REALIZACE ČÍSLICOVÝCH OBVODŮ

Realizace logických a číslicových obvodů závisí:

- ♣ **Složitosti**
- ♣ **Spotřebě**
- ♣ **Rychlosti odezvy**
- ♣ **Ceně**

- **Klasické obvody z řady 74xxx**
- **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*** – µP řadiče, Dnes řadiče PC.
  - ***Assembler*** (Strojový kód) – realizace instrukcemi µP. Dnes – ovladače, knihovny jazyků, kontrola zpoždění
  - ***Jazykem C*** – popis složitějších aplikací operacemi, které programovací jazyk poskytuje.

Číselné základy

OBVYKLÉ ČÍSELNÉ ZÁKLADY

V číslicových systémech standardně používáme tři číselné základy:

- **Základ 2** – se symboly 0 a 1 (někdy 0 a I)
  - Realizace aritmetických a logických operací v počítačích, mikroprocesorech a programovatelných logických polích.
- **Základ 10** – se symboly 0,1,2,3,4,5,6,7,8,9
  - Používaná soustava – slouží k 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 k zadávání 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 asemblerovských programech.
- Základ 7 – Modifikovaný Bootův algoritmus násobení radix 32
- Základ 11 – Modifikovaný Bootův algoritmus násobení radix 256

PRINCIPY ZMĚNY ZÁKLADU

1. Metoda postupného odečítání mocnin základu

Koeficienty hledaného 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, který bude představovat koeficient 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. Získáme celou část (koeficient a_{-1}) a zbytek. Dalším násobení zbytkové části hodnotou B získáváme další koeficienty a_{-k}. Koeficienty získáváme v pořadí tak, jak jdou za sebou za desetinou čárkou. **Rychlý** na procesorech s integrovanou násobičkou.

BINÁRNĚ-DEKADICKÝ PŘEVOD ČÍSLA

- Zadávání čísel do mikropočítače.
  - Násobení hodnotou 10
  - Není-li násobička $10=(2^3+2^1)$, součet posunutých hodnot doleva
  - 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).
  - Metody 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 - dělení základem 16.

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

$$N_2 = 16.\left[a_n.2^{n-4} + a_{n-1}.2^{n-5} + \cdots + a_5.2^1 + a_4.2^0\right] + a_3.2^3 + a_2.2^2 + a_1.2^1 + a_0.2^0$$

$$= 16.Q_1 + R_1$$

R_1 představuje nejnižší koeficient hexadecimálního čísla vyjádřený váhovanou čtveřicí (8421) nejnižších 4 bitů binárního čísla.

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

Opakovaným dělením získáme následující symboly hexadecimálního čísla.

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

Při převodu dvojkové číslo rozdělíme od desetinné tečky na čtveřice bitů, které vyjádříme hexadecimálním symbolem. Například takto:

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

Stejné odvození můžeme provést mezi dvojkovou a osmičkovou soustavou s tím, že dvojkové číslo budeme rozdělovat po trojicích bitů.

Hexadecimální vyjádření v jazyce C – vidím stav jednotlivých **bitů.**

VYJÁDŘENÍ ČÍSEL - KÓDY

Mám binární číslo 10010111 – O jaké číslo se jedná?

DEKADICKÉ KÓDY – dvojkový formát pro vyjádření symbolů 0,1,…8,9. Pokud symboly 0,1,…,8,9 nevyjadřujeme **ASCII** kódem (přenos po komunikačních sběrnicích), využíváme k dvojkovému vyjádření symbolů BCD nebo BCD+3 (komplementární) formát.

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

VÁHOVANÉ BINÁRNÍ KÓDY

VÁHOVANÉ KÓDY - každý bit má přiřazenu určitou váhu w_j. Součet vah bitů $b_j=1$ se rovná dekadickému číslu (4bity).

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

Některé kódy mohou být nejednoznačné např.

6 4 2 -3 číslo 7 (1101 nebo 0111) nebo

2 4 2 1 číslo 7 (1101 nebo 0111).

Kódy, jejichž součet vah je roven 9, mají tu vlastnost, že 9-kový komplement čísla N (9-N) je reprezentován kódem s invertovanými bity čísla N (tzv. jednotkový doplněk).

Kódy mající takovou vlastnost jsou známy jako **komplementární kódy** (self-complementing) a usnadňují implementaci aritmetických operací. Např. Aikenův kód (2,4,2,1).

NEVÁHOVANÉ KÓDY – jednotlivé bity nemají přiřazenu konkrétní váhu.

