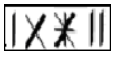
1. Az absztrakt adattípus

Az informatikában az adat alapvető szerepet játszik. A számítógép, mint automata, adatokat gyűjt, tárol, dolgoz fel (alakít át) és továbbít. Mi adatnak fogunk tekinteni minden olyan információt, amelynek segítségével leírunk egy jelenséget, tanulmányunk tárgyát, vagy annak egy részét. Szinte bármi lehet adat. Adat lehet egy szám, a hét egy napja, egy dátum, egy szín, egy illat, egy betű, egy név, valakinek a neme, a családi állapota, a lakcíme, a munkahelyi beosztása, a fényképe, a hangja stb. A felhasználó céljaitól függ, hogy valami az lesz, vagy sem. Általánosan megfogalmazva az adat (absztrakt adat) valamely előre rögzített halmaznak az eleme. (Például ha egy személy egész centiméterben megadott testmagasságáról van szó, akkor az azt leíró adat lehet olyan egész szám, amely mondjuk a {0, 1, 2, …, 299, 300} számhalmazból kerül ki. Ha nagyvonalúak akarunk lenni, akkor vehetjük helyette mondjuk a nemnegatív egész - más szóval a *természetes* - számok halmazát. Ez utóbbit ***N***-nel jelöljük. ***N***={0, 1, 2, 3, …}). Az adat attól absztrakt, hogy csak elméletileg határoztuk meg. Még nem jelent meg, nem jelenítettük meg anyagi mivoltában. Például a természetes számok közül vett adat esetén nincsenek számjegyei az adott számnak, mivel nem mondtuk meg, hogy hogyan fogjuk megjeleníteni, leírni, reprezentálni. Ezen a szinten magának a számjegy fogalmának nincs is értelme. Például a kettőezer-tizenegyet írhatjuk 10-es helyiértékes számrendszerbeli formában is, úgy, mint 2011. Kettes alapú helyiértékes számrendszerben ez 11111011011 alakú lesz, tizenhatos alapú helyiértékes számrendszerben pedig 7DB. A római számírás szerint például MMXI alakban jelenik meg, a magyar rovásírás szerint pedig  formában. (Megjegyezzük, hogy szavakkal leírva a „kettőezer-tizenegy” is már egy megjelenési forma.) Általánosságban nem maga a konkrét adat a fontos számunkra, hanem a jellege, a típusa, azaz, hogy milyen jellegű elemek, adatok közül jön, és mit lehet vele csinálni, milyen műveleteket lehet rajta végrehajtani.

Definíció: Az absztrakt adattípus

Az absztrakt adattípus egy leírás, amely absztrakt adatok halmazát és a rajtuk végezhető műveleteket adja meg (definiálja) nem törődve azok konkrét (gépi) realizálásával.

Tehát egy halmazról van szó a rajta értelmezett műveletekkel együtt. Megadása történhet a T=(A,M) párossal, ahol T jelöli az absztakt adattípust, A az adatok halmazát, M pedig a műveletek halmazát. Például megadhatjuk a *nemnegatív egész szám*, vagy *előjel nélküli egész szám* (*természetes szám*) T absztrakt adattípust úgy, hogy ez a típus T=(***N***, M), ahol ***N***={0, 1, 2, 3, …} az adatok halmaza, és M={+, *} az összeadás és a szorzás, az ***N*** elemein végezhető műveletek halmaza. Azt, hogy a műveleteket konkrétan hogyan kell végrehajtani, a leképezést, amely megadja a művelet eredményét, a művelet leírása tartalmazza. Az absztrakt adattípus egy „fekete doboz”, amelybe beletesszük az adatot tárolásra és kivesszük, ha szükségünk van rá. Nem érdekes, hogy a doboz hogyan végzi a tárolást. Ami viszont fontos az az, hogy az absztrakt adattípushoz elválaszthatatlanul hozzátartoznak azok a műveletek, amelyeket az adatokkal végezni lehet. A művelet fogalmát ezek után tisztáznunk kell.

Definíció: Halmazok Descartes szorzata

Két halmaz Descartes szorzatán egy olyan elempárokból álló halmazt értünk, amely tartalmazza az összes olyan elempárt, amelyben a pár első tagja az első, a második tagja a második halmazból való. Tömören az A és B halmazok *Descartes szorzat*án azt a C=A×B halmazt értjük, amelyre C={(a,b), a∈A, b∈B}.

Például, ha X={a, b, c} és B={0, 1}, akkor X×B={(a, 0), (a, 1), (b, 0), (b, 1), (c, 0), (c, 1)} és B×X={(0, a), (0, b), (0, c), (1, a), (1, b), (1, c)}. (A sorrend nem felcserélhető, a Descartes

szorzat nem kommutatív!) Három halmaz Descartes szorzata hasonló módon elemhármasok halmaza lesz, négy halmaz esetén elemnégyeseké, stb.

Definíció: Halmaz hatványa

Egy A halmaz nulladik hatványa az üres halmaz, $A^0=\emptyset$, első hatványa maga az A halmaz, $A^1=A$. Az A halmaz második hatványa $A^2=A\times A$, és általában, ha $n\in\mathbf{N}$, és $n>2$, akkor $A^n= A^{n-1}\times A$.

Például ha B={0, 1}, akkor B^3={(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)}. Ha X={a, b, c}, akkor X^2={(a, a), (a, b), (a, c), (b, a), (b, b), (b, c), (c, a), (c, b), (c, c)}.

Definíció: *n*-áris művelet halmazon

Egy A halmazon értelmezett *n*-áris (*n*-ér) műveletnek nevezzük a következő leképezést (függvényt): $f : A^n \to A$. Azaz az f leképezés minden az A elemeiből képezett *n*-esnek megfeleltet egy elemet ugyanazon A halmazból.

Ha speciálisan n=1, akkor *unáris* (unér) műveletről beszélünk (egyváltozós művelet). Ha n=2, akkor *bináris* (binér) műveletről beszélünk (kétváltozós művelet). Unáris műveletnek tekinthető például valós számok esetén a szám ellentettjének (-1-szeresének) meghatározása (előjelváltás), de a négyzetreemelés is az, vagy a szám szinuszának a meghatározása is. Bináris művelet például két szám összeadása: $f : A^2 \to A$, ahol $z = f(x, y) = x + y$, $x, y, z \in A$. Leggyakrabban az unáris és a bináris műveletek fordulnak elő a gyakorlatban.

Amennyiben az absztrakt adattípushoz tartozó absztrakt adatot megjelenítjük, akkor már kézzelfoghatóvá válik és anyagi értelemben vett műveleteket végezhetünk rajta. Ekkor adatstruktúráról beszélünk.

Definíció: Adatstruktúra

Az absztrakt adattípus konkrét megjelenési formáját, realizációját *adatstruktúrának* nevezzük.

