Zobrazenie čísiel v počítači

- Celé nezáporné čísla - čísla bez znamienka
  - dvojková sústava
  - iné používané sústavy - šestnástková a osmičková
  - BCD kódovanie - Binary-Coded Decimal
- Všetky celé čísla - čísla so znamienkom
  - priamy kód
  - dvojkový doplnok
  - kód posunutej nuly (kód "excess N")
- Reálne čísla (v skutočnosti podmnožina racionálnych čísiel)
  - s pevnou čiarkou
  - s pohyblivou čiarkou
- Celé a reálne čísla v Pythone a C

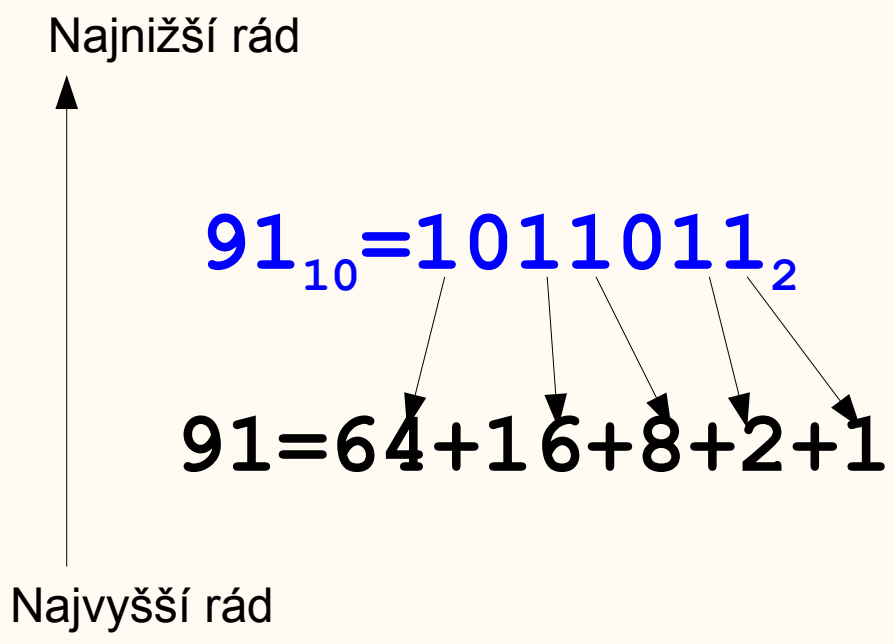
Použité obrázky z učebnice: Andrew. S. Tanenbaum, Structured Computer Organization
http://www.cs.vu.nl/~ast/books/

Celé čísla bez znamienka

Počítače pracujú v dvojkovej sústave

Prevod z desiatkovej do dvojkovej sústavy:

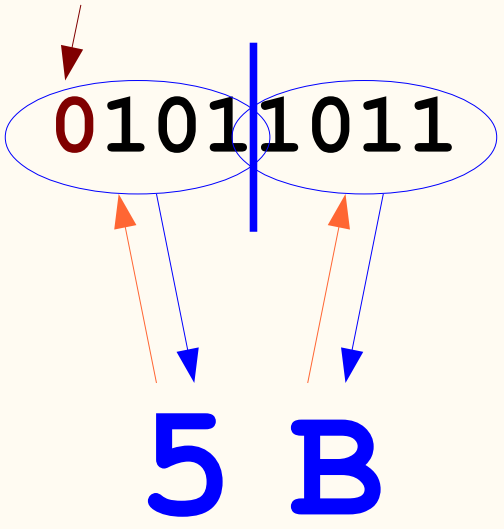
```
91 : 2 = 45 zvyšok 1
45 : 2 = 22 zvyšok 1
22 : 2 = 11 zvyšok 0
11 : 2 =  5 zvyšok 1
5  : 2 =  2 zvyšok 1
2  : 2 =  1 zvyšok 0
1  : 2 =  0 zvyšok 1
```



