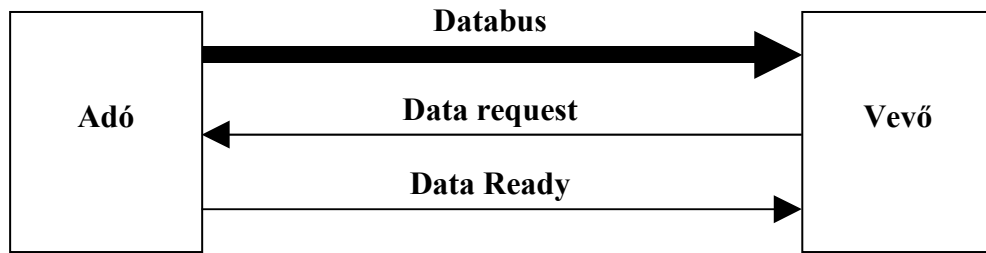


Átvitel folyamata:

Az adó felteszi az adatot az adatsínre, aktiválja a data ready vonalat, a vevő az adat elolvasása után aktiválja a data request vonalat, majd az adó visszaveszi az adatot.

Ezután az adó a data ready -t, a vevő pedig a data request -et deaktiválja

- vevő oldali vezérlés

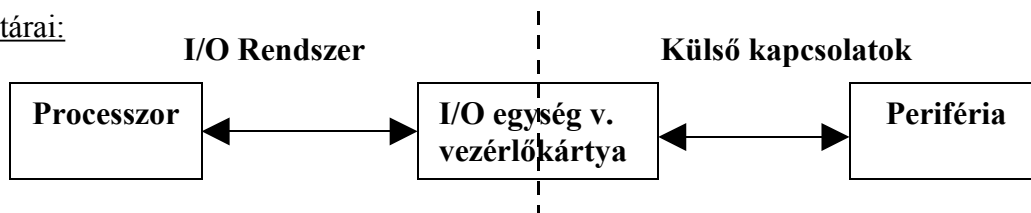


értékelés: az adó visszacsatolást kap az adat célba érkezéséről → megbízható átvitelt biztosít különböző sebességű eszközök esetén is

I/O Rendszer

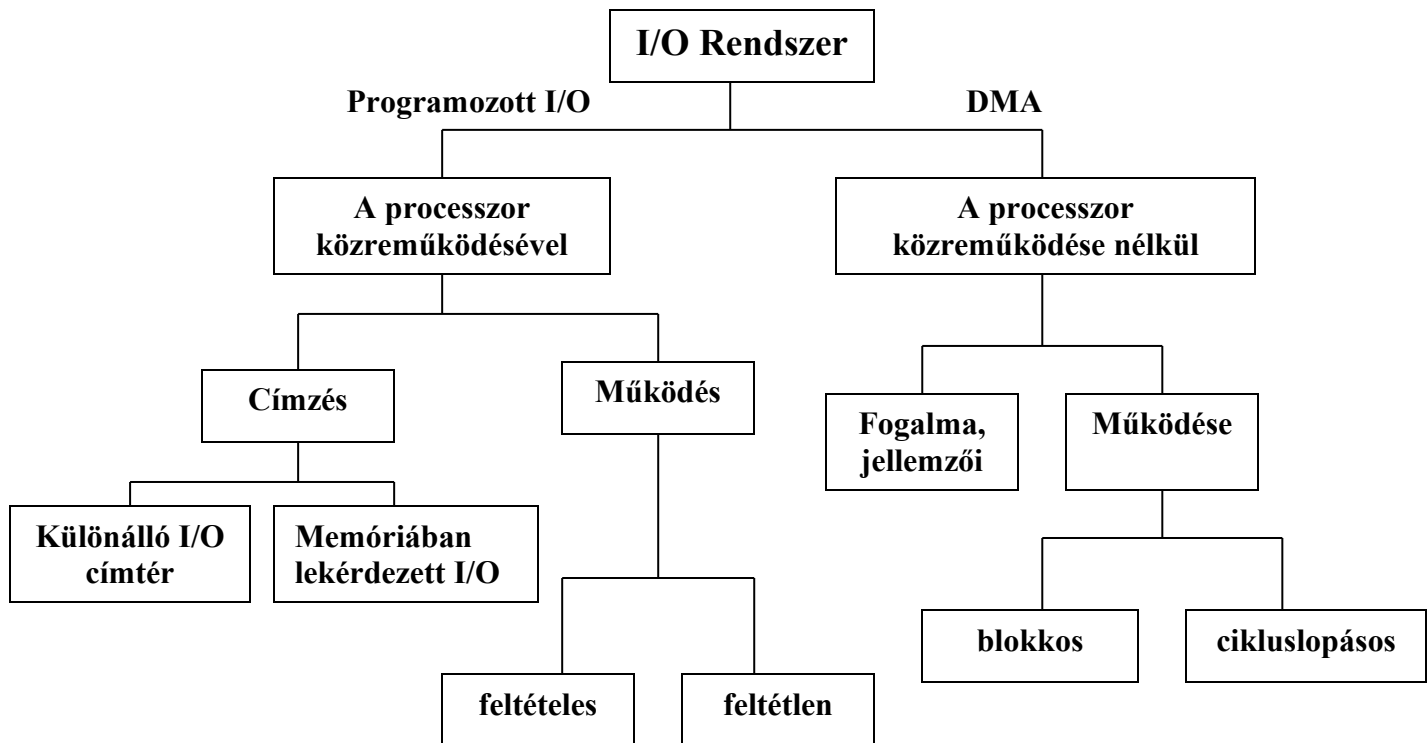
Fogalma: a processzor-memória együtttest a külvilággal összekapcsoló rendszer.

