Zobrazování čísel v počítačích

Čísla v počítačích

Čísla v počítačích je třeba ukládat do logických obvodů (registrů, pamětí, …), příp. přenášet po sběrnicích
⇒ užívání dvojkové soustavy (polyadická soustava o základu $z = 2$):

$$A = a_{n-1}.z^{n-1} + \ldots + a_0.z^0 + a_{-1}.z^{-1} + \ldots + a_{-m}.z^{-m}$$

Polyadické soustavy zobrazují pouze nezáporná čísla
⇒ k zobrazování záporných čísel používáme transformace (*číselné kódy*); nejčastější:

- přímý se znaménkem
- inverzní
- doplňkový
- aditivní

Řádová mřížka

Rozdělení na celou a zlomkovou část označujeme jako *řádová mřížka,* číslo zapsané v mřížce je *slovo*

Nelze-li zapsat číslice v nejvyšších řádech, mluvíme o *přeplnění* (přetečení, overflow), v nejnižších řádech o *ztrátě přesnosti*

n-1	**n-2**		**0**	**-1**			**-m**

n-1 … nejvyšší řád řádové mřížky (celá část má velikost n bitů)

-m … nejnižší řád řádové mřížky (zlomková část má velikost m bitů)

N … délka řádové mřížky (počet obsažených řádů) $N = n + m$

$\varepsilon = 2^{-m}$ … jednotka řádové mřížky (nejmenší zobrazitelné číslo)

$M = 2^n$ … modul řádové mřížky (nejmenší číslo, které již v řádové mřížce není zobrazitelné)

Formáty nezáporných čísel

Čísla bez znaménka (**unsigned**)

- *Celočíselný formát* (tj. $m = 0$, $N = n$):

$$U = u_{n-1}.2^{n-1} + \ldots + u_0.2^0$$

- *Zlomkový (fraction) formát* – řádová čárka se umísťuje těsně za nejvyšší bit, který je řádu 2^0 ($n = 1$, $N = m+1$)

$$U = u_0.2^0 + u_{-1}.2^{-1} + \ldots + u_{-m}.2^{-m}$$

Z pohledu implementace aritmetických operací ± nezáleží na tom, kde je umístěna řádová čárka.

např. 01001 + 00100 = 01101

9 + 4 = 13   nebo   0,5625 + 0,25 = 0,8125

Příklad

Převeďte číslo $(258{,}125)_{10}$ do dvojkové soustavy

Příklad

$(258{,}125)_{10} : 2 = 0100000010{,}001$

129

64

32

16

8

4

2

1

$(0{,}125)_{10} \cdot 2 =$

0,25

0,5

Příklad

Převeďte číslo $(1{,}1)_{10}$ do dvojkové soustavy

Příklad

$(1,1)_{10} : 2 = 1,0001101$ =>

=> $(1 + 0,0625 + 0,03125 + 0,0078125 = 1,1015625)_{10}$

$(0,1)_{10} \cdot 2 =$

0,2

0,4

0,8

0,6

0,2

0,4

0,8… zaokrouhleno nahoru

Příklad

Sečtěte $(213)_{10}$ a $(213)_{10}$

Příklad

Sečtěte $(213)_{10}$ a $(213)_{10}$

$$
\begin{array}{lr}
(213)_{10} = & 1101\ \ 0101 \\
\underline{+(213)_{10} =} & \underline{1101\ \ 0101} \\
 & 11010\ \ 1010 \quad = \quad (426)_{10}
\end{array}
$$

