4. Zobrazení čísel v počítači a realizace aritmetických operací

Základní charakteristiky číslicového počítače, který má tzv. von Neumannovu architekturu jsou:

- Instrukce a data jsou uloženy v téže paměti.
- Paměť je organizována lineárně (tzn. jednorozměrně) a je rozdělena na stejně velké buňky, které se adresují celými čísly (zpravidla 0, 1, 2, 3,...).
- Data ani instrukce nejsou explicitně označeny.
- Explicitně nejsou označeny ani různé datové typy.
- Pro reprezentaci dat i instrukcí se používají dvojkové signály.
- V instrukci zpravidla není uváděna hodnota operandu, ale jeho adresa.
- Instrukce se provádějí jednotlivě, a to v pořadí, v němž jsou zapsány v paměti, pokud není toto pořadí změněno speciálními instrukcemi (nazývanými skoky).

Základem je tedy počítání a zobrazení čísel ve dvojkové soustavě. Pomocí nul a jedniček jsme sice schopni zobrazit jakékoli číslo, ale vždy existují nějaká omezení: velikost registrů, velikost aritmetických jednotek pro realizaci operací, doba za kterou výsledek potřebujeme (tedy rychlost výpočtu) apod. Potřebujeme pracovat s čísly kladnými i zápornými, celými i necelými. Co s tím, to se dozvíme v následujících odstavcích.

4.1. Počítání ve dvojkové soustavě

Poziční číselnou soustavu, která je určena svým základem (bází) z, kde z je přirozené číslo větší nebo rovno 2, nazýváme soustavou *z-adickou*. Nejčastěji používáme soustavu desítkovou (dekadickou, $z = 10$), ale číslicový počítač je založený na soustavě dvojkové ($z = 2$). Protože ve dvojkové soustavě je zápis číselných údajů příliš dlouhý, používáme soustavu šestnáctkovou (hexadecimální, $z = 16$), která je příbuzná soustavě dvojkové (její základ je mocnina základu dvojkové soustavy, 4 cifry ve dvojkové soustavě odpovídají jedné cifře v soustavě šestnáctkové). Mocniny dvou jsou v tabulce na obrázku

4.1, přehledová tabulka čísel od 0 do 16 vyjádřené ve dvojkové, desítkové a šestnáctkové soustavě jsou v tabulce na obrázku 4.2.

Pro převod z dvojkové do desítkové soustavy postupujeme podle mocnin 2:

$$10011,101 = 2^4 + 2^1 + 1 + 1/2 + 1/8 = 19,625$$

Pro opačný převod musíme číslo rozdělit na část před řádovou čárkou, kterou postupně dělíme dvěma a zbytky zapisujeme od nultého řádu, a na část za řádovou čárkou, které násobíme dvěma a hodnotu (nulu nebo jedničku), která vyjde před řádovou čárkou, zapisujeme od řádové čárky vpravo. Protože soustava dvojková a desítková nejsou příbuzné, není možné některá čísla zobrazit přesně. Například

$$0,1_{10} = (0,00011001100....)_2$$

n	2^n	Dec.
0	2^0	1
1	2^1	2
2	2^2	4
3	2^3	8
4	2^4	16
5	2^5	32
6	2^6	64
7	2^7	128

n	2^n	Dec.
8	2^8	256
9	2^9	512
10	2^{10}	1 024
11	2^{11}	2 048
12	2^{12}	4 096
13	2^{13}	8 192
14	2^{14}	16 384
15	2^{15}	32 768
16	2^{16}	65 536

n	2^n	Dec.
20	2^{20}	1 M
30	2^{30}	1 G
32	2^{32}	4 G
40	2^{40}	1 T
-1	2^{-1}	0,5
-2	2^{-2}	0,25
-3	2^{-3}	0,125
-4	2^{-4}	0,0625

Obrázek 4.1.: Tabulka mocnin dvou.

Hex.	Dec.	Bin.
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111

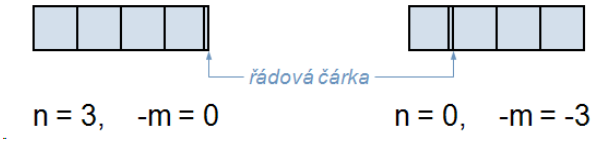
Hex.	Dec.	Bin.
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Obrázek 4.2.: Tabulka čísel 0 až 16 v reprezentaci šestnáctkové, desítkové a dvojkové pro snadné zapamatování.

Při práci s čísly v počítači musíme ještě počítat s omezenou kapacitou registrů, paměti, výkonných jednotek. Řádová mřížka určuje formát zobrazitelných čísel v počítači (tj. definuje nejvyšší řád n a nejnižší řád $-m$). Příklady řádových mřížek jsou na obrázku 4.3.

Základními vlastnostmi jsou:

- Délka řádové mřížky (l) – počet řádů obsažených v řádové mřížce.
- Jednotka řádové mřížky (e) – nejmenší číslo zobrazitelné v řádové mřížce.
- Modul řádové mřížky (M) – nejmenší číslo, které již v řádové mřížce zobrazitelné není.



Obrázek 4.3.: Řádové mřížky délky 4 pro $M = 10000_2$, $e = 1$ a pro $M = 10,000_2$, $e = 0,001$.

Předpokládejme modul $M = 10000_2$ (tedy 16 zobrazitelných čísel od 0 do 15) a zkusme provést následující operaci sčítání:

$$(12 + 7)_{10} = 1100 + 0111 = 10011$$

Získáme výsledek, který se nevejde do řádové mřížky. Jak poznáme, co je správný výsledek: 3_{10} nebo 19_{10}?

Zkusme odečítat:
$(12 - 7)_{10} = 5$, což je ovšem totéž jako $(12 + 9)/mod16$, tedy také 5.

Dvojkově:

$$1100 - 0111 = 1100 + (1000 + 1) = 10101$$

Je správný výsledek 5 nebo 21? A jak poznáme, že je výsledek záporné číslo, když ho neumíme zobrazit? Jako obvykle máme několik možností, ale vždy bude důležité, jaká je řádová mřížka, tedy jaké hardwarové protředky máme k dispozici. A půjde nám vždy o to, abychom nalezli takové algoritmy, způsoby zápisu a realizaci aritmetických operací, které budou maximálně efektivní (tzn. rychlé, škálovatelné a nebude pro ně třeba mnoho hardwarových prostředků). Například si ukážeme, jak použijeme sčítačku i pro odčítání a jak úplně stejně pracovat s čísly nezápornými i v doplňkovém kódu. Důležitý bude bit, který vypadává z řádové mřížky a který nazýváme přenos (*carry*: C).

číslo X	obraz čísla $P(X)$
+0	0 0 0
+1	0 0 1
+2	0 1 0
+3	0 1 1
–0	1 0 0
–1	1 0 1
–2	1 1 0
–3	1 1 1

Tabulka 4.1.: Tabulka pro přímý kód a tříbitová čísla, $M = 1000$.