Šestnástková sústava

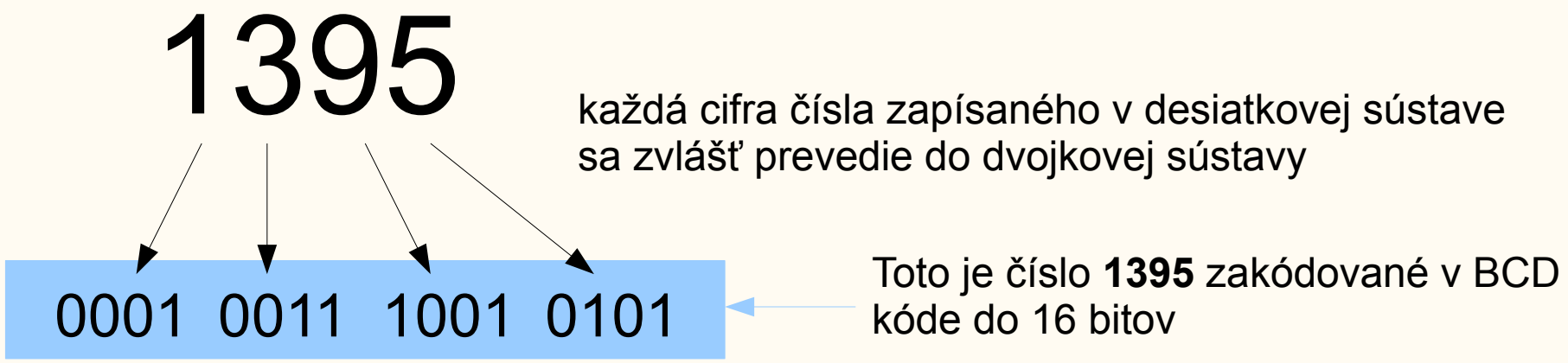
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Ak počet cifier nie je deliteľný 4, tak ho zľava doplníme nulami.



Prevod medzi dvojkovou a šestnástkovou sústavou je jednoduchý lebo pri ňom netreba deliť.
Pri prevode z dvojkovej do šestnástkovej sústavy sa v čísle vytvoria skupiny po 4 cifrách (lebo 2^4=16) a každá z nich sa prevedie na jednu cifru podľa tabuľky.
Pri prevode zo šestnástkovej do dvojkovej sústavy sa každá šestnástková cifra vyjadrí štyrmi dvojkovými ciframi.

BCD - Binary-Coded Decimal



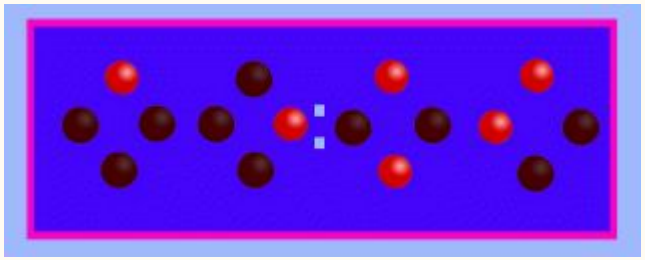
- Používal sa v starších procesoroch
- Pre pomalšie počítače bol dobrým kompromisom dosť rýchleho prevodu z/do textového tvaru a rýchlosťou **jednoduchých** aritmetických výpočtov.

