VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Implementace aritmetických operací a konverze čísel mezi číselnými soustavami v softwarovém prostředí Matlab

Implementation of Arithmetic Operations And Conversion Between Number Systems in the Matlab Software Environment

Michael Foltyn

Bakalářská práce

Vedoucí práce: Ing. Marcel Fajkus, Ph.D.

Ostrava, 2021

Abstrakt

Tato bakalářská práce se zabývá reprezentací čísel a aritmetickými operacemi v číslicových systémech. Cílem bylo vytvořit programy pro převod do formátů, které se používají pro vyjádření reálných čísel ve výpočetní technice, ale také programy, které by uměly základní aritmetické operace v daném formátu. Další skupinou jsou skripty, které slouží ke kódování znaků pro reprezentaci a zpracování textů ve výpočetní technice. Teoretická část práce se tak skládá ze základního popisu jednotlivých formátů, ukázky převodů a aritmetických operací. Skripty byly implementovány ve výpočetním prostředí Matlab. Díky této technologie bylo možné jednotlivé aplikace nasadit na MatlabServer Katedry telekomunikační techniky, díky kterému je možné k těmto aplikacím přistupovat prostřednictvím internetu a využívat je přes webové rozhraní. Potenciální využití jednotlivých programů vidím ve výuce. Součástí bakalářské práce jsou také podpůrné materiály ke studiu předmětu Základy číslicových systémů (ZDS).

Klíčová slova

Soustavy; Převody; Posunutí; Pevná řádová čárka; Plovoucí řádová čárka; Aritmetické operace; kódování znaků; Matlab

Abstract

This bachelor thesis focuses on the representation of numbers and arithmetic operations in digital systems. The aim was to create programs for conversion to formats that are used to express real numbers in computer technologies, but also programs that would be able to perform basic arithmetic operations in a given format. Another group are scripts that are used to encode characters for the representation and processing of texts in computer technology.The theoretical part of the work consists of a basic description of individual formats, examples of conversions and arithmetic operations. The scripts were implemented in the Matlab computing environment. Thanks to this technology, it was possible to deploy individual applications on the MatlabServer of the Department of Telecommunications, thanks to that it is possible to access applications via the Internet and use them via the web interface. I see the potential use of individual programs in teaching. The bachelor's thesis also includes supporting materials for the study of the subject Basics of Numeric Systems.

Keywords

Systems; Conversions; Bias; Fixed Point; Floating Point, Arithmetical Operations; Characters Coding; Matlab

Poděkování

Rád bych na tomto místě poděkoval vedoucímu této bakalářské práce Ing. Marcelu Fajkusovi, za odborné vedení mé práce, cenné poznámky a pravidelné konzultace.

Obsah

Seznam použitých zkratek a symbolů

B	–	Byte
b	–	Bit
BE	–	Big Endian
FP	–	Floating Point
FX	–	Fixed Point
IN	–	Integer
LE	–	Little Endian
LSB	–	Least Significant Bit
MSB	–	Most Significant Bit
UTF	–	Unicode Transformation Format
ULP	–	Unit in the last place
VPN	–	Virtual Private Network

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Tato bakalářská práce je zaměřena na číslicové systémy. Přesněji na formáty číslicových systémů, které se ve výpočetní technice používají pro vyjádření reálných čísel a na základní aritmetické operace v těchto formátech. Číslicový neboli digitální systém je zařízení, které pracuje s informacemi v diskrétní podobě. Jsou složeny z obvodů, které zpracovávají binární číslice 0 a 1.

Moderní číslicové systémy byly vynalezeny v Evropě v 17. stolení Gottfriedem Wilhelmem Leibnizem. V dokumentu, který popisoval číslicové systémy popsal sčítání, odčítání, násobení a dělení v binární soustavě. Tohoto vynálezu se chytli matematici a vědci jako je George Boole, který vynalezl Boolean algebru a Claude Shannon, který implementoval Booleovu algebru a binární aritmetiku použitím elektronických relé a přepínačů. V roce 1937 pak George Rober Stibits vyvinul první počítač zvaný Model K, který uměl sčítat binární čísla. Mezi lety 1935 a 1938 nakonec vznikl počítač zvaný Z1, který využíval Bolean algebru a binární čísla v plovoucí řádové čárce.

Teoretická část bakalářské práce se zabývá formáty, které se využívají, nebo využívaly ve výpočetní technice pro vyjádření kladných, záporných a desetinných čísel. Velkou část práce tvoří popisy a ukázky převodů do jednotlivých formátů a popis aritmetických operací. Popisovány jsou převážně operace sčítání a odčítání.

Hlavní účelem této bakalářské práce je výstup praktické části. Mým úkolem bylo vytvořit skripty, které slouží k převodu do formátů, které se používají pro zobrazení reálných čísel. K těmto formátům pak bylo potřeba implementovat programy, které umí základní aritmetické operace v jednotlivých formátech. Dalším výstupem je pak studijní materiál, kde je podrobný popis řešení převodů a aritmetických operací. Pro vývoj skriptů jsem použil prostředí Matlab, které se využívá pro zpracování dat. K lepšímu a lehčímu využití jsem se rozhodnul jednotlivé aplikace nasadit na MatlabServer Katedry telekomunikační techniky, díky kterému je možné k aplikacím přistupovat prostřednictvím webového rozhraní. Výstup mé bakalářské práce by mohl být použit pro studijní účely. Součástí bakalářské práce jsou materiály ke studiu k předmětu Základy číslicových systémů (ZDS).

Kapitola 2

Soustavy

Existuje spousta způsobů, jak se dá pracovat s čísly. Nejpoužívanější způsob je práce s dekadickými hodnotami. S dekadickou soustavou pracujeme celý život, používáme ji každý den, například k nakupování, měření času, či počítání. Ale existuje více soustav, se kterými si dá pracovat. V rámci této bakalářské práce se budu věnovat decimální, binární a hexadecimální soustavě, protože to jsou soustavy využívané v počítačích a číslicových systémech. [1]

2.1 Endianita

Endianita popisuje, jakým způsobem počítače ukládají čísla do paměti. Čísla, které mají větší hodnotu, než jeden byte musí být rozdělena do menších skupin, označených jako základní jednotky (Atomic Element). Jednou z takových jednotek je například byte, který je nejpoužívanější, ale může to být i slovo, či jiná n-tice. Endianita pak určuje, které hodnota bude uložena v paměti na nejnižší adrese. Základní principy ukládání, kterými se budu zabývat je Little Endian, který využívá ukládání „Least to most significant“. Takže první přenášený bude LSB (least significant bit), tedy nejméně podstatný bit a bude na nejnižší adrese. Druhým způsobem je pak Big Endian. Ten naopak využívá metodu „Most to least significant“, takže první přenášený bude MSB (Most significant bit), tedy nejvíce podstatný bit, a tak bude na nejnižší adrese. [2]

Tabulka 2.1: Ukázka ukládání Little a Big endian

Adresy	a	a+1	a+2	a+3	a+4	a+5	a+6	a+7
Little Endian (byte)	90	78	56	34	12	EF	CD	AB
Big Endian (byte)	AB	CD	EF	12	34	56	78	90
Little Endian (word)	CDAB	12EF	5634	9078				
Big Endian (word)	ABCD	EF12	3456	7890				

v tabulce 2.1 můžeme vidět, jak by se ukládalo 64bitové číslo 0xABCDEF1234567890. Pro tento příklad jsem zvolil jako základní jednotky, byte (8 bitů) a word (16 bitů) a to pro Little Endian i Big Endian. Pokud to z textu nebude vyplývat jinak, tak v mé bakalářské práci budu pracovat se způsobem ukládání Big Endian.

2.2 Značení

Na začátek je nutno podotknout, že některá čísla, která vypadají stejně mají v různých soustavách, různou hodnotu.

Jak můžeme vidět v tabulce 2.2, tak když čísla převedeme do jedné soustavy, tak jsou výsledky naprosto odlišné. Proto je nutné používat značení, abychom věděli, s jakou soustavou aktuálně pracujeme.

Tabulka 2.2: Hodnota stejných čísel v různých soustavách

Soustava	Původní hodnota	Decimální hodnota
Decimální	101	101
Binární	101	5
Hexadecimální	101	257

Když počítáme v desítkové soustavě, tak buď nepíšeme nic, písmeno D, nebo číslo 10 jako dolní index.

$$101 = (101)_{10} = (101)_D$$

Pro binární soustavu používáme písmeno B, nebo číslo 2 jako dolní index.

$$(101)_2 = (101)_B = (5)_D$$

Pro hexadecimální soustavu používáme písmeno H, nebo číslo 16 jako dolní index. Alternativou je ještě prefix "0x", avšak nemůžeme psát prefix i dolní index naráz.

$$(101)_{16} = (101)_H = 0x101 = (257)_D$$

v rámci mé bakalářské práce se budu držet rozšířenější konvence, a to značení písmeny.

2.3 Decimální soustava

Decimální soustava, nebo také dekadická, či desítková soustava je soustava, jejíž základem je číslo 10. Čísla se v této soustavě skládají z desíti znaků 0 až 9. Oproti jiným soustavám má výhodu v tom, že nepotřebuje žádné speciální úpravy pro záporná a reálná čísla. Pro záporná čísla použijeme jednoduše znak zvaný mínus „-“ a pro reálná čísla pak desetinnou tečku, nebo desetinnou čárku. Čísla můžeme rozepsat do polynomu a jednotlivé znaky násobit desítkou s určitou váhou a tím „rozebrat“ číslo na jednotlivé znaky. Při sestavování polynomu násobíme číslo základem soustavy umocněným vahou. [3]

$$1535,23 = (1 \times 10^3) + (5 \times 10^2) + (3 \times 10^1) + (5 \times 10^0) + (2 \times 10^{-1}) + (3 \times 10^{-2}) \tag{2.1}$$

v rovnici 2.1 můžeme vidět příklad polynomu pro číslo 1535,23. Když se podíváme na první závorku, tak číslo 1 značí číslici (koeficient) našeho čísla. Číslo 10 značí základ soustavy a umocnění na 3. značí řád (exponent), který vyjadřuje pozici koeficientu. Obecně můžeme říct, že první číslo zleva bude mít exponent $n - 1$, kde n udává počet celočíselných míst. Při každém dalším koeficientu, se exponent sníží o jedničku.

2.3.1 Využití

Desítková soustava je nejpoužívanější, proto se s ní setkáváme prakticky všude. Používáme ji ve výpočtech, při nákupu, při měření na přístrojích. Možnost využití je prakticky neomezená, takže je pro lidi intuitivní s ní pracovat.

2.3.2 Aritmetické operace

Základní operace, jako je sčítání, odčítání, násobení a dělení jsou všeobecně známé. Jednoduché počty se učíme už v dětství a ve škole se pak naučíme operace, jako je násobení, dělení, sčítání a odčítání. Samozřejmě je spousta dalších operací, které se s čísly dají dělat. Jako je umocňování, absolutní hodnota a podobně.

2.4 Binární soustava

Binární soustava, nebo také dvojková soustava, je soustava jejíž základem je číslo 2. K vyjádření čísel používá dvě hodnoty. Nulu a jedničku. Pokud chceme říct, že číslo, o kterém je řeč je binární, tak ho označíme písmenem B. Čísla psaná v binární soustavě se stejně jako čísla v desítkové soustavě dají zapsat do polynomu. Což pak můžeme využít k převodu na decimální soustavu.Polynom vytvoříme tak, že jednotlivá čísla vynásobíme základem soustavy umocněným váhou. [4]

$$(1110.0111)_B = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4}$$

2.4.1 Zápis binárních čísel

Protože nosičem informace je jednička, tak v případě, že na první pozici zleva máme nulu, tak jí můžeme jednoduše odstranit. Tento krok můžeme opakovat tak dlouho, dokud první znak zleva bude jednička. Samozřejmě v případě, že potřebujeme, tak si ty nuly na začátek můžeme dopsat. Ale zase je musíme psát zleva. Kdybychom je napsali doprava, tak změníme hodnotu.

$$(0011)_B = (11)_B$$

2.4.2 Zobrazení reálných čísel

Když pracujeme s binární soustavou, tak se často bavíme o kladných (neznaménkových) celých číslech. Avšak v případě, že chceme dostat například záporné číslo, tak jsou způsoby, jak toho docílit. A to například pomocí doplňku, přímým kódem, nebo kódem s posunutou nulou. O způsobech, jak interpretovat reálná čísla se budu zabývat dále v této práci.