OPERACE S DEKADICKÝMI ČÍSLY

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

BCD → binární → požadované operace → binární → BCD

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

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

Problém operace s dekadickými čísly spočívá:

- Procesory pracují s dvojkovou soustavou
- Čísla vyjádřená 4 bity = soustava hexadecimální přenos při dosažení čísla 16 – dochází k přenosu do dalšího řádu.
- Desítková soustava, přenos při dosažení čísla 10.
- Rozdíl mezi soustavami je číslo 6

SČÍTÁNÍ DEKADICKÝCH ČÍSEL V KÓDU EXCEESS-3

Dříve používaný kód exceess-3 (BCD+3). Při součtu mohou nastat tyto možnosti, aby výsledek byl zase v kódu exceess-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 $V+6 \in \langle 16;24 \rangle_{mod16}$, 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
```

DEKADICKÁ KOREKCE - se uplatňuje bezprostředně po součtu nebo rozdílu dvou BCD čísel. Korekce spočívá v přičtení hodnoty 6 ke čtveřici (spodních/horních) bitů které představují číslo >9 nebo došlo k přenosu mezi 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	výsledek není BCD
Korekce mezi 10 a 16 soustavou	0	0	0	0	0	1	1	0		
	1	0	0	0	0	0	1	0	= 82	
		8				2				

AC=1

	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	výsledek je BCD ale špatný
Korekce mezi 10 a 16 soustavou	0	0	0	0	0	1	1	0		
	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 problématické. Binární číslo s pevnou desetinnou čárkou rozdělíme na celou a desetinnou část. Celou část vyjádříme Hornerovým schématem

$$N = ((a_n.2 + a_{n-1}).2 + a_{n-2}).2 + ... + a_0$$

Závorky představují součet dvou stejných čísel (.2) s hodnotou 0 nebo 1 ($a_{n-i} \in <0,1>$). 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	0	0	0		a_6
0	0	0	0	0	0	0	0		**dekadická korekce**
0	0	0	0	0	0	0	1		$a_6.2 + a_5$
0	0	0	0	0	0	0	1		**dekadická korekce**
0	0	0	0	0	0	1	1		$(a_6.2 + a_5).2 + a_4$
0	0	0	0	0	0	1	1		**dekadická korekce**
0	0	0	0	0	1	1	0		$(.....).2 + a_3$
0	0	0	0	0	1	1	0		**dekadická korekce**
0	0	0	0	1	1	0	1		$(.....).2 + a_2$
0	0	0	1	0	0	1	1		**dekadická korekce**
0	0	1	0	0	1	1	0		$(.....).2 + a_1$
0	0	1	0	0	1	1	0		**dekadická korekce**
0	1	0	0	1	1	0	0		$(.....).2 + a_0$
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 korekci.
- V opačném případě realizujeme předstih dekadické korekce. Před dvojnásobkem mezivýsledku a přičtením dalšího bitu testujeme spodní i horní 4 bity. Budou-li 4 bity <0÷4> korekce není třeba. K číslu <5÷9> 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	$a_6.2 + a_5$