4.2. Kódy pro zobrazení čísel se znaménkem

Nejpoužívanější číselné kódy pro zobrazení čísel se znaménkem jsou:

- přímý (znaménko a absolutní hodnota, anglicky *sign-magnitude*),
- doplňkový (pro dvojkovou soustavu, *2´s complement*),
- aditivní (s posunutou nulou, *biased*).

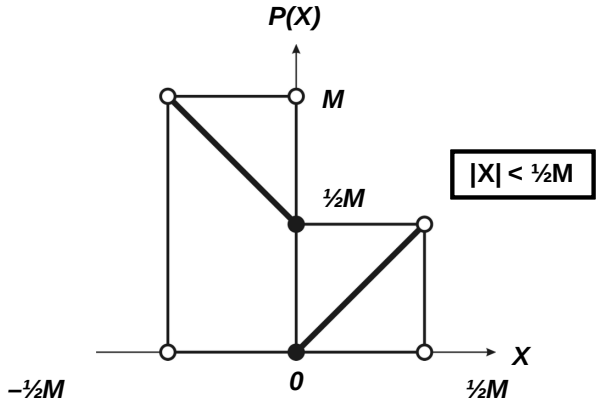
Pro efektivní realizaci aritmetických operací v pevné řádové čárce se používá doplňkový kód (ve vyšších programovacích jazycích jsou takto zobrazena čísla „Integer“). Čísla typu „Real“, tedy v pohyblivé řádové čárce, se vyjadřují pomocí dvou kódů (zvlášť mantisa a exponent). Způsoby vyjádření a počítání s čísly v pohyblivé řádové čárce jsou obsahem odst. 4.3. V následujících odstavcích si ukážeme, že zdánlivě nejpřirozenější vyjádření čísel nemusí vést k jednoduché realizaci výpočtů. Proto začneme s popisem přímého kódu.

4.2.1. Přímý kód

Přímý kód, tedy znaménko a absolutní hodnota, je možné graficky zobrazit podle obrázku 4.4. V tabulce 4.1 nalezneme konkrétní obrazy pro $M = 1000$, tedy pro tříbitová čísla.

Znaménko je reprezentováno číslicí 0 pro plus (+) a číslicí 1 pro mínus (−). Nepříjemností je, že nula má dva obrazy, tzv. kladnou a zápornou nulu, v našem případě 000 a 100. Dalším problémem je, že při realizaci operací musíme pracovat zvlášť se znaménkem a zvlášť s absolutní hodnotou, tedy s nezáporným číslem. Může se stát, že když čísla budou mít nestejná znaménka bude operace sčítání realizována jako odčítání a obráceně. Dále musíme vyzkoumat, jak poznáme, že výsledek odčítání je kladný nebo záporný. Případný záporný výsledek musíme upravit tak, aby byl správný (absolutní hodnota nemůže být záporná).

Hledáme výraz pro odčítání $A - B$. Mějme $M = 1000$ a číslo $B = 101$, $\bar{B} = 010$



Obrázek 4.4.: Přímý kód

potom:

$$B + \bar{B} = 111 = 1000 - 1 = M - 1$$

a odtud:

$$-B = \bar{B} + 1 - M$$

Hledaný rozdíl je tedy:

$$A - B = A + \bar{B} + 1 - M$$

Můžeme udělat závěr, že abychom dostali správný výsledek, musíme mít možnost odečíst modul. To znamená, že musí vyjít přenos!

Příklad:

Mějme modul $M = 10000_2$ a zkusme odečíst $(12 - 7)_{10}$ dvojkově:

$$1100 \ - \ 0111 = 1100 + 1000 + 1 = 10101 = 5_{10}$$

Ale co s tím, když budeme odečítat obráceně, tedy $(7 - 12)_{10}$:

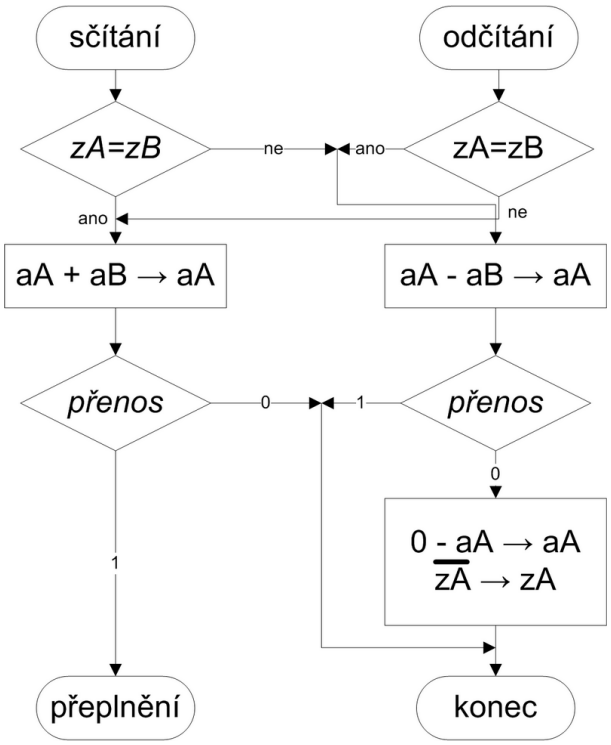
$$0111 - 1100 = 0111 + 0011 + 1 = 01011 = 11_{10}$$

Víme, že musí vyjít –5, ale záporné číslo neumíme zobrazit. Proto musíme použít přímý kód (tedy pracovat se znaménkem) a správné číslice dostat tak, že tento „pseudozáporný výsledek“ odečteme od nuly a změníme znaménko (negujeme znaménkový bit):

$$0 - B = 0 + \bar{B} + 1$$

$$0 + \overline{1011} + 1 \qquad 0 + 0100 + 1 = 0101 \ldots - 5_{10}$$

Algoritmus pro sčítání a odčítání v přímém kódu, kde z označuje bit znaménka a a absolutní hodnotu čísel A a B (výsledek se ukládá do A), je na obrázku 4.5.



Obrázek 4.5.: Algoritmus pro sčítání a odčítání v přímém kódu.

4.2.2. Doplňkový kód

$$\mathrm{D(x)} = \begin{cases} X & pro \quad X \geq 0 \\ M + X & pro \quad X < 0 \end{cases}$$

U doplňkového kódu pracujeme s celým obrazem čísla včetně znaménka. Říkáme, že znaménko je organickou součástí obrazu. Doplňkový kód vyjádřený grafem je na obrázku 4.6. Srovnejte tabulky pro přímý (tabulka 4.1) a doplňkový kód (tabulka 4.2). Všimněte si, že u doplňkového kódu díky jednomu obrazu nuly patří do zobrazitelných čísel i záporná polovina modulu.