Az adatstruktúrához hozzátartoznak természetesen mindazok a műveletek is, amelyekkel az absztrakt adattípus rendelkezett. Ugyanannak az absztrakt adattípusnak számtalan adatstruktúra felelhet meg. Némelyek támogatják a műveletek elvégzését, mások esetében pedig brutálisan nehéz is lehet azok elvégzése. Például könnyen össze tudunk szorozni egész számokat a tizes helyiértékes számrendszerben. Ebben az esetben a művelet elvégzésére van egy egyszerű, kisiskolásoknak is tanítható módszer, az adatstruktúra támogatja, segíti a műveletvégzést. Ugyanakkor hogyan tennénk ezt meg, ha római számokkal jelenítenénk meg az egész számokat? Ez a struktúra nem támogatja a műveletvégzést. Nem lehetetlen elvégezni benne a műveleteket, csak nagyon bonyolulttá válnak a szabályok.

Definíció: Az implementáció

Az absztrakt adattípus digitális számítógépen történő realizációját *implementációnak* nevezzük.

Tehát az implementáció az adatstruktúra egy speciális esete. Számunkra nagyon fontos, mert ebben tudunk nagyon hatékonyan dolgozni, kihasználva a gép nagy tároló-kapacitását és a műveletvégzés nagy sebességét. Az implementáció azonban már általában nem veszteség- és torzulásmentes a kiinduló eredeti absztrakt adattípushoz, vagy egy neki megfelelő adatstruktúrához képest. Ha másért nem, akkor azért, mert bár a gép tároló-kapacitása nagy,

mégiscsak véges. Ebből számtalan, látszatra apró probléma, sajátosság adódik, amelyre viszont figyelemmel kell lennünk, mert sok kínos programhiba forrása lehet.

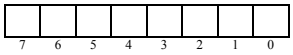
A digitális technikában alapvető fogalom a bit fogalma.

Definíció: Az (adat)bit
Adat*bit*nek nevezzük az implementáció során a memóriában tárolt adatmennyiség legkisebb egységét. Ez tartalmilag egy 0 (zérus), vagy 1 (egy) jelet jelent. Lényegtelen, hogy ezt technikailag hogyan valósítják meg. Tárolni mindig csak ennek a pozitív egész számú többszörösét lehet.

Nincs tehát adatmennyiség szempontjából félbit, vagy másfél bit. Egyetlen bit nagyon kicsiny adatmennyiség.

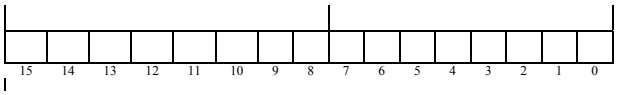
Definíció: A byte (bájt)
Adatbyte-nak nevezünk egy adott sorrendben elhelyezkedő, egymást követő, egy egységbe összefogott bitnyolcast. A byte a memóriában az egyidejűleg elérhető adatmennyiség legkisebb egysége.

A számítógépes memória byte-okból épül föl, egydejűleg legkevesebb egy byte írható oda be, vagy olvasható onnan ki. Ha egyetlen bitre vagyunk kíváncsiak, akkor is legkevesebb az őt tartalmazó byte-ot kell kezelnünk. A byte bitjeit az alábbi módon sorszámozzuk, indexeljük.

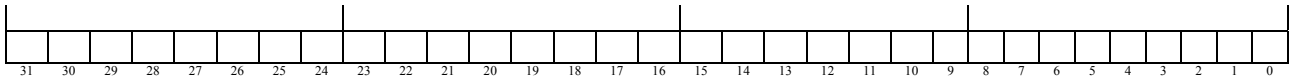


byte és bitjei

A számítógépek központi vezérlő, feldolgozó egységei, a processzorok képesek több összekapcsolt byte-ot is kezelni. Ilyen módon beszélhetünk két egymás mellett álló összekapcsolt byte esetén *szó*ról, négy byte esetén pedig *dupla szó*ról.



szó byte-jai és bitjei



dupla szó byte-jai és bitjei

Byte „üresen”, vagy részben üresen nem állhat. Minden bitje a számunkra fontos pillanatban vagy a 0 vagy az 1 jelet tartalmazza.

Az adat implementációját ez a rendelkezésre álló keret, a byte-ok, szavak, dupla szavak rendszere, valamint a processzor által elvégezhető gépi elemi utasítások szabják meg, határolják be. Ez a keret sugalmazza a kettes számrendszer használatát is például.

A természetes szám absztrakt adattípus

Formálisan, mint fentebb említettük, ez az absztrakt adattípus a T=(***N***, M) párossal írható le, ahol ***N***={0, 1, 2, 3, …} az adatok, a természetes számok halmaza, és M={+, *} az összeadás és a szorzás, az ***N*** elemein végezhető műveletek halmaza. (A számok között megszokott

kivonás és osztás művelete a természetes számok között nem mindig végezhető el.) Megmutatjuk, hogy van módszer a szorzás elvégzésére úgy, hogy a reprezentálással nem törődünk. Ennek a módszernek a neve ma gyakran úgy olvasható, hogy „*orosz paraszt*” módszer. Az elnevezés nem teljesen jogos, mert a módszert már az ókori egyiptomiak is ismerték és használták. A módszer lényege, hogy a szorzatot fokozatosan gyűjtjük össze a zérusból indulva ki. A szorzót vizsgáljuk. A vizsgálat abból áll, hogy megnézzük, hogy a szorzó páratlan-e. Ha igen, akkor a szorzandót hozzáadjuk a szorzathoz, - ami kezdetben zérus volt, - ha a szorzó nem páratlan, akkor nem adjuk hozzá. Ezután a szorzót lecseréljük a felének az egész részére (lefelé kerekítjük egészre), a szorzandót pedig lecseréljük a duplájára. A vizsgálat mindaddig ismétlődik, míg a szorzó zérussá nem válik. (Lássuk be, hogy a módszer mindig a helyes szorzathoz vezet két nemnegatív egész szám esetén!)

Példa: Szorozzuk össze a 79-et és a 47-et az „orosz paraszt” módszerrel!

Szorzandó	szorzó	Szorzó páratlan?	Szorzat
79	47	igen	0+79=79
158	23	igen	79+158=237
316	11	igen	237+316=553
632	5	igen	553+632=1185
1264	2	nem	=1185
2528	1	igen	1185+2528=3713
	0		=3713

Néhány érdekes függvény és művelet

Az anyagban fogunk függvényeket használni. A szokványos függvények mellett meg kell barátkozni az egészrész függvényekkel, a törtrész függvényel és a kerekítő függvénnyel. Ne tévesszen meg senkit, hogy ezek a függvények hasonlítani fognak a programozási nyelvekben előforduló hasonló nevű függvényekhez, mert nem biztos, hogy teljesen azonosak azokkal. Az egyes programozási nyelvek nem mindig konzekvensek. Az ugyanolyan nevű függvény a különböző programozási nyelvekben némileg eltérő módon viselkedhet. Érdemes tanulmányozni a nyelvi leírást, specifikációt. A továbbiakban ***Z***-vel fogjuk jelölni az egész számok halmazát. ***Z***={ …, -3, -2, -1, 0, 1, 2, 3, …}.