2.4.3 Využití

Binární soustava se používá prakticky ve všech digitálních zařízeních, například v telefonech, počítačích, ale i v digitálních měřících přístrojích. Výhodou binární soustavy je, že pomocí jejich dvou znaků, tedy 1/0 můžeme jednoduše vyjádřit stavy vypnuto a zapnuto, nebo také pravda (true) a lež (false). [4]

2.4.4 Aritmetické operace

Stejně jako v desítkové soustavě můžeme v binární soustavě používat aritmetické operace jako je sčítání, odčítání násobení a dělení.

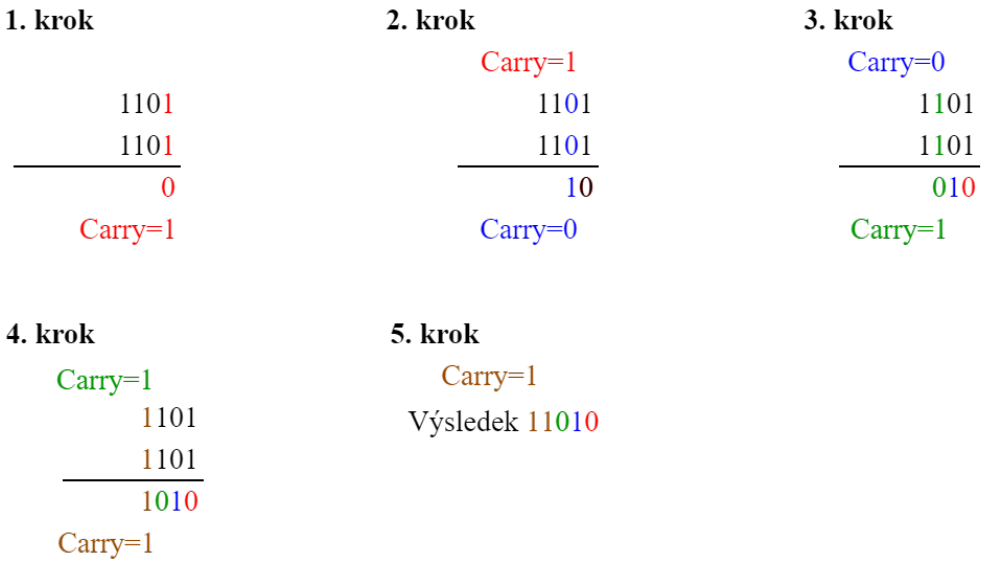
2.4.4.1 Sčítání

K operaci sčítání dvou čísel nám pomůže jednoduchá tabulka 2.3. První 3 sloupce této tabulky jsou stejné, jako pravdivostní tabulka operace XOR, takže můžeme říct, že sčítání probíhá logickou operací XOR. Důležité je začít sčítat od nejnižší hodnoty, to znamená, že sčítáme stejně jako v desítkové, a to zprava doleva.

Tabulka 2.3: Výsledky sčítání s přenosem do vyššího řád

Číslo 1	Číslo 2	Výsledek	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Poslední sloupec v tabulce 2.3 je velmi podstatný pro sčítání, říká nám totiž, že hodnota součtu přešla do vyššího řádu, protože $1 + 1 = 2$, ale binární soustava nezná dvojku, tuto hodnotu pak použijeme v dalším sčítaném znaku. V případě, že nastane situace, kdy první i druhé číslo při součtu je 1 a z minulého kroku se nám také přenesla jednička, tak se řídíme pravidlem, které platí i pro součet více než dvou čísel. Když máme sudý počet jedniček, tak je výsledné číslo 0, naopak, když máme lichý počet jedniček, tak je výsledné číslo 1.



Obrázek 2.1: Postup při sčítání v binární soustavě

Hodnota Carry nám označuje přenos do vyššího řádu, tedy situaci, kdy jsme při součtu dostali číslo vyšší než 1. Carry zapsané nad součtem, je přenos z minulého kroku, tento přenos pak přičítáme k aktuálnímu součtu. Carry zapsané pod součtem je přenos z aktuálního kroku. V případě, že už jsme sečetli všechna čísla, ale zbývá nám carry, tak ho zapíšeme na začátek výsledku. Výpočet krok po kroku můžeme vidět na obrázku 2.1.

2.4.4.2 Odčítání

Pro operaci odčítání se nejčastěji používá dvojkový doplněk. Tomu se budu věnovat později v této práci.

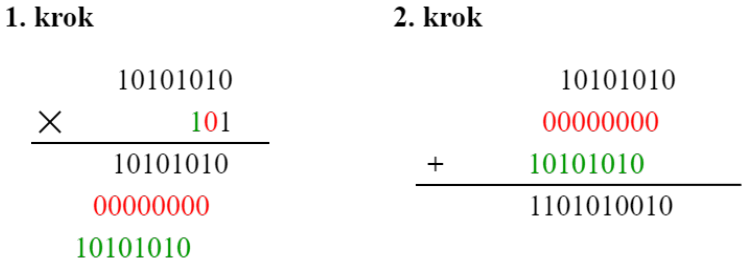
2.4.4.3 Násobení

Násobení stejně jako sčítání probíhá dost podobně, jako v desítkové soustavě. Jediný rozdíl je v tom, že používáme jen číslice 0 a 1, ale stejně jako v normálním násobení, když násobíme nulou, tak dostaneme 0.

Tabulka 2.4: Výsledky operace násobení

Číslo 1	Číslo 2	Výsledek
0	0	0
0	1	0
1	0	0
1	1	1

Jak můžeme z tabulky 2.4 vidět, tak nenulový výsledek dostaneme pouze v případě, že obě čísla jsou jedna. Takže se dá říct, že když máme jedničku, tak násobené číslo zopakujeme, a když máme nulu, tak nám to vyjde nulové. Hlavně nesmíme zapomenout, že když násobíme dalším číslem, tak se u výsledku posuneme o jednu pozici doleva. Jakmile máme číslo vynásobeno všemi čísly druhého čísla, tak provádíme součet.



Obrázek 2.2: Postup při násobení v binární soustavě

Na obrázku 2.2 můžeme vidět postup násobení dvou čísel v binární soustavě. V prvním kroku došlo k roznásobení čísel a v kroku druhém součet a finální výsledek.

2.5 Hexadecimální soustava

Hexadecimální soustava, nebo také šestnáctková soustava je soustava o základu 16. K vyjádření používá 16 znaků. V první řadě používá znaky 0 až 9, pro hodnoty 10 až 15 používá prvních 6 písmen abecedy, tedy písmena A až F. Je nutno podotknout, že v hexadecimální soustavě se začíná nulou, takže první hodnota je 0 a poslední hodnota je 15 ne 16. [5]

Tabulka 2.5: Znaky v hexadecimální soustavě

Hodnota decimálně	Znak hexadecimálně	Hodnota decimálně	Znak hexadecimálně
0	0	8	8
1	1	9	9
2	2	10	A
3	3	11	B
4	4	12	C
5	5	13	D
6	6	14	E
7	7	15	F

2.5.1 Zápis hexadecimálních čísel

Jak můžeme vidět z tabulky 2.5, tak k číselným hodnotám nám přibyly znaky z abecedy. Jelikož hexadecimální soustava není case sensitive, tedy nezáleží, jestli jsou písmena malá, nebo velká, tak nezáleží, jestli napíšeme ff, nebo FF. Obojí má stejný význam. Takže jde čistě o preferenci, jaký zápis, komu vyhovuje.

Stejně jako v případě binární soustavy můžeme umazávat, nebo přidávat nuly zleva, dokud nenarazíme na nenulový znak. Opět platí, že pokud umažeme nulu zprava, tak můžeme drasticky změnit hodnotu našeho čísla.

Příklad zkrácení zápisu hexadecimálního čísla:

$$(00FACE0)_H = (FACE0)_H = 0xFACE0$$

2.5.2 Využití

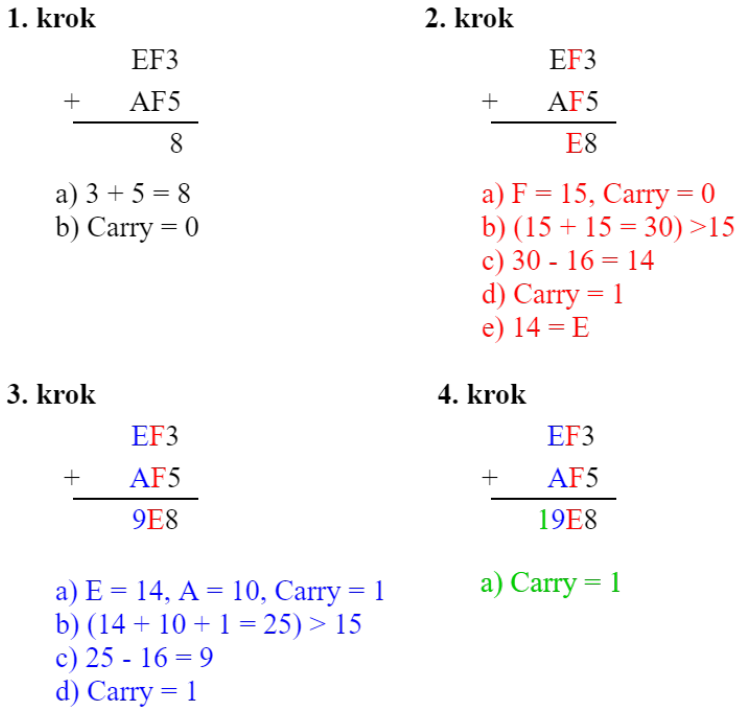
Jelikož hexadecimální zápis dokáže dobře zkrátit zápis binárních hodnot, tak se využívá k definování místa v paměti. Velmi často se také používá, když pracujeme s barvami v grafických editorech, nebo třeba na webových stránkách. v počítačových sítích se používá například v IPv6 adresách anebo v MAC adresách zařízení. [5, 6]

2.5.3 Aritmetické operace

Stejně jako v jiných soustavách můžeme v hexadecimální soustavě využívat aritmetické operace, avšak v tomto případě už je to trochu složitější, protože nemáme jen čísla, ale i znaky. Proto je potřeba znát hodnoty jednotlivých písmen.

2.5.3.1 Součet

Součet v hexadecimální soustavě probíhá podobně jako v desítkové soustavě, je tam však ten rozdíl, že když v součtu překročíme číslo 10, tak nezačínáme znova od nuly, ale pokračujeme až do písmena F, tedy do hodnoty 15. V případě, že překročíme hodnotu 15, tak dochází k přenosu do vyššího řádu a je nutné od čísla odečíst základ soustavy. V tomto případě tedy číslo 16.



Obrázek 2.3: Postup při sčítání v hexadecimální soustavě

Na obrázku 2.3 můžeme vidět detailní postup součtu dvou čísel. V druhém kroku došlo k přenosu do vyššího řádu, a tak jsme museli odečíst od výsledného čísla hodnotu 16, jelikož došlo k přenosu do vyššího řádu, tak jsme do hodnoty Carry napsali číslo 1. Ve čtvrtém kroku už nám zbyla pouze hodnota carry, kterou nemáme k čemu přičíst, tak jí dopíšeme na začátek výsledku.

