Számítógép architektúrák alapjai

Hallgatói jegyzet

Műveletvégző egység

2023

Tartalom

Készítette: Kováts Máté

A jegyzet Durczy Levente 2019 őszi Számítógép architektúrák alapjai előadás videói alapján készült.

Segítséget jelentett Uhrin Ádám és Nagy Enikő korábbi jegyzetei.

1. Az aritmetikai egység felépítése

A processzor szintű fizikai architektúra részei:

- **műveletvégző**
- vezérlő
- I/O rendszer
- megszakítási rendszer

A CPU két fő funkciója a műveletvégzés és vezérlés, melyek az utasítás lehíváshoz (FETCH) és utasítás végrehajtáshoz (EXECUTION) szükségesek.

1.1. CPU típusok

- Szinkron
  - Órajel generátorra működik
  - Hátránya: késleltetés → a következő órajelet mindig meg kell várni, így holtidő keletkezhet
  - Előnye: egyszerű, gyors
- Aszinkron
  - Egy utasítás befejezése után szinte közvetlenül indul a következő utasítás
  - Hátránya: speciális áramkör kell az utasítás befejezésének érzékelésére, ami drága, és érzékelése plusz idővel jár
  - Előnye: nincs holt idő

1.2. Műveletvégző

Részei:

- regiszterek
- adatutak
- kapcsolópontok
- szűkebb értelemben vett ALU

1.2.1. Regiszterek

- látható regiszterek
  - univerzális: bármilyen értéket beletehet a programozó
  - dedikált, pl.: stack
- rejtett regiszterek: adatfeldolgozáshoz szükséges puffer regiszterek. Hivatkozni nem lehet rájuk, de alacsony szintű programozásnál számításba kell venni őket.

1.2.2. Adatutak

FONTOS, hogy ez NEM BUSZ! Adatbuszon értelmezett a címzés, míg adatutak esetében nem. A processzor belső részeit kötik össze.

A műveletvégző egységen belül is vannak regiszterek. Ezek általában olyan regiszterek, melyek megcímzése kívülről történik, így a processzor betudja tölteni a megfelelő műveletvégző megfelelő regiszterébe az értéket. Vannak rejtett regiszterek puffer regiszterek: source és result. Illetve maga a

szűkebb értelemben vett ALU. Az adatút egy vezetékrendszerként fogható fel, mely összeköti a regisztereket, puffer regisztereket és ALU-t. Adatúton egyszerre csak egy adat lehet.

Csatolási módok:

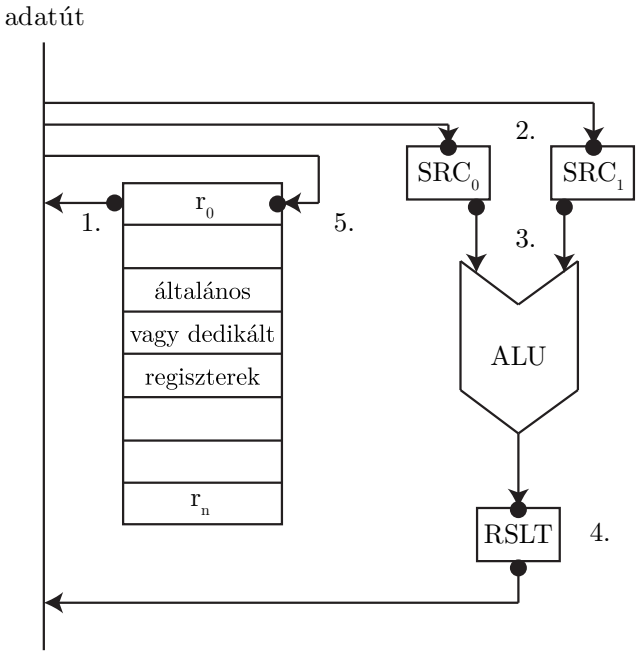
- egyutas: egy vezeték van, ezen keresztül kapcsolódnak az eszközök. Előnye, hogy egyszerű és olcsó, de lassú
- kétutas: egyidőben lehet két adatot betölteni a forrásregiszterekbe
- háromutas: harmadik adatút a RSLT és regiszterek bemenetére van rákötve, így párhuzamosan végezhető a betöltés és visszaírás

Példa:

ADD r0, r1

r0 = r0 + r1

1. r_0 tartalmát el kell juttatni az SRC_0-ba, ezért a vezérlő r_0 kimenő kapcsolópontját kinyitja
2. majd az SRC_0 bemenő kapcsolópontját, így az r_0 tartalma eljut az SRC_0-ba. Miután ez megtörtént, a vezérlő lezárja a kapcsolópontokat. Hasonló lépések végbe mennek r_1 és SRC_1 között.
3. Órajelre szinkronizáltan a vezérlő kinyitja SRC_0 és SRC_1 kimenő kapcsolópontjait az ALU felé, bejut a két érték és a megfelelő áramkörök megnyitásával elvégződik a megfelelő aritmetikai utasítás.
4. Az eredmény a RSLT regiszterben jelenik meg.
5. RSLT tartalma visszaíródik az r_0 regiszterbe



1.2.3. Kapcsolópontok

Regiszterek bemenetén és kimenetén lévő tranzisztorok. Vezérlő feladata a kapcsolók állapotának megváltoztatása.