Definíció: Az alsó egészrész függvény

Az *alsó egészrész* függvény minden valós számhoz egy egész számot rendel hozzá, éppen azt, amely a tőle nem nagyobb egészek közül a legnagyobb. Az alsó egészrész függvény jele: $\lfloor x \rfloor$, ahol x valós szám. Tömören:

$$\lfloor x \rfloor = \max_{\substack{k \le x \\ k \in Z}} k \,. \tag{1}$$

Más szavakkal formálisan: $\lfloor x \rfloor = k$, ahol k olyan egész szám, hogy $k \le x < k+1$.

Példa:

x	$-5{,}2$	-5	5	5,2
$\lfloor x \rfloor$	-6	-5	5	5

Definíció: A felső egészrész függvény

A *felső egészrész* függvény minden valós számhoz egy egész számot rendel hozzá, éppen azt, amely a tőle nem kisebb egészek közül a legkisebb. A felső egészrész függvény jele: $\lceil x \rceil$, ahol x valós szám. Tömören:

$$\lceil x \rceil = \min_{\substack{k \geq x \\ k \in Z}} k \,. \tag{2}$$

Más szavakkal formálisan: $\lceil x \rceil = k$, ahol k olyan egész szám, hogy $k\text{-}1 < x \leq k$.

Példa:

x	$-5{,}2$	-5	5	$5{,}2$
$\lceil x \rceil$	-5	-5	5	6

Az alsó és felső egészrész függvények fontos tulajdonságait az alábbi táblázatban foglaljuk össze: (Lássuk be, hogy ezek valóban teljesülnek!)

1. Ha a egész szám, akkor $\lfloor a \rfloor = a$, $\lceil a \rceil = a$.
2. Ha x valós szám, akkor $\lfloor x \rfloor \leq \lceil x \rceil$.
3. Ha $x \leq y$ valós számok, akkor $\lfloor x \rfloor \leq \lfloor y \rfloor$, $\lceil x \rceil \leq \lceil y \rceil$.
4. Ha x valós, a egész szám, akkor $\lfloor x \pm a \rfloor = \lfloor x \rfloor \pm a$, $\lceil x \pm a \rceil = \lceil x \rceil \pm a$.
5. Ha x valós szám, akkor $-\lfloor x \rfloor = \lceil -x \rceil$, $-\lceil x \rceil = \lfloor -x \rfloor$.
6. Ha x és y valós számok, akkor $\lfloor x + y \rfloor \geq \lfloor x \rfloor + \lfloor y \rfloor$, $\lceil x + y \rceil \leq \lceil x \rceil + \lceil y \rceil$.
7. Ha x és y valós számok, akkor $\lfloor x - y \rfloor \leq \lfloor x \rfloor - \lfloor y \rfloor$, $\lceil x - y \rceil \geq \lceil x \rceil - \lceil y \rceil$.

Definíció: A kerekítő függvény

A *kerekítő függvény* minden valós számhoz a hozzá legközelebb eső egész számot rendeli hozzá. Ha a legközelebbi egész szám nem egyértelmű, akkor a nagyobbat választja. A kerekítő függvény jele: *Round(x)*, ahol x valós szám.

$$Round(x) = \left\lfloor x + \frac{1}{2} \right\rfloor. \tag{5}$$

A legközelebbi egészre kerekít. Pozitív számok esetén, ha a tizedesrész 5/10, vagy annál nagyobb, akkor felfelé, kisebb esetben lefelé kerekít. Negatív számok esetén ha a tízes számrendszer szerinti felírásban a tizedesrész kisebb, mint 5/10, vagy egyenlő vele, akkor felfelé, egyébként lefelé kerekít.

Példa:

x	-6	$-5{,}8$	$-5{,}5$	$-5{,}2$	-5	5	$5{,}2$	$5{,}5$	$5{,}8$	6
$Round(x)$	-6	-6	-5	-5	-5	5	5	6	6	6

Definíció: A törtrész függvény

A *törtrész függvény* minden valós számhoz azt a számot rendeli hozzá, amely azt mutatja meg, hogy a szám mennyivel nagyobb az alsó egészrészénél. A törtrész függvény jele: $\{x\}$, ahol x valós szám. Tömören:

$$\{x\} = x - \lfloor x \rfloor. \tag{4}$$

Mindig fennáll a $0 \le \{x\} < 1$ egyenlőtlenség.

Példa:

x	$-5{,}8$	$-5{,}2$	-5	5	5,2	5,8
$\{x\}$	0,2	0,8	0	0	0,2	0,8

Felhívjuk a figyelmet két műveletre:

Definíció: Az egész hányados képzése, a div művelet
Legyen *a* és *b* egész szám ($a,b \in \mathbf{Z}$), $b \neq 0$. Definíció szerint az egész osztás műveletén az *a/b* osztás eredményének alsó egész részét értjük. Tömören:

$$a \; div \; b = \lfloor a / b \rfloor. \tag{5}$$

Példa: $-9 \; div \; 4 = -3$, $\quad 9 \; div \; 4 = 2$

Definíció: Az egész maradék képzése, a mod művelet
Legyen *a* és *b* egész szám. Definíció szerint

$$a \bmod b \overset{def}{=} \begin{cases} a, & ha \;\; b = 0 \\ a - \lfloor a / b \rfloor \cdot b = a - (a \, div \, b) \cdot b, & ha \;\; b \neq 0. \end{cases} \tag{6}$$

Példa: $-9 \bmod 4 = 3$, $9 \bmod 4 = 1$, $-9 \bmod (-4) = -1$, $9 \bmod (-4) = -3$

Speciális jelentése van az $a \bmod 1$ jelölésnek. Ezt minden valós *a*-ra értelmezzük és jelentése az a valós szám törtrésze, azaz

$$a \bmod 1 \overset{def}{=} \{a\}. \tag{7}$$

Az előjel nélküli egész szám adatstruktúra

A természetes szám absztrakt adattípus egyik realizációja az előjel nélküli egész szám adatstruktúra, melynek a lejegyzéséhez a helyiértékes számrendszert használjuk. Ennek elvégzéséhez el kell döntenünk, hogy a használt helyiértékes számrendszernek mi lesz az alapszáma. Alapszámnak bármilyen egynél nagyobb *b* természetes szám választható. Szükségünk lesz számok különböző alapú számrendszerben történő felírására. Egy szám lejegyzésekor a használt számrendszer alapszámát mindig tizes számrendszerben adjuk meg és a szám jobb alsó sarkához írjuk indexként. Ha a számrendszer alapja a $b \ge 2$ egész szám, akkor az *x* pozitív egész szám számjegyei:

$$c_n, c_{n-1}, \ldots, c_1, c_0, \tag{8}$$

ahol $0 \le c_k < b$, $k = 0, 1, \ldots, n$ és az x szám értéke ezekkel a számjegyekkel és az alappal kifejezve:

$$x = c_n \cdot b^n + c_{n-1} \cdot b^{n-1} + \ldots + c_1 \cdot b^1 + c_0 \cdot b^0. \tag{9}$$

Az n értékét úgy határozzuk meg, hogy $c_n \neq 0$ legyen, és minden $c_k = 0$, ha $k > n$. Ha a számrendszer alapszáma tíznél nagyobb, akkor a 0, 1, 2, ..., 9 számjegyek mellett új számjegyeket kell bevezetni a tíz, tizenegy, ..., $b-1$ számértékekre. Kényelmi és nyomdatechnikai okok miatt a latin ábécé nagybetűit használjuk erre a célra. Ilyen módon tehát az A=10, B=11, C=12, ..., Z=35 jelek használatosak. (Lehet találkozni vegyes jelöléssel is, ahol a számjegyeket tizes számrendszerben jegyzik le. Mi nem fogjuk ezt alkalmazni.)