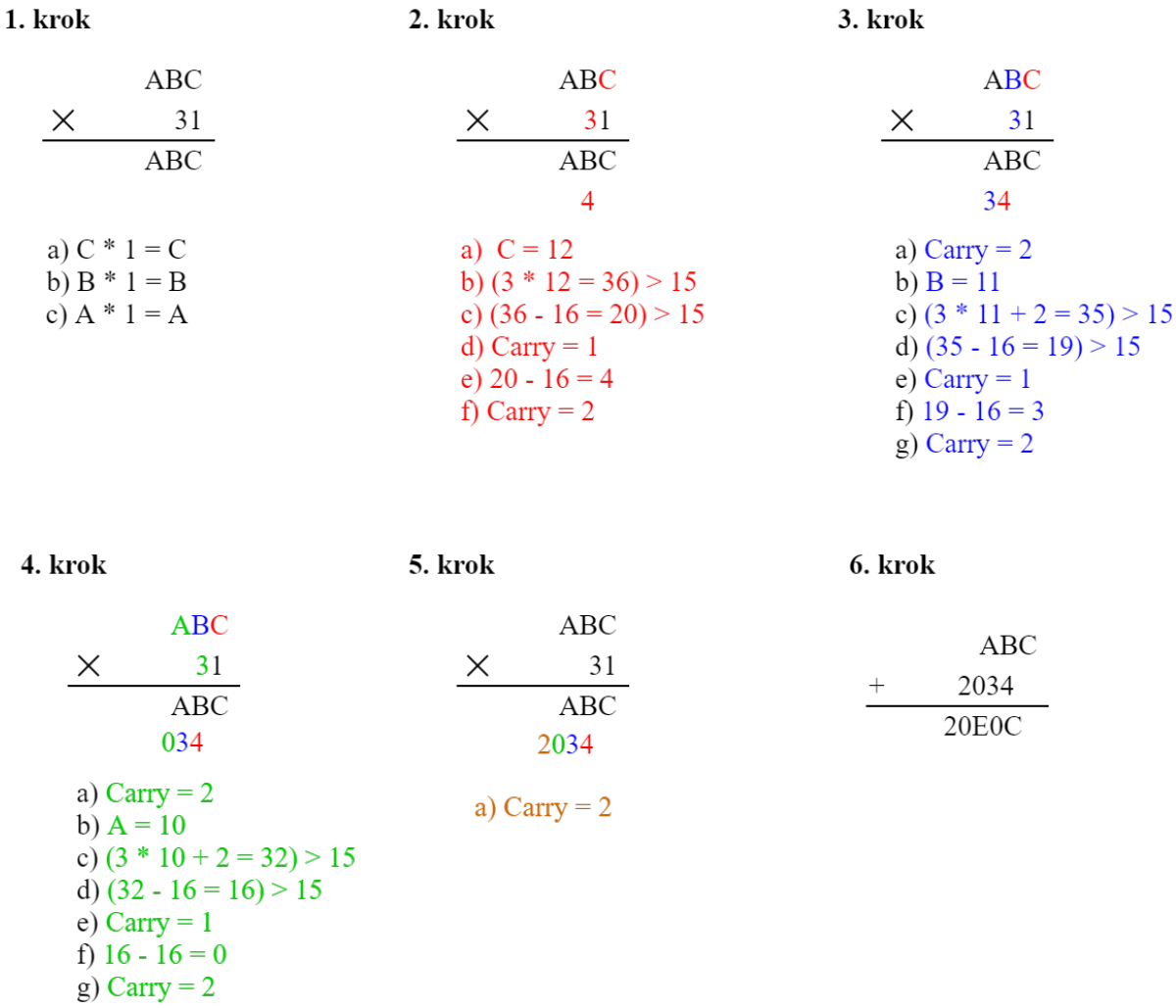
2.5.3.2 Odčítání

Odčítání probíhá obdobně jako sčítání, avšak když se dostaneme pod nulu, tak přičteme hodnotu 16 a v tomto případě se carry bude samozřejmě rovná jedné. Ale je tady jedno velké ale. V případě,

že odčítáme od menšího čísla, číslo větší, tak nám vyjde nesmyslný výsledek, a to z toho důvodu, že hexadecimální soustava bez jakýchkoli úprav nezná záporná čísla. Jak pracovat se zápornými čísly bude v další části mé práce.

2.5.3.3 Násobení

Násobení hexadecimálních čísel už je trošku složitější operace, protože carry může být mnohem vyšší, než 1, protože například F × F, neboli 15 × 15 je rovno 225. Od této hodnoty musíme odečítat číslo 16 tak dlouho, dokud nedostaneme nějakou hodnotu nižší než 16. Po každém jednom odečtení se nám carry zvedne o jedničku.



Obrázek 2.4: Postup při sčítání v hexadecimální soustavě

Jak můžeme z obrázku 2.4 vidět, tak po vynásobení příslušnými znaky je potřeba naše výsledky sečíst do jednoho finálního výsledku.

2.6 Převody mezi soustavami

2.6.1 Převod z decimální do binární soustavy

Převod do binární soustavy probíhá tak, že decimální hodnotu dělíme dvěma se zbytkem. Zbytek si zapíšeme do výsledku a hodnotu po dělení využijeme znova stejným způsobem. Dělení provádíme tak dlouho, dokud nám nezbyde číslo 1. V programovacích jazycích se pro zbytek po celočíselném dělení používá operace modulo, pro kterou se často používá znak „%“. Na obrázku 2.5 můžeme v každém kroku na prvním řádku vidět operaci modulo, na druhém průběžný výsledek a na třetím hodnotu po dělení. V posledním kroku je potřeba celý výsledek otočit, tedy psát ho zleva doprava. Alternativou může být, že si výsledek budeme rovnou psát od konce. Důležitá věc je, že když dělíme číslem 2, tak vždy využíváme zaokrouhlování dolů (floor).

1. krok
133 % 2 = 1
Výsledek = 1
133 / 2 = 66

2. krok
66 % 2 = 0
Výsledek = 10
66 / 2 = 33

3. krok
33 % 2 = 1
Výsledek = 101
33 / 2 = 16

4. krok
16 % 2 = 0
Výsledek = 1010
16 / 2 = 8

5. krok
8 % 2 = 0
Výsledek = 10100
8 / 2 = 4

6. krok
4 % 2 = 0
Výsledek = 101000
4 / 2 = 2

7. krok
2 % 2 = 0
Výsledek = 1010000
2 / 2 = 1

8. krok
1 % 2 = 1
Výsledek = 10100001
1 / 2 = 1

9. krok
Výsledek píšeme od konce
Výsledek = $(10000101)_B$

Obrázek 2.5: Převod z decimální soustavy do binární soustavy

2.6.2 Převod z decimální do hexadecimální soustavy

Převod probíhá prakticky stejně, jako převod do binární soustavy, ale v tomto případě využijeme modulo 16, tedy zbytek po celočíselném dělení číslem 16.

1. krok	2. krok	3. krok	4. krok
268 % 16 = 12	16 % 16 = 0	1 % 16 = 1	Výsledek je třeba otočit
Výsledek = C	Výsledek = C0	Výsledek = C01	Výsledek = $(10C)_H$
268 / 16 = 16	16 / 16 = 1	1 / 16 = 0	

Obrázek 2.6: Převod z decimální soustavy do hexadecimální soustavy

Jak můžeme z obrázku 2.6 vidět, tak je potřeba výsledek otočit, nebo ho rovnou píšeme od konce. Stejně jako u převodu do binární soustavy platí, že když dělíme číslem 16, tak vždy používáme zaokrouhlování floor.

Na závěr je dobré podotknout, že operace modulo můžeme použít pro převod do jakékoliv soustavy, postup bude vždy stejný, jen se změní číslo, podle základu dané soustavy.

2.6.3 Převod z binární do decimální soustavy

Převod probíhá tak, že si binární číslo rozložíme do polynomu. Tedy vezmeme 1 znak a vynásobíme ho základem umocněným váhou. Váha n-tého bitu zprava je n-1.

Převeďte $(10101011)_B$ do decimální soustavy

$(10101011)_B = (1 * 2^7) + (0 * 2^6) + (1 * 2^5) + (0 * 2^4) + (1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (1 * 2^0) =$

$= 128 + 0 + 32 + 0 + 8 + 0 + 2 + 1$

Výsledek = $(171)_D$

Obrázek 2.7: Převod z binární soustavy do decimální soustavy

Jak můžeme z obrázku 2.7 vidět, tak máme 8bitové číslo, takže naše nejvyšší váha bude 8 – 1, tedy 7. Pro výpočet použijeme postupně každý znak zleva a vynásobíme základem soustavy umocněným váhou. Nejvyšší váha je v našem případě na prvním znaku zleva. S každým dalším znakem se pak váha sníží o jedničku. Poté už jen všechna čísla sečteme a máme výsledek v dekadické soustavě.

2.6.4 Převod z binární do hexadecimální soustavy

Pro vytvoření jednoho hexadecimálního čísla potřebujeme celkem 4 bity. To znamená, že si číslo v binární soustavě musíme rozdělit do čtveřic. V našem případě se nejnižší hodnoty binární soustavy nacházejí úplně vpravo, takže je potřeba začít vytvářet čtveřice zprava. V případě, že nedokážeme vytvořit poslední čtveřici, protože nám zbývá méně bitů než 4 tak doplníme nuly zleva. Čtveřice si pak převedeme do dekadické soustavy a přiřadíme znak odpovídající hexadecimální soustavě.

Převeďte binární číslo $(11001011100111)_B$ do hexadecimální soustavy

$(11001011100111)_B = (32E7)_H$
0011 = 3
0010 = 2
1110 = 14 = E
0111 = 7

Obrázek 2.8: Převod z binární soustavy do hexadecimální soustavy

Jak můžeme vidět na obrázku 2.8, tak binární číslo mělo 14 bitů, tak jsme doplnili dvě nuly. Hlavně nesmíme zapomenout na to, jak řadit hexadecimální znaky za sebe. Začínáme sice zprava, ale řazení hexadecimálních znaků bude stejné, jako binárních znaků. Takže když první čtveřice byla vpravo, tak první hexadecimální znak bude také úplně napravo.

2.6.5 Převod z hexadecimální do decimální soustavy

Postup je prakticky stejný, jako v případě převodu do binární soustavy. Vytvoříme polynom, tedy vezmeme hodnotu hexadecimální soustavy a vynásobíme ho základem soustavy umocněným váhou. Základ soustavy je v tomto případě 16. V případě, že nemáme hodnotu, ale znak, například A, tak ho převedeme na hodnotu podle tabulky 2.5.

Převeďte $(FFA0)_H$ do decimální soustavy

$(FFA0)_H = (15 * 16^3) + (15 * 16^2) + (10 * 16^1) + (0 * 16^0) = 61\,440 + 3\,840 + 160 + 0$

Výsledek = $(65\,440)_D$

Obrázek 2.9: Převod z hexadecimální soustavy do decimální soustavy

2.6.6 Převod z hexadecimální do binární soustavy

Převod do binární soustavy probíhá tak, že si jednotlivé znaky převedeme do dekadické soustavy a z dekadické soustavy pak můžeme využít převod do binární soustavy. Poté, co převedeme všechny znaky, tak výsledky převodů spojíme do jednoho výsledku.

Převeďte $(48F1D)_H$ do binární soustavy

$(48F1D)_H = (01001000111100011101)_B$

$4 = (0100)_B$
$8 = (1000)_B$
$F = (15)_D = (1111)_B$
$1 = (0001)_B$
$D = (13)_D = (1101)_B$

Úprava výsledku $(01001000111100011101)_B$

Obrázek 2.10: Převod z hexadecimální soustavy do binární soustavy

Kapitola 3

Doplňky

Doplňky jsou velmi důležitá součást číslicových systémů. Doteď jsme pracovali pouze s neznaménkovými (unsigned) hodnotami, tedy kladnými čísly. Doplňky jsou však jeden ze způsobů vyjádření záporných čísel v binární soustavě. V této práci pracuji hlavně s jedničkovým a dvojkovým doplňkem v binární soustavě, protože je to nejčastější způsob vyjádření záporných čísel, hlavně tedy dvojkový doplněk. [7]

3.1 Jedničkový doplněk

Jedničkový doplněk, také zvaný inverzní kód, se dá vyjádřit logickou unární operací NOT neboli negací. V programovacích jazycích se pro negaci často používá znak „~“ (tilda). Negace binárního čísla je operace, která vymění jedničky za nuly a naopak. Pro znaménko (sign) se používá MSB. V případě, že MSB je 1, tak je číslo záporné, jinak je kladné. Je však třeba dávat pozor na to s jakým rozsahem pracujeme. Je to dané tím, že můžeme umazávat nuly zleva. V případě, že pracujeme s rozsahem 8 bitů, ale máme číslo, které se dá vyjádřit pouze 4 bity, tak přesto musíme znegovat všech 8 bitů. [9, 10, 7, 8]

Jedničkový doplněk můžeme popsat vzorcem:

$$^1A = \sim A,$$

kde ^1A je označení jedničkového doplňku, A je kladné číslo, pro které je doplněk tvořen a $\sim A$ je negace čísla A.

Rozsah jedničkového doplňku je menší než rozsah unsigned čísel, protože se pro znaménko používá první bit. Rozsah jednotkového doplňku je od $-(2^{n-1}-1)$ do $(2^{n-1}-1)$, kde n je rozsah v bitech. Pro rozsah n = 8 dostaneme dekadický rozsah -127 až 127.

