

Realizace logických a číslicových obvodů závisí:

♣ Složitosti

♣ Spotřebě

♣ Rychlosti odezvy

♣ Ceně

- Klasické obvody z řady 74xxx – maximálně 4 ÷ 6 IO
- Programovatelné obvody s popisem logické funkce
 - ❖ *Schématem*
 - ❖ *RTL* - obdoba realizace klasickými logickými obvody
 - ❖ *Jazyky HDL*
 - Behaviorální zápis (popis chování LKO a LSO)
 - Strukturální zápis (LO je rozdělen na funk. části)
- Procesorem a mikroprogramovatelným řadičem
 - ❖ *Mikroprogramování* – μ P řadiče, nyní řadiče PC.
 - ❖ *Assembler* (Strojový kód) – realizace instrukcemi μ P. Nyní – ovladače, knihovny jazyků, dílčí program s jasnou kontrolou zpoždění
 - ❖ *Jazykem C* – popis složitějších aplikací operacemi, které programovací jazyk poskytuje.

Číselné základy

V číslicových systémech standardně používáme tři číselné základy:

- **Základ 2** – se symboly 0 a 1 (někdy 0 a I)
 - ❖ Realizace operací v počítačích, μ P i PLD.
- **Základ 10** – se symboly 0,1,2,3,4,5,6,7,8,9
 - ❖ Běžně používaná soustava – zadávání hodnot do výpočetních systémů a k zobrazování vypočtené informace.
- **Základ 16** – se symboly 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
 - ❖ Používá se nejčastěji k vyjádření (bitových) hodnot v programech, k výpisu obsahu programových a datových pamětí.

Běžně nepoužívané soustavy

- **Základ 8** – se symboly 0,1,2,3,4,5,6,7
 - ❖ Lze použít k zadávání čísel v assemblerovských programech.
- **Základ 7** – Modifikovaný Bootův algoritmus násobení radix 32
- **Základ 11** – Modifikovaný Bootův algoritmus násobení radix 256

1. Metoda postupného odečítání mocnin základu

Koeficienty čísla získáváme od nejvyšší mocniny po nejmenší. Jednoduchá a často používaná.

Nevýhoda – doba převodu závisí na hodnotě převáděného čísla.

2. Metoda postupného dělení základem

Dělíme-li číslo N_M základem B , získáme celou část podílu a zbytek. Zbytek je roven koeficientu a_0 . Dalším dělením celé části získáváme další koeficienty v pořadí od nejmenšího a_0 k největšímu a_k .

Nevýhodná pro procesory bez dělení (podmíněné odečítání) nebo z krátkou délkou slov realizující dělení.

3. Metoda postupného násobení základem

Pro převod čísla $N_M < 1$ do základu B budeme číslo N_M násobit základem B . Celá část = koeficient a_{-1} . Dalším násobením zbytkové části hodnotou B získáváme další koeficienty a_{-k} . **Rychlý** na procesorech s integrovanou násobičkou.

- **Zadáání čísel do mikropočítače.**
 - Násobení hodnotou 10
 - Není-li násobička $10=(2^3+2^1)$.
 - Knihovní funkce `atoi()`, `atol()`, `atof()`, `atold()`, `_atoi64()` nebo `strtol()` nebo `strtod()`.
- **Převod binární hodnoty pro zobrazení na displeji** (celá část).
 - Metoda postupného odečítání mocnin základu
 - Metoda postupného dělení základem
 - Metoda s využitím dekadické korekce – **je-li k dispozici**.
 - Metoda s využitím dekadické předkorekce.
 - Převod binární hodnoty pro zobrazení na displeji `sprintf()`.
- **Převod binární hodnoty pro zobrazení** (desetinná část).
 - Metoda násobení hodnotou 10
 - Knihovní funkce `sprintf()` – rozsáhlý program závislý na architektuře a možnostech procesoru.

ZOBRAZENÍ DVOJKOVÉ INFORMACE

Převod binárního čísla N_2 do hexadecimálního vyjádření.

$$N_2 = a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \dots + a_5 \cdot 2^5 + a_4 \cdot 2^4 + a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

$$N_2 = 16 \cdot [a_n \cdot 2^{n-4} + a_{n-1} \cdot 2^{n-5} + \dots + a_5 \cdot 2^1 + a_4 \cdot 2^0] + a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$$
$$= 16 \cdot Q_1 + R_1$$

Q_1 představuje celou část podílu, R_1 je zbytek podílu = nejnižší koeficient hexadecimálního čísla vyjádřený váhovanou čtveřicí (8421) nejnižších 4 bitů binárního čísla.

$$R_1 = a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

Opakovaným dělením získáme další symboly hexadecimálního čísla.

$$R_2 = a_7 \cdot 2^3 + a_6 \cdot 2^2 + a_5 \cdot 2^1 + a_4 \cdot 2^0$$

Binární číslo rozdělíme od desetinné tečky na čtveřice bitů, které vyjádříme hexadecimálním symbolem. Například:

$$101110010101,0011_2 = 1011 \mid 1001 \mid 0101, \mid 0011 = B95,3_{16}$$

Při převodu do osmičkové soustavy rozdělujeme číslo po trojicích bitů.

ZOBRAZENÍ DVOJKOVÉ INFORMACE

Hexadecimální vyjádření v jazyce C – vidím stav jednotlivých bitů.

`y=znak&223;` `y=znak&0xDF;` `0xDF= 1101 1111`

Výpis obsahu programové nebo datové paměti

Address:	0x8000000		
0x08000000:	68 04 00 20 6D 07 00 08 75 07 00 08 77 07 00 08 79 07 00 08 7B		
0x08000015:	07 00 08 7D 07 00 08 00 00 00 00 00 00 00 00 00 00 00 00 00		
0x0800002A:	00 00 7F 07 00 08 81 07 00 08 00 00 00 00 83 07 00 08 85 07 00		
0x0800003F:	08 87 07 00 08 87 07 00 08 87 07 00 08 87 07 00 08 87 07 00 08		
0x08000054:	87 07 00 08 87 07 00 08 9F 04 00 08 87 07 00 08 87 07 00 08 87		
0x08000069:	07 00 08 87 07 00 08 87 07 00 08 87 07 00 08 87 07 00 08 87 07		
0x0800007E:	00 08 87 07 00 08 87 07 00 08 87 07 00 08 00 00 00 00 00 00 00		

Call Stack + Locals | Watch 1 | Memory 1

Address:	0x8000000		
0x08000000:	20000468 0800076D 08000775 08000777 08000779 0800077B 0800077D		
0x0800001C:	00000000 00000000 00000000 00000000 0800077F 08000781 00000000		
0x08000038:	08000783 08000785 08000787 08000787 08000787 08000787 08000787		
0x08000054:	08000787 08000787 0800049F 08000787 08000787 08000787 08000787		
0x08000070:	08000787 08000787 08000787 08000787 08000787 08000787 08000787		
0x0800008C:	00000000 00000000 00000000 00000000 08000787 08000787 08000787		
0x080000A8:	08000787 08000787 08000551 08000787 08000787 08000787 08000787		

Call Stack + Locals | Watch 1 | Memory 1

Mám binární číslo 10010111 – O jaké číslo se jedná?

