# PJC
# Cvičení #2

# Číselné soustavy
# a binární reprezentace proměnných

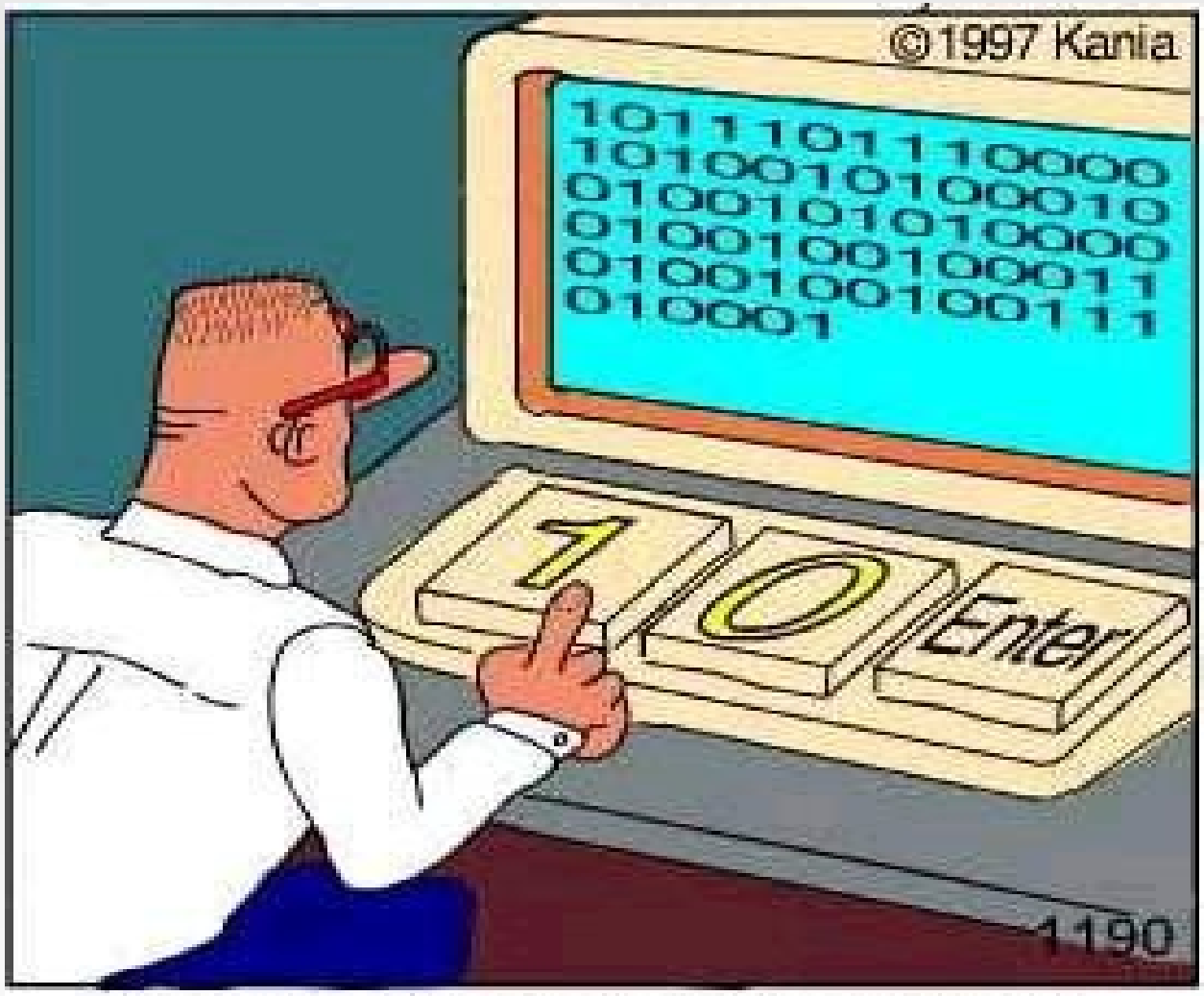
# Číselné soustavy

- **Desítková** (decimální) – kdo nezná, tak ...!!!
- **Dvojková** (binární) - nejjednodušší
- **Šestnáctková** (hexadecimální) - nejpoužívanější po desítkové
- **Osmičková** (oktalová) – přístupová práva v Unixu
- **Dvanáctková** – tucet a veletucet, nepoužívá se
- **Šedesátková** – kopa a velekopa, řády jsou odděleny dvojtečkou, používá cifry desítkové soustavy