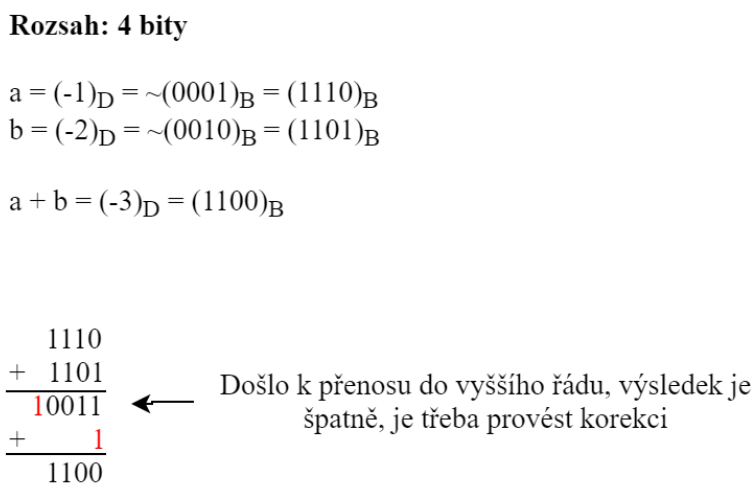
3.1.1 Využití

Jedničkový doplněk se používá jako hradlo NOT v logických obvodech, jinak už v dnešní době moc využití nemá, má totiž pár vad. Jednou z nich je, že má 2 interpretace nuly. Kladnou a zápornou nulu. [9]

Příklad, rozsah 8 bitů:

$$(00000000)_B = +0$$
$$(11111111)_B = -0$$

Dalším problémem je tzv. kruhový přenos. Ten vzniká při operacích sčítání a odčítání. Jde o to, že při práci se zápornými čísly se může stát, že dojde k přenosu jedničky do vyššího řádu. V případě, že se tak stane, je potřeba přičíst k výsledku carry, jinak bude výsledek chybný. Příklad kruhového přenosu můžeme vidět na obrázku 3.1. [11, 8]



Obrázek 3.1: Ukázka kruhového přenosu

3.1.2 Převod z dekadického soustavy do jedničkového doplňku

Záporné číslo převedeme do jedničkového doplňku tak, že ho převedeme do binární soustavy a znegujeme. Kladná čísla se do jedničkového doplňku nepřevádí, takže je převedeme klasickým způsobem do binární soustavy.

Rozsah: 8 bitů

$(113)_D = (01110001)_B$
$(-113)_D = \sim(01110001)_B = (10001110)_B$

Obrázek 3.2: Převod na jedničkový doplněk

3.1.3 Převod z jedničkového doplňku do dekadické soustavy

V případě, že je MSB nula, tak binární hodnotu převedeme klasickým způsobem do dekadické soustavy. V případě, že je MSB jednička, tak binární číslo znegujeme a poté převedeme do dekadické soustavy.

Rozsah: 8 bitů

1. Příklad - Záporné číslo
$(11010010)_B$
MSB = 1
$\sim(11010010)_B = (00101101)_B = -45$

2. Příklad - Kladné číslo
$(110101)_B = (00110101)_B$
MSB = 0
$(00110101)_B = 53$

Obrázek 3.3: Převod z jedničkového doplňku

Jak můžeme vidět na obrázku 3.3, v druhém příkladu bylo potřeba doplnit nuly, aby byl počet bitů stejný jako rozsah v bitech. Kdybychom tak neudělali, dostali bychom špatný výsledek.

3.1.4 Aritmetické operace

Příklad součtu dvou čísel můžeme vidět na obrázku 3.1. Odčítání probíhá prakticky stejně, jen s tím, že druhé číslo převedeme do jedničkového doplňku a pak uděláme součet. Jedničkový doplněk děláme i v případě, že číslo už je v jedničkovém doplňku kvůli tomu, že je záporné. To znamená, že když máme číslo, kterým odčítáme záporné, tak při převodu z decimální soustavy, není potřeba dělat jedničkový doplněk, protože, když uděláme doplněk dvakrát, tak se dostaneme na stejnou hodnotu, v jaké jsme začali.

3.2 Dvojkový doplněk

Dvojkový doplněk je nejpoužívanější způsob zobrazení záporných čísel v počítačích. Na rozdíl od jedničkového doplňku má pouze jednu interpretaci nuly. Také zde platí, že MSB je sign bit. Stále je však potřeba dávat pozor na rozsah v bitech. Další výhodou dvojkového doplňku je to, že při sčítání a odčítání není třeba dělat žádné korekce. [12, 7, 8]

Dvojkový doplněk můžeme popsat vzorcem:

$$^2A = {}^1A + 1 = \sim A + 1,$$

kde ^1A je označení jedničkového doplňku, ^2A je označení dvojkového doplňku, A je kladné číslo, pro které je doplněk tvořen a $\sim A$ je negace čísla A.

Rozsah dekadických čísel ve dvojkovém doplňku je od $-(2^{n-1} - 1)$ do (2^{n-1}), kde n je rozsah v bitech. Pro rozsah $n = 8$, bude dekadický rozsah od -128 do 127. Pro vytvoření dvojkového doplňku použijeme negaci, a k výsledku této operace přičteme jedničku. [8]

3.2.1 Využití

Dvojkový doplněk se používá ve všech počítačových architekturách a programovacích jazycích. V rámci této bakalářské práce se o něm budu bavit v rámci zobrazení reálných čísel ve formátu s pevnou a plovoucí řádovou čárkou. [12]

3.2.2 Převod z dekadické soustavy do dvojkového doplňku

V případě, že převádíme kladné číslo, tak ho převedeme klasicky do binární soustavy. V případě záporného čísla pak postupujeme tak, že číslo převedeme do binární soustavy, znegujeme a přičteme jedničku.

Rozsah: 8 bitů

$(111)_D = (01101111)_B$
$(-111)_D = \sim(01101111)_B + 1$

10010000
+ 1
10010001

Výsledek = 10010001

Obrázek 3.4: Převod na dvojkový doplněk

3.2.3 Převod z dvojkového doplňku do dekadické soustavy

V případě, že je MSB nula, tak číslo převedeme klasickým způsobem do dekadické soustavy. V případě, že je MSB jednička, tak číslo znegujeme, přičteme jedničku a převedeme do dekadické soustavy.

Rozsah: 8 bitů

1. Příklad - Kladné číslo
$(1110111)_B = (01110111)_B = (119)_D$
MSB = 0

2. Příklad - Záporné číslo
$(11110001)_B = \sim(11110001)_B + 1 = (00001111)_B = (-15)_D$
MSB = 1

00001110
+ 1
00001111

Obrázek 3.5: Převod z dvojkového doplňku

3.2.4 Aritmetické operace

Jednou z dalších výhod dvojkového doplňku je, že sčítání a odčítání probíhá prakticky stejně, jak ho známe u neznaménkových čísel. Akorát v případě, že chceme použít operaci odčítání, tak druhé číslo převedeme do dvojkového doplňku a zase provádíme sčítání.

3.2.5 Problémy doplňků

Nevhodným výběrem rozsahu se může jednoduše stát, že po operaci dvou čísel dostaneme hodnotu, která neodpovídá dekadickému rozsahu. To znamená, že dojde k přetečení a výsledek bude chybný. Ke zjištění chybného výsledku existují příznaky (Flag). Ty jsou důležitou součástí procesoru. Na tomto principu je postavená binární sčítačka, tu můžem vidět na obrázku 3.6. [8]

Příznaky:

- C – Carry je příznak, který nám říká, že došlo k přenosu do vyššího řádu (C = 1). To znamená, že výsledek překročil bitový rozsah.
- Z – Zero je příznak, který se nastaví (z = 1) v případě, že bude nulový.
- N - Negative je příznak, který se nastaví (N = 1) v případě, že MSB = 1 a výsledek je chápán jako dvojkový doplněk. Příznak N je tedy sign bit.
- V - Overflow je příznak, který se nastaví (v = 1) v případě, že došlo k přetečení. To nastane ve chvíli, kdy není možné zobrazit výsledek ve dvojkovém doplňku. Stane se tak proto, že binární sčítačka má omezený počet bitů.

Chyby, které mohou nastat se dají popsat vztahy:
Sčítání: $(+A) + (+B) = -C$ nebo $(-A) + (-B) = +C$
Odčítání: $(+A) - (-B) = -C$ nebo $(-A) - (+B) = +C$

Rozsah: 8 bitů

$a = (96)_D = (01100000)_B$
$b = (48)_D = (00110000)_B$

1. Součet
$a + b = (144)_D$

```
  01100000
+ 00110000
  10010000
```

2. Vyznačení příznaků
C = 0 -> Nedošlo k přenosu
Z = 0 -> Výsledek je nenulový
V = 1 -> Došlo k přetečení
N = 1 -> Výsledek je záporný

3. MSB je 1, převod z dvojkového doplňku

$^2(10010000) = (-112)_D$
$(-112)_D \neq (144)_D$

Obrázek 3.6: Sčítání s vyznačením příznaků (princip fungování binární sčítačky)

Na obrázku 3.6 můžeme vidět sčítání dvou čísel, kde dojde k chybnému výsledku. Proč tomu tak je nám napoví příznaky. C nám říká, že nedošlo k přenosu do vyššího řádu. Z nám říká, že výsledek není nulový. V nám říká, že došlo k přetečení a výsledek se tedy nedá zobrazit ve dvojkovém doplňku, takže se dá říct, že výsledek je chybný. A N nám říká, že výsledek je záporný, což vzhledem k tomu, že jsme sčítali dvě kladná čísla není možné.

Kapitola 4

Kód s posunutou nulou

Kód s posunutou nulou, či anglicky offset binary (má i další názvy), je další ze způsobů, jak zaznamenávat signed čísla. Dekadický rozsah zobrazení je od -b (bias, nebo také offset), neboli posunutí do $2^n - 1 - n$, kde n je rozsah v bitech. Bias může být libovolné číslo, avšak při zobrazování čísel v pohyblivé řádové čárce se používá vztah $b = 2^{n-1} - 1$. Narozdíl od doplňků nepoužívá sign bit. [13]

4.1 Využití

Kód s posunutou nulou se používá ke zpracování záporných čísel v A/D (analog to digital) a D/A (digital to analog) převodnících, také ve formátu pohyblivé řádové čárky pro zobrazení exponentu. Často se také používá ve zpracování digitálních signálů. [13]

4.2 Převod

Převod probíhá tak, že k číslu, které chceme přičteme bias, výsledné číslo nesmí vyjít záporné. V případě zpětného převodu bias odečteme. Detailní postup převodu do formátu s posunutou nulou i zpět do dekadické soustavy můžeme vidět na obrázku 4.1.

Pro převod do formátu s posunutou nulou můžeme použít vztah:

$$^{B}A = A + b$$

Pro převod z formátu s posunutou nulou můžeme použít vztah:

$$A = {}^{B}A - b$$

kde ^{B}A je posunuté číslo. Toto číslo musí být přirozené, tedy $^{B}A \geq 0$, A je kladné, nebo záporné celé číslo, pro které posunutí počítáme a b je bias.

Převod do formátu s posunutou nulou

Rozsah: 8 bitů
Bias: 127

1. Příklad - kladné číslo
$A = (56)_D = (00111000)_B$
$^BA = 56 + 127 = (183)_D = (10110111)_B$

2. Příklad - záporné číslo
$A = (-56)_D$
$^BA = -56 + 127 = (71)_D = (01000111)_B$

Převod z formátu s posunutou nulou

Rozsah: 8 bitů
Bias: 127

1. Příklad - kladné číslo
$^BA = (11111101)_B = (253)_D$
$A = 253 - 127 = (126)_D$

2. Příklad - záporné číslo
$^BA = (01111110)_B = (126)_D$
$A = 126 - 127 = (-1)_D$

Obrázek 4.1: Převod do formátu s posunutou nulou a zpět

4.3 Aritmetika s čísly v posunutí

Sčítání a odčítání probíhá tak, že provedeme operaci mezi dvěma čísly a poté je převedeme do kódu s posunutou nulou, tedy přičteme bias. V případě že pracujeme s čísly, která už jsou ve formátu s posunutou nulou, tak v případě sčítání bias odečteme a v případě odčítání bias přičteme. V žádném z těchto případů však nesmí nastat přetečení. [8]

K operaci sčítání můžeme použít vztah:

$$^B Součet = (A + B) + b = {}^BA + {}^BB - b$$

K operaci odčítání můžeme použít vztah:

$$^B Rozdíl = (A - B) + b = {}^BA - {}^BB + b$$

kde A, B jsou kladná, nebo záporná celá čísla, pro která posunutí počítáme, b je bias a ^BA, ^BB jsou přirozená čísla s posunutou nulou.