Határai:



Fejlődése:

- a processzor vezérelte a perifériákat
- az I/O modul vezérelte a perifériákat: *wait for flag*
- az I/O modul vezérli a perifériákat: megszakításos üzemmódban
- DMA → közvetlen memória hozzáférés
- Csatorna: I/O célú utasításokat dolgoz fel, közben a központi operatív tárat használja.
- I/O processzor: I/O célú utasításokat dolgoz fel, saját operatív tárral rendelkezik

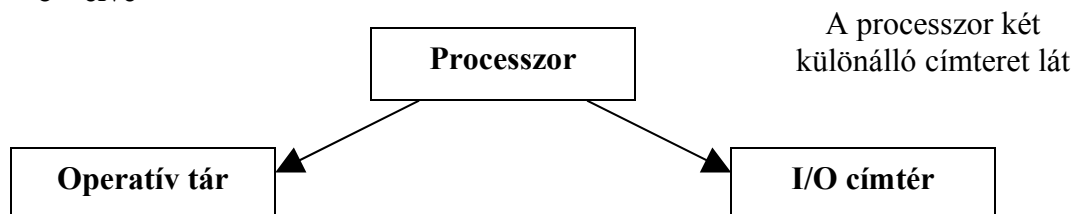
**Programozott I/O**

Fogalma: minden egyes I/O művelethez a processzor egy-egy utasítást hajt végre.

Fajtái címtér szerint:

1. különálló I/O címtér

- elve



- jellemzői

- ugyanazon a címsínen keresztül haladnak a memóriacímek és az I/O címek is (rendszer-sín)
- 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 címsínek az I/O egység címzésére szolgáló része 16 bit hosszú $\rightarrow 2^{16}=65536$ féle I/O cím adható ki
- azon regisztereket, amelyeken keresztül a processzor a perifériákkal kommunikálhat, I/O portnak nevezzük.
- az I/O port fizikailag a vezérlőkártyában helyezkedik el.
- Intel esetében a 32 bites címsínből 16 vesz részt az I/O címzésben is

o I/O port regiszterei

- parancs (Command) regiszter: ebbe írja bele a processzor a kívánságait
- adatregiszterek:
 - bemeneti regiszter: ebből olvassa a processzor a perifériáról kapott adatokat
 - kimeneti regiszter: ebbe írja a processzor a periféria számára küldött adatokat
- állapot (status) regiszter: innen olvassa a processzor a periféria üzeneteit

