Akcelerované algoritmy násobení

Proč akcelerované metody

- Standardní uživatel komerčního procesoru může ovlivnit rychlost provádění operací jen jeho výběrem.
- Jiná situace je při návrhu digitálních systémů na bázi FPGA nebo ASIC.
- Návrh specializovaných funkčních bloků může probíhat i na úrovni obvodového řešení jednotek, jako jsou násobičky, děličky, popř. jiné specializované jednotky.

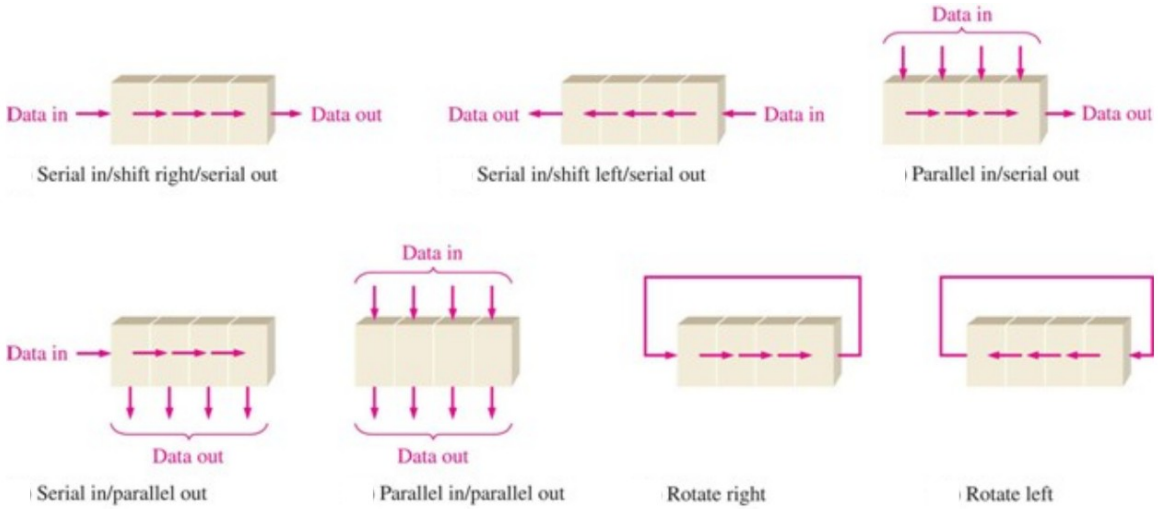
Charakteristika používaných metod a algoritmů pro násobení

- Sekvenční metody
  - Boothův algoritmus a jeho modifikace
- Paralelní – iterativní techniky
  - Paralelní čítače
  - Iterační aritmetická pole
- Použití speciálního kódování
  - Logaritmické zobrazení čísel
  - Modulární aritmetika - kódy zbytkových tříd

Posuvné jednotky (Shifters)

- Rotace (doleva/doprava) : 100111 <- 110011 -> 111001
- posuv logický (doleva/doprava): 100110 <- 110011 -> 011001
- posuv aritmetický (doleva/doprava): 100110 <- 110011 -> 111001

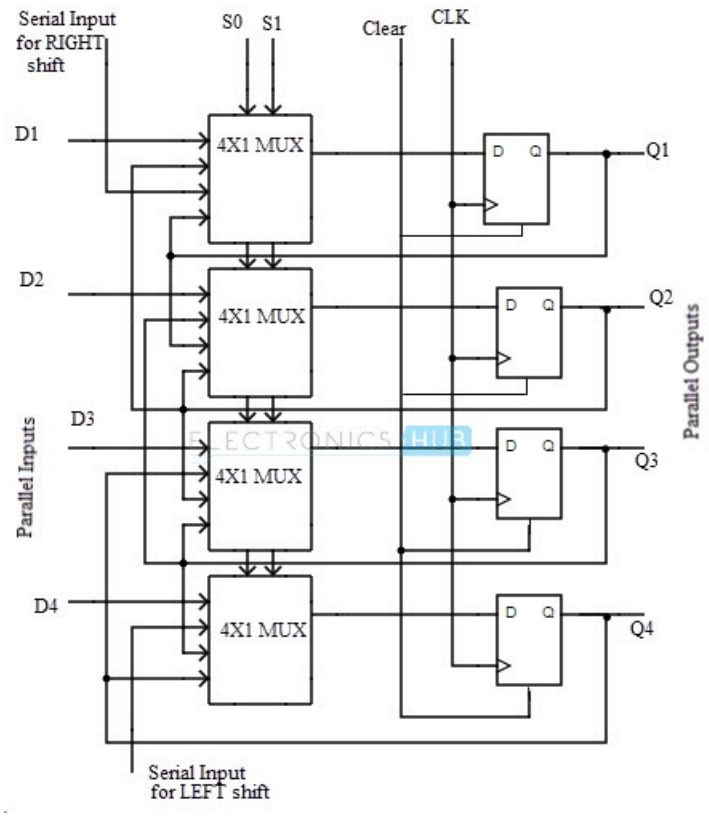
- Vstup sériový/paralelní, výstup sériový/paralelní
- směr vlevo/pravo/volitelný
- o 1 krok, více kroků, volitelné



Univerzální posuvný reg.

Řízení S0,S1 umožnuje volit fci
- Načtení
- posuv vpravo
- posuv vlevo
- bez akce

Na N-bitový posuv
je třeba N-taktů

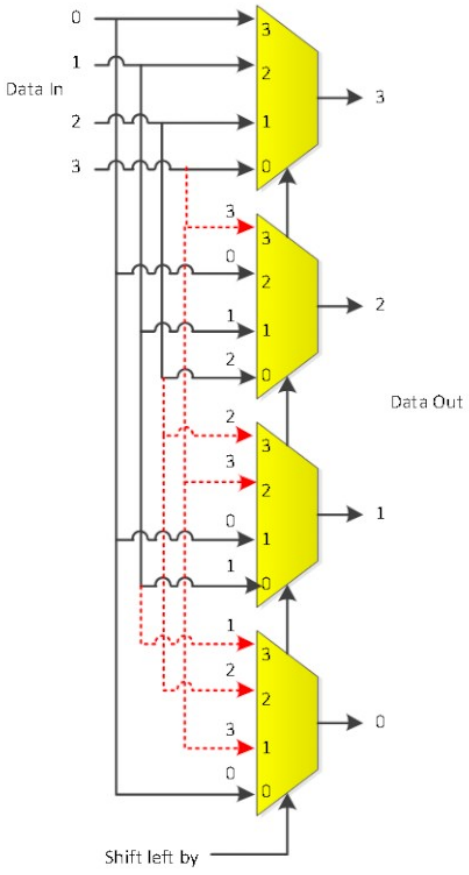


Barrel Shifter

- Univerzální posuvná jednotka založená na čitě kombinační logice

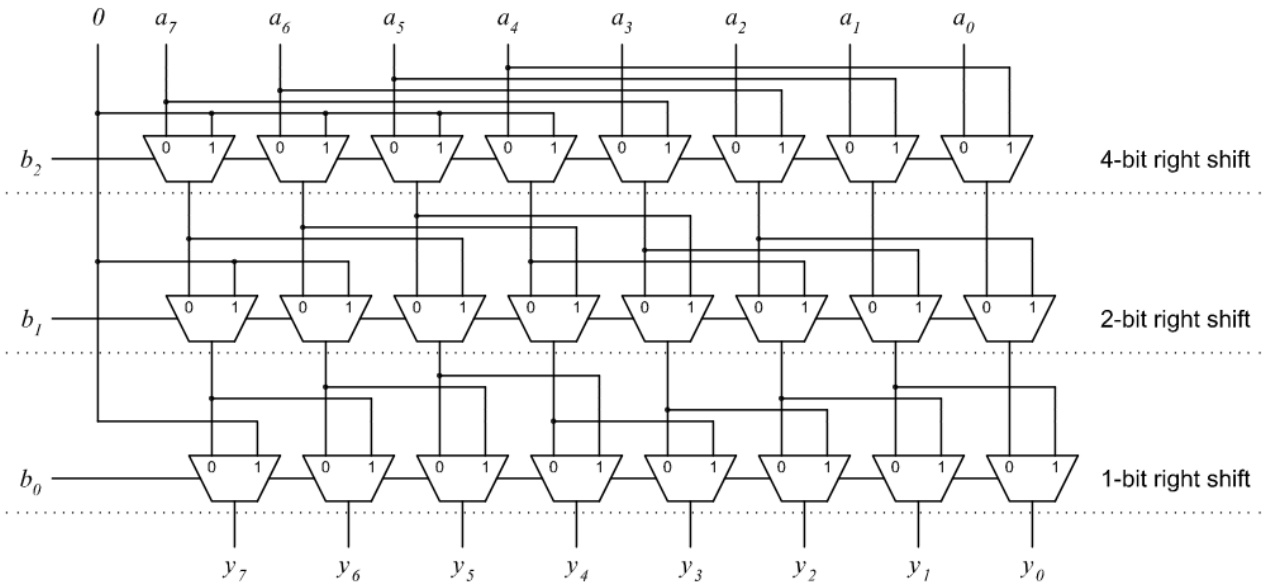
Posuvná jednotka - multiplexer

- Posuvná jednotka je implemetována Multiplexerem
- pro vyšší N – vysoký řád multiplexeru



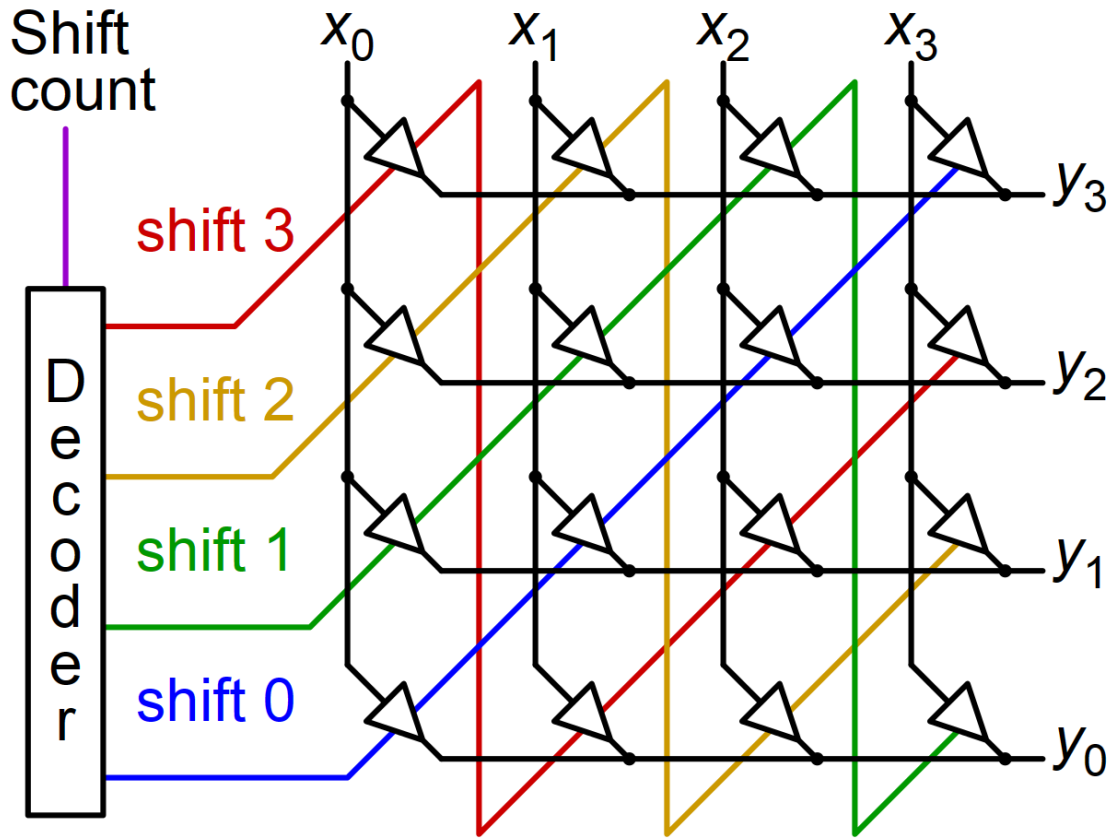
Posuvná jednotka - Logaritmická

K-úrovní, každá posouvá o 2^b (b = 1..K)
Vstupy b pak volba posunout/neposunout v dané úrovni
Pro N-bitový vstup, K=log(N)

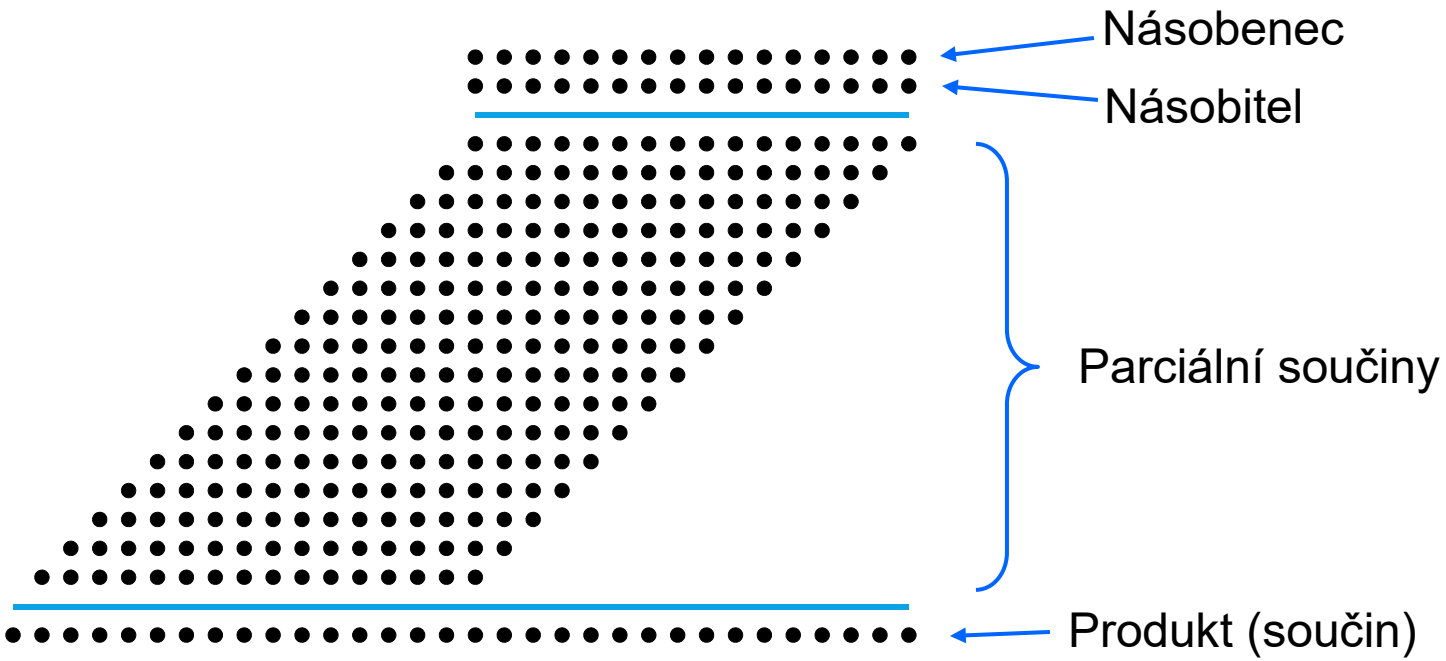


Posuvná jednotka - matice

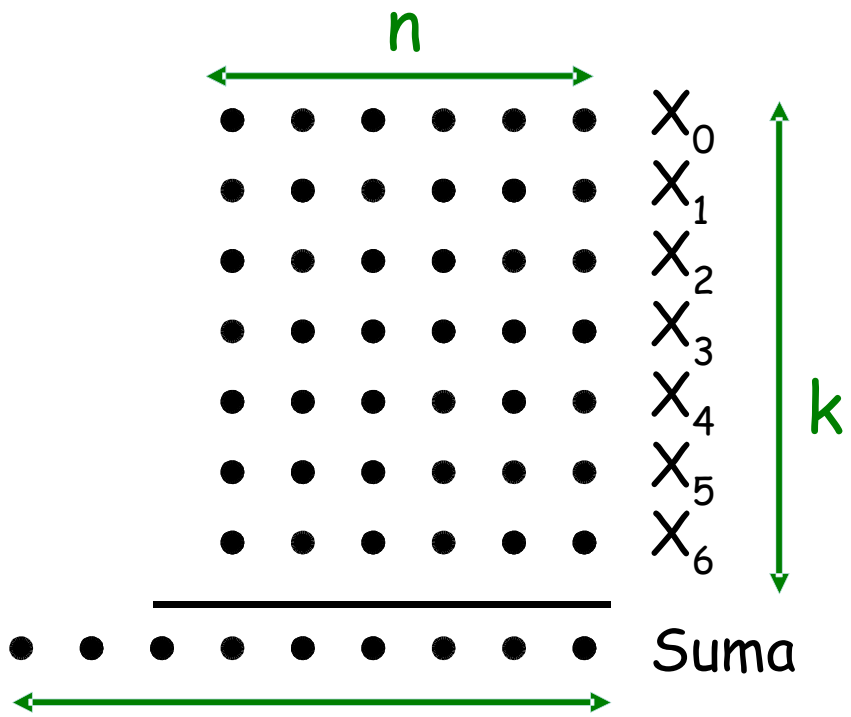
Maticová propojení –> 1 úroveň logiky = velmi rychlé
x=vstup, y´výstup, shift určuje posuv



