OBUDA UNIVERSITY YEAR OF JUBILEES
ÓE 15 145 630
TRADITION AND INNOVATION

Óbudai Egyetem
Alba Regia Kar
Mérnöki Intézet

Műveletvégző egység

Dr. Seebauer Márta
egyetemi docens

seebauer.marta@uni-obuda.hu

Szekvenciális mikroarchitektúra

Egy adat-feldolgozó rendszer funkciója, hogy az input adatok **A** halmazát transzformálja az output adatok **B** halmazába.

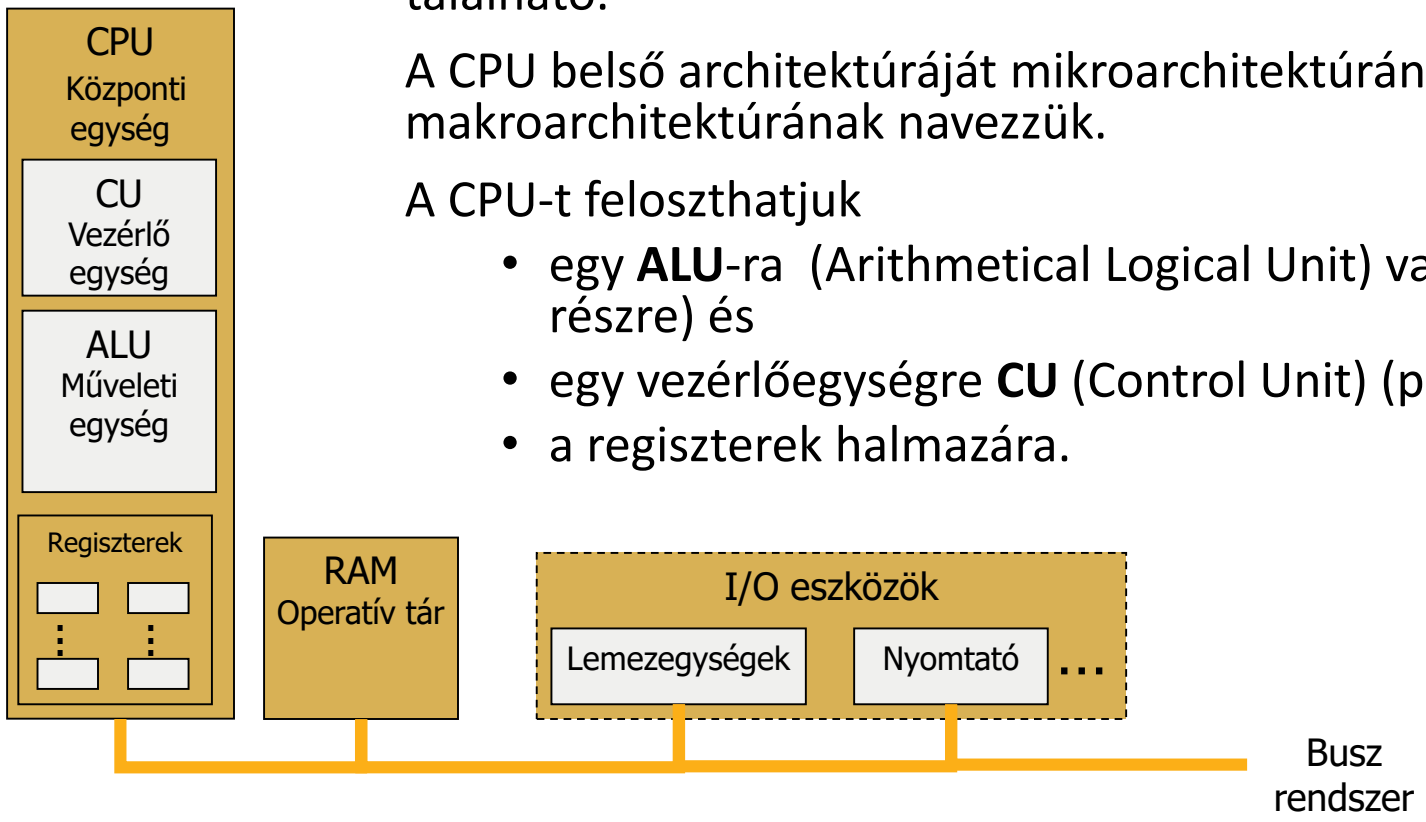
$$B= f(A)$$

Az adatok az operatív tárban helyezkednek el. Az adatfeldolgozást a **CPU** (Central Processor Unit) végzi, amelyet program vezérel, amely ugyancsak az operatív tárban található.

A CPU belső architektúráját mikroarchitektúrának, a processzor környezetét makroarchitektúrának navezzük.

A CPU-t feloszthatjuk

- egy **ALU**-ra (Arithmetical Logical Unit) vagy műveleti egységre (adatfeldolgozó részre) és
- egy vezérlőegységre **CU** (Control Unit) (programvezérlő részre)
- a regiszterek halmazára.



A mikroarchitektúra szintű fizikai architektúra

- műveletvégző egység (ALU – Arithmetical Logical Unit)
- vezérlő egység (CU – Control Unit)
- belső buszrendszer
- általános célú regiszterek
- speciális célú regiszterek
  - programszámláló regiszter (PC – Program Counter)
  - címregiszter (MAR – Memory Address Register)
  - adatregiszter (MDR- Memory Data Register)
  - akkumulátor (AC)
  - utasítás regiszter (IR – Instruction Register)
  - állapotregiszter (PSW Processor State Word)

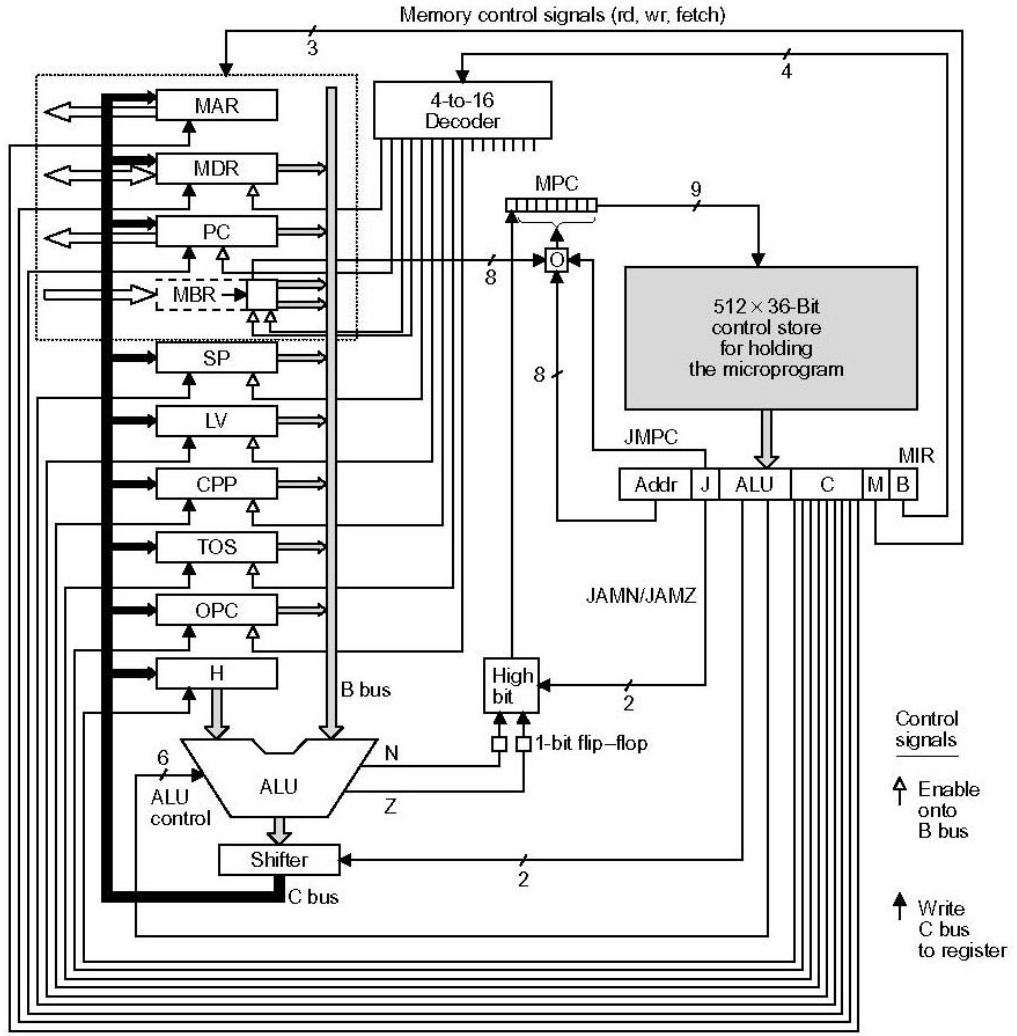
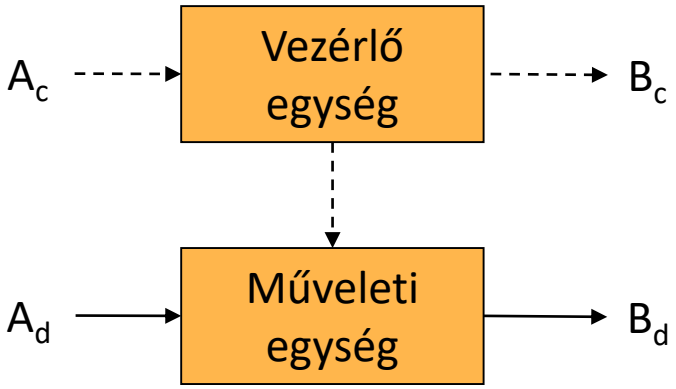


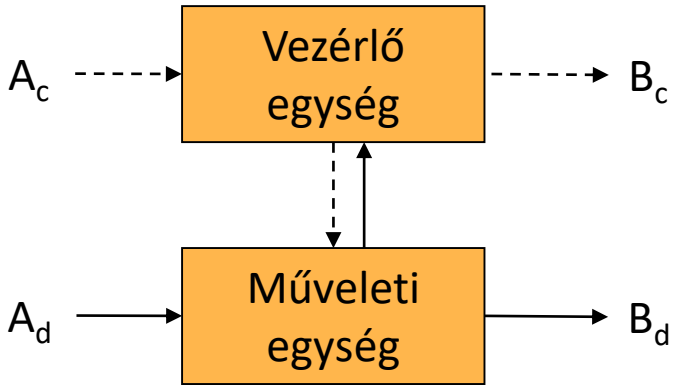
Figure 4-6. The complete block diagram of our example microarchitecture, the Mic-1.

Forrás: Tanenbaum

A műveletvégző és a vezérlő egység kapcsolata



$B_c = f_c(A_c)$
$B_d = f_d(A_d, A_c)$



$B_c = f_c(A_c, A_d)$
$B_d = f_d(A_d, A_c)$

Ha a vezérlőegység logikai áramkörökből áll, és így az f_c alapvetően állandó, akkor a rendszert huzalozottnak hívják.
Ha pedig az f_c megvalósítása memóriában tárolt vezérlési információkkal történik, és a vezérlési funkciók inkább szoftver, mint hardver úton valósulnak meg, akkor mikroprogramozott vezérlésről beszélünk.

Szinkron ütemezés

A szinkron vezérlésű processzor külső és belső órajellel (CLK) rendelkezik. Ezt egy elektronikus áramkör, az órajel generátor állítja el, amely rendszeres és pontos időintervallumonként elektronikus impulzusokat generál, ennek az időtartama a ciklusidő (T), mértékegysége [s]. Az órajelet nem a ciklusidővel, hanem annak reciprokával, az órajel frekvenciával (f) jellemezzük, mértékegysége [1/s= Hz].

Minden műveleti lépésnek egy óraimpulzusra kell kezdődnie. Ez viszonylag egyszerű processzor felépítést jelent, de a hátránya, hogy nem minden művelet igényel ugyanolyan hosszú időt, hiába fejeződött már be az előző művelet, a következő addig nem kezdődhet meg, amíg a következő óraimpulzus nem érkezik meg.

A ciklusidőt az ütemezett leghosszabb adatút össz-késleltetéséhez kell szabni. Amennyiben több párhuzamos és késleltetésében jelentősen különböző adatutat kell a vezérelni, célszerű az órajel ütemet több ciklusra (C) felosztani. Ezért jön létre a processzor külsőnél nagyobb frekvenciájú belső órajele.

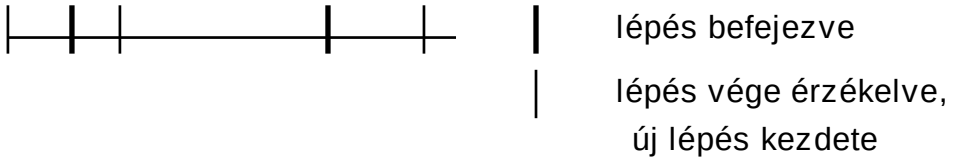
A hátránya ellenére a minden mai processzor ilyen.

Aszinkron ütemezés

Az aszinkron vezérlésnél a következő lépés akkor kezdődik, amikor az előző befejeződött. Ez kiküszöböli a processzornak a következő órajelre való várakozását, következésképpen a processzor működési sebességének növekedését kell, hogy maga után vonja. Ez úgy valósítható meg, hogy egy külön logikai áramkört építenek be, amely érzékeli minden esemény végét. Hátránya:

- a külön áramkör az aszinkron processzort drágábbá teszi
- az esemény végének az érzékelése bizonyos időt igényel, és ez csökkenti a szinkron processzorhoz viszonyított időmegtakarítást.

Az aszinkron üzemmódú CPU egyébként általában gyorsabb működésű lenne, de bonyolultabb és drágább, mint a szinkron üzemmódú, ezért nem használják.



Belső sínrendszer

A számítógépekben, hogy megfelelő sebességgel működjenek, párhuzamos adatáramlást kell biztosítani. Ennek érdekében megfelelő számú vezeték (vonal) szükséges a kapcsolatok kialakításához. Ezt a bizonyos közös tulajdonságokkal rendelkező vezetéknyalábot, úgynevezett dedikált vezeték köteget hívják **sínnek** vagy **busznak**.

A sín funkcionálisan

- adatsínre
- címsínre és
- vezérlősínre

osztható.

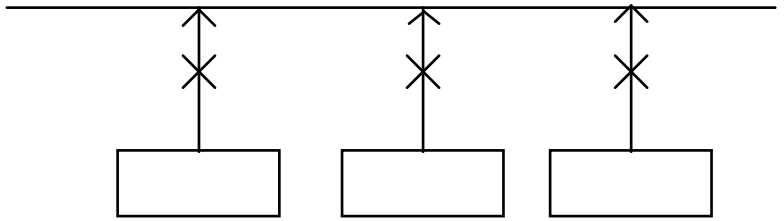
A számítógépen belül a sínrendszer nem csak a processzor és más, a processzoron kívüli eszközök összekötésére szolgál, hanem magán a processzoron belül is kialakítottak egy belső sínrendszert. Ennek megfelelően megkülönböztetünk:

- belső sínrendszert és
- külső sínrendszert.

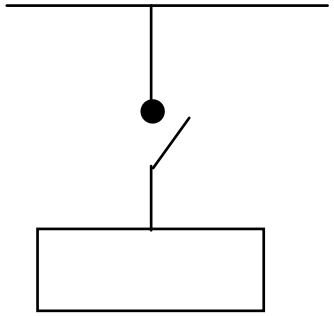
A belső sínrendszer esetében a vezérlésre szolgáló vezetékek nem szerepelnek a sínrendszer részeként, mivel a vezérlés módjából adódóan az nem különül el önálló rendszerként.

Kapcsolópontok

A sín nem megosztható eszköz. Egy sínen egy időben csak egy adó lehet. Ha két adó működik, a bitek ütköznek.



Egy sínre több egység kapcsolódik. A rávezető és az onnan elvezető vonalban van egy-egy kapcsoló, melyek feladata: egy időben az adatúton csak egyetlen egy eszköz lehet nyitva.