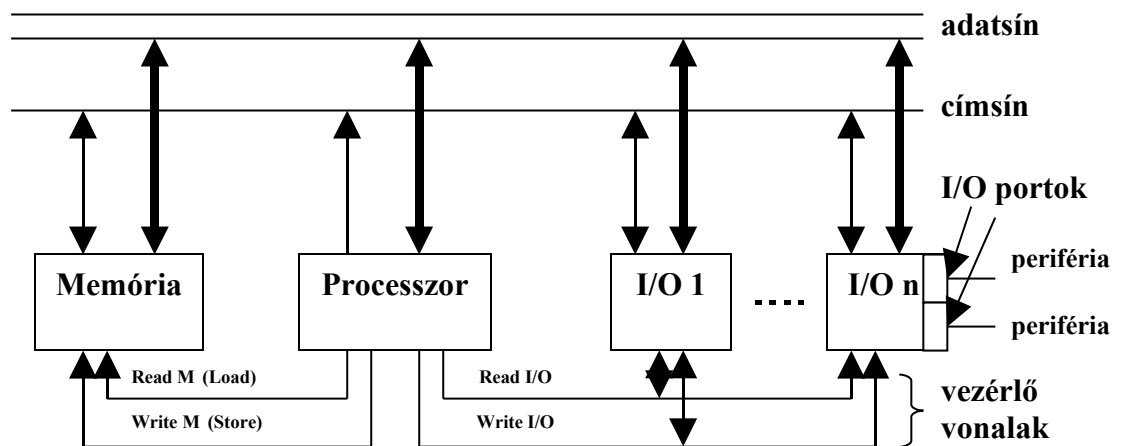
A mai gyakorlat: egy-egy közös regiszterben valósítják meg:

- a parancs- és állapot regisztert,
- a két adatregisztert (be- és kimeneti)

Napjaink további regiszterei az I/O porton belül:

- Az eszköz jelenlétet jelző regiszter
- az eszköz tulajdonságait tartalmazó regiszter (plug&play)
- lehet több regiszterkészlet is

A különálló I/O címtér megvalósítása (AC -n keresztül szállítjuk az adatot, lassú)



- **következmény:**

- a memória műveletekre a load/store utasítások
- az I/O műveletekre, pedig speciális I/O utasítások szolgálnak.
pl. Intel esetén:
 - In X: a processzor olvassa be az X című I/O port adatregiszterét az AC-ba
 - Out X: a processzor írja be az AC tartalmát az X című I/O port adatregiszterébe

- **értékelés**

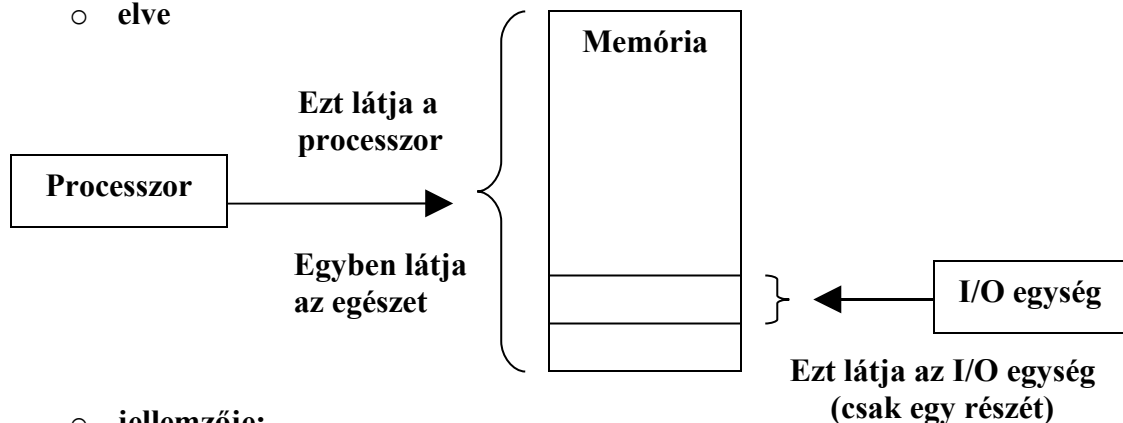
- Előny: egyszerű, olcsó a megvalósítása
- Hátrány:
 - a processzor részt vesz a kommunikációban
 - az AC szűk keresztmetszetet jelent nagy tömegű I/O számára

Ezt az eljárást minden mai architektúra alkalmazza (pl. billentyűzet, soros és párhuzamos port)

Pl.: az IBM PC-nél külön álló I/O címtérrel csatlakozik a hálókártya.