Kimeneti kapcsoló: 3 állapot → 1, 0 vagy zárt

Bemeneti kapcsoló: 2 állapot → zárt vagy nyílt

1.2.4. ALU

Műveletek:

1. FX:
    a. összeadás
    b. kivonás
    c. szorzás
    d. osztás
2. FP
    a. összeadás
    b. kivonás
    c. szorzás
    d. osztás
3. BCD
    a. összeadás
4. egyéb műveletek
    a. eltolás
    b. negálás
    c. léptetés
    d. logikai

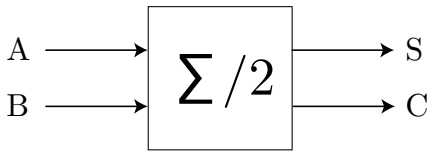
2. Az aritmetikai egység megvalósítása

2.1. 1 bites félösszeadó

A és B bemeneti operandus esetén összeadás eredménye S, ahol C az átvitel a következő igazságtábla alapján:

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

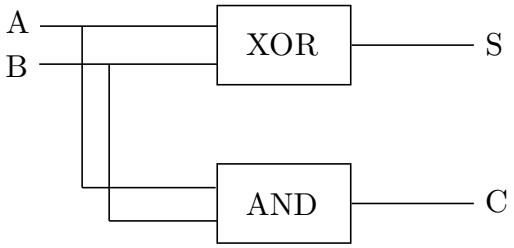
Jelölése:



Megvalósítása:

$S = \overline{A}\ B + A\ \overline{B} = \mathbf{A\ XOR\ B}$

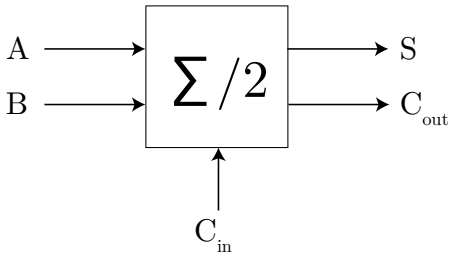
$C = \mathbf{AB}$



2.2. 1 bites teljes összeadó

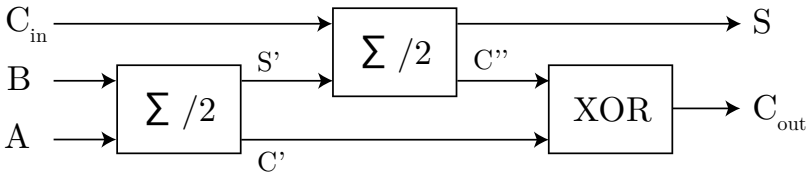
A két bemenet mellett megjelenik egy bemenő átvitel, ezáltal az összeadásra hatással lesz az előző művelet átvitele.

Jelölése:



Megvalósítás félösszeadókkal:

A-t és B-t félösszeadóval összeadom. Megkapjuk ideiglenes S'-t és ideiglenes C'-t. Ahhoz, hogy valóban megkapjam a végleges eredményt kezelnem kell a bemeneti átvitel ezért azt az S'-höz hozzá kell adnom egy félösszeadóval. A rendszerben akkor van átvitel, ha bármelyik megmaradt carrynél (C', C'') keletkezik átvitel.

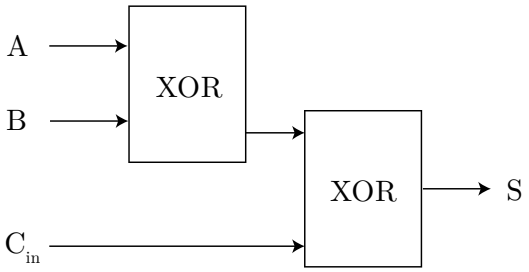


Egyszerűsítéshez szükség van igazságtáblához:

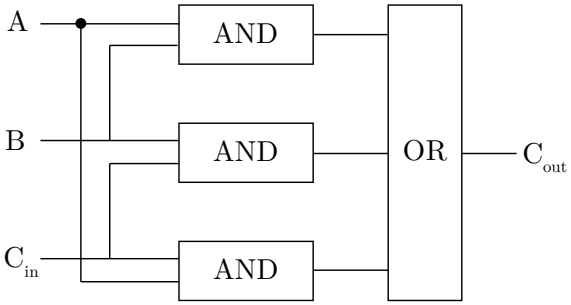
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Az eredményt és keletkező átvitelt egyszerűsítések után a következő áramkörökkel valósíthatjuk meg:

$S = \overline{A}\ BC + \overline{A}\ B\overline{C} + A\overline{B}\ \overline{C} + ABC =$ **A XOR B XOR C**



