REALIZACE ČÍSLICOVÝCH OBVODŮ

Nyní máme několik možných způsobů realizace logických a číslicových obvodů programovatelnými prostředky. Volba závisí:

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

➢ **Programovatelné obvody**

- ❖ ***RTL*** **-** Zápis logické funkce (obdoba realizace logickými obvody)
- ❖ ***Jazyky HDL***
  - Behaviorální zápis (popis chování LKO a LSO)
  - Strukturální zápis (LO je rozděn na funk. části)

➢ **Programem** (popis vývojovým diagramem)

- ❖ ***Mikroprogramování*** – 1950 M.V.Wilkes, logická návrhová technika přinášející řadu výhod oproti technice založené na Boolovských rovnicích a stavových diagramech (μP řadiče, Dnes PC).
- ❖ ***Assembler*** (Strojový kód) – realizace problému instrukcemi daného procesoru. Dnes – ovladače, knihovny jazyků, kontrola zpoždění
- ❖ ***Jazykem C*** – zápis pro popis složitějších problémů operacemi, které programovací jazyk poskytuje. Základní operace jazyka C jsou napsány v assembleru a uloženy v relativním podobě v knihovních funkcích.

Číselné základy

OBVYKLÉ ČÍSELNÉ ZÁKLADY

V číslicových systémech standardně používáme tři číselné základy:

- **Základ 2** – se symboly 0 a 1 (někdy 0 a I)
  - Využívá se k realizaci aritmetických a logických operací v počítačích, mikroprocesorech a programovatelných logických polích. Hodnota v dvojkové soustavě může být použita při psaní programu v jazyce symbolických adres (Asembleru).
- **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

Metoda spočívá v postupném odečítání zmenšujících se mocnin základu B, do kterého chceme číslo z jiného základu převést. Koeficienty hledaného čísla získáváme v pořadí jak jdou za sebou. Jednoduchý a často používaný. **Nevýhoda** – doba převodu závisí na hodnotě převáděného čísla.

2. Metoda postupného dělení základem

Vydělíme-li číslo N_M základem B, potom získáme celou část podílu a zbytek, který bude představovat koeficient a_0. Dalším dělením celé části získáváme jednotlivé koeficienty v pořadí od nejmenšího a_0 k největšímu a_k. **Nevýhodná** pro procesory bez dělení nebo z krátkou délkou slova.

3. Metoda postupného násobení základem

Pro převod čísla $N_M<1$ do základu B budeme násobit číslo N_M 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.

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:

$$10111001010 1,0011_2 = 1011 \mid 1001 \mid 0101, \mid 0011 = B95,3_{16}$$

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

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

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 (obvykle 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

U VÁHOVANÝCH KÓDŮ má každý bit přiřazenu určitou váhu w_j. Součet jednotlivých vah, nabývají hodnoty 1, se rovná dekadickému číslu reprezentovaného 4-bitovou kombinací. Formátu nevyužíváme šest kombinací (10 až 15) ⇒ BCD čísla zabírají větší počet bitů, než jeho dvojkový ekvivalent.

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

Některé kódy mohou být je nejednoznačné např. 6 4 2 -3, 2 4 2 1 u kterých je číslo 7 vyjádřeno stejně (1101 nebo 1101).

Váhované kódy, jejichž součet vah dává hodnotu 9, mají tu vlastnost, že 9-kový komplement čísla N (9-N) je reprezentován kódem, který představuje inverzi jednotlivých bitů kódové reprezentace čísla N (tzv. jednotkový doplněk). Kódy mající takovou vlastnost jsou známy jako **komplementární kódy** (self-complementing) a usnadňují implementaci aritmetických operací. Např. Aikenův kód.

Tuto vlastnost mají pouze 4 pozitivní soustavy (2 4 2 1), (3 3 2 1), (4 3 1 1), (5 2 1 1) a 13 s kladnými i zápornými váhami.

NEVÁHOVANÉ BINÁRNÍ KÓDY

NEVÁHOVANÉ KÓDY – jednotlivé bity nemají přiřazenu konkrétní váhu.

Typickým neváhovaným kódem je kód **excess-3** (shodný s BCD+3). Kód je vytvořen přidáním hodnoty 3 k dekadickému číslu a výsledek je omezen na 4-bitové binární číslo.

Kód **exceess-3** je komplementárním (doplňkovým) kódem a je užitečný při obvodové realizaci dekadických aritmetických operací.

Budeme-li sčítat dvě čísla v kódu excess-3, jejichž součet bude větší jak 9, bude současně se součtem generován přenos do dalšího řádu. K příslušnému řádu bude potřeba přičíst hodnotu 3. Bude-li součet do hodnoty 9, pak bude v daném řádu potřeba odečíst hodnotu 3. Budeme-li čísla v tomto kódu odečítat, potom v daném řádu bude potřeba přičíst hodnotu 3.