Binárne hodiny

Úloha z Informatického bobra 2011/12, kategória Junior

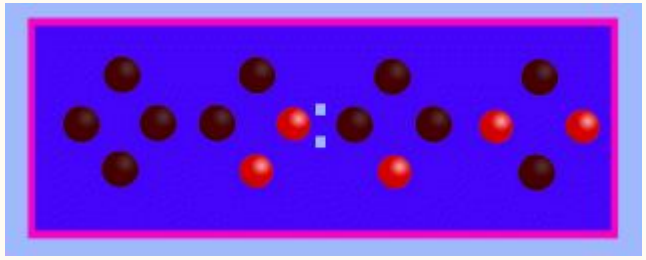
Tieto binárne hodiny ukazujú čas

12:59

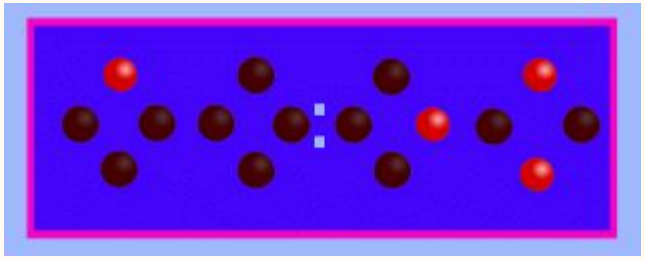


Ktoré z nasledujúcich binárnych hodín ukazujú čas?

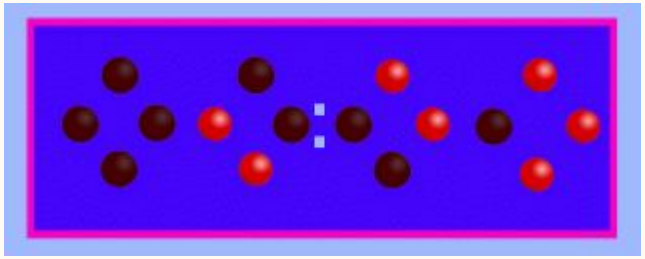
a



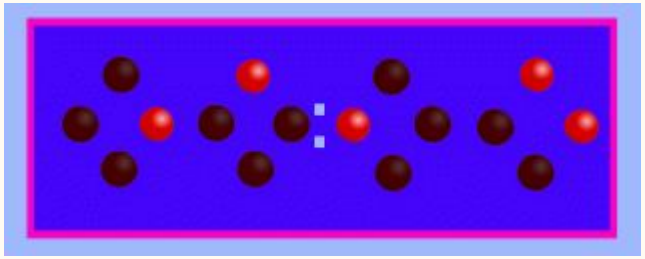
c



b



d



Celé čísla so znamienkom
Priamy kód

Oddelíme jeden bit pre znamienko
V N bitoch teda použijeme

- N-1 bitov na absolútnu hodnotu čísla a
- 1 bit na znamienko: 0 znamená plus, 1 znamená mínus

```
 5 ≈ 00000101
-5 ≈ 10000101
```

Toto zobrazenie sa v praxi používa zriedka, lebo:

- Sčítačka, ktorá vie sčitovať čísla bez znamienka sa nedá použiť na takéto čísla, treba urobiť zložitejší obvod
- existuje v ňom kladná aj záporná nula

Celé čísla so znamienkom
Dvojkový doplnkový kód

Zaveďme operáciu $\oplus$ ako sčítanie modulo 5, teda $a \oplus b = (a+b) \bmod 5$, v takejto aritmetike potom platí napríklad:

$1 \oplus 1 = 2$, $1 \oplus 2 = 3$, $1 \oplus 3 = 4$, $1 \oplus 4 = 0$

$2 \oplus 2 = 4$, $2 \oplus 3 = 0$, $0 \oplus 0 = 0$

Z modrých rovností vidíme, že pri takejto operácii sa 4 chová ako číslo opačné k 1, teda **4 sa chová ako -1** a podobne **3 sa chová ako -2** a 0 je opačná sama k sebe.

Vo všeobecnosti číslo (5-x) mod 5 sa chová vzhľadom k operácii $\oplus$ ako číslo opačné k x.

N- bitová sčítačka počítača počíta na N dvojkových miest - **to je aritmetika modulo 2^N**.

Ak teda zobrazíme -1 ako 2^n-1, -2 ako 2^n-2 atď., tak nemusíme vôbec meniť sčítačku - bude vedieť sčitovať kladné aj záporné čísla

Celé čísla so znamienkom
Dvojkový doplnkový kód (2)

- Príklad: ako je zobrazené číslo -10 v dvojkovom doplnkovom kóde na N=8 bitov?