=> násobení dvěma je posun o 1 bit vlevo a zvětšuje počet bitů o jeden

Přímý kód se znaménkem

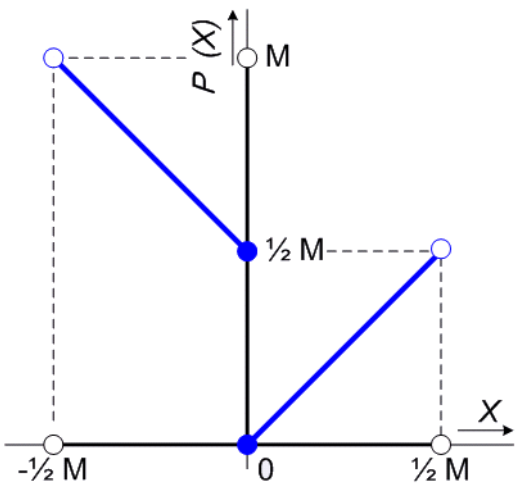
také „přirozený kód“

absolutní hodnota čísla se znaménkovým bitem; $0 \sim (+)$, $1 \sim (-)$

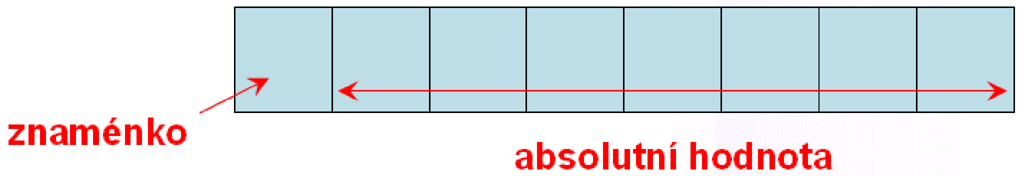
$$P(X) = X \quad \text{pro } X \geq 0 \qquad\qquad P(X) = |X| + \tfrac{1}{2}M \quad \text{pro } X \leq 0$$

- složitá realizace aritmetických operací
  nejprve třeba otestovat znaménko
  pak se použije algoritmus operace
  (sčítání, odečítání)
- nevýhodou jsou dvě reprezentace nuly
  (nutno ošetřit)

$$-\tfrac{1}{2}M \leq X < \tfrac{1}{2}M$$



Přímý kód se znaménkem

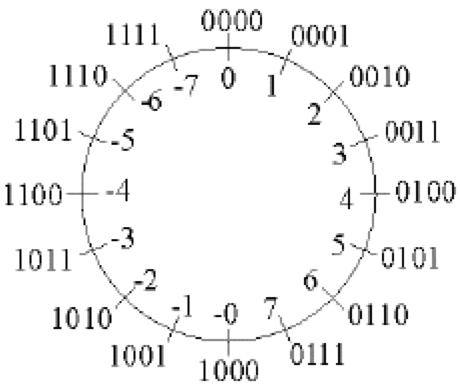


Např. pro 4-bitové slovo (n-bitové)

kladná čísla: 0 … 7 (2^{n-1}-1)

záporná čísla: -7 … 0

dvě nuly: 0000 a 1000



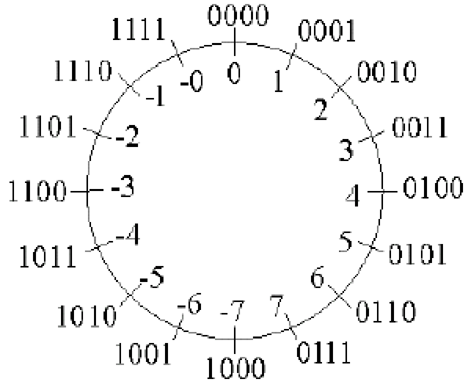
Inverzní kód

Vychází z jednotkového doplňku (one's complement)

$I(X) = X$ pro $X \geq 0$ $I(X) = M - \varepsilon + X$ pro $X \leq 0$

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

- opět problém dvou nul (0000 a 1111)
- obtížnější realizace aritmetických operací
- vznikne negací bitů daného slova
- MSB bit má opět charakter znaménka



Např. $+0{,}8125 \sim 0{,}1101$

$-0{,}8125 \sim 1{,}0010$

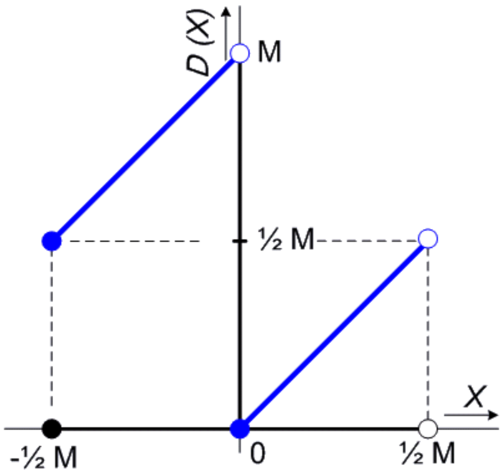
Doplňkový kód

Vychází z dvojkového doplňku (two’s complement)

$$D(X) = X \quad \text{pro } X \geq 0 \qquad\qquad D(X) = M + X \quad \text{pro } X < 0$$