0 0 0 0 0 0 1 0	**předkorekce**
0 0 0 0 0 1 0 1	$(a_6.2 + a_5).2 + a_4$
0 0 0 0 1 0 0 0	**předkorekce**
0 0 0 1 0 0 0 1	$(.....).2 + a_3$
0 0 0 1 0 0 0 1	**předkorekce**
0 0 1 0 0 0 1 1	$(.....).2 + a_2$
0 0 1 0 0 0 1 1	**předkorekce**
0 1 0 0 0 1 1 1	$(.....).2 + a_1$
0 1 0 0 1 0 1 0	**předkorekce**
1 0 0 1 0 1 0 1	$(.....).2 + a_0$
9 5	**Výsledek**

VYJÁDŘENÍ ČÍSEL - KÓDY

Symbol	Gray	Gray + 3
0	0 0 0 0	0 0 1 0
1	0 0 0 1	0 1 1 0
2	0 0 1 1	0 1 1 1
3	0 0 1 0	0 1 0 1
4	0 1 1 0	0 1 0 0
5	0 1 1 1	1 1 0 0
6	0 1 0 1	1 1 0 1
7	0 1 0 0	1 1 1 1
8	1 1 0 0	1 1 1 0
9	1 1 0 1	1 0 1 0
10	1 1 1 1	
11	1 1 1 0	
12	1 0 1 0	
13	1 0 1 1	
14	1 0 0 1	
15	1 0 0 0	

BINÁRNÍ KÓDY – Vyjma zobrazení a vkládání číslicových veličin existují i jiné důvody ke vzniku kódů pro vyjádření čísel.

Cyklický kód - využívá 4-bitovou reprezentaci hexadecimálních nebo dekadických čísel. Sousední kódová slova se liší pouze v jednom bitu.

Grayův kód (zrcadlový) - využití

- Karnaughova mapa
- Mechanické snímání s chybou LSB
- Rychlé A/D převodníky
- Natočení hřídele

Gray+3 stejná vlastnost pro 10 stavů.

PROBLEMATIKA PŘECHODŮ MEZI STAVY - ZABEZPEČENÉ KÓDY

Přechod mezi dvěma sousedními stavy dvojkové informace často generuje nežádoucí stavy v důsledku nestejné doby změny 0 → 1 a 1 → 0). Problém - snímání, rychlé D/A převodníky,atd. Například

001→000→010 nebo 001→011→010

1000→0000→0111 , 1000→1111→0111

Symbol	Johansonův
0	0 0 0 0 0
1	0 0 0 0 1
2	0 0 0 1 1
3	0 0 1 1 1
4	0 1 1 1 1
5	1 1 1 1 1
6	1 1 1 1 0
7	1 1 1 0 0
8	1 1 0 0 0
9	1 0 0 0 0

K odstranění uvedeného problému se používá **teploměrový kód,** jehož stav vytváří sloupec log.1 ukončený řadou log.0.

Uvedené kódy nejsou nijak zabezpečeny proti chybě při přenosu informace. Kódy částečně zabezpečující přenášenou informaci:

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

ZAJÍMAVÉ A APLIKAČNĚ ORIENTOVANÉ KÓDY

Polyadické číselné soustavy – pozice bitu čísla souvisí s vyjádřením jeho hodnoty.

Kód Aikenův – doplňkový kód, vytvořený tvůrcem Harvardské struktury počítačů, který usnadňoval implementaci aritmetických operací.

Telegrafní kódy – vytvořeny pro sériovou komunikaci. Informace doplněna dvěma bity tzv. **start bitem** (log.0) a na konci **stop bitem** (log.1). Dnes se používá pouze mezinárodní telegrafní kód číslo 5 (**ASCII kód**), kdy přenášená informace je vyjádřena 8 bity, jedním start bitem a 1, 1.5, 2 stop bity.