Számoljunk el különböző alapú számrendszerekben 1-től 20-ig! A helyiértékes rendszerben megjelenített eredmények az alábbi táblázatban láthatók. Az informatikában a kettes és a tizenhatos alapú számrendszerek a kitüntetettek a tizes alap mellett, esetleg előfordulhat a nyolcas alap is.

Számrendszer alapszáma → / Számérték ↓	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
egy	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
kettő	10	2	2	2	2	2	2	2	2	2	2	2	2	2	2
három	11	10	3	3	3	3	3	3	3	3	3	3	3	3	3
négy	100	11	10	4	4	4	4	4	4	4	4	4	4	4	4
öt	101	12	11	10	5	5	5	5	5	5	5	5	5	5	5
hat	110	20	12	11	10	6	6	6	6	6	6	6	6	6	6
hét	111	21	13	12	11	10	7	7	7	7	7	7	7	7	7
nyolc	1000	22	20	13	12	11	10	8	8	8	8	8	8	8	8
kilenc	1001	100	21	14	13	12	11	10	9	9	9	9	9	9	9
tíz	1010	101	22	20	14	13	12	11	10	A	A	A	A	A	A
tizenegy	1011	102	23	21	15	14	13	12	11	10	B	B	B	B	B
tizenkettő	1100	110	30	22	20	15	14	13	12	11	10	C	C	C	C
tizenhárom	1101	111	31	23	21	16	15	14	13	12	11	10	D	D	D
tizennégy	1110	112	32	24	22	20	16	15	14	13	12	11	10	E	E
tizenöt	1111	120	33	30	23	21	17	16	15	14	13	12	11	10	F
tizenhat	10000	121	100	31	24	22	20	17	16	15	14	13	12	11	10
tizenhét	10001	122	101	32	25	23	21	18	17	16	15	14	13	12	11
tizennyolc	10010	200	102	33	30	24	22	20	18	17	16	15	14	13	12
tizenkilenc	10011	201	103	34	31	25	23	21	19	18	17	16	15	14	13
húsz	10100	202	110	40	32	26	24	22	20	19	18	17	16	15	14

Példa: 2006 számjegyei tizes számrendszerben $c_3 = 2$, $c_2 = 0$, $c_1 = 0$, $c_0 = 6$. Itt $n=3$ és $2006 = 2 \cdot 10^3 + 0 \cdot 10^2 + 0 \cdot 10^1 + 6 \cdot 10^0$.

Nyílvánvalóan: $c_0 = x \bmod b$ és $c_n c_{n-1} \ldots c_2 c_1 = x \; div \; b$. A következő séma alkalmas a számjegyek egymást követő fordított irányú előhozására:

x	b
$x_1 = x \ div \ b$	$c_0 = x \bmod b$
$x_2 = x_1 \ div \ b$	$c_1 = x_1 \bmod b$
...	...
$x_k = x_{k-1} \ div \ b$	$c_{k-1} = x_{k-1} \bmod b$
...	...
$x_n = x_{n-1} \ div \ b$	$c_{n-1} = x_{n-1} \bmod b$
0	$c_n = x_n \bmod b$

(10)

Példa: Írjuk fel a 2006-ot kettes (= *bináris*) számrendszerben és 16-os (= *hexadecimális*) számrendszerben.

2006	2
1003	0
501	1
250	0
125	0
62	1
31	0
15	1
7	1
3	1
1	1
0	1

2006	16
125	6
7	$13=D_{16}$
0	7

Tehát $2006_{10}=11111010010_2=7D6_{16}$

Átírás tizes alapra a (9) formula átrendezésével történik az úgynevezett *Horner séma* szerint.

$$x = \left(\ldots\left(\left(c_n\right)\cdot b + c_{n-1}\right)\cdot b + \ldots + c_1\right)\cdot b + c_0 \,. \tag{11}$$

Ezzel azt érjük el, hogy kevés műveletet kell használni, a műveletek azonos jellegűek (szorzás *b*-vel és a következő jegy hozzáadása), másrészt a műveleteket végezhetjük tizes számrendszerben.

Példa: $7D6_{16}=((7)\cdot16+13)\cdot16+6=2006$
$11111010010_2=((((((((((1)\cdot2+1)\cdot2+1)\cdot2+1)\cdot2+0)\cdot2+1)\cdot2+0)\cdot2+0)\cdot2+1)\cdot2+0$
$2006_{10}=(((2)\cdot10+0)\cdot10+0)\cdot10+6$

Kellemes az átváltás a két számrendszerbeli ábrázolás között, ha történetesen a bináris és a hexadecimális számrendszerről van szó. Ekkor hexadecimális alakról binárisra történő átírás esetén minden hexadecimális jegyet a jegy bináris megfelelőjével helyettesítünk. Binárisról hexadecimálisra történő átírásnál pedig jobbról balra haladva négyes csoportokra osztva a bináris szám számjegyeit minden csoportot helyettesítünk a hexadecimális megfelelőjével. A megfeleltetés a hexadecimális számjegyek és a bináris négyjegyű csoportok között az alábbi táblázatban látható:

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

(Lássuk be, hogy a javasolt módszer helyes eredményre vezet!) Módszerünkkel kikerüljük egyrészt a tizes számrendszerre történő átmeneti átalakítást, másrészt nem kell nem tizes alapú számrendszerben műveleteket végezni

Pozitív egész szám *b* alapú logaritmusa és a szám *b* alapú számrendszerbeli számjegyei számának a kapcsolatát világítja meg az alábbi tétel.