Rozsah: 8 bitů
b = 127
A = $(36)_D$
${}^BA = 36 + 127 = (163)_D$
B = -43
${}^BB = -43 + 127 = (84)_D$

1. Příklad - součet
1. Způsob
${}^B\text{Součet} = A + B + b = 36 + (-43) + 127 = (120)_D$
2. Způsob
${}^B\text{Součet} = {}^BA + {}^BB - b = 163 + 84 - 127 = (120)_D$

2. Příklad - rozdíl
1. Způsob
${}^B\text{Rozdíl} = A - B + b = 36 - (-43) + 127 = (206)_D$
2. Způsob
${}^B\text{Rozdíl} = {}^BA - {}^BB + b = 163 - 84 + 127 = (206)_D$

Obrázek 4.2: Sčítání a odčítání ve formátu s posunutou nulou

Kapitola 5

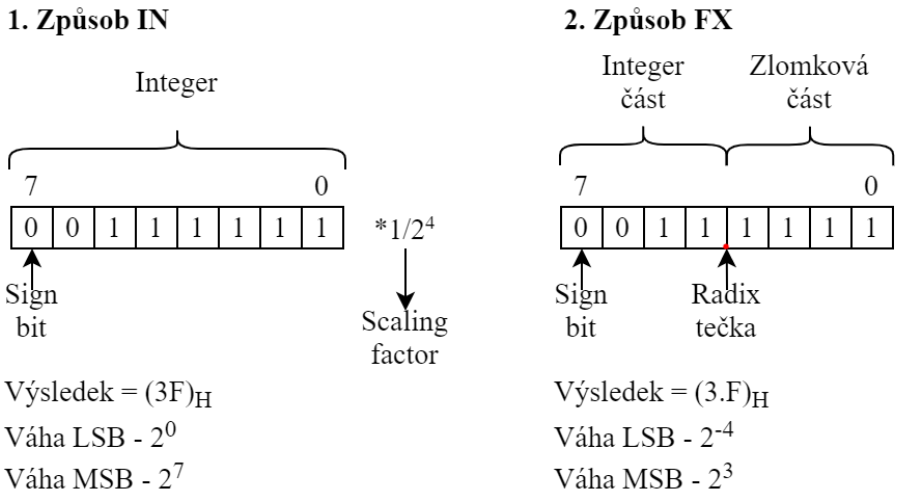
Formát s pevnou řádovou čárkou

Pevná řádová čárka, anglicky Fixed point (FX), jsou čísla s desetinnou tečkou, která je umístěna v předem definované místě. Z matematického hlediska jsou to racionální čísla, která se vyjadřují jako podíl nebo zlomek. Zlomek pak můžeme zapsat jako reálné číslo krát měřítko (scaling factor). Scalling factor může být celé číslo, nebo zlomek ve tvaru $1/jmenovatel$ (měřítko).

Příklady čísel v měřítku:
2,38 v měřítku 100/1 je 238
238 v měřítku 1/100 je 2,38

Měřítko se pak vybírá tak, aby čitatel bylo celé číslo, nejčastěji jsou to mocniny čísla 2 nebo 10. V rámci této práce budu využívat binární měřítko $1/2^n$. Fixed point čísla mohou být znaménková i neznaménková, v rámci této práce se budu věnovat pouze znaménkovými čísly. Pro pevnou řádovou čárku je MSB znaménkový bit. Pro záporná čísla používáme dvojkový doplněk. Protože budeme pracovat s binární soustavou, tak se nehodí používat pojem desetinná tečka, nebo desetinná čárka, proto budu používat pojem radix tečka.

Čísla v pevné řádové čárce můžeme chápat dvěma způsoby. Prvním způsobem je integer (IN) číslo, neboli celé číslo. Druhým způsobem je zobrazování reálného čísla nebo čísla v pevné řádové čárce (FX). V tomto případě je důležitá pozice radix tečky, která nám určuje celou část a desetinnou část. Pozici radix tečky nám v měřítku $1/2^n$ značí hodnota n. V případě, že budeme používat zobrazení FX, tak se změní i váha LSB, to můžeme vidět na obrázku 5.1. Pro zobrazení FX je nalevo od radix tečky váha 0 a napravo od radix tečky je váha -1. [14, 15, 8]



Obrázek 5.1: Způsoby zobrazení čísel v pevné řádové čárce

5.1 Číselný formát $\mathbb{Q}$

V matematice je $\mathbb{Q}$ označení pro racionální čísla. V informatice je to označení fixed point formátu a používají se zápisy jako $\mathbb{Q}$m.f, $\mathbb{Q}$f, $\mathbb{Q}$m.n nebo $\mathbb{Q}$n, kde písmeno m označuje počet bitů celočíselné části a písmeno f nebo n počet bitů zlomkové části. Například formát $\mathbb{Q}$3.12 má 3 bity pro celou část, 12 bitů pro zlomkovou část a MSB je sign bit. Velikost slova je tedy $m + f + 1$, kde $+1$ značí sign bit.

Rozsah fixed point čísel je od $(-2^{m+f})/2^f$ do $+(2^{m+f} - 1)/2^f$. S přesností $\varepsilon = 1/2^f$. Přesnost zobrazení záleží na vhodném výběru f. V případě, že zlomková část má příliš málo bitů, tak může dojít k zaokrouhlení, v případě, že bude příliš vysoké, tak se může hodnota nepatrně zvýšit. [8]

5.2 Využití

Formát fixed point se používá v digitálním zpracování signálů, ta zahrnuje například aplikaci digitálního filtru, digitální zpracování obrazu, převod řeči na text a zpět. Dále se používá pro výpočet komprese JPEG obrázků a obecně v počítačové grafice. [14, 15]

5.3 Převod do formátu $\mathbb{Q}$

Ve formátu $\mathbb{Q}$m.f písmeno m označuje počet bitů celočíselné části a písmeno f počet bitů zlomkové části. Převod pak probíhá tak, že reálné číslo vynásobíme měřítkem, tedy hodnotou 2^f. Výsledek zaokrouhlíme. V případě, že je číslo záporné, tak formát IN převedeme do dvojkového doplňku a poté převedeme do požadovaného formátu (IN nebo FX). Formát IN značí integer, tedy celé číslo a formát FX je formát s radix tečkou. [8]

1. Převod kladného reálného čísla 1,35 do formátu Q2.13

Rozsah: 16 bitů
a)Vyznačení m a f
m = 2, f = 13
b) Vynásobení měřítkem 2^f
$1{,}35 * 2^{13} = 11059{,}2 \approx (11059)_D$
c) Převod do Formátu IN
$(11059)_D = (2B33)_H$
$IN = (2B33)_H$
d) Převod do formátu FX
$(2B33)_H = (0010101100110011)_B$
$FX = (001.0101100110011)_B$
sign bit
$FX = (1.5998)_H$
V případě, že v desetinné části, nemáme počet bitů dělitelný 4, tak si doplníme nuly doprava

Výsledky:
$IN = (2B33)_H$
$FX = (1.5998)_H$

2. Převod záporného reálného čísla -1,35 do formátu Q2.13

Rozsah: 16 bitů
a) Vyznačení m a f
m = 2, f = 13
b)Vynásobení měřítkem 2^f
$-1{,}35 * 2^{13} = -11059{,}2 \approx (-11059)_D$
c) Převod do dvojkového doplňku
$^2(11059)_D = (2^{16} - 11059) = (54447)_D$
rozsah
d) Převod do formátu IN
$(54447)_D = (D4AF)_H$
$IN = (D4AF)_H$
e) Převod do formátu FX
$(D4AF)_H = (1101010010101111)_B$
$FX = (110.1010010101111)$
sign bit
$FX = (6.A578)_H$
V případě, že v desetinné části, nemáme počet bitů dělitelný 4, tak si doplníme nuly doprava

Výsledky:
$IN = (D4AF)_H$
$FX = (6.A578)_H$

Obrázek 5.2: Převod do formátu $\mathbb{Q}$m.f

Jak můžeme vidět na obrázku 5.2, pro záporné číslo je třeba udělat dvojkový doplněk. V tomto případě jsme ho udělali podle vztahu:

$$^2A = 2^n - A$$

kde ^2A je číslo ve dvojkovém doplňku, A je číslo, které převádíme na dvojkový doplněk a n je bitový rozsah.

5.4 Převod z formátu $\mathbb{Q}$

Zjistíme hodnotu MSB. Hexadecimální nebo binární hodnotu převedeme na decimální hodnotu. V případě, že je MSB roven jedné, tak číslo převedeme z dvojkového doplňku a přidáme znaménko mínus. Výsledku pak dosáhneme vynásobením měřítkem $1/2^f$. [8]

1. Převod čísla $(0AF3)_H$ ve formátu Q3.10

Rozsah: 16 bitů

a) Převod do binární soustavy
$(0AF3)_H = (0000101011110011)_B$
b) Zjištění, jestli je číslo kladné, nebo záporné
MSB = 0 -> kladné číslo
c) Převod do dekadické soustavy
$(0AF3)_H = (2803)_D$
d) Vydělení měřítkem 2^f
$(2803)_D$ v měřítku $1/2^{10} = 2,7373046875$

Reálné číslo je 2,7373046875

2. Převod čísla $(DAF3)_H$ ve formátu Q3.10

Rozsah: 16 bitů

a) Převod do binární soustavy
$(DAF3)_H = (1101101011110011)_B$
b) Zjištění, jestli je číslo kladné, nebo záporné
MSB = 1 -> Záporné číslo
c) Převod do dekadické soustavy
$(DAF3)_H = (56051)_D$
d) Převod z dvojkového doplňku
$^2(DAF3)_H = -(2^{16} - 56051) = -9485$
e) Vydělení měřítkem 2^f
$(-9485)_D$ v měřítku $1/2^{10} = -9,2626953125$

Reálné číslo je -9,2626953125

Obrázek 5.3: Převod z formátu $\mathbb{Q}$m.f

5.5 Aritmetika v pevné řádové čárce

Sčítání a odčítání se provádí pomocí celých čísel, které odpovídají číslu v pevné řádové čárce. Bavíme se tedy o formátu Integer (IN). Pro průběh je potřeba, aby obě čísla byly ve stejném formátu.

Toho dosáhneme pomocí vztahu:

$$\mathbb{Q}m1.f \pm \mathbb{Q}m2.f = \mathbb{Q}(max(m1, m2) + 1).f$$

Výsledný formát bude mít o jeden bit v celočíselné části více, než je maximální počet bitů v Integer části obou čísel. V případě záporného čísla nesmíme zapomenout převést na dvojkový doplněk.

1. Příklad
Rozsah: 8 bitů

Q1.3 - (1.25)
Q1.3 - (1.625)

1. Převod do formátu IN
$A = 1.25 * 2^3 = (10)_D$
$A = (10)_D = (0A)_B$

$B = 1.625 * 2^3 = (13)_D$
$(13)_D = (0D)_B$

2. Součet

$$\begin{array}{r} 0A \\ +\ 0D \\ \hline 17 \end{array}$$

Výsledek ve formátu IN je $(17)_H$

3. Kontrola
$(17)_H = (23)_D$
MSB = 0 -> kladné číslo
$23 * 1/2^3 = 2,875$
Výsledný formát Q2.3

2. Příklad
Rozsah: 8 bitů

Q1.3 - (1.25)
Q1.3 - (-1.625)

1. Převod do formátu IN
$A = 1.25 * 2^3 = (10)_D$
$(10)_D = (0A)_H$

$B = -1.625 * 2^3 = (-13)_D$
$^2B = (2^8 - 13) = (243)_D$
$(243)_D = (F3)_H$

2. Součet

$$\begin{array}{r} 0A \\ +\ F3 \\ \hline FD \end{array}$$

Výsledek ve formátu IN je $(FD)_H$

3. Kontrola
$(FD)_H = (253)_D$
MSB = 1 -> záporné číslo
$^2(FD)_H = -(2^8 - 253) = (-3)_D$
$-3 * 1/2^3 = (-0,375)_D$
Výsledný formát Q2.3

Obrázek 5.4: Sčítání ve formátu $\mathbb{Q}$m.f

Odčítání probíhá úplně stejně, s tím rozdílem, že druhé číslo převedeme do dvojkového doplňku. V případě, že zadané číslo je záporné, tak není třeba převádět do dvojkového doplňku, protože když použijeme doplněk na jedno číslo dvakrát, tak ho převedeme do dvojkového doplňku a zase zpět.

Kapitola 6

Formát s plovoucí řádovou čárkou

Formát s pohyblivou řádovou čárkou, anglicky Floating Point (FP) je formát, který se používá pro zobrazování reálných čísel. Reálné číslo se skládá z významných číslic zvaných significand a ze základu neboli base umocněného na řád n. Base je číslo, který definuje základ soustavy. Pro desítkovou soustavu je to číslo 10 a pro binární soustavu je to číslo 2.