$C_{out} = \overline{A}\ BC + A\overline{B}\ C + AB\overline{C} + ABC =$ **AB + (A+B)C**



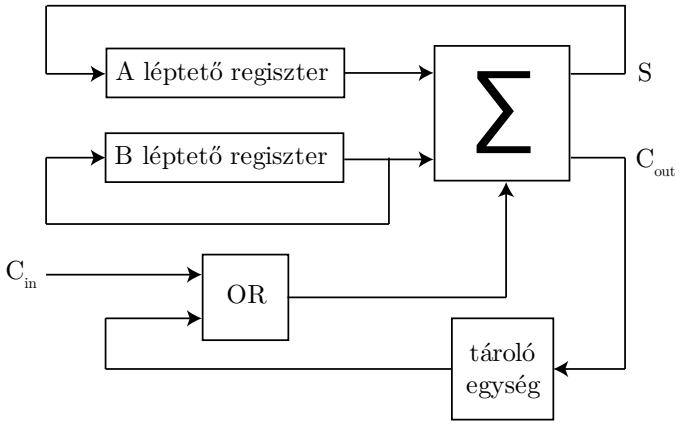
2.3. N bites soros összeadó

Gyakorlatban ritkán van szükség két bit összeadására, inkább bitek, bit sorozatokat adnak össze. A bit sorozatok tárolása regiszterekben történik (byte, szó, dupla szó, stb.). Ennek első megvalósítása az n bites soros összeadó, ami byte esetén a következőképpen valósítható meg.

Szükséges komponensek:

- teljes összeadó
- két léptető regiszter: tárolja a két számot és biztosítja, hogy lehet tologatni a biteket a helyiértékek helyén
- flip-flop: 1 bites késleltető, ami tárolja az átvitelt és biztosítja, hogy a megfelelő időpontban engedje ki a tárolt adatot
- OR kapu: a legelső bit összeadásánál kell figyelembe vennie a bemeneti átvitelt

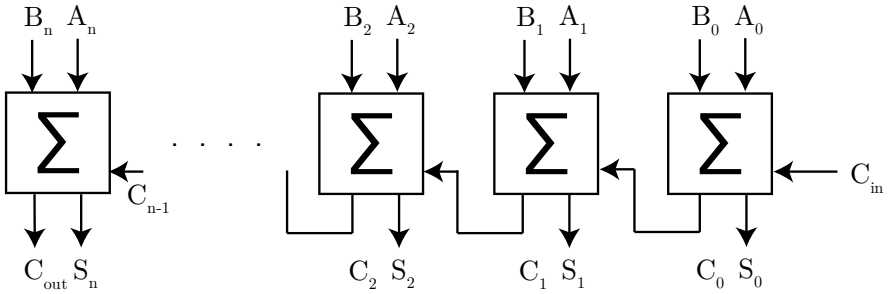
Legkisebb helyiértéktől haladva a nagyobbak felé összeadjuk a biteket, miközben figyelembe vesszük az előző összeadásnál keletkezett átvitelt is. Az eredmény az A regiszterben fog előállni.



Amennyiben egy bit összeadása t idő, akkor n bit esetén n*t.

2.4. N bites párhuzamos összeadó

Legfontosabb, hogy a lehető leggyorsabban végezzük el az összeadást → gyorsítani kell párhuzamosítással. Byte esetén kell 8 darab teljes összeadó, mely bitenként képes egyszerre elvégezni az összeadásokat. Az átviteleket mindig a nagyobb helyiértékű bit összeadásához kell csatolnunk.



Hogy valóban egyszerre történjen az összeadás akadályozó tényező az átvitel, hiszen az n-edik összeadáshoz szükség van az n-1-edik átvitelre. Abban a pillanatban, mikor az n-edik bit összeadását megkezdenénk, n-1-edik összeadás kimeneti átvitele még nem áll elő.

Végrehajtási idő: ~ n*t (hullámzó idő). Ha nem keletkezik átvitel lényegesen gyorsabb.

2.5. Előre jelzett átvitellel felépített n-bites összeadó

Megoldás a szimultán átvitelképzés (Carry look ahead - CLA). Ahogyan a teljes összeadónál már megfogalmazható C_{out} kifejezhető a következő módon:

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

Legyen AB = G (generálja az átvitelt)

A+B = P (propagálja az átvitelt)