SČÍTÁNÍ DEKADICKÝCH ČÍSEL V KÓDU EXCEESS-3

Jednou z dříve používaných možností realizovat operaci s dekadickými čísly je použití kódu exceess-3 (BCD+3). V případě součtu čísel v tomto kódu 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

pro $V+6 \in \langle 16;24 \rangle_{mod16}$, pak k výsledku přičteme číslo 3

```
  535          1000  0110  1000          135          0100  0110  1000
  637          1001  0110  1010          194          0100  1100  0111
------přenos---1------0-------1----------  -----přenos---0-------1-------0---------
 1172   0001  0001  1101  0010          329    0000  1001  0010  1111
          +11   +11   -11   +11                  +11   -11   +11   -11
-----------------------------------------  -----------------------------------------
       0100  0100  1010  0101                 0011  0110  0101  1100
         1     1     7     2                    0     3     2     9

  535          1000  0110  1000          535          1000  0110  1000
  675          1001  1010  1000          114          0011  0011  0111
------přenos---1------1-------1----------  -----přenos---0------0-------0---------
 1210   0001  0010  0001  0000          649    0000  1011  1001  1111
          +11   +11   +11   +11                  +11   -11   -11   -11
-----------------------------------------  -----------------------------------------
       0100  0101  0100  0011                 0011  1010  0110  1100
         1     2     1     0                    0     6     4     9
```

SČÍTÁNÍ DEKADICKÝCH ČÍSEL V KOMPRIMOVANÉM BCD KÓDU

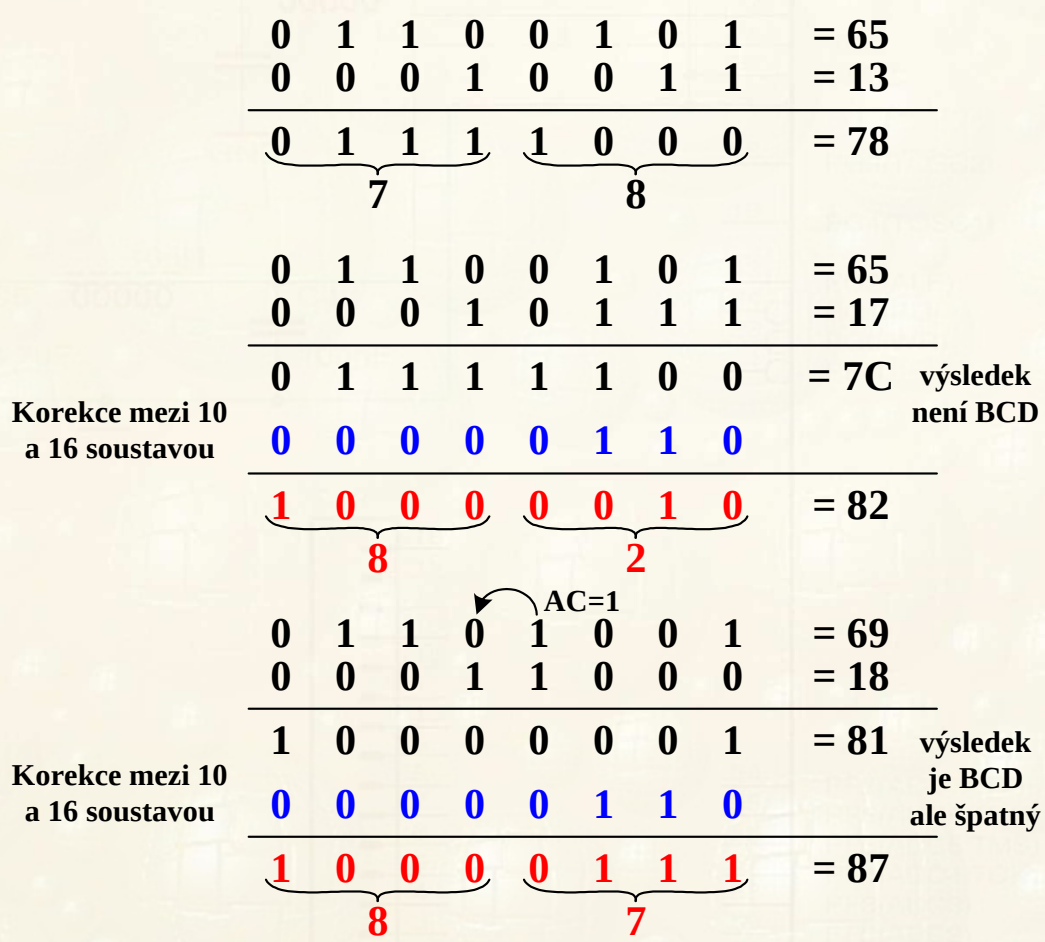
Máme-li operandy v komprimovaném BCD (packed BCD) kódu a potřebujeme realizovat jednodušší operace (sčítání, odečítání, případně snadný součin), je lepší operaci realizovat v BCD kódu. ALU pracují v dvojkové soustavě, takže součet BCD čísel nemusí být správný. Korekci po součtu (někdy i rozdílu) BCD čísel realizuje

DEKADICKÁ KOREKCE.

Musí být uplatněna bezprostředně po součtu nebo rozdílu dvou BCD čísel.

Korekce (mezi soustavou 16 a 10) spočívá v přičtení hodnoty 6 , pokud spodní 4 bity představují číslo >9 nebo došlo k přenosu mezi 4 a 5 bitem (částečný přenos).

Stejná operace je uplatňována na horní 4 bity (cifru), jestliže je >9 nebo došlo k přenosu do dalšího řádu. Přičítá se 60h.



