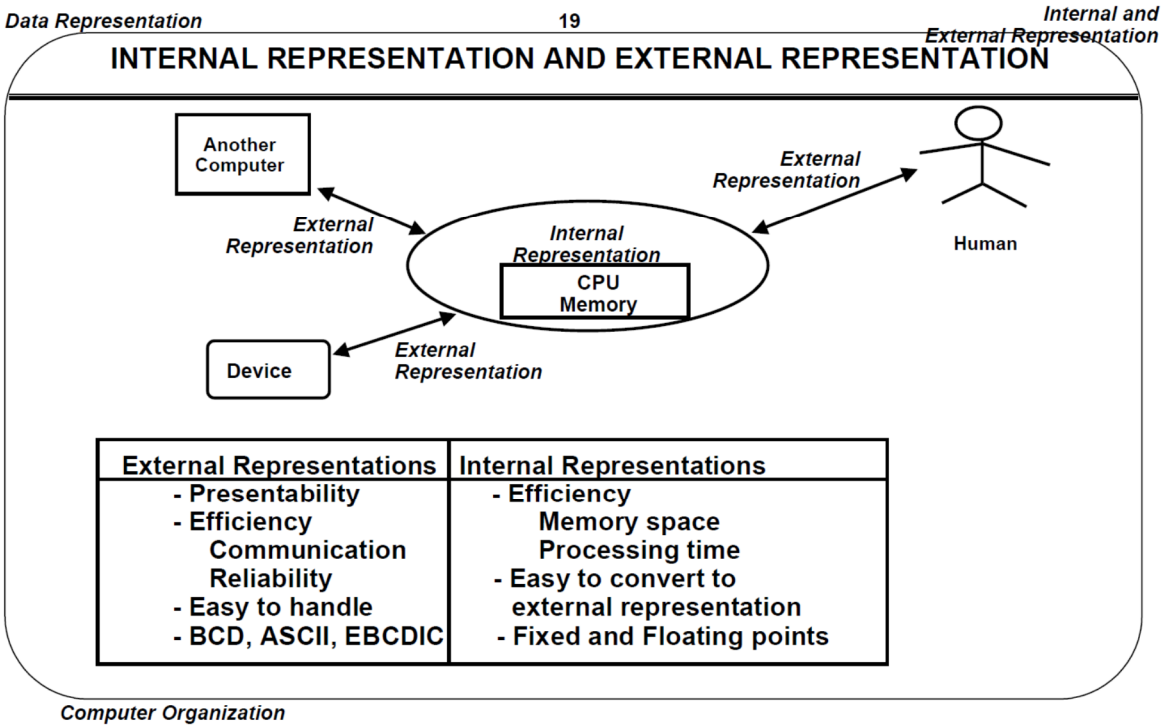
KÓDOK, SZÁMRENDSZEREK, ALGORITMUSOK

Tartalom

Külső – belső adatábrázolás

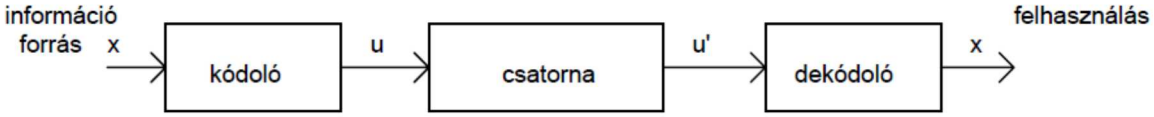
Forrás - haszn --- jóó -- data representation -KÜLSŐ--BELSŐ_.pdf



Kódolás

Forrás Besenóczky

A digitális kódokat és kódolást az információ átvitel oldaláról közelítjük meg.



A kódolás magába foglalja az
ún. *forrás kódolást*, melynek célja az információ tömörítése,
a *titkosítást*, melynek célja, hogy illetéktelen ne tudja visszafejteni az információt,
a *csatorna kódolást*, melynek célja a digitális információ hibátlan átvitele

Egyszerű példa digitális információ átvitelre:

információ forrás:	pl: magyar szöveg
kódolás:	pl: a magyar szöveg betűnkénti kódolása bináris számokká (pl. ASCII kód)
csatorna:	pl: két állapotú, fizikailag két feszültségszint reprezentálja
dekódolás:	pl: a bináris számok magyar karakterekké alakítása

A csatorna lehet több állapotú is de itt csak a két állapotúval foglalkozunk

$A = \{a_1, a_2, \ldots a_n\}$ forrás ABC

Bináris kód esetén a kód ABC 2 elemű: {0, 1}

$K = \{\alpha_1, \alpha_2, \ldots \alpha_n\}$ a kód ABC betűiből képzett véges hosszúságú sorozatok halmaza

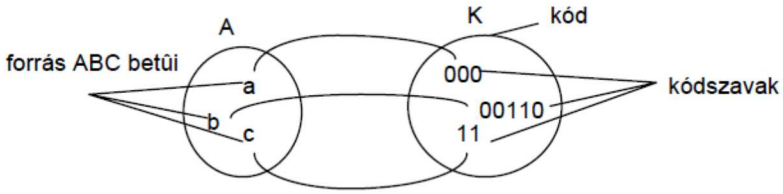
Pl. bináris kód esetén K = {00, 010, 0111, …}

A 2-es számrendszrű számok egy számjegyét *bit*-nek nevezik, a *binary digit* (bináris számjegy) rövidítéseként

Betű szerinti kódolás:

Az *A* halmaz minden karakterének megfeleltetjük K egy-egy elemét

$(a_1 \rightarrow \alpha_1,\ a_2 \rightarrow \alpha_2, \ldots)$



A K halmazt *kódnak* nevezzük.

A K halmaz elemei a *kódszavak*, de a kódszavakat is szokás kódnak nevezni.

Kódok osztályozása:

karakterkészlet alapján:	- bináris, {0,1} - nem bináris pl. {0, 1, 2}
a kódszavak hosszúsága alapján:	- fix hosszóságú, Pl. {100, 010, 111} - változó hosszúságú Pl. {01, 110,1111}
alkalmazási cél alapján:	- aritmetikai (1-es komlemens, offszet stb.), - pozició (Pl. Gray kód, Jophnson kód) - hibajavító (Pl. Hamming kód) stb...

- rejtjelező titkosítás kód
- tömörítő
- pozíció – n-ből 1, n-ből m

Kód definíció

Forrás: - haszn ---- EZ KELL -- DEFINITION of codes .pdf

6.1 Definition of Codes:

'Encoding' or 'Enciphering' is a procedure for associating words constructed from a finite alphabet of a language with given words of another language in a one-to- one manner.

Let the source be characterized by the set of symbols

$S = \{s_1, s_2 \ldots s_q\}$ ..... (6.1)

We shall call '*S*' as the "***Source alphabet***". Consider another set, *X*, comprising of '*r*' symbols.

$X = \{x_1, x_2 \ldots x_r\}$ ..... (6.2)

We shall call '*X*' as the "***code alphabet***". We define "***coding***" as the ***mapping*** of all possible sequences of symbols of *S* into sequences of symbol of *X*. In other words "***coding means representing each and every symbol of S by a sequence of symbols of X such that there shall be a one-to-one relationship"*** Any finite sequence of symbols from an alphabet will be called a "***Word***". Thus any sequence from the alphabet '*X*' forms a "***code word***". The total number of symbols contained in the '***word***' will be called "***word length***". For example the sequences $\{x_1 ; x_1x_3x_4 ; x_3x_5x_7x_9 ; x_1x_1x_2x_2x_2\}$ *form code words. Their word lengths are respectively1; 3; 4; and 5.The* sequences {***100001001100011000***} and {***1100111100001111000111000***} are binary code words with word lengths ***18*** and ***25*** respectively.

Kód tulajdonságok

1. Block codes:

A block code is one in which a particular message of the source is always encoded into the same "***fixed sequence***" of the code symbol. Although, in general, block means '*a group having identical property*' we shall use the word here to mean a '***fixed sequence***' only. Accordingly, the code can be a '***fixed length code***' or a "***variable length code***" and we shall be concentrating on the latter type in this chapter. To be more specific as to what we mean by a block code, consider a communication system with one transmitter and one receiver. Information is transmitted using certain set of code words. If the transmitter wants to change the code set, first thing to be done is to inform the receiver. Other wise the receiver will never be able to understand what is being transmitted. Thus, until and unless the receiver is informed about the changes made you are not permitted to change the code set. In this sense the code words we are seeking shall be always ***finite sequences*** of the code alphabet-they are ***fixed sequence codes***.