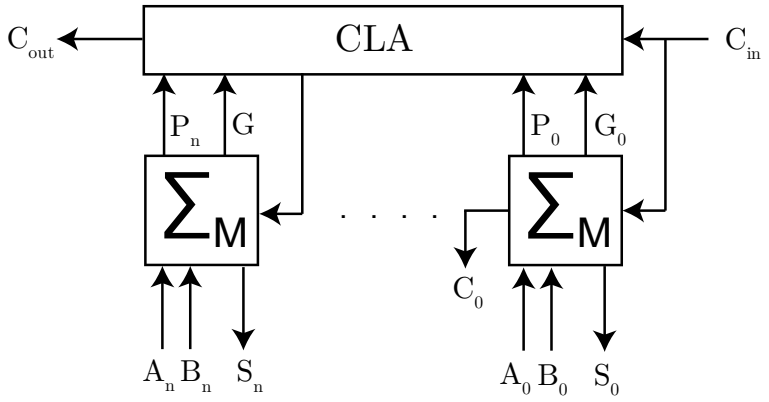
$C_{out} = G + PC_{in}$

$C_0 = G_0 + P_0C_{in}$

$C_1 = G_1 + P_1C_0 = G_1+P_1G_0+P_1P_0C_{in}$

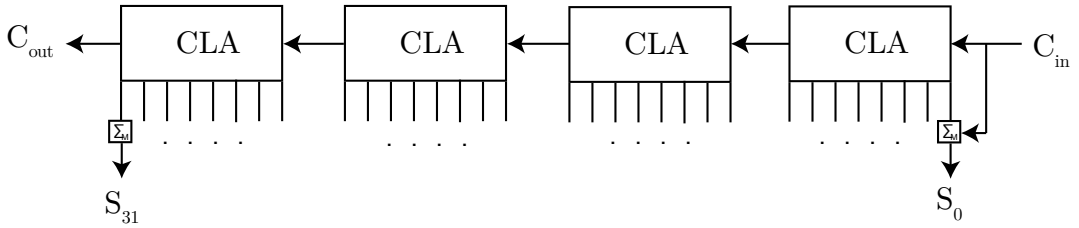
$C_2 = G_2 + P_2C_1 = G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0P_{in}$

Amit előre eddig nem tudtunk kiszámolni az a C_0, C_1, C_2, stb. Viszont ezzel az átírással az előre adott értékekből elő tudjuk állítani az összes átvitelt, és onnantól kezdve működik a párhuzamos összeadó. Végrehajtási idő: 2t+t

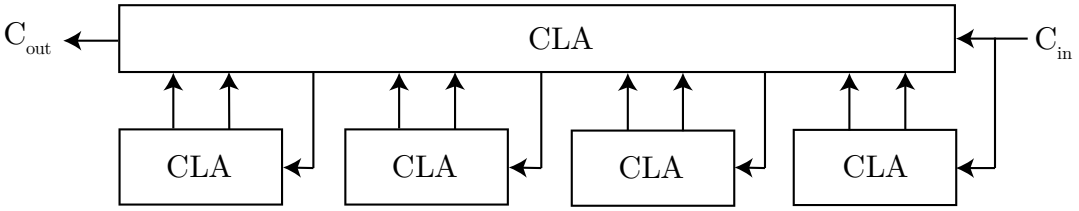


Van egy módosított teljes összeadónk, mely előállítja propagáló és generáló tagot. Tehát az első bitek összeadásánál előáll P_0 és G_0, amikből a bemeneti átvitellel a fentebbi átalakítások alapján a CLA áramkör minden helyiértékre kitudja számolni az átvitelt. Fontos észrevenni, hogy az összedók bemeneti átvitelt a CLA-ból kapják, nem a megelőző összeadóból.

C_2 kifejezésénél látható, hogy jelentősen megnő az összeadandó tagok száma. Általában maximum 8 bemenete lehet egy OR kapunak, ezért 32 bit esetén 4 CLA szükséges, ami ugyanoda vezet, mint párhuzamos összeadónál, hogy az átvitel sorosan terjed.



Ezért készíthetünk CLA-t a CLA-k számára, hogy előállítsuk a CLA-k bemeneti átvitelét előre:



2.6. Fixpontos szorzás megvalósítása

Szorzást is összeadás sorozatára vezetjük vissza, de jóval komplexebb.

Szorzás sebességének gyorsításához rendelkezésre álló elemi műveletek:

- összeadás
- invertálás
- léptetés

2.6.1. Fixpontos szorzás algoritmusai

Hagyományos módszer: a szorzó legkisebb helyiértékétől kezdve megszorozzuk a szorzandót, majd helyiérték váltáskor egyet léptetünk, hogy helyiérték-helyes legyen. Végül a részeredményeket összeadjuk.

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

Első megoldások ez alapján a **léptetéses módszert** használták: létrehozunk egy gyűjtő regisztert, amit a művelet megkezdése előtt kinullázunk. A gyűjtőhöz hozzáadjuk a szorzó legkisebb helyiértékétől haladva annyiszor a szorzandót, amennyiszer a szorzó adott helyiértéke kívánja. Majd léptetünk és folytatjuk a szorzó eggyel nagyobb helyiértékével.

$$\begin{array}{rl} X=A*B=\underline{13}*123 & \\ 0000 & \\ \underline{\quad 39} & \\ 0039 & \\ \underline{\quad 26} & \textit{léptetés} \\ 0299 & \\ \underline{13\quad} & \textit{léptetés} \\ 1599 & \end{array}$$

Probléma, hogyha összeszorzunk két számot, akkor általában nagyobb, mint a két szám és több bit helyet foglal.