$-\tfrac{1}{2}M \leq X < \tfrac{1}{2}M$

- nejvyšší bit má opět charakter znaménka
  (nenese informaci o hodnotě)
- vznikne přičtením ε k inverznímu kódu
- max. záp. číslo nemá kladný ekvivalent
- algoritmus odečítání je stejný jako sčítání
  (sečtou se obrazy a ignoruje se přenos)
- nejpoužívanější



Doplňkový kód

Doplňkový kód je možné definovat také vztahem:

$$D(X) = S = -s_{n-1}.2^{n-1} + \sum_{i=-m}^{n-2} s_i.2^i$$

platí pro celý rozsah X

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

Doplňkový kód chápeme jako **signed** (se znaménkem):

- *Celočíselný formát* (tj. $m = 0$, $N = n$):

$$S = -s_{n-1}.2^{n-1} + \sum_{i=0}^{n-2} s_i.2^i$$

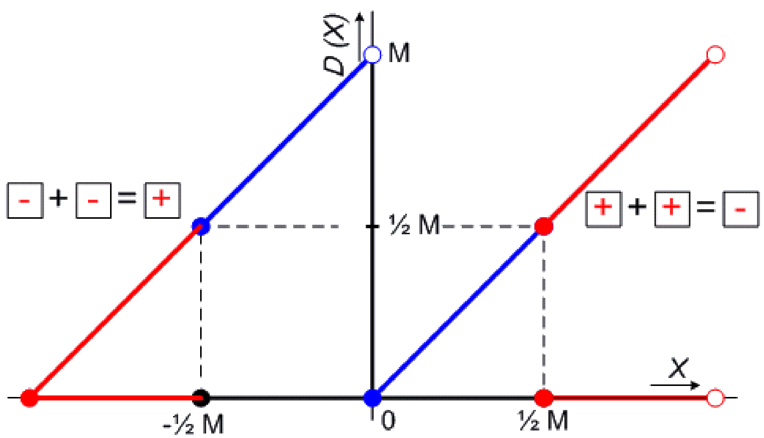
- *Zlomkový (fraction) formát* (tj. $n = 1$, $N = m+1$)

$$S = -s_0 + \sum_{i=-m}^{-1} s_i.2^i$$

Doplňkový kód - sčítání

Při operacích typu (+)+(+) a (-)+(-) může dojít k přetečení
(výsledek je mimo rozsah platných bitů);
v některých počítačích se přetečení detekuje a uchovává se,
v jiných se nedetekuje – musí ohlídat programátor

Při operacích typu (-)+(+) a (+)+(-) nemůže k přetečení dojít



Příklady sčítání v doplňk. kódu

Chybné výsledky při malé délce n-bitového slova (přetečení):

-9	10111	9	01001	001001
-8	11000	8	01000	001000
-17	1|01111	17	10001	010001

Také pro výsledek musí platit: $-\frac{1}{2}M \leq X < \frac{1}{2}M$ $(M = 2^n)$

9	01001	-9	10111
-5	11011	4	00100
4	1|00100	-5	11011

Pokud je výsledek záporný, jeho absolutní hodnotu získáme opět pomocí dvojkového doplňku

Příklad

Sečtěte binární reprezentace $(47)_{10}$ a $(-20)_{10}$

Příklad

Sečtěte binární reprezentace $(47)_{10}$ a $(-20)_{10}$

$(20)_{10}$ = 0001 0100 –> invertovat –> 1110 1011 –> přičíst 1 –> 1110 1100

nebo opsat zprava až za první 1 a zbytek invertovat –> 1110 1100 = $(-20)_{10}$

$$
\begin{array}{lcrcl}
(47)_{10} & = & 0010\ 1111 & & \\
\underline{+(-20)_{10}} & \underline{=} & \underline{1110\ 1100} & & \\
 & & (1)0001\ 1011 & = & (27)_{10}
\end{array}
$$

