# Aritmetické a logické operace (násobení)

INP - cvičení

Zdeněk Vašíček, Vojtěch Mrázek

2025

# Násobení

- Existuje řada přístupů (Riple Carry Array, Carry Save Array, Wallace tree, LUT), které se v zásadě liší
  - v ploše, kterou násobička na čipu zabírá a
  - v rychlosti (zpoždění násobičky).

- Operace násobení se v HW realizuje jako postupné přičítání vhodně posunutého násobence na základě znalosti jednotlivých bitů násobitele.

# Násobení 8b čísel bez znaménka v dvojkové soustavě

```
   01010010  (násobenec)
 × 01111101  (násobitel)
 ----------
    ........
   .........
  ..........
 ...........
............
 ----------------
................  (součin)
```

```
   82
 × 125
 -----
  410
 1640
 8200
 -----
 10250
```

Výsledný součin musí obsahovat 8+8 = 16 bitů

# Násobení 8b čísel bez znaménka v dvojkové soustavě

```
  01010010  (násobenec)
× 01111101  (násobitel)
----------
  01010010
 000000000
0101001000
01010010000
01010010000
01010010000000
01010010000000
----------------
0010100000001010  (součin)
```

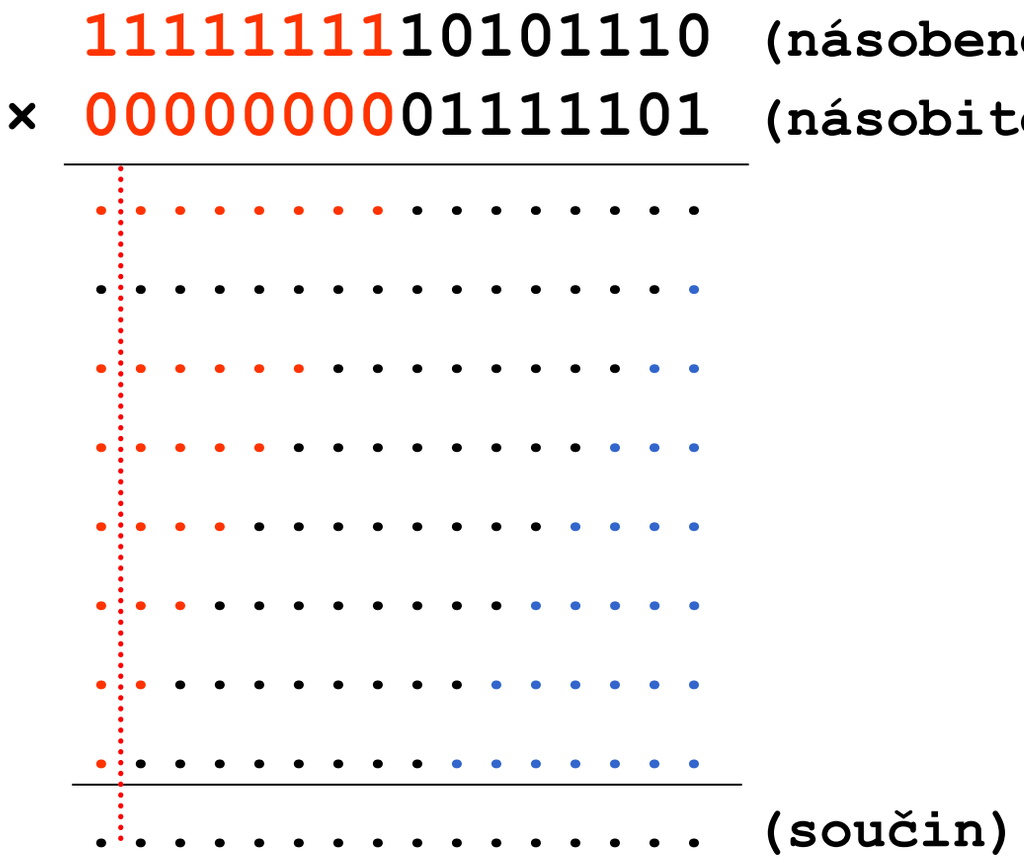
(dílčí součiny 8x)

```
   82
× 125
------
  410
 1640
 8200
------
10250
```

Výsledný součin musí obsahovat 8+8 = 16 bitů

# Násobení 8b čísel se znaménkem v dvojkové soustavě

- Naivní implementace – zapamatování znaménka a převod na kladná čísla (nepoužívá se).
- Lze použít předchozí algoritmus je však nutné rozšířit (na šířku součinu) / šířit znaménko.




```
   -82
 × 125
------
  -410
 -1640
 -8200
------
-10250
```