2. a memóriára leképzett I/O (CPU → RAM blokk → Periféria)

- **elve**



- **jellemzője:**

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

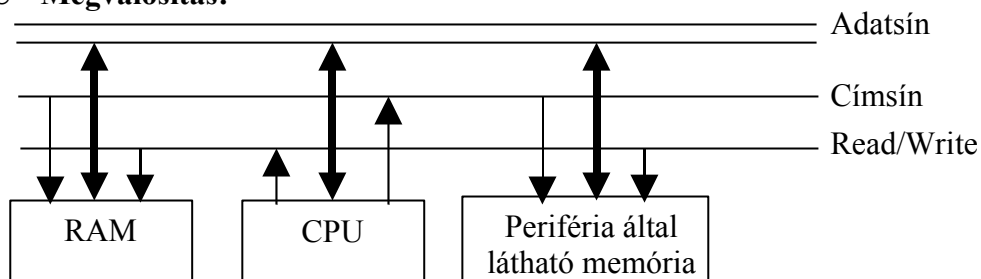
- **értékelése:**

- igen gyors (a különálló címtérnél sokkal gyorsabb)
- továbbra is a processzornak kell utasításokat végrehajtani az I/O során
- Minden mai architektúrában megtalálható

- **Példa:**

- a PC környezetben képernyő (videó) kezelés

- **Megvalósítás:**



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

- pl. 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 proc. időt

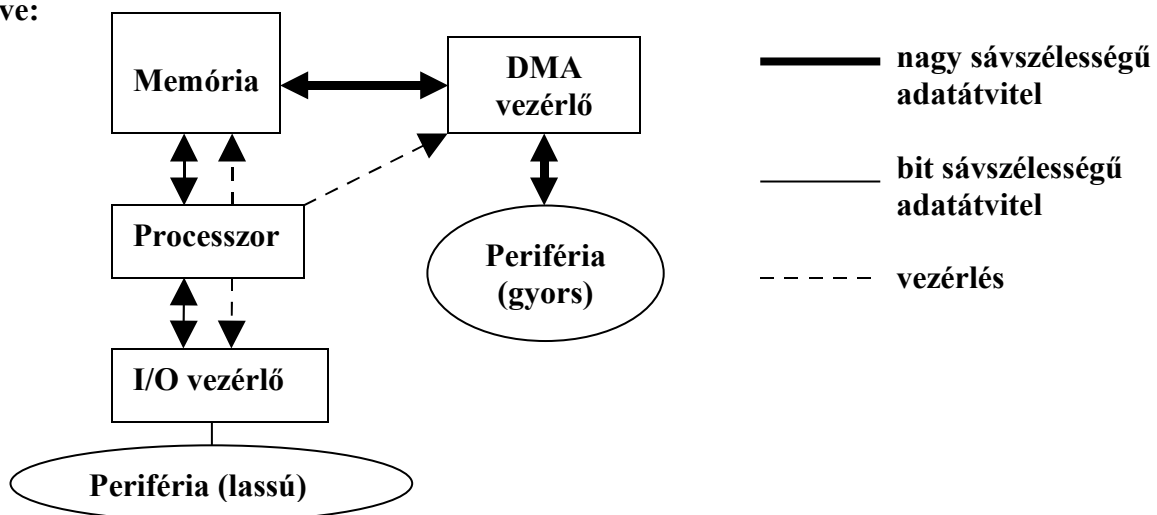
- megszakításos átvitel
 1. a processzor beírja kívánságát az I/O parancsregiszterébe és elkezd mást csinálni
 2. az I/O egység gondoskodik arról, hogy a kívánt perifériáról a kívánt adat beírásra kerüljön az I/O port adatregiszterébe, és ekkor az állapotregiszter „ready” bitjét beállítja, továbbá megszakításjelzést küld a processzor felé
 3. a processzor a következő utasítás-töréspontban észleli a megszakítás tényét 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