Redundantní notace – Aplikačně orientované číselné systémy, kde jeden bit čísla je vyjádřen číslem z množiny S={1,2,3,…,r-1} a všechny prvky S mohou být váhovány pozitivně nebo i negativně.

Pro r=2 je množina čísel $T_{r=2}$= {-1, 0, 1} reprezentována dvěma bity, které vyjadřují čísla takto: 1 je 10 (1-0), -1 je 01 (0-1) a pro reprezentaci 0 máme dvě vyjádření 00 (0-0) a 11 (1-1). Taková definice redundantního číselného systému vede k více vyjádřením daného čísla. Využívá se pro urychlení např. sčítání.

NEPOLYADICKÉ číselné soustavy – hodnoty cifer čísla nesouvisí s pozicí bitů vyjadřujících číslo ⇒ jsou odvozeny jiným, často speciálním způsobem. Hodnotu čísla nelze definovat pomocí polynomu, protože v každém řádu je jiný (nesoudělný) základ. Například:

Římské číslice: Např. 2007=MMVII, 1955=MCMLV

Číselné soustavy zbytkových tříd – v každém řádu čísla platí jiný číselný základ a cifra je definována pomocí operace dělení modulo Z. Jednotlivé základy čísla jsou celá nesoudělná čísla (např. malá prvočísla) Z = 2, 3, 5, 7, atd. Používají se k zabezpečení zpráv proti chybám. Například

$14_{(10)}$ -> ? $_{(2,3,5)}$ 14 modulo 2 = 0
14 modulo 3 = 2
14 modulo 5 = 4

Číslo $14_{(10)}$ je v soustavě zbytkových tříd dáno hodnotou $024_{(2,3,5)}$.

VYJÁDŘENÍ ZÁPORNÝCH ČÍSEL

Chceme-li pracovat s čísly se znaménkem musíme počet bitů rezervovaný na rozsah čísla zvětšit o jeden bit, který ponese informaci o znaménku. V pevné desetinné čárce se k vyjádření reálných čísel používají tyto formáty:

- **± absolutní hodnota** – Nejvyšší bit = znaménko (0 = +, 1 = -). Zbývající bity určují absolutní hodnotu. Rozsah kladných i záporných čísel pro n bitů je <0, 2^n-1>. Ve vyjádření existují **dvě nuly**, použití záporné nuly 10000...00 se většinou zakazuje. Pro obvodovou implementaci je formát nevhodný, při procesorovém zpracování nevadí.
- **Vyjádření záporného čísla jednotkovým doplňkem** – získáme inverzí všech jeho bitů.

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

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

VYJÁDŘENÍ ZÁPORNÝCH ČÍSEL

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

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

Dvojkový doplněk = jednotkový doplněk plus 1 k nejnižšímu řádu (pro celá i desetinná). Rozsah kladných čísel <0, 2^n-1>, záporných <-1, -2^n>. 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>, záporná čísla <-1, -2^n>. 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	±|A|	^{1}A	^{2}A	$^{1/2}A$