Example 6.1: Source alphabet is $S = \{s_1, s_2, s_3, s_4\}$, Code alphabet is $X = \{0, 1\}$ and The Code words are: $C = \{0, 11, 10, 11\}$

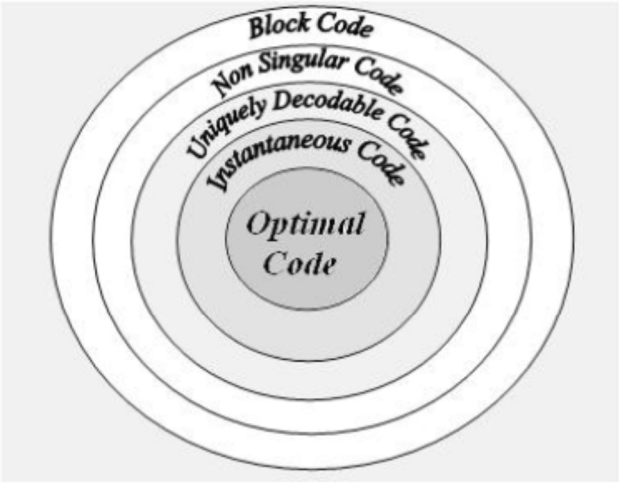


Fig 6.1 Codes - Sub grouping

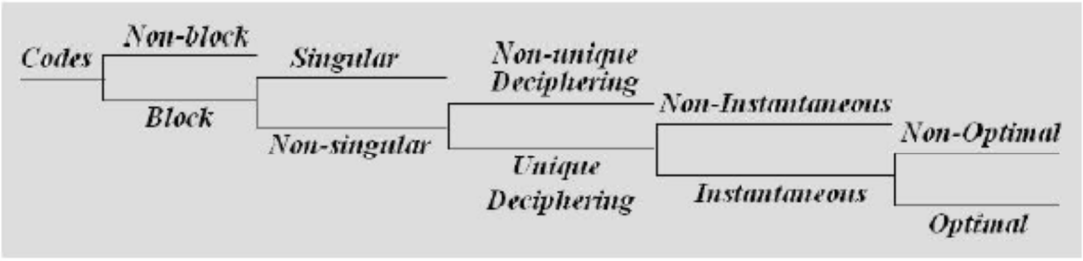


Fig 6.2 Codes - Sub grouping : Indicated as a Tree of Properties

Számábrázolás

1997 NBCD

Forrás Besenóczky

Aritmetikai kódok

NBCD (Natural Binary Coded Decimal) kód: a decimális 0...9 számokhoz 4 bites bináris számokat (0000...1001) rendel.

Pl: 1997 NBCD kódban: 0001 1001 1001 0111

Kényelmes a decimális számok ábrázolására, de nehézkes számolni vele.

Számábrázolások

	előjeles abszolut értékes	egyes komplemens	kettes komplemens	offszet
3	011	011	011	111
2	010	010	010	110
1	001	001	001	101
0	000, 100	000, 111	000	100
-1	101	110	111	011
-2	110	101	110	010
-3	111	100	101	001
-4			100	000

Előjeles abszolut értékes:

A bináris számok abszolút értéke elé egy előjel bit kerül, mely 0, ha pozitív, 1, ha negatív a szám. Kényelmes előjeles számok ábrázolására, de nehézkes számolni vele. A 0-nak két kódja van, ezért nem használja ki az összes lehetőséget.

Egyes komplemens:

A pozitív bináris szám bitenkénti negáltja adja a negatív megfelelőjét. Könnyű egy pozitív szám -1 szeresét képezni, de nehézkes számolni. A 0-nak két kódja van, ezért nem használja ki az összes lehetőséget.

Kettes komplemens:
A pozitív számok ábrázolása azonos az előjeles abszolút értékessel. Egy szám -1 szeresét úgy képezzük, hogy bitenkénti negáltjához 1-et adunk hozzá. Könnyű számolni vele, a megszokott bináris összeadás az előjeles számok között helyes eredményt ad. A 0-nak egy kódja van.

Offszet:
A legnegatívabb számhoz rendeljük a csupa 0 kódot, majd a többihez egyre növekvőket. Az előjeles számok összehasonlítása könnyű, a megszokott bináris összehasonlítás helyes eredményt ad. A 0-nak egy kódja van.

Áttérés az egyes számábrázolások között
Egyes komplemens-> kettes komplemens: Ha MSB=1, akkor X=X+1.

Kettes komplemens -> egyes komplemens: Ha MSB=1, akkor X=X-1

Előjeles absz. értékes -> egyes komplemens: Ha MSB=1, akkor MSB kivételével invertálni.

Egyes komplemens -> előjeles absz. értékes: Ha MSB=1, akkor MSB kivételével invertálni.

Előjeles absz. értékes -> kettes komplemens: Ha MSB=1, akkor MSB kivételével invertálni, majd 1-et hozzáadni.

Kettes komplemens -> előjeles absz. értékes: Ha MSB=1, 1-et kivonni, majd MSB kivételével invertálni.

Offszet -> kettes komplemens: MSB-t invertálni.

A számrendszereket (különösen a 2-es, 10-es és 16-os) készség szintjén ismertnek tekintjük.

Bináris és BCD ábrázolás

Számítógépek esetén a pozitív egész számok tárolása a legegyszerűbb esetben alapvetően megfelel a 2-es számrendszerbeli ábrázolásnak. A számokat alkotó biteket 8-asával byte-okban tároljuk.
Több byte-os adatok esetén a byte-ok sorrendje általában vagy az *LSB* (Least Significant Byte First = legkisebb helyi értékű bájt van elöl, vagyis az alacsonyabb memóriacímen) vagy az *MSB* sorrendet követi. (Kommunikáció esetén érdekes még a byte-ok bitsorrendje is, ami *lsb* = least significant bit first, illetve *msb* lehet.)

Bizonyos alkalmazások esetén elterjedt a *BCD (Binary Coded Decimal)* számábrázolás is, ahol a szám 10-es számrendszerbeli ábrázolásának megfelelő számjegyeket kódolják el binárisan.
Amennyiben egy bájton csak egy számjegyet ábrázolnak akkor azt *zónázott* BCD ábrázolásnak nevezik. Lehetőség van azonban arra is, hogy egy bájt felső és alsó 4 bitjén ábrázoljunk egy-egy (tehát bájtonként kettő) decimális számjegyet (ilyenkor a magasabb helyi értékű bitek hordozzák a magasabb helyi értékű BCD számjegyet). Ezt *tömörített BCD* ábrázolásnak nevezzük.
Gyakorlásként írjuk fel az aktuális évszámot 10-es, 2-es és 16-os számrendszerben! Írjuk fel binárisan a memória egymást követő rekeszeinek tartalmát, ha az aktuális évszámot

- binárisan LSB byte sorrendben ábrázoljuk
- zónázott BCD-ben, LSB byte sorrendben ábrázoljuk
- tömörített BCD-ben, LSB byte sorrendben ábrázoljuk

Negatív számok ábrázolása

Negatív számok ábrázolására – feltehetően?? az egyszerű aritmetikai műveletvégzés érdekében – leggyakrabban a kettes komplemens ábrázolást szokták használni. Egy binárisan ábrázolt szám kettes komplemensét megkapjuk, ha először a szám egyes komplemensét képezzük (0 helyett 1, 1 helyett 0 lesz) majd hozzáadunk 1-et.
Gyakorlásként írjuk fel 8 biten a következő számokat kettes komplemens alakban: -5, -64, -100.
Számoljuk ki a következőket: 10+(-5), (-64)+64, 110+(-100).

Tört számok ábrázolása

Tört számok ábrázolására egyaránt használhatunk *fixpontos* és *lebegőpontos* megoldást is.

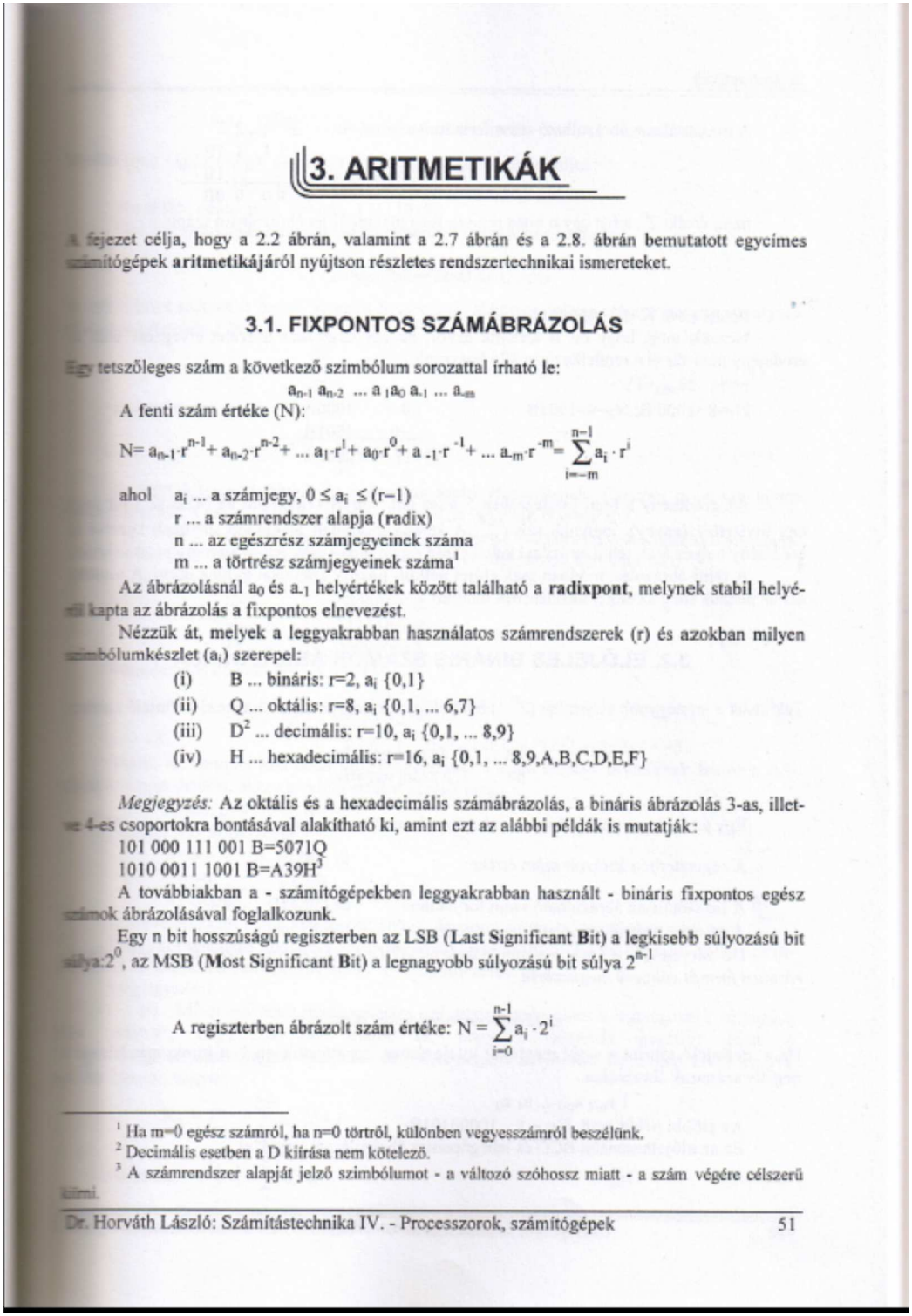
Fixpontos megoldásnál a kettedespont helyét rögzítjük: megmondjuk, hogy hány kettedes jegyig tároljuk a törtrészt. (Például: 32 bitből használjunk 8 bitet a törtrész és 24 bitet az egészek ábrázolására; használjuk az LSB sorrendet.) A kettes komplemens ábrázolás itt is természetes módon adódik.
Lebegőpontos számok ábrázolására használhatjuk az IEEE 754 lebegőpontos számformátumot [7].??
A 4 byte-os formátum röviden: 1 bit előjel (0: +, 1: -), 8 biten tároljuk a +127-tel eltolt karakterisztika értékét, 23 biten tároljuk a mantissza kettedespont utáni részét (előtte úgyis mindig 1-es van).
Gyakorlásként írjuk fel a 253,75-öt a fent megadott a kétféle formátumban!
A 4 byte-os IEEE 754 formátum előállítását a mellékelt példaprogramokkal ellenőrizhetjük.

