B4B33PSY: Seminar 5

Bc. Štěpán Pressl

Fungování dvojkového doplněk (two's complement)

- **POZOR**: dvojkový doplněk je pouze kódování čísla!
- **PŘIPOMENUTÍ**: dvojkový doplněk dělí interval na (skoro) symetrickou zápornou a kladnou část. Platí, že interval všech hodnot je $[-2^{n-1}, 2^{n-1} - 1]$.
- **PŘIPOMENUTÍ**: tímto způsobem lze odčítat čísla pouze jejich sčítáním.
- Dokážeme si dvě vlastnosti:
  1. $2 \times$ aritmetické negování čísla (ne bitové) je idempotentní,
  2. odčítání čísel v dvojkovém doplňku korektně reprezentuje rozdíl čísel.

Co znamená negace čísla?

Bitová negace znamená číslo odečíst od samých 1. Samé 1 (pro n bitů) je $2^n - 1$. Negování tedy je

$$\overline{x} = 2^n - 1 - x$$

Aritmetické znegování čísla v dvojkovém doplňku tedy je

$$x^2 = 2^n - 1 - x + 1 = 2^n - x$$

Vlastnost 1

Vemte jakékoli číslo v dvojkovém doplňku (označíme x).

Důkaz:

1. aritmetická negace

$$y = 2^n - x$$

2. aritmetická negace

$$2^n - y = 2^n - 2^n + x = x$$

Vlastnost 2

Pamatujte si, že numerická hodnota n-bitového čísla musí operovat v aritmetice mod 2^n. Musíme ukázat, že výsledek čísla $x + y^2$ bude odpovídat výsledku $x - y$.

Důkaz:

$$x + y^2 \pmod{2^n} \equiv x - y + 2^n \pmod{2^n} \equiv x - y \pmod{2^n}.$$

V modulo aritmetice toto produkuje správný výsledek. Ještě si ukážeme (na úrovni bitů), co se stane při sčítání, když $|x| > |y|$ a $|x| < |y|$:

1. $|x| > |y|$: rozdíl $x - y > 0$, tedy $x + y^2$ (numericky) $> 2^n$ (proto ten zázračný "carry", který ale není overflow) - v dvojkovém doplňku je to také kladné číslo.
2. $|x| < |y|$: rozdíl $x - y < 0$, tedy $x + y^2$ (numericky) $\in [2^{n-1}, 2^n]$ - v dvojkovém doplňku je to záporné číslo

Ad overflow k dvojkovému doplňku

Jak nastane overflow?

- logicky nemůže nastat overflow při sčítání dvou čísel různých znamének
  - Při příkladu $x - y$ bylo vidět, že musí dojít k přenosu do dalšího řádu, když $|x| > |y|$. To ale není overflow.
- nastane tehdy, když sčítáte dvě čísla stejného znaménka a znaménko je jiné (carry-in na posledním bitu je jiné než carry-out na posledním bitu)

Převody čísel

Převeďte čísla:

1. -57, uvažujte, že výsledek se uloží do `uint8_t` a do `int8_t`, jak to bude interpretované?
2. To stejné v 1., akorát pro `uint16_t` a `int16_t`.
3. Převeďte `0b10101010` do desítkové soustavy, uvažujte že číslo je typu `uint8_t` a `int8_t`.

Operátory v jazyce C

- Logické: `&&`, `||`, `!`,
- Aritmetické: `+`, `-`, `/`, `*`, `%`, `<`, …
- Bitové: `&`, `|`, `^`, `~`, `<<`, `>>`

1. Procvičení C

Co se vytiskne?

```c
unsigned long a = 0x12AB & 0x2448;
printf("%x\n", a);
```

2. Procvičení C (bonus bod)

Co se vytiskne?

```c
uint8_t a = 0x81;
int8_t b = (int8_t) a;
int8_t c = (int8_t) (a & 0x7F);
printf("%u %d %d\n", a, b, c);
```

Bitové posuny

- `(a << x)` posune číslo o x pozic doleva,
- `(a >> x)` posune číslo o x pozic doprava
  - pozor na typ posouvaného čísla, jestliže je číslo znaménkové, je provedeno znaménkové rozšíření (tzv. sign extension)

Maskování v C

Jestli si přejete získat informaci o tom, zda-li je bit nastaven, můžete použít masku:

```c
// je nastaven xty bit?
if (a & (1 << x)) {
  // ano
}
```

Jestli si přejete x-tý bit nastavit na 0

```c
x = x & ~(1 << x);
x &= ~(1 << x); // alternativne
```

3. Procvičení C (bonus bod)

Co se vytiskne?

1.
```c
uint8_t a = 128 | (1 << 7) | (1 << 6) |
                 (128 >> 1);
printf("%x\n", a);
uint8_t b = 0xFE;
b = b & ~0xBB;
printf("%x\n", b);
```

4. Procvičení C

Napište program v C, který vypíše jednotlivé bity neznaménkového 32bitového čísla. Pokuste se upravit program, aby vypisoval bity 32bitového znaménkového čísla.

Nahlížení do paměti

- **UPOZORNĚNÍ**: hodnoty jsou uloženy na konkrétních adresách v počítači
- Pro zkoumání paměti můžete použít code snippet uvedený v cvičení 4 na psy.pages.fel.cvut.cz.
- Vyzkoušejte si různé datové typy (signed a unsigned čísla)
- Různé počítačové architektury uchovávájí čísla jinak v paměti
- Little endian (little end first): LSB (least significant byte) je uložen na nejnižší hodnotě adresy
- Big endian (big end first): MSB (most significant byte) je uložen na nejvyšší hodnotě adresy

5. procvičení (endianita)

Na adrese 0x4000 máte uloženou hodnotu 0x1234ABCD.

1. (sanity check) Kolika bitové je to číslo?
2. Co bude na adresách 0x4000-0x4003 na architektuře s big endian rozložením?
2. Co bude na adresách 0x4000-0x4003 na architektuře s little endian rozložením?

Reprezentace IEEE754

- Reprezentace pomocí vědeckého zápisu
- Znaménkový bit, mantissa, exponent
- NaN, $\pm\infty$
- Pro 32 bitový `float` platí: 1 bit znaménko (msb) poté 8 bitů exponent poté 23 bitů mantissa
- Pro double precision (`double`) reálné číslo platí: 1 bit znaménko (msb) poté 11 bitů exponent, poté 54 bitů mantissa

Obecný zápis IEEE754 čísla

$$\text{NORMALIZED} = (-1)^{\text{sign}} \cdot 2^{\text{exp}\,-127} \cdot (1 + m_1 2^{-1} + m_2 2^{-2} + \ldots)$$

$$\text{DENORMALIZED} = (-1)^{\text{sign}} \cdot 2^{-127} \cdot (1 + m_1 2^{-1} + m_2 2^{-2} + \ldots) =$$

$$(-1)^{\text{sign}} \cdot 2^{-126} \cdot (m_1 2^{-1} + m_2 2^{-2} + \ldots).$$

Vlastnosti IEEE754 a důsledky (bonus bod)

1. Proč se posouvá exponent?
2. Proč se mantissa normalizuje?

6. Převod čísel IEEE754

Na 32 bitů.

1. -0.75
2. 0.375
3. -32.4375