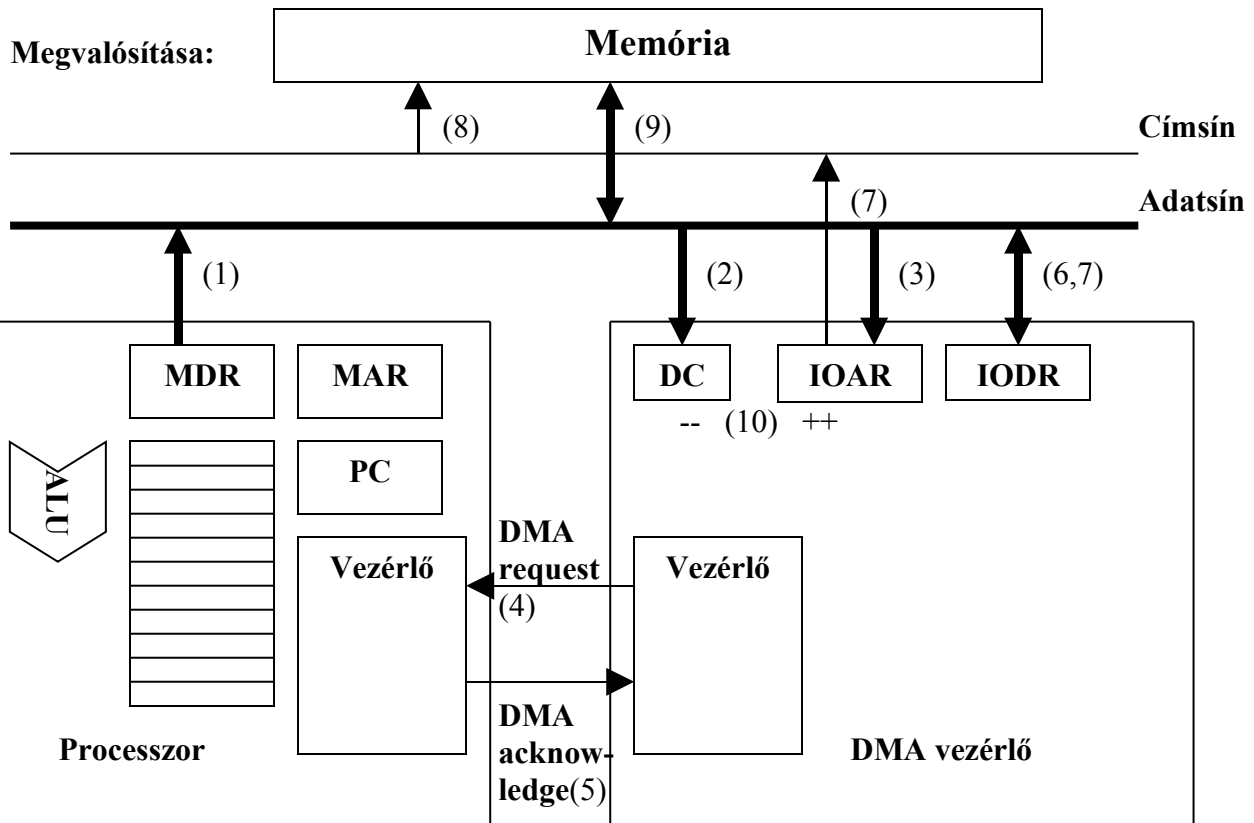
DMA

(Direct Memory Access)

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

Elve:





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 átvendő 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 cikluslopásos módon
 - a résztvevő gységeket (mem.- mem. vagy I/O – I/O átvitel)

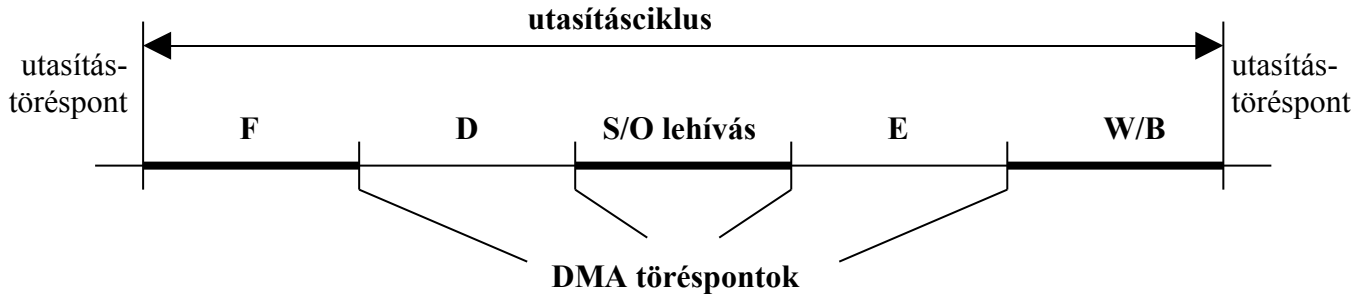
B1. Működés blokkos (burst) átvitel esetén: (pl. winchester esetén)