W1: IEEE lebegőpontos számformátum – Wikipédia

http://hu.wikipedia.org/wiki/IEEE_lebeg%C5%91pontos_sz%C3%A1mform%C3%A1tum

Aritmetikai algoritmusok

Forrás Horváth L



3. ARITMETIKÁK

A fejezet célja, hogy a 2.2 ábrán, valamint a 2.7 ábrán és a 2.8. ábrán bemutatott egycímes számítógépek **aritmetikájá**ról nyújtson részletes rendszertechnikai ismereteket.

3.1. FIXPONTOS SZÁMÁBRÁZOLÁS

Egy tetszőleges szám a következő szimbólum sorozattal írható le:

$$a_{n-1}\, a_{n-2}\, \ldots\, a_1 a_0\, a_{-1}\, \ldots\, a_{-m}$$

A fenti szám értéke (N):

$$N = a_{n-1}\cdot r^{n-1} + a_{n-2}\cdot r^{n-2} + \ldots + a_1\cdot r^1 + a_0\cdot r^0 + a_{-1}\cdot r^{-1} + \ldots + a_{-m}\cdot r^{-m} = \sum_{i=-m}^{n-1} a_i \cdot r^i$$

ahol a_i ... a számjegy, $0 \le a_i \le (r-1)$
r ... a számrendszer alapja (radix)
n ... az egészrész számjegyeinek száma
m ... a törtrész számjegyeinek száma[1]

Az ábrázolásnál a_0 és a_{-1} helyértékek között található a **radixpont**, melynek stabil helyéről kapta az ábrázolás a fixpontos elnevezést.

Nézzük át, melyek a leggyakrabban használatos számrendszerek (r) és azokban milyen szimbólumkészlet (a_i) szerepel:

(i) B ... bináris: r=2, a_i {0,1}
(ii) Q ... oktális: r=8, a_i {0,1, ... 6,7}
(iii) D^2 ... decimális: r=10, a_i {0,1, ... 8,9}
(iv) H ... hexadecimális: r=16, a_i {0,1, ... 8,9,A,B,C,D,E,F}

Megjegyzés: Az oktális és a hexadecimális számábrázolás, a bináris ábrázolás 3-as, illetve 4-es csoportokra bontásával alakítható ki, amint ezt az alábbi példák is mutatják:

101 000 111 001 B=5071Q
1010 0011 1001 B=A39H[3]

A továbbiakban a - számítógépekben leggyakrabban használt - bináris fixpontos egész számok ábrázolásával foglalkozunk.

Egy n bit hosszúságú regiszterben az LSB (**L**ast **S**ignificant **B**it) a legkisebb súlyozású bit súlya:2^0, az MSB (**M**ost **S**ignificant **B**it) a legnagyobb súlyozású bit súlya 2^{n-1}

A regiszterben ábrázolt szám értéke: $N = \sum_{i=0}^{n-1} a_i \cdot 2^i$

[1] Ha m=0 egész számról, ha n=0 törtről, különben vegyesszámról beszélünk.
[2] Decimális esetben a D kiírása nem kötelező.
[3] A számrendszer alapját jelző szimbólumot - a változó szóhossz miatt - a szám végére célszerű írni.

Dr. Horváth László: Számítástechnika IV. - Processzorok, számítógépek 51

A maximálisan ábrázolható szám formátuma: $2^{n-1} \ldots 2^1 2^0$

1 1 1... 1 1B

Adjunk hozzá 1-et: 1B

1 0 0 0... 0 0B

az új érték: 2^n, tehát egy n bites regiszterben maximálisan ábrázolható szám:

$$N_{max}=2^n-1$$

Példa: n=8, N=5 ... 00000101B

Nézzük meg, hogy mi is történik akkor, ha egy összeadás művelet elvégzése után az eredmény nem fér el a rendelkezésre álló hosszon!

n=4 ... N_{max}=15

N_1=8=1000 B; N_2=9=1001B

8 = 1000B

+9 = + 1001B

15<17 = 10001B

Az eredmény a fenti kifejezésben 5 bites lett, vagyis keletkezett az MSB (2^3) bit felett egy **átvitelbit (carry)**, melynek súlya 2^4. A keletkező átvitelt megfelelő súlyúnak kezelve az eredmény helyes lesz, tehát az átvitel keletkezése - nem hiba - csak jelzés a programozó számára.

A fenti ábrázolási módban csak **előjel nélküli pozitív** számokat tudunk leírni. A továbbiakban nézzük meg az előjel kezelésének módját!

3.2. ELŐJELES BINÁRIS SZÁMOK ÁBRÁZOLÁSA

Tekintsük a legnagyobb súlyozású (2^{n-1}) bitet előjelnek, mégpedig a következő definíció szerint:

ha 0, a szám **pozitív**

ha 1, a szám **negatív**

Egy tetszőleges pozitív bináris szám formátuma: $0\ a_{n-2}\ a_{n-1} \ldots a_1\ a_0$ (*)

A regiszterben ábrázolt szám értéke: $N = \sum_{i=0}^{n-2} a_i \cdot 2^i$

A maximálisan ábrázolható szám formátuma: 011 ... 11B

A maximálisan ábrázolható szám értéke: $N= 2^{n-1} - 1$

De milyen lesz a negatív számok ábrázolása? *Legalább három elterjedt negatív számábrázolási formát célszerű megismerni.*

a) Előjel-abszolútértékes ábrázolás

Ha a definíció szerint a *-gal megjelölt kifejezésben az előjelbitben 1-et írunk, már kész is a negatív számunk ábrázolása.

$1\ a_{n-2}\ a_{n-1} \ldots a_1\ a_0$

Az előbbi példa: n=8, N= – 5 ...10000101B

Ez az előjelhasználat BCD és lebegőpontos ábrázolásnál terjedt el.

b) Inverz kódú ábrázolás

Minden bitet - így a (*)-gal megjelölt kifejezés előjelét is - invertáljuk!

$$1\,\bar{a}_{n-2}\,\bar{a}_{n-3}\ldots\bar{a}_1\,\bar{a}_0$$

Az előbbi példa: n=8, N= –5 ...11111010B

Néhány lyukszalag-vezérelt szerszámgépben használatos formula.

c) Komplemens kódú ábrázolás

Ha egy n bites számot és annak inverzét összeadjuk, akkor egy olyan számot kapunk, amelynek minden helyértékén 1 lesz.

$$\sum_{i=0}^{n-2} a_i \cdot 2^i + \sum_{i=0}^{n-2} \bar{a}_i \cdot 2^i = 2^{n-1} - 1 \qquad (**)$$

Átrendezve:

$$-N = -2^{n-1} + \sum_{i=0}^{n-2} \bar{a}_i \cdot 2^i + 1 = N_k \qquad (***)$$

A fenti kifejezést hívjuk **komplemens kódú** ábrázolásnak. Egészen pontosan kettes komplemens kód a neve! (Általánosítva r-es komplemens.) Mit is jelent ez?

(i) Értelmezés: az első tagot - az előjelet - negatív súlyozásúnak kell tekinteni!

(ii) Képzési algoritmus: a szám inverzéhez hozzá kell adni 1-et.

Példa: n=8, N= –5

N=5 ...	00000101B
inverze	11111010B
+1	00000001B

komplemens kódban N= –5..11111011B

$$2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0$$

N= –5 .... 1 1 1 1 1 0 1 1B= –128+64+32+16+8+2+1= –5

Tehát, ha komplemens kódú számábrázolásnál a szám előjelét negatívnak tekintve számítjuk ki annak értékét, akkor a valódi értéket kapjuk meg.

Példa: n=4, A=6, B= –6;

Mennyi lesz az összegük?