Příklady:

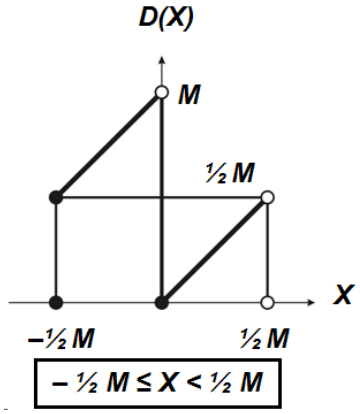
Mějme $M = 10000_2$ a $e = 1$, tzn. že zobrazíme 16 čísel, od –8 do 7.

$$(7 - 4)_{10} ... 0111 + 1100 = 10011$$

Je výsledek 3_{10} *nebo* 19_{10} ?

$$(4 + 7)_{10} ... 0100 + 0111 = 0100 + 0111 = 1011$$

Je výsledek 11_{10} *nebo* -5_{10}?



Obrázek 4.6.: Doplňkový kód.

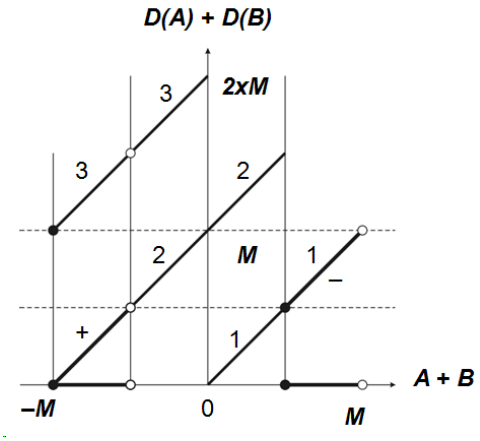
číslo X	obraz čísla $D(X)$
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
–4	1 0 0
–3	1 0 1
–2	1 1 0
–1	1 1 1

Tabulka 4.2.: Tabulka pro doplňkový kód a tříbitová čísla, $M = 1000$.

Jak poznáme, že je výsledek správně? Jestliže si v tabulce vyznačíme všechny možnosti, dojdeme k závěru, že přeplnění pro čísla s různými znaménky nastat nemůže, viz tabulka 4.3. Přičtení modulu M (tedy vlastně přenosu) nemá žádný vliv, takže přenos se ignoruje. Můžeme snadno sčítat i odčítat, protože přičtení záporného čísla znamená odčítání. Detekce nesprávného výsledku pro stejná znaménka znamená nalezení situace, kdy výsledek je číslo, které se nevejde do řádové mřížky pro doplňkový kód. Tato situace je znázorněna na obrázku 4.7, ze kterého vyplývá, že nesprávný výsledek nastane, když sčítáme dvě čísla kladná a dostaneme číslo záporné nebo když výsledkem součtu dvou záporných čísel je číslo kladné. Tento případ nazýváme přetečení nebo přeplnění (overflow), jeho detekci si ukážeme na příkladech. Dobrou pomůckou nám bude znázornění této situace v tabulce pro sčítačku, kde jsou na obrázku 4.8 zvýrazněny tyto dvě možnosti přeplnění. Často dochází k záměně pojmů **přenos** (carry) a **přeplnění** (overflow). Jedním z důvodů může být i podobně znějící výraz v češtině. Přenos je to, co vypadává z řádové mřížky a přeplnění je signalizace nesprávného výsledku.

situace	znaménka čísel A a B	součet obrazů $D(A)+D(B)$	obraz součtu $D(A+B)$
1	$A \geq 0 \wedge B \geq 0$	$A+B$	$A+B$
2	$A \geq 0 \wedge B < 0$	$A+B+M$	$A+B$
2	$A < 0 \wedge B \geq 0$	*dtto*	$A+B+M$
3	$A < 0 \wedge B < 0$	$A+B+M+M$	$A+B+M$

Tabulka 4.3.: Možné výsledky při sčítání v doplňkovém kódu.



Obrázek 4.7.: Graf znázorňující situaci z tabulky 4.3.

4.2.3. Aditivní kód

Další možností, jak vyjádřit čísla se znaménkem jen pomocí 0 a 1 je posunout interval zobrazovaných čísel: přičíst ke všem číslům vhodnou konstantu, viz obrázek 4.9. Nevýhodou je, že nula se nezobrazuje jako nula. Konstanta se obvykle volí jako polovina modulu a obraz čísla X určíme jako $A(X) = X + K$.

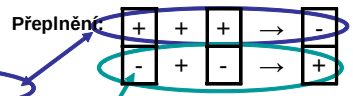
4.3. Pohyblivá řádová čárka

Čísla jsme dosud zobrazovali v pevné řádové čárce (angl. *fixed point*). Rozsah zobrazitelných čísel je vždy omezen (velikostí registrů, ALU, ...). Obzvláště u matematických výpočtů, kde je nutné pracovat zároveň s čísly velmi malými i velkými, je tento způsob zobrazení jen stěží použitelný. Proto se používá zobrazení čísel v pohyblivé řádové čárce (angl. *floating point*). Zobrazujeme uspořádanou dvojici čísel představující mantisu (M, informace o „hodnotě“ čísla) a exponent (e, informace o pozici řádové čárky) a vycházíme ze zobrazení čísla A ve tvaru:

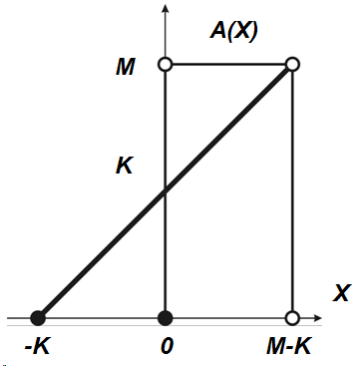
$$A = M.z^e$$

kde z je základ soustavy (obvykle se používá $z = 2$), exponent e bývá obvykle celé číslo. Pro zjednodušení aritmetických operací se používá tzv. **normalizovaný tvar**. Je to takový zápis (tvar) čísla, kdy už nelze mantisu posunout více doleva. Normalizovaný

a	b	p	q	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Obrázek 4.8.: Tabulka pro úplnou sčítačku s vyznačenou situací, kdy dochází k přetečení (overflow).



Obrázek 4.9.: Aditivní kód, $A(X) = X + K$.

tvar operandů nezaručí normalizovaný tvar výsledku, proto je nutné provádět po každé operaci normalizaci, tzn. úpravu výsledku na normalizovaný tvar (posuv mantisy maximálně vlevo). Z popisu vyplývá, že číslo v pohyblivé řádové čárce je zobrazeno ve dvou „podmřížkách“. Podmřížka pro mantisu i exponent musí zobrazovat čísla se znaménkem. Pokud je použit přímý kód pro zobrazení mantisy, tak normalizovaný tvar mantisy bude mít vždy v nejvyšším řádu jedničku. Tuto jedničku můžeme „skrýt“ (tj. vynechat ze zápisu čísla, nezobrazovat) a potom mluvíme o tzv. **skryté jedničce**.