- 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 átvendő 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 esetén)

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

B2. működés cikluslopásos (cycle stealing) átvitel esetén

pl. gyorsnyomtató: karakteres szervezésű adat kezelése esetén

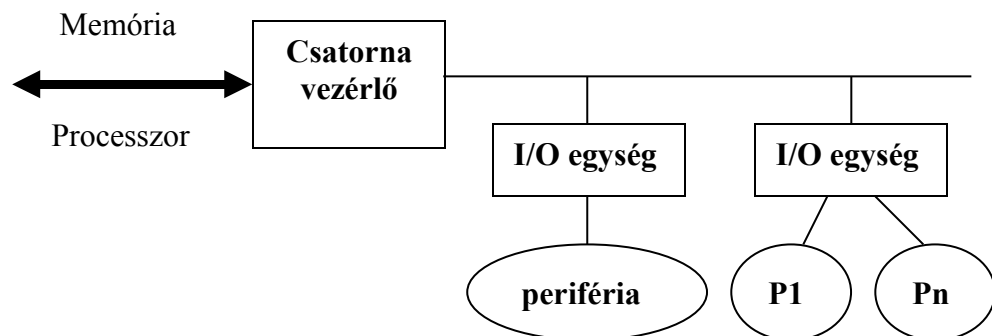


F - fetch S/O – forrás operandus letöltés **————** a processzor használja a rendszersínt
 D – dekóder W/B – visszaírja az eredményt **————** nem használja
 E – executing

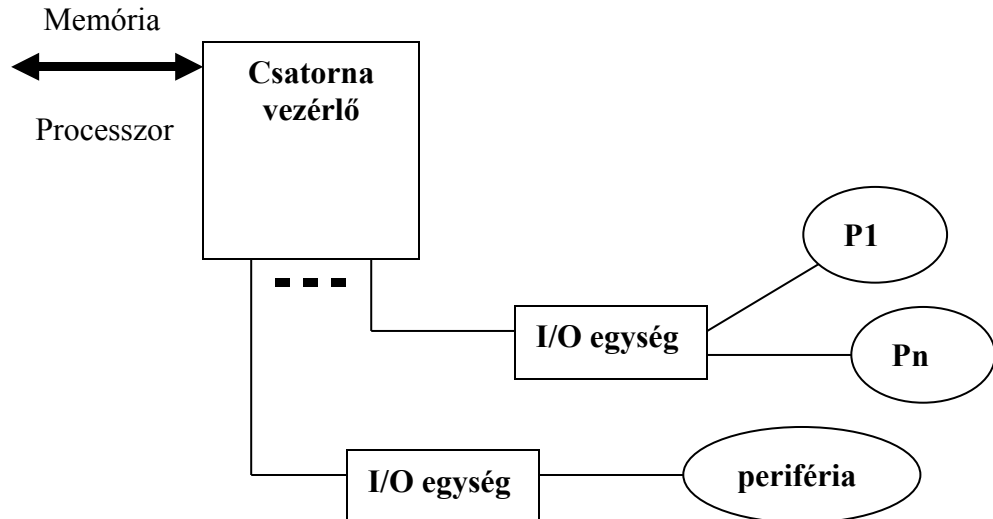
- Nincs értelmezve a címgenerálás
- Míg az utasítás-töréspontban a megszakítás feldolgozással a processzorra további munka várhat, addig a DMA töréspontban a DMA vezérlő a processzor helyett végezhet munkát
- Ez a processzor és a DMA vezérlő általi időosztásos rendszersín használat
- Elve: utasításfeldolgozás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

Csatorna

- ez a DMA koncepció kiterjesztése a lassúbb perifériák irányában
- a csatorna I/O utasításokat kér le a processzorról közös memóriából, majd azokat végrehajtja (nincs saját operatív tára)
- a csatorna által vezérelt műveleteket továbbra is a processzor kezdeményezi
- ebben a koncepcióban is léteznek a perifériák irányítására hivatott I/O egységek vagy vezérlőkártyák, s a csatorna ezek munkáját hangolja össze → a processzor helyébe lép ilyen tekintetben (magát az átvitelt annak kezdete után a csatorna végzi)
- **fajtái:**
 - **szelektor csatorna:**
 - a gyorsabb perifériákat fogja össze, és
 - közülük egyidejűleg csupán egy lehet aktív