1. Tétel: A számjegyek számáról

Pozitív *x* egész szám számjegyeinek a száma *b* alapú számrendszerben eggyel több, mint a szám *b* alapú logaritmusának az alsó egészrésze, azaz ha a szám számjegyei $c_n, c_{n-1}, \ldots, c_1, c_0$, akkor a jegyek száma

$$n+1 = \lfloor \log_b x \rfloor + 1 . \tag{12}$$

Bizonyítás

$$\begin{aligned} x &= c_n \cdot b^n + c_{n-1} \cdot b^{n-1} + \ldots + c_1 \cdot b^1 + c_0 \cdot b^0 = \\ &= b^n \cdot \underbrace{\left(c_n + c_{n-1}/b + \ldots + c_1/b^{n-1} + c_0/b^n \right)}_{=y} = b^n \cdot y. \end{aligned} \tag{13}$$

Világos, hogy $1 \le c_n \le y < b$. Innen az *y* logaritmusára kapjuk, hogy

$$0 \le \log_b y < 1 . \tag{14}$$

(13)-ból logaritmálással

$$\log_b x = n \cdot \log_b b + \log_b y = n + \log_b y \tag{15}$$

adódik. Azaz

$$n + \log_b y = \log_b x . \tag{16}$$

(16) mindkét oldalán az alsó egészrészt véve $n = \lfloor \log_b x \rfloor$ adódik, mivel *n* egész szám és (14) fennáll. Egyet hozzáadva mindkét oldalhoz kapjuk az állításunkat. ■

(Az ■ jel a bizonyítás végét jelzi.)

Ebben az adatstruktúrában a struktúra tárgyalt műveletei egyszerű módon, kisiskolásoknak is tanítható szinten elvégezhetők. Erre nem térünk ki, hiszen elemi iskolai anyag. Megadjuk meg a kettes és a tizenhatos számrendszerbeli összeadó- és szorzótáblát.

Bináris összeadótábla

+	0	1
0	0	1
1	1	10

Bináris szorzótábla

×	0	1
0	0	0
1	0	1

Hexadecimális összeadótábla																
+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Hexadecimális szorzótábla																
×	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

A helyiértékes számrendszerek egymással azonos módon viselkednek. Ez áll a műveletek végzésének a módjára is. A szabályokban csak arra kell ügyelni, hogy mikor lépjük át az alap valamely hatványát, és természetesen a saját összeadó- és szorzótáblát kell használni.

Példa összeadásra:

b=10	b=2	b=16
3 4 9	1 0 1 0 1 1 1 0 1	1 5 D
+ 2 9 7	+ 1 0 0 1 0 1 0 0 1	+ 1 2 9
6 4 6	1 0 1 0 0 0 0 1 1 0	2 8 6

Példa szorzásra:

b=10	b=16
3 4 9 × 2 9 7	1 5 D × 1 2 9
6 9 8	1 5 D
3 1 4 1	2 B A
2 4 4 3	C 4 5
1 0 3 6 5 3	1 9 4 E 5

b=2
1 0 1 0 1 1 1 0 1 × 1 0 0 1 0 1 0 0 1
1 0 1 0 1 1 1 0 1 0 0
1 0 1 1 1 0 1 1 1 0 1 0
1 0 1 0 1 1 1 0 1 0 0
1 0 1 0 1 1 1 0 1
1 1 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1

Az előjel nélküli egész szám implementációja

Az informatikában alapvető dolog, hogy a számok hogyan kerülnek tárolásra a számítógép memóriájában. A memóriát úgy lehet elképzelni, mint egymás mellett lineárisan felsorakoztatott tárolórekeszek sorozata. A rekeszeket egymástól a sorban elfoglalt helyük különbözteti meg, amit egy indexszel (címmel) írunk le. A rekesz fogalom szemléletes, de fizikailag nem pontos. A fizikai rekesz ma a byte (= 8 bit). A byte a memória legkisebb fizikailag címezhető egysége. A memóriából kiolvasni, vagy oda beírni egy byte-nál kevesebb adatmennyiséget nem lehet. Ha csak egy bitet akarunk megváltoztatni, akkor is ki kell olvasni az őt tartalmazó byte-ot, a kívánt bitet átírni, majd a byte-ot visszaírni a memóriába. A byte tartalmát a jobb áttekinthetőség miatt hexadecimális számrendszerben szoktuk megadni. Például az 11001001_2 tartalmú byte hexadecimális alakban $C9_{16}$. A byte bitjeit jobbról balra indexeljük. A jobbszélső bit a nullás indexű bit (a legkevésbé szignifikáns bit, Least Significant Bit, LSB), tőle balra áll az egyes indexű bit, és így tovább. A byte balszélén áll a hetes bit (a legszignifikánsabb bit, a legnagyobb helyiértékű bit, Most Significant Bit, MSB).

	MSB							LSB
	bit_7	bit_6	bit_5	bit_4	bit_3	bit_2	bit_1	bit_0
bitindex	7	6	5	4	3	2	1	0

Byte és bitjei

Már egyetlen byte is alkalmas szám tárolására csak a számtartomány kicsi. A nyolc bit mindegyike lehet zérus, vagy egy. Ennek megfelelően $2^8 = 256$ egymástól különböző byte létezhet. Minden ilyen bitvariációhoz, bitmintázathoz hozzárendelünk egy számot. Természetes módon kínálkozott, hogy a számot kettes számrendszerben leírva tároljuk. Ez történhet egy, kettő, vagy négy byte-on. Ha a szám nem tölti ki a rendelkezésre álló területet, akkor a tároló rekeszben jobbra igazítva az elejét zérusokkal töltjük fel, így a számérték nem változik.

rekesz	legkisebb számérték	legnagyobb számérték
byte	0	2^8-1=255
szó (két byte)	0	2^{16}-1=65 535
dupla szó (négy byte)	0	2^{32}-1=4 294 967 295

A legkisebb szám tiszta zérus bitekből áll, a legnagyobb tiszta egyes bitekből. Az implementáció sajátossága, hogy van benne legnagyobb számérték, szemben az absztrakt adattípussal és az adatstruktúrával, ahol ez a korlátozás nem volt. Ennek megfelelően a műveletek egyes esetekben hibás eredményre vezetnek. Például a legnagyobb számértékhez egyet hozzáadva az eredmény zérus lesz. Ugyanis egy további bitre lenne szükség a tároláshoz, de az a rekeszben már nem fér el, az eredmény túlcsordult. Ez a rekeszből kieső bit a túlcsordulás jele, amit a processzorok számon is tartanak (átvitel bit, angolul carry bit) és a túlcsordulás jelzésére szolgál, de a memóriába nem tárolható. Ezektől eltekintve a műveletek pontos eredményt szolgáltatnak.

Példa helyes és hibás (túlcsordulásos) összeadásra 16 biten.

Helyes (carry=0)

1. összeadandó	1	1	1	0	0	1	1	1	1	0	1	0	1	0	1	0
2. összeadandó	0	0	0	1	0	1	1	1	1	0	0	1	1	0	1	1
összeg	1	1	1	1	1	1	1	1	0	1	0	0	0	1	0	1
átvitel	0	0	0	0	0	1	1	1	1	0	1	1	1	0	1	0

Hibás (carry=1)

1. összeadandó	1	1	1	0	0	1	1	1	1	0	1	0	1	0	1	0
2. összeadandó	0	0	1	1	0	1	1	1	1	0	0	1	1	0	1	1
összeg	0	0	0	1	1	1	1	1	0	1	0	0	0	1	0	1
átvitel	1	1	1	0	0	1	1	1	1	0	1	1	1	0	1	0

Az egész szám absztrakt adattípus

Ez az adattípus annyiban különbözik a természetes szám adattípustól, hogy a számok lehetnek negatívak is. Ennek megfelelően a kivonás művelete is elvégezhető lesz. Formálisan a T=(***Z***, M) párossal írható le, ahol ***Z***={…, -3, -2, -1, 0, 1, 2, 3, …} az adatok, az egész számok halmaza, és M={+, -, *} az összeadás, a kivonás és a szorzás, a ***Z*** elemein végezhető műveletek halmaza.