+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 bit
- **Dvojkový** – inverze všech bitů plus 1 k nejnižšímu řádu.
  - **Sériový převod** spočívá v kopírování bitů od nejnižšího bitu po bity vyšší včetně první jedničky. Po první jedničce jsou všechny bity **invertovány**.

ODČÍTÁNÍ S DVOJKOVÝM DOPLŇKEM

Rozdíl čísel A-B za pomoci dvojkového doplňku čísla B realizujeme pomocí součtu A+^{2}B. 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 A= 0 1 0 1 0 1
B_{10}= 13 ^{2}B=1 1 0 0 1 1

A	0 1 0 1 0 1
^{2}B	1 1 0 0 1 1
	1 0 0 1 0 0 0

Výsledek kladný

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

A_{10}= 21 ^{2}A= 1 0 1 0 1 1
B_{10}= 13 B=0 0 1 1 0 1

^{2}A	1 0 1 0 1 1
B	0 0 1 1 0 1
	1 1 1 0 0 0

Výsledek záporný

OBVODOVÁ REALIZACE OPERACE SČÍTÁNÍ A ODEČÍTÁNÍ

Při sčítání a odečítání nastávají tyto limitní případy:

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

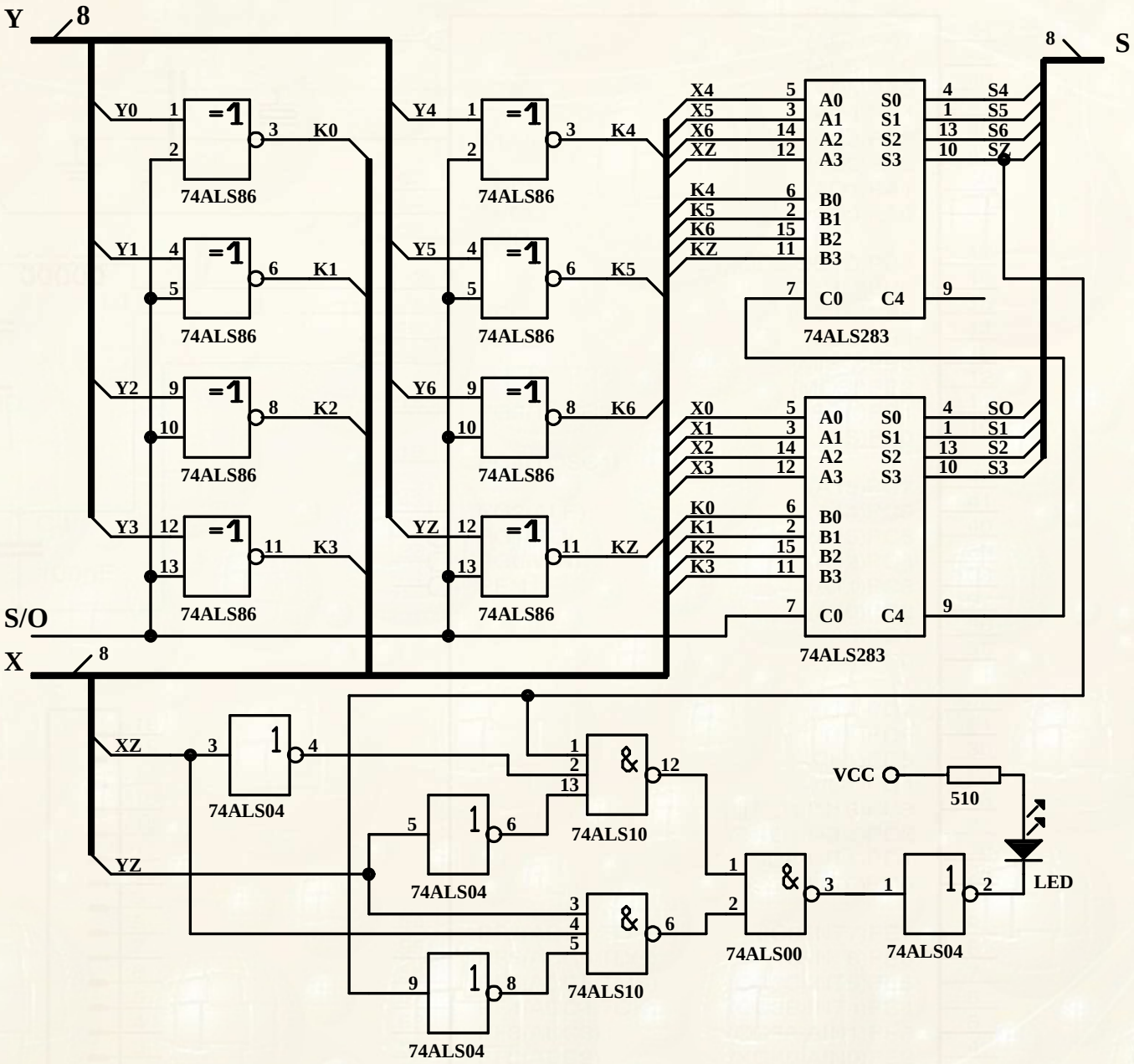
Příznak přetečení identifikuje obvod, který kontroluje znaménka čísel a výsledku.

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

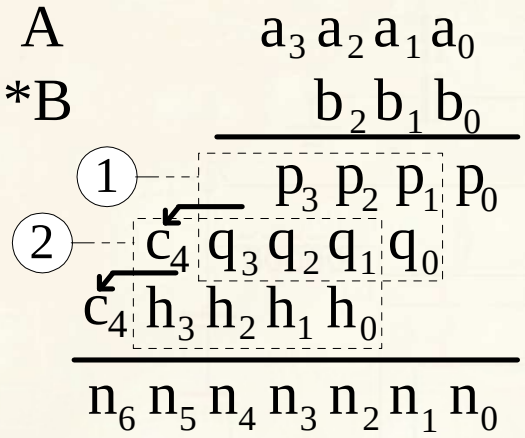
kde z_x, z_y a z_s představují znaménka X,Y a součtu (nebo rozdílu).

Dvojkový doplněk pro odečítání obvodově nevytváříme. Vytvoříme jednotkový doplněk obvody EX-OR (neekvivalence) nebo EX-NOR (ekvivalence) podle polarity signálu S/O. **Dvojkový doplněk** vytvoříme přímo ve **sčítačce** zavedením nenulového počátečního přenosu pro operaci odčítání.

REALIZACE 8-BITOVÉ SČÍTAČKY A ODČÍTAČKY S INDIKACÍ PŘETEČENÍ



OBVODOVÁ REALIZACE OPERACE NÁSOBENÍ



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

Další zkrácení doby výpočtu ⇒ redukce dílčích součinů

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

V Booth nebo modifikovaném algoritmu násobí 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.

		O1(MSB)	O1(LSB)
	*	O2(MSB)	O2(LSB)
		O1(LSB)*O2(LSB)	
	O1(LSB)*O2(MSB)	00000000	
	O1(MSB)*O2(LSB)	00000000	
O1(MSB)*O2(MSB)	00000000	00000000	
VYSL4	VYSL3	VYSL2	VYSL1

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 ⇒ při práci s aritmetikou s pevnou desetinnou tečkou je vhodné **testovat velikost operandů**.

NÁSOBENÍ ČÍSEL SE ZNAMÉNKEM VE DVOJKOVÉM DOPLŇKU

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

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

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