Snaha po universalitě programů a platforem vedla k tomu, že se zavedla norma na zápis čísel v pohyblivé řádové čárce (ANSI/IEEE Std. 754 – 1985) pro 32 a 64 bitová čísla, viz tabulka 4.4. S tímto způsobem zobrazení se běžně setkáte při zobrazení čísel typu „Real“ v programovacích jazycích. (Existují i další možné zápisy, které si vyzkoušíme v sekci Příklady.) Znázornění zobrazení pro 32 bitová čísla je na obrázku 4.10. Exponent používá zobrazení v aditivním kódu s aditivní konstantou $K = 127$ a mantisa je v přímém kódu $|M| < 2$. Podle normy musí být ošetřeny i mezní případy, například obraz nekonečna, nuly apod., viz obrázek 4.11.

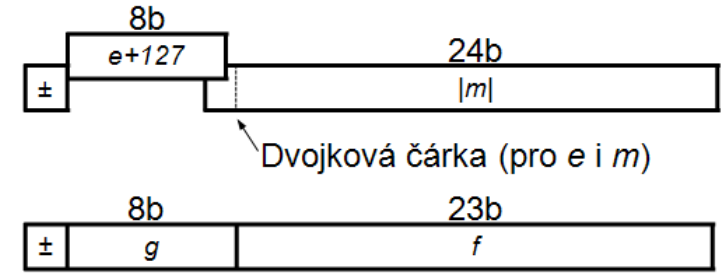
Aritmetické operace realizujeme následujícím způsobem:

- sčítání/odčítání: srovnáním exponentů a sečtením/odečtením mantis,

	znaménko	exponent	mantisa
32b	1b	8b	23(24)b
64b	1b	11b	52(53)b

Tabulka 4.4.: Rozdělení podmřížek podle ANSI/IEEE Std. 754 - 1985.

- násobení: sečtením exponentů a vynásobením mantis,
- dělení: odečtením exponentů a vydělením mantis,
- porovnání: srovnáním exponentů a porovnáním mantis,
- posuv: posunem mantisy nebo zvětšením/zmenšením exponentu.



Obrázek 4.10.: Řádová mřížka pro 32 bitový formát ANSI/IEEE Std. 754 - 1985.

e	*M*	*A*
0	= 0	0
0	≠ 0	$(-1)^s \cdot M \cdot 2^{-126}$
⟨1..254⟩	-	$(-1)^s \cdot (M + 1) \cdot 2^{e-127}$
255	= 0	$(-1)^s \cdot \infty$
255	≠ 0	NaN (*Not a Number*)

Skrytá jednička!

Obrázek 4.11.: ANSI/IEEE Std. 754 – 1985, popis.

4.4. Příklady

1. Realizujte pomocí hradel detekci přetečení (overflow) pro doplňkový kód alespoň dvěma způsoby.

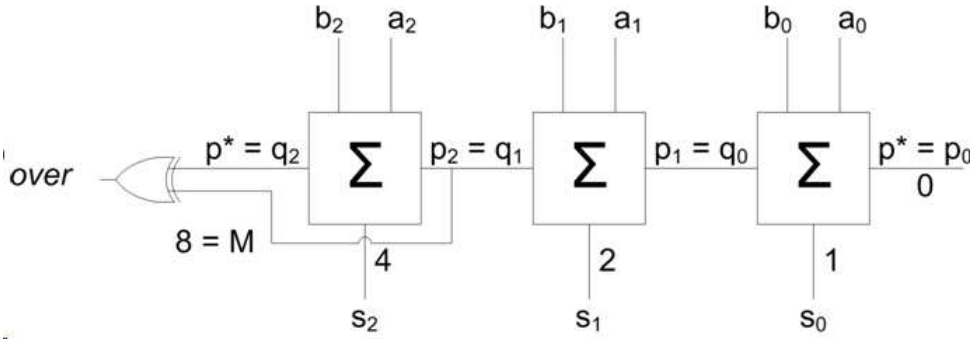
   Řešení
   Vyjdeme z obrázků 4.7 a 4.8. Z nich je patrné, že nesprávný výsledek získáme, pokud při sčítání dvou čísel kladných vyjde číslo záporné (tedy oba sčítance mají v nejvyšším řádu nulu, zatímco v součtu vyjde v nejvyšším řádu jednička) a obráceně. To vyjádříme logickou funkcí:

   a)
   $$over = \overline{a_n} \cdot \overline{b_n} \cdot s_n + a_n \cdot b_n \cdot \overline{s_n}$$

   Z obrázku 4.8 můžeme vysledovat, že tato situace nastává právě tehdy, když přenosy p a q jsou různé, tedy další možnost realizace je pomocí funkce XOR:

   b)
   $$over = p \oplus q$$

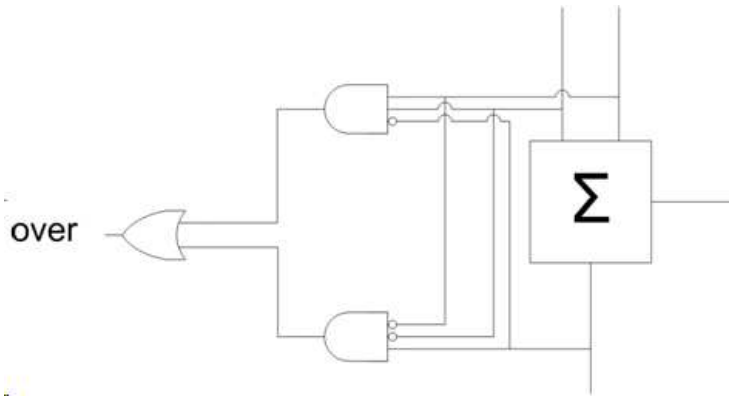
   Realizace pomocí hradel a tříbitovou sčítačku pro realizaci b) je na obrázku 4.12, realizace detekce přetečení podle a) je na obrázku 4.13.



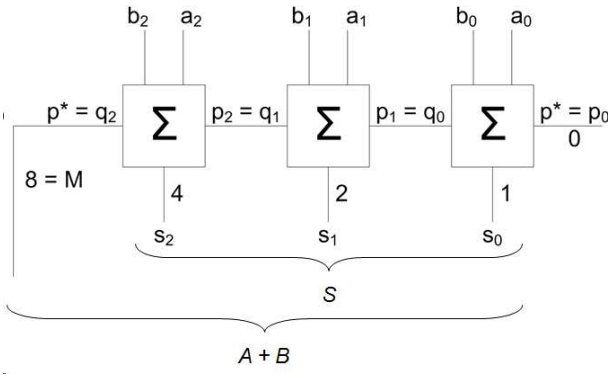
Obrázek 4.12.: Tříbitová sčítačka včetně detekce přetečení.