# Binární soustava

- 0,1
- Řády binární soustavy:
  - … $2^3$ + $2^2$ + $2^1$ + $2^0$
  - … 8 + 4 + 2 + 1
- Ukázka:
  - … 1 0 1 1
  - … 8 + 4 + 0 + 1 = 11

# Kdyby byla jen binární soustava

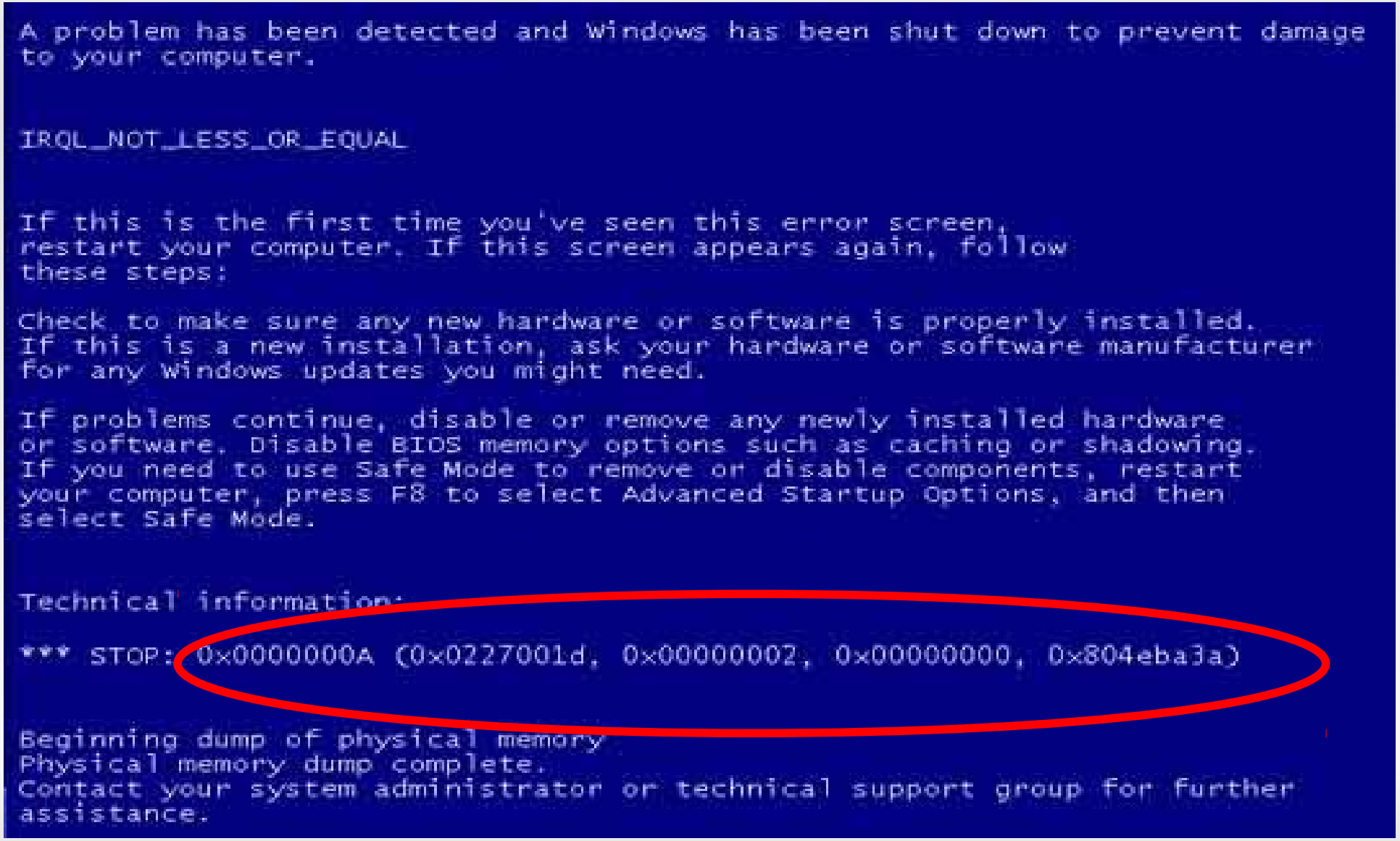



Real programmers code in binary.