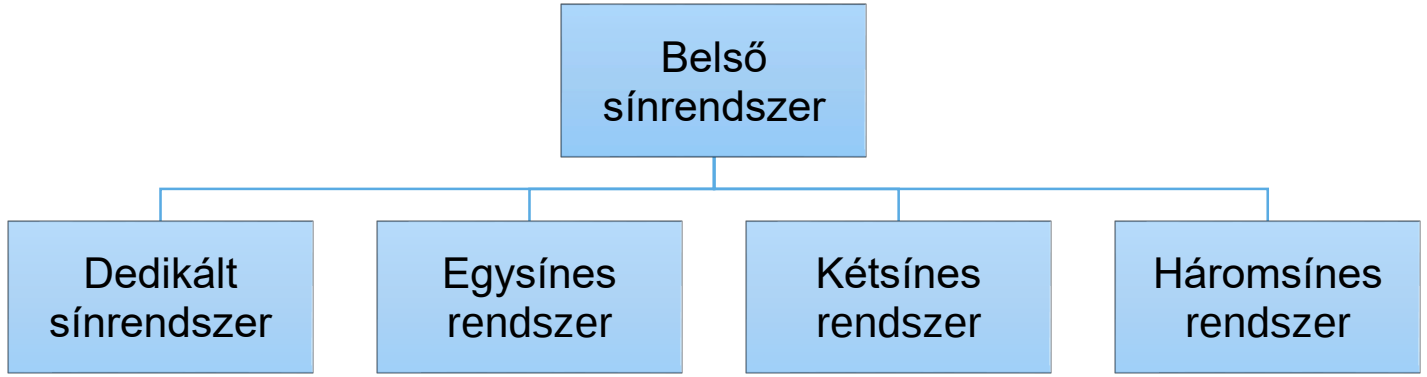
A kapcsoló nem passzív, hanem egy aktív eszköz, egy tranzisztor. A kapcsoló három állapotú

- 0
- 1
- Z, nagyimpedanciás, amikor adat nem folyhat át rajta.

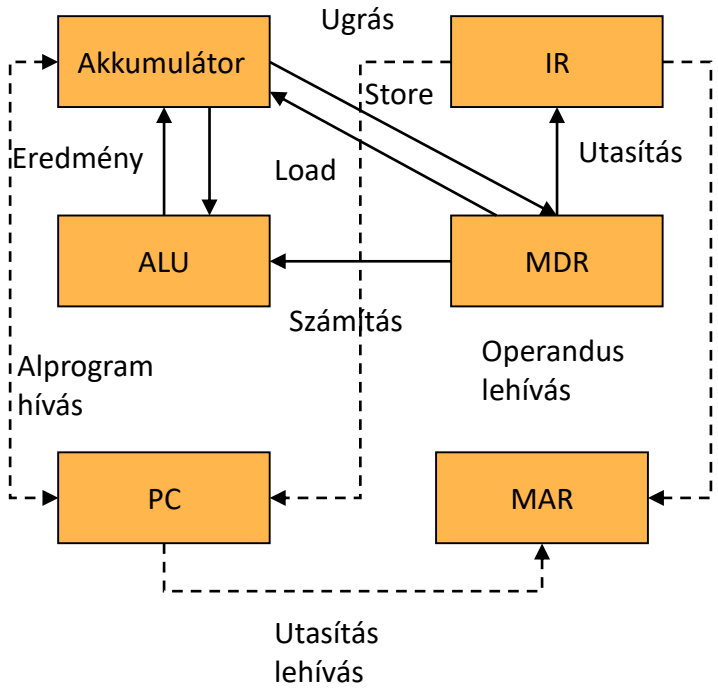
A regiszter output kapuja képes arra, hogy a regisztert elektronikusan lekapcsolja az adatútról, vagy pedig 0-át vagy 1-et helyezzen az adatútra. Mivel ez három lehetőséget támogat, az ilyen kaput három-állapotú (tri-state) kapunak nevezzük. Önálló input-vezérlés is van, ami lehetővé teszi, hogy a regiszter elektronikusan le legyen kapcsolva az adatútról, illetve az adatútról 0-át vagy 1-et fogadjon.

A kapcsolópontok általában a regiszter részét képezik.

A mikroarchitektúra kapcsolati rendszere



Point-to-point vagy dedikált sínrendszer



Minden szükséges információ átvitel számára ebben az esetben dedikált sín biztosított. Az ilyen adatátviteli mód előnye, hogy sok adatátvitel történhet párhuzamosan, így ez a tendencia vezet a gyors CPU felé. A hátránya viszont, hogy igen drága lenne valamennyi sín biztosítása, a duplex kapcsolatok száma $n(n-1) \sim O(n^2)$ az eszközök számával négyzetes arányban növekszik.

Egy egyszerű CPU esetében, mely csupán egyetlen felhasználói regisztert ismer, az akkumulátort, összesen tíz sínnel, nevezetesen

- négy címsínnel és
- hat - utasítást és operandust továbbító adatsínnel kell rendelkeznie.

Nyilvánvaló, hogy amennyiben több regiszter érhető el számunkra és több utasítást biztosítunk, akkor több sínre van szükségünk.

Ha tanulmányozzuk egy gépi utasítás műveletéhez szükséges átviteleket, kiderül, hogy az átvitelek többsége logikailag nem történhet párhuzamosan még akkor sem, ha ezt a lehetőséget fizikailag biztosítjuk. Következésképpen a CPU belső sín szervezésénél egy teljesen dedikált vonalú szervezés gyakorlatilag sohasem használatos.

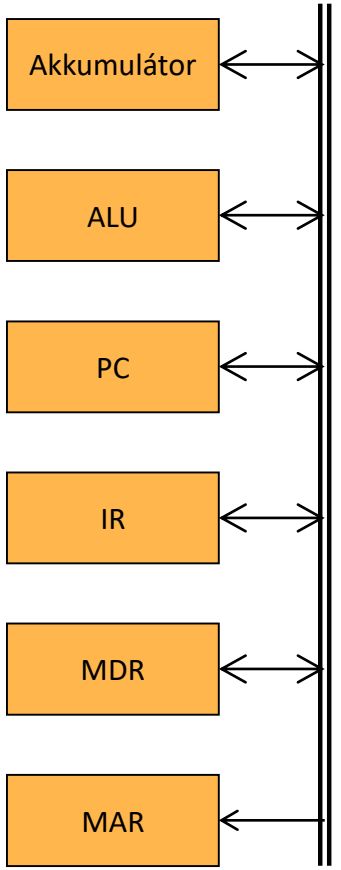
Közös sín vagy egysínes rendszer

A másik extrém esetként szemlélhetjük a közös sínrendszert. Valamennyi adatátvitel a közös sínen történik.

Annak érdekében, hogy lehetővé tegyük az adatoknak az egyik regisztertől a másikhoz való áramlását, néhány logikai kapura (ki/be kapcsolóra) van szükség, amely lehetővé teszi, hogy egy időben csak a szükséges regiszterek egyike kapcsolódjon a sínre.

A közös sínrendszer előnye, hogy igen olcsó, a hátránya viszont, hogy egy időben kizárólag egyetlen átvitel lehetséges. Így nincs lehetőségünk arra, hogy párhuzamos műveleteket végezzünk, következésképpen az ilyen típusú processzor lassú.

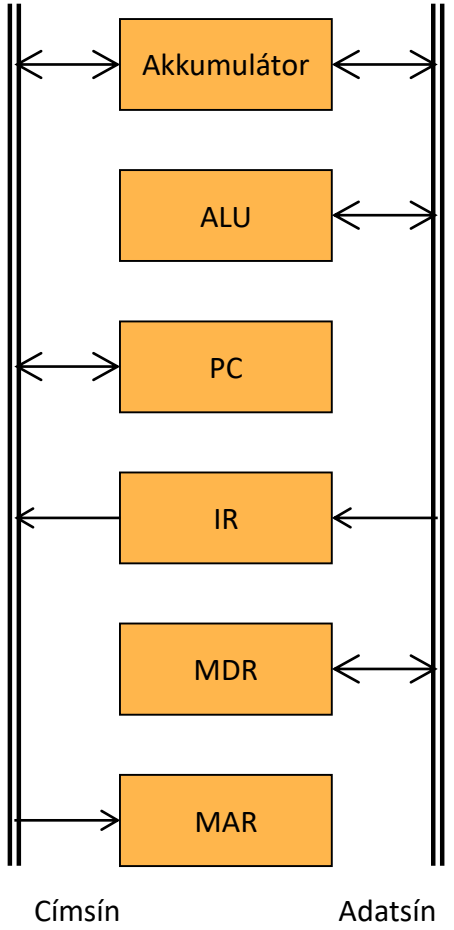
A közös adat és címsín használata csak a nagyon egyszerű, célfeladatokra használt processzoroknál alkalmazott. Gyakorlati jelentősége már nincs is.



Kétsínes rendszer

Egyszerű megoldást ad a kétsínes rendszer, amely különválasztja az adat és címsínt. Ez általánosan elterjedt megoldás a processzorok körében.

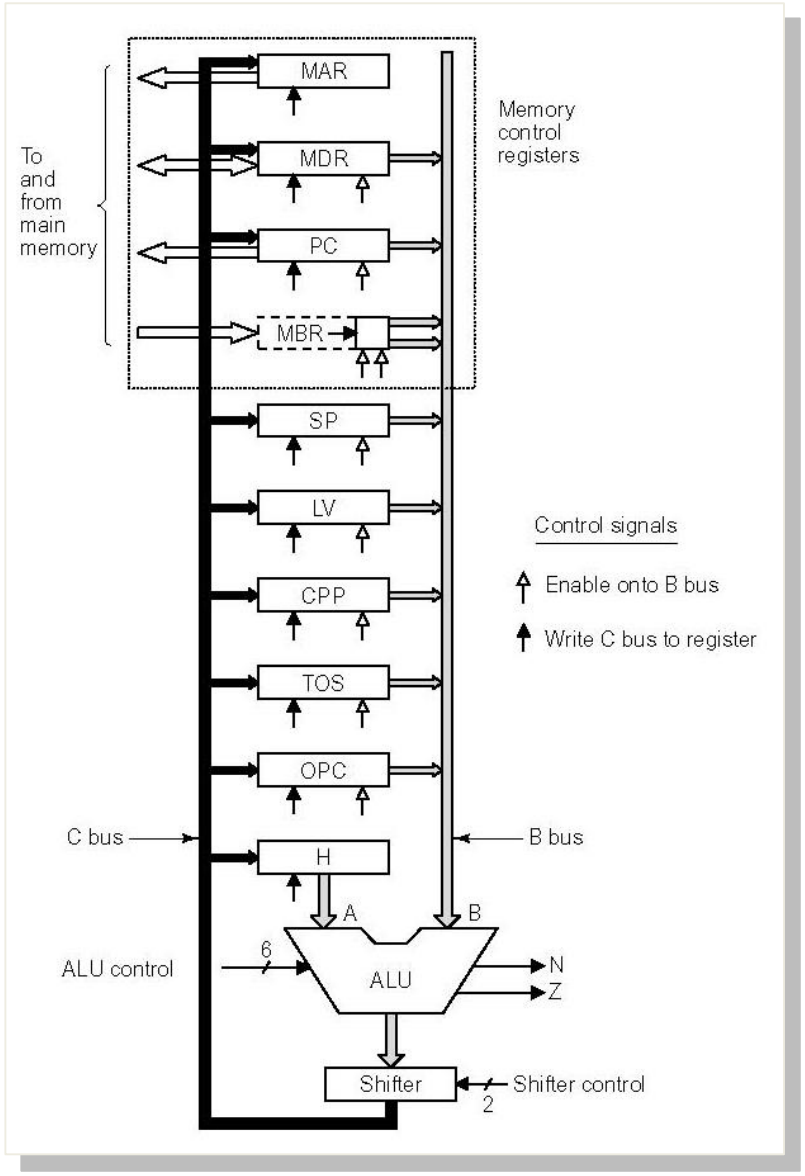
Kétsínes rendszert alkalmaz: LSI-11, MC68020, I386/486/Pentium.



Háromsínes rendszer

Nagyobb teljesítményű processzorok esetében az átvitelek gyorsítása érdekében háromsínes rendszer kialakítása a célszerű, amelynél a címsín mellett külön adatsín van az írásra és az olvasásra. Ezzel a közel egyidejű írás és olvasás megoldható.

Háromsínes rendszert alkalmaz: AM 2900, RISC-processzorok.



Forrás: Tanenbaum

Műveletvégző egység

Regiszterek gyors hozzáférésű átmeneti tárolók, mely a műveletvégző egységen belül helyezkedik el. Ebben adatokat tárolhatunk, elérésük gyors. Lehet engedélyezni vagy tiltani az írást: van vezérlője.

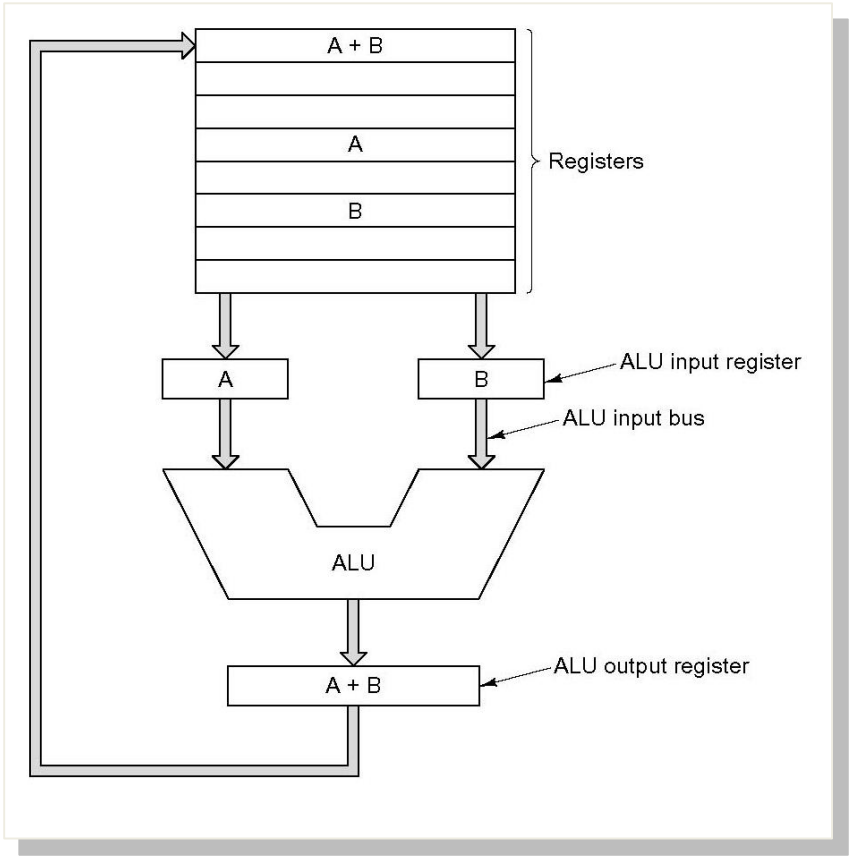
A regisztereket két csoportba sorolhatjuk:

- az egyik csoport már a logikai architektúrában szerepelt: programozható regiszterek (szabadon felhasználhatók).
- "rejtett" (belső) regiszterek (a felhasználó nem látja őket, pl. puffer-regiszterek).

Adatút

A CPU azon része, amely tartalmazza az ALU-t a bemeneteivel és a kimeneteivel együtt

- egy adatutas
- két adatutas
- három adatutas
  - 2 regiszter intput
  - 1 regiszter output

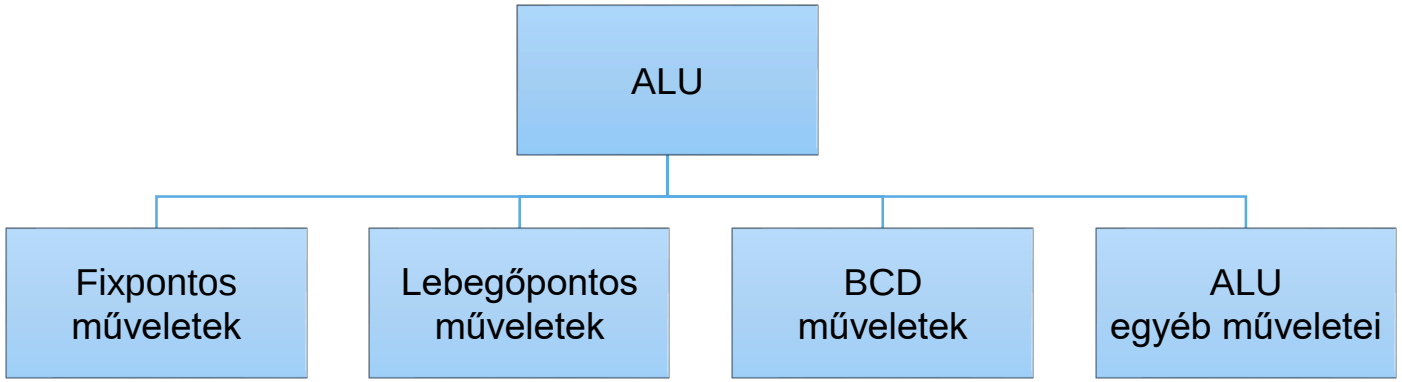


Forrás: Tanenbaum

A műveletvégző egység műveleteinek osztályozása

Az ALU egy komplex műveletvégző egység, amely képes aritmetikai és logikai műveletek végrehajtására. Ez azt jelenti, hogy minden, az ISA szinten meghatározott műveletet, amelyet az ALU nem képes hardver úton elvégezni, a mikroprogramnak mint egy beépített interpreternek kell elvégeznie. Tehát a processzor gyártójának el kell döntenie, hogy egy adott utasításnak megfelelő műveletet hardveresen vagy mikroprogram útján fog-e megoldani.

