Sčítání pro úplné začátečníky

Přenosy

- Vzpomeňme na písemné sčítání, které známe ze základní školy

```
   6 8 9
+  3 4 3
-------
```

Přenosy

- Vzpomeňme na písemné sčítání, které známe ze základní školy

```
   6 8 9
+  3 4 3
     1
-------
       2
```

Přenosy

- Vzpomeňme na písemné sčítání, které známe ze základní školy

```
   6 8 9
+  3 4 3
   1 1
---------
     3 2
```

Přenosy

- Vzpomeňme na písemné sčítání, které známe ze základní školy

```
   6 8 9
+  3 4 3
   1 1
---------
1 0 3 2
```

Přenosy

- Vzpomeňme na písemné sčítání, které známe ze základní školy

```
   6 8 9
+  3 4 3
   1 1
---------
1 0 3 2
```

- Co se děje, když po sečtení dvě čísla jednoho řádu překročí desítku?
  - Přenesou se do dalšího řádu!

Přenosy

- Vzpomeňme na písemné sčítání, které známe ze základní školy

```
          6 8 9
        + 3 4 3
Přenosy:  1 1
        -------
        1 0 3 2
```

- Co se děje, když po sečtení dvě čísla jednoho řádu překročí desítku?
  - Přenesou se do dalšího řádu!

Přenosy

- Vzpomeňme na písemné sčítání, které známe ze základní školy

```
          6 8 9
        + 3 4 3
Přenosy:  1 1
        -------
        1|0 3 2
```

Sčítali jsme na *třech* číslicích, ale na výsledek potřebujeme *čtyři* – něco nám „*vyteklo*“ ven

- Co se děje, když po sečtení dvě čísla jednoho řádu překročí desítku?
  - Přenesou se do dalšího řádu!

Přenosy

- Vzpomeňme na písemné sčítání, které známe ze základní školy

```
      6 8 9
    + 3 4 3
Přenosy:
      1 1
    -------
    1|0 3 2
```

Sčítali jsme na *třech* číslicích, ale na výsledek potřebujeme *čtyři* – něco nám „*vyteklo*“ ven

- Co se děje, když po sečtení dvě čísla jednoho řádu překročí desítku?
  - Přenesou se do dalšího řádu!
  - 1 + 1 = 2
  - 5 + 4 = 9
  - 5 + 8 = 13 = TROJKA a do vyššího řádu se přenáší JEDNA desítka

Přenosy

- Vzpomeňme na písemné sčítání, které známe ze základní školy

```
          6 8 9
        + 3 4 3
Přenosy:  1 1
        -------
        1|0 3 2
```

Sčítali jsme na *třech* číslicích, ale na výsledek potřebujeme *čtyři* – něco nám „*vyteklo*“ ven

- Co se děje, když po sečtení dvě čísla jednoho řádu překročí desítku?
  - Přenesou se do dalšího řádu!
  - 1 + 1 = 2
  - 5 + 4 = 9
  - 5 + 8 = 13 = TROJKA a do vyššího řádu se přenáší JEDNA desítka
- A stejně to je v binárce…

Přenosy

- V dekadické soustavě generuje přenos, když jdeme přes devítku
  - 5 + 8 = 13
- V binární soustavě ale máme jen dvě číslice – přenos se bude generovat, když půjdeme přes jedničku:
  - 0 + 0 = **0**
  - 0 + 1 = **1** = 1 + 0
  - 1 + 1 = **10** – sečtením dvou jedniček vzniká *přenos do vyššího řádu*

Přenosy

- V dekadické soustavě generuje přenos, když jdeme přes devítku
  - 5 + 8 = 13
- V binární soustavě ale máme jen dvě číslice – přenos se bude generovat, když půjdeme přes jedničku:
  - 0 + 0 = **0**
  - 0 + 1 = **1** = 1 + 0
  - 1 + 1 = **10** – sečtením dvou jedniček vzniká *přenos do vyššího řádu*
- Zkusme sčítat na osmi bitech:

```
  0 0 1 0 | 1 1 0 1
+ 1 1 0 1 | 0 1 1 0
-------------------
```

Přenosy

- V dekadické soustavě generuje přenos, když jdeme přes devítku
  - 5 + 8 = 13
- V binární soustavě ale máme jen dvě číslice – přenos se bude generovat, když půjdeme přes jedničku:
  - 0 + 0 = **0**
  - 0 + 1 = **1** = 1 + 0
  - 1 + 1 = **10** – sečtením dvou jedniček vzniká *přenos do vyššího řádu*
- Zkusme sčítat na osmi bitech:

```
  0 0 1 0 | 1 1 0 1
+ 1 1 0 1 | 0 1 1 0
-----------------
          1
```

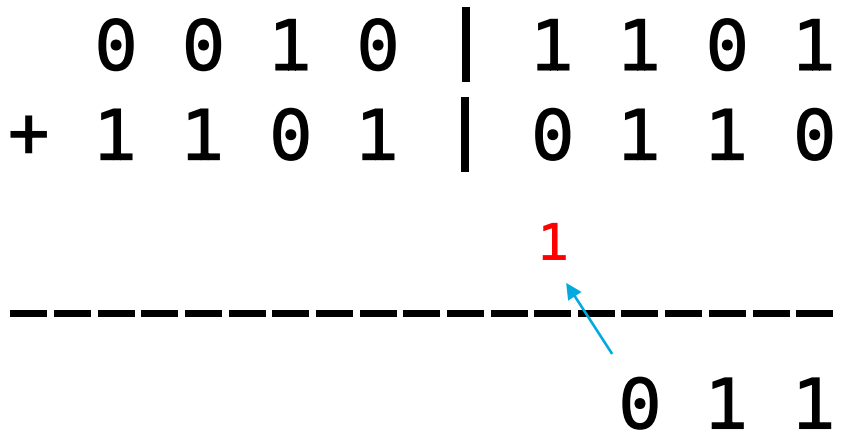
Přenosy

- V dekadické soustavě generuje přenos, když jdeme přes devítku
  - 5 + 8 = 13
- V binární soustavě ale máme jen dvě číslice – přenos se bude generovat, když půjdeme přes jedničku:
  - 0 + 0 = **0**
  - 0 + 1 = **1** = 1 + 0
  - 1 + 1 = **10** – sečtením dvou jedniček vzniká *přenos do vyššího řádu*
- Zkusme sčítat na osmi bitech:

```
  0 0 1 0 | 1 1 0 1
+ 1 1 0 1 | 0 1 1 0
-------------------
          | 1 1
```