Vytváření součinu



Víceoperandové sčítání

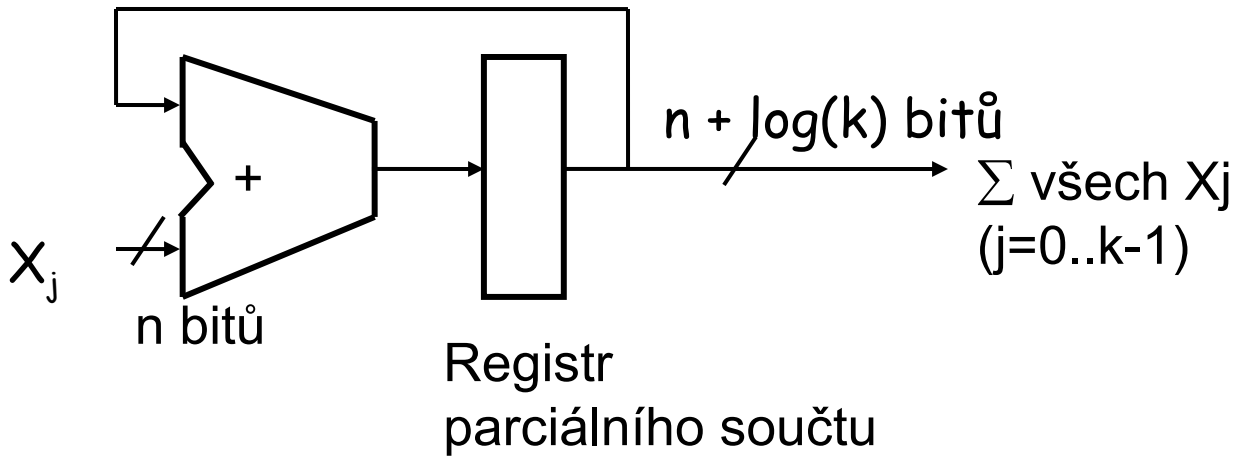


$\lceil \log_2(k.(2^n-1)) \rceil \approx n + \log_2(k)$

Součet většího počtu čísel:

- Násobení vektorů
- Výpočet středních hodnot
- Digitální filtrace

Seriová implementace



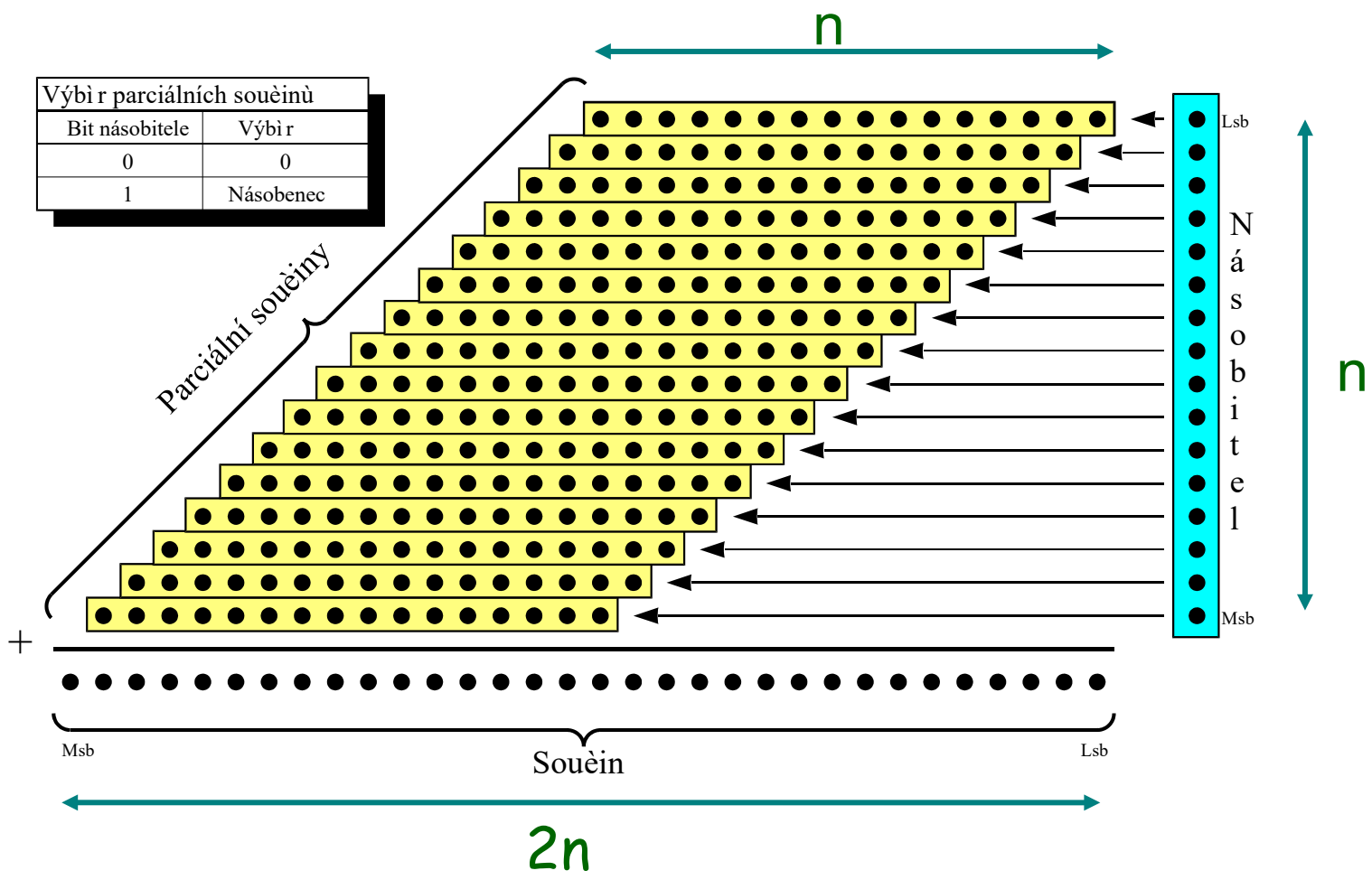
$T_{sériové\text{-}multiadd} = O(k.(n + log(k))$ ... RCA adder (basic)

$T_{sériové\text{-}multiadd} = O(k.log(n + log(k)))$ ... CLA adder (fast)

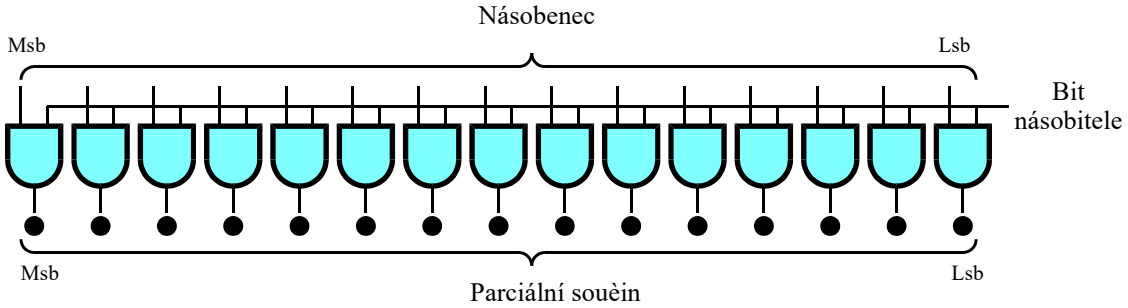
Doba součtu roste superlineárně s k je-li n konstantní a logaritmicky s n pro konstantní k.

(superlineární ≈ loglineární, označení k.log(k))

Vytvoření výsledku - součinu



Vytváření parciálních součinů



- Vytváření parciálních součinů pro základní metodu násobení (test jednoho bitu, postup po jednom bitu)
- Parciální součin vznikne jako součin násobence a jednoho bitu násobitele.

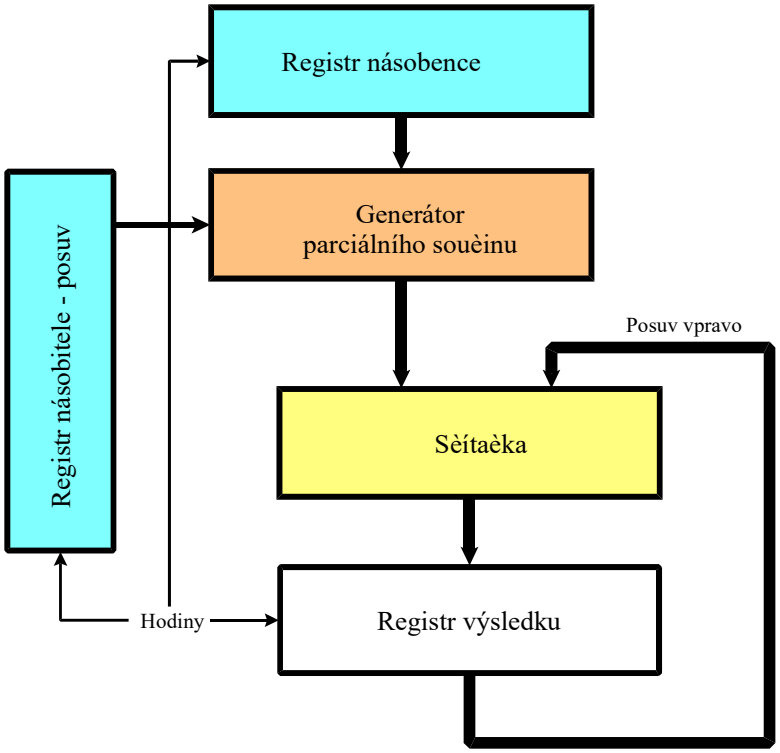
$$T_{sér\text{-}mult} = O(n.\log(n))$$

Doba násobení roste superlineárně s n (použijeme-li sčítačku s logaritmickou závislostí doby součtu)

Násobení - posuv & součet

- Posuv vlevo a součet
  - Parciální součiny se sčítají zdola nahoru
  - Metoda vyžaduje sčítačku o šířce 2n bitů (=> menší rychlost)
- Posuv vpravo a součet
  - Parciální součiny se sčítají shora dolů
  - Metoda vyžaduje sčítačku o šířce n bitů

Násobička (sériovo-paralelní, sériová)

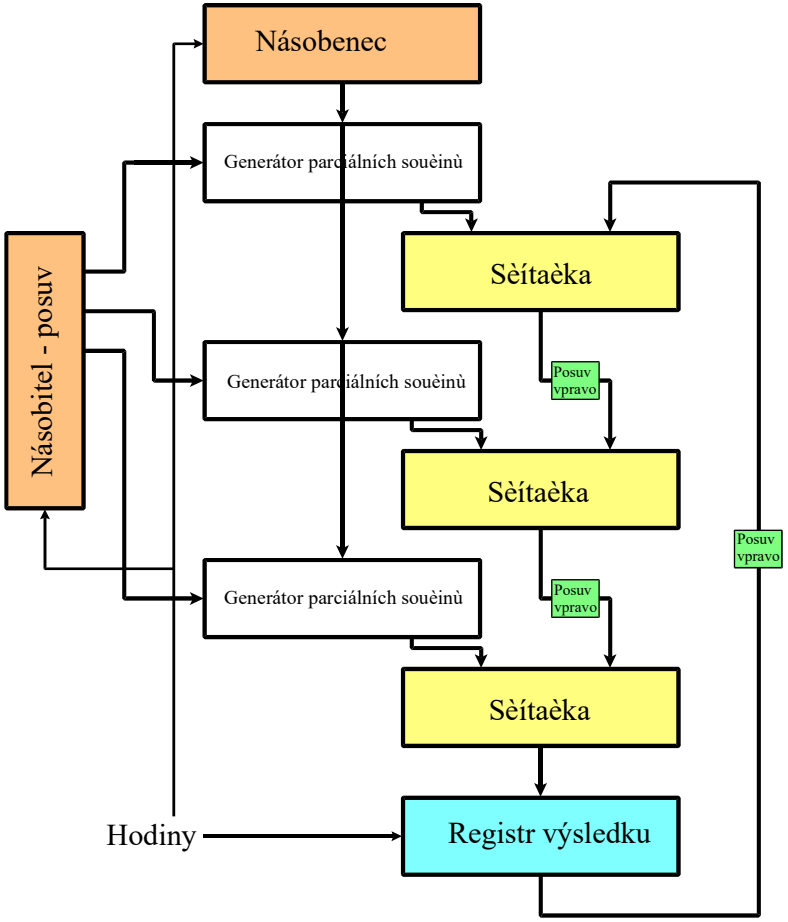


Základní sekvenční metoda binárního násobení:

- Počet cyklů lineárně závislý na délce zobrazení (n)
- Délka součinu – výsledku rovna 2*n
- Délka sčítačky je rovna n-bitů
- Sčítačka musí dokončit součet před začátkem dalšího cyklu
- Postup od nižších řádů násobitele (výhodnější - ! délka sčítačky !)

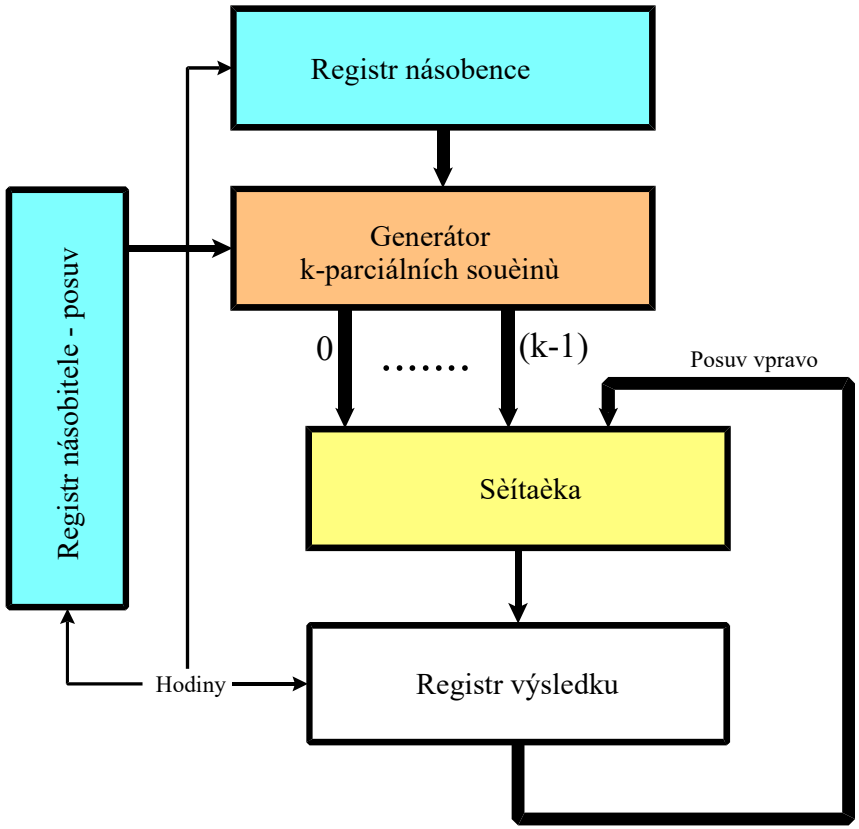
Sčítačka buď paralelní n-bitová,
pak T=O(n*?)
(?=dle typu sčítačky, např.log(n) pro CLA)
nebo sériová (1bitová),
pak T=O(n*n)

Akcelerace – použití většího počtu sčítaček



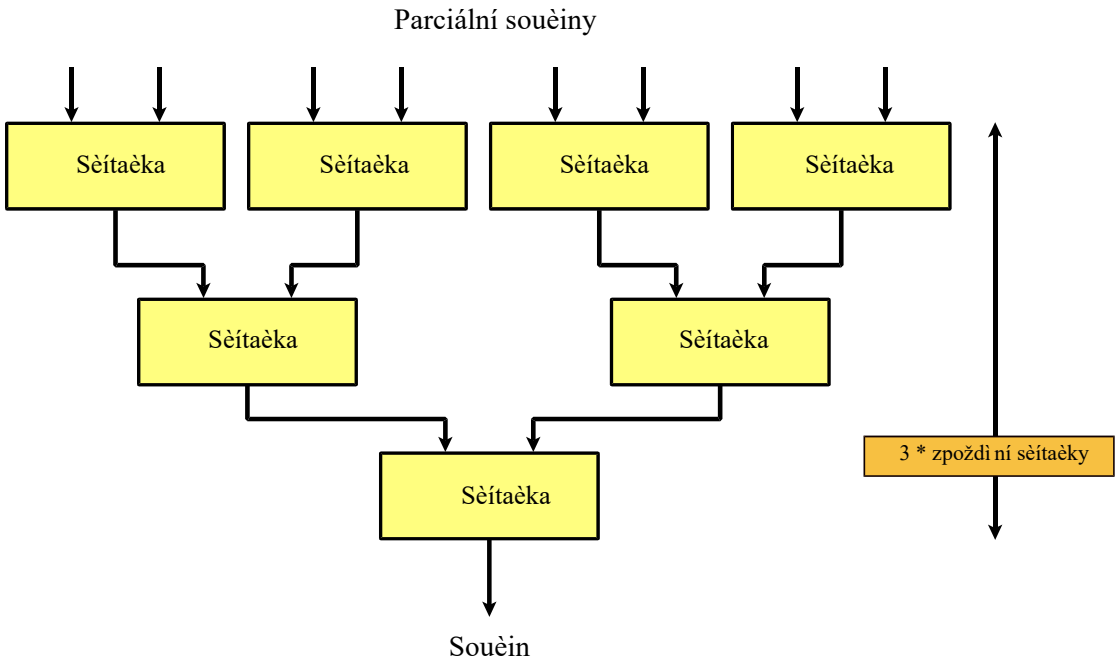
- Proces lze zrychlit zavedením většího počtu sčítaček
- Zvýšení rychlosti omezeno tím, že sčítačky jsou vlastně řazeny sériově.
- Předchozí nevýhodu lze odstranit:
  - použitím speciálně navržených vícevstupových sčítaček
  - použitím stromové struktury sčítaček (HW náročné)

Akcelerace – použití vícevstupových sčítaček



- Redukuje se počet cyklů
- Větší obvodová složitost sčítačky
- Doba součtu stejná jako pro sčítání dvou operandů

Stromové uspořádání sčítaček



Počet úrovní „stromu“ je úměrný logaritmu počtu parciálních součinů a tedy log(n).

Celková rychlost:

N+log(N) ..vse paralelne

N.log(N) .. pipeline po vrstvách

Sčítačky ležící v jedné vrstvě pracují paralelně !

Násobení záporných čísel

Nechť $B = (b_{n-1}\ b_{n-2} \ldots b_2\ b_1\ b_0)$

znaménkový bit

$P_0 = 0$

$P_1 = P_0 + b_0 A 2^0$

$P_2 = P_1 + b_1 A 2^1$

. . .