```
A        1.0 0 1 1     = - 13
B        0.1 0 0 1     =   9
-----------------------------
1.1 1 1 1 0 0 1 1
0.0 0 0 0 0 0 0
0.0 0 0 0 0 0
1.1 0 0 1 1
-----------------------------
1.1.1 0 0 0 1 0 1 1    = -117

A        0.1 1 0 1     =  13
B        1.0 1 1 1     =  - 9
-----------------------------
0.0 0 0 0 1 1 0 1
0.0 0 0 1 1 0 1
0.0 0 1 1 0 1
0.0 0 0 0 0
-----------------------------
0 0.0.0 1 0 1 1 0 1 1
  - 0.1 1 0 1
-----------------------------
1 1.1.1 0 0 0 1 0 1 1  = -117

A        1.0 0 1 1     = - 13
B        1.0 1 1 1     =  - 9
-----------------------------
1.1 1 1 1 0 0 1 1
1.1 1 1 0 0 1 1
1.1 1 0 0 1 1
0.0 0 0 0 0
-----------------------------
1 0.1.1 0 1 0 0 1 0 1
  - 1.0 0 1 1
-----------------------------
1 0.0.0 1 1 1 0 1 0 1  = 117
```

Algoritmus je vhodný pro sériovou realizaci násobení.

MODIFIKOVANÝ BOOTHŮV ALGORITMUS

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

Klasický součin B*bit realizuje součet čtyřech produktů

$$B*00111100 = B*2^5 + B*2^4 + B*2^3 + B*2^2$$

Násobitele můžeme nestandardně vyjádřit **0 1 0 0 0 0 0 0 –1 0 0**

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

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

Vyjádření	**Bitové vyjádření**										**Nenulový počet bitů**	
	0	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**, 0**	**Průměrný**	**Max.**
Binární			2^6	2^5		2^3	2^2	2^1			**n/2**	**n**
Kanonické znaménkové		2^7			-2^4			-2^1			**n/3**	**n/2**
Modif.Bootův algoritmus R.4	**0**		**+2** 2.2^6		**-1** -1.2^4		**0**		**-2** -2.2^0		**---**	**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 ⇒ 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 a –B.

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

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

Y	0 1 1 0 1	=	13
X	0 0 1 1 1	=	7
0 0 0 0 0 1 1 0 1			součin s x0
0 0 0 0 1 1 0 1			součin s x1
0 0 0 1 1 0 1			součin s x2
0 0 0 0 0 0			součin s x3
0 0 0 1 0 1 1 0 1 1		=	91

Y	0 1 1 0 1	=	13
X	0 0 1 1 1	=	7
1 1 1 1 1 0 0 1 1			součin s x1,x0 -B
0 0 1 1 0 1 0			součin s x3,x2 (01+1).4 B
1 0 0 1 0 1 1 0 1 1		=	91

přenos

MODIFIKOVANÝ BOOTHŮV ALGORITMUS - RADIX4