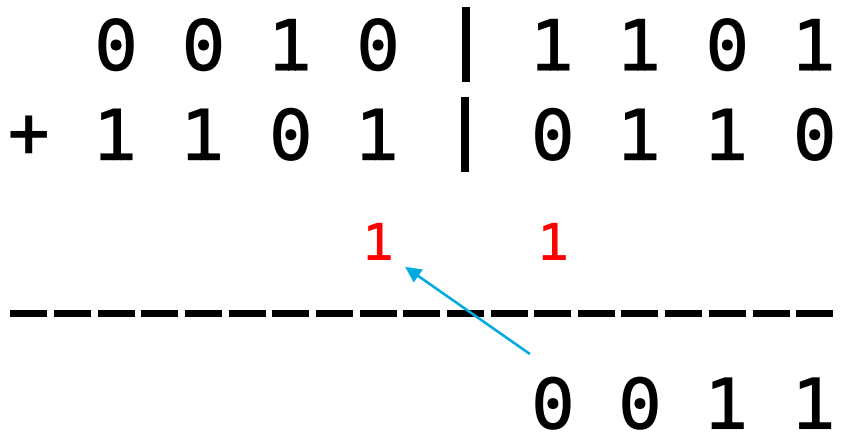
Přenosy

- V dekadické soustavě generuje přenos, když jdeme přes devítku
  - 5 + 8 = 13
- V binární soustavě ale máme jen dvě číslice – přenos se bude generovat, když půjdeme přes jedničku:
  - 0 + 0 = **0**
  - 0 + 1 = **1** = 1 + 0
  - 1 + 1 = **10** – sečtením dvou jedniček vzniká *přenos do vyššího řádu*
- Zkusme sčítat na osmi bitech:



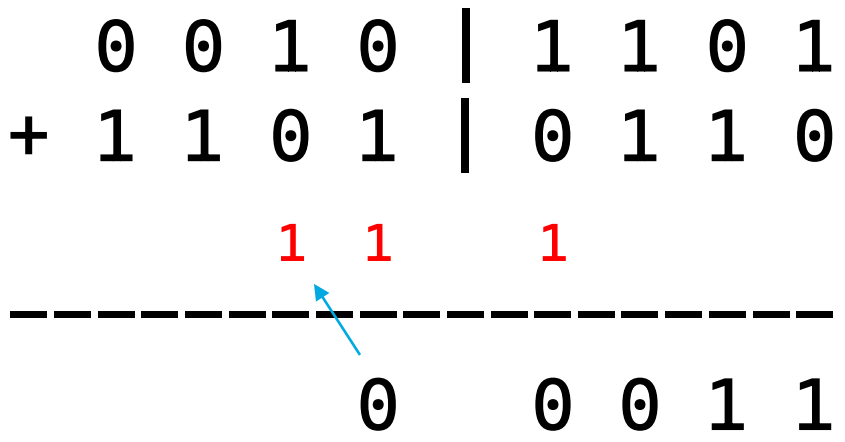
Přenosy

- V dekadické soustavě generuje přenos, když jdeme přes devítku
  - 5 + 8 = 13
- V binární soustavě ale máme jen dvě číslice – přenos se bude generovat, když půjdeme přes jedničku:
  - 0 + 0 = **0**
  - 0 + 1 = **1** = 1 + 0
  - 1 + 1 = **10** – sečtením dvou jedniček vzniká *přenos do vyššího řádu*
- Zkusme sčítat na osmi bitech:



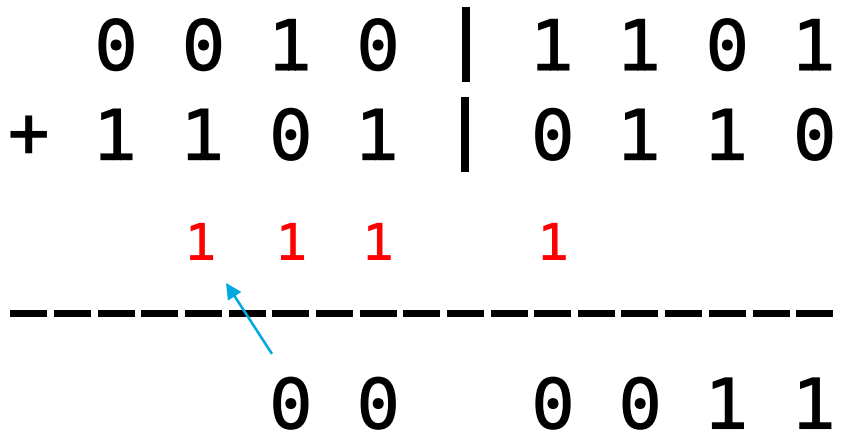
Přenosy

- V dekadické soustavě generuje přenos, když jdeme přes devítku
  - 5 + 8 = 13
- V binární soustavě ale máme jen dvě číslice – přenos se bude generovat, když půjdeme přes jedničku:
  - 0 + 0 = **0**
  - 0 + 1 = **1** = 1 + 0
  - 1 + 1 = **10** – sečtením dvou jedniček vzniká *přenos do vyššího řádu*
- Zkusme sčítat na osmi bitech:



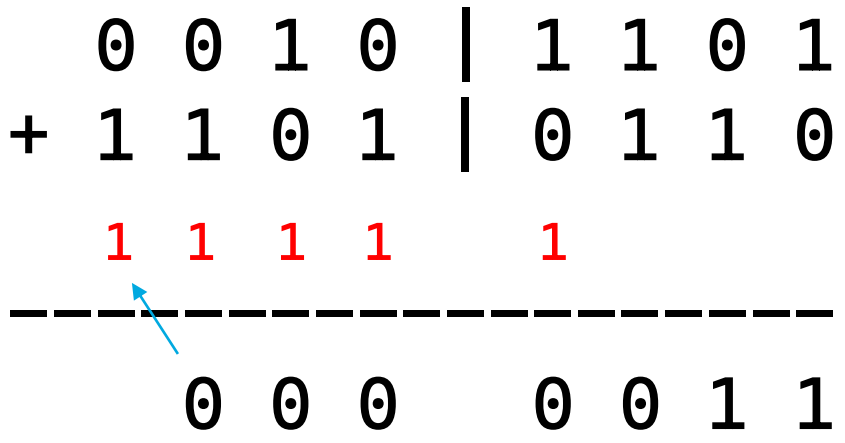
Přenosy

- V dekadické soustavě generuje přenos, když jdeme přes devítku
  - 5 + 8 = 13
- V binární soustavě ale máme jen dvě číslice – přenos se bude generovat, když půjdeme přes jedničku:
  - 0 + 0 = **0**
  - 0 + 1 = **1** = 1 + 0
  - 1 + 1 = **10** – sečtením dvou jedniček vzniká *přenos do vyššího řádu*
- Zkusme sčítat na osmi bitech:



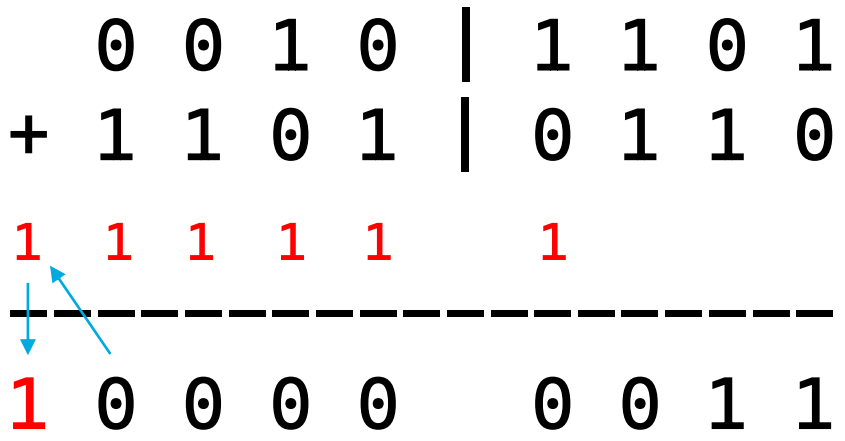
Přenosy

- V dekadické soustavě generuje přenos, když jdeme přes devítku
  - 5 + 8 = 13
- V binární soustavě ale máme jen dvě číslice – přenos se bude generovat, když půjdeme přes jedničku:
  - 0 + 0 = **0**
  - 0 + 1 = **1** = 1 + 0
  - 1 + 1 = **10** – sečtením dvou jedniček vzniká *přenos do vyššího řádu*
- Zkusme sčítat na osmi bitech:

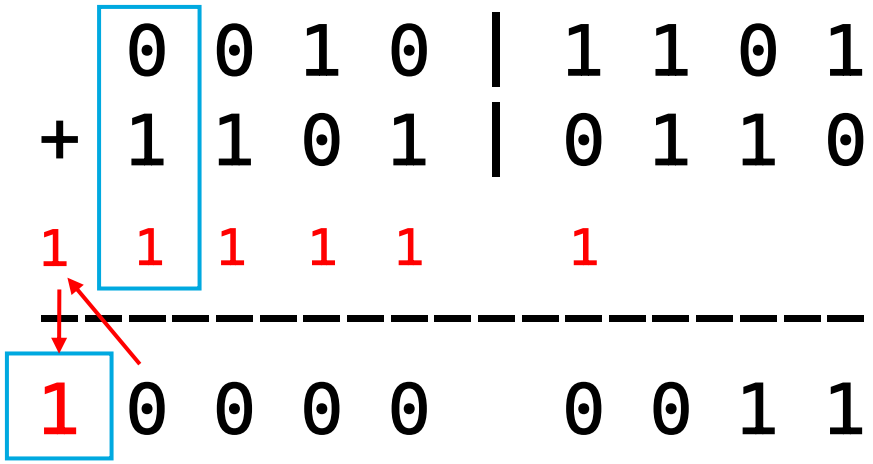


Přenosy

- V dekadické soustavě generuje přenos, když jdeme přes devítku
  - 5 + 8 = 13
- V binární soustavě ale máme jen dvě číslice – přenos se bude generovat, když půjdeme přes jedničku:
  - 0 + 0 = **0**
  - 0 + 1 = **1** = 1 + 0
  - 1 + 1 = **10** – sečtením dvou jedniček vzniká *přenos do vyššího řádu*
- Zkusme sčítat na osmi bitech:



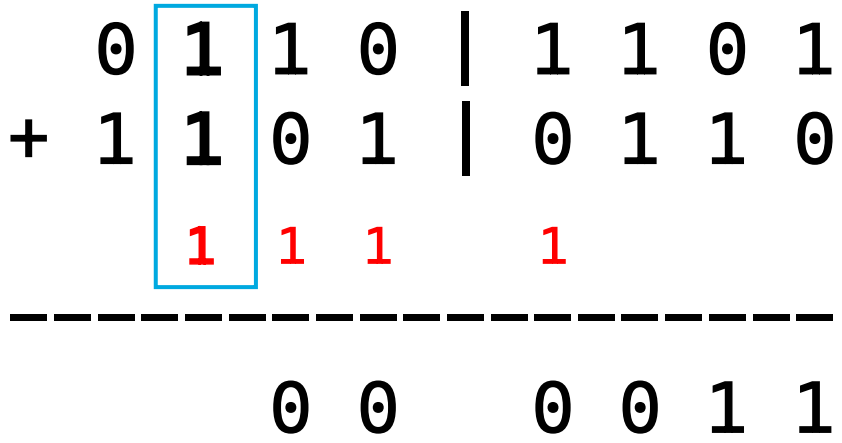
Carry



- Podobně jako v našem dekadickém příkladu, i zde se vyskytlo 1 + 1 v nejvyšším řádu, něco nám tedy „*vyteklo*“ ven z původně osmi bitů
- Tento bit označujeme jako **carry**
- Slovem „**přenos**“ při sčítání někdy myslíme právě tento jeden – poslední, *ven* přenesený bit
- Dobře si jej zapamatujme, za chvíli se k němu vrátíme

Závěrem

- Co když nad sebou máme tři jedničky?
  - Pozměněný příklad:



Závěrem

- Co když nad sebou máme tři jedničky?
  - Pozměněný příklad:

```
   0 1 1 0 | 1 1 0 1
+  1 1 0 1 | 0 1 1 0
     1 1 1   1
-------------------
       0 0   0 0 1 1
```

- 1 + 1 + 1 = 10 + 1 = 11
- Do vyššího řádu přeneseme jedničku, ve výsledku bude také jednička

Závěrem

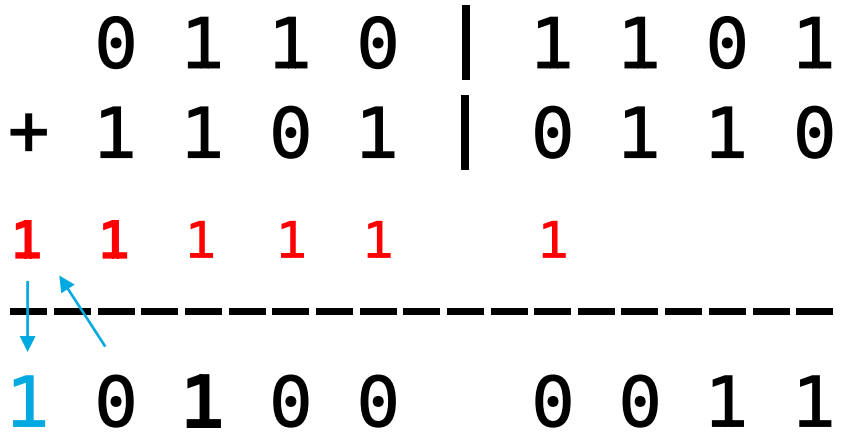
- Co když nad sebou máme tři jedničky?
  - Pozměněný příklad:

```
  0 1 1 0 | 1 1 0 1
+ 1 1 0 1 | 0 1 1 0
  1 1 1 1   1
-----------------
  1 0 0   0 0 1 1
```

- 1 + 1 + 1 = 10 + 1 = 11
- Do vyššího řádu přeneseme jedničku, ve výsledku bude také jednička

Závěrem

- Co když nad sebou máme tři jedničky?
  - Pozměněný příklad:



- 1 + 1 + 1 = 10 + 1 = 11
- Do vyššího řádu přeneseme jedničku, ve výsledku bude také jednička

Sčítání pro mírně pokročilé

Doplňkový kód, carry a overflow

Co je číslo?

- Běžně pracujeme s čísly kladnými, čísly zápornými a s nulou
- Jak je rozlišujeme?
  - Napíšeme před ně + nebo -
- Jak ale vidí *číslo* procesor?
  - Jako posloupnost binárních číslic
  - … + a – bohužel nejsou číslice
- Jak tedy pracovat v počítači se znaménky?
  - **Doplňkový kód**!