A=6 ...	0110B
B= –6....	1010B
Az összeg:	10000B

A műveletvégzés során keletkezett átvitel, de ennek itt nincs jelentősége, elhagyjuk.

Tehát komplemens kódban a zérus ábrázolása egyértelmű. (Előjel-abszolútértékes és inverz kódú ábrázolásnál van „pozitív zérus" és „negatív zérus" is.)

Megjegyzések:

(i) Mivel egy adott hardverelemre - pl. regiszterre - nincs a legmagasabb súlyozású bitre „ráírva", hogy „δ" az előjel és, hogy a súlyozás negatív, ezért a (***)-gal megjelölt összefüggést valójában az előjel negatív súlyozását figyelmen kívül hagyva kell értelmezni, vagyis:

$$N_k = 2^{n-1} + \sum_{i=0}^{n-2} \bar{a}_i \cdot 2^i + 1 \qquad (1)$$

másrészt: $$|N| = \sum_{i=0}^{n-2} a_i \cdot 2^i \qquad (2)$$

az (1) és a (2) összege: $N_k + |N| = 2^{n-1} + \sum_{i=0}^{n-2} \overline{a}_i \cdot 2^i + 1 + \sum_{i=0}^{n-2} a_i \cdot 2^i = 2 \cdot 2^{n-1}$

ahonnan megkapjuk a komplemens kód másik számítási módját:

$$N_k = 2^n - |N| \qquad (3)$$

Példa: n=3, N= –2

$N_k = 2^3 - |-2| = 6$ ... 110B, valóban –2.

(ii) Vegyünk egy 4 bites reverzibilis számláncot! Töröljük, majd számláltassuk v szafelé! A kódok sorban a következők lesznek: 0000, 1111, 1110, 1101, ... Ha jól megnéz akkor n=4 szóhossz esetén komplemens kódban ábrázolt 0, –1, –2, –3 ... számok, ami a vissz lé számlálás müveletét figyelembe véve helyes is.

A fent leírt úton is bevezethető a komplemens kódú számábrázolás!

3.3. ÖSSZEADÁS ÉS KIVONÁS KOMPLEMENS KÓDBAN

A számítógépek az egész számokat - programozási nyelvtől függetlenül - bináris fixpontos mában ábrázolják, és INTEGER-nek nevezik. Az ábrázolt számok nagysága szóhossztól f Az előforduló szóhosszak: 8, 12, 16, 32 bit. Ezeknél az értéktartomány:

-128/+127, –2048/+2047, –32k/+32k és –2G/+2G.

A továbbiakban nézzük meg, hogy milyen problémák adódhatnak két komplemens ban ábrázolt szám összeadása során!

A szóhossz legyen n bites, tehát a legmagasabb helyértékü bit - az előjel - súlyozása az átvitel súlyozása 2^n lesz. (A példákban a szóhossz legyen n=4 bit!)

(i) Mindkét operandus pozitív és az eredmény is pozitív. Itt a megoldás trivi S=X+Y.

Példa: X=3 ... 0011B
Y=4 ... 0100B
S=7 ... 0111B, ami korrekt eredmény.

(ii) Pozitív és negatív szám összege negatív eredménnyel. Ebben az esetben $|X|<|Y|$, akkor (3) alapján a negatív operandust $2^n - |Y|$ formában ábrázoljuk.

$S=|X|+2^n-|Y|=2^n-|Y-X|$, ami egy $|Y-X|$-ü negatív számnak felel meg.

Példa: X=3 ... 0011B
Y= –4 ... 1100B
S= –1 ... 1111B, ellenőrizve: –S= ... 0001, korrekt eredmény.

(iii) Pozitív és negatív szám összege pozitív eredménnyel. Itt a negatív operand $-|X|$, ha $|X|<|Y|$.

$S=2^n-|X|+|Y|=2^n+|Y-X|$, ami egy $|Y-X|$-ü pozitív számot és egy át - 2^n helyen - jelent.

Példa: X= –3 ... 1101B
Y= 4 ... 0100B
S= 1 ... 10001B, ha az átvitelt elhagyjuk: S= ...0001B, ko eredmény.

(iv) Két negatív operandus esetén az eredmény is negatív. Ebben az esetben: $2^n - |X|$ és $2^n - |Y|$ formában ábrázoljuk a számokat.

$S = 2^n - |X| + 2^n - |Y| = 2^n + 2^n - |Y+X|$, ahol az első tag egy átvitelbit, a következő tagok pedig egy $|Y+X|$-ű negatív számot jelentenek.

Példa: X= –3 ... 1101B
Y= –4 ... 1100B
11001B, hagyjuk el az átvitelt!
S=–7... 1001B, ellenőrizzük!
–S= 7 ... 0111B, korrekt eredmény.

A fentiekből látszik, hogy komplemens kódú számok összeadása esetén, ha mindkét operandus negatív, *keletkezik* átvitel, ha az operandusok különböző előjelűek *keletkezhet* átvitel, ha az operandusok pozitívak *nem keletkezik* átvitel. A keletkező átvitelt - minden esetben - figyelmen kívül kell hagyni!

Mielőtt továbblépnénk, nézzünk még két számpéldát!

a) X=7 ... 0111B
Y=6 ... 0110B
S=X+Y=13... 1101B, a két **pozitív** operandus összege **negatív** lett! Nem fért el az eredmény az ábrázolási tartományban **túlcsordulás** (Overflow) keletkezett.

b) X= –7 ... 1101B
Y= –6 ... 0110B
S=X+Y= –13 ... 0111B, két **negatív** operandus összege **pozitív** lett! Ismét túlcsordulás keletkezett, mivel –13 komplemens kódban nem ábrázolható 4 biten.

Túlcsordulás csak komplemens kódú ábrázolásnál keletkezhet. Az operandusok és az eredmény előjeléből az alábbi logikai egyenletekkel számítható ki:

összeadás esetén: $OVF_+ = \overline{X}_e \cdot \overline{Y}_e \cdot S_e + X_e \cdot Y_e \cdot \overline{S}_e$
kivonás esetén: $OVF_- = \overline{X}_e \cdot Y_e \cdot S_e + X_e \cdot \overline{Y}_e \cdot \overline{S}_e$

Az előjel (S), az átvitel (CY) és a túlcsordulás (OVF) az úgynevezett programállapot- jelző (angolul: Flag) bitek, vagy más néven feltétel kódok CC (Condition Code) közé tartoznak. Értékük minden aritmetikai és logikai utasítás után automatikusan beáll.

3.4. BINÁRIS SZORZÁSI ALGORITMUSOK

A szorzási algoritmusok erősen függenek a számábrázolástól. Az egyszerűbb algoritmusokkal csak előjel nélküli számok, a bonyolultabbakkal komplemens kódban ábrázoltak is összeszorozhatók. Nézzünk az egyszerűbb esetre egy algoritmust!

a) Szorzás ismételt összeadással

Írjuk fel a műveletet n bites operandusokkal:

$$A \cdot B = A \cdot (b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + ... + b_1 2^1 + b_0 2^0)$$

Átalakítva: $A \cdot B = (...(0 + 2^n \cdot A \cdot b_0)2^{-1} + 2^n \cdot A \cdot b_1)2^{-1} + ... + 2^n \cdot A \cdot b_{n-1})2^{-1}$

Elemezzük a fenti összefüggést

(i) A műveletet a legkisebb súlyozású b_0 bittel kezdjük.

(ii) Nem a szorzandót, hanem annak n-szer balra léptetett változatát - 2^n ·A-t - szorozzuk meg. Ebből következik, hogy a szorzat legkisebb helyértékű bitje n bittel jobbra lesz a szorzandó azonos helyértékű bitjétől.

(iii) A szorzásokat összegzések követik.

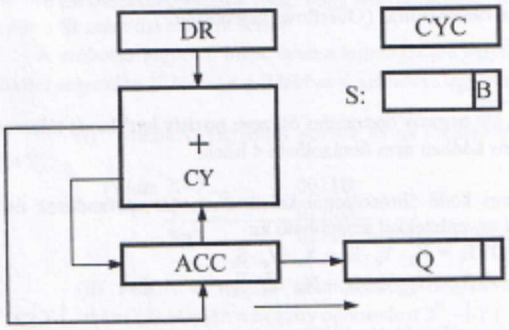
(iv) A részletösszeg tárolót kezdetben törölni kell!

(v) A részletösszeget jobbra kell léptetni. Ezt jelenti a 2^{-1}-vel való szorzás.

(vi) Az algoritmus n lépéses.

(vii) Az algoritmusban nincs előjel kezelés, tehát csak előjel nélküli pozitív számok szorzására alkalmas. (Természetesen az operandusok előjelének antivalenciája megadja a szorzat előjelét.)

Az algoritmushoz tartozó regiszter elrendezés a 3.1. ábrán látható.