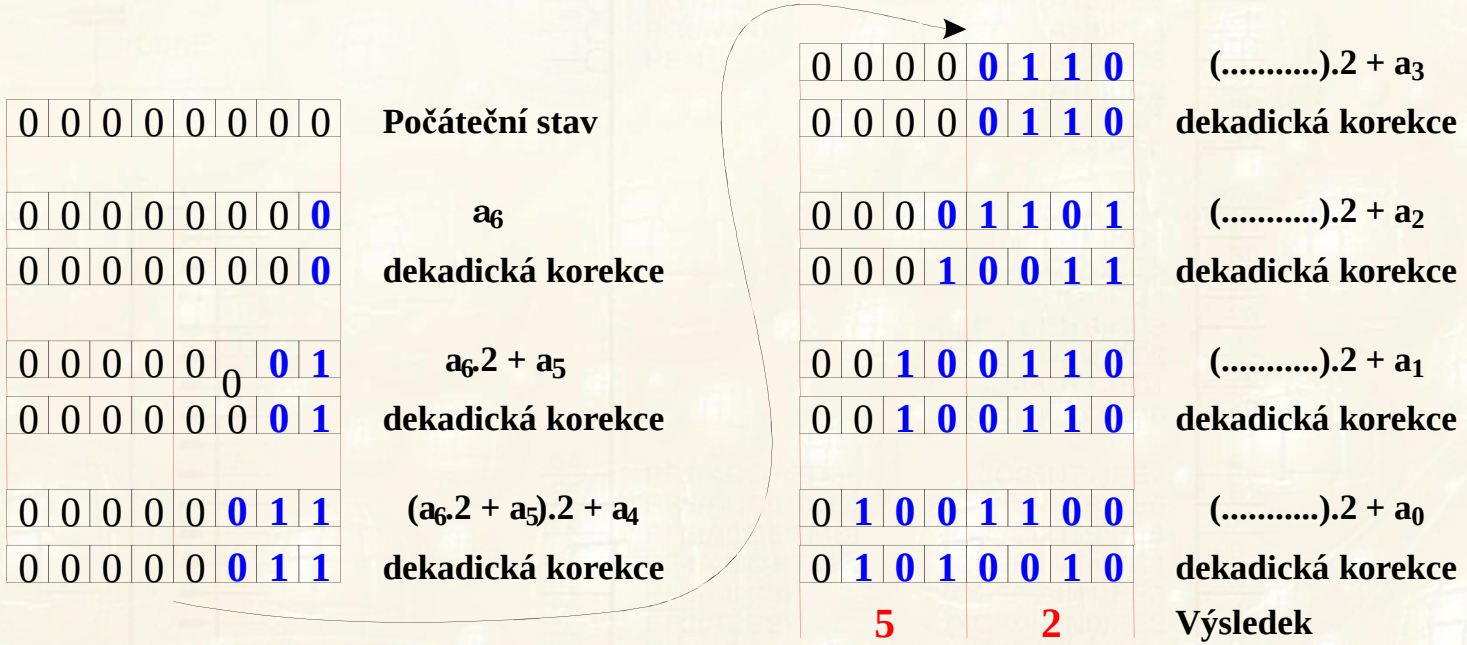
PŘEVOD BINÁRNÍ HODNOTY DO BCD KÓDU

V jazyce C můžeme použít funkci sprintf – díky univerzálnosti se strojový kód výrazně zvětší (může být problém u malých procesorů).

Binární číslo s pevnou desetinnou čárkou rozdělíme na celou a desetinnou část. Celou část převáděného čísla vyjádříme Hornerovým schématem

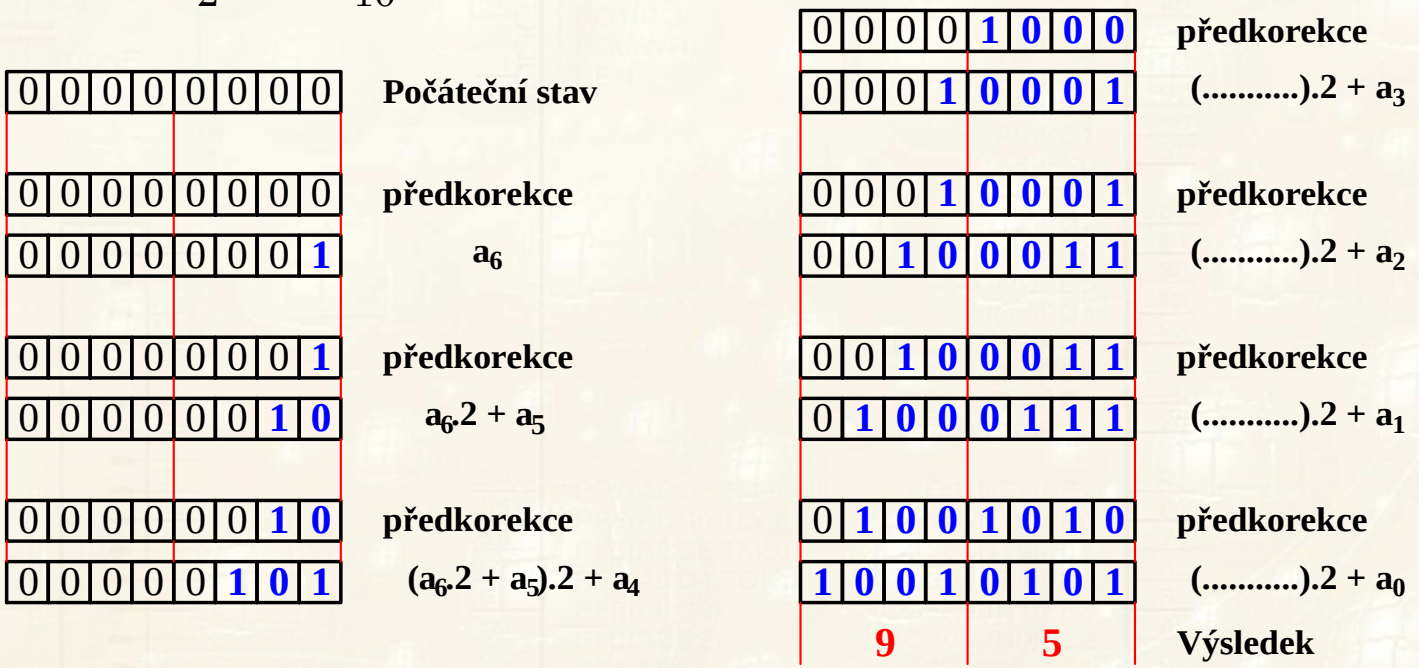
$$N = ((a_n.2 + a_{n-1}).2 + a_{n-2}).2 + ... + a_0$$