# Hexadecimální soustava

- Základem soustavy 16 jednotek
- 0-9, A, B, C, D, E, F
- Úsporný zápis binárních hodnot
- Převod do binární soustavy a naopak je čistě mechanický, proveditelný bez výpočtu
- Používá se hodně v operačních systémech pro zobrazení binárních dat (adresy v paměti, hexaeditor)

# Tohle asi všichni známe...



# Desítková → dvojková

- **Metoda dělení základem**

| Dělení | Podíl | Zbytek |
|---|---|---|
| $(109)_{10}/2$ | 54 | 1 pravá pozice |
| $(54)_{10}/2$ | 27 | 0 |
| $(27)_{10}/2$ | 13 | 1 |
| $(13)_{10}/2$ | 6 | 1 |
| $(6)_{10}/2$ | 3 | 0 |
| $(3)_{10}/2$ | 1 | 1 |
| $(1)_{10}/2$ | 0 | 1 levá pozice |

- Metoda je univerzální, lze ji použít pro převod desítkové soustavy na jakoukoliv jinou

# Dvojková ↔ šestnáctková

- Tohle jde opravdu z hlavy:

$$(1001\ 0010\ 1101\ 1110\ 0101\ 0111)_2$$

---

$$(\quad 9 \quad 2 \quad D \quad E \quad 5 \quad 7)_{16}$$

# Kódování čísel

- Co znamená tento byte?

  $(1011\ 0010)_2$

- Všechny následující odpovědi jsou správné:
  - 178 (binární kód)
  - -50 (přímý kód)
  - -77 (inverzní kód)
  - 50 (aditivní kód)
  - -78 (doplňkový kód)

# Přímý kód

- Prakticky stejný jako binární
- První bit určuje znaménko
- Obsahuje tzv. zápornou a kladnou nulu

  00001000 → 8

  10001000 → -8

  00000000 → 0

  10000000 → -0
- Rozsah od -127 do 127
- Špatně se implementuje aritmetický hardware

# Inverzní kód

- Kladná čísla jsou stejná jako v binárním
- Záporná čísla se tvoří bitovou negací
- Kód obsahuje opět dvě reprezentace nuly
- Kód je snadno pochopitelný
- Opět špatně implementovatelný v hardwaru
- Rozsah také od -127 do 127, takže opět využívá 254 hodnot z 255 možných
- V době 8-bitových procesorů to bylo plýtvání ;-)

# Aditivní kód

- Opět vychází z binárního kódu
- Dokáže využít rozsah hodnot beze zbytku
- Rozsah od -128 do 127
- K binárnímu kódu se přičte známá konstanta

  00000000 → -128

  10000000 → 0

  11111111 → 127
- Sčítání se implementuje snadno ale násobení je problematické
- Používá se pro exponent reálných čísel

# Doplňkový kód I.

- V současnosti nejpoužívanější
- Je podobný inverznímu kódu
- Narozdíl od inverzního kódu využívá rozsah bytu naplno od -128 do 127
- Záporné číslo je negací kladného čísla zvětšeného o jedničku
- Jednoduchá implementace aritmetických operací
- Používá se pro reprezentaci celých čísel v jazycích C/C++

# Doplňkový kód II.

- Převod čísla $(-78)_{10}$ do doplňkového kódu:
  - Začneme s kladným číslem $(78)_{10}$