Doplňkový kód – letem světem

- Musíme si uvědomit, co znamená, že jde o **kód**:
  - „Je to způsob, jakým nějaký náš *problém* (znaménka čísel) *vyjádřit* ve světě, který jej sám o sobě vyjádřit nezvládá (binární číslo v počítači je prostě číslo bez znaménka).“
- Chceme být schopni s takto *zakódovanými čísly* provádět *operace cílového světa*
  - Sčítání binárních čísel, které jsme se naučili v předchozí sekci
- Chceme, aby po *dekódování* zpět do našeho světa (znaménkových čísel) byly výsledky správné
  - Takové, jaké by byly, kdybychom si ta původní čísla sečetli třeba na papíře
- **Doplňkový kód** je velmi elegantní způsob, jakým pracovat v počítači se znaménkovými čísly tak, aby „fungovala“, jak od nich čekáme
  - Když sečtu dvě záporná, vyjde záporné, když sečtu větší kladné s menším záporným, vyjde kladné a tak dále…
- V počítačích typicky pracujeme s pevně daným *počtem bitů*
  - Sčítáme tedy třeba „na 8 bitech“ – naše dva operandy i výsledek mají vždy 8 bitů, maximálně může jeden bit vytéct ven (carry), jak už víme

Doplňkový kód – letem světem

- Kladná čísla zapíšeme prostě v jejich binární podobě (s daným počtem bitů – na začátku budou nuly):

$$42 = 0\ 0\ 1\ 0\ \ 1\ 0\ 1\ 0$$

- Záporná čísla *zakódujeme* tak, že vezmeme kladný protějšek čísla, otočíme všechny jeho bity a přičteme jedničku:

$$-42 = 1\ 1\ 0\ 1\ \ 0\ 1\ 0\ 1$$

Doplňkový kód – letem světem

- Kladná čísla zapíšeme prostě v jejich binární podobě (s daným počtem bitů – na začátku budou nuly):

$$42 = 0\ 0\ 1\ 0\ \ 1\ 0\ 1\ 0$$

- Záporná čísla *zakódujeme* tak, že vezmeme kladný protějšek čísla, otočíme všechny jeho bity a přičteme jedničku:

$$\begin{aligned} -42 = 1\ 1\ 0\ 1\ \ 0\ 1\ 0\ 1 \\ +\ \ \ 1 \\ \hline 1\ 1\ 0\ 1\ \ 0\ 1\ 1\ 0 \end{aligned}$$

- Všimněme si, že bit úplně nalevo je v záporné variantě **1**
- V doplňkovém kódu podle bitu úplně nalevo (*„most significant bit“ – nejvýznamnější bit – na nejvyšší pozici)* vždy určíme znaménko
  - 0 = kladné, 1 = záporné
  - *Zkuste si tu -42 vyjádřit na 16 bitech*

Doplňkový kód – příklad

- A proč to děláme?
  - Protože v tom bez větších starostí funguje aritmetika tak, jak bychom čekali!
- Zkusme spočítat 9 **+** (-17):

```
  0 0 0 0  1 0 0 1
+ 1 1 1 0  1 1 1 1
-------------------
```

- Běžným způsobem (jak jsme si ukázali v první části prezentace) tato dvě binární čísla sečteme

Doplňkový kód – příklad

- A proč to děláme?
  - Protože v tom bez větších starostí funguje aritmetika tak, jak bychom čekali!
- Zkusme spočítat 9 **+** (-17):

```
  0 0 0 0  1 0 0 1
+ 1 1 1 0  1 1 1 1
-------------------
  1 1 1 1  1 0 0 0
```

Doplňkový kód – příklad

- A proč to děláme?
  - Protože v tom bez větších starostí funguje aritmetika tak, jak bychom čekali!
- Zkusme spočítat 9 **+** (-17):

```
  0 0 0 0  1 0 0 1
+ 1 1 1 0  1 1 1 1
--------------------
  1 1 1 1  1 0 0 0
```

- Na začátku je jednička, máme tedy nějaké *zakódované* záporné číslo
  - Jeho „kladného protějška“ zjistíme opět otočením bitů a přičtením 1

Doplňkový kód – příklad

- A proč to děláme?
  - Protože v tom bez větších starostí funguje aritmetika tak, jak bychom čekali!
- Zkusme spočítat 9 **+** (-17):

```
  0 0 0 0  1 0 0 1
+ 1 1 1 0  1 1 1 1
--------------------
  1 1 1 1  1 0 0 0
```

- Na začátku je jednička, máme tedy nějaké *zakódované* záporné číslo
  - Jeho „kladného protějška“ zjistíme opět otočením bitů a přičtením 1

```
0 0 0 0  0 1 1 1
                +1
0 0 0 0  1 0 0 0
```

Doplňkový kód – příklad

```
0 0 0 0  0 1 1 1
         + 1
0 0 0 0  1 0 0 0
```

- V dekadické soustavě má toto číslo hodnotu **8**
- My si ale pamatujeme, že to je jen nějaký náš mezikrok pro *dekódování záporného* čísla **1111 1000** – jeho „kladný protějšek“
- Toto binární číslo tedy kóduje hodnotu **-8**
- … což je 9 + (-17)

Co je číslo pro procesor?

- Je pěkné, že jsme si tak užitečně zakódovali kladná a záporná čísla, ale co to znamená pro procesor?
- *V podstatě* **nic** – procesor vždycky pracuje s binárními čísly, u kterých nerozlišuje nějaká znaménka – *to je přece ta podstata* ***kódování****!*
- Podívejme se znovu na náš příklad 9 + (-17):

Co je číslo pro procesor?

- Je pěkné, že jsme si tak užitečně zakódovali kladná a záporná čísla, ale co to znamená pro procesor?
- *V podstatě* **nic** – procesor vždycky pracuje s binárními čísly, u kterých nerozlišuje nějaká znaménka – *to je přece ta podstata* ***kódování****!*
- Podívejme se znovu na náš příklad 9 + (-17):

```
  0 0 0 0  1 0 0 1
+ 1 1 1 0  1 1 1 1
-------------------
  1 1 1 1  1 0 0 0
```

Co je číslo pro procesor?

- Je pěkné, že jsme si tak užitečně zakódovali kladná a záporná čísla, ale co to znamená pro procesor?
- *V podstatě* **nic** – procesor vždycky pracuje s binárními čísly, u kterých nerozlišuje nějaká znaménka – *to je přece ta podstata* ***kódování****!*
- Podívejme se znovu na náš příklad 9 + (-17):

```
  0 0 0 0  1 0 0 1
+ 1 1 1 0  1 1 1 1
--------------------
  1 1 1 1  1 0 0 0
```

- My víme, že druhý sčítanec je nějaký náš *kód* pro „naše“ číslo -17
- Co ale vidí počítač?
  - Ten vidí prostě **`1110 1111`**!