Kettes számrendszerben elmondható:

A: m bit

B: n bit hosszú

A*B <= m+n bit hosszú

Például 8 bit * 8 bit esetén 16 bit hosszú tárolót kell biztosítani az eredménynek.

2.6.2. Fixpontos szorzás gyorsítási lehetőségei

Bitcsoporttal történő szorzás:

10-es számrendszerben: 7*9=63

2-es számrendszerben: 0111*1001

Ahelyett, hogy bitenként hajtjuk végre a szorzást, történhet ez bitcsoportonként:

$$\begin{array}{rl} \underline{0111}*10|01 & \\ \underline{0000} & \textit{gyűjtő} \\ \underline{01\overset{\wedge}{1}1} & \textit{egyszerese és léptetés kétszer} \\ 01\overset{\wedge}{1}1 & \\ \underline{1\overset{\wedge}{1}1000} & \textit{kétszerese és léptetés kétszer} \\ 1\overset{\wedge}{1}1111 & \end{array}$$

00: léptetünk kettőt

01: hozzáadjuk az egyszeresét és léptetünk kettőt

10: hozzáadjuk a kétszeresét, majd léptetünk kettőt

11: négyszeresét adjuk hozzá és kivonjuk belőle az egyszeresét

(kétszeres: kétszeres szorzat a léptetés 110_2 (6_{10}) → 1100_2 (12_{10}))

(négyszeres: négyszeres szorzat a kétszer való léptetés 111_2 (7_{10}) → 11100_2 (28_{10}))

(kivonás: negáljuk a kivonandót és hozzáadjuk a kisebbítendőhöz)

Booth algoritmus:

Ha a szorzóban sok 1-es van, akkor lassú a szorzás, mert sok összeadást kell végezni. Az **A*62 (111110)** szorzás több lépcsőben kellene elvégezni. Helyette:

Keressük meg a szorzóhoz legközelebbi számot és állítsuk elő az eredményt úgy, hogy:

A*(64-2) = A*64 – A*2

64 (1000000) → 1 összeadás

2 (10) → 1 összeadás

kivonás → 1 összeadás

5 darab összeadás, helyett 3-ra csökkentjük.

2.7. Lebegőpontos számok

A fixpontos számok értelmezési tartománya viszonylag kicsi, pontosság sem túl jó.

Például 16 bit esetén -32768 → +32767

Ezt küszöböli ki a bonyolultabb, de bizonyos körülmények között pontosabb lebegőpontos számítás.

2.7.1. Történeti áttekintés

Konrad Zuse 1941-ben alkotta meg a Z3 computert. Elsőnek tudta kezelni a lebegőpontos számokat. Tartalmazott ALU-t, vezérlőegységet, memóriát, I/O-t és használt rejtett bitet. Tömege 1 tonna volt. Összeadás ~0,7 sec, szorzás ~3 sec volt.

Howard Aiken 1944-ben MARK 1 számítógép. Csak FX számításokat tudott. 3-5 sec volt egy művelet.

1946-ban megalkotott Eniacot tekintik az első igazi számítógépnek, melynek súlya 30 tonna volt. Teljesítménye 160 kW. 1000x-es növekedés a MARK 1-hez képest.

Lebegőpontos számok szabványosítása csak 1985-ben jött létre: IEEE754.

2.7.2. Jellemzői

$FP = M * r^k$ M: mantissza, r: radix (számrendszer alapja), k: karakterisztika

Elvárás, hogy a radix egyezzen meg a mantisszánál használt számrendszer alapjával.

2-es számrendszerben: $0{,}10011*01^{101}$

2.7.3. Ábrázolás

Normalizált formátumban történik, ami azt jelenti, hogy az első értékes jegy a tizedespont után van és a karakterisztikát ennek megfelelően adjuk meg.

$0{,}001*2^k \rightarrow 0{,}1*2^{k-2}$

Mantissza értéke:

10-es számrendszernél:	$0{,}1 <= M < 1$
2-es számrendszer:	$½ <= M < 1$
általános számrendszer:	$1/r <= M < 1$

2.7.4. Értelmezési tartomány

Függ:

- karakterisztika számára rendelkezésre álló bitek számától
- radixtól

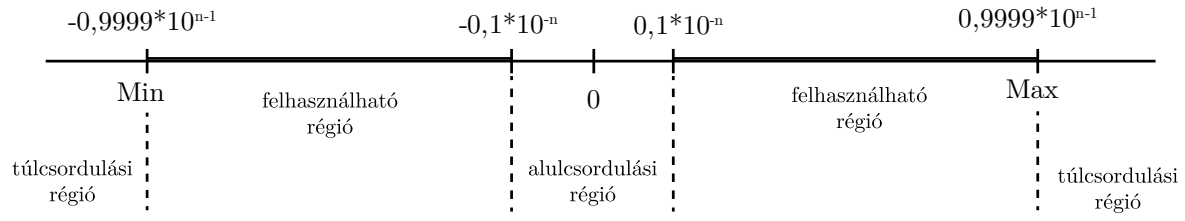
Karakterisztika bitek száma	Legnagyobb érték$_{10}$	Értelmezési tartomány$_{10}$	Legnagyobb érték$_2$	Értelmezési tartomány$_2$
1	±9	$10^{\pm 9}$	1=1	$2^{\pm 1}$
2	±99	$10^{\pm 99}$	11=3	$2^{\pm 3}$
3	±999	$10^{\pm 999}$	111=7	$2^{\pm 7}$
4	±9999	$10^{\pm 9999}$	1111=15	$2^{\pm 15}$ (=FX16 32768)

2.7.5. Pontosság

Mantisszai bitjeinek számától függ.

Például, ha 3 bit a mantissza, akkor a $0{,}3014*10^6$ esetén a tízezredet már nem tudjuk ábrázolni.

Példa: 10-es számrendszer, 4 mantissza bit esetén:



Az architektúrának biztosítania kell a túlcsordulás és alulcsordulás felfedezését, jelzését és kezelését.

Túlcsordulás esetén:

- kijelzi és beállítja a legnagyobb megengedett értéket
- előjeles végtelent jelez ki

Alulcsordulás esetén:

- kijelzi és 0-ra konvertál

- denormalizált számot jelzi ki

Ha a mantissza 0, akkor a karakterisztika is 0 legyen.

Pontosság növelése:

1. Rejtett bit használata
   Mivel normalizált formátumban az első értékes bit van a tizedespont után és az minden esetben 1-es, így tárolásnál annak nincs információ tartalma.

   0,101
   0,1110
   0,10001 Ezért csak a második bittől történik a tárolás

   Ha a mantissza 23 bit, akkor 24 bitet tudunk eltárolni → nő a pontosság
2. Őrző bitek
   Elvárás a lebegőpontos számokra, hogy a relatív hiba kisebb legyen, mint a normalizált eredmény legkisebb számjegye.
   Megoldás: a CPU-n belül a regiszterek több biten tudják tárolni a mantisszát (általában plusz 3-15 bit)
   Felhasználásuk:
   - a rejtett bit balra léptetésekor értékes bitet tudunk beléptetni (helyreállítás)
   - tárolási formátum kérésekor kerekített értéket tárolhatunk
   - normalizáláskor értékes biteket tudunk felhasználni

2.7.6. Kódolás

- Mantissza kódolása: 2-es komplemens
- Karakterisztika: többletes kódolás → kialakítása gyorsabb, de csak alapműveletekre alkalmas

2.8. IEEE 745

1985-ben jelent meg az eddigi legjobb megoldások alapján. Célja megkönnyíteni a különböző CPU-k esetén az adatszintű kompatibilitást, portabilitást.

Rendszerszintű megoldás, vagyis a hardvernek és a szoftvernek együtt kell biztosítania a szabványnak való megfelelést.

Fejezetei:

- adattípus
- formátumok
- műveletek
- kerekítések
- kivételek

2.8.1. Formátumok

- szabványos: háttértáron való tároláshoz, kötött
  - egyszeres pontosságú (32 bit)
    - kisebb, gyorsabb
    - pontatlanabb

1 előjel bit	8 bit karakterisztika	23 bit mantissza

    - kétszeres pontosságú (64 bit)
        - nagyobb, lassabb
        - jóval pontosabb

1 előjel bit	11 bit karakterisztika	52 bit mantissza

- kiterjesztett: CPU-n belül, nagyobb szabadság
    - egyszeres pontosságú (min. 43 bit)
    - kétszeres pontosságú (min. 79 bit)

2.8.2. Műveletek

- 4 aritmetikai művelet
- maradékképzés
- négyzetgyökvonás
- bináris, decimális konverzió
- végtelennel való műveletvégzés
- kivételek kezelése

2.8.3. Kerekítések

- legközelebbire való kerekítés
- 0-ra kerekítés (őrző bitek levágását jelenti)
- kerekítés pozitív végtelen felé
- kerekítés negatív végtelen felé

2.8.4. Kivételkezelés

Felbukkanásuk általában megszakítást eredményez.

- túlcsordulás
- alulcsordulás
- 0-val való osztás
- gyökvonás negatív számból

2.8.5. Példa IEEE 754 használatára

A szabvány használó első CPU az 1981-ben megjelent Intel 8087.

- radix: 2
- rejtett bitet használt
- őrző biteket használt
- megvalósította mind az egyszeres, mind a kétszeres pontosságot is (programozó döntötte el, hogy melyiket használja)
- kiterjesztett formátum: 80 bit

2.9. Műveletvégzés lebegőpontos számokkal

Legyen két lebegőpontos számunk:

$$A = \pm m_a * r^{kA}$$

$B = \pm m_b * r^{kB}$

2.9.1. Összeadás

Csak azonos kitevőjű számok adhatók össze. Amennyiben a kitevők nem egyenlők, akkor a kisebb kitevőjű szám mantisszájának törtpontját balra léptetjük, és közben inkrementáljuk a karakterisztika értékét. A ciklus addig fut, amíg a kitevők meg nem egyeznek. Mantisszákat összeadjuk, karakterisztikákat változatlanul hagyjuk. Normalizálás szükség esetén.