VÁHOVANÉ KÓDY - každý bit má přiřazenu určitou váhu w_j . Součet vah jednotlivých bitů b_j (4bity) se rovná hodnotě čísla.

$$N = b_3 \cdot w_3 + b_2 \cdot w_2 + b_1 \cdot w_1 + b_0 \cdot w_0$$

Váhy w_j mohou být kladná i záporná čísla. Díky tomu mohou být některé kódy nejednoznačné např. (6 4 2 -3) nebo (2 4 2 1).

KOMPLEMENTÁRNÍ KÓDY (self-complementing) usnadňují implementaci aritmetických operací. Součet jejich vah je roven 9. Díky tomu je 9-kový komplement čísla N ($9-N$) reprezentován jednotkovým doplňkem čísla N . Nejznámější např. Aikenův kód (2,4,2,1).

DEKADICKÉ KÓDY – dvojkový formát používaný pro vyjádření symbolů 0,1,...8,9. Často používaný pro binárně-dekadický převod při zobrazování na segmentových a bodových LED zobrazovačích.

DEKADICKÉ KÓDY

Nejčastěji používaným dekadickým kódem je kód **BCD (8,4,2,1)**.

V procesorové technice se často setkáváme s

- **Komprimovaným BCD kódem** - dvě BCD cifry v jednom 8 bitovém slově.
- **ASCII kódem** – používaný při komunikačních přenosech, LCD

Symbol	BCD (8421)	(7421)	(8221)	(84-2-1)	BCD + 3
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 0 0 1	0 0 0 1	0 1 1 1	0 1 0 0
2	0 0 1 0	0 0 1 0	0 0 1 0	0 1 1 0	0 1 0 1
3	0 0 1 1	0 0 1 1	0 0 1 1	0 1 0 1	0 1 1 0
4	0 1 0 0	0 1 0 0	1 0 0 0	0 1 0 0	0 1 1 1
5	0 1 0 1	0 1 0 1	0 1 1 1	1 0 1 1	1 0 0 0
6	0 1 1 0	0 1 1 0	1 1 0 0	1 0 1 0	1 0 0 1
7	0 1 1 1	1 0 0 0	1 1 0 1	1 0 0 1	1 0 1 0
8	1 0 0 0	1 0 0 1	1 1 1 0	1 0 0 0	1 0 1 1
9	1 0 0 1	1 0 1 0	1 1 1 1	1 1 1 1	1 1 0 0

NEVÁHOVANÉ KÓDY – jednotlivé bity nemají přiřazenu konkrétní váhu umožňující stanovení hodnoty čísla. Nejsou předurčeny pro výpočetní operace, ale disponují jinými vlastnostmi potřebnými při přenosu signálů a jeho změn.

Přechod mezi **sousedními stavy binární** informace generuje nežádoucí stavy způsobené rozdílným časem změny $0 \rightarrow 1$ a $1 \rightarrow 0$.

001 $\rightarrow$ 000 $\rightarrow$ 010 nebo 001 $\rightarrow$ 011 $\rightarrow$ 010

1000 $\rightarrow$ 0000 $\rightarrow$ 0111 , 1000 $\rightarrow$ 1111 $\rightarrow$ 0111

Grayův kód (zrcadlový) – sousední stavy se liší v jednom bitu

- Karnaughova mapa – grafická algebraická minimalizace
- Mechanické snímání s maximální chybou jednoho LSB
- Rychlé A/D a D/A převodníky
- Mechanické natočení hřídele (dnes magneticky, CORDIC)

Gray+3 stejná vlastnost pro 10 stavů.

Teploměřový kód se využívá k odstranění nebo potlačení problémů se změnou binární (váhované) informace. Hodnota je vyjádřena počtem log.1 ve sloupci a ukončena řadou log.0. Např.

$$0001111111=7.$$

Telegrafní kódy – vytvořeny pro sériovou komunikaci. Informace doplněna **start bitem** (log.0) a na konci **stop bitem** (log.1). Dnes se používá kód číslo 5 (**ASCII**). Vysílá se 8 nebo 9 bitů se start bitem a 1, 1.5, 2 stop bity. Při multiprocessorové komunikaci slouží nejvyšší bit k identifikaci **data/adresa** nebo zabezpečuje informaci **paritou**.

Kódy s částečným zabezpečením

- Telegrafní kódy 2 z 5 nebo 2 ze 7
- **Johansonův kód** – spojením vlastností Grayova kódu a částečného zabezpečení. Významný pro realizace velmi rychlých čítačů.
- Hammingovy kódy, Konvoluční kódy, Cyklické kódy

I když výkony μP stále vzrůstají, zůstává otázkou zda jednoduché operace s dekadickými čísly realizovat

BCD $\rightarrow$ binární $\rightarrow$ požadované operace $\rightarrow$ binární $\rightarrow$ BCD

Pro realizaci jednoduchých operací s dekadickými čísly (součet, rozdíl, snadný součin) vznikly:

- **Kódy** – excess-3 (BCD+3), Aikenův kód
- **Operace/instrukce** – dekadické korekce

Problém binárního vyjádření dekadických čísel spočívá:

- Procesory pracují s dvojkovou soustavou
- Čísla vyjádřená 4 bity = soustava hexadecimální, **k přenosu dochází** až při dosažení čísla 16.
- Desítková soustava, k přenosu dochází při čísle 10.
- **Rozdíl mezi soustavami je číslo 6**

SČÍTÁNÍ DEKADICKÝCH ČÍSEL V KÓDU EXCESS-3

Dříve používaný kód excess-3 (BCD+3). Při součtu mohou nastat tyto možnosti, aby výsledek byl zase v kódu excess-3.

$$(A+3) + (B+3) = V+6$$

pro $V+6 \in \langle 6; 15 \rangle$, pak od výsledku odečteme číslo 3 (jedna 3 přebývá)

pro $V+6 \in \langle 16; 24 \rangle_{\text{mod } 16}$, pak k výsledku přičteme číslo 3 (obě 3 jsou spotřebované na korekci mezi soustavou 16 a 10)

535	1000	0110	1000	
637	1001	0110	1010	
-----přenos---	1	0	1	-----
1172	0001	0001	1101	0010
	+11	+11	-11	+11

	0100	0100	1010	0101
	1	1	7	2

135	0100	0110	1000	
194	0100	1100	0111	
-----přenos---	0	1	0	-----
329	0000	1001	0010	1111
	+11	-11	+11	-11

	0011	0110	0101	1100
	0	3	2	9

SČÍTÁNÍ DEKADICKÝCH ČÍSEL V KOMPRIMOVANÉM BCD KÓDU

DEKADICKÁ KOREKCE - ihned po součtu (8051) nebo i rozdílu (Zilog, PC) dvou BCD čísel koriguje součet přičtením hodnoty 6 ke čtveřici (spodních/horních) bitů, které představují číslo >9 nebo došlo k přenosu do dalšího řádu (mezi bity 4 a 5 bitem (částečný přenos) nebo 8 a 9 (přenos)).