$P_{i+1} = P_i + b_i A 2^i = (\sum_{j=0}^{i} b_j 2^j) A$

Potom

$P_{n-1} = (\sum_{j=0}^{n-2} b_j 2^j) A = B * A$

Záporný násobenec (A)

- Expanzí znaménkového bitu, dostaneme správný výsledek

- Součet čísel (A*bi) v doplňkovém kód dává správný výsledek
- Násobení probíhá bez (nulového) znaménkového bitu B – viz dále
- 4*součet 2*n bitového čísla

Záporný násobitel (B)

- Pro doplňkový kód platí:

$$A = -a_{n-1}2^{n-1} + \sum_{j=0}^{n-2} a_j 2^j \quad a \quad B = -b_{n-1}2^{n-1} + \sum_{j=0}^{n-2} b_j 2^j$$

Zatím jsme určili (součin bez znam.bitu B – dříve jsme ukázali že je ok):

$$P_{n-1} = \left(\sum_{j=0}^{n-2} b_j 2^j\right) A$$

(suma jen do (n-2), tedy bez b_{n-1})

potřebujeme ale:

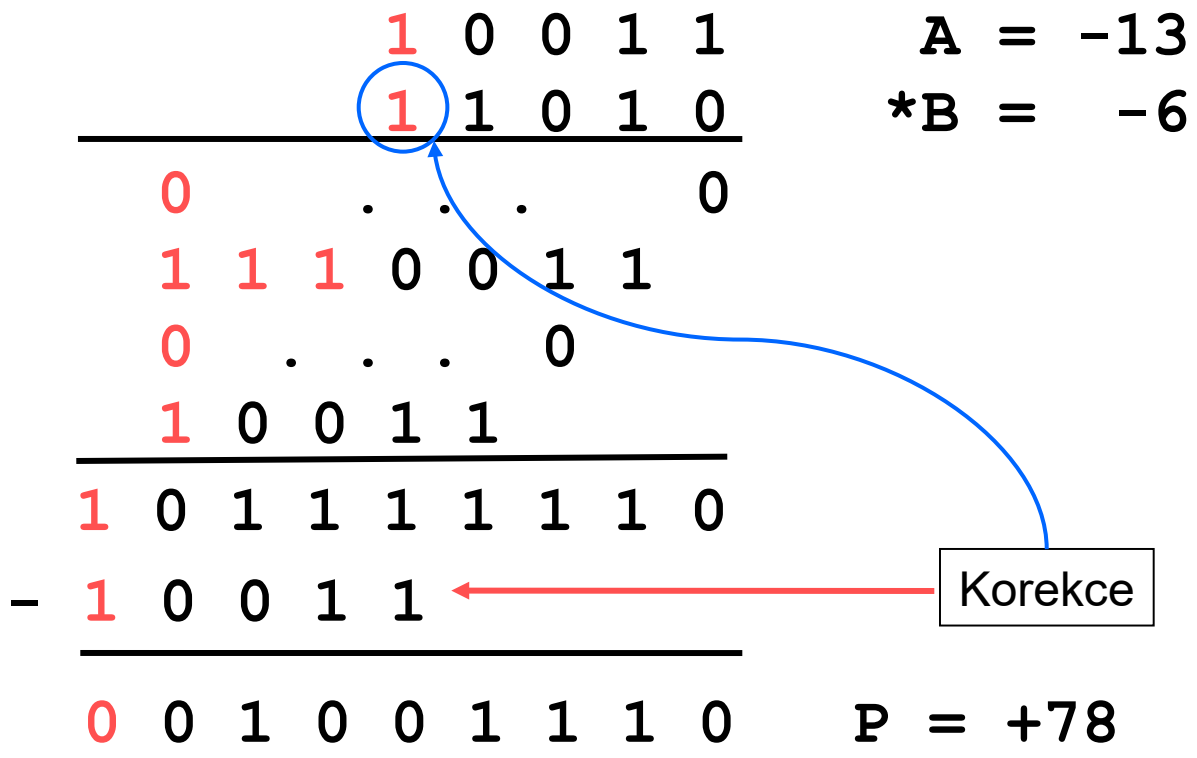
$$P = B * A = -b_{n-1}2^{n-1} A + \left(\sum_{j=0}^{n-2} b_j 2^j\right) A$$

- Korekční člen pro doplňkový kód je roven:

$$P = P_{n-1} - b_{n-1}.2^{n-1}.A$$

tj. je li b záporné (bn-1<>0) pak korigujeme a to odečtením A v nejlevější části výsledku

Záporný násobitel – příklad (s korekcí)



- 4* součet 2N bitvého čísla + korekce

Záporný násobitel – příklad (bez korekce)

-4*-14 na 4b+1b na znaménko => 2x5b => zn. až do 10b !!!

```
(-4)                1111111100
(-14)               1111110010
                          00000
                     1111111100
                        00000
                       00000
                  1111111100
                 1111111100
                1111111100
               1111111100
              1111111100
              0000111000b      (+56d)
```

Nutnost šířit znaménko dílčích součinů a sčítat dílčí součiny i pro bity rozšířeného znaménka násobitele.

- 10* součet 2N bitvého čísla (-> nevýhodné → boothův alg.)

Boothův algoritmus (2,1)

(pozn. (2,1) = 2 testované bity, postup po 1bitu)

- Průběžné překódování násobitele – každá skupina jedniček je nahrazena dvěmi jedničkami, jejichž váhy mají opačná znaménka.

Překódování (překádovávaný bit, bit vpravo → kód

```
0   0 →    0
0   1 →    1
1   0 →   -1
1   1 →    0
```

Násobitel, vyjádřený ciframi {0,1} je převeden do soustavy s ciframi {-1,0,1} - ! Redundance !

Příklad: 14 = 01110(0) → 1 0 0 -1 0, -14 = 10010(0) → -1 0 1 -1 0 (=-16+4-2)

- Pro zjištění počátku a konce skupiny jedniček se testují 2 sousední bity násobitele. Postup metody je po 1 bitu.

Výhoda: Metoda uniformním způsobem umožňuje násobení záporných čísel zobrazených v doplňkovém kódu.

Nevýhoda metody (2,1): Postup pouze po jednom bitu – počet parciálních součinů je stále n

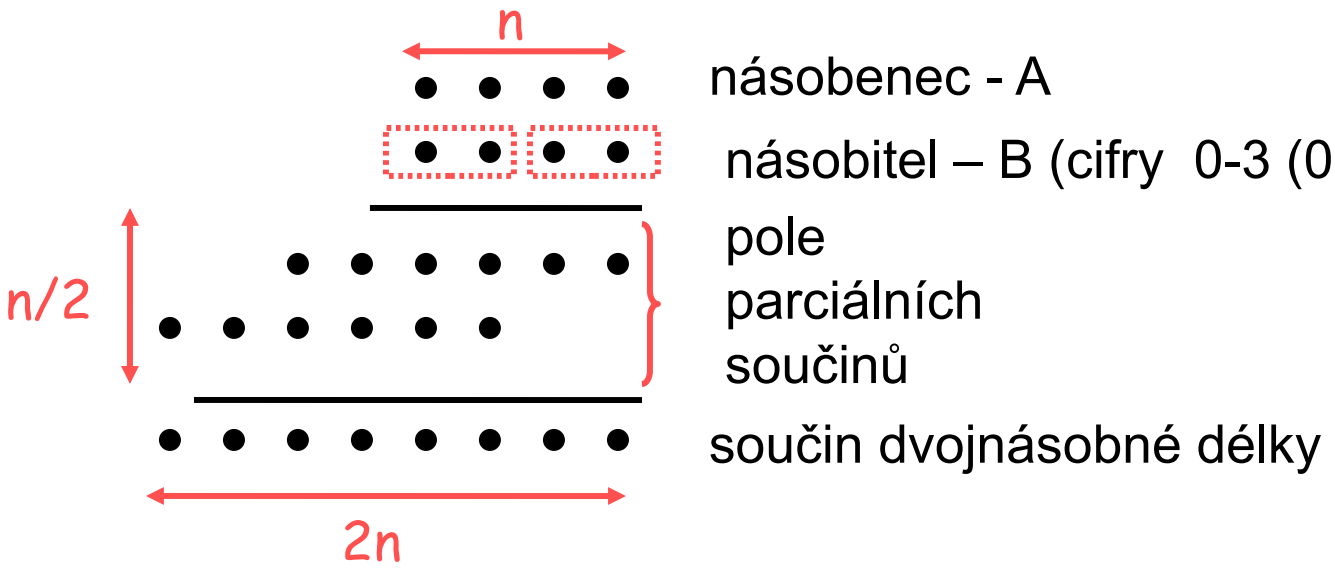
```
      111 00  (-4)
     -101-10  (-14)
-------------------
            0  (0)
     00100     (+4)
  1111100      (-4)
        0      (0)
  00100        (+4)
-------------------
1000111000 (=56)
```

Násobení v soustavě se základem > 2

- Počet kroků sekvenčního násobení závisí na počtu cifer operandů (násobitele)
- Provádíme-li násobení v soustavě 2^k, kde délka násobitele je n-bitů, je třeba sečíst $\lceil n/k \rceil$ parciálních součinů
- Parciální součiny nelze obecně v tomto případě generovat jednoduše, jako tomu bylo u binárního násobení (ale je možno si je přepřipravit)

Násobení se základem 4

Při použití základu 4 (k=2) pro násobení je třeba jen polovina operací součtu, probíhá tedy 2x rychleji



kde $P_{i+1} = P_i + 4^i.b_{i+1}||b_i A)$, kde $P_0 = 0$ a

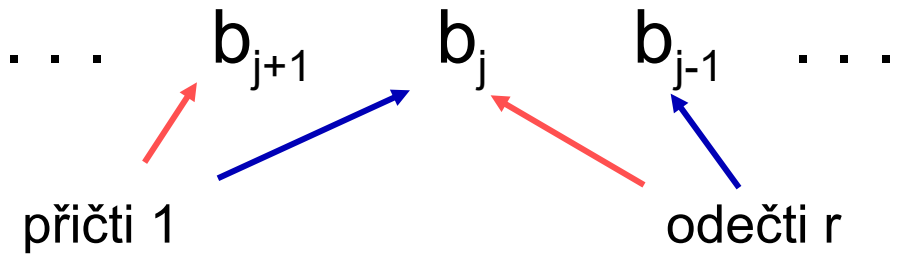
$$P_{n-1} = \left(\sum_{j=1}^{n-1} b_{n-j}\, 2^{-j}\right) A = B * A$$

Formování násobků násobence

- Metoda používá operandů 0*A, 1*A, 2*A, 3*A
- Lze je snadno vytvořit posuvy, kromě 3*A
  - výpočet v každém cyklu (3A = 2A + 1A) příliš pomalé!
  - výpočet předem a uložení do registru
  - Nebo náhrada 3A operandem 4A-A (tj. přenos do nejbližšího vyššího řádu násobitele) a –A … překódování násobitele tak, že použití „nepříjemného“ operandu odpadá

Překódování násobitele

- Pro základ soustavy r=4 mohou být cifry b_i [0,1,2,3] nahrazeny ciframi b_i z rozsahu [$^{-}$2,$^{-}$1,0,1,2]
- Tato transformace se provede tak aby algebraická hodnota násobitele B zůstala zachována



$$r^{(j+1)}b_{j+1} + r^j b_j = r^{(j+1)}(b_{j+1} + 1) + r^j(b_j - r)$$

Navzájem se kompenzují

Cíl překódování

- Odstranit „nepříjemné“ násobky násobence
  - 3A (5A, 6A, 7A, … pro vyšší základy)
- Maximalizovat počet nul

  0111 1111 $\rightarrow$ 1000 000^{-}1
- Eliminovat možnost výskytu párů cifer 11 nebo $^{-}$1^{-}1

  0111 0111 $\rightarrow$ 100^{-}1 100^{-}1

  $\rightarrow$ 1000 $^{-}$100^{-}1

Metoda překódování

- Označíme-li cifru modu m_j a cifru kódu b_j', pak platí:

$$r^{(j+1)}b_{j+1}' + r^{j}b_j' + r^{(j-1)}b_{j-1}' =$$

$$r^{(j+1)}(b_{j+1}+m_{j+1}) + r^{j}(b_j - r.m_{j+1} + m_j) + r^{(j-1)}(b_{j-1} - r.m_j)$$

kompenzace kompenzace

- Dostaneme $b_j' = b_j - r.m_{j+1} + m_j$

Nová cifra Stará cifra

Násobení ve vyšších soustavách

Lze provést překódování pro základ 8, 16 nebo 32?

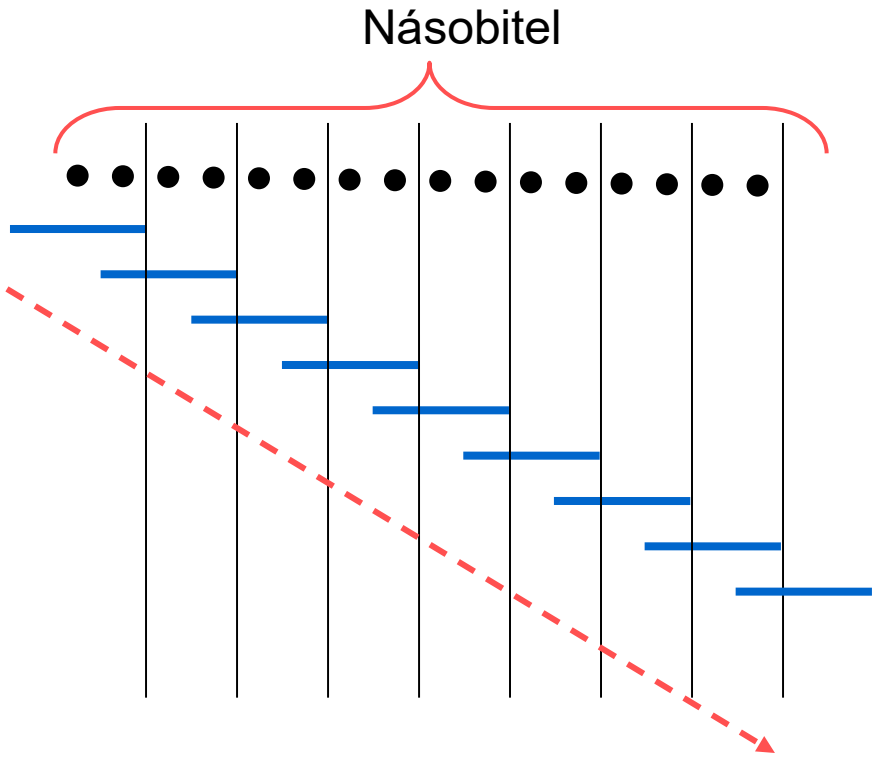
-7 až 7 mnoho "obtížných" násobků (-3,3,-5,5,-6,6,-7,7)
max. redundance
-6 až 6 mnoho "obtížných" násobků (-3,3,-5,5,-6,6)
-5 až 5 některé "obtížné" násobky (-3,3,-5,5)
-4 až 4 únosný počet "obtížných" násobků (-3,3)
min. redundance

- Řešením je vytvořit požadované násobky předem (za cenu určité ztráty času)

Boothův algoritmus (3, 2)

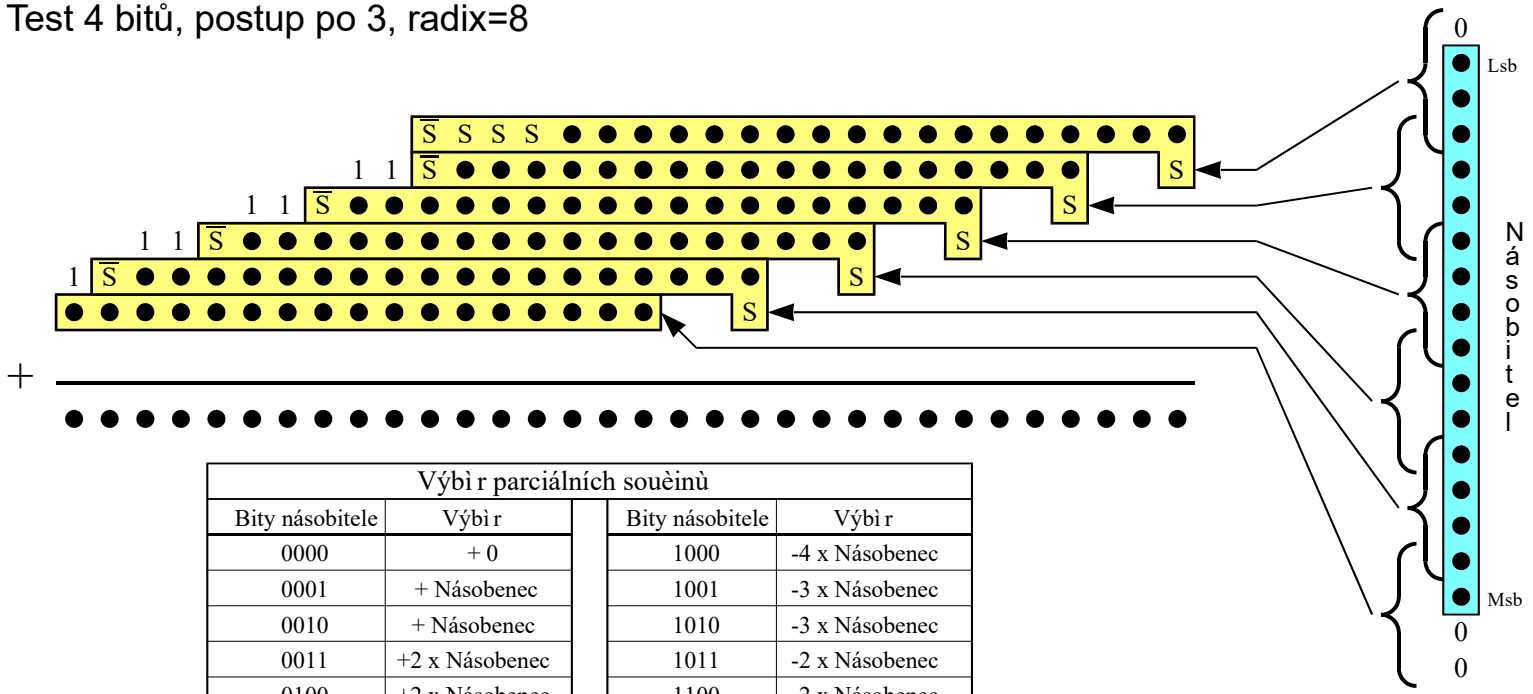
- Lze využít i jiné sady „jednoduchých“ násobků

Bity násobitele	Operace
000	-
001	+2*Násobenec
010	+2*Násobenec
011	+4*Násobenec
100	-4*Násobenec
101	-2*Násobenec
110	-2*Násobenec
111	-



Boothův algoritmus (4, 3)

Test 4 bitů, postup po 3, radix=8



Výbìr parciálních souèinù				
Bity násobitele	Výbìr		Bity násobitele	Výbìr
0000	+ 0		1000	-4 x Násobenec
0001	+ Násobenec		1001	-3 x Násobenec
0010	+ Násobenec		1010	-3 x Násobenec
0011	+2 x Násobenec		1011	-2 x Násobenec
0100	+2 x Násobenec		1100	-2 x Násobenec
0101	+3 x Násobenec		1101	- Násobenec
0110	+3 x Násobenec		1110	- Násobenec
0111	+4 x Násobenec		1111	- 0

S = 0 je-li parciální souèin kladný (levá strana tabulky)

S = 1 je-li parciální souèin záporný (pravá strana tabulky)

Test 4 bitů, posuv o 3 bity

Redukuje se počet parciálních součinů

Operand (3*násobenec) nutno získat vyhrazeným krokem !