# Násobení 8b čísel se znaménkem v dvojkové soustavě

- Naivní implementace – zapamatování znaménka a převod na kladná čísla (nepoužívá se).
- Lze použít předchozí algoritmus je však nutné rozšířit (na šířku součinu) / šířit znaménko.

```
  1111111110101110  (násobenec)
× 0000000001111101  (násobitel)
------------------
  1111111110101110
  0000000000000000
  1111111010111000
  1111110101110000
  1111010101110000
  1110101110000000
  1101011100000000
  0000000000000000
------------------
  1101011111110110  (součin)
```

```
   -82
×  125
-------
  -410
 -1640
 -8200
-------
-10250
```

# Násobení 8b čísel se znaménkem v dvojkové soustavě

```
  00000000 01111101  (násobenec)
× 11111111 10101110  (násobitel)
-------------------
  0000000000000000
  0000000 01111101 0
  000000 01111101 00
  00000 01111101 000
  0000000000000000
  000 01111101 00000
  0000000000000000
  0 01111101 0000000
  1 01111101 0000000
  1 11101 0000000000
  1 1101 00000000000
  1 101 000000000000
  1 01 0000000000000
  1 1 00000000000000
-------------------
  1101011111110110  (součin)
```

```
      125
  ×   -82
-----------
     -250
    -10000
-----------
    -10250
```

# Optimalizace z pohledu zpoždění

- Násobení sestává z řízeného přičítání dílčích součinů (násobence) k průběžnému výsledku. Pokud je cílem proces zrychlit, musíme
  - buď urychlit operaci přičtení dílčího součinu nebo
  - počet operací přičtení redukovat (použitím jiné reprezentace).

- Motivace: hodnotu $15_{10} = 1111_2$ lze reprezentovat také jako rozdíl

$$16 - 1 = 10000_2 - 00001_2$$

- Při použití předchozího algoritmu se pro výpočet součinu 7 x 15 musí vyčíslit výraz $0111_2 + 01110_2 + 011100_2 + 0111000_2$ avšak při použití této reprezentace stačí vyčíslit výraz $01110000_2 - 0111_2$. Lze tedy ušetřit 50% operací (pozor na interpretaci).

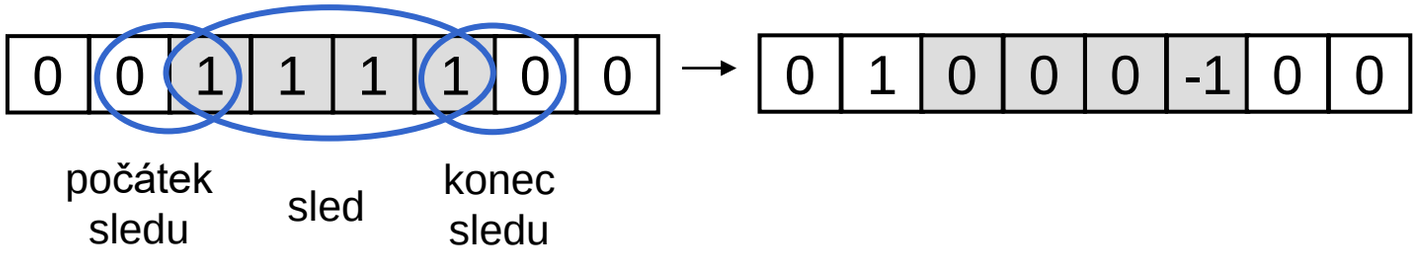
# Boothův algoritmus

- Boothův algoritmus je algoritmus určený k redukci počtu operací, který využívá mimo operaci přičtení násobence také jeho odečtení.
- Princip: detekovat a nahradit řadu po sobě následujících jedniček (sled jedniček) v násobiteli a tím redukovat počet operací.


- Existuje několik variant, základní verze je Boothovo překódování radix 2 využívající znalosti bitu a jeho pravého souseda (je-li MSB vlevo).

# Boothův algoritmus (radix 2) – tvorba kódu

- Lze detekovat sled jedniček pomocí hodnoty bitu a jeho souseda?