$$\tilde{A}.\tilde{B} = \left[-\tilde{a}_{n-1}.2^{n-1} + \sum_{i=0}^{n-2} \tilde{a}_i.2^i \right].\tilde{B} + \tilde{a}_{-1}.\tilde{B}.2^0 =$$

$$-\tilde{a}_{n-1}.\tilde{B}.2^{n-1} + \tilde{a}_{n-2}.\tilde{B}.2^{n-2} + \ldots + \tilde{a}_1.\tilde{B}.2^1 + \tilde{a}_0.\tilde{B}.2^0 + \tilde{a}_{-1}.\tilde{B}.2^0 =$$

$$= (\tilde{a}_{n-2} - \tilde{a}_{n-1}).\tilde{B}.2^{n-1} + (\tilde{a}_{n-3} - \tilde{a}_{n-2}).\tilde{B}.2^{n-2} + \ldots$$

$$\ldots + (\tilde{a}_2 - \tilde{a}_3).\tilde{B}.2^3 + (\tilde{a}_1 - \tilde{a}_2).\tilde{B}.2^2 + (\tilde{a}_0 - \tilde{a}_1).\tilde{B}.2^1 + (\tilde{a}_{-1} - \tilde{a}_0).\tilde{B}.2^0 =$$

$$= (\tilde{a}_{n-2}.2 - \tilde{a}_{n-1}.2 + \tilde{a}_{n-3} - \tilde{a}_{n-2}).\tilde{B}.2^{n-2} + \ldots$$

$$\ldots + (-\tilde{a}_2.2 - \tilde{a}_3.2 + \tilde{a}_1 - \tilde{a}_2).\tilde{B}.2^2 + (\tilde{a}_0.2 - \tilde{a}_1.2 + \tilde{a}_{-1} - \tilde{a}_0).\tilde{B}.2^0 =$$

$$= (-\tilde{a}_{n-1}.2 + \tilde{a}_{n-2} + \tilde{a}_{n-3}).\tilde{B}.2^{n-2} + \ldots$$

$$\ldots + (-\tilde{a}_3.2 + \tilde{a}_2 + \tilde{a}_1).\tilde{B}.2^2 + (-\tilde{a}_1.2 + \tilde{a}_0 + \tilde{a}_{-1}).\tilde{B}.2^0$$

Bude-li koeficient 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?

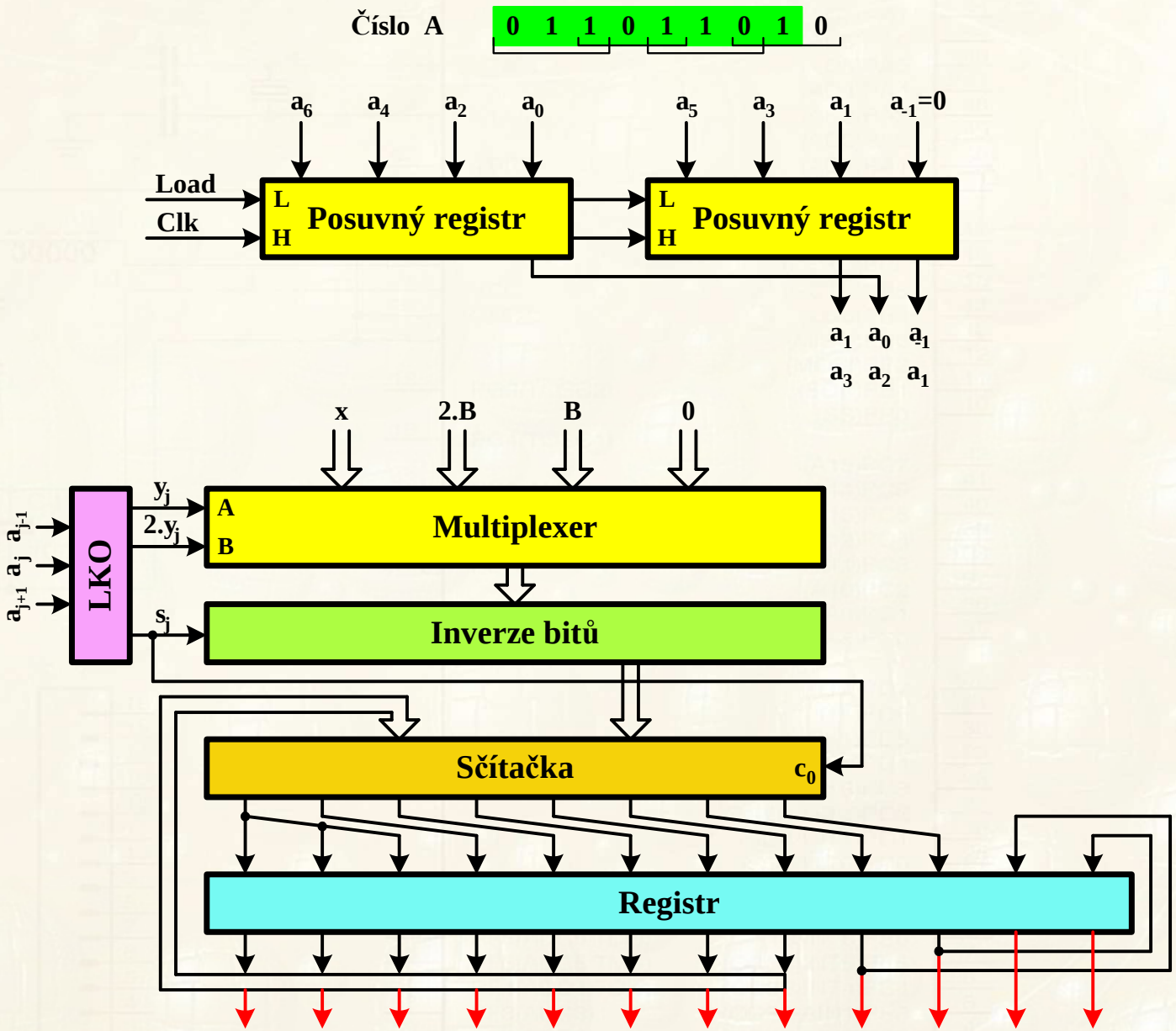
Proces se dá plně **paralelizovat**.

MODIFIKOVANÝ BOOTHŮV ALGORITMUS - RADIX4

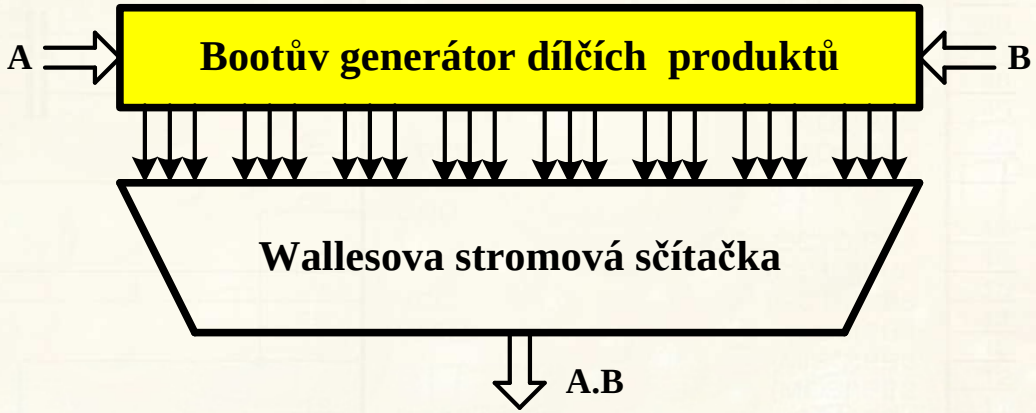
Příklad na sériově realizovaný součin čtyřbitových čísel 1110 (-2) a 1101 (-3) (Výsledek = 0000 0110 (+6)) modifikovaným Boothovým algoritmem-Radix4. Součin vzniká v **ALU** a pomocném posuvném **registru** i s pomocným **bitem** pro počáteční hodnotu a_{-1}.

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

MODIFIKOVANÝ BOOTHŮV ALGORITMUS - RADIX4 - SÉRIOVÝ



MODIFIKOVANÝ BOOTHŮV ALGORITMUS - RADIX4 – PARALELNÍ



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**)

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

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

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

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