Příklad

Sečtěte binární reprezentace $(-100)_{10}$ a $(-20)_{10}$

Příklad

Sečtěte binární reprezentace $(-100)_{10}$ a $(-20)_{10}$

$(100)_{10}$ = 0110 0100 –> 1001 1100 = $(-100)_{10}$

$(20)_{10}$ = 0001 0100 –> 1110 1100 = $(-20)_{10}$

$$
\begin{array}{lr}
(-100)_{10} = & 1001\ \ 1100 \\
\underline{+(-20)_{10} =} & \underline{1110\ \ 1100} \\
 & (1)1000\ \ 1000 \quad = \quad (-120)_{10}
\end{array}
$$

Násobení v doplňkovém kódu

(+2)·(-3) = (-6)

				0	0	1	0
				1	1	0	1
				0	0	1	0
			0	0	0	0	
		0	0	1	0		
	0	0	1	0			
1	1	1	0				
1	1	1	1	1	0	1	0

Nakonec jsme přičetli (-2), výsledek je ve dvojk. doplňku

(-3)·(+2) = (-6)

			1	1	0	1
			0	0	1	0
0	0	0	0	0	0	0
1	1	1	1	0	1	
0	0	0	0	0		
0	0	0	0			
1	1	1	1	0	1	0

Vlevo šíříme znaménka mezivýsledků, výsledek je ve dvojkovém doplňku

Příklady násobení – Boothův alg.

(+2).(-3) = (-6)

A	0010 0000 0	(= +2)
B	1110 0000 0	(= -2)
C	0000 1101 0	(= -3) „10“ -> C = C + B a arit. posun vpravo
C	1110 1101 0	
C	1111 0110 1	„01“ -> C = C + A a arit. posun vpravo
C	0001 0110 1	
C	0000 1011 0	„10“ -> C = C + B a arit. posun vpravo
C	1110 1011 0	
C	1111 0101 1	„11“ či „00“ -> opíše a arit. posun vpravo
C	1111 0101 1	
C	1111 1010	(= -6)

Dělení

Obtížnější realizace, obecně zdlouhavé.

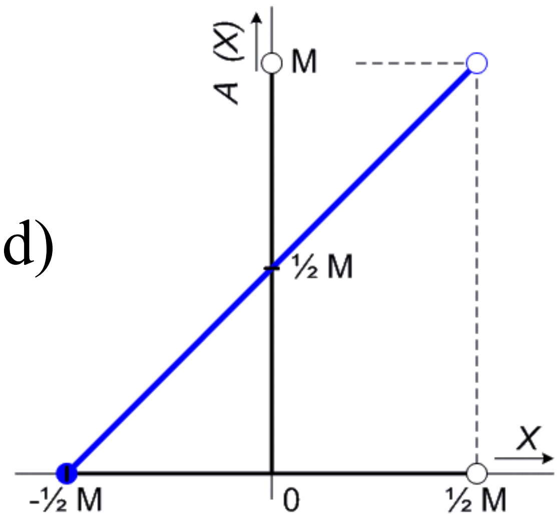
Nejčastější algoritmy:

- *s restaurací nezáporného zbytku* (s návratem přes nulu), odečítá se příslušně posunutý dělitel od dělence
- *bez restaurace nezáporného zbytku* (bez návratu přes nulu) – obdoba předešlého algoritmu, rychlejší
- *násobení převrácenou hodnotou*
- *podmíněné odečítání* – zjišťujeme kolikrát se dělitel vejde do dělence, porovnávání a odečítání

Aditivní (posunutý) kód

Kód s posunutou nulou – k číslu připočteme známou konstantu K: $A(X) = X + K$ (binary offset)

- zachovává relace (</>)
- 2 varianty:
  - $\Rightarrow K = 2^{n-1}$, tj. $\frac{1}{2}M$ (sudý aditivní kód)
  - $K = 2^{n-1} - \varepsilon$ (lichý aditivní kód)