Co je číslo pro procesor?

```
  0 0 0 0  1 0 0 1
+ 1 1 1 0  1 1 1 1
--------------------
  1 1 1 1  1 0 0 0
```

- My víme, že druhý sčítanec je nějaký náš *kód* pro „naše“ číslo -17
- Co ale vidí počítač?
  - Ten vidí prostě **`1110 1111`**!
- Kdybychom zapomněli, že jsme použili nějaký kód, jaké by to bylo číslo?
  - **239** ($2^7+2^6+2^5+2^3+2^2+2^1+2^0$)
- „Co je **`1111 1000`** (výsledek) za číslo v dekadické soustavě?“
  - Pokud *nevíme*, že jde o nějaký doplňkový kód, odpovíme **248**.
  - A procesor nezajímá, že mu předhazujeme čísla v nějakém našem kódu, *neví* to.
- Pro procesor je tedy **9 + (-17) = -8 totéž** jako **9 + 239 = 248**.

Co je číslo pro procesor – závěr

- Procesor vidí prostě *n*-bitové číslo
- S tímto číslem je schopen provádět aritmetické operace
  - A vždy se k němu chová stejně – jako k (bezznaménkovému) číslu
- Význam těm jedničkám a nulám dává až programátor *(tedy my)*
- Doplňkový kód poskytuje způsob, jak pouze pomocí bezznaménkových čísel a aritmetiky s nimi *simulovat* práci se znaménkovými čísly

- Kladná čísla (která mají rozsah od 0 do 2^{n-1}) v něm *vypadají stejně* jako bezznaménková čísla!
  - Příklad: číslo 123 je na 8 bitech binárně **`0111 1011`**, ať už se na něj díváme jako na běžné bezznaménkové číslo, nebo jako na *kladné* znaménkové číslo v doplňkovém kódu.

Přenosy a přetečení

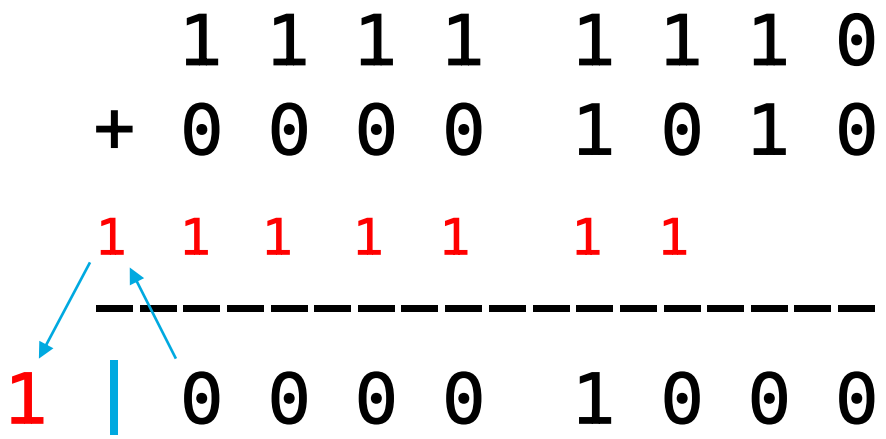
- Řekli jsme si, že typicky pracujeme s nějakým fixním počtu bitů
  - V této prezentaci pracujeme s osmibitovými čísly
- Viděli jsme ale, že někdy může při sčítání vzniknout přenos mimo původní počet bitů, se kterým pracujeme
  - **Carry** – přenos **z** **nejvyššího** bitu výsledku
- Co to tedy znamená?
  - Znamená to, že se výsledek **nevleze** do daného počtu bitů – říkáme, že došlo k **přetečení**
  - Výsledek *na* cílovém počtu bitů není správný

Přenosy a přetečení

- Příklad: 8bitová čísla *(zatím **bez znamének**, ta, se kterými pracuje procesor – k našemu oblíbenému doplňkovému kódu se dostaneme záhy)* mají rozsah **0 až 255**
  - Osm nul = 0
  - Osm jedniček = $2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0 = 255$
- Co se tedy stane, když zkusíme sečíst třeba 254 + 10?

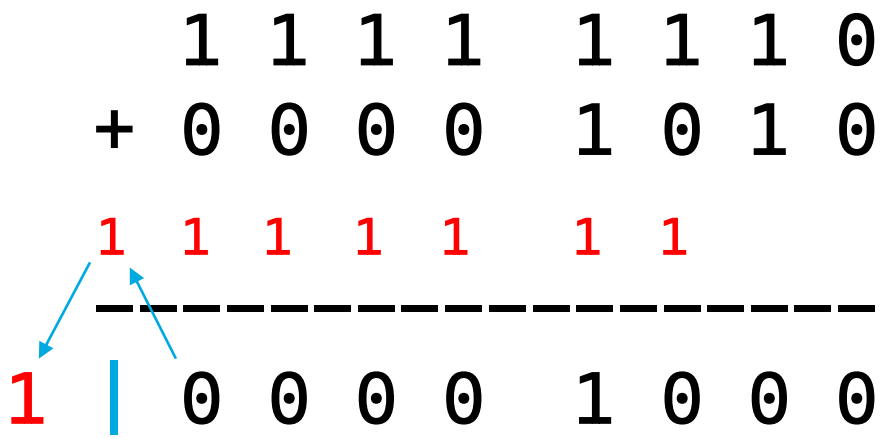
Přenosy a přetečení

- Příklad: 8bitová čísla *(zatím **bez znamének**, ta, se kterými pracuje procesor – k našemu oblíbenému doplňkovému kódu se dostaneme záhy)* mají rozsah **0 až 255**
  - Osm nul = 0
  - Osm jedniček = $2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0 = 255$
- Co se tedy stane, když zkusíme sečíst třeba 254 + 10?



Přenosy a přetečení

- Příklad: 8bitová čísla *(zatím **bez znamének**, ta, se kterými pracuje procesor – k našemu oblíbenému doplňkovému kódu se dostaneme záhy)* mají rozsah **0 až 255**
  - Osm nul = 0
  - Osm jedniček = $2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0 = 255$
- Co se tedy stane, když zkusíme sečíst třeba 254 + 10?



- Vznikl nám přenos **z nejvyššího** bitu **ven** – carry
- **Počítáme ale na osmi bitech!**
  - V 8 bitech výsledku vidíme číslo **8**, což rozhodně **není 254+10**!

Přenosy a přetečení

Při práci **s bezznaménkovými čísly** signalizuje **carry** **přetečení**!

Zkrátka – něco nám vyteklo ven z nejvyššího bitu, nemáme dost bitů na celý výsledek, výsledek je špatně.
Velmi dobře si toto moudro zapamatujme.