2. Realizujte 3 bitovou sčítačku-odčítačku v doplňkovém kódu včetně detekce přetečení.

   Řešení
   Celý obvod bude kombinační, jeden blok sčítačky už jsme navrhovali, tedy 3 bitová paralelní sčítačka je sestavena z těchto 3 bloků, viz obrázek 4.14. Odčítačka v doplňkovém kódu znamená realizovat negaci odčítance a přičtení „horké“ jedničky do nejnižšího řádu, viz obrázek 4.15. Realizace možnosti negace vstupů b_i v případě, že řídícím vstupem „odčítej“ na jedničce zvolíme odčítání, je možné snadno realizovat pomocí hradel XOR, viz obrázek 4.16. Sada hradel XOR vlastně realizuje funkci multiplexoru, kde v každém řádu do bloku sčítačky vstupuje místo b_i upravené b'_i podle:
   $$b'_i = \overline{odcitej} \cdot b_i + odcitej \cdot \overline{b_i}$$



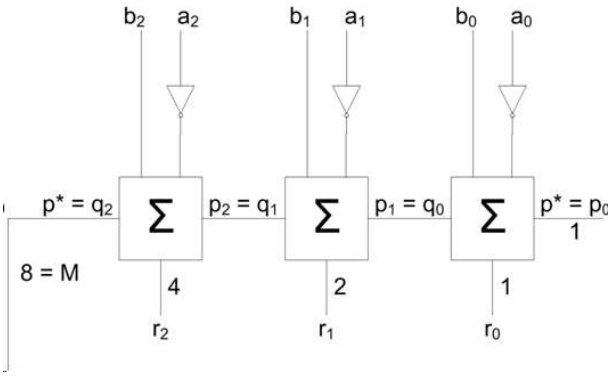
Obrázek 4.13.: Detekce přetečení.



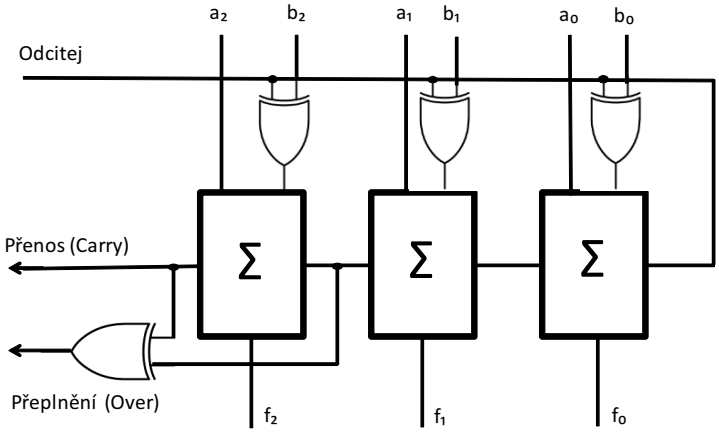
Obrázek 4.14.: Paralelní sčítačka.

3. Navrhněte část aritmeticko-logické jednotky, která bude umět sčítat a odčítat čísla nezáporná a čísla se znaménkem v doplňkovém kódu.

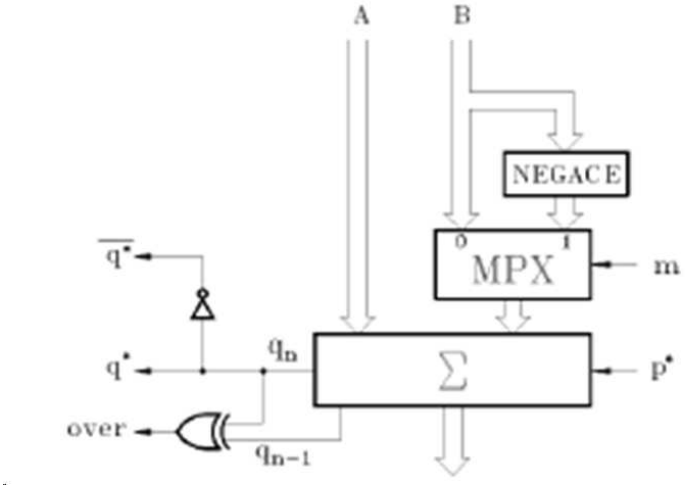
   Řešení
   Sčítání a odčítání je v podstatě stejná operace, jen někdy musíme vstupy B negovat, přivést horkou jedničku a jinak detekovat, zda je výsledek správný. U odčítání se místo přenosu používá jeho negace, která se nazývá výpůjčka. Celé schema (kde blok MPX spolu s blokem NEGACE je možné realizovat pomocí XORů jako na obrázku 4.16), řídící vstupy p^* a m generuje řadič procesoru) je na obrázku 4.17. Takto je běžně realizována aritmeticko-logická jednotka v procesoru. Záleží na programátorovi, které výstupy (informace o výsledcích operace), použije např. v podmíněných skocích, (viz kapitola 5).



Obrázek 4.15.: Paralelní odčítačka.



Obrázek 4.16.: Paralelní sčítačka/odčítačka.



Obrázek 4.17.: Realizace sčítání i odčítání pro čísla nezáporná i v doplňkovém kódu.

4. Navrhněte obvod pro rozšíření řádové mřížky pro doplňkový kód. Uvažujte čtyřbitová a osmibitová čísla, tedy $l = 4$ a $l = 8$.

   Řešení
   Zásada je, že číslo musí mít stejnou hodnotu, jen je vyjádřeno pomocí více bitů (řádů). Rozsah zobrazitelných čísel v doplňkovém kódu je pro 4bitová celá čísla –8 až 7 a pro 8bitová čísla –128 až 127. Obrazy čísel 6 a –6 v obou mřížkách jsou:

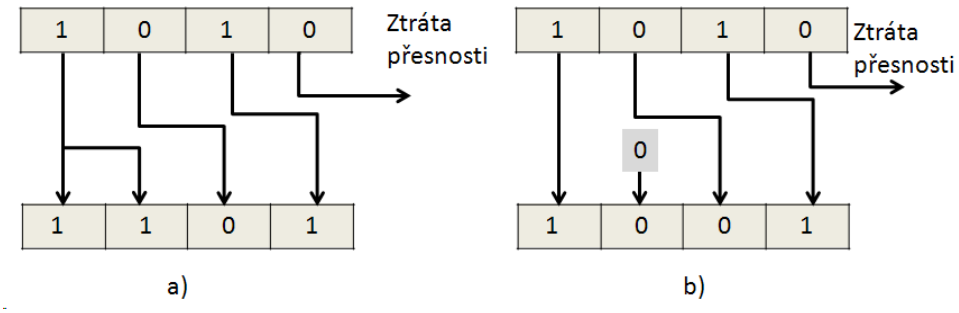
$$l = 4 : D(6) = 0110; D(-6) = 1010$$

$$l = 8 : D(6) = 00000110; D(-6) = 11111010$$

   Odtud je vidět, že realizace rozšíření je jen rozkopírování nejvyššího řádu menší mřížky, tedy pro realizaci není potřeba žádné hradlo, pouze vodiče.