Pro FP obecně platí vztah:

$$Significand \times base^n,$$

kde $Significand$ je významná číslice se znaménkem, $base$ je základ číselné soustavy, n je exponent, který musí být celé číslo a $base^n$ tvoří měřítko.

Desítková soustava

$133 * 10^0 = 13{,}3 * 10^1 = 1{,}33 * 10^2 = 0{,}133 * 10^3$
$0{,}133 * 10^0 = 1{,}33 * 10^{-1} = 13{,}3 * 10^{-2} = 133 * 10^{-3}$

Dvojková soustava

$110 * 2^0 = 11.0 * 2^1 = 1.10 * 2^2 = 0.110 * 2^3$
$0.11 * 2^0 = 1.1 * 2^{-1} = 11 * 2^{-2} = 110 * 2^{-3}$

Obrázek 6.1: Příklad čísel s měřítkem v binární a decimální soustavě

Velkou výhodou tohoto formátu je, že můžeme jednoduše popsat velmi velké hodnoty, jako jsou například vzdálenosti ve vesmíru, ale také velmi malé hodnoty jako jsou výrobní parametry procesorů a podobně. FP má velký rozsah a maximální přesnost. V IT (informačních technologiích) je pak třeba zvážit jaký formát vybrat, aby rozsah nebyl zbytečně velký, popřípadě, aby byla zachována co nejvyšší přesnost. [17, 16, 8]

6.1 Způboby zápisu

Vědecký zápis – formát, který obsahuje significand, který je jakékoliv reálné číslo, základ a exponent. Příklad vědeckého zápisu $134.33 * 10^{-3}$, 11010.11×2^6.
Technický zápis – zápis, který odpovídá prefixům SI soustavy. Exponent je tedy násobkem čísla 3. Příklad technického zápisu 75*10^{-3} m (metrů), neboli 75 mm (milimetrů), 13*10^6 b, neboli 13 Mb.
Normalizovaný vědecký zápis – zápis, ve který má v celočíselné části pouze jednu číslici, která není rovná nule. Příklad normalizovaného zápisu 1.133 * 10^{13}, 1.1001 * 2^5. [8]

6.2 Formáty BinaryX

V rámci této bakalářské práce se budu věnovat formáty, které jsou definované standardem IEEE 754-2019. Konkrétně to budou formáty binary16, binary32 v programování známý pod názvem single precision (jednoduchá přesnost), binary64 známý jako double precision (dvojitá přesnost), binary128, protože to jsou formáty, které jsou vhodné pro ukládání reálných čísel v počítači, tedy až na binary16, tento formát je vhodný pro účely, kde není nutná vysoká přesnost. [17, 16, 8]

Tabulka 6.1: Parametry formátů binaryX

Formát	Počet bitů pro significant	Počet bitů pro exponent (n)	Minimální exponent (E min) decimálně	Maximální exponent (E min) decimálně	Bias	šířka pole significandu (w)
binary16	11	5	-14	15	15	10
binary32	24	8	-126	127	127	23
binary64	53	11	-1022	1023	1023	52
binary128	113	15	-16382	16383	16383	112

Jak můžeme z tabulky 6.1 vidět, tak binary16 má velmi malý rozsah. Co se týče počtu bitů significandu, tak je třeba myslet na to, že první bit je sign bit, takže pro samotné číslo máme o bit méně. Avšak první jedničku v significandu můžeme vynechat, takže se dostaneme zpět na stejný počet bitů.

Na obrázku 6.2 můžeme vidět složení jednotlivých formátů.

binary16

Sign - 1 bit	Posunutý exponent (e) - 5 bitů	Significand (w) - 10 bitů

binary32

Sign - 1 bit	Posunutý exponent (e) - 8 bitů	Significand (w) - 23 bitů

binary64

Sign - 1 bit	Posunutý exponent (e) - 11 bitů	Significand (w) - 52 bitů

binary128

Sign - 1 bit	Posunutý exponent (e) - 15 bitů	Significand (w) - 112 bitů

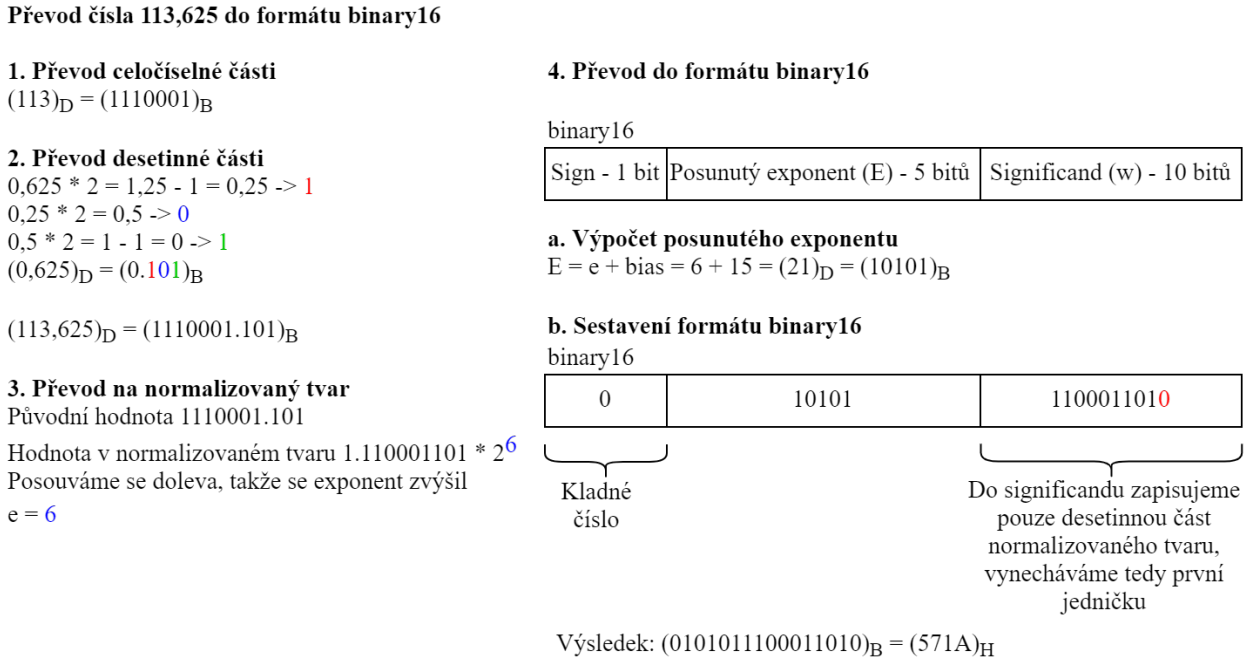
Obrázek 6.2: Grafické znázornění složení formátů binaryX

6.3 Využití

Jak už jsem zmínil, tento formát je vhodný pro zápis velmi velkých, nebo také velmi malých hodnot ve vysoké přesnosti. V technice se používá pro zápis reálných čísel. Můžeme ho také nalézt ve spoustě programovacích jazycích (double, single precision).

6.4 Převod do formátu FP

Po převod je nutné znát formát, do kterého převádíme a samozřejmě jeho parametry. Pak postupujeme tak, že převedeme celočíselnou část a desetinnou část do binární soustavy. Převedeme na normalizovaný stav, kde se zaznamenáme exponent, který použijeme pro výpočet posunutého exponentu a poté už jen podle obrázku 6.2 sestavíme formát. [8]



Obrázek 6.3: Převod do formátu FP

Na obrázku 6.3 můžeme vidět sestavení FP formátu pro reálné číslo. Significand a posunutý exponent mají přesně stanovenou délku v bitech to znamená, že pokud nějaká z těchto hodnot má menší počet bitů, je třeba doplnit nuly. V případě exponentu doplňujeme nuly zleva, v případě significandu zprava, což můžeme vidět na červeně vyznačené nule v significand části. Druhá důležitá věc je, vhodně zvolený formát. V případě, že by reálná, nebo desetinná část měla větší počet bitů, tak by došlo k zaokrouhlení a výsledné číslo by mohlo být mírně odlišné. K zaokrouhlení dochází také v případě, že desetinná část je periodická. V případě, že máme číslo menší než nula, tak je potřeba v significandu odstraňovat nuly zleva tak dlouho, dokud nenarazíme na první jedničku. V případě velmi malých čísel, například 1,3 * 10^{-50} nám v desetinné části vznikne spousta nul, v případě, že bychom je neodstranily, tak se číslo zaokrouhlí, lépe řečeno significand část by byla nulová.

6.5 Převod z formátu Floating point

Zpětný převod funguje tak, že si číslo rozdělíme na sign bit, posunutý exponent a significand. Z posunutého exponentu poté vypočítáme exponent. Poté jsou dvě možnosti, jak převést significand na reálnou hodnotu. První je, že ho převedeme z normalizovaného tvaru a poté převedeme do decimální soustavy. Druhá, že ho převedeme v normalizovaném tvaru a poté vynásobíme měřítkem. K tomu nám poslouží dříve vypočítaný exponent. [8]

Převedení $(D3A0)_H$ na reálné číslo

1. Převedení z Hexadecimální soustavy do binární soustavy
$(D3A0)_H = (1101001110100000)_B$

2. Rozdělení na S | E | significant

1	10100	1110100000

S = 1 -> záporné číslo

3. Výpočet exponentu
bias = 15
E = 20
e = E - bias = 20 - 15 = 5

4. Reálné číslo
Nezapomenout "skrytý" bit

$-1.11101 * 2^5$

a. Převod z normalizovaného tvaru
$-(1.11101)_B = (-1.90625)_D * 2^5$
$-1.90625 * 32 = (-61)_D$

b. Převod bez normalizovaného tvaru
$-1.11101 * 2^5 = -111101 * 2^0$
$-(111101)_B = (-61)_D$

Obrázek 6.4: Převod z formátu Floating point

6.6 Aritmetika ve formátu Floating Point

6.6.1 Zaokrouhlování

Zaokrouhlování slouží k tomu, aby výsledek ve formátu FP byl co nejpřesnější. Pro zaokrouhlování používáme LSB, Round a sticky bit, které nám určují, zda-li máme zaokrouhlit. LSB bit je nejméně podstatný bit significandu. Round bit se nachází na první pozici za significantem. Sticky bit je vždy logický OR zbývajících nejméně významných bitů za Round bitem. Rozložení jednotlivých bitů můžeme vidět na obrázku 6.5. [8]

Significant	Round	Sticky

Obrázek 6.5: Pozice LSB, Round a Sticky bitu

V této práci budu používat pouze zaokrouhlování k nejbližší a sudé, protože to je výchozí způsob zaokrouhlování v FP aritmetice. Avšak existují i další způsoby zaokrouhlování, a to například zaokrouhlení směrem k nule, směrem k plus nekonečnu a směrem k mínus nekonečnu.

V počítačích je zaokrouhlení často prováděno přičtením konstanty k číslu. K tomuto slouží termín ULP (unit in the last place neboli jednotka na posledním místě).

Rozhodnout o zaokrouhlení můžeme pomocí této tabulky:

Tabulka 6.2: Tabulka rozhodující o zaohrouhlování

LSB	R	S	Operace
0	0	0	+ 0
0	0	1	+ 0
0	1	0	+ 0
0	1	1	+ 1/2 ULP
1	0	0	+ 0
1	0	1	+ 0
1	1	1	+ 1/2 ULP

+1/2 ULP znamená, že na pozici p přičteme číslo 1. Pozice p je první pozice za significandem.