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

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

Krajní hodnoty exponentu e+127=0 a e+127=255 se využívají k vyjádření 0 a ±∞. Číslo je vyjádřeno 4 byty

$$\text{MSB} = z\ e_7e_6e_5e_4e_3e_2e_1$$
$$e_0\ m_{-1}m_{-2}m_{-3}m_{-4}m_{-5}m_{-6}m_{-7}$$
$$m_{-8}m_{-9}m_{-10}m_{-11}m_{-12}m_{-13}m_{-14}m_{-15}$$
$$\text{LSB} = m_{-16}m_{-17}m_{-18}m_{-19}m_{-20}m_{-21}m_{-22}m_{-23}$$

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

Číslo		Vyjádření		
		z	exponent	mantisa
$1{,}000.10^0$	$1{,}000000.2^0$	0	01111111	0000000000
$8{,}000.10^0$	$1{,}000000.2^3$	0	10000010	0000000000
$0{,}000.10^0$	$0{,}000000.2^0$	0	00000000	0000000000
$-1{,}750.10^0$	$1{,}750000.2^0$	1	01111111	1100000000
$1{,}250.10^2$	$1{,}953125.2^6$	0	10000101	1111010000
$-1{,}250.10^2$	$-1{,}953125.2^6$	1	10000101	1111010000
$-7{,}812.10^{-2}$	$-1{,}249920.2^{-4}$	1	01111011	0011111111

VYJÁDŘENÍ ČÍSEL V POHYBLIVÉ DESETINNÉ ČÁRCE

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

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

Počet bitů exponentu **m** bude dán vztahy

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

Násobení čísel A a B v pohyblivé čárce můžeme vyjádřit takto

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

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 ⇒ operace není **lineární a výsledek je zatížen chybou**

$$A.B = (-1)^{z_A \oplus z_B}.M_A.M_B.2^{e_A+e_B} = (-1)^{z_V}.m_V.2^{e_A+e_B} =$$

$$= \textit{normování} = (-1)^{z_V}.M_V.2^{e_V}$$

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

OPERACE V POHYBLIVÉ DESETINNÉ ČÁRCE

Při sčítání a odečítání je situace následující

$$A \pm B = (-1)^{z_A} . M_A . 2^{e_A} \pm (-1)^{z_B} . M_B . 2^{e_B} =$$

$$(-1)^{z_A} . M_A . 2^{e_A} \pm (-1)^{z_B} . m_B . 2^{e_A} =$$

$$\left[(-1)^{z_A} . M_A \pm (-1)^{z_B} . m_B\right] . 2^{e_A} =$$

$$(-1)^{z_V} . m_V . 2^{e_A} = normování = (-1)^{z_V} . M_V . 2^{e_V}$$

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

!

- **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 nepoužívat **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;**