5. Navrhněte realizaci aritmetického posuvu v doplňkovém kódu a v přímém kódu o jedno místo vpravo. Uvažujte 4bitová čísla.

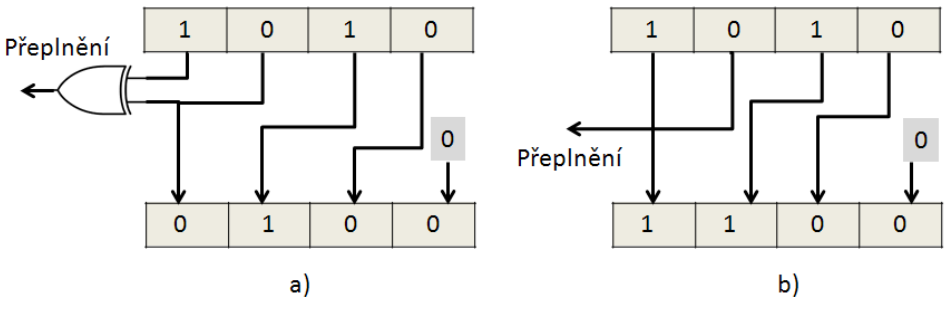
   Řešení Existují tři typy posuvů: logický, cyklický a aritmetický. Rozlišují se podle toho, co se má nasunout na uvolněná místa a zda je důležité to, co vypadává. Obvykle se v procesoru při všech posuvech vypadlý bit ukládá do příznaku „carry“, ale jen někdy se s ním dále počítá. Pro logický posuv se vypadlý bit ignoruje a nasouvají se nuly, pro cyklický posuv se na uvolněná místa nasouvá bit, který vypadne (někdy přes příznak carry, takže je cyklický posuv $l + 1$ bitový). Pro aritmetické posuvy platí, že výsledek musí odpovídat násobení (resp. dělení) mocninou 2 pro posuv vlevo (resp. vpravo). A tady samozřejmě záleží na kódu pro zobrazení čísel se znaménkem, takže příslušná realizace posuvu bude pro každý kód jiná. Konkrétně pro posuv obrazu 1010 což je v doplňkovém kódu –6 a v přímém kódu –2 musí vyjít –3 resp. –1, jestliže vypadne 1, jde o ztrátu přesnosti, viz obrázek 4.18.



Obrázek 4.18.: Aritmetický posuv vpravo, tzn. dělení dvěma a) v doplňkovém a b) v přímém kódu.

6. Navrhněte realizaci aritmetického posuvu v doplňkovém kódu a v přímém kódu o jedno místo vlevo. Uvažujte 4bitová čísla.

   Řešení
   Posuv o jedno místo vlevo musí odpovídat násobení dvěma. Realizace je na obrázku 4.19 a jsou použita stejná čísla jako v předchozím příkladu. V doplňkovém kódu

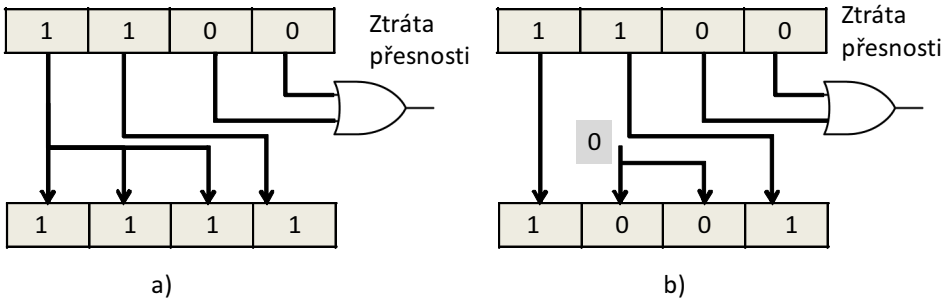


Obrázek 4.19.: Aritmetický posuv vlevo, tzn. násobení dvěma a) v doplňkovém a b) v přímém kódu.

je číslo –12 mimo rozsah zobrazitelných čísel, takže dochází k přeplnění (overflow), v přímém kódu dvojnásobek –2 (tedy –4) lze zobrazit, takže k přeplnění nedojde.

7. Navrhněte realizaci aritmetického posuvu v doplňkovém kódu a v přímém kódu o dvě místa vpravo. Uvažujte 4bitová čísla.

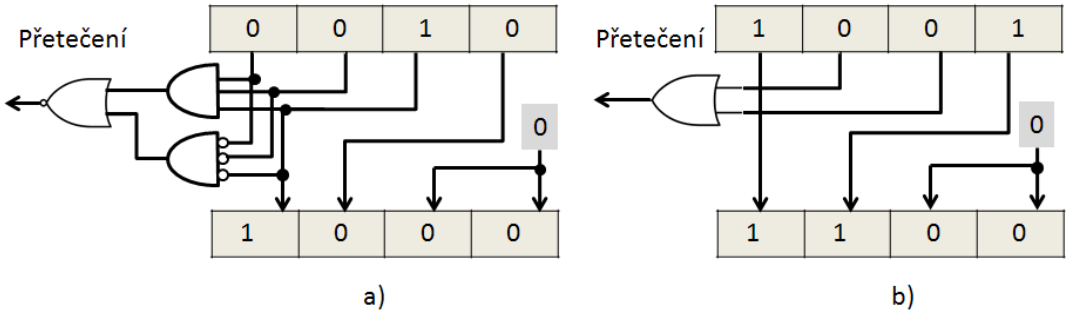
Řešení
Aritmetický posuv o dvě místa vpravo znamená dělení čtyřmi. Na obrázku 4.20 je případ dělení čísla –4, což musí vyjít správně v obou kódech. Realizace je ovšem různá, podle významu jednotlivých bitů.



Obrázek 4.20.: Aritmetický posuv vpravo o dvě místa, tzn. dělení 4 a) v doplňkovém a b) v přímém kódu.

8. Navrhněte realizaci aritmetického posuvu v doplňkovém kódu a v přímém kódu o dvě místa vlevo. Uvažujte 4bitová čísla.

Řešení
Aritmetický posuv o dvě místa vlevo znamená násobení čtyřmi. Na obrázku 4.21 je ukázka pro obraz čísla 2 v doplňkovém kódu (výsledek není správně, protože +8 je mimo rozsah zobrazitelných čísel, vyšlo vlastně –8), v přímém kódu jsme posouvali –1 a vyšlo –4, což je správný výsledek.