	0	1	1	0	0	1	0	1	= 65
	0	0	0	1	0	0	1	1	= 13
	0	1	1	1	1	0	0	0	= 78
				7			8		
	0	1	1	0	0	1	0	1	= 65
	0	0	0	1	0	1	1	1	= 17
	0	1	1	1	1	1	0	0	= 7C
Korekce mezi 10 a 16 soustavou	0	0	0	0	0	1	1	0	výsledek není BCD
	1	0	0	0	0	0	1	0	= 82
				8			2		
	0	1	1	0	1	0	0	1	= 69
	0	0	0	1	1	0	0	0	= 18
	1	0	0	0	0	0	0	1	= 81
Korekce mezi 10 a 16 soustavou	0	0	0	0	0	1	1	0	výsledek je BCD ale špatný
	1	0	0	0	0	1	1	1	= 87
				8			7		

PŘEVOD BINÁRNÍ HODNOTY DO BCD KÓDU

U μP s malou pamětí může být použití funkce **sprintf** problematické. Binární číslo s pevnou desetinnou čárkou rozdělíme na **celou a desetinnou část**. Celou část vyjádříme Hornerovým schématem

$$N = ((a_n \cdot 2 + a_{n-1}) \cdot 2 + a_{n-2}) \cdot 2 + \dots + a_0$$

Závorky představují součet dvou stejných čísel ($\cdot 2$) s hodnotou 0 nebo 1 ($a_{n-i} \in \langle 0, 1 \rangle$). Bude-li mezivýsledek v BCD kódu můžeme součet korigovat dekadickou korekcí.

Doba převodu je nezávislá na velikosti čísla. Příklad $0110100_2 \rightarrow N_{10}$.

0 0 0 0 0 0 0 0	Počáteční stav	0 0 0 0 0 1 1 0	(.....).2 + a_3
0 0 0 0 0 0 0 0		0 0 0 0 0 1 1 0	dekadická korekce
0 0 0 0 0 0 0 0	a_6	0 0 0 0 1 1 0 1	(.....).2 + a_2
0 0 0 0 0 0 0 0	dekadická korekce	0 0 0 1 0 0 1 1	dekadická korekce
0 0 0 0 0 0 0 1	$a_6 \cdot 2 + a_5$	0 0 1 0 0 1 1 0	(.....).2 + a_1
0 0 0 0 0 0 0 1	dekadická korekce	0 0 1 0 0 1 1 0	dekadická korekce
0 0 0 0 0 0 1 1	$(a_6 \cdot 2 + a_5) \cdot 2 + a_4$	0 1 0 0 1 1 0 0	(.....).2 + a_0
0 0 0 0 0 0 1 1	dekadická korekce	0 1 0 1 0 0 1 0	dekadická korekce
		5 2	Výsledek

PŘEVOD BINÁRNĚ DEKADICKÝ BEZ DEKADICKÉ KOREKCE

Jsou procesory (AVR, PIC, ARM), které nemají instrukci dekadické korekce. Má-li procesor

- Příznak částečného přenosu můžeme implementovat dekadickou korekcí programově.
- V opačném případě realizujeme **předstih dekadické korekce**. Před dvojnásobkem mezivýsledku a přičtením dalšího bitu testujeme spodní i horní 4 bity. Budou-li 4 bity $\langle 0 \div 4 \rangle$ korekce není třeba. K číslu $\langle 5 \div 9 \rangle$ **realizujeme předstih korekce přičtením hodnoty 3**.
Příklad $1011111_2 = 95_{10}$.

0 0 0 0 0 0 0 0	Počáteční stav
0 0 0 0 0 0 0 0	předkorekce
0 0 0 0 0 0 0 1	a_6
0 0 0 0 0 0 0 1	předkorekce
0 0 0 0 0 0 1 0	$a_6 \cdot 2 + a_5$
0 0 0 0 0 0 1 0	předkorekce
0 0 0 0 0 1 0 1	$(a_6 \cdot 2 + a_5) \cdot 2 + a_4$

0 0 0 0 1 0 0 0	předkorekce
0 0 0 1 0 0 0 1	$(\dots\dots\dots) \cdot 2 + a_3$
0 0 0 1 0 0 0 1	předkorekce
0 0 1 0 0 0 1 1	$(\dots\dots\dots) \cdot 2 + a_2$
0 0 1 0 0 0 1 1	předkorekce
0 1 0 0 0 1 1 1	$(\dots\dots\dots) \cdot 2 + a_1$
0 1 0 0 1 0 1 0	předkorekce
1 0 0 1 0 1 0 1	$(\dots\dots\dots) \cdot 2 + a_0$
9 5	Výsledek

VYJÁDŘENÍ ZÁPORNÝCH ČÍSEL V PEVNÉ DESETINNÉ ČÁRCE

Čísla se **znaménkem** mají počet bitů rezervovaný na rozsah čísla zvětšený o jeden bit, který ponese informaci o znaménku.

➤ **± absolutní hodnota** – Nejvyšší bit = znaménko (0 = +, 1 = -). Zbývající bity určují absolutní hodnotu.

Rozsah kladných i záporných čísel pro n bitů je $\langle 0, 2^{n-1}-1 \rangle$.

Ve vyjádření existují **dvě nuly**, použití záporné nuly 10000...00 se většinou zakazuje.

Vyjádření je pro obvodovou implementaci nevhodné, při procesorovém zpracování nevadí.

➤ **Vyjádření záporného čísla jednotkovým doplňkem** – získáme inverzí všech jeho bitů.

$${}^1A = 2^n - 1 - A = 2^n - 1 - \sum_{i=0}^{n-2} a_i 2^i$$

Rozsah čísla pro n bitů se shoduje s vyjádřením ± absolutní hodnota. Existují **dvě nuly** kladná (000..0), záporná (111..1). Použití v rychlých sčítačkách a násobičkách.

- **Vyjádření záporného čísla dvojkovým doplňkem** – vypočteme z následujícího vztahu

$$^2A = 2^n - A = 1 + ^1A = 2^n - \sum_{i=0}^{n-2} a_i 2^i$$

Dvojkový doplněk = jednotkový doplněk plus 1 **k nejnižšímu řádu** (pro celá i desetinná). Rozsah kladných čísel $<0, 2^{n-1}-1>$, záporných $<-1, -2^{n-1}>$. Nejvyšší bit = znaménko (0 = +, 1 = -). Existuje již jen jedna nula (000 .. 0).