3.1 ábra Szorzás ismételt összeadással regiszterelrendezése

A regiszterek tartalma szorzás *előtt* (amit előre be kell állítani):

DR - szorzandó

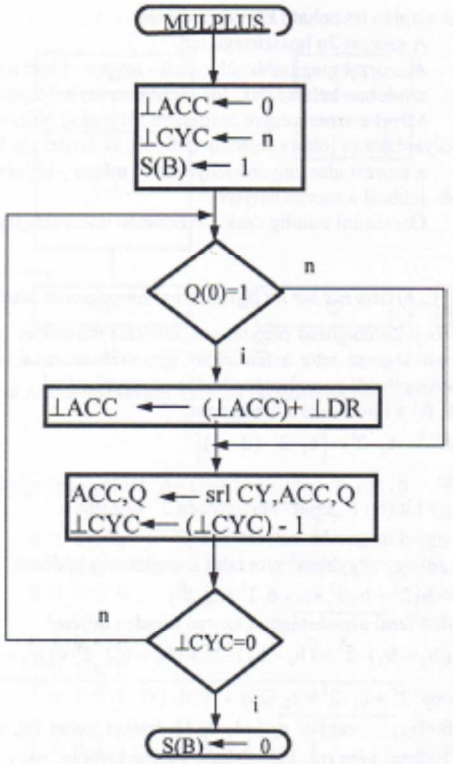
Q - szorzó

A regiszterek tartalma szorzás *után*:

ACC,Q - szorzat

DR - szorzandó

A folyamatábra a 3.2. ábrán látható. Az ábrákon megjelenő CYC (**Cycle Counter**) **ciklusszámláló** a szorzó bitjeinek számával (n) töltődik fel a **kezdeti beállítás** állapotában. Az algoritmus minden lépését CYC dekrementálása követi, tehát CYC=0 jelzi a művelet végét. A szorzó áramkör mindenkori állapotát az S (Status) regiszter B (**B**usy) bitje jelzi. A művelet elején S(B)←1 állításával az eszköz **foglaltságát**, S(B)←0 állapottal az eszköz **üzemkészségét** (Ready) jelzi a külvilág felé.



3.2. ábra Szorzás ismételt összeadással folyamatábrája

Nézzünk egy számpéldát is: A=7 ; B=5 . (A számpéldában tekintsünk minden számot bi-nárisnak, és csak az ettől eltérőeket jelöljük.) Legyen n=3!

	szorzandó	$b_2b_1b_0$ szorzó	
	1 1 1	1 0 1	
Az induló részletösszeg....	0 0 0		
$2^n \cdot A \cdot b_0$ ...	1 1 1		
Első részletösszeg	1 1 1		léptetés jobbra
	0 1 1	1	
$2^n \cdot A \cdot b_1$ ...	0 0 0		
Második részletösszeg ...	0 1 1	1	léptetés jobbra
	0 0 1	1 1	
$2^n \cdot A \cdot b_2$ ...	1 1 1		
Harmadik részletösszeg ...	1 0 0 0	1 1	léptetés jobbra; *a keletkezett átvitel: belép*
A szorzat	1 0 0	0 1 1	

A számpélda alapján levonható következtetések:

(i) A szorzat 2n hosszúságú lett.

(ii) A szorzat magasabb súlyozású - **major** - bitjei a részletösszeg tárolóban keletkeztek. Ide mindig az átvitel lépett be.

(iii) Mivel a szorzó egyre magasabb súlyozású bitjeire van szükség, a szorzót is célszerű folyamatosan jobbra léptetni. (Mindig az átvitel lép be!)

(iv) a szorzat alacsonyabb súlyozású - **minor** - bitjei folyamatosan beléptethetők jobbról a szorzó helyére.

(v) Összeadni mindig csak a szorzandó hosszúságában, tehát n biten kellett.

(vi)

b) Szorzás két bit figyelésével komplemens kódban

A valósidejű digitális jelfeldolgozás nagyon sok szorzási műveletet igényel. Ezen feladatok tömeges elterjedése szükségessé tette a fentieknél gyorsabb szorzási algoritmusok kifejlesztését, melyben a számok közvetlenül komplemens kódban szorozhatók.[4]

Először írjuk fel a következő azonosságot:

$$b_i \cdot 2^i = b_i \cdot 2^{i+1} - b_i \cdot 2^i = \left[b_i \cdot 2^i \cdot (2-1) \right]$$

Értelmezzük:

(i) Egy bit (b_i) a „saját" helyértékén 2^i *negatív,*

(ii) eggyel magasabb helyértéken (2^{i+1}) *pozitív.*

Most írjunk fel egy négybites[5] szorzatot komplemens kódban:

$$A \cdot B = A \cdot (-b_3 2^3 + b_2 2^2 + \ldots + b_1 2^1 + b_0 2^0)$$

Alkalmazzuk a fenti azonosságot a szorzó minden bitjére!

$$A \cdot B = A \cdot \left[(b_2 - b_3) \cdot 2^3 + (b_1 - b_2) \cdot 2^2 + (b_0 - b_1) \cdot 2^1 + (b_{-1} - b_0) \cdot 2^0 \right] =$$

$$A \cdot (c_3 \cdot 2^3 + c_2 \cdot 2^2 + c_1 \cdot 2^1 + c_0 \cdot 2^0) \qquad (*)$$

ahol: $c_i = b_i - b_{i+1}$ vagyis: $c_i \in \{-1, 0, 1\}$ értéket vehet fel, tehát azonos b_i , b_{i-1} bitek esetén nem kell műveletet végezni, különbözők esetén kivonni, vagy összeadni kell. A b_{-1} a 2^{-1} súlyozású bit lenne, egészeknél természetesen zérus. Az algoritmust leíró forma (*)-ból:

$$A \cdot B = ((((0 + 2^4 \cdot A \cdot c_0)2^{-1} + 2^4 \cdot A \cdot c_1)2^{-1} + 2^4 \cdot A \cdot c_2)2^{-1} + 2^4 \cdot A \cdot c_3)2^{-1}$$

Az algoritmus a fentiek alapján általánosítva:

(i) A szorzandót n-szer balra kell léptetni.

(ii) A szorzást a legkisebb helyértéken kell kezdeni.

(iii) Kezdetben $b_{-1} = 0$.

(iv) Mindig két szorzóbit figyelésével, hol össze kell adni, hol ki kell vonni a szorzandót és a részletösszeget, hol pedig nem kell műveletet végezni.

(v) A részletösszeget és a szorzót jobbra kell - aritmetikailag - léptetni.

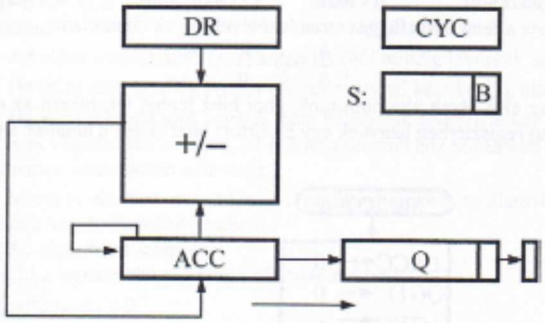
Az algoritmus regiszterelrendezése a 3.3. ábrán, folyamatábrája a 3.4. ábrán látható.

Az egyes regiszterek tartalma szorzás előtt:

	DR - szorzandó	Q - szorzó
Szorzás után:	DR - szorzandó	ACC,Q - szorzat

[4] Az irodalom - felfedezőjéről - Booth-féle kétbit-figyeléses algoritmusnak is nevezi.

[5] A négybites szóhossz nem megy az általánosság rovására!



3.3. ábra Kétbit-figyeléses szorzás regiszterelrendezése

Végezetül nézzünk egy számpéldát: A=12=01100; B= –2=11110

		DR	Q	Q_{-1}	
		0 1 1 0 0			
		ACC			nincs művelet
		0 0 0 0 0	1 1 1 1 0	0	léptetés jobbra: 1
					kivonás
		0 0 0 0 0	0 1 1 1 1	0	
–DR	+	1 0 1 0 0			
		1 0 1 0 0	0 1 1 1 1	0	léptetés jobbra: 2
					nincs művelet
		1 1 0 1 0	0 0 1 1 1	1	léptetés jobbra: 3
					nincs művelet
		1 1 1 0 1	0 0 0 1 1	1	léptetés jobbra: 4
					nincs művelet
		1 1 1 1 0	1 0 0 0 1	1	léptetés jobbra: 5
A szorzat:		1 1 1 1 1	0 1 0 0 0	1	

Értelmezzük: - ACC,Q ... 00000 11000=24, korrekt eredmény!

Meg kell jegyezni, hogy a módosított Booth-féle szorzási algoritmus kettőnél több bitet figyel, illetve a szorzást és a léptetést egyszerre elvégző hálózat több bites. Így ez az algoritmus sokkal gyorsabban végzi el a műveletet.

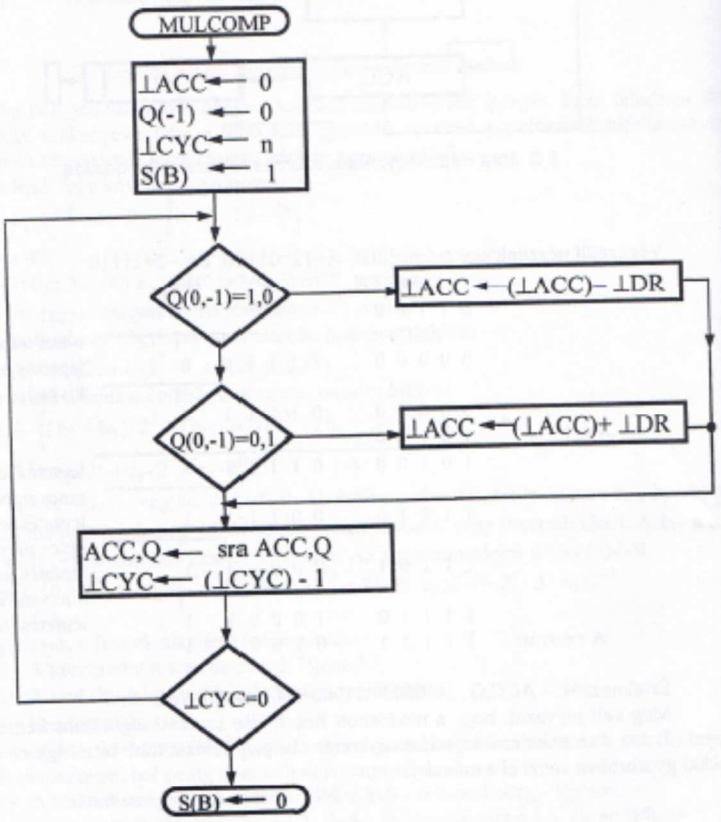
3.5. BINÁRIS OSZTÁS

Az osztási algoritmusok közül nézzük meg a legelterjedtebbet, az úgynevezett **osztás visszaállítással-t**! Az algoritmus nem kezel előjelet, tehát csak pozitív egészekkel képes a műveletet elvégezni.