A magasabb szintű műveleteket aritmetikai alapműveletekre (összeadás, kivonás, szorzás, osztás), az aritmetikai műveleteket pedig logikai műveletekre vezetik vissza. A szorzás és az osztás megoldható hardveresen, külön áramkörök segítségével, vagy összeadások, illetve kivonások és léptetések sorozatával mikroprogram segítségével. A kivonást visszavezetjük komplemens képzésre és összeadásra.



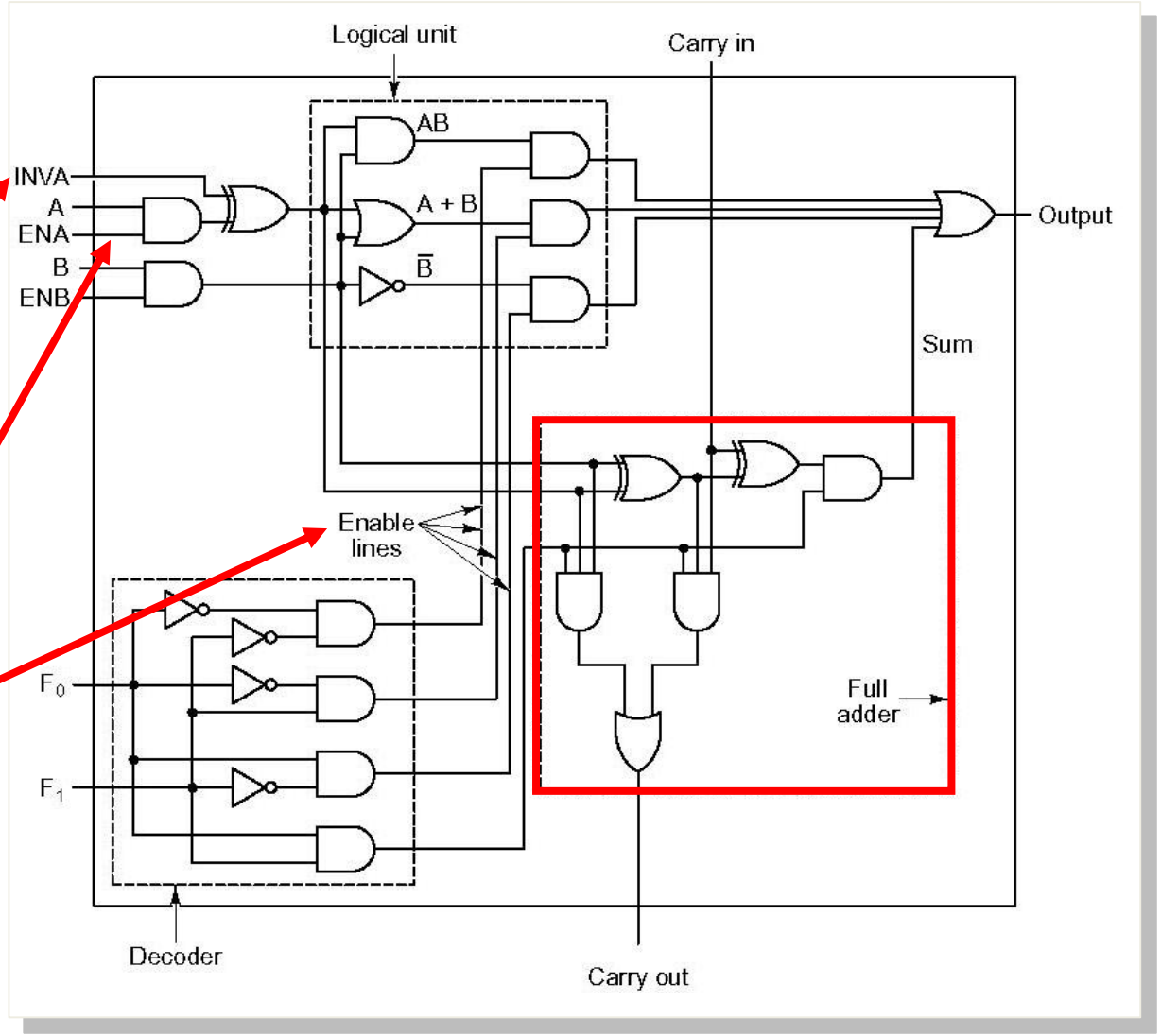
Fixpontos ALU

Egybites komplex ALU

Komplemens képzés

Az ALU központi eleme egy teljes összeadó

Vezérlő pontok



Forrás: Tanenbaum

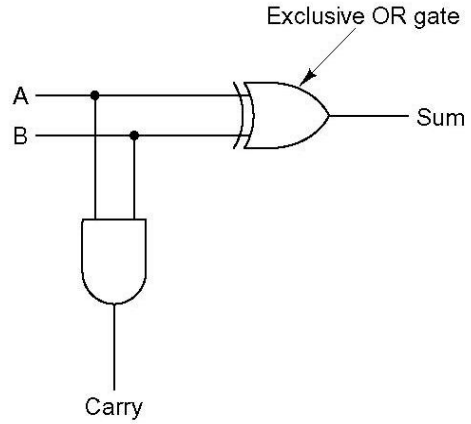
Egybites összeadó

Tervezés lépései

1. Igazságtábla felírása
2. Logikai függvények felállítása
3. Azonos átalakítások
   - elemek számának minimalizálása
   - végrehajtási idő optimalizálása
4. Megvalósítás

Félösszeadó

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

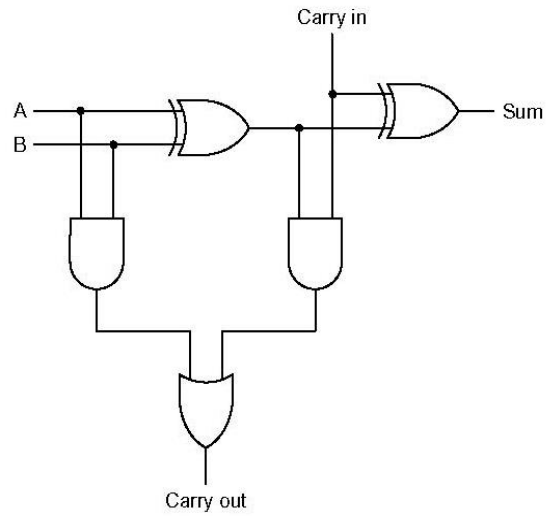


$$S = \overline{A}B + A\overline{B} = A \oplus B$$

$$C = AB$$

Teljes összeadó

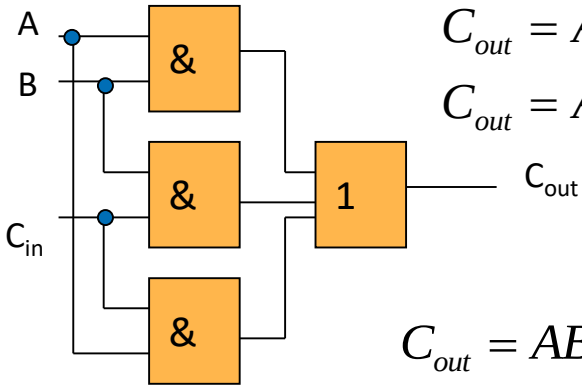
A	B	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + (A \oplus B) \cdot C_{in}$$

$$C_{out} = AB + (A + B) \cdot C_{in}$$



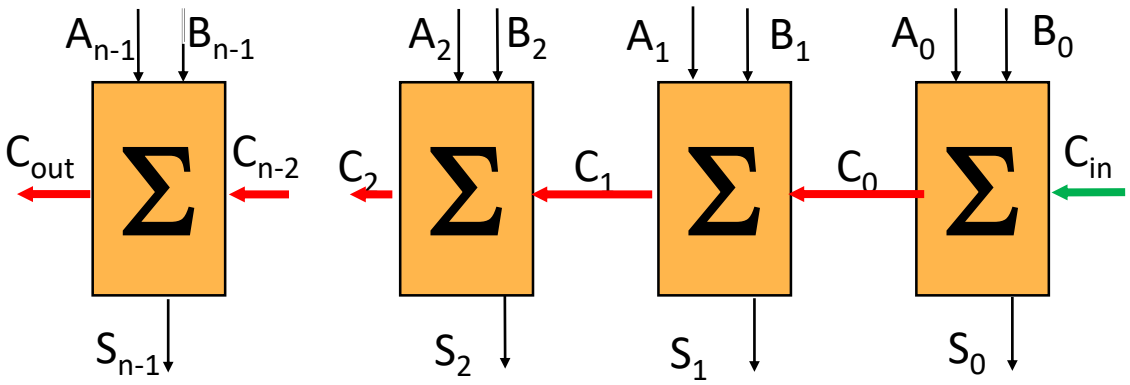
$$C_{out} = AB + AC_{in} + BC_{in}$$

n-bites párhuzamos összeadó

A valamennyi n bit-párt szimultán módon összeadó áramkört párhuzamos összeadónak nevezzük. Egy egyszerű párhuzamos összeadót kialakíthatunk **n db** teljes összeadóból. Minden egyes teljes összeadó fokozat balról egy carry bemenetet tartalmaz.

Könnyen belátható, hogy a vázolt módon a párhuzamos összegzéshez annyi idő szükséges, amennyi idő alatt a legkisebb helyiértéken keletkező átvitel (**carry**) hatása el tud jutni a legnagyobb helyértékig. Az összeadandó számok változása után ugyanis az S_i kimeneten mindig tranziens változás játszódik le, amíg minden egyes C_{i-1} bemenet értéke nem állandósul. Ez a legnagyobb helyértéken következhet be legkésőbb. A legkedvezőtlenebb esetben ugyanis minden helyiértéken történhet változás a keletkező átvitel értékében.

A maximális összeadási időigény tehát **nd**, ahol d egy teljes összeadó fokozat időigénye. Láthatjuk, hogy nagy mennyiségű többlet-áramkörrel ugyanaz a probléma, mint a soros összeadónál, azaz az időszükséglet a bitek számával lineárisan nő.



"Szimultán" átvitelképzés - carry look-ahead vagy rekurzív módszer

$$C_{out} = AB + (A+B) \cdot C_{in}$$

A carry tehát független az előző helyiértéken képződő carry-től, kizárólag a befolyó operandusoktól és a kívülről beérkező átviteltől függ. Mindent tehát előre ismerünk, hisz a mindhárom forrásadat előre ismert és az összeadás végrehajtása megkezdésekor már a rendelkezésünkre áll.

AB bitcsoport = G Generate: a carry-t generáló bitcsoport

A+B bitcsoport = P Propagate: a carry-t terjesztő bitcsoport

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

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

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

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

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

$C_{n-1}=G_{n-1}+P_{n-1}(G_{n-2}+P_{n-2}*(G_{n-3}+P_{n-3}*(...)))$

Elvégezzük a kijelölt műveleteket:

$C_2=G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0*C_{in}$

$C_3=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0+P_3P_2P_1P_0*C_{in}$

Akármennyire fejtjük is ki, mindig lesznek benne

- szorzatok
- azokat kell összeadni

és ez együtt kettő kapumélység.

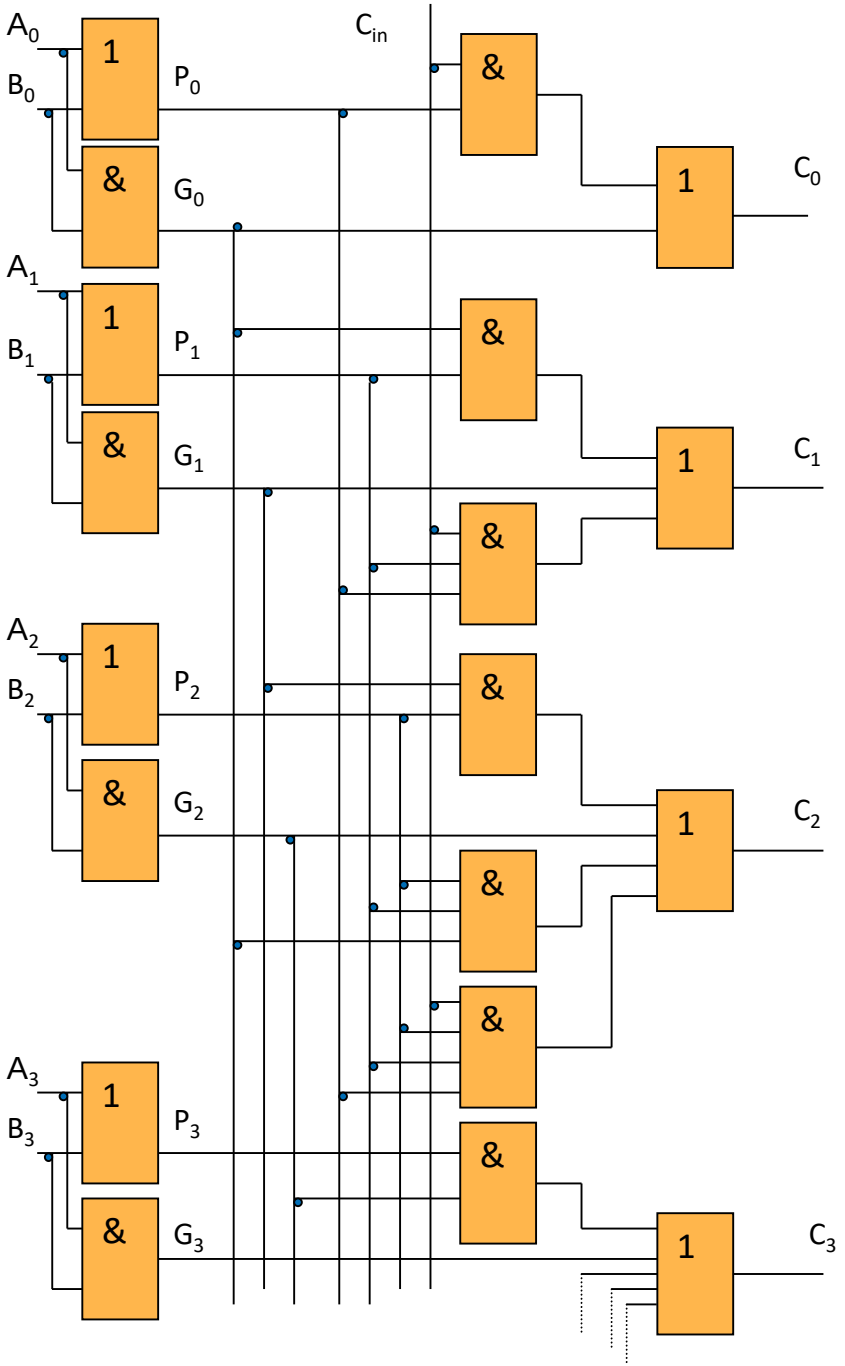
Ehhez jön a P és G meghatározásához szükséges további kapumélység, tehát összesen 3 kapumélység, azaz a carry look-ahead (CLA) átvitelgyorsító összeadó teljes ideje **3d**

CLA megvalósítása

Mivel a carry-generáló egyenletek összetettsége a fokozatok számával nő, ezért gyakorlati megfontolásokból a carry-lookahead fokozatok száma kisebb nyolcnál

$C_0=G_0+P_0*C_{in}$
$C_1=G_1+P_1G_0+P_1P_0*C_{in}$
$C_2=G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0*C_{in}$
$C_3=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0+P_3P_2P_1P_0*C_{in}$
…

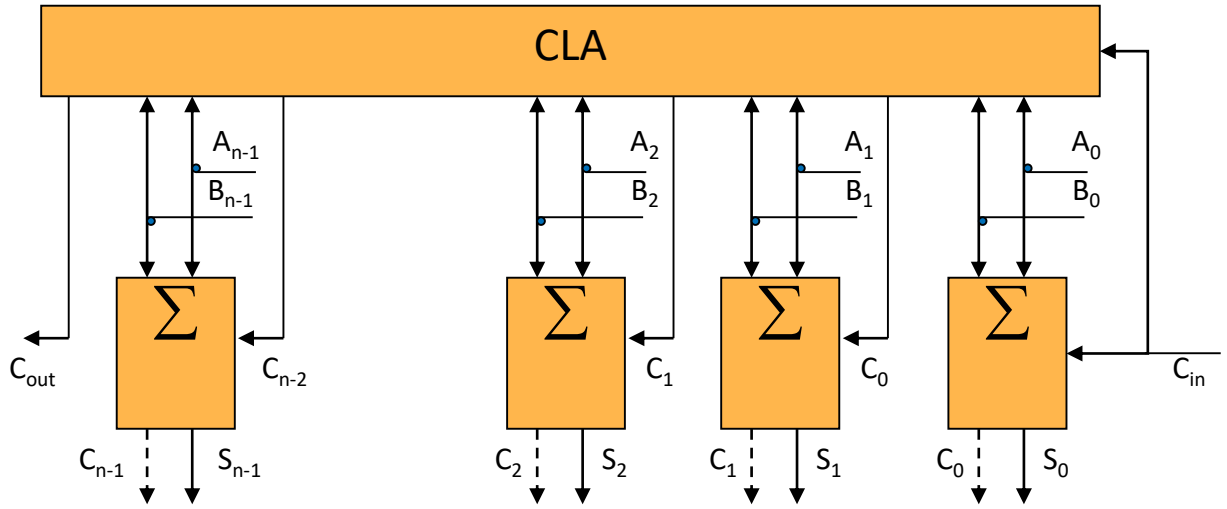
AB bitcsoport = G
A+B bitcsoport = P



Összeadó és CLA áramkör

Az átvitelgyorsító (CLA) az átvitel értékét az összeadandó számok bitjeiből és a C_{in}-ből még az összeadás elvégzése előtt, minden helyértéken egyszerre állítja elő.