- **+1/2 intervalu (s posunem)** – Číslo v tomto formátu vypočteme ze vztahu

$$^{1/2}A = 2^{n-1} + A = 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

kde n je počet bitů čísla. Kladná čísla leží v intervalu $<0, 2^{n-1}-1>$, záporná čísla $<-1, -2^{n-1}>$. Bit s nejvyšší váhou opět značí znaménko s tím, že 0 = -, 1 = +. Stejně jako v případě dvojkového doplňku má vyjádření jen jednu nulu (100 .. 0).

VYJÁDŘENÍ ZÁPORNÝCH ČÍSEL

Číslo	bin.hodnota	$\pm A $	1A	2A	$^{1/2}A$
+10	+1010	01010	01010	01010	11010
+0	+0000	00000	00000	00000	10000
-0	-0000	10000	11111	-----	-----
-2	-0010	10010	11101	11110	01110
-14	-1110	11110	10001	10010	00010
-3,5	-0011,10	10011,10	11100,01	11100,10	01100,10
-0,5	-0000,10	10000,10	11111,01	11111,10	01111,10
-16	nelze vyjádřit	nelze vyjádřit	nelze vyjádřit	10000	00000

Vytváření doplňků

- **Jednotkový** – inverze všech bitů
- **Dvojkový** – inverze všech bitů plus 1 k nejnižšímu řádu.
 - **Sériový převod** spočívá v kopírování bitů od nejnižšího bitu po bit nejvyšší včetně první jedničky. Po první jedničce jsou všechny bity **invertovány**.

ODEČÍTÁNÍ S DVOJKOVÝM DOPLŇKEM

Rozdíl čísel $A-B$ za pomoci dvojkového doplňku čísla B realizujeme pomocí součtu $A+{}^2B$. Tím získáváme požadovaný rozdíl $A-B$ zvětšený o hodnotu 2^n . Pro jednotlivé případy můžeme psát

pro $A > B$

$$A + {}^2B > 2^n$$

$$A - B = A + {}^2B - 2^n$$

pro $A < B$

$$A + {}^2B \leq 2^n$$

$$A - B = 2^n + A - B = 2^n - R = {}^2R$$

K dosažení správné hodnoty rozdílu $A-B$ potřebujeme odečíst (oříznout) hodnotu 2^n .

$$A_{10} = 21$$

$$B_{10} = 13$$

$$A = 0\ 1\ 0\ 1\ 0\ 1$$

$${}^2B = 1\ 1\ 0\ 0\ 1\ 1$$

$$\begin{array}{r} A \\ {}^2B \\ \hline 1\ 0\ 1\ 0\ 0\ 0 \end{array}$$

↖ Výsledek kladný

Je-li rozdíl záporný, pak získáme přímo jeho dvojkový doplněk.

$$A_{10} = 21$$

$$B_{10} = 13$$

$${}^2A = 1\ 0\ 1\ 0\ 1\ 1$$

$$B = 0\ 0\ 1\ 1\ 0\ 1$$

$$\begin{array}{r} {}^2A \\ B \\ \hline 1\ 1\ 1\ 0\ 0\ 0 \end{array}$$

↖ Výsledek záporný

Při sčítání a odečítání nastávají tyto limitní případy:

- Součtem dvou kladných čísel bez znaménka dochází k přenosu za nejvyšší bit = indikováno **příznakem přenosu C**
- Součtem dvou kladných čísel se znaménkem získáme číslo záporné. Indikováno **příznakem přetečení OV**
- Součtem dvou záporných čísel získáme číslo kladné. Indikováno **příznakem přetečení OV**

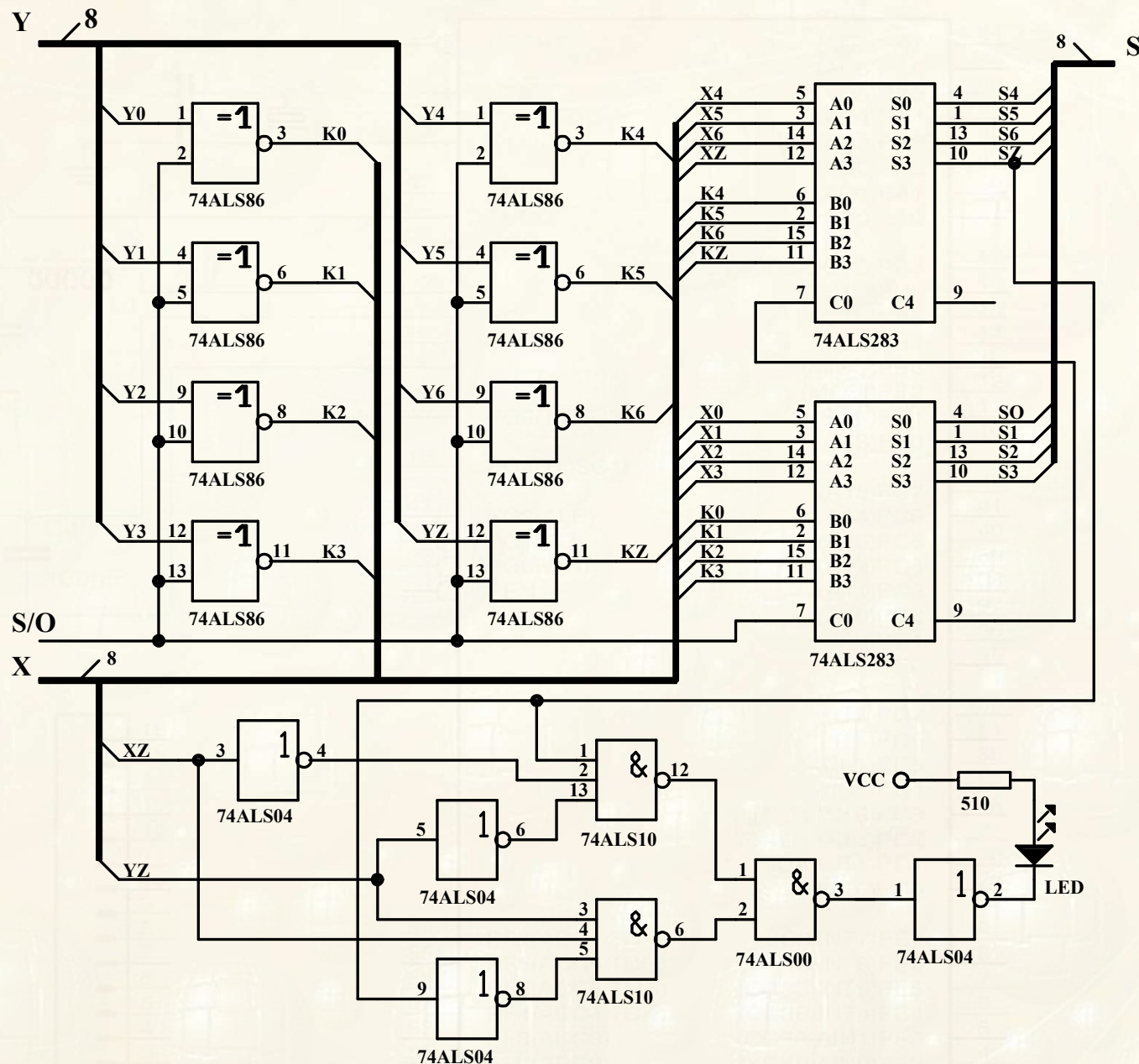
Příznak přetečení identifikuje obvod, který kontroluje znaménka čísel a výsledku.