- **multiplex csatorna**
 - lassabb perifériákat csatlakoztat
 - közülük egyszerre több is aktív lehet



két fajtája:

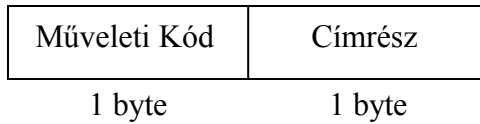
- byte multiplexer:
 - ✓ az átvitel a lehető legnagyobb sebesség biztosítását tűzi ki célul
pl. 3 db egység kommunikál egyszerre, melyek adatai: $A_1A_2 A_3A_4 B_1B_2B_3B_4 C_1C_2C_3C_4$
az eredő adatfolyam lehet pl.
 $A_1B_1C_1A_2C_2A_3B_2C_3\dots$
- blokk multiplexer
 - ✓ a byte multiplexerhez hasonlóan blokkszinten végzi a munkáját

A csatornára példa: IBM/360-as gépcsalád

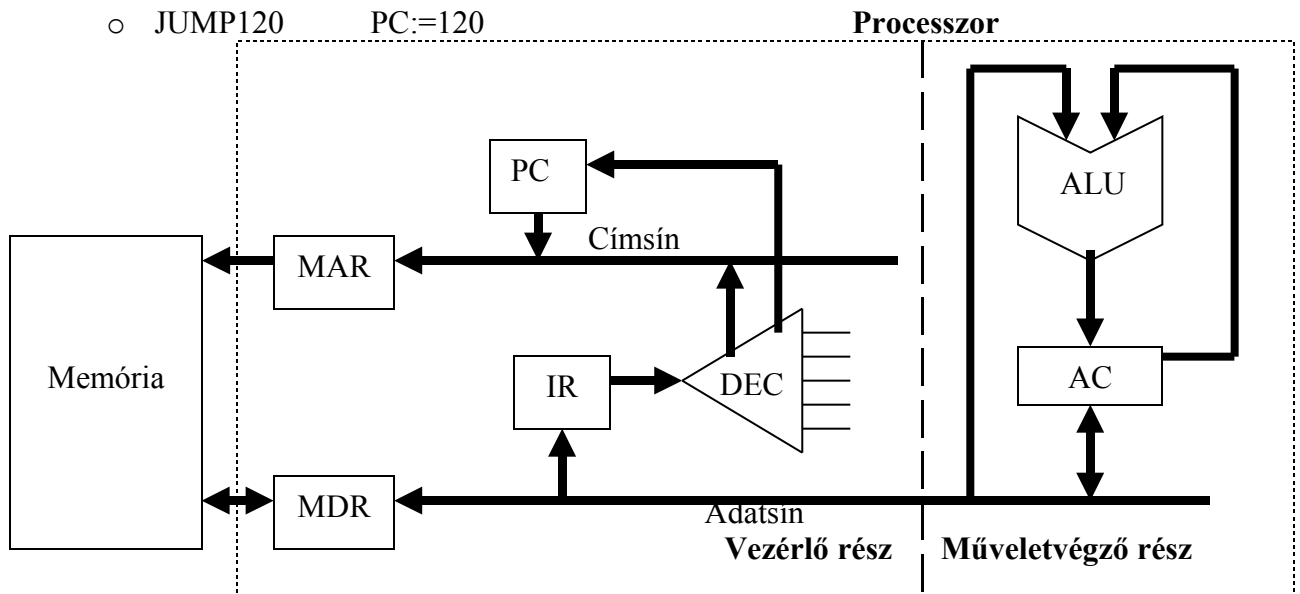
Egy hipotetikus számítógép tervezése

Jellemzői – korlátai:

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



- 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)

$MAR \leftarrow PC$
 $MDR \leftarrow (MAR)$
 $IR \leftarrow MDR$
 $PC \leftarrow PC + 1$

Ehhez hasonlóan a fent felsorolt többi műveletet is ki kell tudni fejteni, pl. Load, ADD, feltétlen vezérlésátadás (16. oldal)

PC tartalma:

100 LOAD[200]
 102 ADD[201]
 104 STORE[202]
 106 NUL
 108 JUMP 150

2-vel inkrementálódik itt a PC

$(Ac \leftarrow 0)$
 (108 -ről 150 -re átírjuk a PC -t)