- před násobením se musí odečíst K
- převod do doplňkového kódu provedeme negací nejvyššího bitu (pro sudý aditivní kód)
- použití: reprezentace exponentu reálných čísel, ADC…

Aditivní (posunutý) kód

sudý:

$A(X) = X + 2^{5-1} = X + 16$

X_{10}	X_2	A_2
3	00011	10011
2	00010	10010
1	00001	10001
0	00000	10000
-1		01111
-2		01110
-3		01101

lichý:

$A(X) = X + 2^{5-1}-1 = X + 15$

X_{10}	X_2	A_2
3	00011	10010
2	00010	10001
1	00001	10000
0	00000	01111
-1		01110
-2		01101
-3		01100

Příklad

	přímý	**inverzní**	**doplňkový**	**aditivní sudý**	**aditivní lichý**
9					
-9					
9/16					
-9/16					

Příklad

	přímý	**inverzní**	**doplňkový**	**aditivní sudý**	**aditivní lichý**
9	0 1001	0 1001	0 1001	1 1001	1 1000
-9					
9/16					
-9/16					

Příklad

	přímý	**inverzní**	**doplňkový**	**aditivní sudý**	**aditivní lichý**
9	0 1001	0 1001	0 1001	1 1001	1 1000
-9	1 1001	1 0110	1 0111	0 0111	0 0110
9/16					
-9/16					

Příklad

	přímý	**inverzní**	**doplňkový**	**aditivní sudý**	**aditivní lichý**
9	0 1001	0 1001	0 1001	1 1001	1 1000
-9	1 1001	1 0110	1 0111	0 0111	0 0110
9/16	0, 1001	0, 1001	0, 1001	1, 1001	1, 1000
-9/16					

Příklad

	přímý	inverzní	doplňkový	aditivní sudý	aditivní lichý
9	0 1001	0 1001	0 1001	1 1001	1 1000
-9	1 1001	1 0110	1 0111	0 0111	0 0110
9/16	0, 1001	0, 1001	0, 1001	1, 1001	1, 1000
-9/16	1, 1001	1, 0110	1, 0111	0, 0111	0, 0110

Pohyblivá řádová čárka

Floating point (FP) – oproti FX zvětšení rozsahu (nezvyšuje se počet rozlišitelných hodnot)

$$X_{FP} = M.z^{E}$$

Mantisa (M) – informace o hodnotě čísla (určuje přesnost)

Exponent (E) – informace o pozici řádové čárky (určuje rozsah)

z … základ použité číselné soustavy (nejčastěji 2)

Dvě řádové podmřížky: většinou každá v různém formátu - mantisa ve zlomkovém tvaru v přímém kódu, exponent celočíselný většinou v kódu s posunutou nulou

IEEE 754 - formát reálného čísla

- nejvyšší bit mantisy je vždy 1 a nezobrazuje se (hidden one); toto neplatí, je-li obraz exponentu nulový (zobrazení 0)
- myšlená řádová čárka je za nejvyšším bitem mantisy
- posunutí exponentu je $K = 2^7-1=127$; $2^{10}-1=1023$…
- kladné začíná 0, záporné 1 (znaménkový bit S)

Jednoduchá přesnost (binary32 – single prec.): 32 bitů (4 byte)

0|00000000|00000000000000000000000

1b znaménko mantisy, 8b exponent, 23b mantisa

Dvojnásobná přesnost (binary64 – double prec.): 64 bitů

1b znaménko mantisy, 11b exponent, 52b mantisa

IEEE 754

Hodnota exponentu v aditivním kódu: $exp = E + K$

Zpětná transformace: $X_{FP} = (-1)^{S} \cdot 2^{exp-K} \cdot (1,M_{IEEE})$

Formát IEEE 754 má speciální hodnoty (výjimky):

exp = max. (255) a $M_{IEEE} = 0$	=>	$\pm\infty$
exp = max. (255) a $M_{IEEE} \neq 0$	=>	neplatné č. „NaN“
$exp = 0$ a $M_{IEEE} = 0$	=>	± 0 (daná S)
$exp = 0$ a $M_{IEEE} \neq 0$	=>	$E=E+1$ a zároveň
(denormalizovaná čísla, subnormal numbers)		nula před M_{IEEE}