Postup 1:
-10 bude v 8 bitoch zobrazené ako 2^8-10=256-10=246.
Preveďme teda 246 do dvojkovej sústavy:
$\mathbf{246_{10}=11110110_2=F6_{16}}$

Skúška správnosti
`00001010` (zobrazuje 10)
`+11110110` (zobrazuje -10)
`00000000` vyšlo nám to: 10+(-10)=0

Postup 2:
Číslo zmeníme na opačné na N bitov:
•doplníme zľava nuly na N bitov
•zameníme nuly za jedničky a naopak
•pripočítame 1

doplnené nuly

$10_{10}=00001010_2$
`11110101` (zameníme 0 a 1)
`+00000001` (pripočítame 1)
`11110110` (toto je -10)

- Je to **najpoužívanejší kód** pre kódovanie celých čísiel v súčasných procesoroch

Celé čísla so znamienkom Dvojkový doplnkový kód (3)

- Na sčítanie a odčítanie možno použiť rovnaký hardware ako pre čísla bez znamienka
- Pre násobenie a delenie ale treba vedieť, či sú čísla bez znamienka alebo so znamienkom (výsledok násobenia sa vždy zapisuje do dvojnásobného počtu bitov):

  Neznamienkovo: 05*FF=5*255=1275=04FB
  Znamienkovo: 05*FF=5*(-1)=-5=FFFB

- Aj pri porovnaniach treba vedieť, či ide o čísla so znamienkom alebo bez znamienka:

  Neznamienkovo: 05 < FF lebo 5 < 255
  Znamienkovo: 05 > FF lebo 5 > -1

Celé čísla so znamienkom
Kód posunutej nuly

- Číslo x zobrazíme ako x+S kde S je pevne dané číslo.
- Napríklad pre 8 bitov a S=127 dostaneme takéto kódovanie (nazývané aj „excess 127“):

číslo	zobrazené	dvojkovo
-127	0	00000000
-126	1	00000001
...		
-1	126	01111110
0	127	01111111
1	128	10000000
2	129	10000001
...		
127	254	11111110
128	255	11111111

- Výhodou je, že pre porovnávanie zakódovaných čísiel netreba vedieť ich kódovanie (menšie čísla sa zobrazia do menších, väčšie do väčších)
- Používa sa pre špeciálne účely (napr. exponent čísla v pohyblivej čiarke)

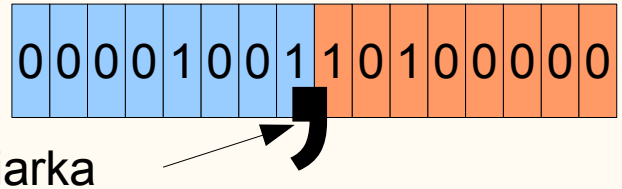
Reálne čísla
Zobrazenie s pevnou čiarkou

- V dvojkovej sústave sa dajú zobraziť aj necelé čísla rovnako ako v desiatkovej sústave:

  $12.35_{10} = 1*10^{1}+2*10^{0}+3*10^{-1}+5*10^{-2}$

  $1001.101_{2}=2^{3}+2^{0}+2^{-1}+2^{-3}=8+1+0.5+0.125=9.625_{10}$

- V N bitoch teda môžeme zobraziť čísla s pevnou čiarkou tak, že prvých K cifier zľava je „pred rádovou čiarkou“ a zvyšok je „za ňou“ (K<=N).
- Napríklad N=16, K=8



- Takéto čísla sa sčitujú a odčitujú rovnako ako celé čísla, ale násobenie a delenie je iné
- Majú malý rozsah zobraziteľných hodnôt, preto sú nevhodné na vedecké výpočty
- Majú menej problémov so zaokrúhľovaním, preto sa niekedy používajú na výpočty s peniazmi, kde malý rozsah až tak nevadí

Reálne čísla
Zobrazenie s pevnou čiarkou (2)

Príklad 1

Zobrazte 2.375 v tvare čísla s pevnou čiarkou v 8 bitoch, kde 4 bity sú pred čiarkou a 4 za ňou.