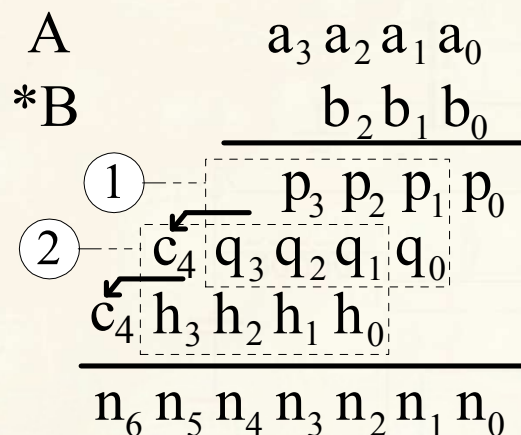
$$OV = \bar{z}_x \cdot \bar{z}_y \cdot z_s + z_x \cdot z_y \cdot \bar{z}_s$$

kde z_x , z_y a z_s představují znaménka X, Y a součtu (nebo rozdílu).

Při odečítání vytvoříme jednotkový doplněk obvodu EX-OR (neekvivalence) nebo EX-NOR (ekvivalence) podle polaroty signálu S/O. **Dvojkový doplněk** vytvoříme při sčítání ve **sčítačce** zavedením **nenulového počátečního přenosu** pro operaci odčítání viz. následující blána.

REALIZACE 8-BITOVÉ SČÍTAČKY A ODEČÍTAČKY S INDIKACÍ PŘETEČENÍ





Aritmetický součin bitů shodný s operací AND.

Součet dílčích součinů

- Ve sčítačkách s kaskádním přenosem
- Jedno nebo více stupňovým zrychleným kanálem přenosu

Doba k vytvoření součinu

- Zřetězený $t_{\text{součinu}} = (n-1) \cdot t_{\Sigma}$ n-počet bitů
- Stromový $t_{\text{součinu}} = (\log(n)/\log(2)) \cdot t_{\Sigma}$
- Walesova stromová sčítačka (kompresory)

Další zkrácení doby výpočtu $\Rightarrow$ redukce počtu dílčích součinů

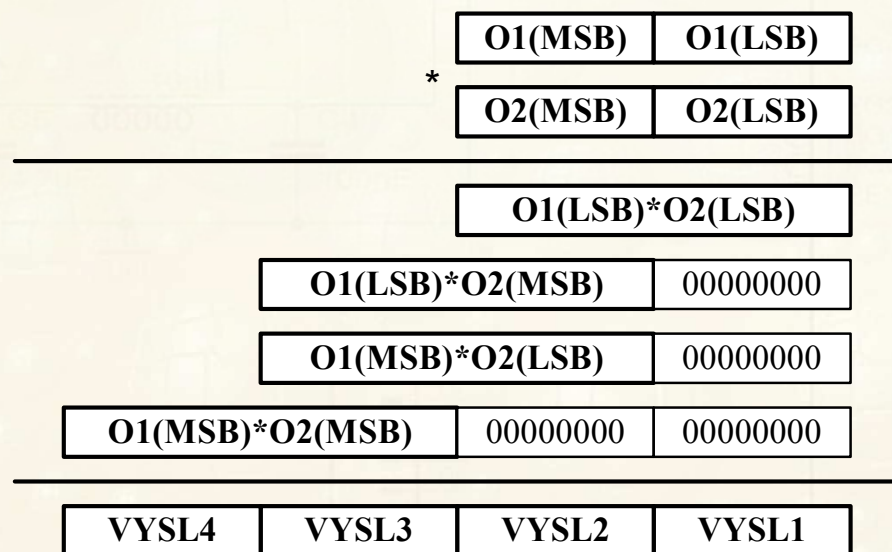
- Booth algoritmus
- Modifikovaný Booth algoritmus (radix4, 8, 32, 256)

V Booth nebo modifikovaném algoritmu násobíme **dvěma** nebo **více bity** najednou. Dochází tak ke snížení sčítaných dílčích součinů na polovinu, třetinu atd.

PROCESOROVÁ REALIZACE OPERACE NÁSOBENÍ

Programové řešení může být

- Sériové na μP bez násobičky.
- Procesor má násobičku s rozsahem bitů potřebným pro operandy.
- Program využívá násobičku μP , která má nedostatečný počet bitů (aplikace dávného řešení s pamětmi ROM). Příklad součinu 16x16bitů na μP s násobičkou 8x8bitů.



Má-li v jazyce C výsledek násobení stejný počet bitů jako operandy, přináší to výhody i úskalí.

`unsigned int x , y;    (int=16bitů).`

`x=1000; y=100; z=x*y;`

Kolik bude z=?

Poslední součin v knihovně jazyka C není $\Rightarrow$ při práci s aritmetikou s pevnou desetinnou tečkou je žádoucí **testovat velikost operandů**.

NÁSOBENÍ ČÍSEL SE ZNAMÉNEM VE DVOJKOVÉM DOPLŇKU

$$\tilde{X} = (-1).\tilde{x}_z + \sum_{i=1}^b \tilde{x}_{-i}.2^{-i} \quad \tilde{Y} = (-1).\tilde{y}_z + \sum_{j=1}^b \tilde{y}_{-j}.2^{-j}$$

$$\tilde{X}.\tilde{Y} = \left[(-1).\tilde{x}_z + \sum_{i=1}^b \tilde{x}_{-i}.2^{-i} \right] \cdot \left[(-1).\tilde{y}_z + \sum_{j=1}^b \tilde{y}_{-j}.2^{-j} \right] =$$

$$= (-1).\tilde{x}_z \cdot \left[(-1).\tilde{y}_z + \sum_{j=1}^b \tilde{y}_{-j}.2^{-j} \right] + \sum_{i=1}^b \tilde{x}_{-i} \cdot \left[(-1).\tilde{y}_z + \sum_{j=1}^b \tilde{y}_{-j}.2^{-j} \right] .2^{-i}$$

A	<u>1</u> 0011	= -13
B	<u>0</u> 1001	= 9
	1 11110011	
	0 0000000	
	0 000000	
	1 10011	
	1 <u>1</u> 10001011	= -117

A	<u>0</u> 1101	= 13
B	<u>1</u> 0111	= -9
	0 00001101	
	0 0001101	
	0 001101	
	0 00000	
	00 <u>0</u> 01011011	
	- <u>0</u> 1101	
	11 <u>1</u> 10001011	= -117

A	<u>1</u> 0011	= -13
B	<u>1</u> 0111	= -9
	1 11110011	
	1 1110011	
	1 110011	
	0 00000	
	10 <u>1</u> 10100101	
	- <u>1</u> 0011	
	10 <u>0</u> 01110101	= 117

Algoritmus je vhodný pro sériovou realizaci násobení.