Závorky představují součet dvou stejných čísel (.2) s hodnotou 0 nebo 1 ($a_{n-i} \in <0,1>$). Bude-li mezivýsledek v BCD kódu, součtem stejných hodnot + 0/1 můžeme výsledek korigovat dekadickou korekcí. Doba převodu je nezávislá na velikosti čísla. Příklad $0110100_2 \rightarrow N_{10}$.



PŘEVOD BINÁRNĚ DEKADICKÝ BEZ DEKADICKÉ KOREKCE

Jsou procesory (AVR, PIC, ARM), které nemají instrukci dekadické korekce. Má-li procesor příznak částečného přenosu (half carry flag) můžeme převod realizovat jak bylo uvedeno.
Nemáme-li částečný přenos realizujeme předstih dekadické korekce. Před dvojnásobkem mezivýsledku a přičtením dalšího bitu testujeme spodní i horní 4 bity, zda po dvojnásobku dojde k potřebě korekce či nikoliv. Budou-li 4 bity číslo <0÷4> korekce není třeba. K číslu <5÷9> realizujeme předstih korekce přičtením hodnoty 3, která po zdvojnásobení bude představovat rozdíl mezi 10 a 16 soustavou. Příklad $1011111_2 = 95_{10}$.



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 (kdy nedochází k současné změně 0 → 1 a 1 → 0) - možná existence nežádoucích stavů (snímání, rychlé D/A převodníky).

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 při realizaci převodníků **A/D flash** a **segmentovaných D/A** se setkáváme s **teploměrovým kódem,** jehož stav vytváří sloupec log.1 ukončený řadou log.0.

Uvedené kódy nejsou nijak zabezpečeny proti chybě při přenosu informace.

Johansonův kód – Spojením vlastností Grayova kódu a částečného zabezpečení vznikl kód, který má pro svoji konstrukci význam při realizacích velmi rychlých čítačů.

ZAJÍMAVÉ A APLIKAČNĚ ORIENTOVANÉ KÓDY

Kód Aikenův – tvůrce Harvardské struktury počítačů, původně určené pro počítací stroje (kalkulačky) vytvořil doplňkový kód, 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.

Zatím byly popisovány tzv. **polyadické číselné soustavy**, u kterých se vyjadřované číslo skládá z bitů, jejichž pozice souvisí s vyjádřením řádového místa.

Nepolyadické číselné soustavy – hodnoty cifer čísla nesouvisí s pozicí bitů vyjadřujících číslo ⇒ jsou odvozeny jiným, často speciálním způsobem. Soustavy nelze definovat pomocí polynomu, protože v každém řádu je jiný (nesoudělný) základ. Takové soustavy se využívají při konstrukci bezpečnostních kódů.

Číselné soustavy zbytkových tříd – patří do nepolyadických soustav. 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. Pro n bitů můžeme zobrazovat kladná i záporná čísla v intervalu <0, 2^n-1>. Ve vyjádření existují **dvě nuly**, použití záporné nuly 10000...00 se většinou zakazuje. Formát je nevhodný pro obvodové implementace, při procesorovém zpracování nevadí.
- **Vyjádření záporného čísla jednotkovým doplňkem** – získáme inverzí všech jeho bitů. $^1A = 2^n - 1 - A = 2^n - 1 - \sum_{i=0}^{n-2} a_i 2^i$

  Hodnota 2^n-1 představuje n bitové číslo 1111…111. Odečteme-li od něj n-1 bitové číslo A, získáme číslo A s invertovanými bity. Rozsah čísla pro n bitů i počet nul (kladná (000..0), záporná (111..1) se shoduje s vyjádřením ± absolutní hodnota. 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 získáme z jednotkového doplňku po přičtení 1 k nejnižšímu řádu (pro celá i desetinná). Pro n bitů vyjadřujeme kladná čísla v intervalu <0, 2^n-1> a záporná čísla v intervalu <-1, -2^n>. Nejvyšší bit opět značí znaménko (0 = +, 1 = -). Ve vyjádření 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. Při n platných bitech můžeme zobrazovat kladná čísla v intervalu < 0, 2^n-1> a záporná čísla v intervalu. < -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

V tabulce jsou uvedeny příklady vyjádření některých racionálních čísel v pevné desetinné čárce.

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 bitů

– **dvojkového** – inverze všech bitů plus hodnota 1 přičtená k nejméně významnému bitu v daném vyjádření. **Sériový převod** dvojkového doplňku 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 $A+{}^2B$. Tím získáváme požadovaný rozdíl A-B zvětšený o hodnotu 2^n. Pro jednotlivé případy můžeme psát

pro A>B

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

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

K dosažení správné hodnoty rozdílu A-B potřebujeme odečíst (oříznout) hodnotu 2^n. Protože se jedná o 1 na bitu vlevo od znaménka do dalšího zpracování ji neuvažujeme.