6.6.2 Sčítání

Operace sčítání probíhá tak, že si číslo rozdělíme na sign bit, exponent a significand. Significand si pak zobrazíme v normalizovaném tvaru. Tak to uděláme pro obě čísla. V případě, že čísla mají rozdílný exponent, tak je potřeba jedno z čísel převést na stejný exponent. Poté je potřeba rozšířit oba significandy o dva bity zleva. Jeden bit je sign bit a druhý pro případ, že by došlo k přenosu do vyššího řádu. Hlavně nesmíme zapomenout na to, že pokud je číslo záporné, je potřeba ho převést na dvojkový doplněk. Provedeme operaci součtu binárních čísel a výsledek převedeme na normalizovaný tvar. V případě, že je výsledek nulový, tak vyznačíme Guard bit. Jedná se o první bit za significandem a v případě, že je vyznačený je potřeba provést logický posun doleva, abychom dostali normalizovaný tvar. Nakonec vyznačíme significand, LSB, Round a Sticky bity. A rozhodneme o zaokrouhlení. Jakmile máme zaokrouhleno, tak výsledek převedeme zpět do požadovaného formátu binaryX a případně do požadované soustavy. [16, 18]

Sčítání ve formátu binary16

$A = (C310)_H = (1100001100010000)_B$
$B = (5456)_H = (0101010001010110)_B$

1. Rozdělení na S | E | significant

A =

1	10000	1100010000

A - je záporné číslo

B =

0	10101	0001010110

2. Převod na normalizovaný tvar
bias = 15

A - E = e - bias = 16 - 15 = 1
$1.1100010000 * 2^1$

B - E = e - bias = 21 - 15 = 6
$1.0001010110 * 2^6$

3. Sjednocení exponentů
Posuneme A o 5 pozic doprava (e + 5)
$A = 0.00001110001 * 2^6$

4. Doplníme 2 vedoucí bity
A = 000.00001110001
B = 001.0001010110

5. Převedení záporných čísel na dvojkový doplněk
$^2A = 111.11110001111$

6. Součet

```
  111.11110001111
+ 001.0001010110
  001.00000111011
```

7. Převod na normalizovaný tvar
$1.00000111011 * 2^6$

8. Vyznačení LSB, R a S bitů
$1.00000111011 * 2^6$
LSB = 1
R = 1
S = 0
Výsledek je třeba zaokrouhlit
+1/2 ulp

```
  1.00000111011
+              1
  1.00000111100
```

9. Převedení výsledku do binary16
$1.00000111100 * 2^6$
E = e + bias = 21

0	10101	0000011110

Výsledek: $(541E)_H$

Obrázek 6.6: Operace sčítání reálných čísel ve formátu FP

6.6.3 Odčítání

Operace odčítání je identická se sčítáním. Stačí znegovat sign bit u druhého čísla a pak je postup úplně stejný.

Kapitola 7

Unicode Transformation Format

Unicode je technická norma definující jednotnou znakovou sadu pro reprezentaci a zpracování textů. Nejnovější verze Unicode obsahuje více než 142 000 znaků a momentálně pokrývá 154 moderních a historických písem, ale také mnoho sad symbolů a smajlíků. UTF je pak způsob kódování jakým ukládáme řetězce složené z Unicode znaků do paměti počítače. Momentálně jsou 3 formáty a to UTF-8, UTF-16 a UTF-32. [19, 20, 21]

7.1 UTF-8

Jeden z nejpoužívanějších formátů, protože je prostorově úsporný. Je odolný proti chybám a je zpětné kompatibilní s ASCII. Setkáme se s ním na většině webových stránkách, ale často i v některých operačních systémech. Pro kódování znaků používá 1 – 4 Byty podle jejich kódu v Unicode. [19, 20, 21]

Tabulka 7.1: Rozdělení UTF-8 podle bytů

Významové bity	První Unicode pozice	Poslední Unicode pozice	Vedoucí Byte 1 binárně	Byte 2 binárně	Byte 3 binárně	Byte 4 binárně
7	U+0000	U+007F	0xxx xxxx			
11	U+0080	U+07FF	110x xxxx	10xx xxxx		
16	U+0800	U+FFFF	1110 xxxx	10xx xxxx	10xx xxxx	
21	U+1 0000	U+1F FFFF	1111 0xxx	10xx xxxx	10xx xxxx	10xx xxxx

V tabulce 7.1 můžeme vidět princip tvoření formátu UTF-8. Formát je rozdělený do čtyř skupin a každá skupina má svá pravidla, respektive prefix pro vytvoření formátu.

7.1.1 Převod Unicode do UTF-8

Převod do formátu probíhá tak, že Unicode znak převedeme do binární soustavy a poté rozdělíme do skupin podle tabulky 7.1. Například, když budeme mít Unicode do 007F, tak použijeme všech 7 bitů pro vytvoření UTF-8. Vedoucí neboli leading Byte využívá při každém formátu jiný binární prefix, a tedy jiný počet bytů, který můžeme doplnit. U dalších bytů pak vždy doplňujeme k binárnímu prefixu 10 šest znaků.

Vytvoření UTF8 pro znak U+15698

1. Převod do binární soustavy
15698 = $(0001010101101001 1000)_B$

2. Přiřazení k bytům
Binární číslo si rozdělíme zprava po 6, poslední skupina bude mít pouze 2 znaky
1. Skupina 011000
2. Skupina 011010
3. Skupina 010101
4. Skupina 00
Číslo má 20 bitů, avšak máme k dispozici 21 významových bitů, proto k poslední skupině přiřadíme 0 zleva
4. Skupina 000

3. Vytvoření jednotlivých bytů
Za x pak doplníme jednotlivé skupiny znaků, jelikož jsme začali tvořit skupiny od konce, tak i Byty musíme tvořit od konce
Byte4 = 10xx xxxx = 10011000
Byte3 = 10xx xxxx = 10011010
Byte2 = 10xx xxxx = 1001 0101
Byte1 = 1111 0xxx = 1111 0000

4. Seřadíme Byte a převedeme do Hexadecimální soustavy
+ v tomto případě neznamená součet, ale zřetězení
Byte1 + Byte2 + Byte3 + Byte4 = 11110000100101011001101010011000
$(11110000100101011001101010011000)_B = (F0959A98)_H$

Výsledek ve formátu UTF8 0xF0 0x95 0x9A 0x98

Obrázek 7.1: Převod z Unicode do formátu UTF-8

7.1.2 Převod UTF-8 do Unicode

Zpětný převod probíhá tak, že Hexadecimální znaky převedeme do binární soustavy, rozdělíme do skupiny po Bytech a odstraníme v prefixy podle tabulky 7.1. Poté složíme do jedné binární sekvence znaků a převedeme zpět do hexadecimální soustavy.

Převod 0xF0 0x9F 0xAA 0xB3 na Unicode

1. Převod do binární soustavy
$(F09FAAB3)_H = (11110000100111111010101010110011)_B$

2. Rozdělení na Byty
Byte1 = 11110000
Byte2 = 10011111
Byte3 = 10101010
Byte4 = 10110011

3. Odstranění prefixů
Mínus v tomto případě používáme pro odstranění prefixu zleva
Skupina1 = 1111 0000 - 1111 0xxx = 000
Při vytváření UTF jsme jednu nulu přidali, tentokrát ji tedy musíme odstranit
Skupina1 = 00
Skupina2 = 1001 1111 - 10xx xxxx = 011111
Skupina3 = 1010 1010 - 10xx xxxx = 101010
Skupina4 = 1011 0011 - 10xx xxxx = 110011

4. Spojení do jedné binární posloupnosti
Plus v tomto případě znamená zřetězení
Skupina1 + Skupina2 + Skupina3 + Skupina4 = 000111111010101110011

5. Převedeme do Hexadecimální soustavy
$(000111111010101110011)_B = (1FAB3)_H$

Výsledek U+1 FAB3

Obrázek 7.2: Převod z UTF-8 do Unicode

7.2 UTF-16

UTF-16 používá až dvě 16 bitové dvojice, aby pokryl celý 21 bitový prostor. Pokud je využita pouze jedna z dvojic, tedy v případě hodnot do U+FFFF, tak se hodnoty kódují, tak jak jsou bez žádné změny. V případě, že je Unicode větší než U+FFFF je potřeba využít obě dvojice. Pro tento případ existuje speciální dvojice takzvaná surrogate pair. První z dvojic, zvaná lead surrogate je v rozsahu 0xD800 až 0xDBFF. Druhá z dvojic, zvaná trail surrogate je v rozsahu 0xDC00 až 0xDFFF.

UTF-16 se používá ve všech podporovaných operačních systémech od společnosti Microsoft. Společnost Apple využívá tento formát pro SMS (Short Massage Service). [19, 20, 22, 21]

7.2.1 Převod Unicode do UTF-16

V případě, že máme Unicode hodnoty do U+FFFF, tak se vytvoří formát s hodnotou tak jak je. Například pro U+1234 je UTF16 formát roven 0x1234. V případě, že potřebujeme vyjádřit Unicode hodnotu, která je vyšší než U+FFFF, tak musíme využít lead a trail surrogate. Samotný převod pak funguje tak, že od Unicodu odečeteme číslo 0x01 0000. Poté převedeme do binární soustavy a rozdělíme do dvou dvojic po 10 bitech zprava ty pak převedeme zpět do Hexadecimální soustavy. Nižších 10 bitů (bity s nižší váhou) náleží trail surrogate, vyšších 10 bitů (bity s vyšší váhou) náleží lead surrogate. K lead surrogate poté přičteme 0xD800 a k trail surrogate přičteme 0xDC00.

Převod U+2 FACE do formátu UTF16

1. Odečíst 0x10000
2FACE - 10000 = $(1FACE)_H$

2. Převést do Binární soustavy
$(1FACE)_H = (00011111101011001110)_B$

3. Rozdělení po 10 bitech
lead = $(0001111110)_B = (07E)_H$
trail = $1011001110 = (2CE)_H$

4. Přičtení D800 a DC00
lead = D800 + 07E = D87E
trail = DC00 + 2CE = DECE

Výsledný formát v UTF-16 Big Endian 0xD87E 0xDECE
Výsledný formát v UTF-16 Little Endian 0x7ED8 0xCEDE

Obrázek 7.3: Převod z Unicode do UTF-16

7.2.2 Převod UTF-16 do Unicode

Převod se provátí tak, že od lead surrogate odečteme hodnotu 0xD800 a od trail surrogate odečteme hodnotu 0xDC00. Čísla převedeme do binární soustavy a odstraníme, nebo přidáme nuly tak, abychom vytvořili 2 skupiny po 10. Skupiny poté zřetězíme, převedeme do hexadecimální soustavy a přičteme hodnotu 0x10000.

Převod 0xDB80 0xDF13 Big Endian do Unicode

1. Odečíst 0xD800 a 0xDC00
lead = DB80 - D800 = $(380)_H$
trail = DF13 - DC00 = $(313)_H$

2. Převod do binární soustavy
$(380)_H = (001110000000)_B$
$(313)_H = (001100010011)_B$

3. Rozdělení do skupin po 10 a zřetězení
Skupina1 = 1110000000
Skupina2 = 1100010011

Znak + v tomto případě slouží pro zřetězení
Skupina1 + Skupina2 = 11100000001100010011

4. Převod do Hexadecimální soustavy
$(11100000001100010011)_B = (E0313)_H$

5. Přičíst 0x10000
E0313 + 10000 = $(F0313)_H$

Výsledek U+F0313

Obrázek 7.4: Převod z UTF-16 do Unicode

7.3 UTF-32

Tento formát využívá pro slovo 32 bitů. Jelikož Unicode má maximálně 21 bitů, tak při převodu do UTF-32 se doplní nuly do 32 bitů. Tento formát jinak odpovídá Unicodu.

7.3.1 Převod do formátu UTF-32 a zpětný převod

Převod funguje tak, že vezmeme Unicode tak jak je a doplníme nuly do 4 bytů (32 bitů). Převod z formátu UTF-32 pak funguje tak, že nuly odstraníme. Jediné, na co je třeba dávat pozor je endianita.