Boothův algoritmus - obecně

Boothovo překódování lze definovat pro skupiny bitů libovolné velikosti.
Obecný postup je takový:
V každé skupině zopakujeme nejvyšší bit,
K této hodnotě přičteme k nejnižšímu bitu nejvyšší bit ze skupiny vpravo.
Výsledné číslo pak považujeme za relativní číslici v doplňkovém kódu.

Příklad:
-10 = 110110 = (radix 4 = po 2bitech) = 111+0, 001+1, 110+0 = -1 2 -2
110110 = (radix 4 = po 2bitech) = 111+0, 001+1, 110+0 = -1 2 -2
(zpet: -1 2 -2 → -1*4^2 + 2*4^1 + -2*4^0 = -16+8-2 = -10)
-600 = 1 1 0 1 1 0 1 0 1 0 0 0 = (radix 8 = po 3 bitech) = -1 -1 -3 0

Pozn. Číslo je třeba doplnit vlevo znaménkovými bity tak aby počet bitů byl soudělný s posunem. Např:
10 = 1010 = (radix 8 = po 3bitech) = 001 010 = 0001+0 0010+0 = 1 1
14 = 1110 = (radix 8 = po 3bitech) = 001 110 = 0001+1 1110+0 = 2 -2
(zpet: 2 -2 → 2*8^1 + -2*8^0 = 16-2 = 14)

Paralelní algoritmy násobení

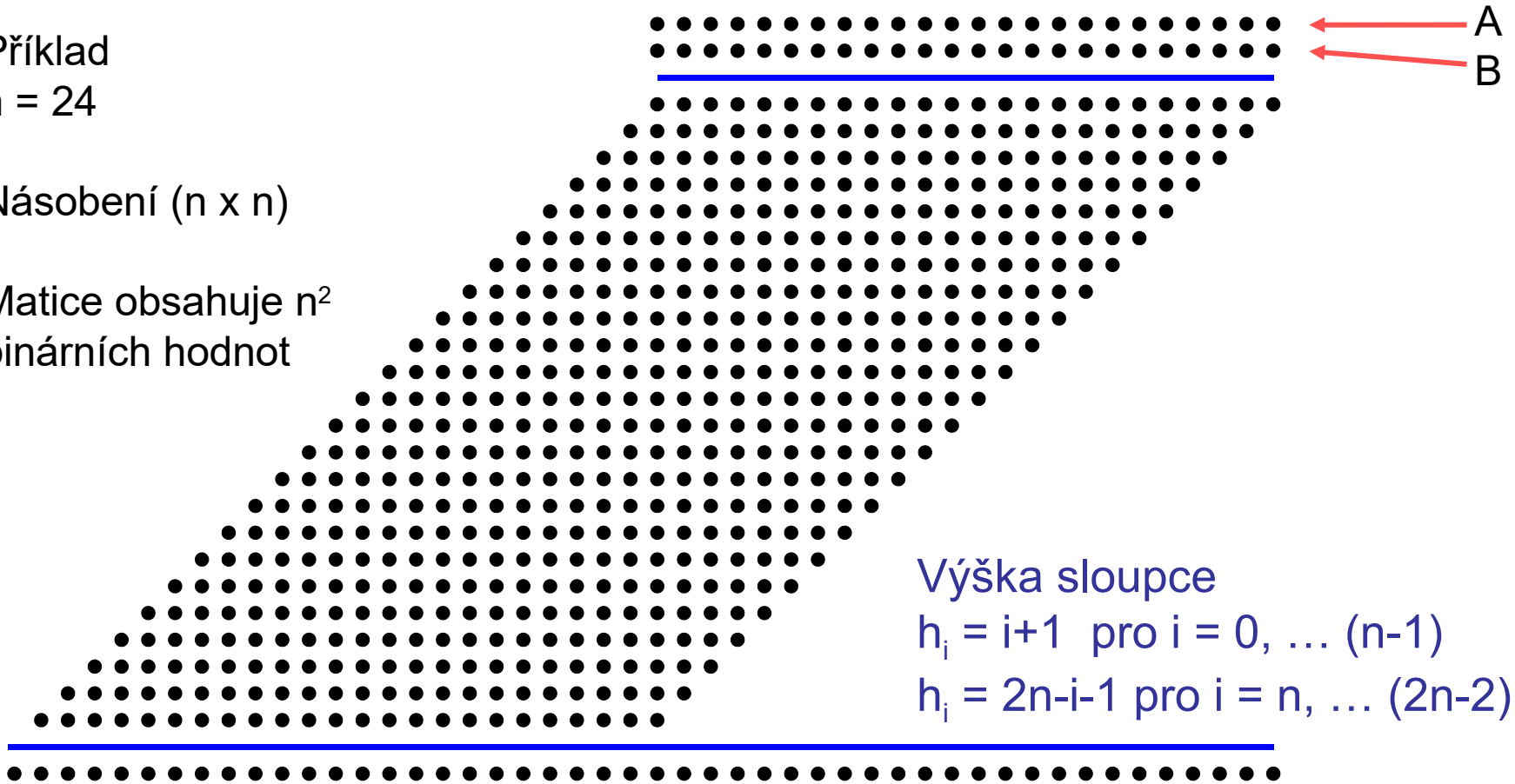
- Předchozí sekvenční metody vyžadovaly dokončení součtu z předchozího cyklu dříve, než mohl být zahájen nový cyklus.
- Součet parciálních součinů nemusí být dokončen během aktuálního cyklu => vzniká prostor pro zrychlení metody
- Iterační techniky násobení

Parciální součiny

Příklad
n = 24

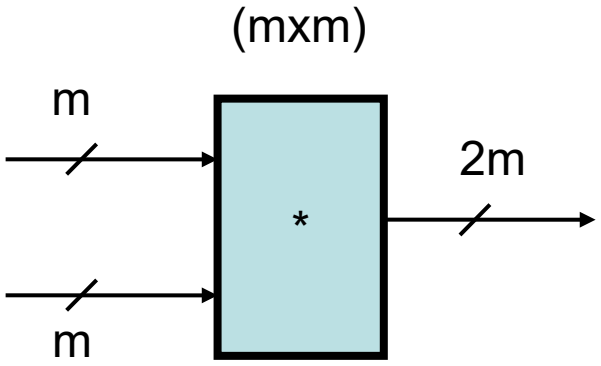
Násobení (n x n)

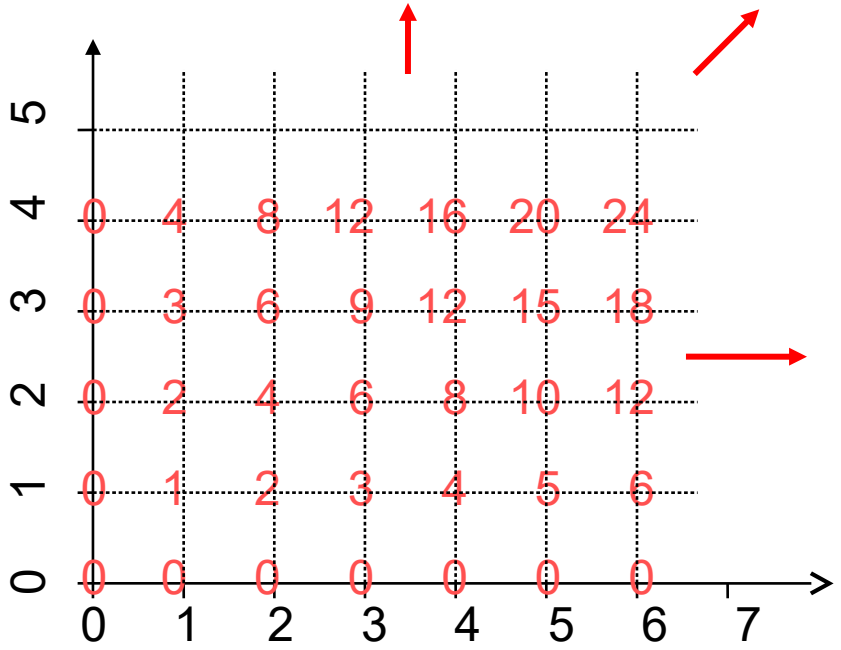
Matice obsahuje n^2
binárních hodnot



Vytváření parciálních součinů

- Vytváření parciálních součinů v soustavě o základu 2^m
- Použití dílčích násobiček (tvoří součin cifer)





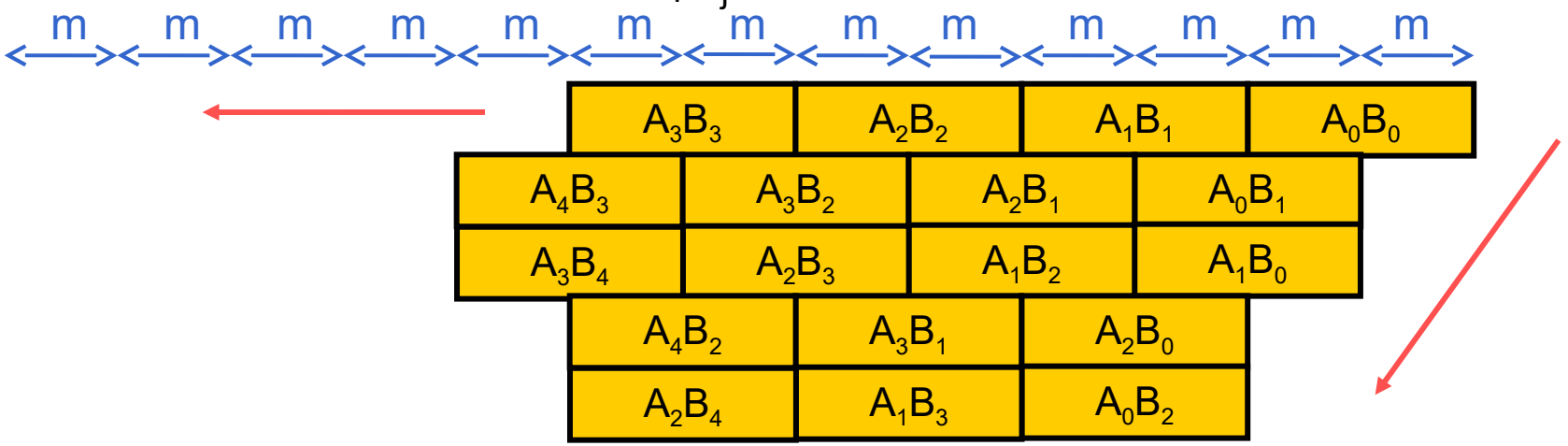
Skládání parciálních součinů

$A = (A_{k-1}, \ldots, A_1, A_0)$ $A_i = 0,1, \ldots, 2^m-1$

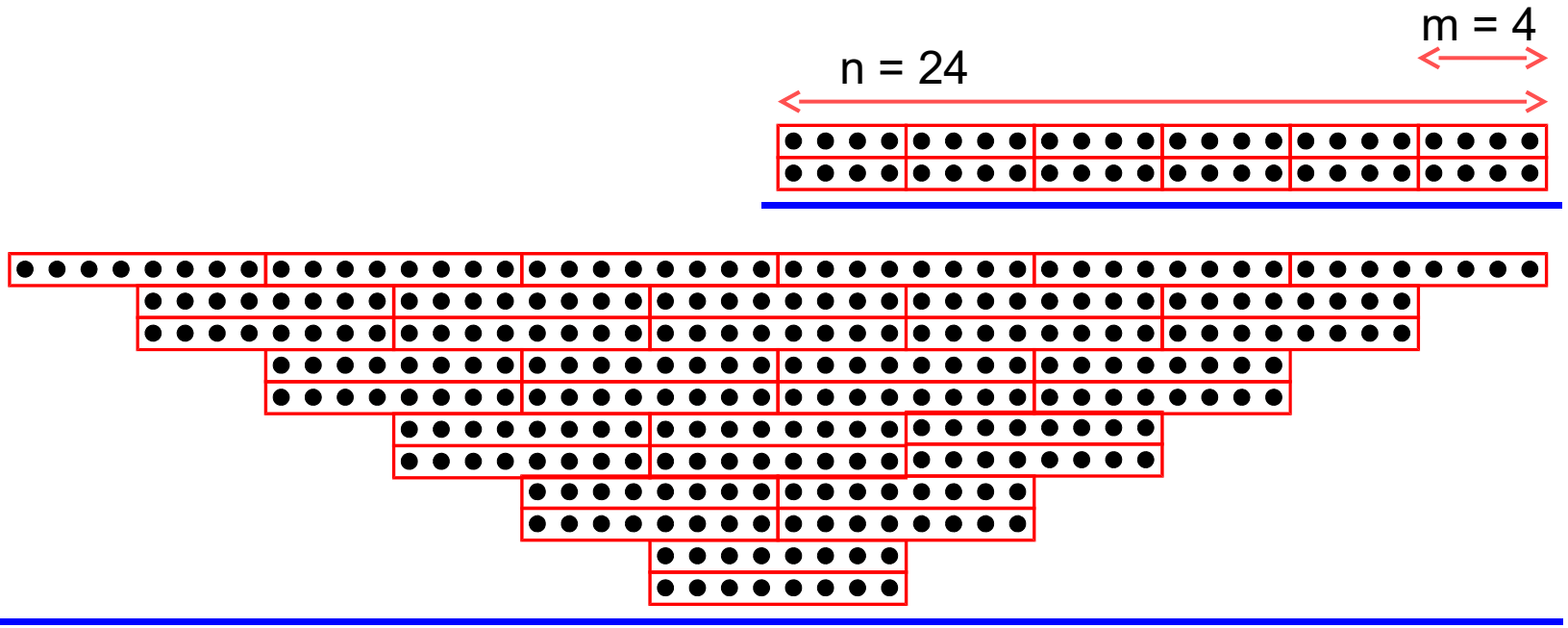
$B = (B_{k-1}, \ldots, B_1, B_0)$ $B_i = 0,1, \ldots, 2^m-1$

Pro délku slova **n** vznikne $(\lceil n/m \rceil)^2$ výstupních segmentů o šířce 2m.

Nejnižší bit segmentu A_iB_j začíná na řádu $2^{m(i+j)}$



Redukce parciálních součinů pomocí dílčích násobiček



Násobení v soustavě 2^m

Výška sloupce je rovna:

$$h_i = 2\left\lfloor \frac{i}{m} \right\rfloor + 1 \quad i = 0,\ldots,(n-1)$$

$$h_i = 2\left\lfloor \frac{2n-i-1}{m} \right\rfloor + 1 \quad i = n,\ldots,(2n-2)$$

Redukce výšky matice zobecněnými sčítačkami

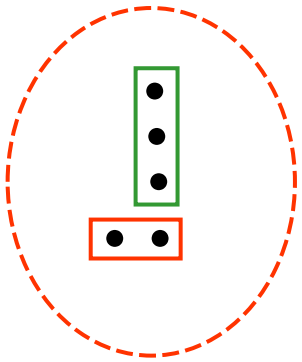
Formální popis paralelního čítače: $(c_{k-1}, c_{k-2}, c_{k-3}, \ldots, c_1, c_0, d)$

c_i … výška i-tého vstupního sloupečku

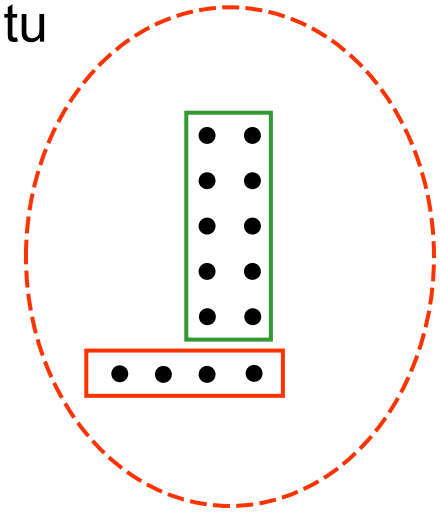
d … šířka výstupního segmentu

k … počet vstupních sloupců

Redukce 3:2



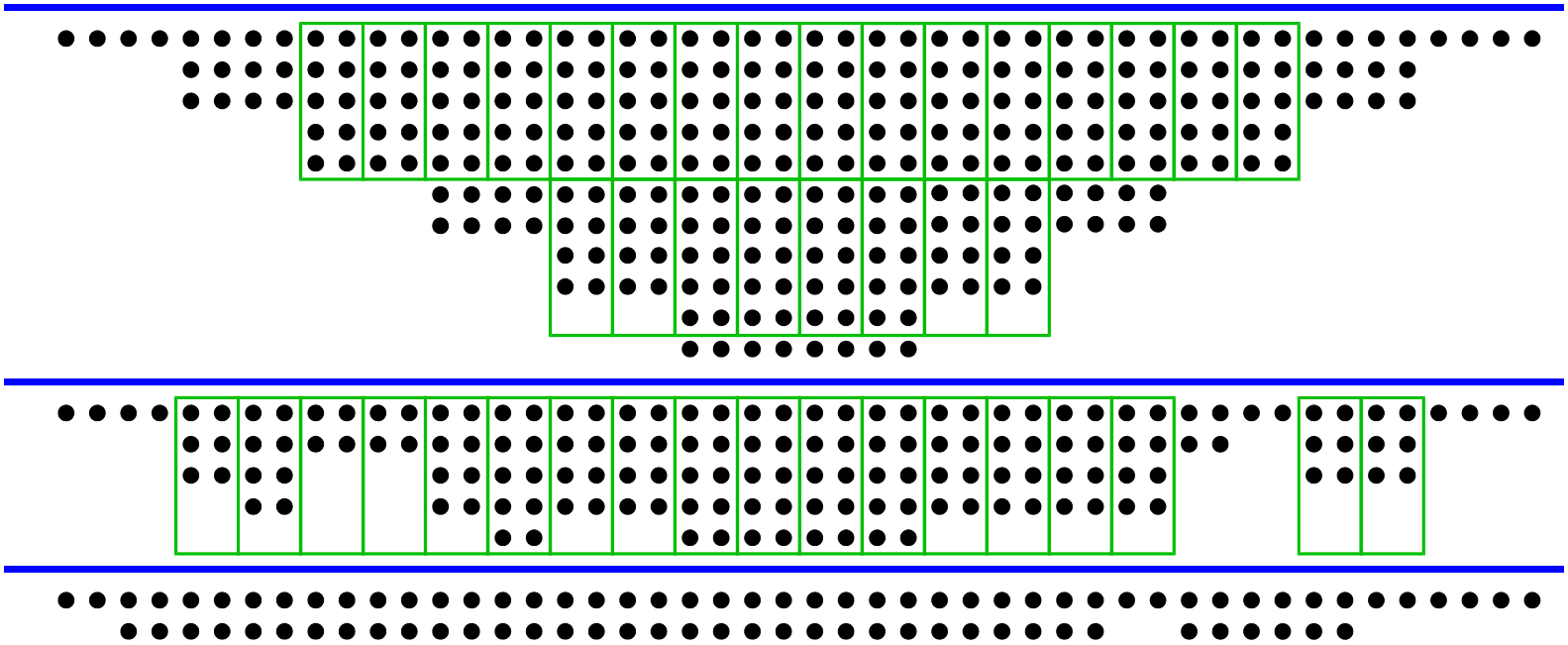
Paralelní čítač (3,2)



Paralelní čítač (5,5,4)

Redukce 10:4

Redukce parciálních součtů paralelními čítači (5,5,4)



Výsledný součin je získán po sečtení posledních dvou řádek sčítačkou (CLA, RCA, …)

Počet redukčních úrovní

- Budeme zatím uvažovat jen čítače s konstantní výškou vstupních sloupců $c_i = r$ (konstanta) $\forall$ i, i=0,1, …, k-1
- Pro správnou činnost musí platit nerovnost

  $2^d - 1 \geq r.(2^k - 1)$
- Čítače s plně využitým výstupním polem vyhovují rovnosti

  $r = (2^d - 1)/(2^k - 1)$

Za jakých podmínek vzniká z pásu konstantní výšky opět pás konstantní výšky?