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

pro A<B

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

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

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:

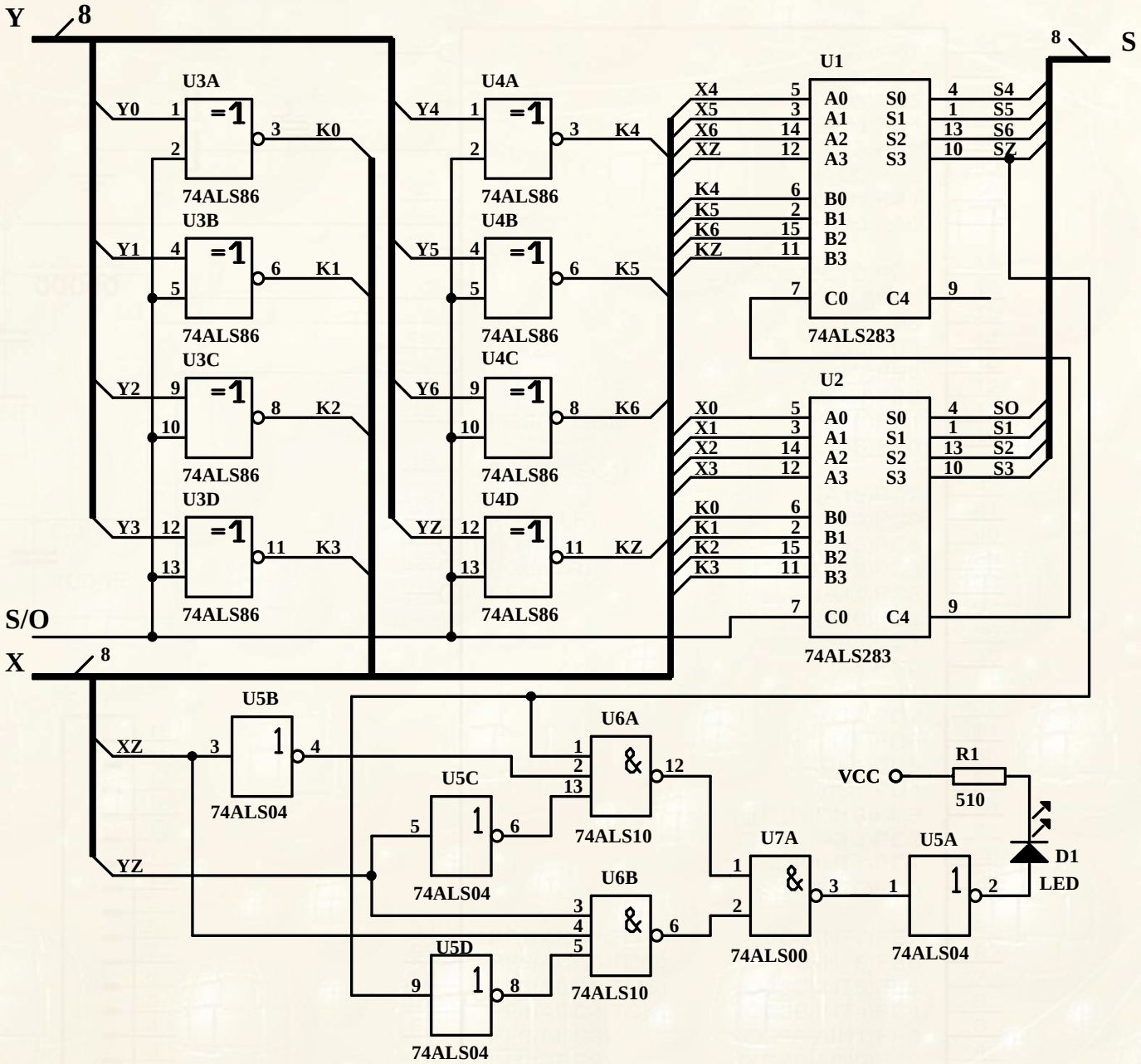
- Součtem dvou kladných čísel bez znaménka dochází k přenosu za nejvyšší bit = indikováno **příznakem přenosu**
- Součtem dvou kladných čísel se znaménkem získáme číslo záporné. Indikováno **příznakem přetečení**
- Součtem dvou záporných čísel získáme číslo kladné. Indikováno **příznakem přetečení**

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

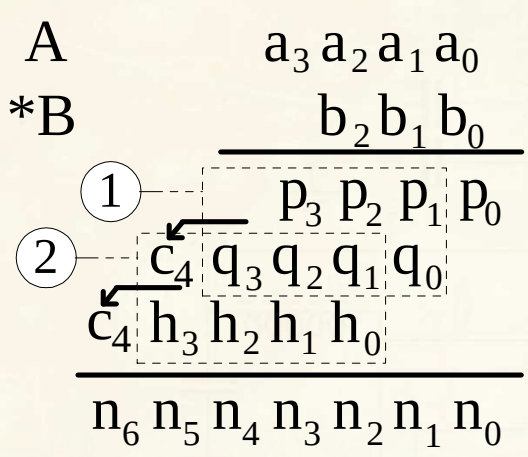
$$Ch = \bar{z}_x.\bar{z}_y.z_s + z_x.z_y.\bar{z}_s$$

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 pomocí obvodů EX-OR (neekvivalence) nebo EX-NOR (ekvivalence) podle polarity signálu S/O a **dvojkový doplněk** vytvoříme přímo na **sčítačce** zavedením nenulového počátečního přenosu pro operaci odčítání.