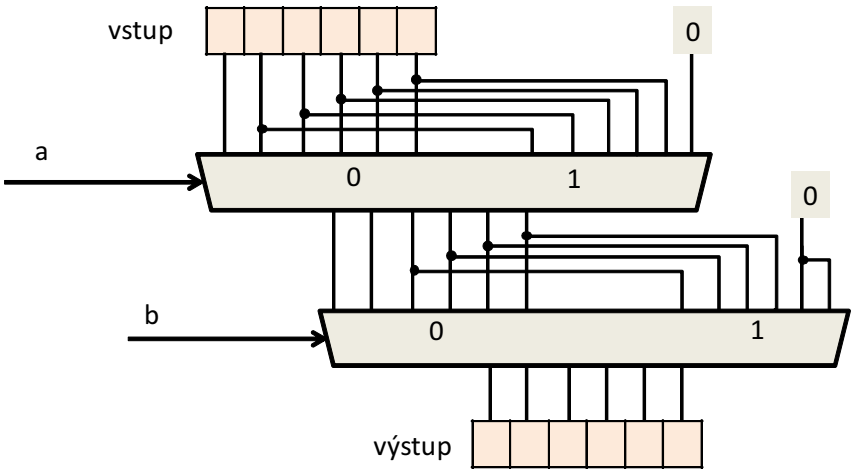


Obrázek 4.21.: Aritmetický posuv vlevo o dvě místa, tzn. násobení 4 a) v doplňkovém a b) v přímém kódu.

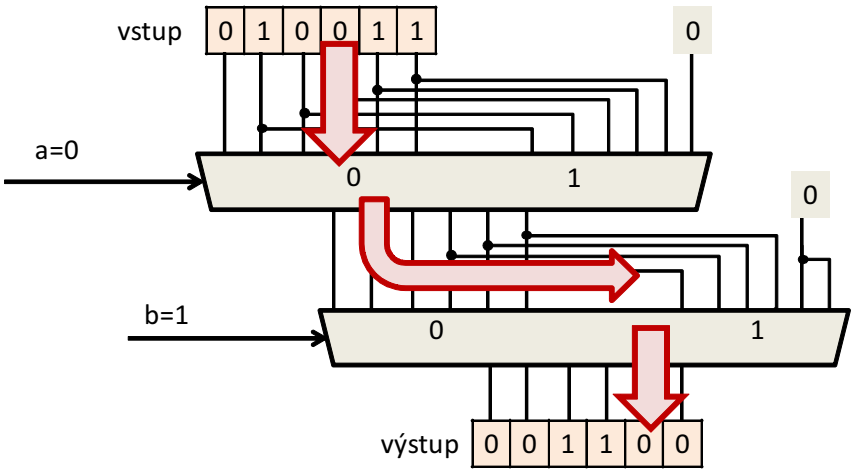
9. Navrhněte realizaci logického posuvu o 0 až 3 místa vlevo. Uvažujte 6bitová čísla.

Řešení

Řešením je tzv. „barrel shifter“, který se realizuje pomocí multiplexorů a obvodů pro posuv. Posuvy jsou realizovány pouze vodiči podle konkrétního typu posuvu. Pro náš konkrétní případ budeme potřebovat dva řídící vstupy a a b (řídící vstup a bude posouvat o 1 bit a b o dva bity) a dva multiplexory s dvakrát šesti vstupy. Detekovat přeplnění pro logický posuv není třeba. Výsledné schéma je na obrázku 4.22. Příklad konkrétního posuvu pro $ba = 10$, tedy posuv o dvě místa vlevo je znázorněn na obrázku 4.23.



Obrázek 4.22.: Realizace posuvů z příkladu 9.



Obrázek 4.23.: Posuv 010011 o dvě místa vlevo.

10. Jsou dány operandy 21, 95 a E5, AF v šestnáctkové soustavě. Uvažujte následující případy:
a) jde o čísla nezáporná (bez znaménka)
b) jde o čísla v doplňkovém kódu
c) jde o čísla v přímém kódu
Stanovte hodnotu součtů (v šestnáctkové soustavě) a určete zda došlo k přeplnění. Předpokládejte použití 7 bitové sčítačky v případě c) a 8 bitové v ostatních případech. V případě b) a c) určete znaménka obou operandů a výsledku, pro všechny sčítance a výsledky určete čísla, jichž jsou obrazem, tzn. např. AA je v doplňkovém kódu obrazem čísla –56.

Řešení
Víme, že sčítání je realizováno úplně stejně pro případy a) a b), takže výsledek ve sčítačce bude stejný, rozdílná může být pouze jeho interpretace a rozhodnutí o tom, zda je správný. Naopak pro přímý kód musíme pracovat zvlášť se znaménkem a zvlášť s absolutní hodnotou a může se stát, že místo sčítání budeme muset realizovat odčítání. Budeme se snažit počítat v šestnáctkové soustavě:

$$21_{16} + 95_{16} = B6_{16}$$

Výsledek je pro nezáporná čísla správně a pro čísla v doplňkovém kódu také, protože sčítanci mají opačná znaménka. Můžeme si to ukázat i ve dvojkové soustavě:

$$00100001_2 + 10010101_2 = 10110110_2$$

tedy vlastně sčítáme:

$$+00100001_2 + (-01101011_2) = 21_{16} - 6B_{16} = -4A_{16}$$

$$D(-4A) = B6$$

Pro případ c) a stejná čísla musíme realizovat odčítání, protože 95 v přímém kódu je -15:

$$21_{16} - 15_{16} = 0C_{16}$$

dvojkově (ale na 7 bitech):

$$0100001_2 - 0010101_2 = 0001100_2$$

Podobně pro operandy E5 a AF:

$$E5_{16} + AF_{16} = (1)94_{16}$$

$$11100101_2 + 10101111_2 = (1)10010100_2$$

Vyšel přenos, takže správný výsledek se pro nezáporná čísla nevejde do 8 bitů. Správný výsledek pro nezáporná čísla získáme jen když budeme počítat i s přenosem, tedy zobrazením na 9 bitů. Naopak pro doplňkový kód vyšel správný výsledek, protože sčítáme obrazy záporných čísel a výsledek je také záporný (nebo jinak, nastává přenos mezi předposledním a posledním řádem i z nejvyššího řádu):

$$D(-1B) + D(-51) = D(-6C)$$

C5 + 99	Výsledek z ALU	Interpr. výsledku	C/Over	op. C5	op. 99
nezáporná čísla	(1)5E	15E	1	C5	99
doplňkový kód	5E	5E	1	–3B	–67
přímý kód	DE	–5E	0	–45	–19

Tabulka 4.5.: Výsledky součtů C5+99h.

AC + 69	Výsledek z ALU	Interpr. výsledku	C/Over	op. AC	op. 69
nezáporná čísla	(1)15	115	1	AC	69
doplňkový kód	15	15	0	–54	+69
přímý kód	3D	+3D	0	–2C	+69

Tabulka 4.6.: Výsledky součtů AC+69h.

Pro případ c) sčítáme dvě záporná čísla:

$$-65_{16} + -2F_{16} = -94_{16}$$

dvojkově:

$$-1100101_2 + -0101111_2 = -(1)0010100_2$$

Ale pozor! Máme jen 7bitovou sčítačku, takže výsledek se nevejde do 7 bitů (8 včetně znaménka)! Závěr je, že došlo k přeplnění a výsledek je pro zadaná čísla E5 a AF správný jen pro případ b).