Először tisztázzuk az operandusokat: A=Q·B+R
ahol: A - **osztandó**; B - **osztó**; Q - **hányados**; R - **maradék**
Az eredményt a fenti összefüggés átrendezésével fogjuk ellenőrizni:

$$\frac{A}{B} = Q + \frac{R}{B}$$

Az osztás az első olyan algoritmusunk, ahol hiba léphet fel, hiszen az osztó **nem lehet zérus**! Az S (Status) regiszterben hagyunk egy E (Error) bitet ennek a hibának a jelzésére.



3.4. ábra A kétbit-figyeléses szorzás folyamatábrája

Az algoritmusban a részmaradék képzési szabálya a következő:

$r_i = 2 \cdot r_{i+1} - q_i \cdot B$

Az algoritmus:

(i) Az előző részmaradék (r_{i+1}) kétszereséből mindig kivonjuk az osztót (B).

(ii) Ha az új részmaradék pozitív (ilyenkor átvitel keletkezik), akkor $q_i=1$.

(iii) Ha az új részmaradék negatív (nem keletkezik átvitel), akkor $q_i=0$, és a kivonást nem végezhettük volna el, tehát a részmaradékhoz hozzá kell adni az osztót. (*Innen a visszaállítás elnevezés.*)

(iv) Mivel az előző részmaradék (r_{i+1}) indexe nagyobb, az algoritmust a legmagasabb helyértéken kell kezdeni!

(v) Az algoritmus *n* lépéses.

Induljunk ki a legnagyobb súlyozású részmaradékból!

$r_{n-1} = 2 \cdot r_n - q_{n-1} \cdot B$

$r_{n-2} = 2 \cdot r_{n-1} - q_{n-2} \cdot B = 4 \cdot r_n - 2 \cdot q_{n-1} \cdot B - q_{n-2} \cdot B$

$r_{n-3} = 2 \cdot r_{n-2} - q_{n-3} \cdot B = 2^3 \cdot r_n - 2^2 \cdot q_{n-1} \cdot B - 2 \cdot q_{n-2} \cdot B - q_{n-3} \cdot B$

:

$r_0 = 2^n \cdot r_n - 2^{n-1} \cdot q_{n-1} \cdot B - \ldots - 2 \cdot q_1 \cdot B - q_0 \cdot B$

$r_0 = 2^n \cdot r_n - B \cdot \sum_{i=0}^{n-1} q_i \cdot 2^i$

mivel $r_0 = R$ a maradék; és $R = A - B \cdot Q$; valamint $A = 2^n \cdot r_n$

így kezdéskor: $r_n = A \cdot 2^{-n}$ (Tehát az osztót n-szer jobbra kell léptetni)

A hányados: $Q = \sum_{i=0}^{n-1} q_i \cdot 2^i$, ami egész szám.

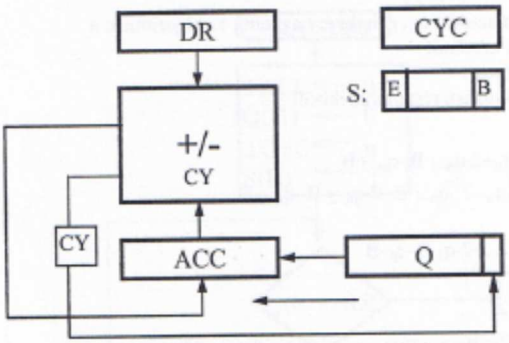
Nézzünk egy számpéldát: A=11 D=1011; B=3 D=0011; $n=4$

B(osztó)		0 0 1 1		
$r_4 = A \cdot 2^{-4}$		0 0 0 0	1 0 1 1	
$2 \cdot r_4$		0 0 0 1	0 1 1	
−B	+	1 1 0 1		
Negatív (nincs átvitel)		1 1 1 0		$q_3=0$
+ B	+	0 0 1 1		*visszaállítás!*
r_3		1 0 0 0 1	0 1 1	
$2 \cdot r_3$		0 0 1 0	1 1	
−B	+	1 1 0 1		
Negatív (nincs átvitel)		1 1 1 1		$q_2=0$
+ B	+	0 0 1 1		*visszaállítás!*
r_2		1 0 0 1 0	1 1	
$2 \cdot r_2$		0 1 0 1	1	
−B	+	1 1 0 1		
Pozitív (van átvitel) r_1		1 0 0 1 0	1	$q_1=1$
$2 \cdot r_1$		0 1 0 1		
−B	+	1 1 0 1		
Pozitív (van átvitel) r_0		1 0 0 1 0		$q_0=1$

A hányados: Q=0011=3; A maradék: r_0=R=0010=2

Ellenőrizzük az eredményt $\frac{A}{B}=Q+\frac{R}{B}$ alapján: $\frac{11}{3}=3+\frac{2}{3}=\frac{9}{3}+\frac{2}{3}=\frac{11}{3}$

Az osztási algoritmus regisztere elrendezése a 3.5. ábrán, folyamatábrája a 3.6. ábrán látható.



3.5. ábra Az osztásregiszter elrendezése

Az egyes regiszterek tartalma művelet elvégzése *előtt*:

DR - osztó
Q - osztandó

A művelet *után*:

DR - osztó
Q - hányados
ACC - maradék

Megjegyzés: Ha előjel-abszolútértékes számokat szeretnénk elosztani ezzel az algoritmussal, akkor a hányados előjele az operandusok előjelének antivalenciája, a maradék előjele pedig az osztandó előjele lenne.

3.6. LEBEGŐPONTOS SZÁMÁBRÁZOLÁS

Ha nem fér el a számunk a fixpontosok által definiált tartományban, akkor át kell térnünk a **lebegőpontos** ábrázolásra.

Egy lebegőpontos szám:

$$N=(-1)^s \cdot f \cdot r^e \qquad (*)$$

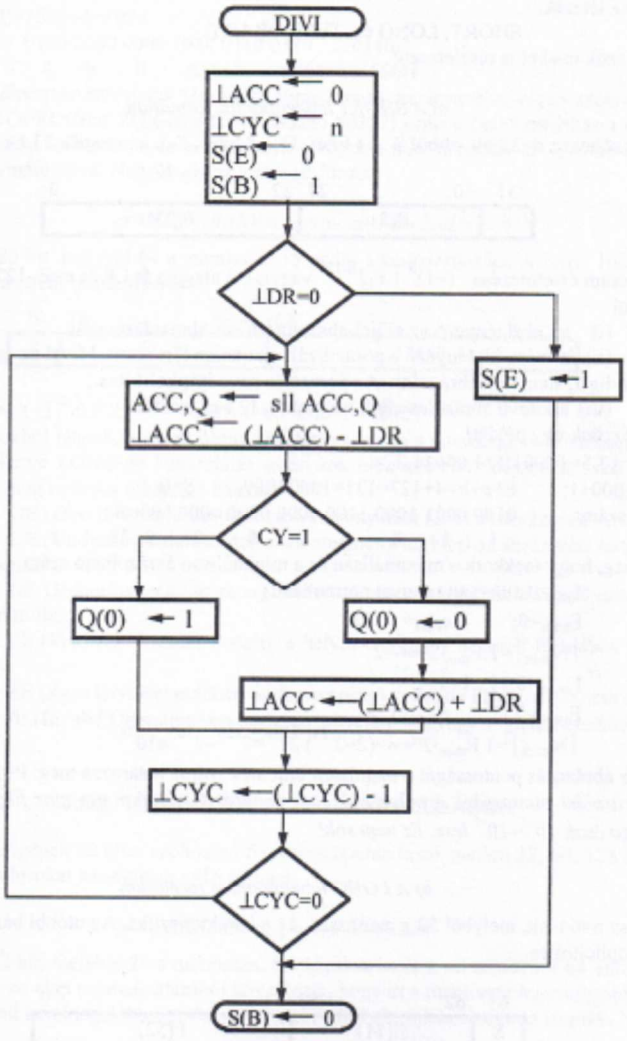
ahol: S (Sign) ... a szám előjele (Előjel-abszolútértékes ábrázolás)
f...a szám tört része, a szám normalizált értéke (Egyes irodalmakban m a jele és mantissza az elnevezése)
r ...a számrendszer alapja
e (exponent) ... a szám kitevője (Egyes irodalmakban karakterisztika az elnevezése)

Például: $-1.75 \cdot 10^{+2}$, ahol S... negatív; f ...1.75; r ...10; e ...+2

A számítógépekben alkalmazott lebegőpontos számábrázolások alapvetően csak egy helyen térnek el a fenti példától, ez pedig a kitevő ábrázolása. Az exponensnél *nem* a mantisszánál is használt előjel-abszolútértékes és *nem* a megszokott komplemens kódú számábrázolás terjedt el, hanem az ún. **eltolt nullpontú**. Ennek lényege egy egyszerű, lineáris transzformáció: a kisebb kitevőt (a legnegatívabbat) zérussá transzformáljuk egy konstans hozzáadásával: E= e+b

ahol E ... az ábrázolt kitevő
e ... a kitevő
b ... a transzformáció értéke.