REALIZACE 8-BITOVÉ SČÍTAČKY A ODČÍTAČKY S INDIKACÍ PŘETEČENÍ



OBVODOVÁ REALIZACE OPERACE NÁSOBENÍ



Klasická obvodová i programová realizace násobení ⇒ aritmetický součin bitů = AND. Dílčí mezivýsledky sčítáme ve sčítačkách s kaskádním přenosem, se zrychleným nebo více stupňovým kanálem přenosu, Wallesovou stromovou sčítačkou.

♣ Programové řešení může být sériové μP bez násobičky. ♣ Procesor má násobičku pro daný počet bitů operandů. ♣ Využití násobičky μP pro rozsáhlejší operandy (viz. staré řešení s pamětmi ROM). Násobená čísla A a B rozdělíme podle možností násobičky do součtu dvou nebo více čísel (pro jednoduchost 4 bity).

$$A=\sum_{i=0}^{3} a_i.2^i=(a_3a_2a_1a_0)=(a_3a_200)+(00a_1a_0)$$

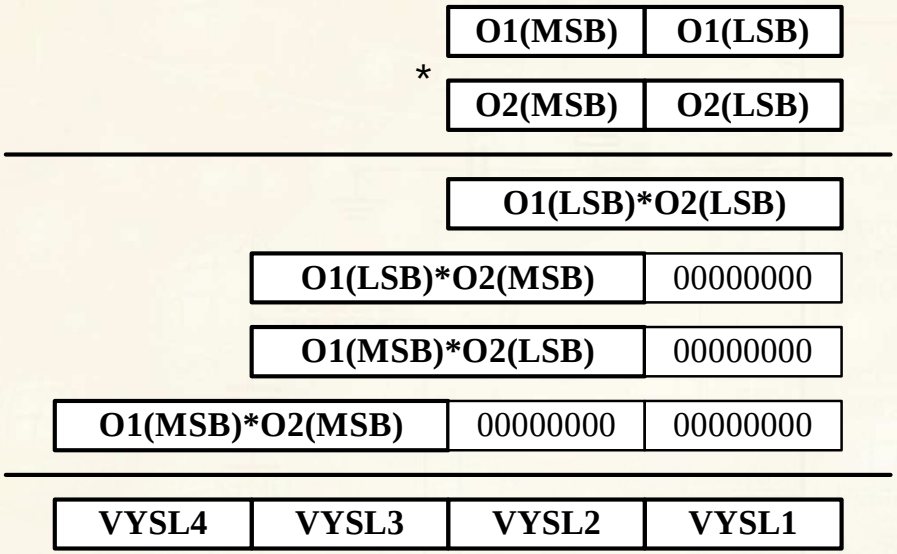
$$B=\sum_{i=0}^{3} b_i.2^i=(b_3b_2b_1b_0)=(b_3b_200)+(00b_1b_0)$$

Výsledný součin N čísel A a B je dán

$$N=(n_7n_6n_5n_4n_3n_2n_1n_0)=(p_3p_2p_1p_0)+(g_3g_2g_1g_000)+(h_3h_2h_1h_000)+(r_3r_2r_1r_00000)$$

kde $p_3p_2p_1p_0=(b_1b_0)*(a_1a_0)$, $g_3g_2g_1g_0=(b_3b_2)*(a_1a_0)$, atd.

PROCESOROVÁ REALIZACE OPERACE NÁSOBENÍ



Příklad výpočtu 16x16bitů na µP s násobičkou 8x8bitů. Má-li v jazyce C výsledek násobení stejný počet bitů jako operandy, přináší to výhody i úskalí. Součin int x a y, int =16bitů. x=1000; y=100; z=x*y;

Bude z=100000 nebo 34464.

Poslední součin v knihovně není ⇒ nutno testovat velikost operandů.

Chceme-li u obvodové realizace násobičky zkrátit dobu výpočtu

- Můžeme urychlit sčítání dílčích součinů
- Zmenšit počet dílčích součtů ⇒
  - Booth algoritmus
  - Modifikovaný Booth algoritmus

Oproti klasickému (bitovému) algoritmu se v Boothově algoritmu násobí dvěma nebo více bity najednou, tak dochází ke snížení počtu sčítaných dílčích součinů na polovinu, třetinu atd.

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

Budeme- li násobit dvěma bity a_{2j+1},a_{2j} = 0, 1, 2, 3. Realizujeme dekodér, který 3 nahrazuje 4–1. B, A jsou bity pro multiplexer, který vytvoří dílčí součin 0, B, 2B a –B. Bit p_{2j+2} je přenos do dalšího řádu (čtyř násobek do dalších dvou bitů). Na příkladu zobrazen součin dvou čísel klasicky a pomocí Boothova algoritmu druhého řadu.