$d = k.s$ kde $s \in N$ (d=počet výst.bitů,k=počet vst.sloupců)

Počet redukčních úrovní

- Jedna vrstva paralelních čítačů redukuje souvislý pás o výšce r na výšku s
- Při vyšetřování budeme postupovat opačně. Vytvoříme tzv. redukční posloupnost paralelního čítače daného typu.

$$l_0 = s$$

$$l_1 = r$$

$$\ldots$$

$$l_{j+1} = r.\lfloor l_j/s \rfloor + l_j \bmod s$$

- Redukční posloupnost dovoluje určit počet úrovní redukce, které je nutno pro danou výšku pásu binárních vstupů zařadit, aby výstupní pás měl výšku s.

Redukční posloupnosti vybraných čítačů

Typ paralelního čítače:

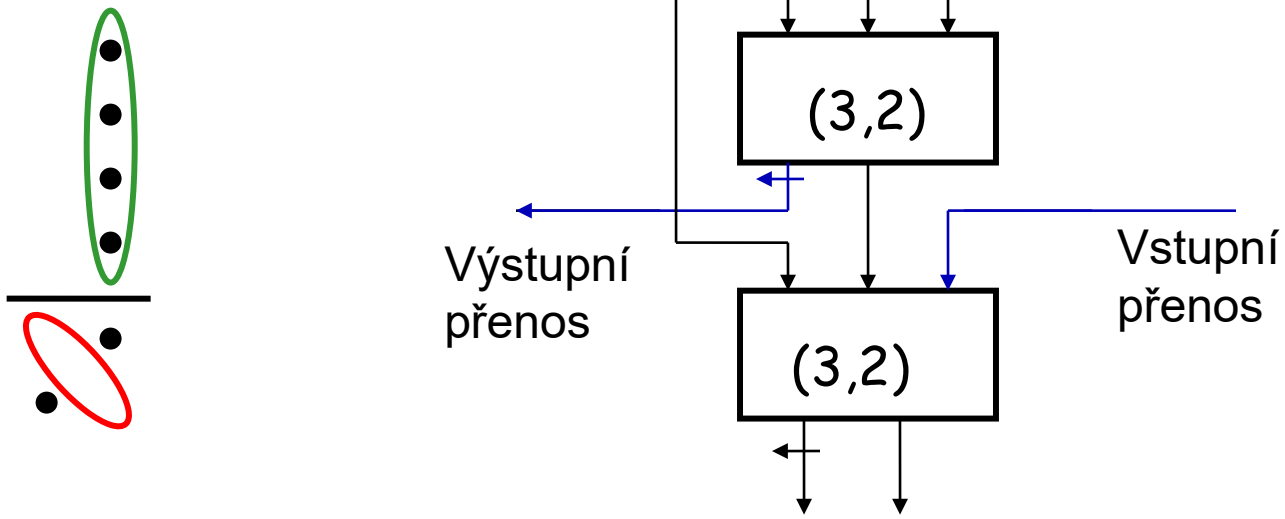
(3,2)	r=3, s=2	2, 3, 4, 6, 9, 13, …
(7,3)	r=7, s=3	3, 7, 15, 35, …
(5,5,4)	r=5, s=2	2, 5, 11, 26, 65, …

Implementace paralelních čítačů

- Obecné paralelní čítače
  - sčítají váženě jedničky v k-bitových sloupcích
    - plně vytížené - (3,2), (7,3), (15,4), (5,5,4), (2,2,2,3,5)
    - částečně vytížené – (10,4), (3,3,3,3,6)
- Specializované paralelní čítače
  - sčítají váženě jedničky v k-bitových sloupcích plus j interních vstupních přenosů. Generují výstupní pole a j interních výstupních přenosů
    - (4,2), (7,2), (11,2)

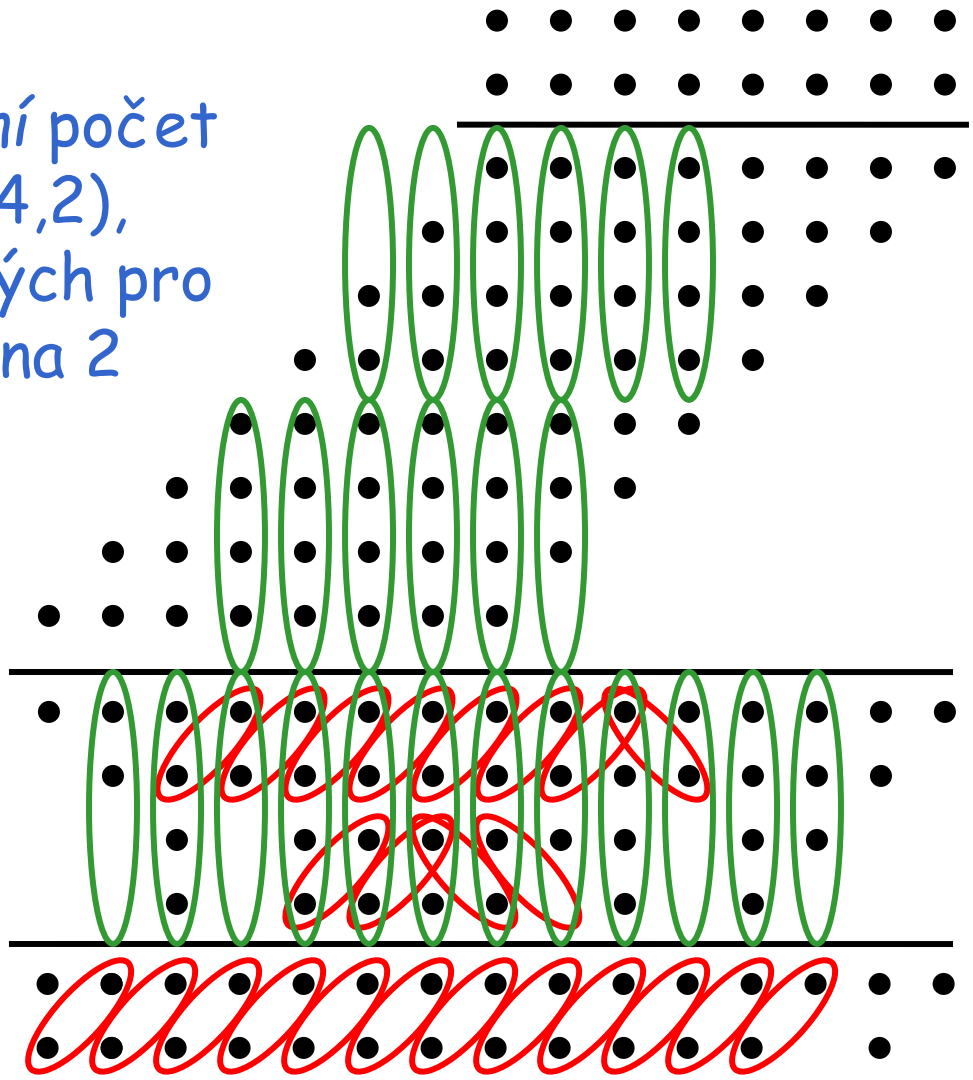
Čítač (4,2)

- postavený ze dvou čítačů (3,2), ale s 1 interním vstupním „přenosem“ a 1 interním výstupním „přenosem“ (pozn. šipka v obrázku značí vyšší bitový řád)



Redukce násobičky 8x8

Jaký je *minimální* počet čítačů (4,2), potřebných pro redukci na 2 řádky?



násobenec
násobitel

pole parciálních součinů

Odpověď: 24

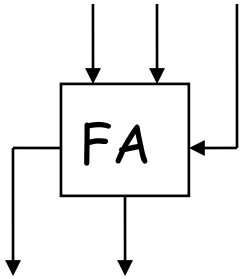
redukované pole parciálních součinů

vstup pro 14-bitovou rychlou sčítačku

Sčítačka typu CSA

- Obecně typ sčítačky nepropagující carry vlevo ale postupuje ke zpracování další vrstvě.

  (příklad: 1011 + 1001 = 2 0 1 2)
- Úplná sčítačka (FA), která postupuje přenos další vrstvě místo toho, aby přenos procházel napříč celou paralelní sčítačkou.

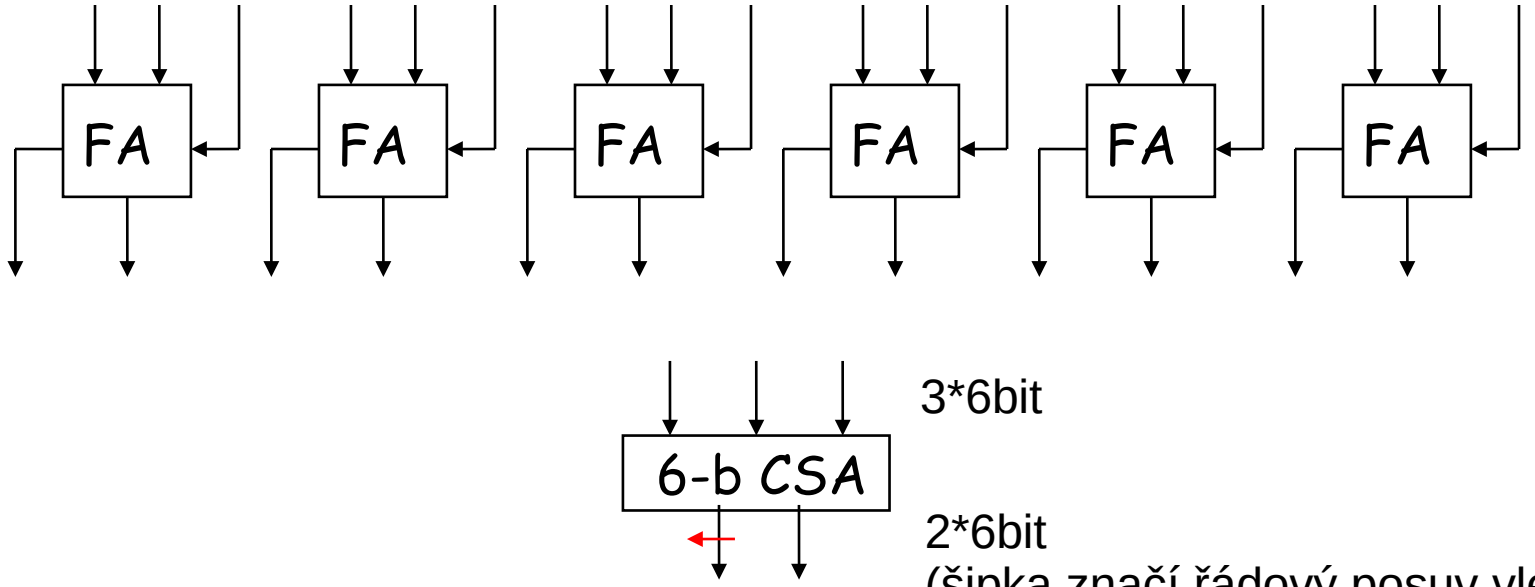


- Lze označit jako paralelní čítač typu (3,2).

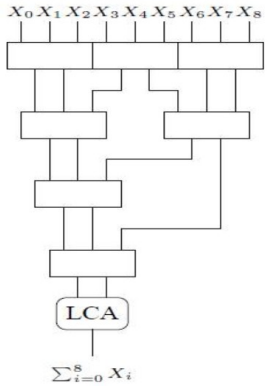
CSA … Carry Save Adder

Sčítačka CSA pro redukci matice parciálních součinů

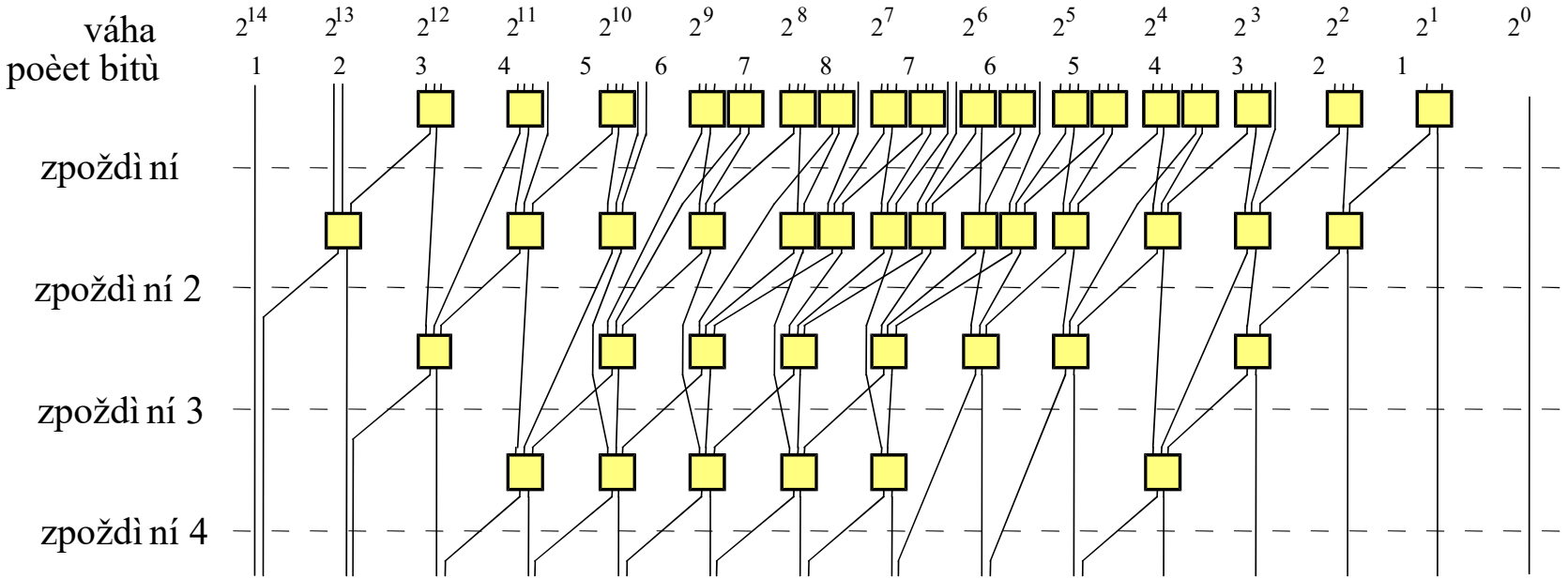
- 6 bitová CSA redukuje tři 6-bitová vstupní slova na jeden 6-bitový a jeden 6-bitový (posunutý) výstup



Násobička Wallace



Number of operands	Number of levels
3	1
4	2
$5 \leq k \leq 6$	3
$7 \leq k \leq 9$	4
$10 \leq k \leq 13$	5
$14 \leq k \leq 19$	6
$20 \leq k \leq 28$	7
$29 \leq k \leq 42$	8
$43 \leq k \leq 63$	9



Násobièka Wallace (strom. struktura)

Iterační aritmetická pole

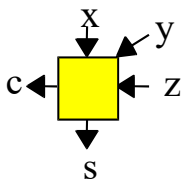
- Představují pole buněk stejného typu (nebo několika málo typů) - nejčastěji organizace 2D, které mají pravidelnou propojovací strukturu a jsou určena pro vykonávání aritmetických operací
- Funkce buněk nemusí být konstantní (implementace množiny operací jedním polem).