Az előjeles egész szám adatstruktúra

Csak annyiban különbözik az előjel nélküli egész szám adatstruktúrától, hogy a negatív számok leírását egy mínusz (-) jellel kezdjük. A műveletvégzésre nem térünk ki, elemi iskolás anyag, a természetes számokhoz képest az előjelkezelési szabályok jelennek meg többletként, szükség esetén zárójelezünk, hogy két műveleti jel ne kerüljön egymás mellé.

Az előjeles egész szám implementációja

Ha előjeles egész számokat szeretnénk tárolni, akkor az úgynevezett *kettes komplemens* ábrázolást szokás előnyei miatt alkalmazni. Ekkor az előjeles egész szám hozzárendelése úgy történik, hogy ha a MSB=0, akkor a rekesztartalmat az előjelmentes esetnek megfelelően értelmezzük. Ha MSB=1, akkor annyival kell többet tennünk ezután még, hogy a kapott

előjelmentes számból kivonjuk a 2^n számot, ahol n=8, 16, vagy 32 aszerint, hogy a rekesz byte, szó, vagy duplaszó, és amely kivonás eredménye biztosan negatív lesz. Ezen a módon az ábrázolható számok tartománya az alábbi.

rekesz	legkisebb számérték	legnagyobb számérték
byte	-2^7=-128	2^7-1=127
szó (két byte)	-2^{15}=-32 768	2^{15}-1=32 767
dupla szó (négy byte)	-2^{31}=-2 147 483 648	2^{32}-1=2 147 483 647

Kettes komplemens képzési szabálya.
Egy szám kettes komplemensének (-1-szeresének) a felírása azon egyszerű szabály szerint történhet, hogy a rekesz jobbszéléről balfelé elindulva sorra leírjuk a biteket mindaddig, amíg zérusok. Ha ezzel elfogytak a bitek, akkor be is fejeztük az eljárást. Ha még lettek volna meg nem vizsgált bitek, akkor a következő 1 darab 1-es bitet is leírjuk. Ezután, ha még mindig lennének bitjeink, akkor azokat ellentétesen írjuk le a továbbiakban, 0 helyett 1-et, 1 helyett zérust.

Példa 16 bitre:

szám	0	0	0	1	1	1	0	1	0	1	0	1	1	0	0	0
kettes komplemense	1	1	1	0	0	0	1	0	1	0	1	0	1	0	0	0

Vegyük észre, hogy a két számot 16 biten összeadva az eredmény zérus, tehát az egyik szám a másik -1-szerese.

Másik szabályt is megfogalmazhatunk a kettes komplemens képzésére. Leírjuk a szám bitjeit úgy, hogy mindegyik helyett az ellentétes bitet írjuk le, majd a kapott számhoz egyet hozzáadunk. (Ellenőrizzük, hogy valóban ez is helyes eredményt ad!)

Az implementációnak a sajátosságai közé tartozik, hogy két olyan szám van, amelyiknek a kettes komplemense (-1-szerese) saját maga! Az egyik természetszerűleg a zérus, amit el is várunk, a másik azonban az az eset, amikor a rekesz balszélén egyetlen egyes van és a többi bit zérus. (Például byte rekesznél a -128 kettes komplemense saját maga, azaz a -1-szeres nem 128-at ad, mivel az nem is ábrázolható.) Ebben az implementációban is előfordulhat túlcsordulás, például amikor két negatív szám összegeként pozitívat kapunk. (Adjuk össze byte-on például a -100-at és a -50-et!) Hogyan tudnánk jelezni a túlcsordulást? Csak a carry bit figyelése ebben az esetben már nem elegendő!

A valós szám absztrakt adattípus

Ez az adattípus lényegesen különbözik az előzőektől. A valós szám fogalma, definíciója már bonyolultabb. Ebbe beletartoznak az egész és törtszámok, valamint az irracionális számok is. Nem részletezzük a valós szám mibenlétét, tapasztalatok révén van elképzelésünk róla a középiskolából. A valós számok között az osztás is elvégezhető művelet, amennyiben az osztó nem zérus. Formálisan leírva T=(***R,*** M) párossal írható le, ahol ***R***={a valós számok} az adatok, a valós számok halmaza, és M={+, -, *, /} az összeadás, a kivonás, a szorzás és az osztás, az ***R*** elemein végezhető műveletek halmaza.

A valós szám adatstruktúra

A valós számokat reprezentálhatjuk a törtvesszős felírással, amikor is a szám egész része után vesszőt teszünk és azután írjuk a törtrész jegyeit. Tizes számrendszer esetén tizedes törteknek

nevezzük őket. A műveleteket a tizedes törtekkel történő számolás szabályai szerint végezzük. Kettes számrendszer esetén talán kettedes törteknek kellene nevezni őket. A számolási szabályok is a tizes számrendszerhez hasonlóan történnek. Hogyan írjuk át egyik számrendszerből egy másikba az így megadott számokat? A szám egész részét (a tizedes vessző előtti részét), mint egész számot továbbra is átírhatjuk, ahogy azt korábban az előjeles egészeknél láttuk. Ezután vesszőt írva elkezdhetjük a törtjegyek írását. A törtjegyeket a szorzásos módszerrel határozzuk meg a kereszt sémából. Tekintsük egy szám törtrészének a tizes számrendszerbeli felírását. A szám legyen $0,c_1c_2c_3\ldots$ alakú, ahol $c_1,c_2,c_3\ldots$ a törtrész egymást követő tizedesjegyei. Ha 10-zel szorzunk, akkor az első tizedesjegy kicsúszik az egészek helyére, amit levághatunk. További szorzásokkal a többi jegy is előjön egymás után. Ha minket egy b alapú számrendszerbeli felírás jegyei érdekelnek, akkor világos, hogy a 10 helyett b-vel kell szorozgatni. A tevékenység egy sémába foglalható, a számjegyek egyenes sorrendben keletkeznek:

b	x
$c_1 = \lfloor x \cdot b \rfloor$	$x_1 = (x \cdot b) \bmod 1$
$c_2 = \lfloor x_1 \cdot b \rfloor$	$x_2 = (x_1 \cdot b) \bmod 1$
...	...
$c_k = \lfloor x_{k-1} \cdot b \rfloor$	$x_k = (x_{k-1} \cdot b) \bmod 1$
...	...