A carry-k továbbra is képződnek, csak nem használjuk őket semmire, mert igen lassan jönnek létre.



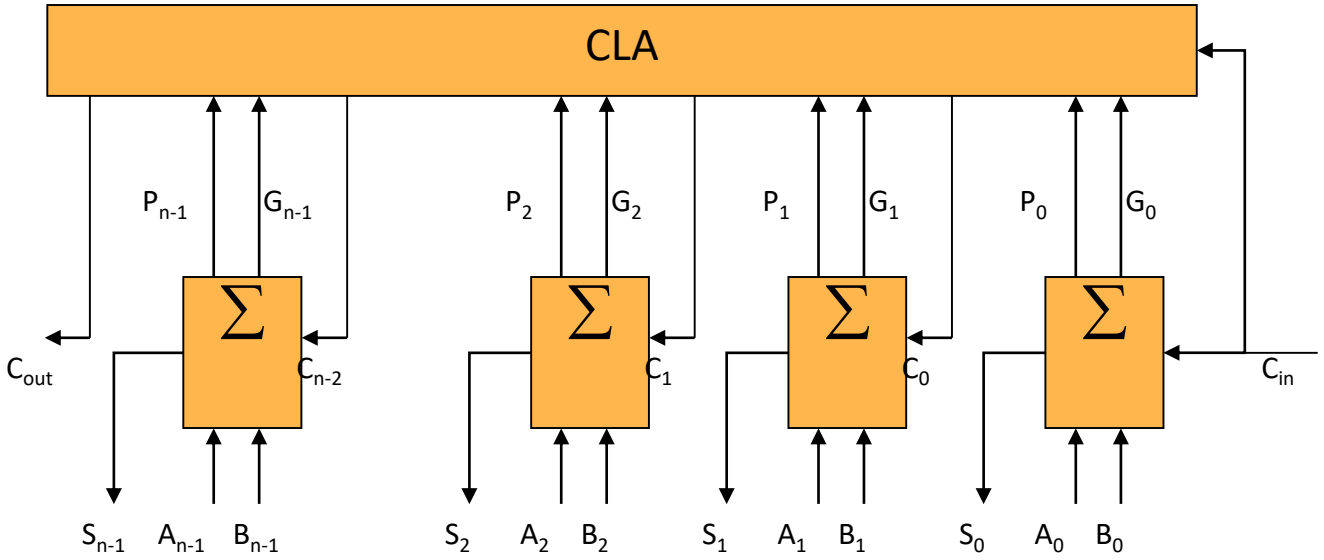
P és G előállítása az összeadókban

A gyakorlatban alkalmazott másik módszer szerint minden egyes teljes összeadónak a C_i carry output vonalát helyettesítik a két carry generate G_i és propagate P_i jelt előállító áramkörrel.

Két kapuval többet tesznek be a tokba, ennek nincs semmi akadálya.

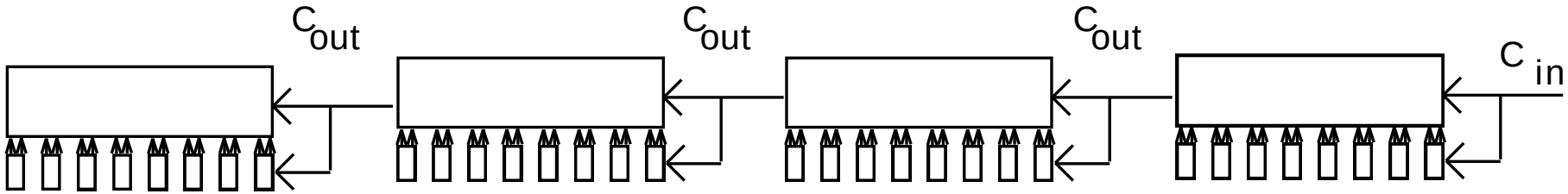
Itt a művelet három lépésben zajlik:

1. az összeadókban bit-helyiértékenként párhuzamos összeadás valamint P és G képzés;
2. a CLA-ban előállítjuk egyszerre az összes carry-t:
3. az összeadókban bit-helyiértékenként párhuzamosan hozzáadjuk a kapott összegekhez

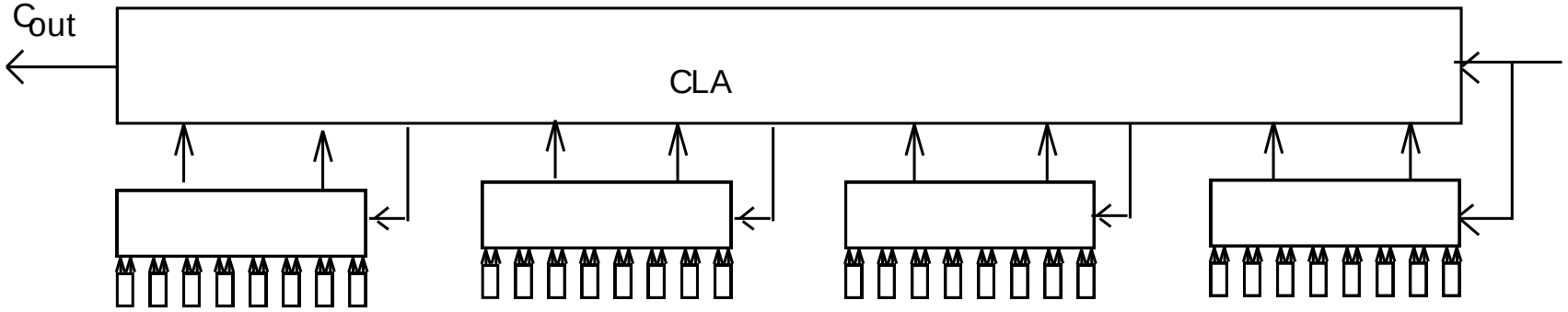


32 bites összeadó

Egy CLA 8-bites, hogy 32-bites szót tudjunk feldolgozni, ezért négy kell belőle.



A carry sorosan terjed a 8-bites egységek között. Ezért eggyel magasabb szintre hozzárendelünk egy ugyanolyan carry előrejelzőt.



Áramkörileg a két felső szint teljesen megegyezik. A gyakorlatban ezt használják.

Fixpontos kivonás

A kivonás művelete az összeadáshoz hasonlóan végezhető el, ha biztosítjuk, hogy a kisebbítendő helyére a nagyobb, a kivonandó helyére pedig a kisebb szám kerüljön. Ellenkező esetben ugyanis az eredmény nem lesz helyes, ha bitenként formailag helyesen is végezzük el a műveletet. A kivonás megkezdése előtt tehát a két számot össze kellene hasonlítani és ennek eredményétől függően kellene azokat beírni a megfelelő regiszterekbe. Ez nem hatékony eljárás.

Az aritmetikai műveleteket a számítógépben előjeles számok között kell végezni. Ezért az ismertetett eljárásokkal nagyon gyakran kellene komparálást végezni, és ettől függően összeadást vagy kivonást végezni.

Ennek elkerülésére célszerű az előjeles számokat olyan kódban ábrázolni, amely lehetővé teszi, hogy az aritmetikai alapműveleteket - így a szorzást és osztást is - az előjeltől függetlenül összeadási lépésekre vezessük vissza.

Ilyen kódok

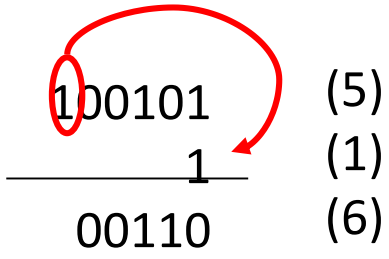
- egyes komplemens
- kettes komplemens
- többletes kód

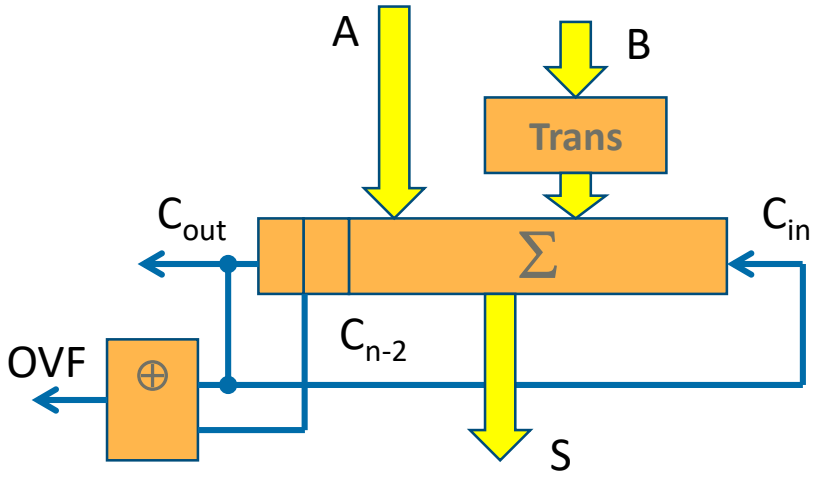
Műveletvégző egyes komplemens kódban

A negatív szám egyes komplemensének képzése a Trans egységgel történik. Az előjelbitet a többi bittel azonos módon kell kezelni, így az eredményben keletkezett előjelbit szintén helyesen jelzi az eredmény előjelét. Ha az eredmény negatív, akkor az előjelbitje 1 és az abszolút értéke az egyes komplemens képzés útján állapítható meg. 7 egyes komplemense, az előjelbittel együtt: 11000 (-7)

A	13	01101
B	+(-7)	11000
	(+5)	100101

Az eredmény nem helyes. A helyes eredmény érdekében a legnagyobb helyiértéken keletkező átvitelt adjuk hozzá a legkisebb helyiértékhez :



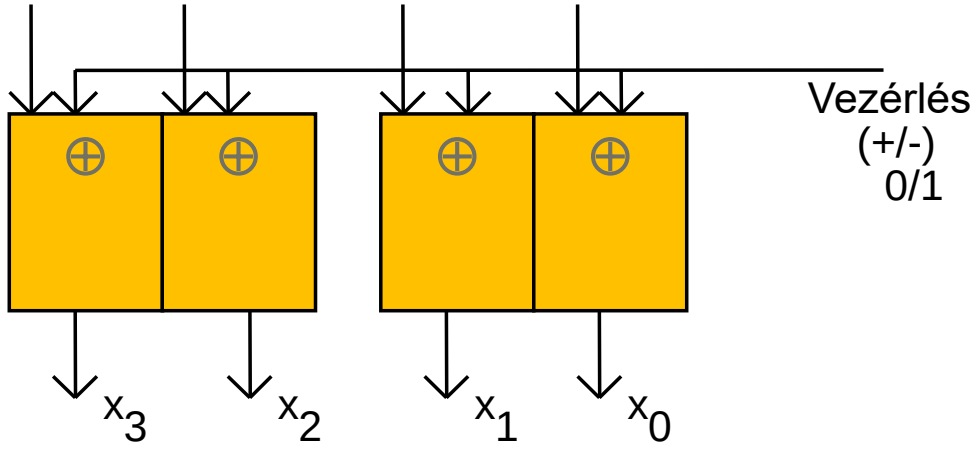


A TRANS egység

A Trans egység vezérlése az Opkód megfelelő bitje és a kivonandó előjel bitje alapján történik.

Vezérlés +/-	Input B	Output X
0	0	0
0	1	1
1	0	1
1	1	0

B bitjei



Túlcsordulás

Bármilyen is a számábrázolás, előfordulhat, hogy egy aritmetikai művelet eredményeként olyan számot kapunk, amely nem ábrázolható annyi helyiértéken, amennyi az adatszóban rendelkezésre áll. Ebben az esetben a kapott eredmény természetesen nem helyes, ezért ezt az ALU-nak jeleznie kell. Ezt a jelenséget túlcsordulásnak nevezzük.

$$OVF = C_i \oplus C_{i-1}$$

A legnagyobb helyiértékből kifolyó és a legnagyobb helyiértékbe befolyó carry XOR (kizáró VAGY) kapcsolata.

A túlcsordulás jelzésére egy flag-et alkalmaznak. Amikor egy összeadási vagy kivonási művelet túlcsordulást eredményezett, megszakítás következhet be. A programozó feladata, hogy döntsön a szükséges intézkedésekről.

Pozitív számok összeadása

A. az első bit az előjel

01000	pozitív
01100	pozitív
10100	negatív?

B. az első két bit az előjel

001000	pozitív
001100	pozitív
010100	?

$0 \oplus 1 = 1$: van túlcsordulás

Negatív számok összeadása

A. az első bit az előjel

10000	negatív
10100	negatív
00100	pozitív?

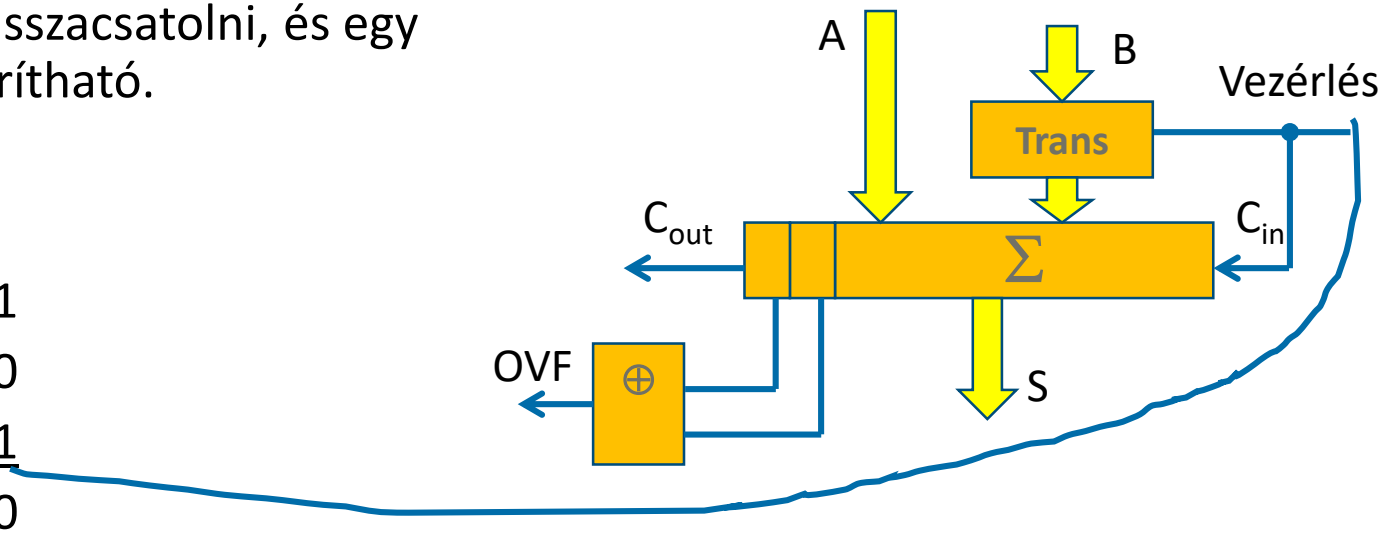
B. az első két bit az előjel

110000	negatív
110100	negatív
100100	?

$1 \oplus 0 = 1$: van túlcsordulás

Kettes komplemens kód

A legkisebb helyiértéken az egyes hozzáadását a kettes komplemens képzésekor az összeadás művelettel együtt hajtjuk végre. Így a legnagyobb helyiértéken keletkező átvitelt nem kell visszacsatolni, és egy összeadás művelet megtakarítható.



A	13	01101
B	+(-7)	11000
	+1	1
	(+6)	000110

Az egyes komplemens-képzéshez képest két különbség:

- az egyes komplemens képzés helyett mindenütt kettes komplemens képzést kell alkalmazni;
- nem kell az eredményt átvitel-módosítással korrigálni.

Fixpontos szorzás/osztás

A szorzás és az osztás jóval komplexebb művelet, mint az összeadás és a kivonás. A végrehajtási idejük jóval hosszabb, mint az egyszerű összeadás. Ennek oka, hogy a megvalósításuk során az ALU-ban egy összeadási/kivonási - léptetési sorozat zajlik, amit a mai gépeken mikroprogram vezérel.