$X=A+B=(m_a+m_b)*r^{kA}$, ahol $r^{kA}= r^{kB}$

$A=0{,}95*10^4= 0{,}95*10^4$
$B=0{,}90*10^3= 0{,}09*10^4$
$1{,}04*10^4= 0{,}104*10^5$

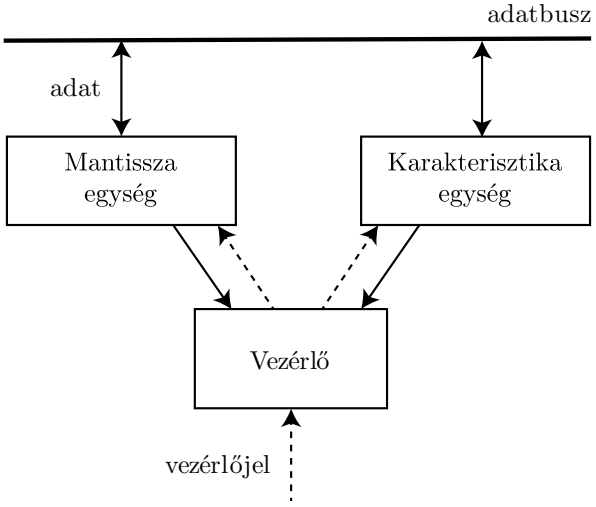
2.9.2. Szorzás

A mantisszákat össze kell szorozni, a karakterisztikákat meg össze kell adni. A két művelet párhuzamosan végezhető.

$X=A*B=(m_a*m_b)*r^{kA+bB}$

$A=0{,}95*10^4$
$B=0{,}90*10^3$
$A*B=0{,}855*10^7$

Megvalósítása a **dedikált FP műveletvégző**: az adatbuszon keresztül a lebegőpontos számok mantisszája a mantissza egységbe, a karakterisztika a karakterisztika egységbe kerül, majd a vezérlő vezérlőjelek segítségével szinkronizálja és határozza meg az egységek működését, melyek eredményét végül összerakja.



2.10. BCD számábrázolás

Megjelenésének oka: FX és FP számábrázolás pontatlansága. Elsősorban adminisztratív alkalmazásoknál használják. A kódolás pontos megfeleltetés, vagyis nem kell kerekíteni.

2.10.1. Ábrázolás

4 biten történik 0-9 tartományban, amit tetrádnak hívnak. A 10-15 tartományba esőket pedig érvénytelen tetrádnak.

0	0000	
9	1001	
A	1010	Érvénytelen tetrádok
B	1011	
C	1100	
D	1101	
E	1110	
F	1111	

Érvénytelen tetrádot úgy lehet megállapítani, hogyha az első bitje 1-es, akkor vagy a második vagy harmadik bitje is egyes.

2.10.2. Formátum

Két formátum létezik:

- Zónázott: 1 byte egy számjegy, vagyis az alsó 4 bitjét használják, a felső 4 bit a zóna, ami szabadon felhasználható
    - pazarló

1 számjegy		1 számjegy		1 számjegy	
Zóna 4 bit	BDC 4 bit	Zóna 4 bit	BDC 4 bit	Zóna 4 bit	BDC 4 bit
1 byte		1 byte		1 byte	

- Pakolt: 1 byte 2 számjegy
    - pl: Intel 10 byte → 1 byte előjelnek, 9 byte 18 helyiértéknek

2 számjegy		2 számjegy		2 számjegy	
BDC 4 bit	BDC 4 bit	BDC 4 bit	BDC 4 bit	BDC 4 bit	BDC 4 bit
1 byte		1 byte		1 byte	

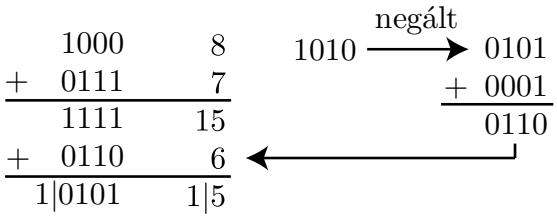
A hossz lehet fix vagy változó. Változó esetén specifikálni kell.