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

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 0	
	1,1 0 0 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 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 –1 0**

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

Při násobení konstantou nám to 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 2^3 2^2 2^1	**n/2**	**n**
Kanonické znaménkové	2^7 -2^4 -2^1	**n/3**	**n/2**
Modif.Bootův algoritmus	**0** **+2** 2.2^6 **-1** -1.2^4 **0** **-2** -2.2^0	**---**	**n/2**

MODIFIKOVANÝ BOOTHŮV ALGORITMUS - RADIX4

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

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

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

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

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

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

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

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

Pokud bude koeficient $a_{-1}=0$, pak výpočet součinu z trojic 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

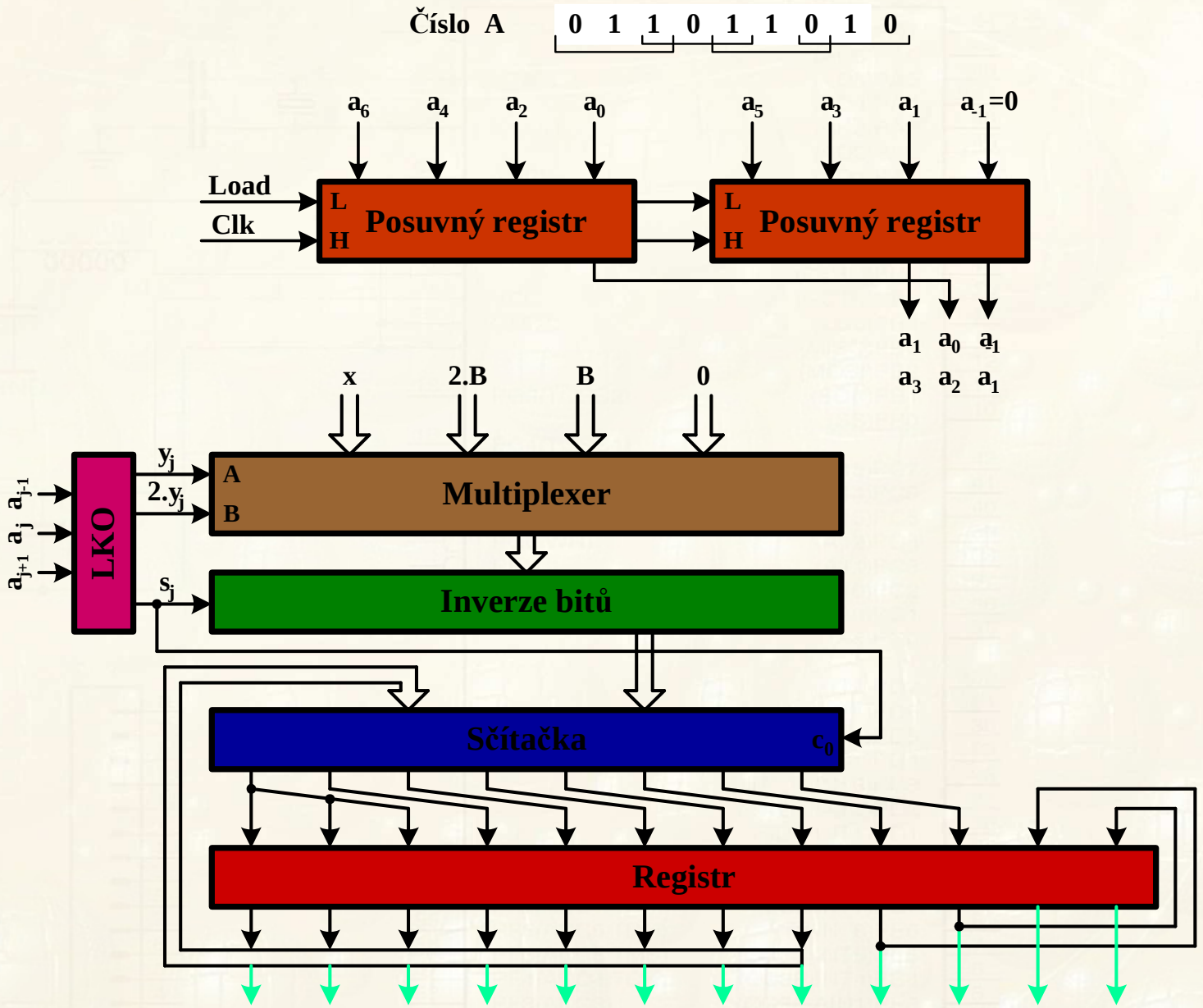
Příklad na sériově realizovaný součin čtyřbitových čísel 1110 (-2) a 1101 (-3) (Výsledek = 0000 0110 (+6)) modifikovaným Boothovým algoritmem-Radix4. Součin vzniká v **ALU** a pomocném **registru** i s pomocným **bitem** pro počáteční hodnotu a_{-1}.

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

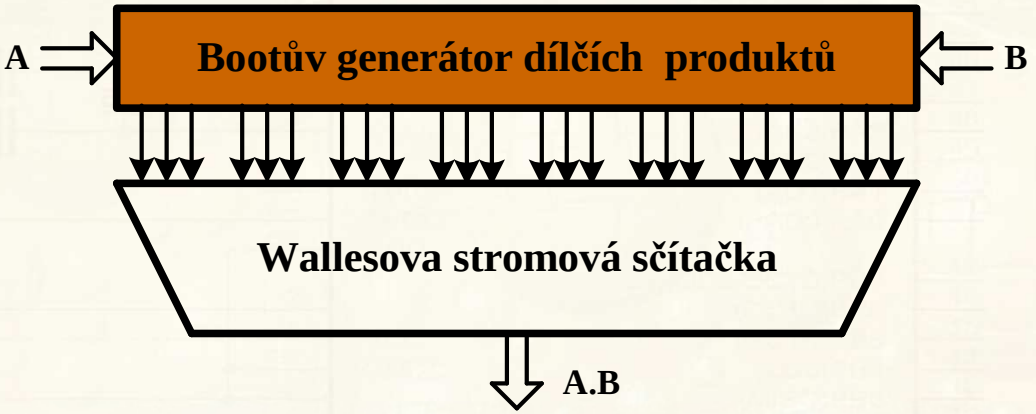
Stejně dospějeme k modifikovanému Boothovu algoritmu – RADIX8 (práce se 4 bity) – dílčí součiny (0, B, 2B, 3B, 4B, -4B, -3B, --2B a –B.