A nagyteljesítményű gépeken hardver úton megvalósított szorzókat és osztókat használnak, hogy növeljék az aritmetikai műveleti sebességet. Mivel az ADD, SUBTRACT és SHIFT elérhető gépi utasítás formájában, ezért mind a szorzást, mind pedig az osztást meg lehet valósítani szoftver úton, program-eljárás segítségével is.

- olcsó gép — gépi kódú eljárás
- közepes gép — mikroprogram
- gyors, drága gép — áramköri szorzó és osztó

POWERPC

- 2 db fixpontos műveletvégző az egyszerű műveletekhez (+,-)
- 1 db fixpontos műveletvégző az összetett műveletekhez (*,/)

Pentium

- fixpontos összeadó/kivonó
- fixpontos szorzó
- fixpontos osztó

Bináris szorzási algoritmusok

C=A*B

7*10=70 $7_h * C_h = 46_h$

a) a szorzó legkisebb helyiértékű jegyével szorzunk, majd a szorzatot balra eltolva összeadjuk.

```
 0111*1010
 0000
 0111
0000
0111
1000110
```

b) algoritmikusabb: ciklikusan ismételve a szorzást és a részletszorzatnak egy gyűjtővel való összeadását

```
 0111*1010
 0000
 0000
 0000
 0111
 01110
 0000
 001110
 0111
1000110
```

c) amennyiben a szorzó LSB 0, nem adunk össze, csak léptetünk

Input A	Input B	Output C
0	0	0
0	1	0
1	0	0
1	1	1

Bináris szorzás időigénye

A szorzás ciklusmagját annyiszor ismételjük, ahány jegye van a szorzónak. Hasonlítsuk össze a decimális és a bináris szám hosszát

A decimális helyiértékek száma	Példa	A bináris helyiértékek száma	Példa
1	9	4	1001
2	99	7	1100011
3	999	10	11011001001

A 999-as szorzó binárisan tíz számjeggyel írható fel, így a ciklus három helyett tízszer fut, azaz több időt vesz igénybe. Ez rögtön rámutat a szorzás jelentős időigényére is, hisz elméletben annyi összeadást hajtunk végre, ahány egyes van a szorzóban, és annyi léptetést, ahány jegyű a szorzó.

Bináris szorzat hossza

A szorzás elvégzésekor az eredmény maximum kétszerese az operandusok hosszának.

Operandusok	A decimális helyiértékek száma			Általános forma	Bináris helyiértékek száma
A	1	1	2	m	4
B	1	2	2	n	7
C	2	3	4	max. m+n	max. 11
Példa	9x9	9x99	99x99		

Maguk az operandusok szóhosszúságúak. Architekturálisan a regiszterek az adatszó hosszával egyeznek meg, az eredményt befogadó akkumulátornak (vagy más e célra használt) regiszternek kétszeres hosszúságúnak kellene lennie.
Az algoritmust tehát annyiszor hajtjuk végre, ahány jegye van a szorzónak, ezután erre az operandusra már nincs szükség. A feleslegessé váló szorzó-jegyek helyén tároljuk el az eredménynek már lépésenként elkészülő eredmény-helyiértékeit.

Előjeles számok szorzása

A számok abszolút értékes kódban vannak

- az abszolút érték előállítása az előjel eltávolításával, eredmény előjel (signum) meghatározása: $S=S_1 \oplus S_2$
- elvégezzük az előjel nélküli szorzást
- visszaállítjuk a helyes előjelet.

A módszer egyértelmű, csak bonyolult

- az előjel levágása és az
- új előjel visszaállítása

A számok kettes komplemens kódban vannak

Az algoritmus a nem inverz (egyenes) kódhoz hasonló, a különbség csupán annyi, hogy amikor a szorzó legmagasabb helyiértékű bitjével szorzunk, akkor a részletszorzatot nem hozzáadjuk, hanem levonjuk a részletösszegből.

A kettes komplemens kódú ábrázolás úgy is tekinthető, mintha a legmagasabb helyértékű bit súlyozása negatív, a többié pedig pozitív lenne. Mivel azonban a szorzandó maga is lehet, hogy negatív szám, a részletösszeg jobbra léptetésekor az előjelbitre különös tekintettel kell lennünk, azaz a korábbi előjelbitet jobbra léptetjük, ugyanakkor a legmagasabb pozícióban változatlanul megismételjük ("csíkot húz maga után").

A szorzás gyorsítása

Bit-csoporttal történő szorzás

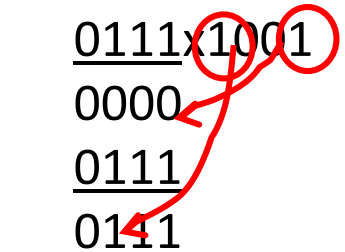
A szorzónak egyidejűleg nem egy, hanem több, például két egymás melletti jegyét vesszük figyelembe és egy-helyiértékes léptetés helyett több helyiértékes léptetést végzünk. A szorzó egymás melletti két bináris jegye 00, 01, 10 vagy 11 lehet:

- 00 nem kell a szorzandót a részösszeghez hozzáadni, csak kettőt léptetünk balra;
- 01 a szorzandó egyszeresét adjuk hozzá, majd kettőt léptetünk balra,
- 10 a szorzandó kétszeresét (egy helyiértékkel balra tolt értékét) adjuk hozzá, majd kettőt léptetünk balra
- 11 a szorzandó háromszorosát kell a részösszeghez hozzáadni, majd kettőt léptetünk balra. A gyakorlatban másképpen csinálják: néggyel szoroznak, majd kivonják belőle a szorzandó egyszeresét, így gyorsabb.

Példa

$(7x9)_{10} = 63$

$(0111x1001)_2$



7x9=63

A szorzás gyorsítása

Booth-féle algoritmus

Azért lassú egy szorzás, mert sok 1-es van a szorzó bitjeiben, és ez sok összeadást jelent. Az algoritmus próbálja dekomponálni úgy, hogy kevesebb egyes legyen benne. Nem működik minden számra. Azoknál jó, ahol hosszú, csupa egyes vagy csupa nullás csoportok vannak.

Például 62-vel kell szoroznunk: 0011 1110 = 62

Mivel 5 db egyest tartalmaz, ez 5 db egységnyi műveletet eredményez (az összeadást tekintve egységnek)

62 közel van a 64-hez $64-2= 2^6-2^1$

Így a szorzandót megszorozzuk 64-gyel, majd kettővel ,és vesszük a két eredmény különbségét

0100 0000 64
-0000 0010 -2

A binárisan végzendő műveletek:

- hatszoros léptetés balra;
- a szorzandó egyszeresének hozzáadása a gyűjtőhöz;
- egyszeres léptetés balra;
- a szorzandó egyszeresének hozzáadása a gyűjtőhöz;
- a két eredmény kivonása egymásból.

C=AxB B=X-Y

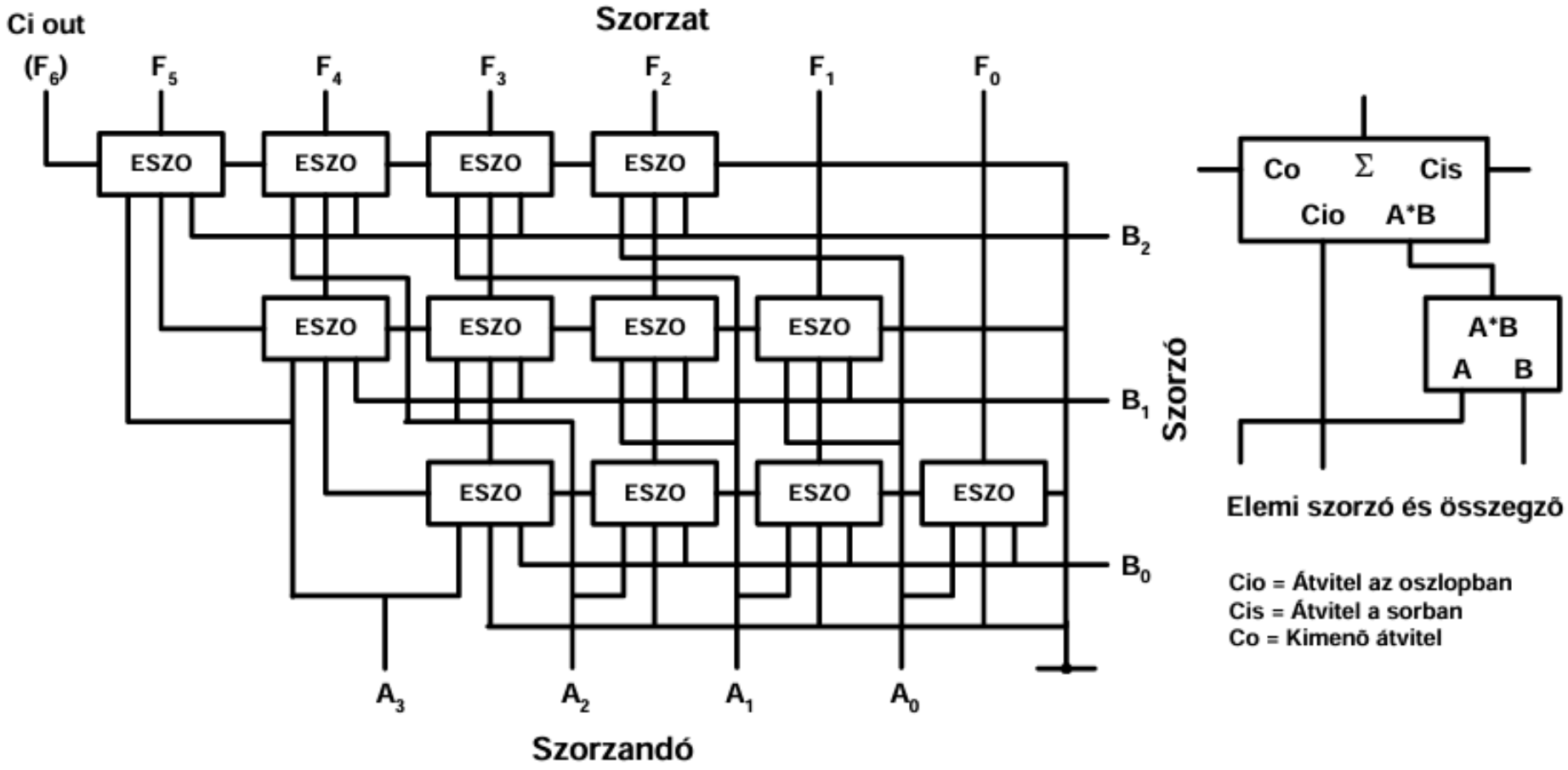
C=A(X-Y)

C=AxX-AxY

Mivel a léptetés rendkívül gyors, tehát összesen 2 db összeadást, majd 1 db kivonást végzünk, ami 3 db egységnyi műveletet jelent. Az eredeti megoldással szemben

40%-os gyorsítást sikerült elérnünk!

Hardver szorzó áramkör



Elemi szorzó és összegző

Cio = Átvitel az oszlopban
Cis = Átvitel a sorban
Co = Kimenõ átvitel

Forrás: Grosz Imre Élő digitronika

A hagyományos osztás kivonásra visszavezetve

Algoritmus

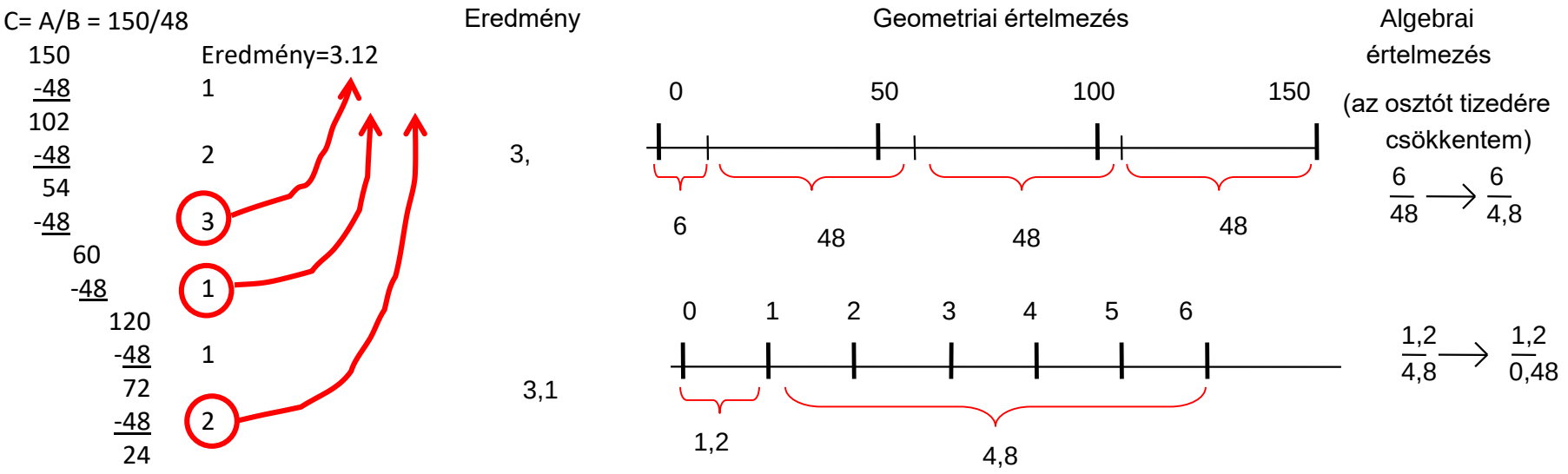
- megvizsgáljuk, hogy az osztandó nagyobb-e, mint az osztó, ha igen, akkor az osztót kivonjuk az osztandóból. Külön gyűjtjük, hányszor sikerült kivonni.
- amennyiben már nem teljesül a feltétel, akkor a gyűjtőből kiírjuk az eredményt, és az osztót tizedére csökkentjük .

Geometriai értelmezés

- legfontosabb alapelv, hogy az osztandóra elkezdjük **elölről** rámérni az osztót, és számoljuk, hányszor fér rá,
- majd pedig amikor már nem fér rá, az eredményt kiírjuk, az osztót tizedére csökkentjük, és ismét elölről rámérjük az osztót.

Példa

C= A/B = 150/48

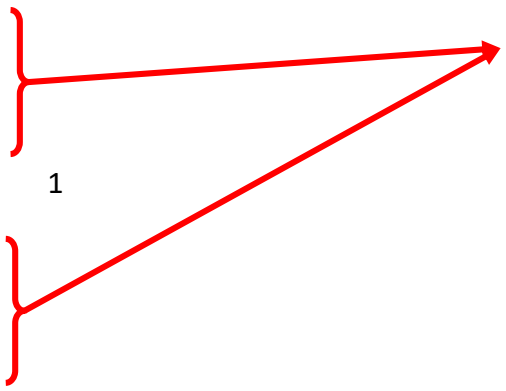


Visszatérés a nullán át (restoring)

Példa

C= A/B = 150/48

150		Eredmény=3.12
-48		1
102		
-48		2
54		
-48		3
6		
-48		
-42		
+48		
60		
-48		1
12		
-48		
-36		
+48		
120		
-48		1
72		
-48		2
24		



Az előző módszer hátránya, hogy minden lépést egy komparálás előz meg, ami rendkívül időigényes. Ezt kivonás és előjelvizsgálattal is kiválthatjuk. Amikor az előjel negatívvá válik, akkor meghívjuk a következő alprogramot:

- kiírjuk a gyűjtött eredményt,
- a levont osztót hozzáadjuk a maradékhoz,
- szorozzuk a kapott maradékot tízzel, majd

visszaadjuk a vezérlést a főprogramnak, és folytatjuk tovább az eljárást.

Az algoritmus hiányossága, hogy ciklikusan fölösleges műveleteket végzünk.

Visszatérés nélküli osztás (nonrestoring)

Ezt kivonás és előjelvizsgálattal is kiválthatjuk, és kétféle eljárást alkalmazunk:
amikor az előjel negatívvá válik (negatív maradék)

- kiírjuk a gyűjtött eredményt, ezt a lépést 0-nak tekintjük
- a maradékot szorozzuk tízzel, hozzáadjuk az osztót, ezt a lépést 9-nek tekintjük;
- majd ciklikusan hozzáadjuk az osztót, (visszafele számolva az eredményt), amíg csak előjelváltás nem történik.

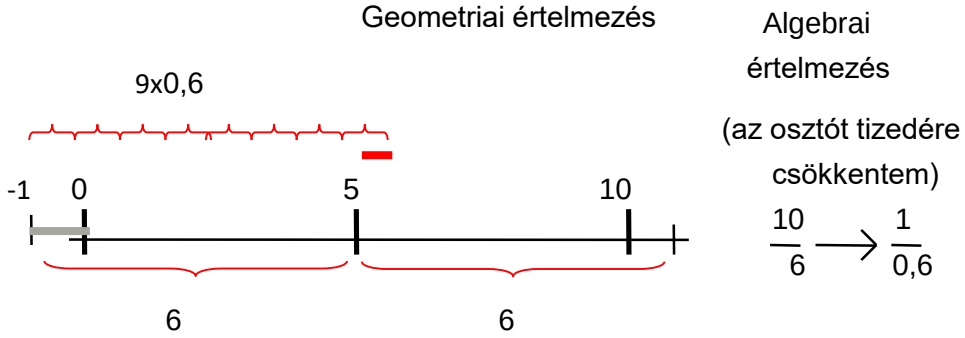
amikor az előjel pozitívvá válik (pozitív maradék)

- kiírjuk a gyűjtött eredményt,
- a maradékot szorozzuk tízzel;
- kivonjuk az osztót, ezt a lépést 1-nek tekintjük.
- majd ciklikusan kivonjuk az osztót, (előre számolunk), amíg csak előjelváltás nem történik.