Příklad – IEEE 754

Zobrazte ve formátu IEEE na 4 bytech reálné číslo $(-258{,}125)_{10}$

$(258{,}125)_{10} = (100000010{,}001)_2 = 1{,}00000010001 \cdot 2^8$

exponent: $2^7-1+8 = (10000111)$

$\mathbf{(-258{,}125)_{10} = (1100\ 0011\ 1000\ 0001\ 0001\ 0000\ 0000\ 0000)_{IEEE}} =$

$\mathbf{= (\ C\ \ 3\ \ 8\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ )_{16}}$

Pozn.:

konvence uspořádání vícebytových slov (pořadí v paměti):

big endian – nejvýznamnější byte první (C3 81 10 00)

little endian – nejnižší byte první (00 10 81 C3)

Příklad

Převeďte číslo $(4291\ \ 4000)_{16}$ do dekadické soustavy (dle IEEE 754)

Příklad

Převeďte číslo $(4291\ \ 4000)_{16}$ do dekadické soustavy (dle IEEE 754)

4	2	9	1	4	0	0	0
0100	0010	1001	0001	0100	0000	0000	0000

$$X_{FP} = (-1)^0 \cdot 2^{133-127} \cdot 1{,}001000101 = 1001000{,}101 \cdot 2^6$$
$$X_{FP} = (72{,}625)_{10}$$

Příklad

Převeďte číslo $(C396\ \ 7000)_{16}$ do dekadické soustavy (dle IEEE 754)

Příklad

Převeďte číslo $(C396\ \ 7000)_{16}$ do dekadické soustavy (dle IEEE 754)

C	3	9	6	7	0	0	0
1100	0011	1001	0110	0111	0000	0000	0000

$E = 10000111 = (135)_{10}$ $\qquad$ $M = 1{,}00101100111000000000000$

$e = E - 127 = 135 - 127 = 8$

$FLT = -100101100{,}111 = -300 + 2^{-1} + 2^{-2} + 2^{-3} = (-300{,}875)_{10}$

Příklad

Převeďte číslo $(47{,}0625)_{10}$ do binární soustavy (dle IEEE 754)

Příklad

Převeďte číslo $(47{,}0625)_{10}$ do binární soustavy (dle IEEE 754)

$(47{,}0625)_{10} = (101111{,}0001)_2 = 1{,}011110001 \cdot 2^5$

exponent: $5+127 = (10000100)$

$\mathbf{(47{,}0625)_{10} = (0100\ 0010\ 0011\ 1100\ 0100\ 0000\ 0000\ 0000)_2 =}$

$\mathbf{= (\ 4 \quad 2 \quad 3 \quad C \quad 4 \quad 0 \quad 0 \quad 0\ )_{16}}$

Příklad

Převeďte číslo $(-16{,}484375)_{10}$ do hexadecimální soustavy (dle IEEE 754)

Příklad

Převeďte číslo $(-16{,}484375)_{10}$ do hexadecimální soustavy (dle IEEE 754)

$FLT = (10000{,}011111)_2 = 1{,}0000011111 \;.\; 2^4$

$e = 4$

$E = e + 127 = (131)_{10} = (10000011)_2$

$$(1100\ 0001\ 1000\ 0011\ 1110\ 0000\ 0000\ 0000)_2 =$$

$$= (\ \ C\ \ \ 1\ \ \ 8\ \ \ 3\ \ \ E\ \ \ 0\ \ \ 0\ \ \ 0\ \ )_{16}$$

Doplnění k single/float a double

Chyby při výpočtech:

např. S = S * 11 – 1 (při počáteční hodnotě S = 0,1 má výsledek být 0,1)

```
float   S;
uint8_t I;

S = 0.1f;
for(I = 0 ; I<11 ; I++)
{
   WriteLn(I,' : ',S);
   S = S * 11 - 1;
}
```

Výsledek:

```
0 : 0.100000001490116
1 : 0.100000016391277
…
7 : 0.129038155078888
8 : 0.419419705867767
9 : 3.613616704940 8
10 : 38.7497825622559
```