Základní typ násobičky – asychronní iterační pole

Asynchronní pole

Kritická cesta

$$A = \sum_{i=0}^{4} a_i.2^i \qquad B = \sum_{i=0}^{5} b_i.2^i \qquad P = \sum_{i=0}^{10} p_i.2^i$$



Buòka FA
(Full Adder)

s = x !$ y !$ z
c = x&y # x&z # y&z
2*c + s = x + y + z

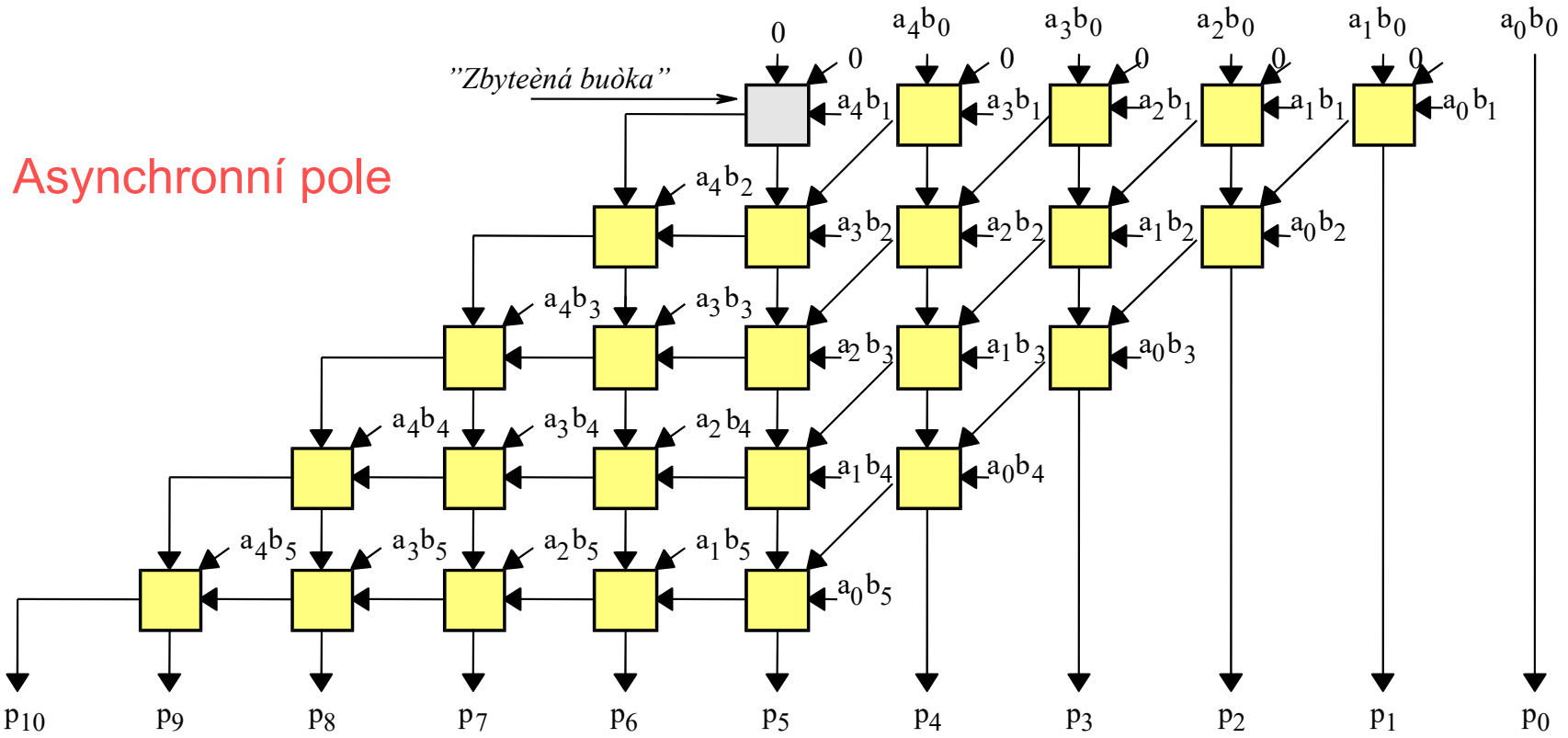
"Naivní" násobièka

Úpravy základního typu pole pro násobení

Cíle úprav:

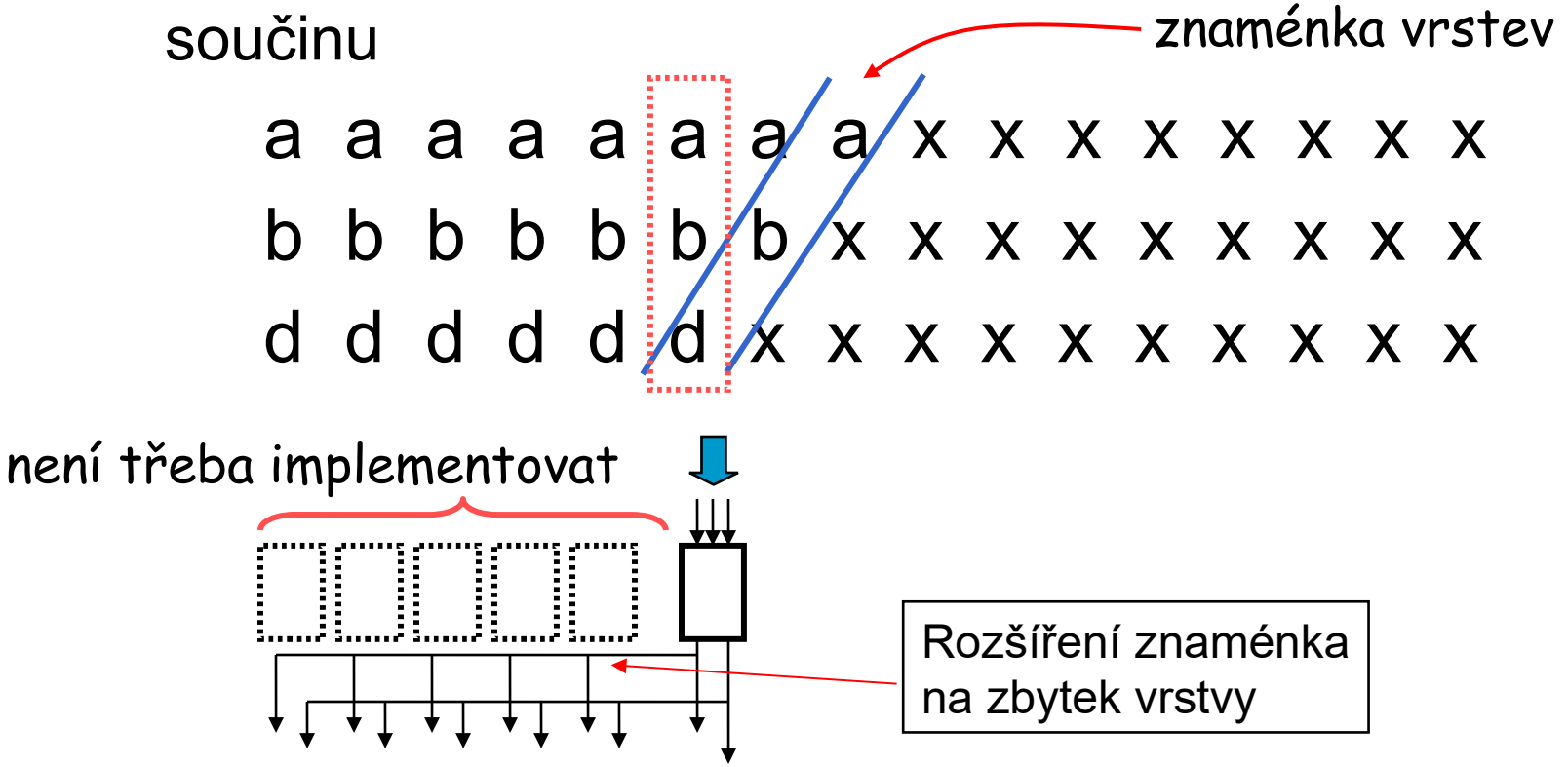
- Zkrácení kritické cesty signálu v poli
- Rozšíření o násobení záporných čísel
- Generace parciálních součinů uvnitř buněk
- Zavedení pipeliningu
- Zlepšení topologie pole pro snazší implementaci

Násobička s úpravou



Pole pro násobení se znaménkem

- Expanze znaménka každého parciálního součinu (a provedení závěrečné korekce) na šířku výsledného součinu



Násobička Baugh-Wooley

a_3 a_2 a_1 a_0

b_3 b_2 b_1 b_0

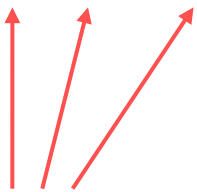
---

a_3b_0 a_2b_0 a_1b_0 a_0b_0

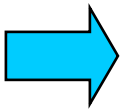
a_3b_1 a_2b_1 a_1b_1 a_0b_1

a_3b_2 a_2b_2 a_1b_2 a_0b_2

a_3b_3 a_2b_3 a_1b_3 a_0b_3



Členy a_ib_j … mají záporné váhy !!!