$2.375_{10} = 2+0.25+0.125 = 2+1/4+1/8 = 2^{1}+2^{-2}+2^{-3}$

$= 10.011_{2}$, zapísané do 8 bitov: 00100110

Príklad 2

Aké číslo s pevnou čiarkou je zobrazené v 8 bitovom zápise 01011010 s 3 bitmi pred rádovou čiarkou a 5 bitmi za ňou?

01011010 $10.1101_{2}=2^{1}+2^{-1}+2^{-2}+2^{-4}=2+1/2+1/4+1/16=$

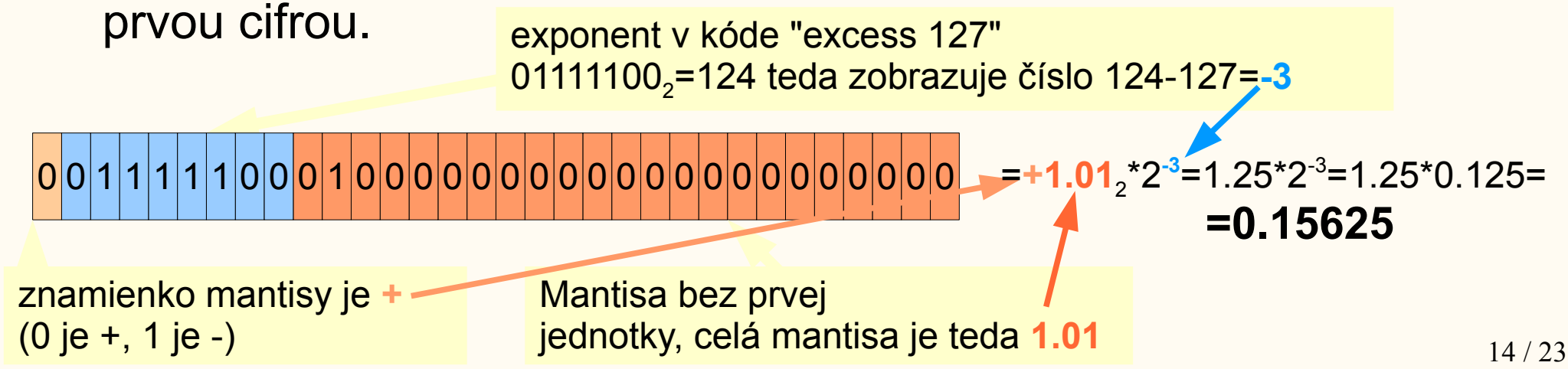
$2+0.5+0.25+0.0625=\mathbf{2.8125}_{10}$

Reálne čísla
Zobrazenie s pohyblivou čiarkou

- Hmotnosť slnka je asi $2*10^{33}$g, hmotnosť elektrónu je asi $9*10^{-28}$g. Rozdiel medzi nimi je 61 desiatkových rádov, čo je asi 202 dvojkových rádov.
- Teda na fyzikálny výpočet s oboma týmito číslami v pevnej rádovej čiarke by sme potrebovali aspoň 202 bitové čísla. Pritom ale takú veľkú presnosť vôbec nevyužijeme lebo obe čísla poznáme s presnosťou ledva 5 cifier.
- Preto sa pri vedeckých výpočtoch používa zápis čísla v pohyblivej rádovej čiarke, nazývaný tiež **semilogaritmický tvar** alebo **vedecká notácia**.
- Kalkulačky používajú takýto zápis so základom 10. Teda 2308 zapíšu ako 2.308E3 čo znamená $2.308*10^3$
- 2308 sa ale dá zapísať aj ako $23.08*10^2$ alebo $0.002308*10^6$. Prednosť má ale zápis $2.308*10^3$, ktorý má pred desatinnou bodkou práve jednu nenulovú cifru. Nazývame to **normalizovaný tvar** čísla.
- V počítačoch sa používa základ 2, teda čísla sa zapisujú v tvare $a*2^b$, teda treba uložiť dve čísla **a** a **b**, obe môžu byť kladné aj záporné.