Přetečení v doplňkovém kódu

- Už víme, že když nám *přeteče* bezznaménkové číslo, poznáme to tak, že se objeví carry – jednička „před výsledkem“ navíc.
- Jak ale poznáme, že nám přeteklo číslo, když pracujeme v doplňkovém kódu *(když si pomocí bezznaménkové aritmetiky hrajeme na znaménkovou…)*?
- Carry samo o sobě nám **nestačí**!
- *Proč?*

Přetečení v doplňkovém kódu – carry nestačí

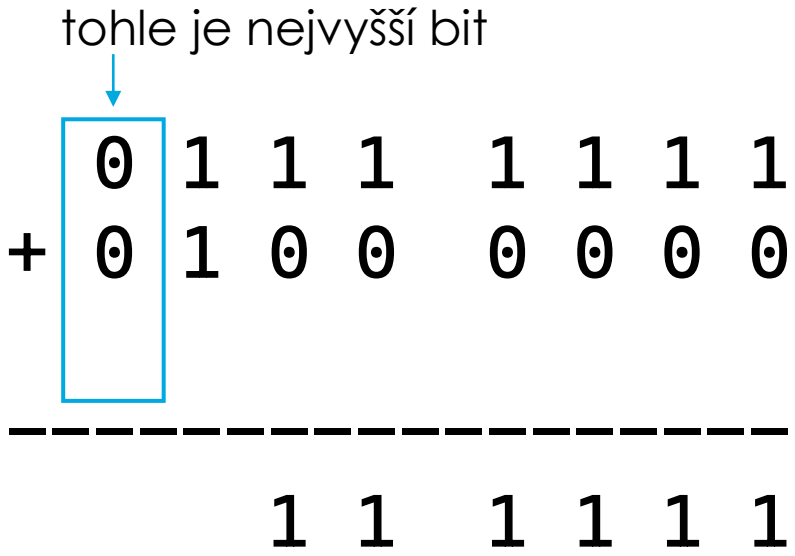
- Pro zopakování: v doplňkovém kódu jsou čísla s jedničkou *nahoře* (tedy v nejvyšším bitu) záporná, s nulou *nahoře* kladná
- Na osmi bitech:
  - Kladná čísla jsou **0000 0001** (**1**) až **0111 1111** (**127**)
  - Záporná čísla jsou **1000 0000** (**-128**) až **1111 1111** (**-1**)
  - Nula je nula: **0000 0000**
- *Vidíme problém?*
- Pokud k *nejvyššímu* vyjádřitelnému kladnému číslu přičtu jedničku, mělo by dojít k *přetečení* (protože vyšší už na daném počtu bitů nezvládnu správně vyjádřit)
- Co se stane, když k **0111 1111** přičtu jedničku?
  - Dostanu **1000 0000**, to je v našem kódu přece **-**128!
  - V bezznaménkové aritmetice by to nevadilo – přešel jsem z +127 na +128.
  - My se na to číslo ale díváme jako na znaménkové – v doplňkovém kódu.
  - 127 + 1 se rozhodně nerovná **-**128! Zjevně došlo k přetečení.
  - Carry se ale nenastavilo – z čísla nic nevyteklo ven, přeteklo to „jen v našem kódu“.

Přetečení v doplňkovém kódu

- Jak tedy poznat, že v doplňkovém kódu došlo k přetečení?
- *Kromě* carry se budeme dívat *také* na přenos **do nejvyššího** bitu.
- Co to znamená? Spočítejme **127 + 64**:

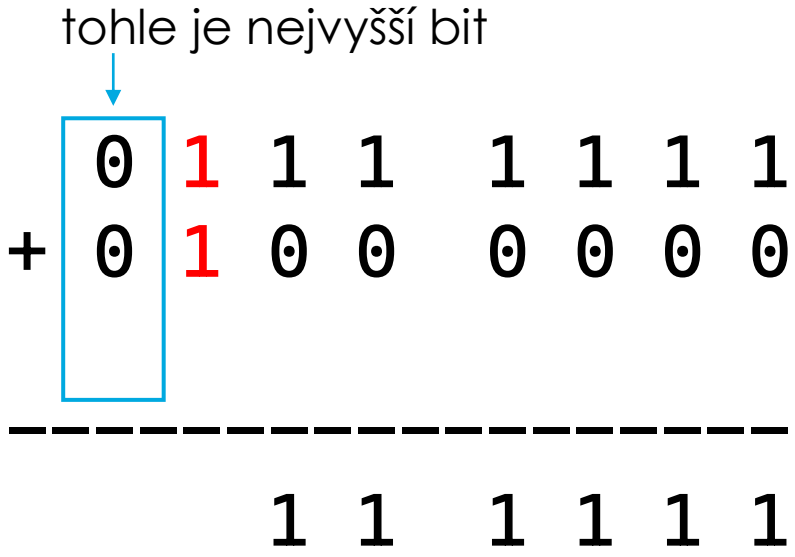
Přetečení v doplňkovém kódu

- Jak tedy poznat, že v doplňkovém kódu došlo k přetečení?
- *Kromě* carry se budeme dívat *také* na přenos **do nejvyššího** bitu.
- Co to znamená? Spočítejme **127 + 64**:



Přetečení v doplňkovém kódu

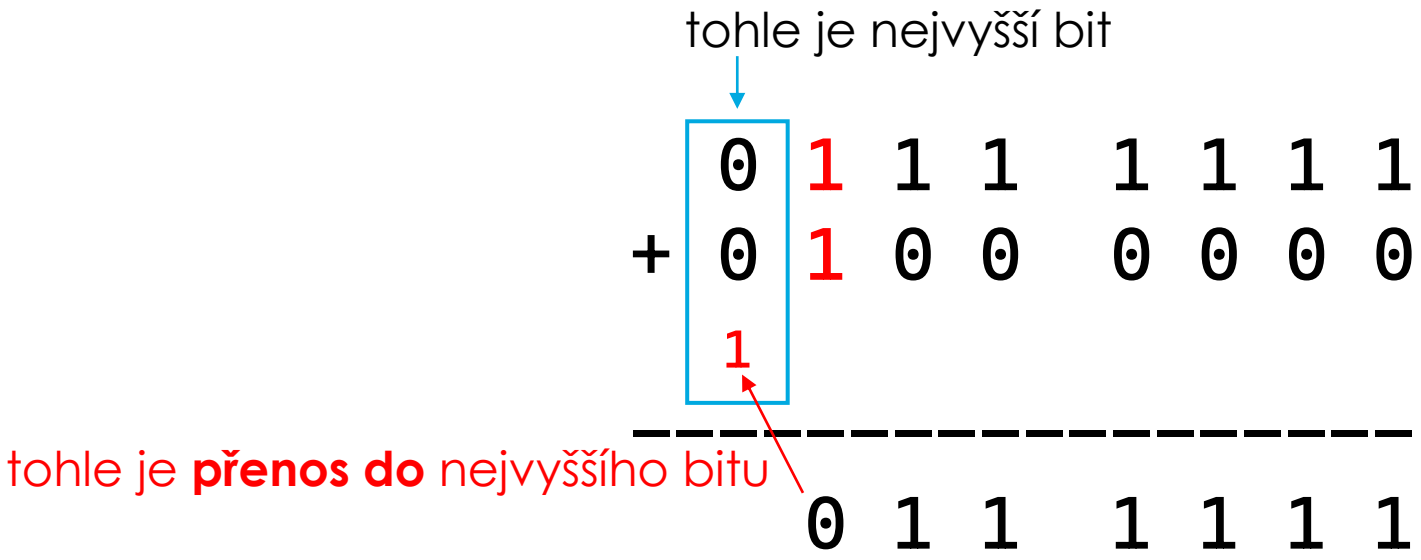
- Jak tedy poznat, že v doplňkovém kódu došlo k přetečení?
- *Kromě* carry se budeme dívat *také* na přenos **do nejvyššího** bitu.
- Co to znamená? Spočítejme **127 + 64**:



- Na *před*posledním bitu máme **1 + 1**: po sečtení tedy vznikne **přenos do** dalšího – **nejvyššího bitu**

Přetečení v doplňkovém kódu

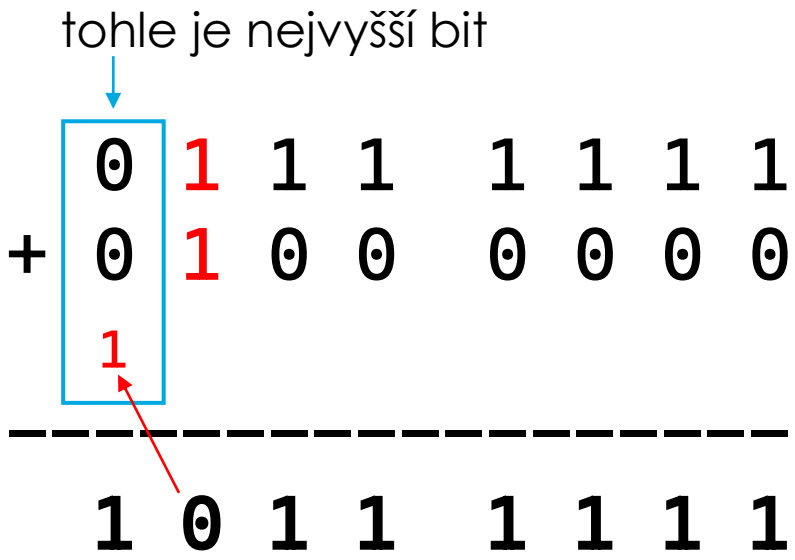
- Jak tedy poznat, že v doplňkovém kódu došlo k přetečení?
- *Kromě* carry se budeme dívat *také* na přenos **do nejvyššího** bitu.
- Co to znamená? Spočítejme **127 + 64**:



- Na *před*posledním bitu máme **1 + 1**: po sečtení tedy vznikne **přenos do** dalšího – **nejvyššího bitu**

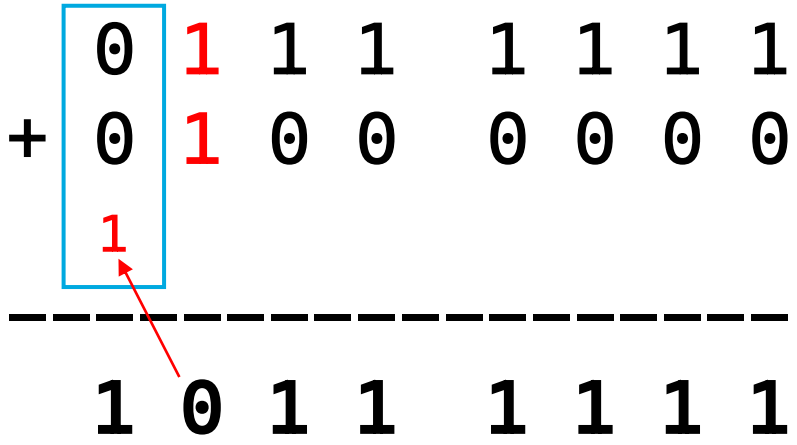
Přetečení v doplňkovém kódu

- Jak tedy poznat, že v doplňkovém kódu došlo k přetečení?
- *Kromě* carry se budeme dívat *také* na přenos **do nejvyššího** bitu.
- Co to znamená? Spočítejme **127 + 64**:



- Na *před*posledním bitu máme **1 + 1**: po sečtení tedy vznikne **přenos do** dalšího – **nejvyššího bitu**.
- Nazveme tento přenos **P**, tedy **P = 1**. Povšimněme si, že **C**arry = **0**.
  - Nevznikl přenos **ven z** nejvyššího bitu (carry), ale vznikl přenos **do** nejvyššího bitu (P).

Přetečení v doplňkovém kódu



- Nevznikl přenos **ven z** nejvyššího bitu (**C**arry)
- Vznikl přenos **do** nejvyššího bitu (**P**)
- Také vidíme, že po sčítání dvou kladných čísel *(nula na začátku!)* nám vyšlo záporné číslo *(jednička na začátku!)*, určitě tedy došlo k přetečení
- Co nám tyto informace naznačují?

Přetečení v doplňkovém kódu

Při práci **se znaménkovými čísly** v doplňkovém kódu nastává **přetečení**, když se přenos **do** nejvyššího bitu **nerovná** přenosu **z** nejvyššího bitu.

Přetečení v doplňkovém kódu

- Při práci **se znaménkovými čísly** v doplňkovém kódu nastává **přetečení**, když se přenos **do** nejvyššího bitu **nerovná** přenosu **z** nejvyššího bitu.
- *Znaménkové* přetečení nazýváme **overflow**, označujeme jako **O** nebo **OF** *(overflow flag)* a symbolicky můžeme zapsat, že

$$O = (C \neq P)$$

- Což je jen symbolický zápis věty z prvního bodu – pokud se **C**arry *nerovná* přenosu *do* nejvyššího bitu (**P**), bude **O**verflow, příznak *znaménkového* přetečení, nastaven na **1**, jinak bude **0**.
- Uvědomme si, že zatímco **C**arry je ve výsledku „hned vidět“ (jednička, která se objeví před našimi *n* bity), **O**verflow tam na první pohled „vidět není“ – musíme se podívat na carry a na přenos **do** nejvyššího bitu.
  - Zatím si tedy *overflow flag* spíš jen představujeme, ale uvidíme, že při práci s čísly v assembleru nám jej procesor bude, spolu s *carry flagem*, vyhodnocovat

Věříte mi?

- Opravdu to tak je.
- Pokuste se v tom ten smysl najít… nebo ho najděte na Google.
- Na prvním příkladu jsme viděli, že dojde k přetečení, když sečteme dvě příliš velká (kladná) čísla.
- Ukážeme si ještě několik dalších příkladů s různými možnostmi, které mohou nastat.

Příklad – validní operace

- Ukažme si, že k přetečení (signalizaci Overflow) nedojde, když k němu dojít nemá – použijme opět příklad **9 + (-17)**:

```
  0 0 0 0  1 0 0 1
+ 1 1 1 0  1 1 1 1

---------------------
```

Příklad – validní operace

- Ukažme si, že k přetečení (signalizaci Overflow) nedojde, když k němu dojít nemá – použijme opět příklad **9 + (-17)**:

```
  0 0 0 0  1 0 0 1
+ 1 1 1 0  1 1 1 1
       1    1 1 1
-------------------
    1 1  1 0 0 0
```

Příklad – validní operace

- Ukažme si, že k přetečení (signalizaci Overflow) nedojde, když k němu dojít nemá – použijme opět příklad **9 + (-17)**:

```
  0 0 0 0   1 0 0 1
+ 1 1 1 0   1 1 1 1
        1   1 1 1
-----------------
  1 1 1   1 0 0 0
```