a_3 a_2 a_1 a_0

b_3 b_2 b_1 b_0

$a_3!b_0$ a_2b_0 a_1b_0 a_0b_0

$a_3!b_1$ a_2b_1 a_1b_1 a_0b_1

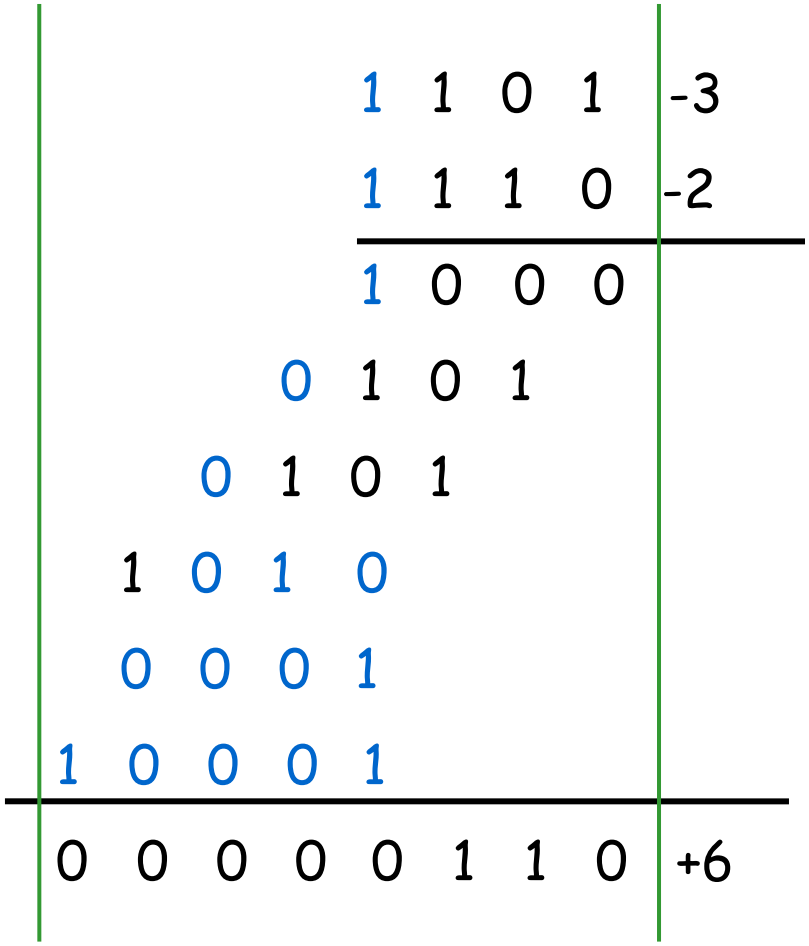
$a_3!b_2$ a_2b_2 a_1b_2 a_0b_2

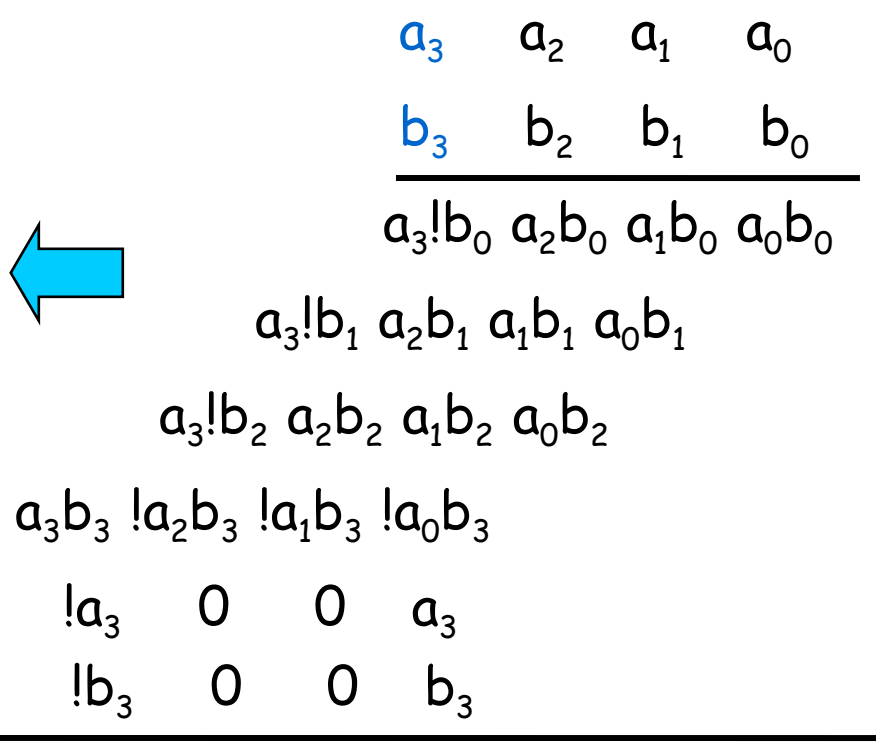
a_3b_3 $!a_2b_3$ $!a_1b_3$ $!a_0b_3$

$!a_3$ 0 0 a_3

1 $!b_3$ 0 0 b_3

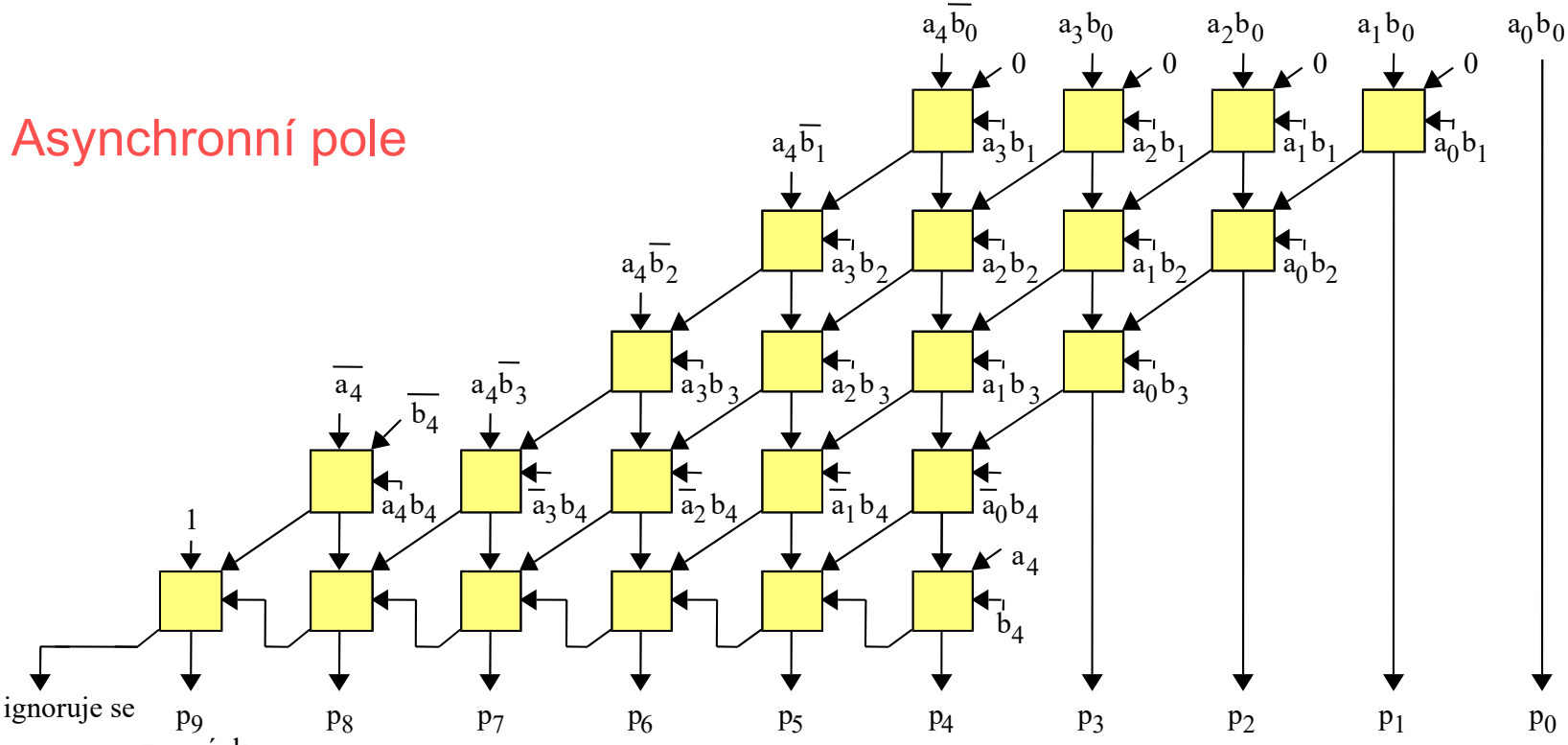
Násobička Baugh-Wooley - příklad





Násobička Baugh-Wooley

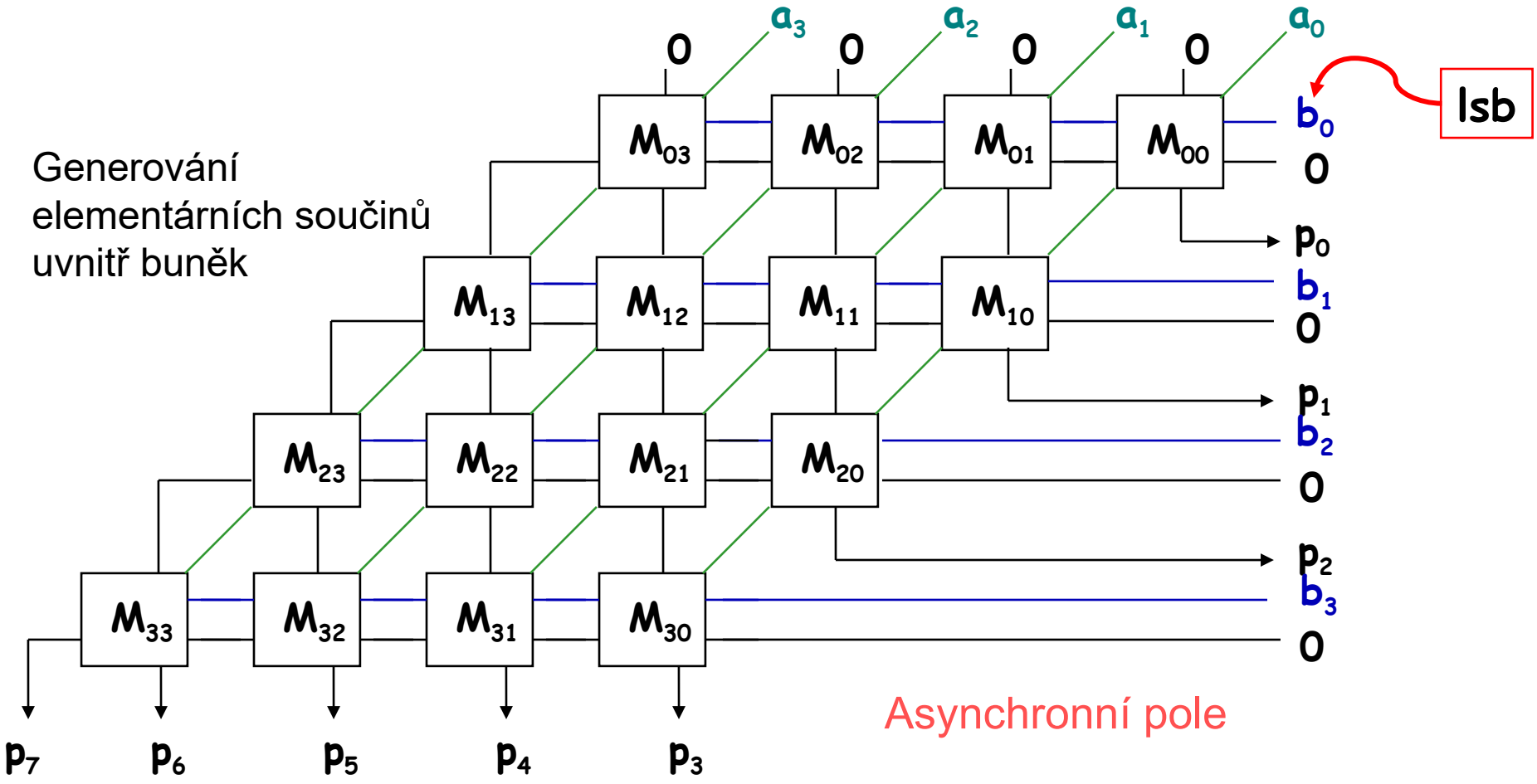
Asynchronní pole



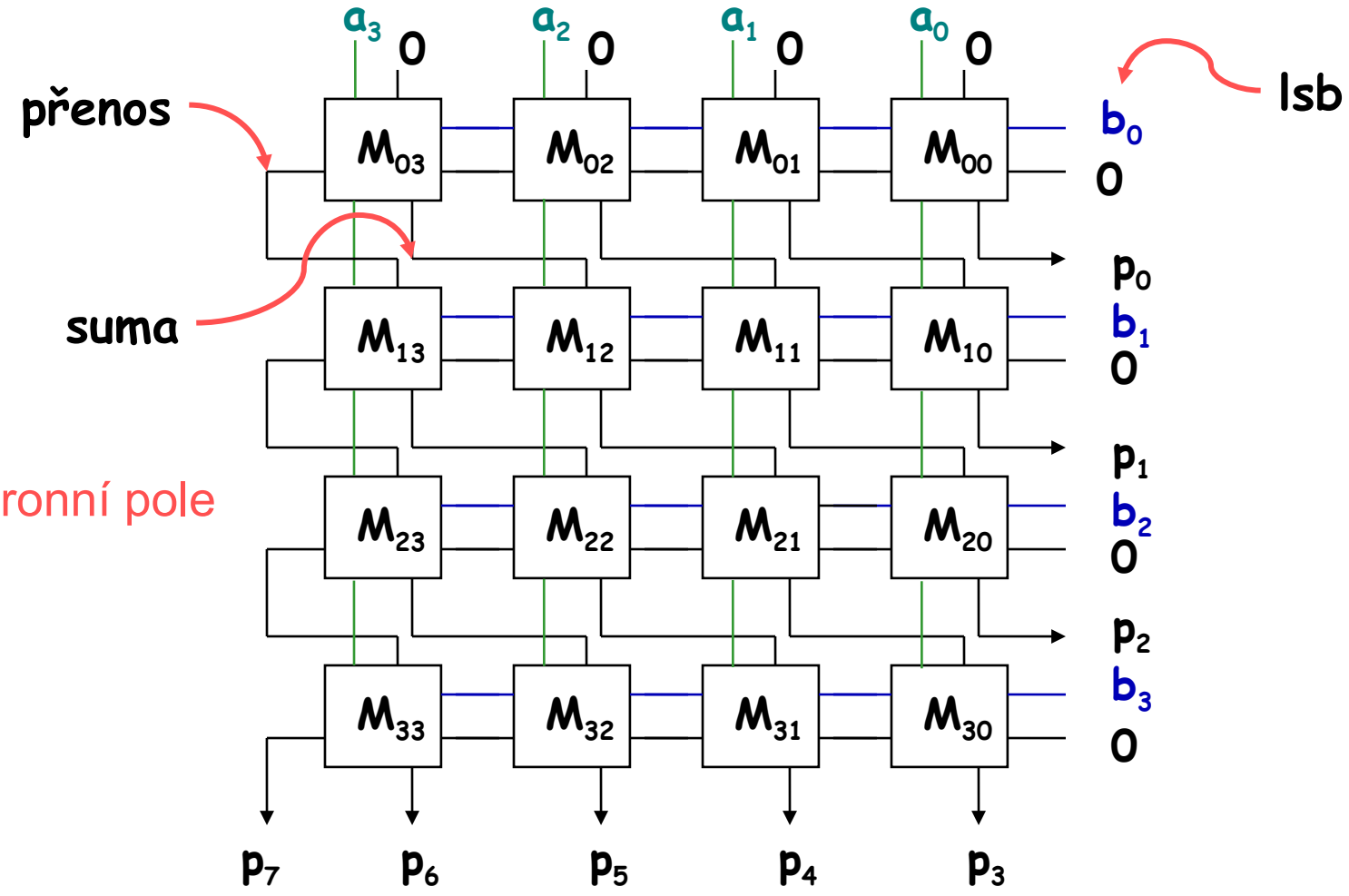
Násobièka Baugh-Wooley

Příklad pole pro násobení

Generování elementárních součinů uvnitř buněk



Čtvercový formát pole



Asynchronní pole

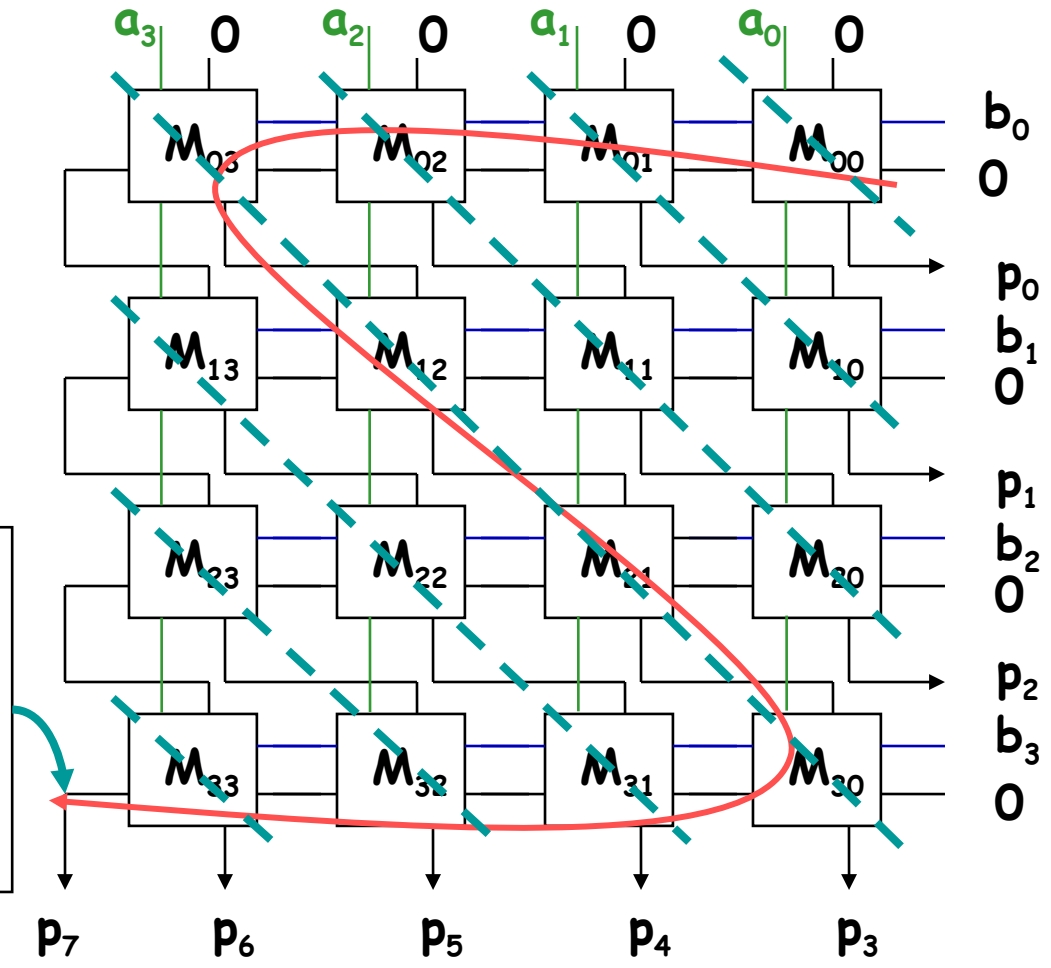
Časová zpoždění násobičky

Asynchronní pole

Délka kritické cesty:

$2n + n - 2$

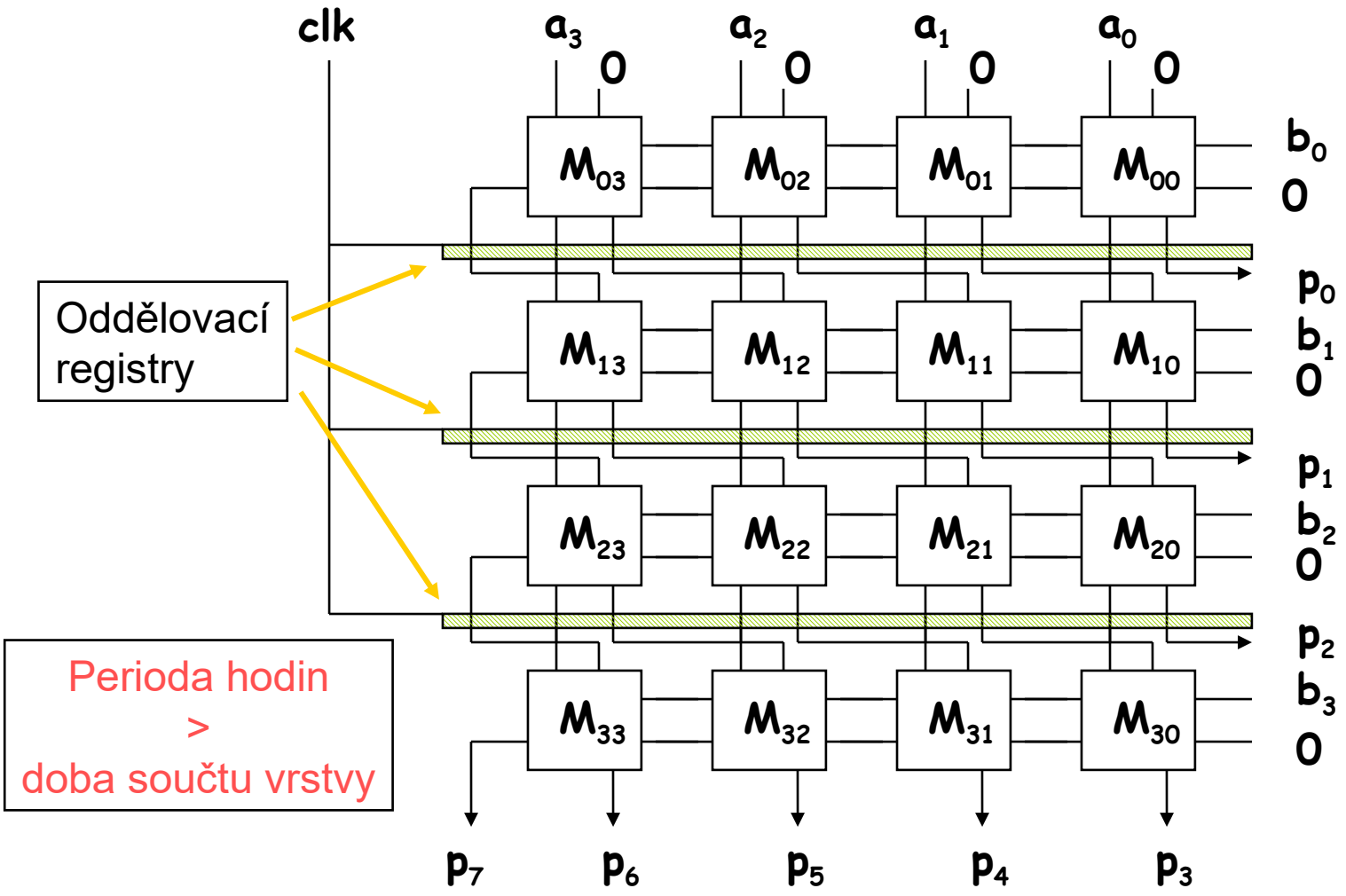
$= 3n - 2$



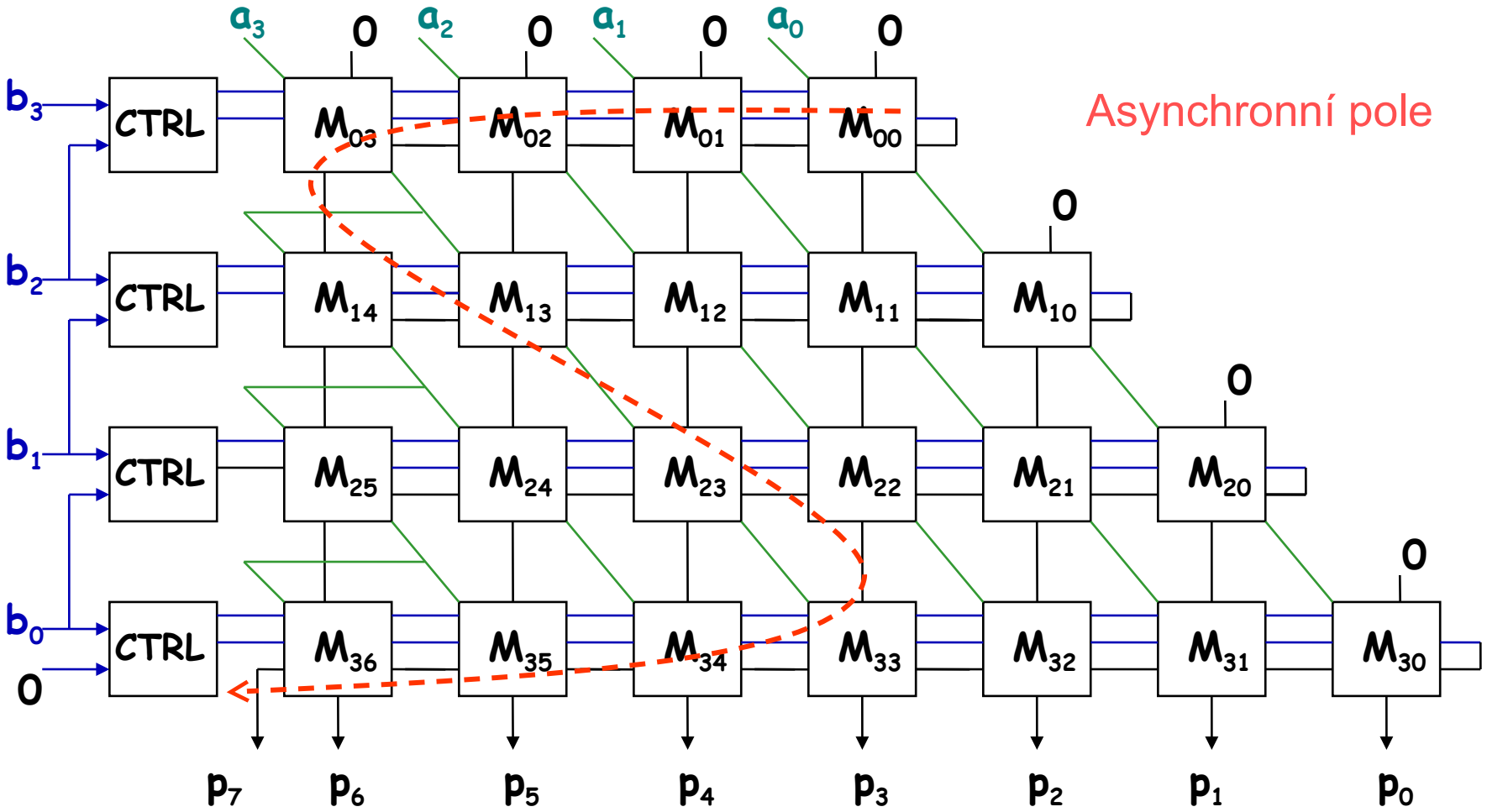
Pipelining

- Rozdělení prováděcí jednotky do stupňů s přibližně stejnou operační dobou
- Oddělení stupňů pomocí registrů (latch), aby bylo možno činnost separovat
- Hodinový kmitočet je určen nejpomalejším stupněm
  - Velmi rychlé hodiny
- Delší doba latence – doba, která uplyne od vložení vstupních operandů do výstupu výsledku pocházejícího od těchže operandů
- Lze dosáhnout velké šířky pásma, pokud se provádí velké množství (nezávislých) násobení. S každým taktem hodin se generuje jeden výsledek