MODIFIKOVANÝ BOOTHŮV ALGORITMUS

Booth při hledání metody zmenšující počet sčítaných produktů přišel s nápadem (příklad $B \cdot 00111100$)

Klasický součin B -bit realizuje součet čtyřech produktů

$$B \cdot 00111100 = B \cdot 2^5 + B \cdot 2^4 + B \cdot 2^3 + B \cdot 2^2$$

Násobitele můžeme nestandardně vyjádřit **0 1 0 0 0 0 0 0 -1 0 0**

$$B \cdot (0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 - 1 \ 0 \ 0) = B \cdot 2^6 - B \cdot 2^2$$

Při násobení konstantou nám to v některých případech pomůže, **obecně ne**. Analýza ukázala cestu ke snížení počtu součinů.

Vyjádření	Bitové vyjádření 0 0 1 1 0 1 1 1 0 , 0	Nenulový počet bitů	
		Průměrný	Max.
Binární	$2^6 \ 2^5 \quad 2^3 \ 2^2 \ 2^1$	$n/2$	n
Kanonické znaménkové	$2^7 \quad -2^4 \quad -2^1$	$n/3$	$n/2$
Modif.Boothův algoritmus R.4	0 +2 -1 0 -2 $2 \cdot 2^6$ $-1 \cdot 2^4$ $-2 \cdot 2^0$	---	$n/2$

NÁSOBÍCÍ ALGORITMY VYŠŠÍHO ŘÁDU - BOOTHŮV ALGORITMUS

a_{2j+1}	a_{2j}	$p_{2(j-2)+4}$	p_{2j+2}	B	A
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	1	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	0	0

Násobíme-li dvěma bity

$$a_{2j+1}, a_{2j} = 0, 1, 2, 3.$$

Hodnotu 3 realizujeme jako $4-1 \Rightarrow$ realizovat dekodér, který k bitům a_{2j+3}, a_{2j+2} přičte hodnotu 1.

Výstupy dekodéru B, A adresují multiplexer, který přivede dílčí součin 0, B, 2B nebo $-B$.

Bit p_{2j+2} je přenos do dalšího řádu.

NEVÝHODA: **Nelze paralelizovat v důsledku kaskádního přenosu.**

$$\begin{array}{r}
 Y \quad 01101 \\
 X \quad 00111 \\
 \hline
 000001101 \\
 00001101 \\
 0001101 \\
 000000 \\
 \hline
 0001011011
 \end{array}
 \begin{array}{l}
 = 13 \\
 = 7 \\
 \text{součin s } x_0 \\
 \text{součin s } x_1 \\
 \text{součin s } x_2 \\
 \text{součin s } x_3 \\
 = 91
 \end{array}$$

$$\begin{array}{r}
 Y \quad 01101 \\
 X \quad 00111 \\
 \hline
 111110011 \\
 0011010 \\
 \hline
 1001011011
 \end{array}
 \begin{array}{l}
 = 13 \\
 = 7 \\
 \text{součin s } x_1, x_0 \quad -B \\
 \text{součin s } x_3, x_2 \quad (01+1).4B \\
 = 91
 \end{array}$$

↑
přenos

$$\begin{aligned}
 \tilde{A} \cdot \tilde{B} &= \left[-\tilde{a}_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} \tilde{a}_i \cdot 2^i \right] \cdot \tilde{B} + \tilde{a}_{-1} \cdot \tilde{B} \cdot 2^0 = \\
 &= -\tilde{a}_{n-1} \cdot \tilde{B} \cdot 2^{n-1} + \tilde{a}_{n-2} \cdot \tilde{B} \cdot 2^{n-2} + \dots + \tilde{a}_1 \cdot \tilde{B} \cdot 2^1 + \tilde{a}_0 \cdot \tilde{B} \cdot 2^0 + \tilde{a}_{-1} \cdot \tilde{B} \cdot 2^0 = \\
 &= (\tilde{a}_{n-2} - \tilde{a}_{n-1}) \cdot \tilde{B} \cdot 2^{n-1} + (\tilde{a}_{n-3} - \tilde{a}_{n-2}) \cdot \tilde{B} \cdot 2^{n-2} + \dots \\
 &\dots + (\tilde{a}_2 - \tilde{a}_3) \cdot \tilde{B} \cdot 2^3 + (\tilde{a}_1 - \tilde{a}_2) \cdot \tilde{B} \cdot 2^2 + (\tilde{a}_0 - \tilde{a}_1) \cdot \tilde{B} \cdot 2^1 + (\tilde{a}_{-1} - \tilde{a}_0) \cdot \tilde{B} \cdot 2^0 = \\
 &= (\tilde{a}_{n-2} \cdot 2 - \tilde{a}_{n-1} \cdot 2 + \tilde{a}_{n-3} - \tilde{a}_{n-2}) \cdot \tilde{B} \cdot 2^{n-2} + \dots \\
 &\dots + (-\tilde{a}_2 \cdot 2 - \tilde{a}_3 \cdot 2 + \tilde{a}_1 - \tilde{a}_2) \cdot \tilde{B} \cdot 2^2 + (\tilde{a}_0 \cdot 2 - \tilde{a}_1 \cdot 2 + \tilde{a}_{-1} - \tilde{a}_0) \cdot \tilde{B} \cdot 2^0 = \\
 &= (-\tilde{a}_{n-1} \cdot 2 + \tilde{a}_{n-2} + \tilde{a}_{n-3}) \cdot \tilde{B} \cdot 2^{n-2} + \dots \\
 &\dots + (-\tilde{a}_3 \cdot 2 + \tilde{a}_2 + \tilde{a}_1) \cdot \tilde{B} \cdot 2^2 + (-\tilde{a}_1 \cdot 2 + \tilde{a}_0 + \tilde{a}_{-1}) \cdot \tilde{B} \cdot 2^0
 \end{aligned}$$

Bude-li koeficient $\tilde{a}_{-1}=0$, pak výpočet součinu z trojice bitů násobitele bude roven součinu čísel A a B se znaménkem. Kombinace tří bitů pak představuje hodnotu násobence 0, B, 2B, -2B, -B. Jak udělat posun trojce bitů o dva bity?