Példa

C= A/B = 11/6

		Eredmény=1.83
+11		
-6		1
+5		
-6		0
-10		
+6		9
-4		
+6		8
+20		
-6		1
+14		
-6		2
+8		
-6		3
+2		
-6		
-4		0



Algebrai értelmezés

(az osztót tizedére csökkentem)

$\frac{10}{6} \rightarrow \frac{1}{0,6}$

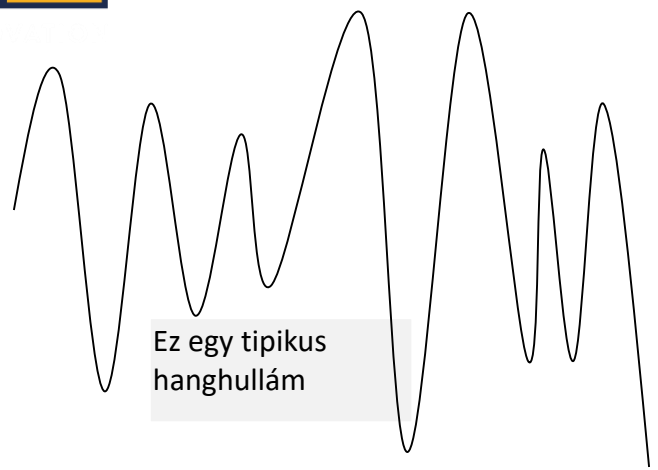
Geometriai értelmezés

Legfontosabb alapelv, hogy az osztandóra elkezdjük **hátulról** rámérni az osztót, tehát a negatív maradékot mérjük rá.

Mivel a negatív maradékra mérjük az osztó tizedét, ezért a végéről, ha egyszeri ráméréskor következik be az előjelváltás, akkor 9-tized valamennyi az eredmény, ha a második ráméréskor következik be az előjelváltás, akkor 8-tized valamennyi az eredmény.

Fixpontos multimédiás feldolgozás

Hangfeldolgozás



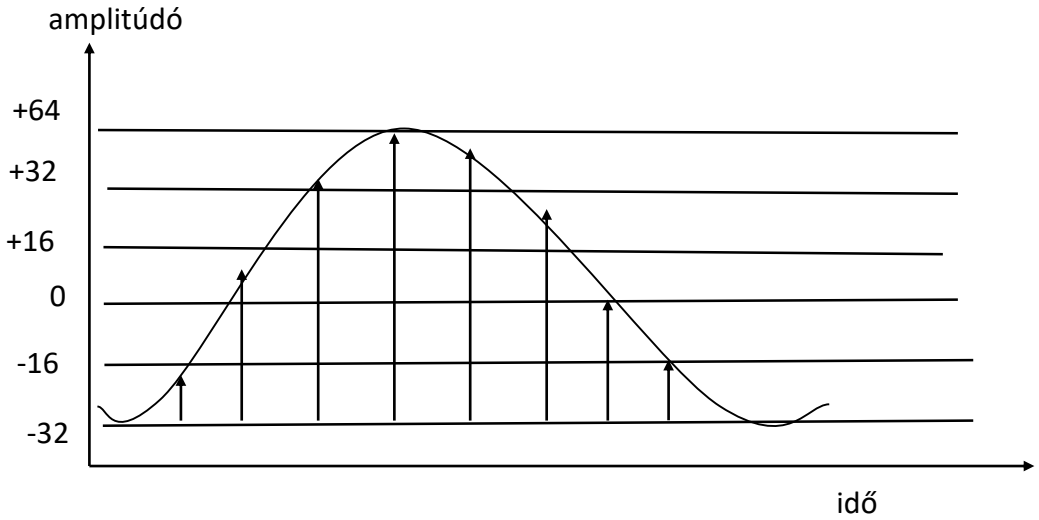
Ez egy tipikus hanghullám

A hanghullám eredetileg **analóg** jellegű, ezt először digitálissá kell konvertálni annak érdekében, hogy a számítógépen feldolgozható legyen. Az analóg hullámból meghatározott időnként mintát vesznek.

A minta amplitúdóját egy elektronikus **mintavételező és tartó** áramkör határozza meg, amely kimeneti értékét azután egy **A-D**, azaz analóg-digitális **konverternek** nevezett áramkör **digitális** kóddá alakítja. Így alakul át az analóg jel digitálissá.

Amplitúdó (y tengely)
Felbontás, azaz hány pontra bontjuk (x tengely) – **mintavételi frekvencia**
Az analóg hullám pozitív szélső értékéhez rendelhetjük a legnagyobb ábrázolható pozitív számot, a negatív szélső értékéhez pedig a legnagyobb ábrázolható negatív számot.

8 bit	256 hangszint
16 bit	64k hangszint
24 bit	kísérleti stádiumban



ÓE 15 145 630

A **mintavétel gyakoriságát** úgy kell megválasztani, hogy az analóg jel minél kisebb változásait is le tudjuk képezni, tehát a mintavételi frekvencia legalább a jelfrekvencia kétszerese kell legyen – **Shanon-Nyquist kritérium**.

Alkalmazás	Mintavételi frekvencia (kHz)
Digitális telefon	8
Audio CD	44,1
DVD Audio Codec '97 Professional audio recording	48
DVD minőségi	96

A feldolgozandó adatok jellege és mennyisége

A felbontás és a mintavételi sebesség határozza meg azt az adatmennyiséget, amit a digitalizálási folyamat során előállítunk és rögzítünk. Sztereo esetén ez a mennyiség megduplázódik.

A 44,1 kHz mintavételi sebességgel dolgozó 16-bites audio CD ilyen módon minden másodpercben
44100 x 2 byte-ot dolgoz fel
44100 x 2 byte/s = 88200 byte/s ~ 86 kB/s mono esetén,
sztereo esetén: 86x2 = 172 kB/s x 60 = 10 MB/perc.

A hangok digitális feldolgozása tehát két vonatkozásban jelent mennyiségi erőpróbát, ugyanis hatalmas mennyiségű fixpontos adatot kell

- tárolni, esetleg Interneten átvinni és
- feldolgozni.

A hangadatokat fixpontosan ábrázoljuk. A lebegőpontos ábrázolás jóval nagyobb értelmezési tartományt biztosítana. A hang- és beszédfeldolgozás real-time történik, ezért igen fontos a feldolgozási sebesség. Míg a gyors integer feldolgozás viszonylag olcsón megvalósítható, a gyors lebegőpontos feldolgozásnak később teremtődtek meg a hardver-feltételei. A hangfeldolgozásra DSP processzorokat használnak.

Képfeldolgozás - bittérképes vagy raszteres megjelenítés

A képet ábrázolhatjuk a memóriában, a képernyőn, papíron is, mint egy mátrixot, azaz egy kétdimenziós tömböt, melynek minden egyes eleme egy **pixel** jellemzőit tartalmazza. Például, az elterjedt képernyők 800x600 pixelt tartalmaznak, a mai jó felbontásúak 1280 x1024 pixelt.

A színes képeket háromféle elemi színből állítjuk elő: RGB, tehát minden egyes pixelhez három számértéket kellene tárolni. Ennek kiküszöbölésére a színskálát a gyakorlatban kódolják, és a színkódnak megfelelő szín a színfordítási táblázat segítségével értelmezhető a szín megjelenítéséről gondoskodó periféria (nyomtató, képernyő) által. Jjelelnleg az RGB-n kívül más színskálákat is alkalmaznak.

- 1 biten csak azt tudjuk megmondani, hogy az adott pixel világos legyen vagy sötét.
- 8-biten már 256 féle színkódot tudunk ábrázolni, ami még nyers színábrázolást biztosít.
- Ma a tipikus alkalmazások (high color) 16-bites színskálát használnak
- Az úgynevezett igazi színes (true color) pedig 24 bitest.
- Létezik már 32-bites rendszer is, azonban ennél az alfa-csatornának nevezett 8 többletbiten már nem színeket, hanem effektust, a pixel átlátszósági mutatóját tárolják

	Memória igény (Byte)	
Felbontás (pixel)	**8 bites színskála**	**16 bites színskála**
800x600	480 000	960 000
1280x1024	1 310 720	2 621 440

Műveletek a pixeles képeken (képfeldolgozó utasítások)

Tipikus műveletek

- a bit-blokk átvitel
- a rajzolás-festés
- napjainkban az ablakkezelés.
  Minden megnyitott ablakot egy bit-blokként kezel a gép. Így az ablak-műveletek nem bájtonként, hanem blokkonként értelmezettek, tehát sokkal gyorsabbak.

A feldolgozandó adatok jellege és mennyisége

- A képadatok tárolására a legalkalmasabb a fixpontos (integer) adat-ábrázolás;
- A felbontás és a színskála határozza meg azt az adatmennyiséget, amit a digitalizálási folyamat során előállítunk és rögzítünk.

Egy 1280x1024 pixeles képernyő pillanatnyi tartalma 16-bites színskála esetén 2,5 MB memória felhasználásával definiálható.

A raszteres képek digitális feldolgozása tehát két vonatkozásban jelent mennyiségi erőpróbát, ugyanis hatalmas mennyiségű fixpontos adatot kell

- tárolni, esetleg Interneten átvinni és
- feldolgozni.

A fixpontos multimédia feldolgozási irányai

Adattárolás és -átvitel

- A digitális multimédia adatok természetesen tömöríthetők, mint bármely más adat. A tömörítést igen sokféle algoritmussal elvégezhetjük. A gyakorlatban elterjedt az MPEG (Moving Picture Expert Group) szabvány, mely a hang és a kép tömörítését egyaránt szabályozza.

Hangtömörítés

- Bár az MPEG szabvány alapvetően videóra vonatkozik, de leírja a filmeket kísérő audio-szabványt is. Hang vonatkozásában sztereót tesz lehetővé.
- Az MPEG-nek több szintje van, minél nagyobb a szám annál komplexebb és eredményesebb a tömörítés. A szám közömbös a minőség szempontjából, hiszen mindegyik a 32, a 44 és a 49 KHz-es mintavételi sebességet engedi meg. Az MPEG szabvány nem definiálja a tömörítési algoritmust. Minden MPEG állomány egy fejjel indul, mely tartalmazza a tömörítési szintet és a tömörítési módszert.
- a tömörítés nem igényli a real-time feldolgozást, sőt a gyakorlatban sohasem fut valós üzemmódban, míg
- a kicsomagolásnak a lejátszás alatt mindig valós üzemmódban kell

A fixpontos multimédia feldolgozási irányai

Képtömörítés

JPEG

- Igen jó minőségű képtömörítést biztosít. Jól tömörít, nem romlik a képminőség, rétegelt továbbítást biztosít: a kontúrokat gyorsan átküldhetjük vele, amit majd folyamatosan finomít. Egy sor tömörítési technika használatát engedélyezi, minden tömörítési részlet a kép file fejében található, ami elsőként kerül átküldésre.

MPEG

- Az MPEG csak a video-képek különbségét tárolja, így ér el igen nagyfokú tömörítést. Az MPEG-1 1992. októberében vált nemzetközi szabvánnyá. A Pentium MMX processzorok képesek real-time dekódolásukra.
- Az MPEG-2 jobb képfelbontást (720x480) és surround hangot biztosít.
- Az MPEG-3 tovább javítja a képfelbontást: 1920x1080, 30 Hz sebesség mellett.
- Az MPEG-4 pont ellenkező irányú, az igen alacsony átviteli sebességű alkalmazásokat célozza. Így mozgó képet lehet továbbítani 4800-64000 bit/s mellett, azaz a hagyományos modemeken. A képfelbontás kb. 176-144 pixel, 10Hz sebesség mellett. Például videofonok vagy videokonferenciák alkalmazhatják.

Multimédiás adatok feldolgozása PC-s környezetben

A multimédia feldolgozás számítástechnikai szempontból hatalmas mennyiségű fixpontos és lebegőpontos adat - általában valós idejű - feldolgozását jelenti.

Architekturálisan ezt a feladatot kétféleképpen valósíthatjuk meg:

multimédia segédprocesszorral

A grafikus segédprocesszor eredeti feladata a megjelenítő rendszer gyorsítása úgy, hogy az alapfeldolgozást végző mikroprocesszor kiegészítették egy olyan mikroprocesszorral, amit a video-orientált parancsok végrehajtására optimalizáltak. A Windows elterjedésével minden PC intenzív grafikus feldolgozást végez, ezért jelentősége megnőtt.

- az általános célú processzorunk multimédia célú bővítésével, az MMX kiterjesztéssel.
  Az MMX rövidítés a Matrix Math eXtension, vagy MultiMédia eXtensions-ként dekódolható. 1997. januárjában jelent meg a Pentium MMX processzorban.

A mai mobil és kliens processzorokban ezt a feladatot a grafikus processzor mag GPU tölti be.

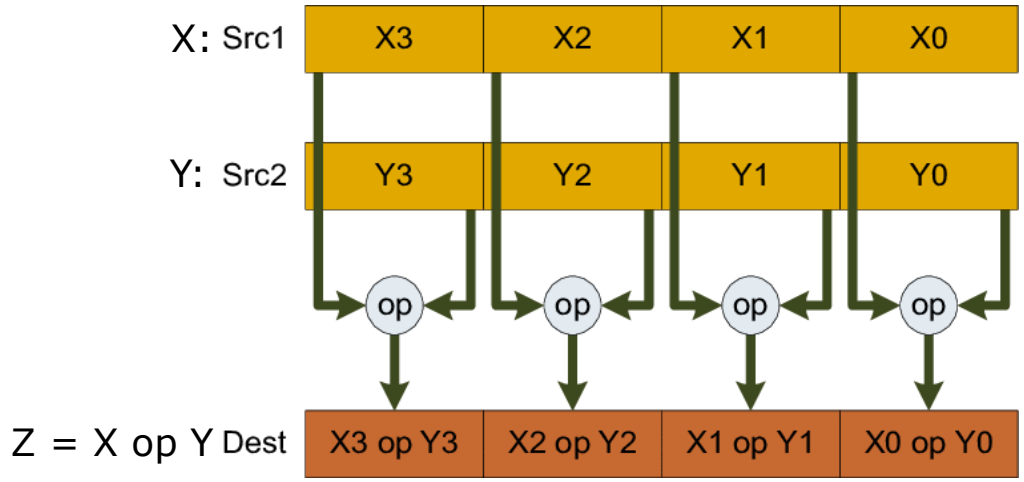
A fixpontos MMX feldolgozás

A pakolt adattípusok

Az Intel a Pentium MMX bevezetésével sajátos 64 bit hosszúságú MMX-csoportot hozott létre, annak érdekében, hogy ezzel is segítse a 64-bites belső adatsín hatékonyabb kihasználását.

Mértékegység	Bitek száma	Mezők száma
Pakolt bájt	bájt (8 bit)	8
Pakolt félszó	félszó (16 bit)	4
Pakolt szó	szó (32 bit)	2

Az MMX kiterjesztés a multimédia műveletek gyors végrehajtására szolgál. Mai szemmel nézve: a grafikus gyorsítót (graphic accelerator) leemeli a video-kártyáról és beviszi a processzorba. Az MMX egy viszonylag olcsó multimédia gyorsítást eredményez.



Összeadás pakolt adattípussal

Feladat: adjunk össze a memóriában két raszteres képet, hogy megjelenítsük az eredőjét. Mindkét kép minden pixeljét két bájt ír le a memóriában.

Megoldás hagyományos architektúrával

1. beolvassuk az első kép első bájtját a memóriából az akkumulátorba;
2. összeadjuk az akkumulátorban lévő első kép első bájtját a memóriában lévő második kép első bájtjával;
3. az eredményt kiírjuk az akkumulátorból az eredmény tárolására szolgáló memória-terület első bájtjaként;
4. megismételjük az 1-3 lépést annyiszor, ahány pixelből áll a kép.

Láttuk, hogy egy 800x600-as kép 960.000 byte-ot jelent. Ez nyilvánvalóan tetemes idő.