A bázis értéke célszerűen a rendelkezésre álló számtartomány fele: $b=E_{max}/2$. Ezzel a kissé szokatlan - eltolt nullpontú - ábrázolással elértük, hogy a már csak pozitívnak ábrázolt kitevők segítségével két lebegőpontos szám összehasonlítása nagyon egyszerűvé válik.



3.6 ábra Az osztás folyamatábrája

3.6.1. BINÁRIS ALAPÚ ÁBRÁZOLÁSOK

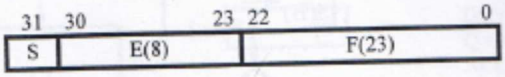
Az eddig leírt ábrázolási forma az IEEE - P754 számú szabványként került forgalomba. Legismertebb alkalmazási helye az IBM PC-k lebegőpontos aritmetikája. A szabványban három ábrázolási forma létezik:

SHORT, LONG és TEMPORARY.

Nézzük ezeket is részletesen:

a) A SHORT lebegőpontos formátum

A szóhossz n=32 bit, ebből S ... 1 bites, E ... 8 bites, F ... a maradék 23 bit. A bázis: 127.



A szám értelmezése: $(-1)^S \cdot 1.F \cdot 2^{E-127}$ vagyis (*) alapján f=1.F és e=E−127

ahol

(i) az első tényező az előjel-abszolútértékes ábrázolásra utal

(ii) a második tényező a normalizált mantissza tört része. Mivel az egész rész mindig 1, nem kell ábrázolni. Az ábrázolás neve **implicitbites.**

(iii) a kitevő ábrázolás eltolt nullpontú, 127-es bázissal.

Nézzünk egy példát!

N=17.5=10001.1=1.00011·2^4

F=00011; E=e+b=4+127=131=10000011; S=0

A szám: 0100 0001 1000 1100 0000 0000 0000 0000B

4 1 8 C 0 0 0 0 H

Nézzük meg, hogy mekkora a maximálisan és a minimálisan ábrázolható szám abszolút értéke:

f_{min}=1.0B=1.0D (mivel normalizált)

E_{min}=0; e_{min}= −127

$|N_{min}|=1.F_{min}\cdot 2^{e_{min}}=2^{-127}$

F_{max}=0.11...1=1−2^{-23}

E_{max}=255; e_{max}=128

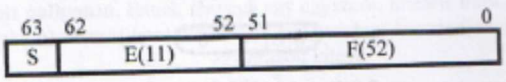
$|N_{max}|=1.F_{max}\cdot 2^{e_{max}}=(2-2^{-23})\cdot 2^{128}=2^{129}-2^{105}\cong 10^{38}$

Az ábrázolás pontosságát a mantissza bitjeinek száma határozza meg: $P=2^{-24}\cong 10^{-7}$

Ez az ábrázolási pontosság! A műveletek csak rontják. Várhatóan egy sinx függvény értékének pontossága csak 10^{-3}–10^{-4} lesz. Ez nem sok!

b) A LONG lebegőpontos formátum

A szóhossz n=64 bit, melyből 52 a mantissza, 11 a karakterisztika. Az utóbbi bázisa 1023, az ábrázolás implicitbites.



$N=(-1)^S \cdot 1.F \cdot 2^{E-1023}$ vagyis f=1.F és e=E−1023

A hosszabb karakterisztika miatt jelentősen megnött az ábrázolási tartomány ($N_{max} \cong 2^{1023}$), a hosszabb mantissza miatt pedig a pontosság nőtt meg,($P_{max} \cong 2^{-53}$)

Nézzünk egy példát: a szám legyen végtelen szakaszos bináris tört! N=3,3

11.010011001 1001 1001 ...=1.101001 1001 1001...$\cdot 2^1$

S=0; E=1023+1=1024

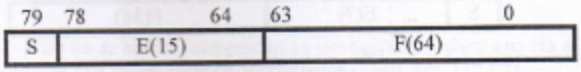
A szám: 0100 0000 0000 1010 0110 0110 ..... 0110B

4 0 0 A 6 66666666666H

A lebegőpontos műveletek elvégzése során romlik az ábrázolás véges szóhossza miatti pontosság. A KOPROCESSZOR-ok (pl.8087,80287,80387) ezért a belső számításra egy hosszú átmeneti ábrázolást használnak, és a műveletek elvégzése után kerekítik az eredményt SHORT vagy LONG formátumúvá. Nézzük ezt az átmeneti formát:

c) TEMPORARY lebegőpontos ábrázolás

A szóhossz n=80 bit, melyből 64 a mantissza, 15 pedig a karakterisztika. A bázis 16383-ra adódik. Az ábrázolás **nem implicitbites**!



$N=(-1)^S \cdot 0.F \cdot 2^{E-16383}$ vagyis f=0.F ás e=E–16383

Az adatokból látszik, hogy az ábrázolási tartomány és a pontosság is jelentősen megnőtt. A SHORT-tá illetve LONG-gá konvertálás során sok hiba adódhat. Végezetül csak felsorolás szerűen nézzük meg ezeket a hibajelző biteket!

(i) OE (**O**verflow) túlcsordulás: az eredmény nem fér el az ábrázolási tartományban

(ii) UE (**U**nderflow) alulcsordulás: az eredmény túl kicsi az ábrázolási tartományhoz képest, de nem nulla.

(iii) DE (**D**enormalized Operand) normalizálatlan operandus: a kitevő minimális, de a mantissza nem nulla.

(iv) PE (**P**recision Inexact Result): a helyes eredmény az adott formában nem ábrázolható pontosan.

(v) ZE (**Z**ero Divider) nullával osztás: az osztó nulla, de az osztandó nem az.

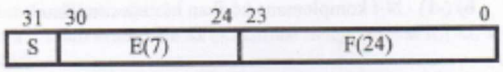
(vi) IE (**I**nvalid Operation) érvénytelen művelet: 0/0, vagy végtelen eredmény.

3.6.2. TIZENHATOS ALAPÚ ÁBRÁZOLÁSOK

A „nagy" számítógépek 32 bites szóhosszú fixpontos operandusok mellett 32, 64, 128 bites lebegőpontos operandusokat használnak r=16 alappal.

a) REAL lebegőpontos formátum

A szóhossz n=32 bit, melyből 24 a mantissza, 8 a karakterisztika, az adatokból 64 eltolású bázis adódik. Az r=16-os alap normalizálásából következik, hogy itt a mantissza legmagasabb súlyozású négy bitjén kell lenni legalább egy darab 1-esnek, tehát nincs lehetősége az implicit bitnek.



$N=(-1)^S \cdot 0.F \cdot 16^{E-64}$ vagyis f=0.F és e=E–64

Nézzük ismét egy példát: N= –17,25

S=1 (negatív); $10001.01=0.0001\ 0001\ 01\cdot16^{2}$ → Itt kell 1-es bitnek lennie!

E=64+2=66

1100 0010 0001 0001 0100 0000 0000 0000B

C 2 1 1 4 0 0 0 H

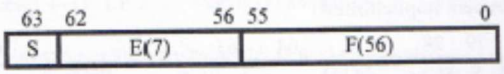
Ennél az ábrázolásnál: $N_{max}=(1-2^{-24})\cdot16^{63}\cong2^{252}\cong10^{76}$,

A pontosság maximuma pedig[6]: $P_{max}\cong2^{-24}$,

Ahol ez a pontosság kevés, ott dupla szóhosszat használunk.

b) DOUBLE lebegőpontos formátum

A 64 bitből 56 a mantissza, amely meghatározza a pontosságot, a 7 bites kitevő pedig a REAL-hoz hasonló ábrázolási tartományt



$N=(-1)^{S}\cdot0.F\cdot16^{E-64}$ vagyis f=0.F és e=E–64

Néhány, főként térképészeti számításnál még ez a pontosság is kevés, ezért tovább emelték a szóhosszt.

c) EXTENDED lebegőpontos formátum

A 120 bites mantisszával 10^{-36}-ra - hihetetlen nagyra nőtt meg a pontosság.

$N=(-1)^{S}\cdot0.F\cdot16^{E-64}$ vagyis f=0.F és e=E–64

Mindenképpen meg kell jegyeznünk, hogy a pontosság növelése nem csak dupla/négyszeres memória helyfoglalást, hanem négy-tízszeres végrehajtási időt is eredményez. Erről - vagyis a lebegőpontos összeadás és szorzás műveletekről - a példatárban olvashatunk.

3.7. GYAKORLÓ FELADATOK

3.7.1. Egy 16 bites komplemens kódban ábrázolt bináris szám hexadecimális formában: N = 09ABH. Írja fel

a) N-t decimálisan

b) (-1) · N-t komplemens kódban oktálisan!

3.7.2. Egy 12 bites komplemens kódban ábrázolt bináris szám oktális formában: N = 1514Q. Írja fel

a) N-t decimálisan

b) (-1) · N-t komplemens kódban hexadecimálisan!

[6] Azért a maximum, mert lehet, hogy csak 21 érvényes bit van.

3.7.3. N = FF6EH 16 bites komplemens kódú bináris szám hexadecimális formában.
Írja fel a) N-t decimálisan
b) (–1) · N-t oktálisan
c) 16 ·N-t hexadecimálisan
d) 15N-t hexadecimálisan! (15N = 16N –N !)