- Detekci můžeme provádět na základě znalosti sousedního bitu

| hodnota překódovávaného bitu | hodnota sousedního bitu (bit vpravo) | kód | popis situace |
|---|---|---|---|
| 0 | 0 / neexistuje | | |
| 0 | 1 | | |
| 1 | 1 | | |
| 1 | 0 / neexistuje | | |

- Výhoda: překódování lze provést najednou pro všechny bity
- Neefektivita: za sled je považována i osamocená jednička (010), která je nahrazena dvěma operacemi (1-10). Řešení: použít vyšší radix

# Násobení 8b čísel – Boothův algoritmus – radix 2

**82 x 125 = 01010010 x 01111101**

**125:**

| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

→

| 1 | 0 | 0 | 0 | 0 | -1 | 1 | -1 |
|---|---|---|---|---|---|---|---|

Kontrola: $125 \stackrel{?}{=} 1 \cdot 2^7 + -1 \cdot 2^2 + 1 \cdot 2^1 + -1 \cdot 2^0 = 128 - 4 + 2 - 1 = 125$

# Násobení 8b čísel – Boothův algoritmus – radix 2

**82 x 125 = 01010010 x 01111101**

**125:**

| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

→

| 1 | 0 | 0 | 0 | 0 | -1 | 1 | -1 |
|---|---|---|---|---|---|---|---|

```
                        0  1  0  1  0  0  1  0     (82)
×                       1  0  0  0  0 -1  1 -1     (125)
-----------------------------------------------
                        .  .  .  .  .  .  .  .
                        .  .  .  .  .  .  .  .
                        .  .  .  .  .  .  .  .
                        .  .  .  .  .  .  .  .
-----------------------------------------------
 0  0  1  0  1  0  0  0  0  0  0  0  1  0  1  0
```

# Násobení 8b čísel – Boothův algoritmus – radix 2

**82 x 125 = 01010010 x 01111101**

**125:**

| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

→

| 1 | 0 | 0 | 0 | 0 | -1 | 1 | -1 |
|---|---|---|---|---|---|---|---|

```
                                0 1 0 1 0 0 1 0    (82)
×                               1 0 0 0 0-1 1-1    (125)
-------------------------------------------------
  1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 0    (-82)
  0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0    (+82 << 1)
  1 1 1 1 1 1 1 0 1 0 1 1 1 0 0 0    (-82 << 2)
  0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0    (+82 << 7)
-------------------------------------------------
  0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0
```

Počet operací se redukoval z 6 na 4 (pozor na interpretaci)

Kontrola: 82x125 = 82x(128 + -4 + 2 + -1) = -82 + 82x2 + -82x4 + 82x128

# Boothův algoritmus (radix 4) – tvorba kódu

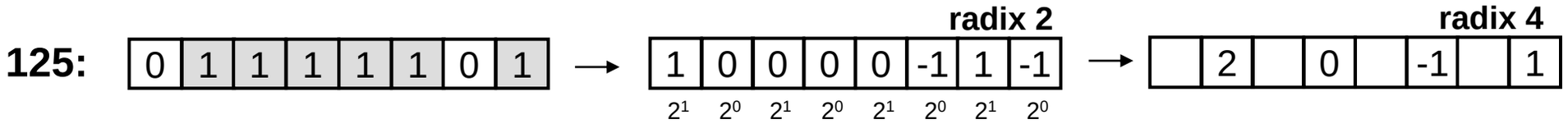
- Umožníme-li používat dvojnásobnou zápornou a kladnou hodnotu násobence, můžeme dále redukovat počet operací (vyrobit v HW požadované násobky není obtížné).
- **Kód vyššího řádu lze vytvořit z kódu radix 2 přiřazením vah $2^i$ jednotlivým pozicím.**

| hodnota překódovávaných bitů | hodnota sousedního bitu (bit vpravo) | kód radix 2 | kód radix 4 |
|---|---|---|---|
| 00 | 0 | 0 0 → | 0 |
| 00 | 1 | 0 1 → | 1 |
| 01 | 0 | 1-1 → | 1 |
| 01 | 1 | 1 0 → | 2 |
| 10 | 0 | -1 0 → | -2 |
| 10 | 1 | -1 1 → | -1 |
| 11 | 0 | 0-1 → | -1 |
| 11 | 1 | 0 0 → | 0 |

- Rychlá kontrola - nejvyšší bit určuje znaménko (1 – záporné, 0 – kladné)

# Násobení 8b čísel – Boothův algoritmus – radix 4

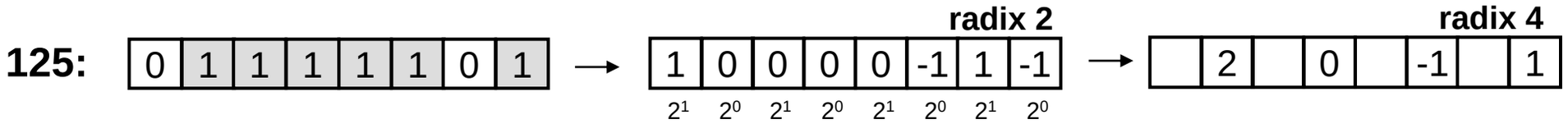
**82 x 125 = 01010010 x 01111101**




Kontrola: $125 \stackrel{?}{=} 2 \cdot 2^6 - 1 \cdot 2^2 + 1 \cdot 2^0 = 128 - 4 + 1 = 125$