Reálne čísla
Zobrazenie s pohyblivou čiarkou (2)

- Norma IEEE 754 - najrozšírenejší tvar čísla v pohyblivej rádovej čiarke. Procesory Intel používajú tento tvar (ale pridávajú si ďalší vlastný typ).
- Normalizované číslo má pred rádovou čiarkou nenulovú cifru. Ale v dvojkovej sústave **jediná nenulová cifra je 1**. Preto si ju nemusíme pamätať a ušetríme jeden bit.
- **Binary32** (single precision): 1 bit znamienko mantisy, 8 bitov exponent v kóde "excess 127" a 23 bitov absolútna hodnota mantisy bez prvej jednotky, teda s pevnou rádovou čiarkou pred prvou cifrou.



Reálne čísla
Zobrazenie s pohyblivou čiarkou (3)

Ďalšie typy podľa IEEE-754:

- **Binary16** (half precision): 1 bit znamienko mantisy, 5 bitov exponent v kóde "excess 15" a 10 bitov absolútna hodnota mantisy bez prvej jednotky, teda s pevnou rádovou čiarkou pred prvou cifrou. Tento formát je určený len na ukladanie do pamäti, nie na vykonávanie výpočtov.
- **Binary64** (double precision): 1 bit znamienko mantisy, 11 bitov exponent v kóde "excess 1023" a 52 bitov absolútna hodnota mantisy bez prvej jednotky.
- **Binary128** (quadruple precision): 1 bit znamienko mantisy, 15 bitov exponent v kóde "excess 16383" a 112 bitov absolútna hodnota mantisy bez prvej jednotky.

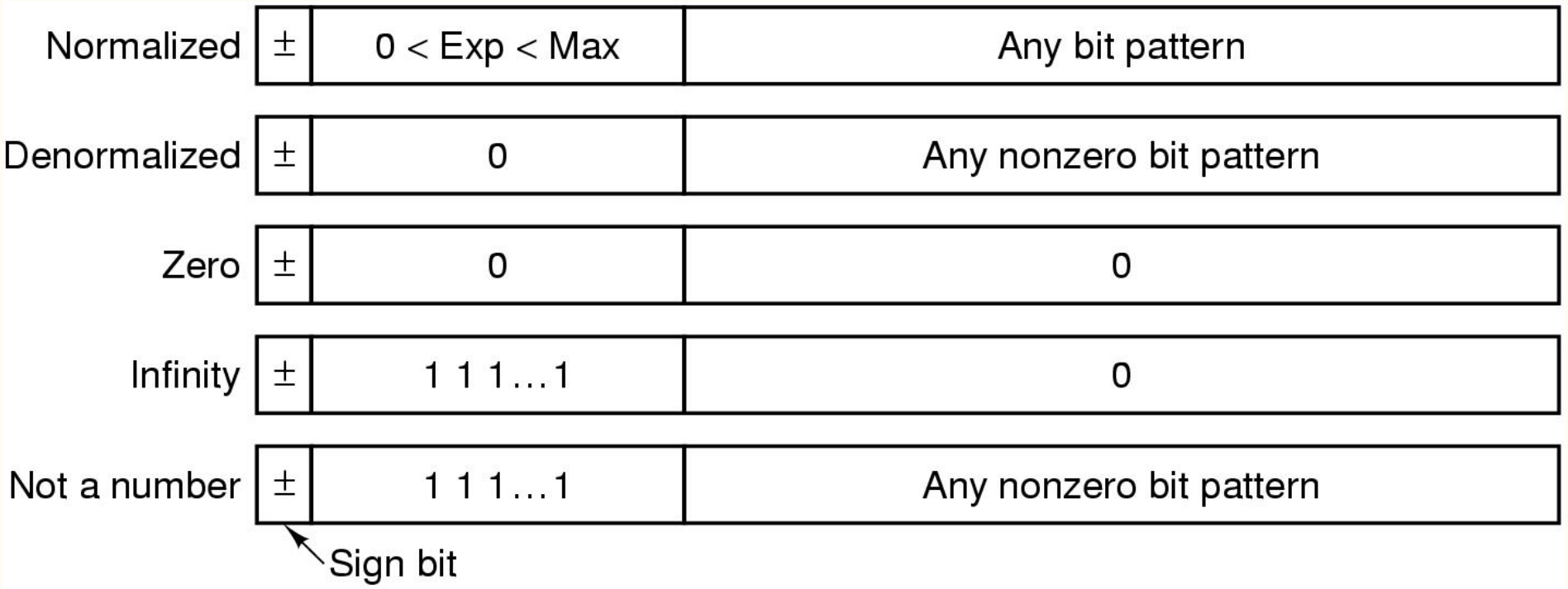
Reálne čísla
Zobrazenie s pohyblivou čiarkou (4)