3.7.4. M = 584 decimális szám. Írja fel (–1) · M-t 16 bitesen
a) komplemens kódban
b) inverz kódban
c) előjel-abszolútértékesen!
(Mindig hexadecimálisan!)

3.7.5. Végezze el a kijelölt műveleteket! (n = 8 bites , kettes komplemens ábrázolás)
A = 6CH B = –15D C = 11101100B
N = A+B – C
Tüntesse fel a CY-t és az OVF-t!

3.7.6. Mutassa be az alábbi számpéldán az ismételt összeadású szorzás algoritmusát! Végezze el a szorzást Booth-féle algoritmussal is!
(13)· (–7)

3.7.7. Készítse el az alábbi adatokkal rendelkező, komplemens kódú számokat összeszorozni képes egység tervét és működésének folyamatábráját!
- Az egyik operandus 8 bites
- A másik operandus 16 bites

3.7.8. Végezze el a következő műveletet binárisan kódolt számokkal!
(13) / (3)

3.7.9. Számítsa ki az alábbi kifejezés értékét binárisan kódolt számokkal!
N = ((5)·(4 – 7)) / (2)

3.7.10. Írja fel a 3.7.3. példa N számát lebegőpontos SHORT és REAL formátumban! (Hexadecimálisan is!)

3.7.11. Írja fel a 3.7.4. példa N számát lebegőpontos SHORT és REAL formátumban! (Hexadecimálisan is!)

3.7.12. M = –63,4. Ábrázolja SHORT formában! Írja fel hexadecimálisan is!

3.7.13. M = –33,3. Ábrázolja REAL formában! Írja fel hexadecimálisan is!

3.7.14. Egy szám hexadecimális formában C0F00000H.
Mekkora a szám értéke, ha az ábrázolást lebegőpontos SHORT formátumúnak tekintjük?
Mekkora a szám értéke, ha az ábrázolást lebegőpontos REAL formátumúnak tekintjük?

3.7.15. Egy szám hexadecimális formában 40B0000000000000H.
Mekkora a szám értéke, ha az ábrázolást lebegőpontos LONG formátumúnak tekintjük?
Mekkora a szám értéke, ha az ábrázolást lebegőpontos DOUBLE formátumúnak tekintjük?

3.7.16. Adott négy lebegőpontos szám hexadecimális formában:
A = 40A0000 B = 41400000H C = 41880000H
a) Mekkora C értéke, ha SHORT az ábrázolási forma?
b) Mekkora C értéke, ha REAL az ábrázolási forma?
c) Melyik ábrázolási forma esetén lehet: C = A + B?
d) Adja meg a B szám 16-szorosát,
- ha a számábrázolás SHORT,
- ha a számábrázolás REAL!

3.7.17. Adott négy lebegőpontos szám hexadecimális formában:
A = 3FCCCCCCH B = 4040000000000000H C = 4000000000000000H
D = 4210000000000000H
a) Az A szám SHORT formátumú.
Adja meg LONG és REAL formátumban hexadecimálisan is!
b) Adja meg 16·A-t REAL és LONG formátumban!
c) Melyik ábrázolási forma esetén lehet: B = C · D ?

3.7.18.(*) Egy lebegőpontos SHORT ábrázolású szám: 40B66666H.
Mennyi a pontos decimális értéke?

3.8. ÚJ FOGALMAK A FEJEZETBEN

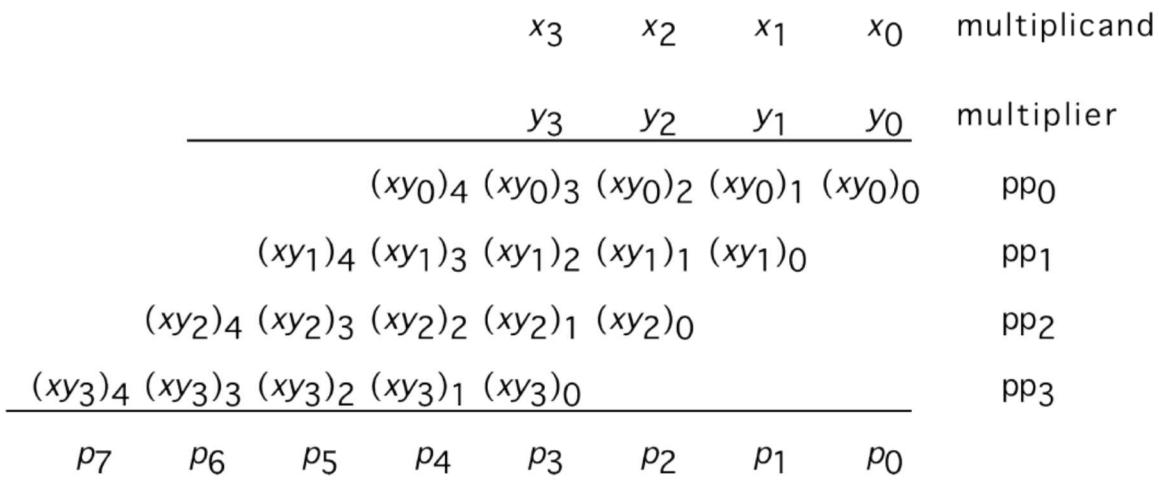
átvitelbit (carry), 52
B (Busy) bit, 56
bináris, 51
ciklusszámláló, 56
CYC (Cycle Counter), 56
DE (Denormalized Operand), 65
decimális, 51
double, 66
E (Error) bit, 60
e (exponent), 62
előjel nélküli pozitív, 52
eltolt nullpontú, 62
extended, 66
fixpontos, 51
hexadecimális, 51
implicitbit, 65
integer, 54
IE (Invalid Operation), 65
kétbit-figyeléses, 59
kezdeti beállítás, 56
komplemens kód, 53
lebegőpontos ábrázolás, 62
LONG, 64
LSB (Last Significant Bit), 51
major, 58
minor, 58
MSB (Most Significant Bit), 51
OE (Overflow), 65
oktális, 51
osztás visszaállítással, 59
PE (Precision Inexact Result), 65
radix, 51
radixpont, 51
REAL, 65
S (Status) regiszter, 56
SHORT, 64
TEMPORARY, 64
UE (Underflow), 65
üzemkészség (Ready), 56
ZE (Zero Divider), 65

Szorzók

Forrás: Forrás: - haszn ---- Ch6CSDA -- NUMBERS -- ALU algorithms .pdf

42-43 old-44

Fig. 6.5 Digital Multiplication Schema



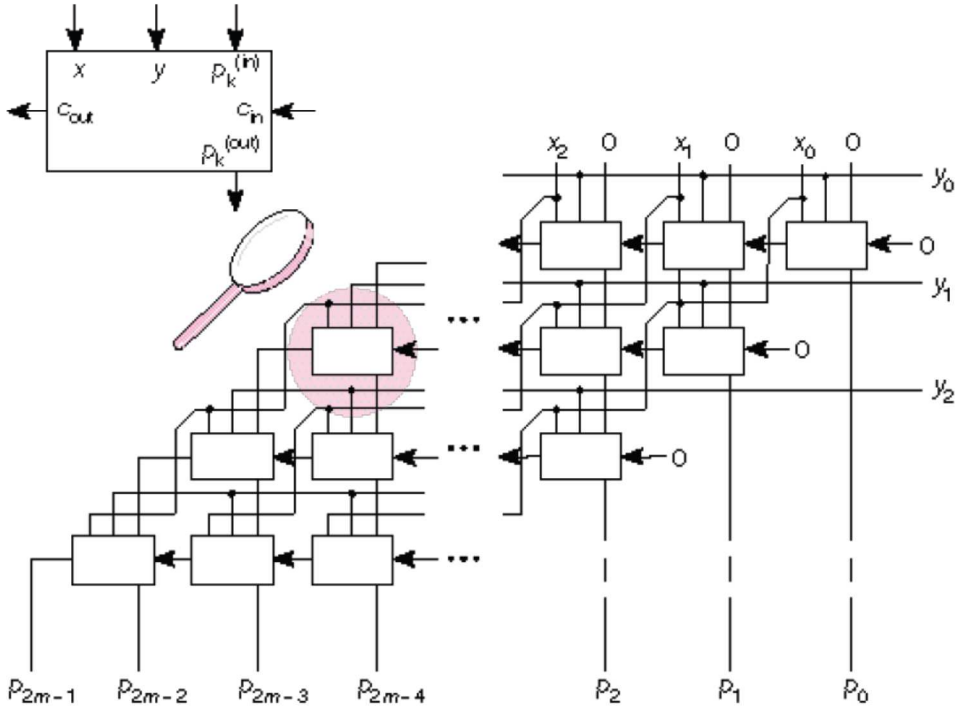
p: product pp: partial product

Serial By Digit of Multiplier, Then By Digit of Multiplicand

```
1.      for i := 0 step 1 until 2m-1
2.              p_i := 0;
3.      for j := 0 step 1 until m-1
4.              begin
5.                      c := 0;
6.                      for i := 1 step 1 until m-1
7.                              begin
8.                                      p_{j+i} := (p_{j+i} + x_i y_j + c) mod b;
9.                                      c := ⌊(p_{j+i} + x_i y_j + c)/b⌋;
10.                             end;
11.                     p_{j+m} := c;
12.             end;
```

- If $c \le b-1$ on the RHS of 9, then $c \le b-1$ on the LHS of 9 because $0 \le p_{j+i}, x_i, y_j \le b-1$

Fig. 6.6 Parallel Array Multiplier for Unsigned Base b Numbers



Forrás: - haszn ---- Ch6CSDA -- NUMBERS -- ALU algorithms .pdf

81. old