# Násobení 8b čísel – Boothův algoritmus – radix 4

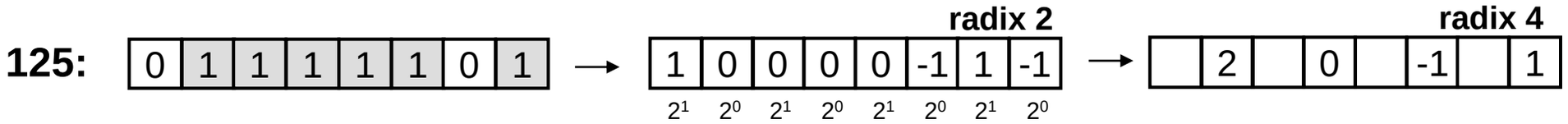
**82 x 125 = 01010010 x 01111101**




```
          0  1  0  1  0  0  1  0      (82)
   ×      2     0    -1     1       (125)
   ------------------------------
          .  .  .  .  .  .  .  .
          .  .  .  .  .  .  .  .  .  .
          .  .  .  .  .  .  .  .  .  .  .  .
   ------------------------------
          .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
```

# Násobení 8b čísel – Boothův algoritmus – radix 4

**82 x 125 = 01010010 x 01111101**

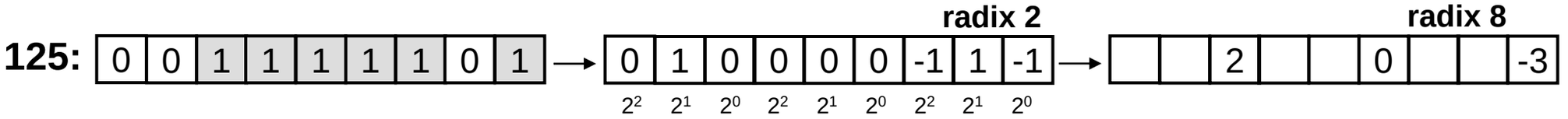



```
          0 1 0 1 0 0 1 0      (82)
×       2   0  -1   1        (125)
-------------------------------
  0000000001010010           (+82)
  1111111010111000           (-82 << 2)
  0010100100000000           (+164 << 6)
-------------------------------
  0010100000001010
```

- Pozor na důslednou práci se znaménkem a potřebný počet bitů pro zakódování dvojnásobku násobence. Jednotlivé dílčí součiny je zapotřebí posouvat o 2 bity vlevo!
- Počet operací se redukoval z 6 na 3 (maximálně 4 operace)

# Násobení 8b čísel – Boothův algoritmus – radix 8

**82 x 125 = 01010010 x 01111101**

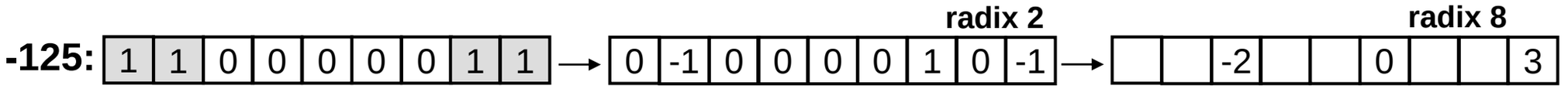



```
      0  0  1  0  1  0  0  1  0      (82)
  ×      2        0       -3       (125)
  ---------------------------------
   1111111100001010                 (-246)
   0010100100000000                 (+164 << 6)
  ---------------------------------
   0010100000001010
```

- Počet bitů musí být násobkem 3 (přechod na 9-bitová čísla)
- Počet operací se redukoval z 6 na 2.

# Násobení 8b čísel – Boothův algoritmus – radix 8

**-82 x -125 = 110101110 x 110000011**




```
      1  1  0  1  0  1  1  1  0      (-82)
  ×     -2        0        3       (-125)
  ---------------------------------------
   1111111100001010                 (-246)
   0010100100000000                 (+164 << 6)
  ---------------------------------------
   0010100000001010
```

- Počet bitů musí být násobkem 3 (přechod na 9-bitová čísla)
- Počet operací se redukoval z 4 na 2 (viz počet jedničkových bitů násobitele).