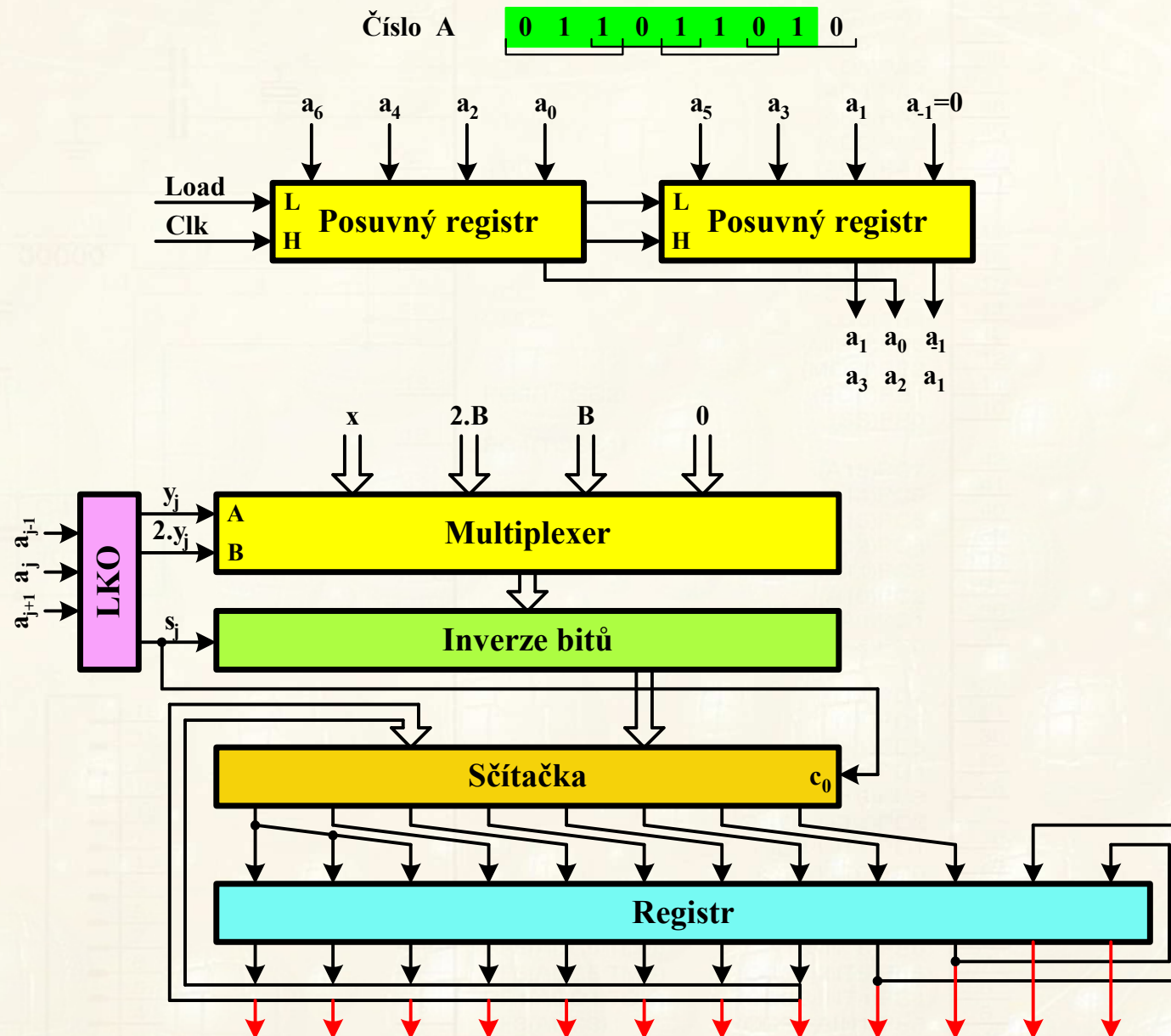
Proces se dá plně paralelizovat.

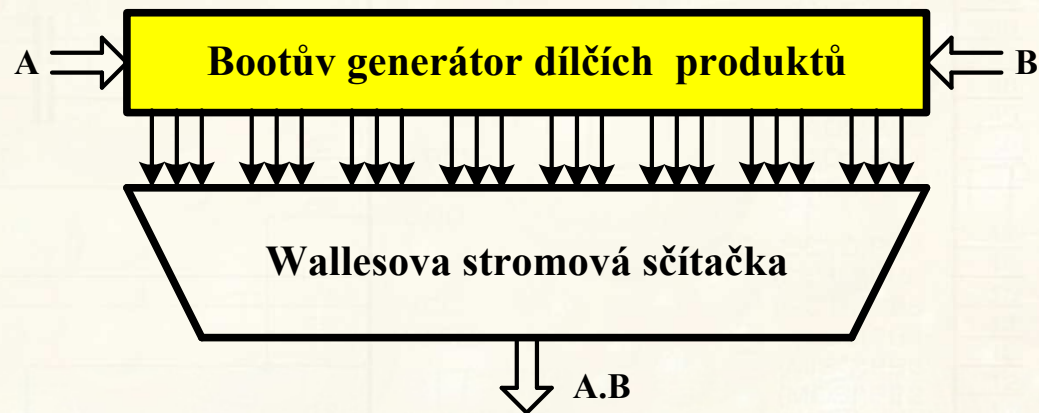
MODIFIKOVANÝ BOOTHŮV ALGORITMUS - RADIX4

Příklad na sériově realizovaný součin čtyřbitových čísel **1110** (-2) a **1101** (-3) (Výsledek = **0000 0110** (+6)) modifikovaným Boothovým algoritmem-Radix4. Součin vzniká v **ALU** a pomocném posuvném **registru** i s pomocným **bitem** pro počáteční hodnotu a_{-1} .

Iterace	B	Modifikovaný Boothův algoritmus – Radix4	
		Krok	Součin
0	1110	Inicializace	0000 1101 0
1	1110	010 $\Rightarrow$ výsl=výsl+B	1110 1101 0
	1110	2x posun doprava	1111 1011 0
2	1110	110 $\Rightarrow$ výsl=výsl-B	0001 1011 0
	1110	2x posun doprava	0000 0110 1

MODIFIKOVANÝ BOOTHŮV ALGORITMUS - RADIX4 - SÉRIOVÝ





Stejně dospějeme k modifikovanému Boothovu algoritmu – RADIX8 (práce se 4 bity) vede na dílčí součiny (0, B, 2B, 3B, 4B, -4B, -3B, -2B a -B).

Součin 3B (-3B) je třeba realizovat a otázkou zůstává kolik ušetříme na výpočtu 1/3 součinů a spotřebujeme na realizaci 3B.

U algoritmů RADIX32 a RADIX256 je situace s dílčími součiny složitá a využívají se při ní číselné soustavy mod7 a mod11.

VYJÁDŘENÍ ČÍSEL V POHYBLIVÉ DESETINNÉ ČÁRCE

Číslo v pohyblivé desetinné čárce je rozšířeno o hodnotu exponentu včetně jeho znaménka. Standardem je **IEEE 754-1985** (inovovaný **IEEE 754-2008**)

$$A = 0, \quad A = \infty, \quad A = (-1)^z \cdot (1 + m) \cdot 2^{e+127}$$

kde **z** - znaménko čísla, **e** - exponent a **m** - mantisu (desetinné číslo) danou výrazem

$$m = a_{-1}2^{-1} + a_{-2}2^{-2} + \dots + a_{-n+1}2^{-n+1} + a_{-n}2^{-n}$$

Závorka $(1+m)$ leží v intervalu $<1,0;2,0)$ (**normovaná mantisa**). Mantisa je ve formátu $\pm$ absolutní hodnota. Exponent se znaménkem je ve formátu s posunutou nulou ($+1/2$ intervalu zmenšeném o hodnotu 1)

$$exponent = e + 127 = a_{m-1}a_{m-2}a_{m-3}\dots a_0$$

se pohybuje v intervalu $<-126;127>$ pro dvojnásobnou přesnost $<-1022;1023>$).