- Najmenšia a najväčšia hodnota exponentu sa nepoužíva pre normalizované čísla. Napr. v binary32 môže byť exponent len od -126 po +127, hodnoty exponentu -127 (kód 0) a +128 (kód FF) sú rezervované pre špeciálne čísla:
- **Denormalizované číslo** má exponent -127 (kód 0) a k mantise nie je pridávaná prvá jednotka - **tak možno zobraziť čísla blízko nuly** až do 2^{-149} za cenu znižujúcej sa presnosti
- **Nula** má exponent -127 (kód 0) a mantisu 0, znamienko môže byť + aj -
- **Plus nekonečno** má znamienko +, exponent 128 (kód FF), mantisu 0
- **Mínus nekonečno** má znamienko -, exponent 128 (kód FF) a mantisu 0
- **Not a Number (NaN)** má znamienko ľubovoľné, exponent 128 (kód FF) a mantisu nenulovú
- špeciálne hodnoty sa používajú aj pri výpočtoch, napríklad 1/0 je plus nekonečno, nekonečno deleno nekonečno je NaN a pod.

(exponenty -127 a 128 platia pre binary32, pre binary64 sú -1023 a 1024, pre binary16 sú -15 a 16 a pre binary128 sú -16383 a 16384)

Reálne čísla
Zobrazenie s pohyblivou čiarkou (4)

Zhrnutie špeciálnych formátov v obrázku



Ďalšie informácie nájdete napr. na
http://en.wikipedia.org/wiki/IEEE_floating_point

Rozsahy čísiel podľa IEEE-754

Typ	Rozsah	presnosť (des. cif.)	bajtov
`binary16`	$6.1 \times 10^{-5} .. 6.5 \times 10^{4}$	3-4	2
`binary32`	$1.5 \times 10^{-45} .. 3.4 \times 10^{38}$	7-8	4
`binary64`	$5.0 \times 10^{-324} .. 1.7 \times 10^{308}$	15-16	8
`binary128`	$6.4 \times 10^{-4966} .. 1.2 \times 10^{4932}$	34	16

Čísla v Pythone

```
import sys

def tryInteger(n):
    x = 1
    print(sys.getsizeof(x))
    for i in range(n):
        x = x*10
        print(i,x)
    print(sys.getsizeof(x))

def tryReal(n):
    x = 1.0
    print(sys.getsizeof(x))
    for i in range(n):
        x = x*10.0
        print(i,x)
    print(sys.getsizeof(x))
```

- Nájdite rozdiel medzi funkciami **`tryInteger`** a **`tryReal`**.
- Vysvetlite nájdený rozdiel.
- Skúste:
  `tryInteger(400)`
  `tryReal(400)`
- Na základe pokusu odpovedzte na otázky:
  - Ako je v Pythone uložené celé číslo?
  - Ktorý tvar reálneho čísla podľa IEEE-754 zrejme používa Python pre reálne čísla?