# Boothův algoritmus – jiný přístup

**Postup podle obecného Boothova algoritmu:**

1. Zopakujeme nejvyšší bit skupiny (skupina je *k*-tice bitů, *k* dáno radixem)
2. K takto získané hodnotě přičteme hodnotu nejvyššího bitu skupiny vpravo od překódovávané skupiny
3. Výsledná hodnota je binárním vyjádřením relativní číslice

**Překódujte číslo -121 na 9 bitech pomocí radix 8 ($2^k = 8$, k = 3)**

**-121 =**

| 1 1 0 | 0 0 0 | 1 1 1 |
|---|---|---|
| 1 1 1 0 | 0 0 0 0 | 1 1 1 1 |
| 1 1 1 0 | 0 0 0 1 | 1 1 1 1 |
| -2 | 1 | -1 |

Ověření správnosti výsledku:

$$-121 = -2 \cdot 2^6 + 1 \cdot 2^3 - 1 \cdot 20 = -2 \cdot 64 + 1 \cdot 8 - 1 \cdot 1$$

# Boothův algoritmus

- Q: Překódujte číslo -257 na 12 bitech do Boothova kódu radix 8.
- Q: Vynásobte čísla -81 x -78 s využitím Boothova algoritmu.
- Q: Kolik operací ušetříme při násobení čísel -81 x -78 na 8 bitech zavedením Boothova překódování radix 4?

# Boothovo překódování s radixem 4 ve VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity RECODER is
  port (
    DIN  : in  std_logic_vector(15 downto 0);
    BIT3 : in  std_logic_vector( 2 downto 0);
    DOUT : out std_logic_vector(16 downto 0));
end RECODER;

architecture RTL of RECODER is
  constant N : integer := 16;
  subtype bitn1 is std_logic_vector(N downto 0);

  function COMP2 (D : in bitn1) return bitn1 is
    variable Dn : bitn1;
  begin
    Dn := not D;
    return(Dn+1);
  end COMP2;
```

Funkce ve VHDL vracející dvojkový doplněk čísla D

# Boothovo překódování s radixem 4 ve VHDL (2)

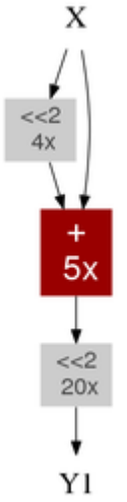
```
begin
  process (DIN, BIT3)
  begin
    case BIT3 is
      when "001" | "010" =>
        DOUT <= DIN(N-1) & DIN;
      when "101" | "110" =>
        DOUT <= COMP2(DIN(N-1) & DIN);
      when "100" =>
        DOUT <= COMP2(DIN & '0');
      when "011" =>
        DOUT <= DIN & '0';
      when others =>
        DOUT <= (others => '0');
      end case;
  end process;
end RTL;
```

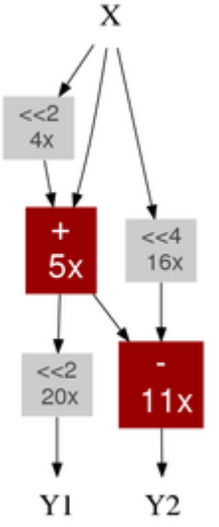
| překódovávaný blok | | | kód |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 2 |
| 1 | 0 | 0 | -2 |
| 1 | 0 | 1 | -1 |
| 1 | 1 | 0 | -1 |
| 1 | 1 | 1 | 0 |

# Násobička s konstantním koeficientem

- Násobení konstantou (více konstantami) vede obecně na jednodušší HW implementaci (Multiplierless Constant Multiplication problem).
  Nalezení optimální HW implementace je NP problém.

- Příklad:
  - Y = 20 * X (X zakódováno na $w$ bitech)
  - Varianta I (jedna sčítačka $w$-bitová)
    - Y = 20 * X = (16 + 4) * X = 16*X + 4*X
  - Varianta II (jedna sčítačka $w$-bitová)
    - Y = 20 * X = 4 * 5 * X  = 4 * (1 + 4) * X




- Příklad:
  - Y = {20*X, 11*X}
  - Y = {4*5*X, (16-5)*X}

# Násobička s konstantním koeficientem

- Q: Nakreslete co možná nejefektivnější implementaci 8-bit násobičky násobící koeficientem 15.
- Q: Kolik sčítaček/odčítaček bude zapotřebí pro implementaci optimální násobičky s konstantním koeficientem 17 (49, 82)?
- Q: Jak široké odčítačky budou zapotřebí pro realizaci optimální násobičky s konstantním koeficientem 47?