2.10.3. Műveletvégzés

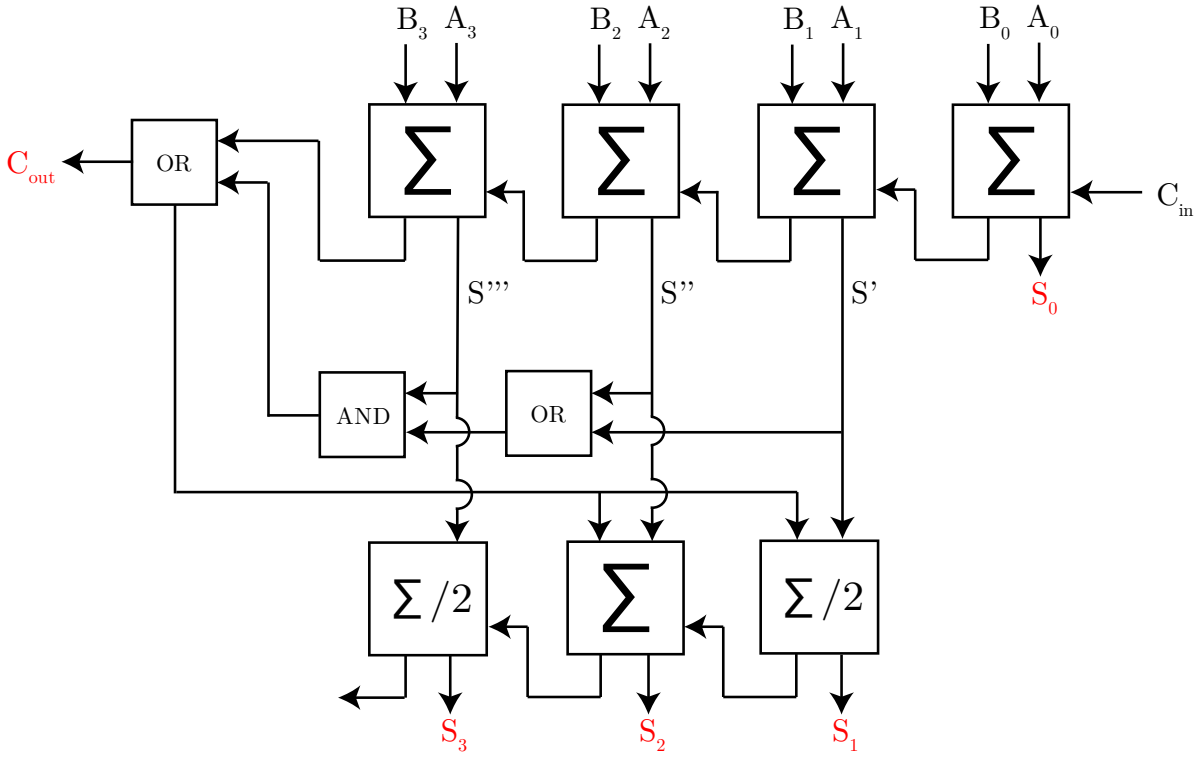
Például: 8+7=15

Mielőtt kettes számrendszerben felírnánk láthatjuk, hogy a 15 érvénytelen tetrád lesz. Ha ezt sikerül felismernünk áramköri szinten is, kezelni úgy tudjuk, hogy hozzáadunk -10-et kettes komplemens kódolással (ami tízes számrendszerben 6). Ilyenkor természetesen keletkezik átvitel.

(kettes komplemens képzés: vesszük a negáltját és hozzáadunk egyet)



Áramköri szinten: 4 biten ábrázoljuk, tehát kell hozzá 4 teljes összeadó. Ha megvannak bitenként az eredmények detektálnunk kell az érvénytelen tetrádot. Akkor érvénytelen egy tetrád, ha első bitje 1 és vagy a második vagy harmadik bitje 1. Ezt valósítjuk meg AND és OR kapukkal, melynek kimenetén 1-et kapunk, ha érvénytelen. Ezt OR kapuval a kimeneti átvitellel együtt kivezetjük, hiszen mindkét esetben biztos, hogy számolhatunk átvitel. Amennyiben van átvitel az előbb bemutatott módon 6-ot kell hozzáadnunk bithelyesen. Félösszeadóval hozzáadjuk az S'-höz (0110), teljessel S"-höz (0110), majd S"'-höz a keletkező átvitellel.



Előnye: teljesen pontos (ezért számológépeknél, pénzügyi rendszereknél használják)

Hátrány: komplexebb hardvert, több tranzisztor, több memóriát igényel

2.11. Összegzés (FX, FP, BCD)

FX

Előny:

- gyors
- egyszerű a megvalósítása
- kevés helyet igényel
- kezeli a 8, 16, 32, 64 bites formátumot

Hátrány:

- kicsi értelmezési tartomány
- tört számoknál / osztásnál pontatlan lehet

FP

Előny:

- nagy értelmezési tartomány
- általában elég nagy pontosságot biztosít

Hátrány:

- több erőforrást igényel
- legtöbbször kerekíteni kell

2.12. ALU egyéb műveletei

- mind a 16 fajta BOOLE művelet (AND, NOR, OR, XOR, …)
- léptetés
- invertálás
- komparálás (+feltételes ugrás)
- LOAD/STORE címszámítás → végrehajtási időben
- karakteres műveletek