    $(78)_{10} \rightarrow (0100\ 1110)_2$
  - Provedeme bitovou negaci

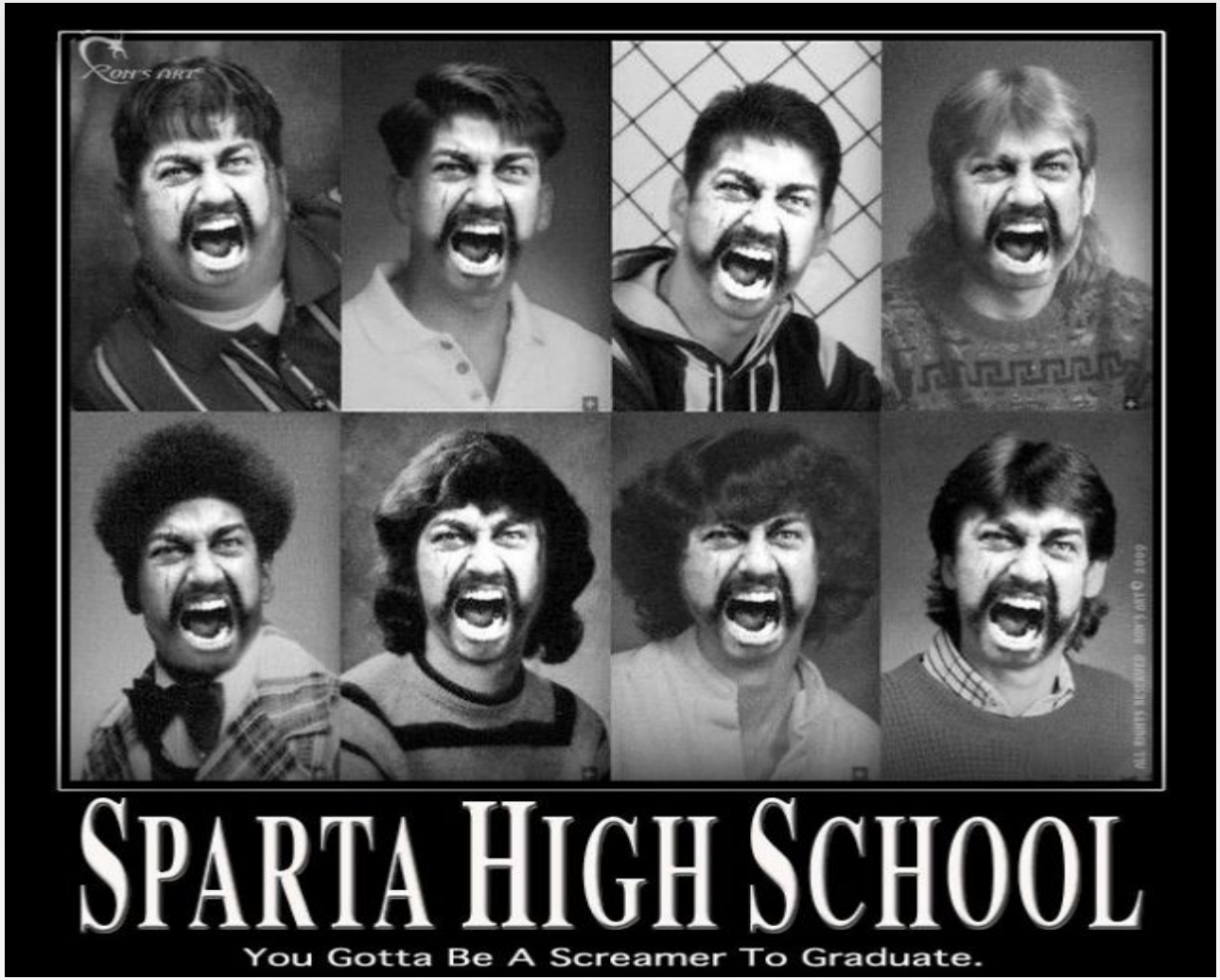
    $neg[(0100\ 1110)_2] \rightarrow (1011\ 0001)_2$
  - Nakonec přičteme jedničku

    $(1011\ 0001)_2 + (0000\ 0001)_2 \rightarrow (1011\ 0010)_2$
- Převod z doplňkového kódu do desítkové soustavy je zřejmý

# Přetečení rozsahu

- Situace, kdy překročíme rozsah čísla daný velikostí čísla v bitech a použitým kódováním
- Např. inkrementujeme-li číslo 127 v aditivním kódu pro 8 bitů:
  - $(0111\ 1111) + (0000\ 0001) \rightarrow (1000\ 0000)_2 = (-128)_{10}$

# Ještě nemáte dost?




- Tak se kouknem na reálná čísla...

# Reprezentace reálných čísel I.

- Floating point
- Číslo 435,5 se dá vyjádřit ve tvaru

  $4{,}355 \times 10^2$

  - 4,355 se nazývá mantisa
  - 2 se nazývá exponent
- Bity čísla se rozdělí na mantisu a exponent
- Podíl bitů stanovují normy a výrobci hardwaru
- Počet bitů mantisy určuje přesnost čísla
- Počet bitů exponentu určuje rozsah

# Reprezentace reálných čísel II.

- Ztráta přesnosti znamená, že mantisa reprezentovaného čísla se nevejde do bitového rozsahu pro mantisu
- Skutečná hodnota reprezentovaných reálných čísel se často na různých hardwarových platformách liší
- Proč myslíte, že se stále sleduje výkon procesorů ve floating point operacích?

# Little endian vs big endian I.

- Problém vícebytových reprezentací čísel
- Rozdíl v pořadí uložených bytů v paměti
- Mějme číslo $(AA\ BB\ CC\ DD)_{16}$
  - Little endian uložení v paměti:

    nižší adresa → DD CC BB AA → vyšší adresa
  - Big endian uložení v paměti:

    nižší adresa → AA BB CC DD → vyšší adresa
- Ani jedna reprezentace není horší nebo lepší než druhá

# Little endian vs big endian II.

- Například PC procesory používají little endian, zatímco stanice SPARC (OS Solaris) používají big endian
- Při programování v jazyce C/C++ nás endianita zajímá pouze v případě bitových posuvů
- Endianitu lze velmi snadno identifikovat přímo za chodu programu

# Použité zdroje

- Číselné soustavy
- Převody číselných soustav
- Reprezentace floating point čísel
- Endianita