VYJÁDŘENÍ ČÍSEL V POHYBLIVÉ DESETINNÉ ČÁRCE

Krajní hodnoty exponentu $e+127=0$ a $e+127=255$ se využívají k vyjádření 0 a $\pm\infty$. Číslo je vyjádřeno 4 byty

$$\begin{aligned} \text{MSB} &= z \ e_7 e_6 e_5 e_4 e_3 e_2 e_1 \\ &\quad e_0 \ m_{-1} m_{-2} m_{-3} m_{-4} m_{-5} m_{-6} m_{-7} \\ &\quad m_{-8} m_{-9} m_{-10} m_{-11} m_{-12} m_{-13} m_{-14} m_{-15} \\ \text{LSB} &= m_{-16} m_{-17} m_{-18} m_{-19} m_{-20} m_{-21} m_{-22} m_{-23} \end{aligned}$$

Jednička z výrazu $(1+m)$ je tzv. **skrytá jednička** (ve vyjádření se neobjevuje).

Číslo		Vyjádření		
		z	exponent	mantisa
$1,000 \cdot 10^0$	$1,000000 \cdot 2^0$	0	01111111	0000000000
$8,000 \cdot 10^0$	$1,000000 \cdot 2^3$	0	10000010	0000000000
$0,000 \cdot 10^0$	$0,000000 \cdot 2^0$	0	00000000	0000000000
$-1,750 \cdot 10^0$	$1,750000 \cdot 2^0$	1	01111111	1100000000
$1,250 \cdot 10^2$	$1,953125 \cdot 2^6$	0	10000101	1111010000
$-1,250 \cdot 10^2$	$-1,953125 \cdot 2^6$	1	10000101	1111010000
$-7,812 \cdot 10^{-2}$	$-1,249920 \cdot 2^{-4}$	1	01111011	0011111111

VYJÁDŘENÍ ČÍSEL V POHYBLIVÉ DESETINNÉ ČÁRCE

Pro atypické vyjádření čísel s pohyblivou čárkou určíme nezbytný počet bitů mantisy **n** ze vztahů :

$$2^{-n} \leq 10^{-\text{pocet desetinných míst}} \quad n \geq \frac{\text{pocet desetinných míst}}{\log 2}$$

Počet bitů exponentu **m** bude dán vztahy

$$2^{2^{m-1}} \geq 10^{\text{rozsah exponentu}} \quad m \geq 1 + \frac{\log \left(\frac{\text{rozsah exponentu}}{\log 2} \right)}{\log 2}$$

Násobení čísel A a B v pohyblivé čárce můžeme vyjádřit takto

$$A.B = \left[(-1)^{z_A} \cdot M_A \cdot 2^{e_A} \right] \cdot \left[(-1)^{z_B} \cdot M_B \cdot 2^{e_B} \right]$$

M_A M_B - normované mantisy ($1+m$), $M_A.M_B$ - součin čísel v pevné desetinné čárce. Výsledek součinu bude mít dvojnásobný počet bitů a bude ležet v intervalu $<1;4$). Pro $M_A.M_B \geq 2$, nutný posun o pozici doprava (normovaná mantisa) a exponent + 1. Po **normování** dojde k omezení/zaokrouhlení mantisy $\Rightarrow$ operace není **lineární a výsledek je zatížen chybou**

$$\begin{aligned} A.B &= (-1)^{z_A \oplus z_B} \cdot M_A \cdot M_B \cdot 2^{e_A + e_B} = (-1)^{z_V} \cdot m_V \cdot 2^{e_A + e_B} = \\ &= \text{normování} = (-1)^{z_V} \cdot M_V \cdot 2^{e_V} \end{aligned}$$

Při **dělení** je situace stejná, násobení nahrazeno dělením, součet exponentů rozdílem. Pro mantisu <1 , posun o jednu pozici doleva a exponent zmenšen o jedničku.

Při sčítání a odečítání je situace následující

$$\begin{aligned}
 A \pm B &= (-1)^{z_A} \cdot M_A \cdot 2^{e_A} \pm (-1)^{z_B} \cdot M_B \cdot 2^{e_B} = \\
 &= (-1)^{z_A} \cdot M_A \cdot 2^{e_A} \pm (-1)^{z_B} \cdot m_B \cdot 2^{e_A} = \\
 &= \left[(-1)^{z_A} \cdot M_A \pm (-1)^{z_B} \cdot m_B \right] 2^{e_A} = \\
 &= (-1)^{z_V} \cdot m_V \cdot 2^{e_A} = \textit{normování} = (-1)^{z_V} \cdot M_V \cdot 2^{e_V}
 \end{aligned}$$

Mantisa menšího čísla posunována doprava až dojde ke srovnání exponentů. Následuje součet mantis v pevné desetinné čárce. Podle výsledku je mantisa upravena tak, aby ležela v intervalu $<1;2$). Rozšířenou mantisu opět omezíme nebo zaokrouhlíme na standardní počet bitů $\Rightarrow$



- **Operace není lineární, výsledek je zatížen chybou.**
- **Součet nebo rozdíl řádově velkého a malého čísla se nemusí projevit ve výsledku.**

PROBLÉMY PŘI POUŽITÍ POHYBLIVÉ ČÁRKY

Formát v pohyblivé čárce nepokrývá všechna racionální čísla.

- Pohyblivou čárku je nevhodné používat **pro finanční operace**. Např. hodnotu 0,1 nelze přesně vyjádřit.

Problémy s Excel v 90 letech 20. století

- Následující zápis vede na nekonečnou smyčku

```
double y=0.0; while(y!=1.0) { y+=0.1; ..... atd. }
```

- Zápis programu vede na špatný počet cyklů

```
double y=0.0; int počet=0;
while(y<1.0) { y+=0.1; počet++; ..... }
```

Pro (y<1.0) cyklů 11 místo 10. Pro (y<2.0) správně 20 cyklů.

Řada případů špatné implementace pohyblivé čárky vedla až k tragickým událostem.

Např. systém Patriot v Irácké válce.

PROBLÉMY PŘI POUŽITÍ POHYBLIVÉ ČÁRKY

Nejčastější problémy souvisí s rozsahem operandů a mantisy.

➤ Rozsah mantisy

`float x = 4194304.0; x+=0.25; ⇒ výsledek x= 4194304.0;`

`float u=0.001; y=1000000;`

`for (i=0; i<1000; i++) {y=y+u} ⇒ výsledek y= 1000000.0;`