Celočíselné typy v C

- Keď nezáleží na presnom zobrazení čísla, tak používame typy:
  - **`int`**, ak chceme číslo so znamienkom alebo
  - **`unsigned int`**, ak chceme číslo bez znamienka
- Ak záleží na presnom zobrazení, tak sú typy podľa normy C99:
  int8_t, **int16_t**, **int32_t**, **int64_t**, **uint8_t**, **uint16_t**, **uint32_t**, **uint64_t**
- Tabuľka uvádza aj identifikátory pre 32-bitové programy v MS Visual C++

Bitov	Znamienko	Typ v C99	Typ v 32-bitovom programevo Visual C++	Minimum	Maximum
8	áno	**`int8_t`**	**`char`** **`signed char`**	-128	127
8	nie	**`uint8_t`**	**`unsigned char`**	0	255
16	áno	**`int16_t`**	**`short`** **`signed short`**	-32 768	32 767
16	nie	**`uint16_t`**	**`unsigned short`**	0	65 535
32	áno	**`int32_t`**	**`long`** **`signed long`**	-2 147 483 648	2 147 483 647
32	nie	**`uint32_t`**	**`unsigned long`**	0	4 294 967 297
64	áno	**`int64_t`**	**`long long`** **`signed long long`**	-9 223 372 036 854 775 808	9 223 372 036 854 775 807
64	nie	**`uint64_t`**	**`unsigned long long`**	0	18 446 744 073 709 551 615

Ďalšie informácie napríklad na: https://en.cppreference.com/w/c/types/integer

Celočíselné typy v C (2)

Čo vypíše tento program?

```
int main()
{
   int n, i;
   n = 1;
   for (i = 0; i < 40; i++) {
      n *= 2;
      printf("%d %u %x\n",n,n,n);
   }
}
```

tento printf vypisuje premennú **n** v desiatkovej sústave so znamienkom (formát **d**), bez znamienka (formát **u**) a v šestnástkovej sústave (formát **x**).

Čo sa stane keď zmeníme typ premennej z **int** na **int8_t**? (Vtedy je rozumné zmeniť aj formáty za hhd, hhu a hhx, o formátoch viď napr. http://www.cplusplus.com/reference/cstdio/printf/)

Celočíselné typy v C (3)

- Pozor na použitie neznamienkových čísiel v cykloch typu **while**:

```
uint8_t i;
i = 10;
while (i >= 0) {
     printf("%d\n",i);
     --i;
}
```

Zacyklí sa lebo pre ľubovoľné **i** typu **uint8_t** platí i>=0

- Pozor aj na použitie malých čísiel v cykloch keď hranica cyklu je zároveň hraničná hodnota pre daný typ:

```
int8_t i;
i = 0;
while (i <= 127) {
     printf("%d\n",i);
     ++i;
}
```

Čo bude robiť tento program?

- V C nám nepomôže ani cyklus **for**, lebo ten je len skratkou pre **while**:

```
int8_t i;
for (i = 120; i <= 127; i++)
     printf("%d\n",i);
```

Reálne typy v C

- FPU (floating point unit - aritmeticko-logická jednotka pre pohyblivú rádovú čiarku) procesorov Pentium ponúka reálne čísla **single precision**, **double precision** a **extended precision**. Single a double sú definované ako v norme IEEE 754, extended precision je špeciálny typ procesorov Intel - nepoužíva rozšírenie o jeden bit mantisy, teda ukladá aj prvú jednotku mantisy.

Typ	Rozsah	presnosť (des. cifier)	bajtov	bitov
`float`	1.5×10^{-45} .. 3.4×10^{38}	7-8	4	32
`double`	5.0×10^{-324} .. 1.7×10^{308}	15-16	8	64
`long double`	$3.6 \times 10^{-4951(-4966)}$ .. 1.1×10^{4932}	19-20 (33-36)	10(16)	80(128)

- Typ **long double** môže byť implementovaný pomocou typu extended precision v CPU Intel alebo ako **binary128** podľa IEEE 754 (potom platia údaje v zátvorkách). Ak nie je možná žiadna implementácia presnejšia než **double**, tak je implementovaný rovnako ako **double**.