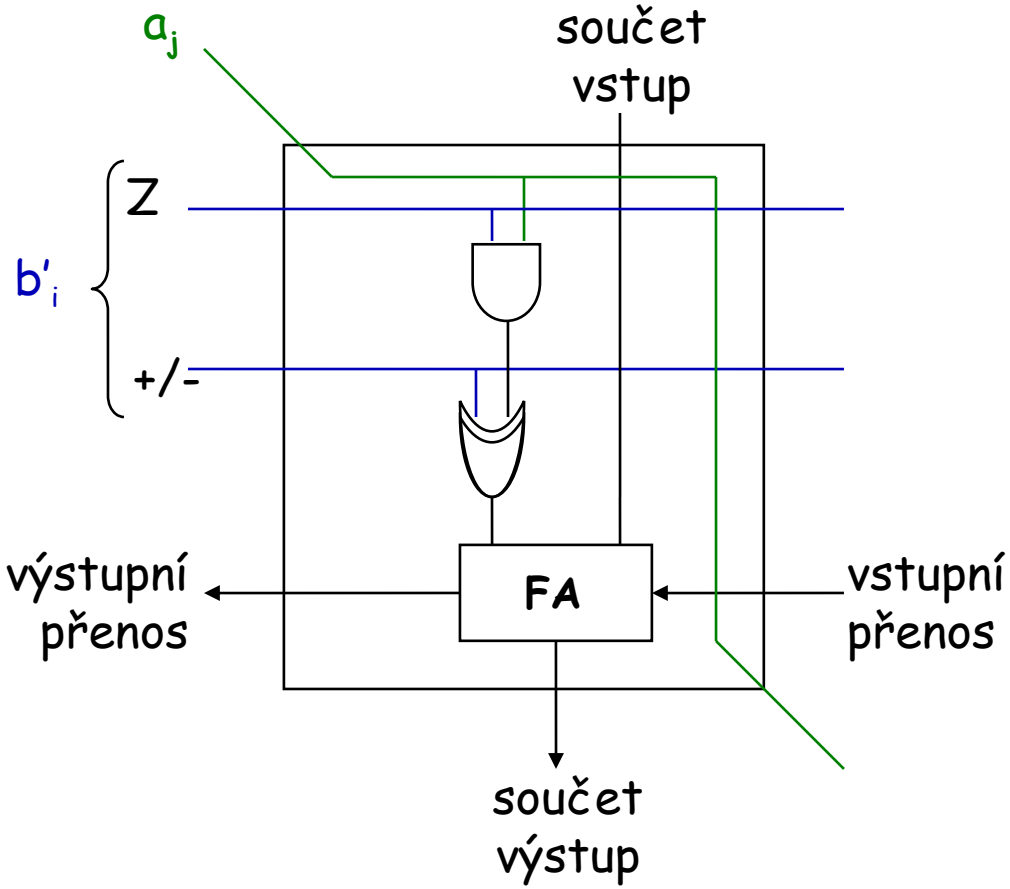
Zavedení pipeliningu



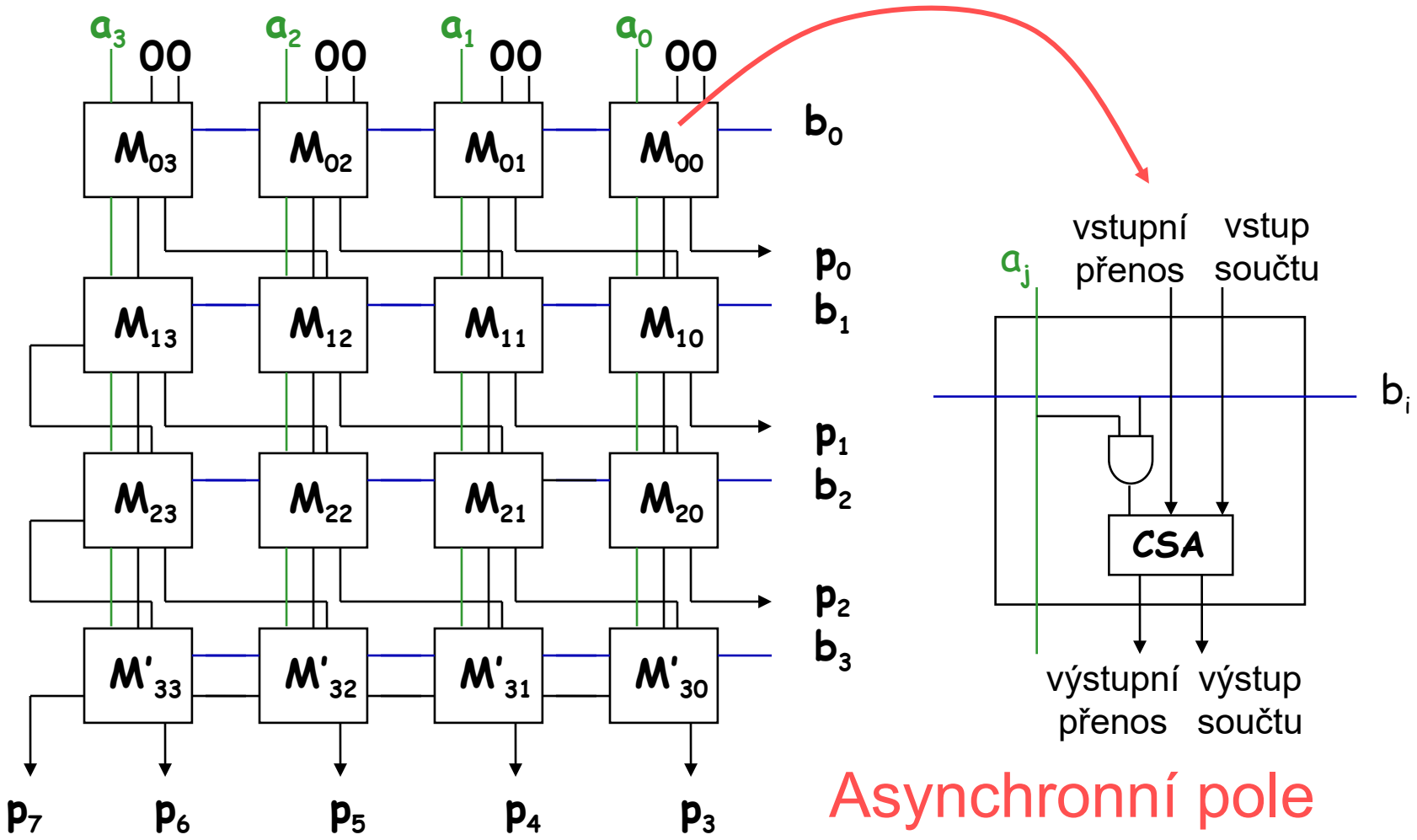
Pole pro násobení s překódováním (Booth)



Struktura buňky násobičky

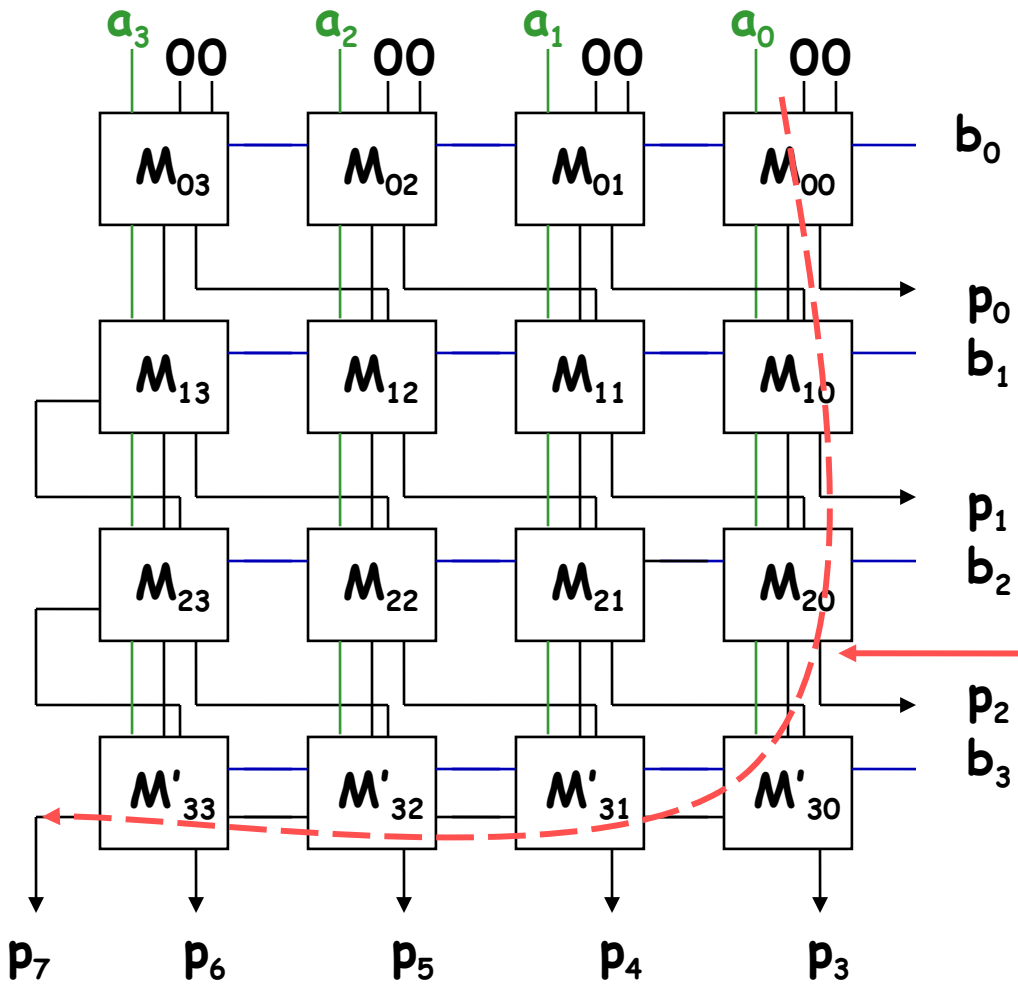


Pole pro násobení CSA



Asynchronní pole

Pole pro násobení CSA



Asynchronní pole
Carry se propaguje dolů

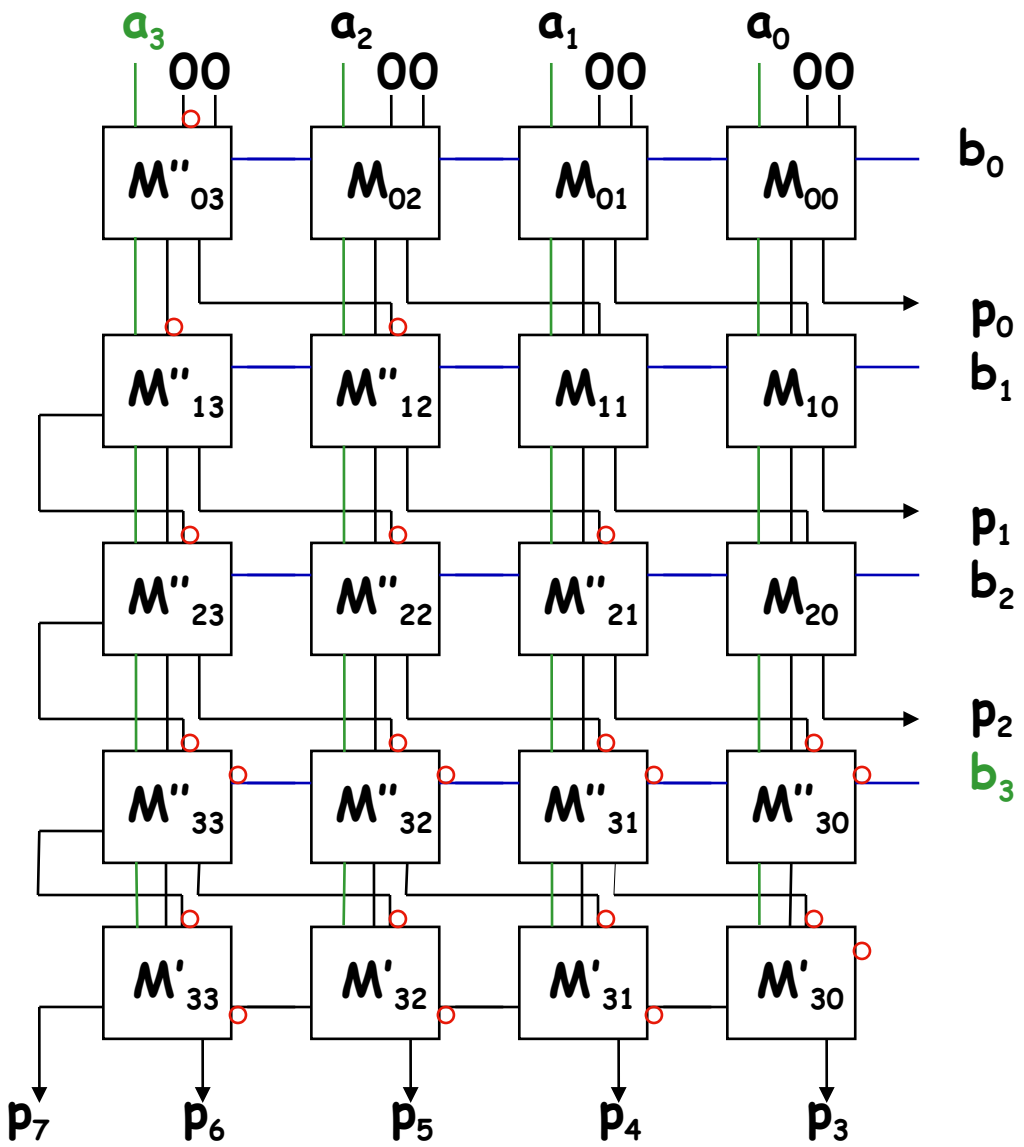
Délka kritické cesty

$n + n - 1$

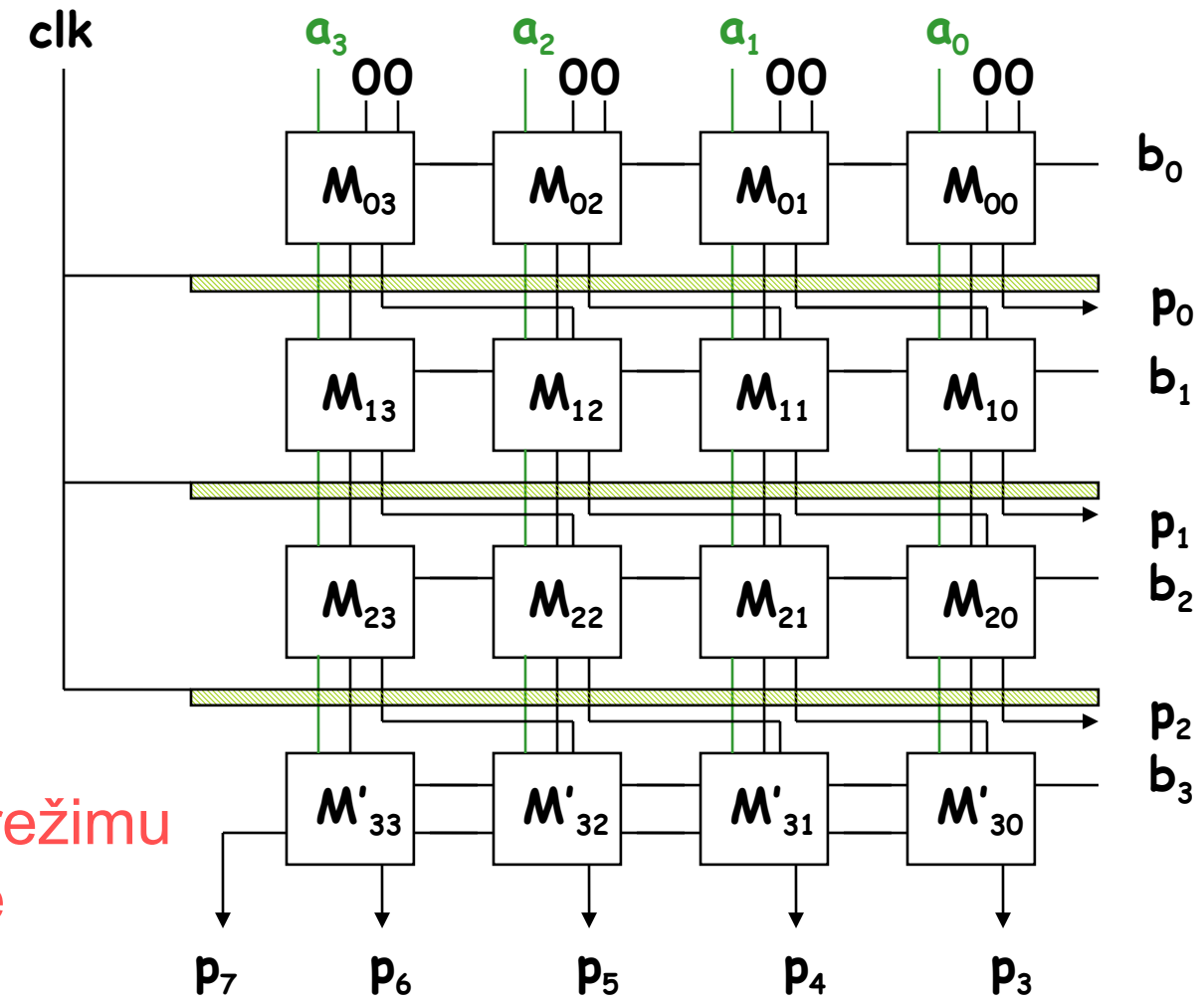
$= 2n - 1$

Pole CSA pro násobení se znaménkem

Asynchronní pole



Násobička CSA - pipelining



Pole v režimu pipeline

Pole pro násobení

- Nepatří mezi nejrychlejší implementace, ale
- Regulární struktura
- Používají pouze krátké vodiče k nejbližším sousedním buňkám
- Jednoduchý a efektivní návrh v technologii VLSI
- Velmi jednoduše lze zavést pipelinnig a tím podstatně zvýšit výkon
- Pole lze využít pro více matematických fcí