A visszaalakítás pedig történhet szintén egy Horneres séma szerint. Az $x = 0,c_1c_2c_3\ldots c_n$ szám értelmezése ugyanis

$$x = c_1/b + c_2/b^2 + c_3/b^3 + \ldots + c_n/b^n \tag{17}$$

Ez úgy is számolható, hogy

$$x = (\ldots(((c_n)/b + c_{n-1})/b + c_{n-2})/b + \ldots c_1)/b \tag{18}$$

A séma kényelmes, felváltva kell számjegyenként osztást és összeadást végezni a jegyek fordított sorrendjében.

Példa: Írjuk fel a 0,52734375-öt kettes (= *bináris*) számrendszerben és 16-os (= *hexadecimális*) számrendszerben.

2	0,52734375
1	0,0546875
0	0,109375
0	0,21875
0	0,4375
0	0,875
1	0,75
1	0,5
1	0,0

16	0,52734375
8	0,4375
7	0,0

Tehát $0,52734375_{10}=0,10000111_2=0,87_{16}$

Példa: Visszaírás tizes számrendszerre: $0,87_{16}=0+((7)/16+8)/16=0,52734375$
$0,10000111_2=0+(((((((1)/2+1)/2+1)/2+0)/2+0)/2+0)/2+0)2+1)/2$
$0,52734375_{10}=0+(((((((5)/10+7)/10+3)/10+4)/10+3)/10+7)/10+2)/10+5)/10$

Némi kellemetlenséget jelenthet, hogy előfordul, hogy az egyik számrendszerben véges sok törtjegyet tartalmaz a szám, a másikban pedig végtelen sokat. Próbáljuk meg a 0,1 tizes számrendszerbeli számot átírni kettes számrendszerbe például!

Kettesből tizenhatosba az átírás egyszerű, mert a törtvesszőtől balra és jobbra négyes csoportokat írunk át.

Példa: $2006{,}52734375_{10}=11111010010{,}10000111_2=7D6{,}87_{16}$

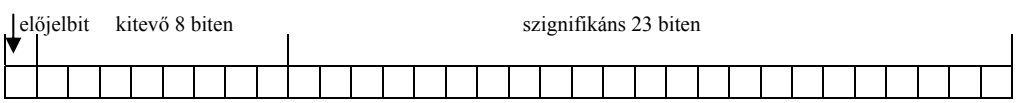
Definíció: Törtvesszős alakú szám értékes jegyeinek a száma
A törtvesszős alakú számot a gyakorlatban véges sok jegyével tüntetjük fel, így a többi jegyét nem ismerjük. Azt mondjuk, hogy csak valamilyen pontossággal adott a szám. Ennek a pontosságnak az egyik jelzőszáma az értékes jegyek száma, amit úgy kapunk meg, hogy a számot a felírásában balról jobbra vizsgáljuk és az első nemzérus számjeggyel kezdve a számlálást megszámláljuk a számjegyeket a lejegyzés utolsó számjegyével bezárva. A kapott darabszámot nevezzük a szám értékes jegyei számának.

Példa: 2006,52734375 értékes jegyeinek a száma 12. $11111010010{,}10000111_2$ értékes jegyeinek a száma 19, $7D6{,}87_{16}$ értékes jegyeinek a száma 5 Ugyanakkor -0,00012 értékes jegyeinek a száma 2.

A valós szám lebegőpontos implementációja

A valós számok implementálása nem egyszerű feladat. Meg kell alkudni. Erőteljesen. Nem tudunk minden valós számot implementálni és nem csak a számtartomány korlátozott mivolta miatt, mint az egész számoknál láttuk, hanem azért is, mert ezt csak a tartományból praktikusan jó sűrűn választott számokra tudjuk megtenni. Tulajdonképpen a törtvesszős átírást célozzuk meg. A processzorok korábbi nemzedékeiben tapasztalható tarkaságot, amely gyakran eltérő számítási eredményekre vezetett, ma szabvány szabályozza. Ez az IEEE 754-es szabvány, amely 1985 óta érvényes és William Kahan a Berkeley egyetem professzora nevéhez fűződik. A szabvány pontos előírásokat ad a valós számok ábrázolására, amely ábrázolást *lebegőpontos számok*nak nevezzük. Ezen ábrázolási formát nem tárgyaljuk teljes részletezettséggel, de a két együtt ismertethető esetről - az egyszeres pontosság és a dupla pontosság esete - szólunk néhány szót.

A szabvány szerinti lebegőpontos számábrázolás négy byte-on történik egyszeres pontosság esetén és nyolc byte-on dupla pontosság esetén. A kettő között eltérés igazán csak a pontosságban és az átfogott számtartományban van, az ábrázolás elve azonos. Tekintsük először a *normalizált szám* esetét. Legyen a szám nemzérus. Ekkor a binárisan felírt számot átalakítjuk olyan formára, hogy a törtvesszőt a legelső egyes jegyet közvetlenül követően helyezzük el és megjegyezzük, hogy ezen művelethez a törtvesszőt hány bitpozícióval kellett balra mozgatni. Ez a szám balra mozgásnál pozitív, jobbra mozgásnál negatív lesz és azt mutatja, hogy az átalakítás utáni számot a 2 milyen kitevőjű hatványával kell megszorozni, hogy a kiinduló számot megkapjuk. A törtvesszőt követő bitek sorozatának neve: *szignifikáns*. Ezen információkat kell elhelyeznünk a rendelkezésre álló négy illetve nyolc byte-on. A bitek kiosztása az egyszeres pontosság esetén:



3. byte	2. byte	1. byte	0. byte

Egyszeres pontosságú lebegőpontos szám

Az előjelbit pozitív szám esetén zérus, negatív szám esetén 1. A kitevő részére fenntartott 8 bites mezőbe a kitevő 127-tel megnövelt (eltolt) értékét helyezzük el előjel nélküli egész számként. A szignifikáns (a vezető egyes nélkül, implicit egyes bit) kerül a hátra maradt 23 bites mezőbe.

Példa: Példa: 2006,52734375 hogyan néz ki egyszeres pontosságú lebegőpontos számként? A szám binárisan, ahogy már kiszámoltuk: $11111010010{,}10000111_2$. Normalizált alakban: $1{,}11110100101000011 1_2 \times 2^{10}$, ahol a kitevő decimálisan 10. Ez eltolva $10+127=137=10001010_2$. (Negatív kitevőt 8 biten kettes komplemens módon tárolunk, majd így adjuk hozzá a 127-et.) A szignifikáns 23 bitre zérusokkal kiegészítve: 11110100101000011100000. Végül a 32 bit 0100 0101 0111 1010 0101 0000 1110 0000, vagy hexadecimálisan 45 7A 50 E0.