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

MODIFIKOVANÝ BOOTHŮV ALGORITMUS - RADIX4 - SÉRIOVÝ



MODIFIKOVANÝ BOOTHŮV ALGORITMUS - RADIX4 – PARALELNÍ



VYJÁDŘENÍ ČÍSEL V POHYBLIVÉ DESETINNÉ ČÁRCE

Chceme-li pracovat s čísly v pohyblivé desetinné čárce, musíme je rozšířit o číslo vyjadřující hodnotu exponentu včetně jeho znaménka. Číslo s pohyblivou čárkou pak bývá vyjádřeno ve tvaru (standard **IEEE 754-1985**)

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

kde **z** -představuje znaménko čísla, **e** -jeho exponent a **m** -mantisu (desetinné číslo), kterou můžeme vyjádřit výrazem

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

Hodnota závorky (1+m) leží v intervalu <1,0;2,0) (tzv. **normovaná mantisa**). Přesnost mantisy ve formátu ±absolutní hodnota určuje n bitů. Exponent je celé číslo se znaménkem ve formátu čísla s posunutou nulou (+1/2 intervalu zmenšeném o hodnotu 1)

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

kde e leží intervalu <-126;127> pro dvojnásobnou přesnost <-1022;1023>). Krajní hodnoty exponentu e+127=0 a e+127=255 se využívají k vyjádření čísel, která nelze ve formátu daném první rovnicí vyjádřit (tzn. 0 a ±∞).

VYJÁDŘENÍ ČÍSEL V POHYBLIVÉ DESETINNÉ ČÁRCE

Standardizovaný formát čísla v jednoduché přesnosti je vyjádřen 32 bity, které jsou rozděleny do 4 bytů takto:

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

Číslo v dvojnásobné přesnosti je vyjádřeno 64 bity, z nichž 11 bitů je exponent a 52 bitů je mantisa. 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	0111111	0000000000
$8{,}000.10^{0}$	$1{,}000000.2^{3}$	0	1000010	0000000000
$0{,}000.10^{0}$	$0{,}000000.2^{0}$	0	0000000	0000000000
$-1{,}750.10^{0}$	$1{,}750000.2^{0}$	1	0111111	1100000000
$1{,}250.10^{2}$	$1{,}953125.2^{6}$	0	1000101	1111010000
$-1{,}250.10^{2}$	$-1{,}953125.2^{6}$	1	1000101	1111010000
$-7{,}812.10^{-2}$	$-1{,}249920.2^{-4}$	1	0111011	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

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

kde M_A M_B - jsou normované mantisy představující závorku (1+m). Součin $M_A.M_B$ představuje součin čísel v pevné desetinné čárce s tím, že výsledek součinu bude mít dvojnásobný počet bitů a jeho velikost bude ležet v intervalu <1;4). Je-li součin mantis ≥2, bude výsledek posunut o jednu pozici doprava (normovaná mantisa) a exponent zvětšen o 1. Na konci **normování** se realizuje omezení nebo zaokrouhlení součinu mantis na standardní počet bitů. Teoreticky ⇒ 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, jen násobení s pevnou desetinnou čárkou je nahrazeno dělením a součet exponentů je nahrazen rozdílem. Pokud výsledná mantisa bude menší jak jedna, bude posunuta o jednu pozici doleva a exponent zmenšen o jedničku.

OPERACE V POHYBLIVÉ DESETINNÉ ČÁRCE

Při sčítání a odečítání je situace následující

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

Mantisa čísla s menším exponentem je posunována doprava tak dlouho, dokud nedojde ke srovnání exponentů obou čísel. Následuje součet mantis obou čísel (součet v pevné desetinné čárce). Podle výsledku musí být mantisa upravena tak, aby byla v intervalu <1;2). V případě součtu se bude jednat o případný jeden posun doprava a inkrementaci exponentu, v případě rozdílu i o několik posunů doleva a zmenšení exponentu. Nakonec 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 (FP) nepokrývá všechna racionální čísla.

- FP nepoužívat pro finanční operace. Hodnotu 0,1 nelze přesně vyjádřit.
- Nekonečná smyčka vytvořená zápisem programu

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

- Nesprávný počet cyklů při běhu programu

```
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.

- Využití tolerančního rozsahu - možné řešení

```
#define  delta  0.0001
#define  FLOAT_EQ (x,v)    (((v-delta)<x)&&(x< (v+delta)))
double  y=0.0;
while(! FLOAT_EQ (x,1.0) { y+=0.1;  ..... atd.  }
```

- Rozsah mantisy

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

PROBLÉMY PŘI POUŽITÍ POHYBLIVÉ ČÁRKY

Formát v pohyblivé čárce (FP) nepokrývá všechna racionální čísla.

- FP nepoužívat pro finanční operace. Hodnotu 0,1 nelze přesně vyjádřit.
- Nekonečná smyčka vytvořená zápisem programu

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

- Nesprávný počet cyklů při běhu programu

```
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.

- Využití tolerančního rozsahu - možné řešení

```
#define  delta  0.0001
#define  FLOAT_EQ (x,v)    (((v-delta)<x)&&(x< (v+delta)))
double  y=0.0;
while(! FLOAT_EQ (x,1.0) { y+=0.1;  ..... atd.  }
```

- Rozsah mantisy

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