Poznámka Dále bude názorně v tabulkách 4.5, 4.6 a 4.7 uvedeno řešení pro různé hodnoty sčítanců (všechna čísla jsou v šestnáctkové soustavě).

11. 32bitové číslo Y je obrazem čísla X v pohyblivé řádové čárce ve formátu IEEE, tzn. 1. bit zleva znaménko, dalších 8 bitů exponent v aditivním kódu s aditivní konstantou 127, dalších 23 bitů je použito pro zobrazení absolutní hodnoty mantisy s modulem 2, tj. absolutní hodnota mantisy $|m| < 2$ (jeden řád před řádovou čárkou). Princip skryté jedničky je použit. Řádová čárka pro exponent i mantisu je mezi 9. a 10. bitem zleva (počítáno od prvního bitu). Určete o jaké číslo jde, jestliže je v paměti šestnáctkově zapsáno C3C20000.

Řešení
Podle normy IEEE (viz obrázek 4.10) pro 32bitové zobrazení je na 4 slabikách od adresy 3000h zapsáno v šestnáctkové soustavě:

$$C3C20000$$

7B + 6F	Výsledek z ALU	Interpr. výsledku	C/Over	op. 7B	op. 6F
nezáporná čísla	EA	EA	0	7B	6F
doplňkový kód	EA	–16	1	+7B	+6F
přímý kód	(1)6A	–	1	+7B	+6F

Tabulka 4.7.: Výsledky součtů 7B+6Fh.

Tedy dvojkově:

$$1100\ 0011\ 1100\ 0010\ 0000\ 0000\ 0000\ 0000$$

První bit je znaménko (v našem případě záporné), dalších 8 bitů je exponent v aditivním kódu s aditivní konstantou 127:

$$g = 10000111_2.....(128 + 7 - 127)_{10} = 8_{10}$$

Nesmíme zapomenout na skrytou jedničku, tedy hledané číslo je:

$$-1,100001_2 \times (2^8)_{10} = -110000100_2 = -(256 + 128 + 4)_{10} = -388_{10}$$

12. 32bitové číslo Y je obrazem čísla X v pohyblivé řádové čárce ve formátu IEEE jako v předchozím příkladu. Zobrazte v tomto formátu číslo $X = 108_{10}$.

Řešení
Nejprve si převedeme zadané číslo do dvojkové soustavy a posuneme ho maximálně vlevo, tak aby před řádovou čárkou byla jednička (tu potom nezobrazujeme):

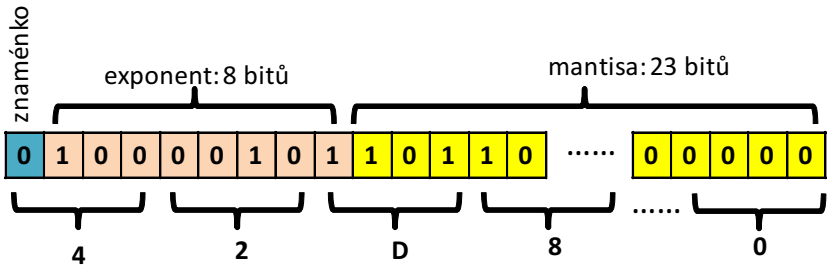
$$108_{10} = (64 + 32 + 8 + 4)_{10} = (1101100)_2 = 1,1011_2 \times 2^6_{10}$$

Zobrazovaná mantisa na 23 bitech se skrytou jedničkou podle 4.10:

$$f = 1011.....00$$

$$g = (128 + 5)_{10} = 10000101_2$$

Schematicky znázorněné řešení je na obrázku 4.24, tedy $42D80000$.



Obrázek 4.24.: Řádová mřížka pro příklad 12.

13. 16bitové číslo Y je obrazem čísla X v pohyblivé řádové čárce. Prvních 12 bitů (zleva) obrazu Y je obrazem $D(M)$ mantisy M v doplňkovém kódu; modul řádové mřížky pro mantisu je roven 2, tzn.: $-1 \leq M < 1$. Zbývající 4 bity jsou rovny exponentu e zvýšenému o 8 (aditivní kód), $A(e) = e + 8$. Princip skryté jedničky není použit. Určete hodnotu čísla X, je-li $Y = 0C0A$ (šestnáctkově). Je obraz Y v normalizovaném tvaru? Pokud ne, najděte obraz čísla X v normalizovaném tvaru.

Řešení
Struktura řádové mřížky podle zadání je na obrázku 4.25. Tedy zobrazené číslo je:

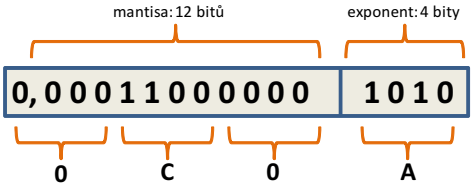
$$0,00011 \times 2^{10-8} = 0,011_2 = 0,375_{10}$$

Normalizovaný tvar, tedy maximální posuv doleva je:

$$0,11 \times 2^{-1}$$

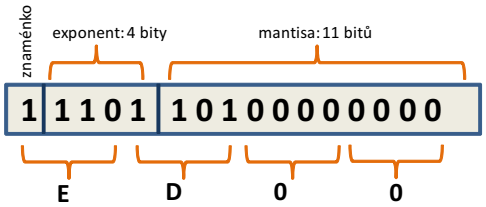
Zobrazení v požadovaném formátu s vyjádřením v šestnáctkové soustavě je následující:

$$0110000000000111_2 = 6007_{16}$$



Obrázek 4.25.: Řádová mřížka pro příklad 13.

14. 16bitové číslo Y je obrazem čísla X v pohyblivé řádové čárce. První bit (zleva) je znaménko mantisy, následuje 4 bitový exponent zvětšený o 7 (aditivní kód). Zbývajících 11 bitů je použito k uložení absolutní hodnoty mantisy při využití principu skryté jedničky. Modul řádové mřížky pro absolutní hodnotu mantisy je roven 2, tzn. $-2 < M < 2$. Určete hodnotu čísla X, je-li $Y = ED00$ (šestnáctkově). K číslu X přičtěte $-10000010,1$ (dvojkově) a výsledný součet uložte ve stejném formátu jako číslo X.



Obrázek 4.26.: Řádová mřížka pro příklad 14.

Řešení

Struktura řádové mřížky podle zadání je na obrázku 4.26. Tedy zobrazené číslo je:

$$-1,101 \times 2^{13-7} = -1101000_2 = -104_{10}$$

Přičtení zadané konstanty:

```
-1000 0010,1
-0110 1000
------------
-1110 1010,1
```

Zobrazení v požadovaném formátu s vyjádřením v šestnáctkové soustavě je následující:

$$F6A8_{16}$$

.