Összeadás pakolt adattípussal

Megoldás MMX architektúrával

Az MMX kiterjesztés 8 db 80 bit hosszúságú lebegőpontos regisztert használ 8 db 64 bit hosszúságú MMX regiszterként. Ezek felhasználásával a feladat megoldása a következő:

1. beolvassuk az első kép első nyolc bájtját a memóriából az MMX regiszterek egyikébe, az adatformátum pakolt bájt;
2. beolvassuk a második kép első nyolc bájtját a memóriából egy másik MMX regiszterbe, az adatformátum pakolt bájt;
3. elvégzünk egy pakolt bájt formátumú összeadást, ami fizikailag 8 db összeadó párhuzamos működését jelenti, melyek az eredményt az erre kijelölt MMX regiszterben képezik;
4. az eredményt kiírjuk ebből az MMX regiszterből az eredmény tárolására szolgáló memória-terület első nyolc bájtjaként;
5. megismételjük az 1-3 lépést annyiszor, ahány pixelből áll a kép osztva néggyel.

A feladatot tehát négyszer gyorsabban oldottuk meg, mert egyetlen utasítással több adat feldolgozását végezhetjük el, tehát ez a Single Instruction Multi Data (SIMD) feldolgozási kategóriához tartozik. Az MMX SIMD FX utasításkészlet tartalmazza a négy alapművelet és a logikai műveletek végrehajtására szolgáló utasításokat, a pakolt bájt, a pakolt félszó és a pakolt szó vonatkozásban. Az MMX kiterjesztés annyira sikeresnek bizonyult, hogy a Pentium II sorozatban már két futószalagon hajtják végre ezeket az utasításokat.

Lebegőpontos ALU

Műveletek az IEEE 754 lebegőpontos szabvány szerint

A szabvány szerint megvalósítandó műveletek

- négy alapművelet
- négyzetgyökvonás
- maradék-képzés
- bináris-decimális átalakítás
- kerekítés
- kivételek kezelése
- rejtett bit valóságossá alakítása
- karakterisztika többletes kódból kettes komplemenssé alakítása (opcionális)

A pontosságnak az LSB pozíción lévő bit értékének a felénél kisebbnek kell lenni. Ez azt jelenti, hogy legalább három őrző bitet kell alkalmazni.

Kerekítés

Bármely lebegőpontos művelet eredményének mantisszája egy hosszabb ALU-regiszterben keletkezik. Ezt amikor a memóriába töltjük (store) akkor kerekítenünk kell.

- **legközelebbire való kerekítés**
  Az eredményt a legközelebbi ábrázolható számra kerekíti. Például, amennyiben az őrző bitek értéke 101, akkor az nagyobb, mint az LSB bit értékének a fele. Ekkor tehát az LSB-hez hozzá kell adni 1-et;
- **kerekítés a pozitív végtelen felé - kerekítés a negatív végtelen felé**
  A lebegőpontos műveletsorozat végén a hardver-korlátok okozta kerekítések miatt nem ismerhetjük a pontos eredményt. Amennyiben viszont minden műveletet kétszer hajtunk végre, egyszer fölfele, egyszer pedig lefele kerekítve, akkor két eredményt kapunk, és biztos, hogy a kettő között helyezkedik el a helyes eredmény. Ha a két eredmény közötti különbség csekély, akkor elég pontos az eredményünk. Amennyiben viszont nagy, az eredmény nem exakt.
- **nullára kerekítés**
  Gyakorlatilag az őrző bitek egyszerű levágását (trunc) jelenti. Ez azt eredményezi, hogy a mantissza a pontos értéknél mindig kisebb vagy azzal egyenlő lesz.

A kivételek kezelése

Kivételek felbukkanása szoftver-megszakítást eredményez, melynek kezelését a felhasználó végzi:

- **nem egyértelmű művelet** - például: egy negatív szám négyzetgyökvonása;
- **nullával való osztás** - amikor az osztandó egy véges nem nulla szám, az osztó pedig nulla, akkor az eredmény pozitív végtelen;
- **alulcsordulás** - állapotjelző beállítása és az eredmény értékének nullára konvertálása vagy a denormalizált szám ábrázolása.
- **túlcsordulás** - állapotjelző beállítása és egyidejűleg vagy a legmagasabb létező érték beállítása vagy előjeles végtelen ábrázolása.
- **nem exakt kivétel** - amikor egy művelet kerekített eredménye nem exakt, ez a kerekítési szabályokból eredő

Műveletek lebegőpontos számokkal

A lebegőpontos aritmetikai műveletek a mantisszán és a karakterisztikán végzett többszörös fixpontos műveletekre vezethetők vissza.

Legyen A és B a két lebegőpontos szám, ahol m_A és m_B a normalizált mantisszát, k_A és k_B a karakterisztikát jelöli. Az r pedig a radix, a számrendszer alapja.

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

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

Összeadás/kivonás

Algoritmus

- k_A és k_B karakterisztikák összehasonlítása;
- a kisebb karakterisztikájú szám léptetése jobbra, miközben növeljük a karakterisztika értékét, amíg a két karakterisztika meg nem egyezik;
- a mantisszákon az összeadás/kivonás elvégzése;
- ha szükséges, az eredmény normalizálása.

Példa: összeadunk két számot

$0{,}900 \times 10^2$

$0{,}993 \times 10^4$

Először is azonos kitevőjű alakra hozzuk őket, azaz a kisebb karakterisztikájú számot léptetjük jobbra:

$0{,}900 \times 10^2$

$0{,}090 \times 10^3$

$0{,}009 \times 10^4$

Amikor a karakterisztikák már azonosak, elvégezzük az összeadást.

$0{,}009 \times 10^4$

$0{,}993 \times 10^4$

$1{,}002 \times 10^4$

Végül, ha szükséges, mint az adott esetben is, az eredményt nullára normalizáljuk.

$0{,}100 \times 10^5$

Szorzás

$$C = A \times B$$

$$C = \pm m_A \cdot m_B \cdot r^{k_A + k_B}$$

Osztás

$$C = A/B$$

$$C = \pm m_A / m_B \cdot r^{k_A - k_B}$$

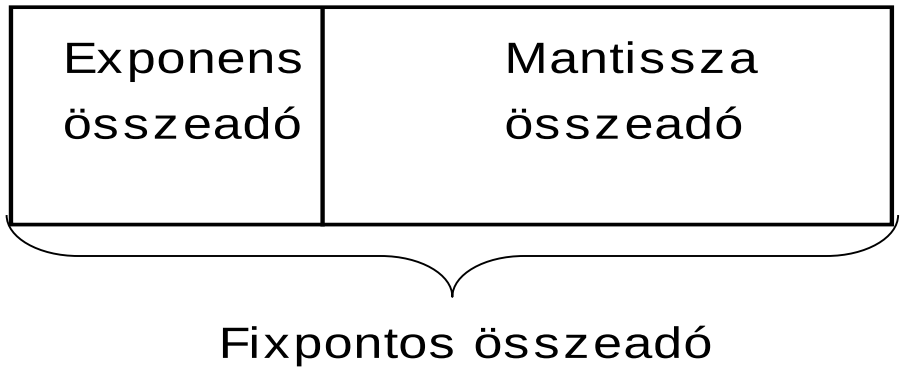
Algoritmus

- a mantisszák előjelhelyes osztása
- a karakterisztikák kivonása
- ha szükséges, normalizálás végrehajtása.

A mantissza és a karakterisztika műveletei párhuzamosan végezhetőek. Amíg a karakterisztikák összeadása/kivonása gyors, a mantisszák szorzása/osztása lassú, ezért érdemes áramköri szorzót alkalmazni.

Kombinált (univerzális) aritmetikai egység

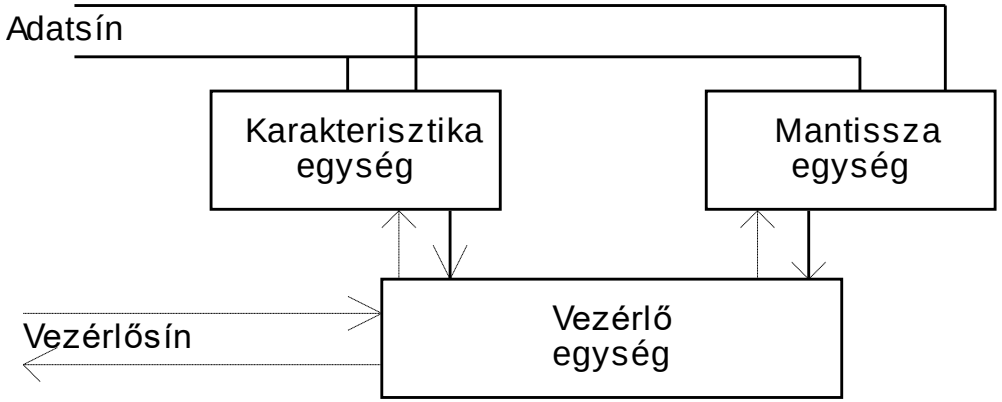
A fixpontos és a lebegőpontos aritmetika egyetlen aritmetikai egységben megvalósítható. A vezérlőegység ebben az esetben bonyolult lesz. Lényegében ez egy egyszavas fixpontos aritmetikai egység, amelyben mind a regiszterek, mind pedig az összeadó feloszthető karakterisztikára és mantisszára, amikor lebegőpontos műveletet kell végrehajtani



A tipikus megoldás azonban, amikor a karakterisztika is és a mantissza is külön-külön regiszterben van tárolva, egymás után végrehajtjuk velük az utasítás által előírt műveleteket és azok befejeződése után a letöltésük során újraegyesítjük a kettőt.

Dedikált lebegőpontos egység

A lebegőpontos aritmetikai egység megvalósítható két, fixpontos aritmetikai áramkör: egy karakterisztika egység és egy mantissza egység laza kapcsolata révén.



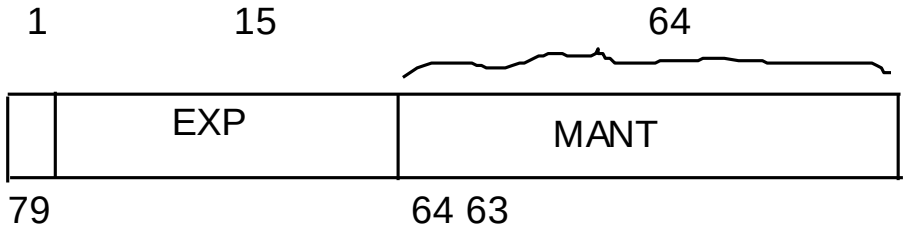
- a mantissza egységnek egy általános célú fixpontos aritmetikai egység valamennyi műveletét tudnia kell, tehát a szorzás és osztást is.
- egy egyszerű, csupán összeadni és kivonni tudó áramkör elegendő viszont a karakterisztika egységhez;
- a két egység működhet párhuzamosan. Ebben az esetben viszont a bonyolultabb, s egyben az időigényesebb feldolgozást végző mantissza egységet igen gyors elemekből kell építeni.

Megvalósítási esettanulmány
Az Intel család

Az 1981-ben megjelent Intel 8087 lebegőpontos segédprocesszor volt az IEEE szabvány egyik első megvalósítása, még a szabvány 1985-ös megjelenése előtt.

- radix 2;
- a szabványnak megfelelően kezeli az alulcsordulást és a túlcsordulást;
- létezik a rejtett bit
- használja az őrző biteket.

A formátumok közül mindkét szabványos formátum (az egyszeres és a kétszeres pontosság) megvalósításra került. Ez a két formátum csupán a memóriában és a háttértárolón létezik. Mihelyt egy ALU regiszterbe kerül a szám, azonnal átalakításra kerül bővített pontosságú számmá:



Hossza: 10 bájt (80 bit),

- mantissza 64 bit. A mantisszához tartozik a nem ábrázolt, de benne foglalt egyes
- karakterisztika 15 bit. 16.383 többletes az exponens értéke.

Értéktartománya: $3{,}4*10^{-4932} \leq x \leq 1{,}2*10^{4932}$

Pontossága 19 decimális számjegy.

A Checkit teljesítményteszt Math benchmarkja

Processzor	Relatív teljesítmény	Megjegyzés
8088	1	
386, 25 MHz	18	8087 segédprocesszor külön lapkán (kezdeti ára 180 eFt)
486dx, 66MHz	1730	általános és lebegőpontos segédprocesszor egy lapkán
Pentium, 133	5974	futószalagos műveletvégzés

Lebegőpontos multimédia feldolgozás

Lebegőpontos multimédia feldolgozás

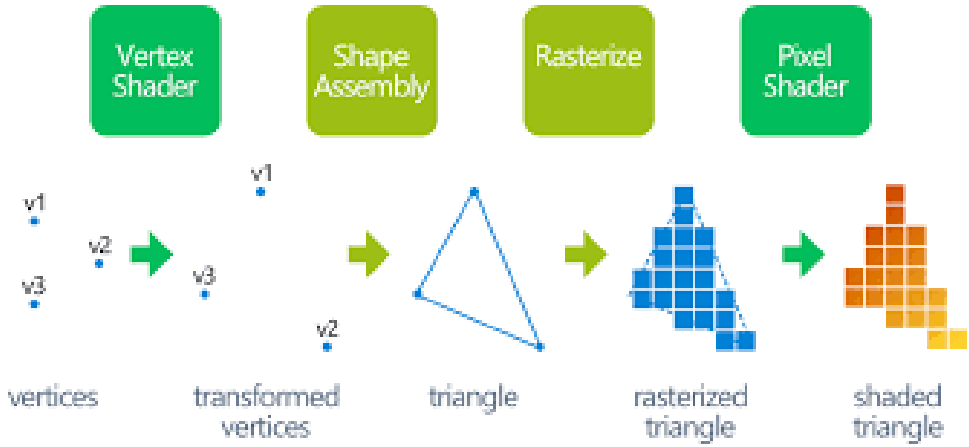
2D vektoros megjelenítés

A vonalakkal, görbékkel grafikailag körvonalazható képek geometriailag definiálhatók. Ezen esetekben elegendő az egyes objektumok geometriai jellemzőinek tárolása.

Egy komplex objektumot sok, kicsi, matematikailag könnyen leírható elemekre, sokszögekre, leggyakrabban háromszögekre bontják és a számítógép ezen háromszögeket dolgozza fel.

Műveletek

- mozgatás
- kicsinyítés – nagyítás
- forgatás
- torzítás (például, egy ellipszist készíthetünk egy körből úgy, hogy a függőleges és a vízszintes méreteit különféleképpen írjuk le)
- zárt objektumokat kitöltése színekkel
- objektum elemek összekapcsolása egy komplexebb objektummá



A feldolgozandó adatok jellege és mennyisége

Kevés adatot kell tárolni, de azokon sok számítást hajtunk végre: a jellemzőket tehát lebegőpontos formátumban kell ábrázolni. Meghatározóvá válik a lebegőpontos feldolgozási sebesség.

3D képek és mozgatásuk

A 3D képek olyan vektoros 2D képek, melyhez a térhatás érdekében egy **harmadik dimenziót** adunk hozzá.
Egy komplex kép akár 20.000 sokszögből is állhat, amit a **mozgási effektus** érdekében másodpercenként minimum 15-ször kell átszámolni, azaz újra előállítani. A grafikus gyorsítónak tehát ebben az esetben 300.000 sokszöget kell interpretálnia másodpercenként.
A valós objektumok árnyaltabbak, fény-árnyék hatás élénkíti és tompítja a színeket. Ezt a számítógép úgynevezett **shadereléssel** éri el. A kép előállítását **renderelésnek** nevezzük.