A szám ebben a formában történő ábrázolása csak akkor megengedett, ha az eltolt kitevő nem zérus és 255 közé esik, tehát a szélső eseteket (0 és 255) kizárjuk. Ez a két szélső eset más célra van fenntartva. A tiszta zérus biteket tartalmazó kitevő mező és a zérus szignifikáns együtt *zérus*ként van definiálva. Van pozitív zérus és negatív zérus az előjeltől függően, de valójában a processzornak ezeket azonosként kell kezelnie. Ha a kitevő mező zérus, de a szignifikáns mező nem zérus, akkor *nem normalizált* (denormalizált) lebegőpontos számról beszélünk. Ekkor az implicit egyes bit is tárolásra kerül, mint a szignifikáns része, mivel ő a törtvessző mögé kerül. Denormalizált tárolásnál komoly jegyveszteségre lehet számítani! Például a 2^{-126} még normalizált módon tárolódik, de a tőle kisebb kitevőjűeknél már az eddig elhagyott egyes bitet is tároljuk. Az alábbi táblázat illusztrál néhány esetet.

Szám	Byte-ok binárisan	Byte-ok hexában
2^{-126}	0000 0000 1000 0000 0000 0000 0000 0000	00 80 00 00
2^{-127}	0000 0000 0100 0000 0000 0000 0000 0000	00 40 00 00
2^{-128}	0000 0000 0010 0000 0000 0000 0000 0000	00 20 00 00
…	…	…
2^{-149}	0000 0000 0000 0000 0000 0000 0000 0001	00 00 00 01

A kitevő mező legmagasabb értékéhez szintén két eset tartozik. Ha a szignifikáns mező zérus, akkor a tárolt információ *előjeles végtelen*ként van definiálva.

Szimbólum	Byte-ok binárisan	Byte-ok hexában
$+\infty$	0111 1111 1000 0000 0000 0000 0000 0000	7F 80 00 00
$-\infty$	1111 1111 1000 0000 0000 0000 0000 0000	FF 80 00 00

A végtelen kezelése során a processzor a végtelennel végezhető műveletek tulajdonságait megtartja. Például végtelen plusz véges eredménye végtelen, vagy véges osztva végtelennel zérust ad. Ha a szignifikáns rész nem zérus, akkor ezt a szituációt *nem szám*ként definiálták (NaN=Not a Number).

Szimbólum	Byte-ok binárisan
NaN	0111 1111 1xxx xxxx xxxx xxxx xxxx xxxx
NaN	1111 1111 1xxx xxxx xxxx xxxx xxxx xxxx

A sémában az x-szel jelölt bitek nem lehetnek egyszerre mind zérusok. Ilyen eset (NaN) lehet például a végtelen osztva végtelennel művelet eredménye. Azok a számok, értékek, állapotok,

amelyek a fenti sémába nem férnek bele, nem ábrázolhatók, velük közvetlen módon számolni nem tudunk.

A dupla pontosságú esetben a nyolc byte-ban a kitevő mező 11 bites, a szignifikáns mező 52 bites. A kitevő eltolás mértéke 1023. A kitevő mező két kitüntetett értéke a zérus és az 2047. Egy táblázatban mellékeljük a lebegőpontos aritmetika lehetőségeit, korlátait:

Jellemzők	Egyszeres pontosság	Dupla pontosság
Előjelbitek száma	1	1
Kitevő bitek száma	8	11
Törtrész bitek száma	23	52
Összes bitek száma	32	64
Kitevő ábrázolása	127-es eltolás	1023-as eltolás
Kitevő tartománya	-126 - +127	-1022 - +1023
Legkisebb normalizált szám	2^{-126}	2^{-1022}
Legnagyobb normalizált szám	kb. 2^{128}	kb. 2^{1024}
Decimális számtartomány	kb. 10^{-38} - 10^{38}	kb. 10^{-308} - 10^{308}
Legkisebb nem normalizált szám	kb. 10^{-45}	kb. 10^{-324}
Értékes decimális jegyek száma normalizált esetben	≈7	≈16

FELADATOK

1. a. Bizonyítsuk be az alsó és felső egészrész függvényeknek a szövegben összefoglalt 1.-7- tulajdonságait!

2. Töltsük ki az alábbi táblázatot!

Számrendszer alapszáma és a szám						
2	3	8	10	12	16	36
10101110						
	120120					
		1234567				
			1973			
				1234		
					9AB	
						XYZ

3. Töltsük ki az alábbi táblázatot!

Számrendszer alapszáma és a szám						
2	3	8	10	12	16	36
101110,111						
	120,111					
		4567,111				
			973,111			
				234,111		
					AB,111	
						YZ,111

4. a. Töltsük ki az alábbi táblázatot decimálisan!

Byte hexában	00	01	12	13	20	30	35	62	80	81	A0	E9	FF
Előjel nélküli egész													
Előjeles egész													

b. Töltsük ki az alábbi táblázatot decimálisan!

Szó hexában	00 01	12 13	20 30	35 A0	80 81	E9 FF
Előjel nélküli egész						
Előjeles egész						

c. Töltsük ki az alábbi táblázatot decimálisan! (Lebegőpontos esetben 7 értékes jegyre, szükség esetén decimálisan normalizálva, ha a szám kisebb, mint 10^{-7}, vagy nagyobb, mint 10^{7}.)

Duplaszó hexában	00 01 12 13	20 30 35 A0	80 81 E9 FF
Előjel nélküli egész			
Előjeles egész			
Lebegőpontos			

5. Egy millió bitet akarunk elhelyezni a memóriában egymást követő byte-okon. Hány byte-nak foglaljunk helyet? Hány szónak? Hány dupla szónak? Hány quadr-nak (dupla duplaszó)?

6. a. Töltsük ki az alábbi táblázatot!

Jegyek száma / Számrendszer / Szám	2	10	16
2^2			
10^{10}			
16^{16}			

   b. Ha egy pozitív egész szám 34 jegyű 2-es számrendszerben, hány jegyű 16-osban? Adható-e általános formula bináris *n*-jegyű számok esetére? Ha igen, adjon ilyet, ha nem, indokolja meg, miért nem!

   c. Ha egy pozitív egész szám 9 jegyű 16-os számrendszerben, hány jegyű 2-es számrendszerben? Adható-e általános formula hexadecimális *n*-jegyű számok esetére? Ha igen, adjon ilyet, ha nem, indokolja meg, miért nem!

7. Adja meg a 3-as, a 4-es, a 8-as, és a 12-es összeadó- és szorzótáblákat!

8. Milyen módon kapcsolódik egymáshoz az „orosz paraszt” módszer és a kettes számrendszer?

9. Legyen $x = \alpha + k \cdot \beta$, $k = 0,1,2,3,4,5$! Itt $\alpha = -23$ és $\alpha = 10$ lehet, valamint $\beta = -0,72$ és $\beta = 0,91$. Határozza meg az $\lfloor x \rfloor$, $\lceil x \rceil$, $\{x\}$, és $Round(x)$ értékeket az összes lehetséges alfa, béta és k esetére. Készítsen az eeredmények számára alkalmas szemléltető táblázatokat!

10. Legyen A={◊, □} és B={·, ○, *}! Határozza meg az A×B, B×A, A^3, és B^2 halmazokat!

11. Legyen az *a* a 9. feladat végeredményei közül az alsó egészrész, *b* pedig legyen rendre -5, -2, 0, 2, 5! Határozza meg az *a* div *b* és *a* mod *b* értékeket!