- Sčítání na předposledním bitu **ne**vygenerovalo přenos do nejvyššího bitu

Příklad – validní operace

- Ukažme si, že k přetečení (signalizaci Overflow) nedojde, když k němu dojít nemá – použijme opět příklad **9 + (-17)**:

```
  0 0 0 0  1 0 0 1
+ 1 1 1 0  1 1 1 1
       1    1 1 1
-------------------
  1 1 1 1  1 0 0 0
```

- Sčítání na předposledním bitu **ne**vygenerovalo *přenos do* nejvyššího bitu
- Sčítání na nejvyšším bitu **ne**vygenerovalo *přenos z* nejvyššího bitu
- **C = P** = 0 → **O** = 0, znaménkové přetečení **nenastalo** – jak jsme čekali

Příklad – příliš malá čísla

- Při sečtení dvou „příliš malých“ (záporných) čísel může také nastat overflow. Použijme příklad **-120 + (-100)**:

```
  1 0 0 0  1 0 0 1
+ 1 0 0 1  1 1 0 0
-----------------
```

Příklad – příliš malá čísla

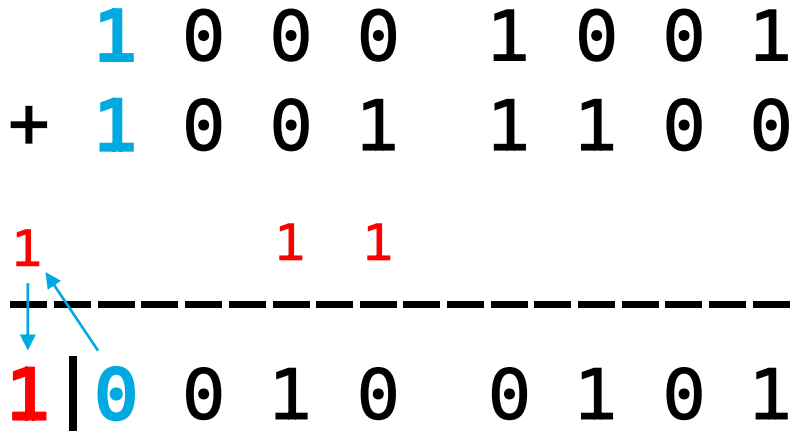
- Při sečtení dvou „příliš malých“ (záporných) čísel může také nastat overflow. Použijme příklad **-120 + (-100)**:

```
  1 0 0 0  1 0 0 1
+ 1 0 0 1  1 1 0 0
      1 1
-------------------
  0 1 0 0  0 1 0 1
```

- Sčítání na předposledním bitu **ne**vygenerovalo přenos do nejvyššího bitu

Příklad – příliš malá čísla

- Při sečtení dvou „příliš malých“ (záporných) čísel může také nastat overflow. Použijme příklad **-120 + (-100)**:



- Sčítání na předposledním bitu **ne**vygenerovalo přenos do nejvyššího bitu
- Sčítání na nejvyšším bitu **vygenerovalo** přenos **z** nejvyššího bitu (carry)
- C = 1, P = 0 → **C ≠ P** → **O** = **1**, znaménkové přetečení **nastalo** – jak jsme čekali

Příklad – validní operace se zápornými čísly

- Nyní zkusme sečíst dvě záporná čísla tak, aby byl výsledek v rozsahu (čili aby nenastal overflow) – **(-41) + (-17)**:

```
  1 1 0 1  0 1 1 1
+ 1 1 1 0  1 1 1 1

---------------------
```

Příklad – validní operace se zápornými čísly

- Nyní zkusme sečíst dvě záporná čísla tak, aby byl výsledek v rozsahu (čili aby nenastal overflow) – **(-41) + (-17)**:

```
  1 1 0 1  0 1 1 1
+ 1 1 1 0  1 1 1 1
    1 1 1  1 1 1
-------------------
      0 0  0 1 1 0
```

Příklad – validní operace se zápornými čísly

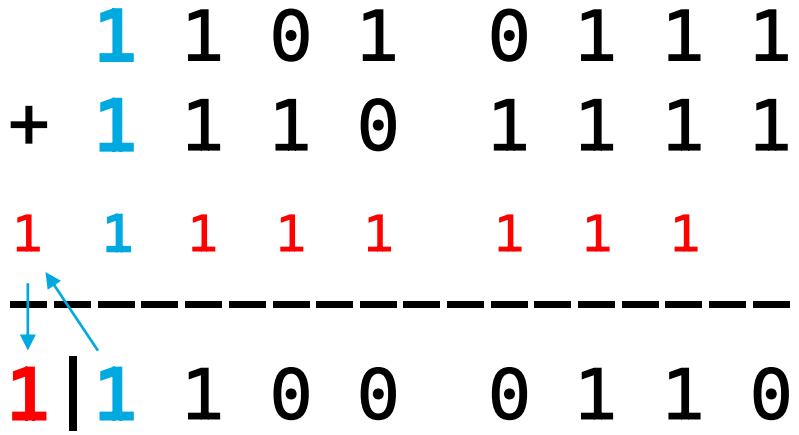
- Nyní zkusme sečíst dvě záporná čísla tak, aby byl výsledek v rozsahu (čili aby nenastal overflow) – **(-41) + (-17)**:

```
  1 1 0 1  0 1 1 1
+ 1 1 1 0  1 1 1 1
  1 1 1 1  1 1 1
-------------------
  1 0 0 0  0 1 1 0
```

- Sčítání na předposledním bitu **vygenerovalo** přenos do nejvyššího bitu

Příklad – validní operace se zápornými čísly

- Nyní zkusme sečíst dvě záporná čísla tak, aby byl výsledek v rozsahu (čili aby nenastal overflow) – **(-41) + (-17)**:



- Sčítání na předposledním bitu **vygenerovalo** přenos do nejvyššího bitu
- Sčítání na nejvyšším bitu **vygenerovalo** přenos **z** nejvyššího bitu (carry)
- **C = P** = 1 → **O = 0**, znaménkové přetečení **nenastalo** – jak jsme čekali

Bezznaménkové vs. znaménkové přetečení

```
  1 1 0 1   0 1 1 1
+ 1 1 1 0   1 1 1 1
  1 1 1 1   1 1 1
-------------------
1|1 1 0 0   0 1 1 0
```

- Povšimněme si, že v tomto příkladu se nastavil *carry* bit, ale výsledek je správně, ke **znaménkovému přetečení nedošlo**, O = 0!
- Co by ale nastavení tohoto bitu znamenalo, kdybychom *nevěděli*, že počítáme s nějakými zakódovanými *znaménkovými* čísly?
  - *Bezznaménkově* k přetečení došlo – něco vyteklo ven!

Bezznaménkové vs. znaménkové přetečení

Carry signalizuje přetečení *(nesprávný výsledek, který se nevleze do daného počtu bitů)*, **pouze** pokud čísla bereme jako **bezznaménková**!

Overflow naopak signalizuje přetečení, **pouze** pokud se na čísla díváme jako na **znaménková** (v doplňkovém kódu).

Jde jen o to, jak s čísly *chceme* pracovat *my*.
Víme, že pracujeme se znaménky? Tak se budeme dívat jen na overflow flag, carry nás nezajímá.
Víme, že nepracujeme se znaménky? Tak se budeme dívat jen na carry flag, overflow nemá žádný význam.