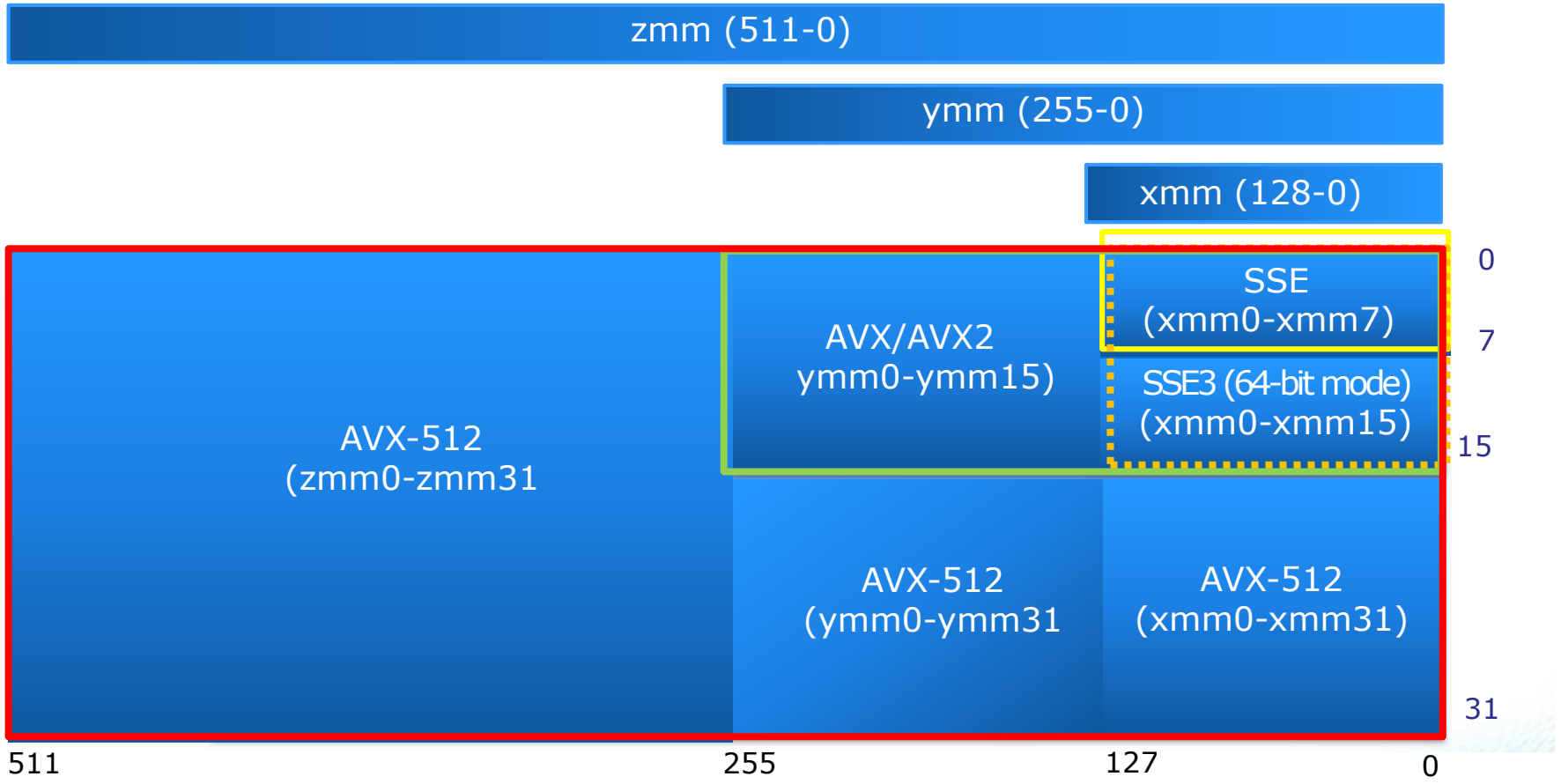
Ezen kívül biztosított a perspektivikus ábrázolás, a távolabbi objektumok kisebbek, a párhuzamosok pedig a végtelenben összetartanak. A valósághűbb ábrázolás érdekében az atmoszferikus perspektívát is alkalmazzák: a közeli objektumok tisztábban látszanak, a távolabbiak pedig fakóbbak és kékesebbek.

Az X86 ISA SIMD fejlődése

SIMD	Kiterjesztés	Processzor	Szószélesség	Év	Regiszter tér
64-bit FX	MMX	Pentium-MMX	32-bit	1997	8x64-bit MMX regiszter, FP regiszterek alkalmazásával
128-bit FP	SSE	Pentium III	32-bit	1999	8x128-bit Xmm regiszter
128-bit FX/FP	SSE2	Pentium 4 Willamette	32-bit	2000	
	SSE3	Pentium 4 Prescott	32-bit	2004	
	SSE3	Pentium 4 Prescott F-series	64-bit	2004	16x128 Xmm regiszter 64-bites módban
	SSE4	Core Merom (1st gen.)	64-bit	2006	16x128-bit Xmm regiszter
256-bit FP	AVX	Core Sandy Bridge (2nd gen.)	64-bit	2011	16x256-bit Ymm regiszter
256-bit FX/FP	AVX2	Core Haswell (4th gen.)	64-bit	2013	16x256-bit Ymm regiszter
512-bit FX/FP	AVX-512	Core Cannon Lake (8th gen.)	64-bit	2018	32x256-bit Zmm regiszter

Forrás: Sima

Lebegőpontos SIMD regiszterek fejlődése



Műveletek BCD számokkal

A BCD számok összeadása

A BCD kódban megadott számok bináris számrendszer szabályai szerint összeadhatók, de ha az összeadás eredménye 9-nél nagyobb, akkor a tízes számrendszer szabályai szerint kell képezni az átvitelt. Ezt korrekciónak nevezzük: az összegből $10_{10}=1010_2$ kell kivonni. Ennek kettes komplemensét: 0110_2 hozzáadjuk, azaz végső soron 6-ot hozzáadunk. Az eredmény előjelének megállapítása külön történik.

8	1000	
+ 7	0111	
15	1111	→ érvénytelen tetrád
	0110	
	10101	

átvitel

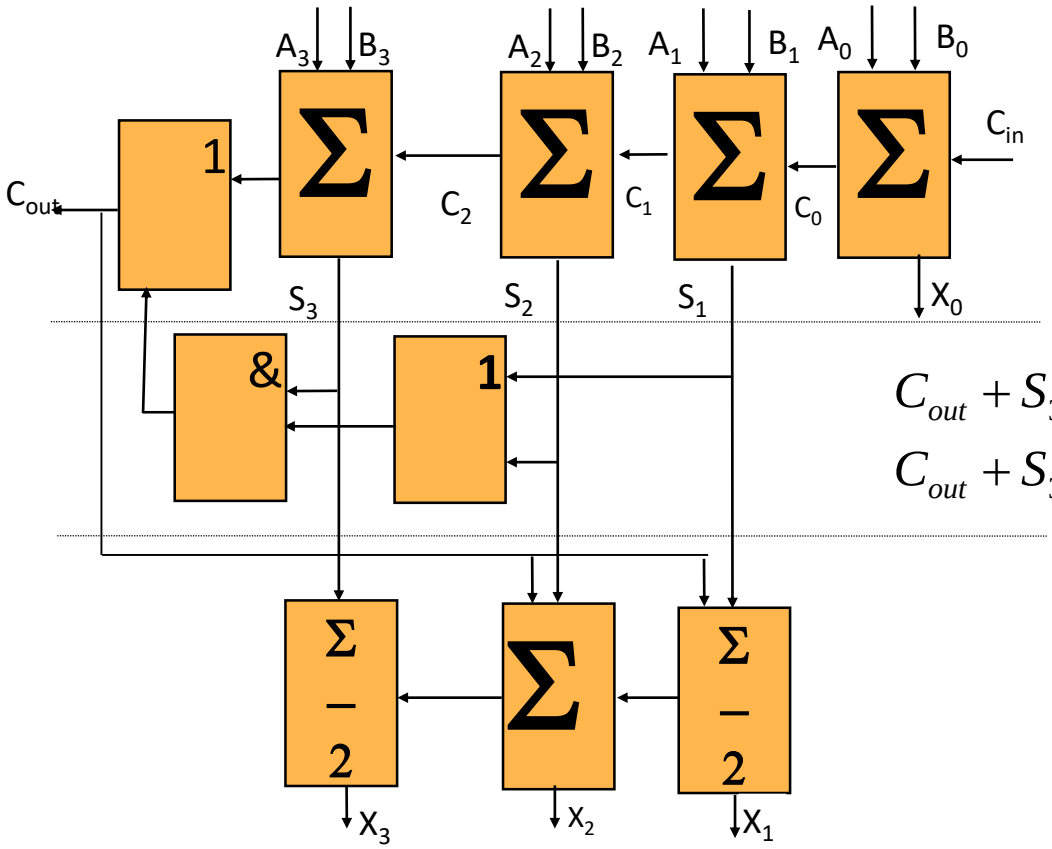
BCD összeadó

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

Érvényes tetrádok

Érvénytelen tetrádok

vagy ennél nagyobb, azaz van átvitel



$$C_{out} + S_3 \cdot \overline{\overline{S_2} \cdot \overline{S_1}} =$$

$$C_{out} + S_3 \cdot (S_2 + S_1)$$

Megvalósítás lehet

- dedikált
- univerzális műveletvégző egység

Az átvitel gyorsabb terjedése érdekében itt is használnak CLA-t

Esettanulmány
Intel processzorok

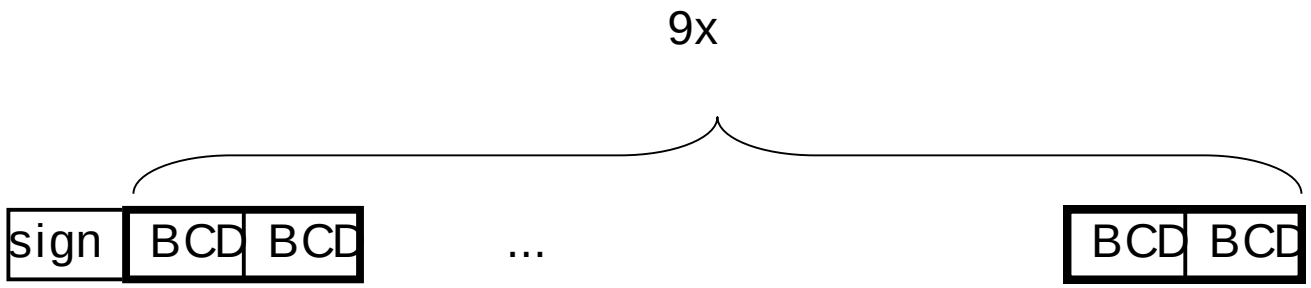
Zónázott ábrázolási módnál az alsó négy biten ábrázolja a BCD számokat, a felső négy bitre pedig 3 kerül. Műveleteket (+,-,*,/) is hajthatunk végre zónázott számokon.

Pakolt decimális: 80 bit hosszú (10 egymást követő bájt). A legfelső helyiértékű bájt legfelső bitje az előjel. Ha értéke 1, negatív a szám, ha 0, pozitív. A legfelső bájt többi bitje kihasználatlan. A többi bájt egyenként két BCD számjegyet tárol.

Értéktartománya:

18 db 18 db

$-9...9 \leq x \leq 9....9$



BCD aritmetika értékelése

Előnye:

- pontosság;

Hátránya:

- bonyolultabb a műveletvégzés, így drágább a műveletvégző és lassabb a műveletvégzés - a mai áraknál és teljesítményeknél ez a hátrány már elhanyagolható;
- rosszabb a memória-kihasználtsága, mint a bináris számoké: a BCD szám a 16 lehetséges kombinációnak csak kb. 60%-át, azaz 10-et használ ki. A mai memória-árak és méretek mellett ez a hátrány szintén elhanyagolható.

Pénzügyi programoknál lényeges az input adatok pontosságának megőrzése. Mivel napjainkban egyre meghatározóbbak az alkalmazói szempontok, így a BCD ábrázolásnak jelenleg az előnyei dominálnak és így perspektivikus az alkalmazása.

Érdekességek:

A BCD gyakran található a különféle alkalmazásoknál, például:

- a mikroprocesszorok többsége biztosítja a használatát;
- a Basic fordítók jelentős része valamennyi numerikus műveletet BCD kódot használva végeznek;
- például a zsebszámológépek és a digitális órák általában ezt a formátumot használják.

Komplex ALU

Fixpontos - lebegőpontos – BCD Melyiket használjam?

Fixpontos

A fixpontos, azaz az integer számításokat a processzor **könnyebben** hajtja végre, ezért nyilvánvalóan gyorsabbak. Az operandusok az operatív memóriában általában **kevesebb helyet** igényelnek (a byte formátum csak 8-bites, míg a legrövidebb lebegőpontos formátum is 32 bites) és így a **lehívásuk is gyorsabb** lehet (a 32-bites longint gyorsabban olvasható be egy 32-bites sínen, mint egy 64-bites duplapontosságú lebegőpontos szám).

A számok ábrázolása is **pontosabb**, mint a lebegőpontos számoké (egy IF a=1 utasításban például veszélyes lenne lebegőpontos formátumú a változót használni); .

Műveletvégzés során már az ábrázolt számok pontossága már erősen korlátozott, hiszen ha a törtpontot a szám végére képzeljük, az egész számok közötti számtartományok a gép számára nem léteznek. Ez azzal az eredménnyel jár, hogy **egyes műveletek pontatlanul** végezhetők. (pl. 7/4=1).

A mai magas szintű programozási nyelvek általában biztosítják a 8, a 16 és a 32 bit hosszúságú integer formátumot. Ezek közül a változónk várható értelmezési tartományának megfelelően válasszunk. Igyekezzünk a lehető legrövidebb formátumot választani.

Fixpontos - lebegőpontos – BCD Melyiket használjam?

Lebegőpontos

A lebegőpontos számokat akkor használjuk, amikor a változónk vagy konstansunk **nem csak egész**, hanem **törtrészt is** tartalmaz, a számunk **kívül eshet az integer értelmezési tartományon**, az igényelt pontosság túllépi azt a **pontosságot**, amit a leghosszabb rendelkezésünkre álló integer formátumunk biztosít.

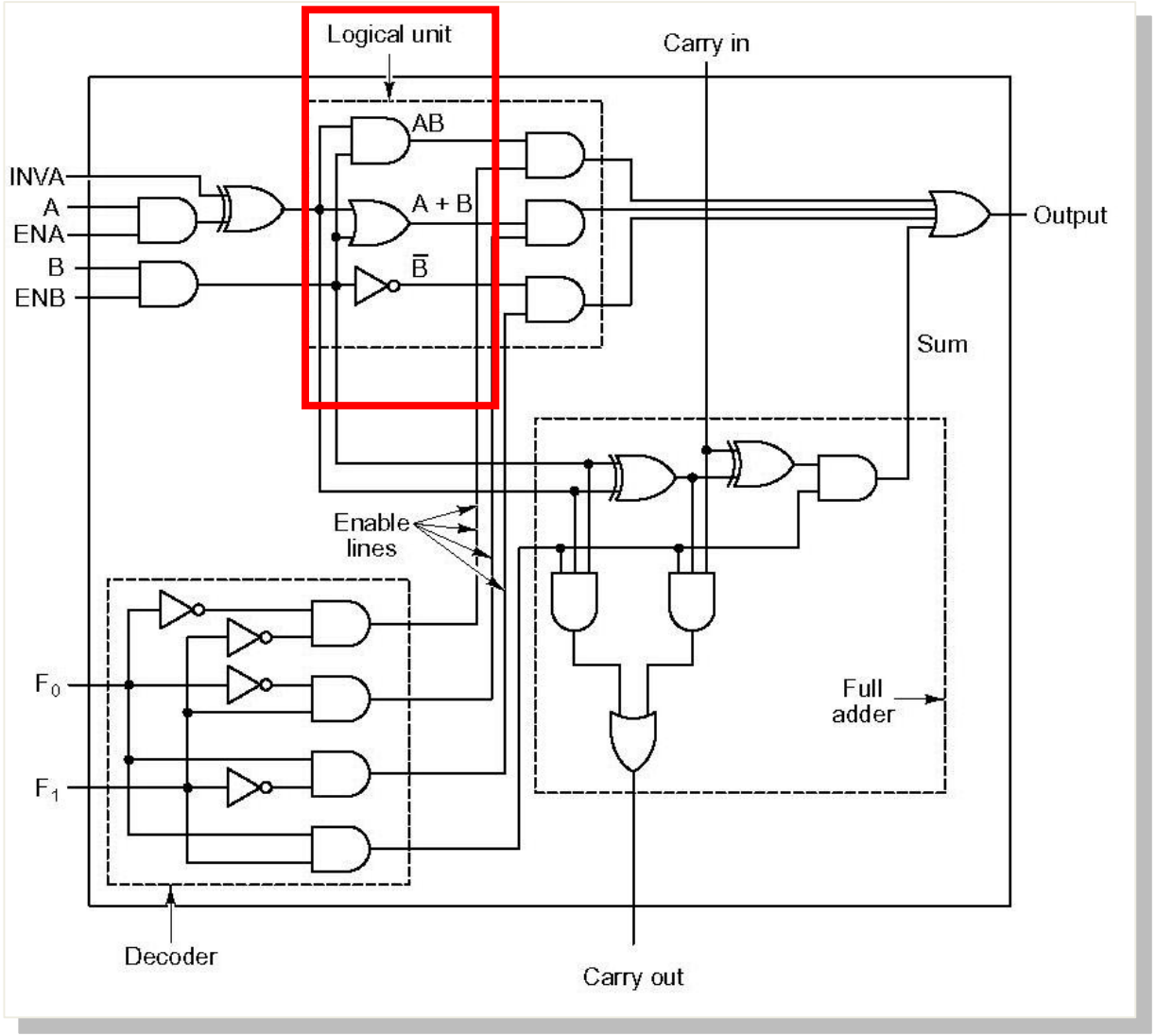
Ugyanúgy, mint az integer esetében, a lebegőpontos számoknál is a rövidebb formátummal gyorsabb a műveletvégzés, ezért csak akkor használjuk a pontosabb formátumot, amikor arra valóban igény van.

BCD

Azon programozási nyelvek esetében, melyek lehetővé teszik a BCD formátum alkalmazását, ott a **gazdasági alkalmazások** által igényelt pontosság elérése céljából a lebegőpontos ábrázolásnak egy attraktív alternatívája a BCD ábrázolás.

Az ALU egyéb funkciói

- Logikai műveletek
- Léptetések
- Címgenerálás - indexelés
- Összehasonlítás (kivonás és az előjel vizsgálata) és feltételes ugrás



Forrás: Tanenbaum