Převod U+12345 do UTF32

1. Přidání nul do 4 Bytů
12345 má 2,5 Bytů (20 bitů) -> přidáme 3 nuly
$(00012345)_H$

Výsledný formát v UTF-32 Big Endian 0x00 0x01 0x23 0x45
Výsledný formát v UTF-32 Little Endian 0x45 0x23 0x01 0x00

Převod UTF-32 BE 0x00 0x00 0xDE 0xDA do Unicode

1. Odstraníme nuly zleva
Výsledný formát v Unicode U+DEDA

Převod UTF 32 LE 0xF1 0xF2 0x01 0x00 do Unicode

1. Odstraníme nuly zprava
F1F201

2. Převedeme na BE
Výsledný formát v Unicode U+01 F2F1

Obrázek 7.5: Převod z unicode do UTF-32 a zpětný převod

7.4 Byte order mark

Byte order mark, zkráceně BOM je Unicode znak, který se používá k označení endianity, či verze UTF v textovém souboru nebo přenosu. Použití je dobrovolné, ale když už se použije, tak by měl být na začátku textového souboru, nebo na začátku proudu přenosu. [19, 8]

Kapitola 8

Vývoj webové aplikace Číselné soustavy

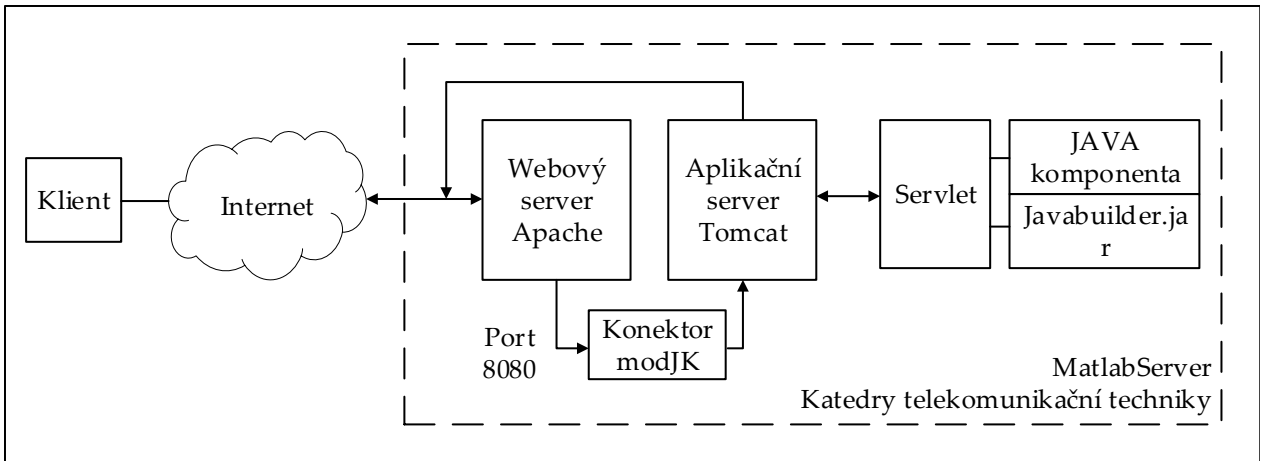
Z důvodu vysokého počtu skriptů jsem se rozhodnul vytvořit grafické webové rozhraní, které zjednoduší používání jednotlivých aplikací. Mé skripty běží na MatlabServeru Katedry telekomunikační techniky vytvořený Ing. Marcelem Fajkusem v rámci jeho diplomové práce. V této kapitole budu popisovat, jak jsem postupoval při vývoji aplikací, které slouží k převodům do formátů zmíněných v teorii a k nim základní aritmetické operace a jak probíhala implementace na MatlabServer.

8.1 MatlabServer Katedry telekomunikační techniky

Matlab server Katedry telekomunikační techniky je implementován na virtuálním stroji, který běží ve školní síti na webovém serveru s IP adresou `http://158.196.109.157/` nebo na doménovém jméně `http://matlab-comtech.vsb.cz`. Pro přístup k MatlabServeru je potřeba být ve školní sítí, v případě, že chceme k serveru přistupovat z domu, je potřeba se do školní sítě připojit pomocí Virtuální privátní sítě (VPN).

8.1.1 Architektura MatlabServeru

Architektura MatlabServeru (obr. 8.1) je založena na webovém serveru Apache (verze 2.4), na kterém běží uživatelské rozhraní. To je implementováno v HTML ve spojení s PHP jazykem, Javascriptem a databází. Z tohoto uživatelského rozhraní je možné provádět veškerou správu MatlabServeru (správa aplikací, uživatelů, kategorií apod.) a spouštět požadované aplikace. [23]

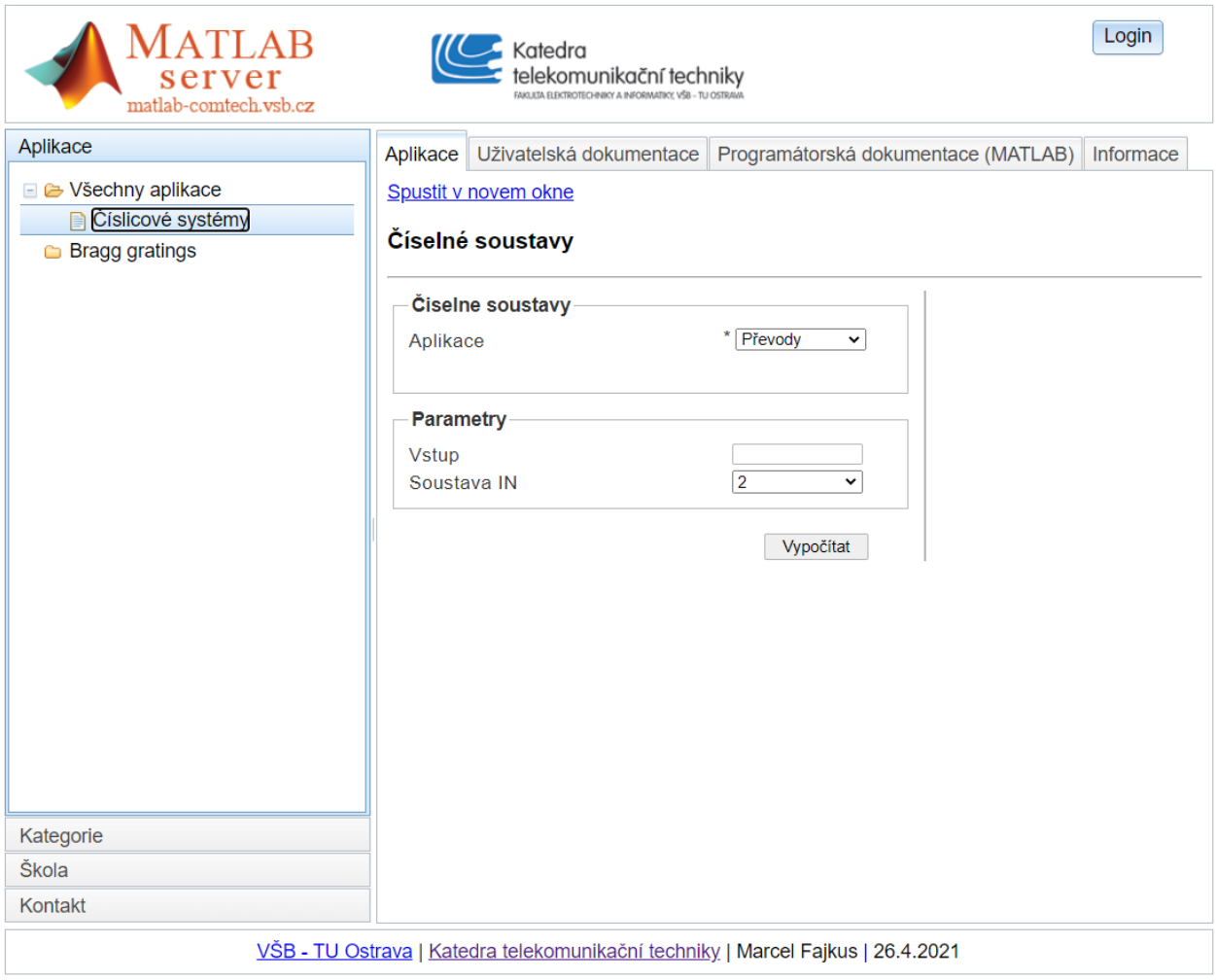


Obrázek 8.1: Architektura MatlabServeru Katedry telekomunikační techniky

Při spuštění webové aplikace (dále WebMatlab aplikace) se zobrazí uživatelské rozhraní dané aplikace. Po vyplnění vstupních parametrů se odešle požadavek na MatlabServer s definovanou URL adresou. V takovém případě webový server přepošle požadavek přes konektor modJK na aplikační server Tomcat. Zde se spustí odpovídající servlet, který převezme vstupní parametry a zavolá požadovanou JAVA komponentu, která obsahuje Matlab kód aplikace. Sestavení JAVA komponenty z Matlab kódu se realizuje pomocí toolboxu Matlab Builder JA a Matlab Compiler. JAVA komponenta tedy provede výpočetní operace a výsledky odešle nazpět servletu, který vygeneruje odpověď. Tato odpověď se poté odešle klientovi a zobrazí se ve webovém prohlížeči. Pro svou činnost servlet vyžaduje také knihovnu javabuilder.jar, která je součástí Matlabu a obsahuje potřebné funkce pro datové převody parametrů mezi Javou a Matlabem apod. [23]

8.1.2 Grafické uživatelské rozhraní MatlabServeru

Grafické uživatelské rozhraní MatlabServeru je zobrazeno na obrázku 8.2. V levé části jsou umístěny záložky, které slouží pro kategorizaci jednotlivých aplikací. V pravé části je zobrazena pracovní plocha, ve které se zobrazují grafické uživatelské rozhraní jednotlivých aplikací pro zadávání vstupních dat, napravo od svislé čáry je potom prostor, ve kterém se zobrazují generované výsledky z aplikací.



Obrázek 8.2: Grafické rozhraní MatlabServeru

Při spuštění aplikace se primárně otevře její uživatelské rozhraní. V horní části pak jsou další záložky, ve kterých si lze otevřít uživatelskou dokumentaci k aplikaci, dokumentací ke zdrojovému kódu v Matlabu a další informace jako je kategorizace aplikace, autora aplikace apod.

8.1.3 Vývoj WebMatlab aplikace

Každá aplikace je tvořena následujícími částmi:

- Uživatelské rozhraní (HTML + javascript), které slouží pro zadávání vstupních parametrů
- Servlet, který čte požadavky od uživatele, spouští MATLAB kód (Java komponenta) a generuje výsledky, které odesílá klientovi
- Java komponenta, do které je zabalen MATLAB kód pomocí toolboxu Matlab Builder JA a Matlab Compiler

- Konfigurační soubory, které se starají o mapování názvu aplikací a umístění jednotlivých částí aplikace

Mým úkolem bylo poskytnout zdrojové kódy v Matlabu a upravit je tak, aby je bylo možné použít na webovém rozhraní, jinak řečeno vytvoření Java komponenty.

8.1.4 Úkoly při vývoji webové aplikace

Vývoj WebMatlab aplikace vyžaduje řadu činností, které je potřeba provádět v předem definovaných fázích vývoje aplikace. Při vývoji WebMatlab aplikace a jejich nasazení na MatlabServer zajišťují následující osoby:

1. Matlab programátor (MP)
2. Java programátor (JP)
3. HTML programátor (HP)
4. Správce aplikačního serveru (SAS)
5. Správce webového rozhraní MatlabServeru (SWRM)

Vývoj webové aplikace a její nasazení na MatlabServer se skládá z těchto činností. V závorce je typ vývojáře viz předchozí seznam.

1. Implementace Matlab aplikace (MP)
2. Sestavení JAVA komponenty (MP)
3. Implementace servletu (JP)
4. Tvorba uživatelského rozhraní webové aplikace (HP)
5. Tvorba uživatelské dokumentace (MP)
6. Sestavení programátorské dokumentace Matlab kódu (MP)
7. Sestavení a nasazení webové aplikace na web (SAS)
8. Přidání webové aplikace do uživatelského rozhraní MatlabServeru (SWRM)

8.2 WebMatlab aplikace číselné soustavy

V následující části je popsán postup vývoje Matlab aplikací.

8.2.1 Základní operace v Matlabu

Hlavní náplní této práce, bylo vytvoření scriptů v programu Matlab. Matlab je nástroj pro vědecké a technické výpočty, analýzu dat a vývoj algoritmů. Součástí je vlastní programovací jazyk, který má rozsáhlou knihovnu funkcí pro práci s daty, jako jsou čísla nebo stringy (textové řetězce). Matlab nabízí také převodníky do jiných soustav, či jiné funkce pro práci s číslicovými systémy. Tyto funkce jsem nepoužil a každou funkci jsem si napsal sám přesně tak, jak jsem jí potřeboval, abych jí mohl využít dále ve své práci. [24]

Velkou část scriptů tvoří právě práce se stringy. Spousta funkcí využívá textové řetězce pro vstupní i výstupní parametry, ale i pro mezi výpočty. Je důležité zmínit, že matlab nevyužívá číslování od nuly, jako ostatní programovací jazyky, ale využívá číslování od jedničky. To znamená, že když procházíme textový řetězec, tak první znak nebude mít index 0, ale bude mít index 1. Pro práci se stringy jsem používal převážně funkce `append`, která spojí textové řetězce do jednoho, `extractAfter`, která vrátí část řetězce po určitém znaku, či pozici, `extractBefore`, která vrátí část textového řetězce před určitým znakem, či pozicí, `extractBetween`, která vrátí část řetězce v určitém rozmezí od do a `fliplr`, která vrátí řetězec obráceně, tedy psaný zprava doleva.

Příklad využití funkcí pro práci se stringy:

```
append("1111", "0000") %Výstupem bude "11110000"
extractAfter("111.01010", ".") %Vrátí všechny znaky za tečkou, výstupem bude
    "01010"
extractAfter("111.01010", 1) %Vrátí všechny znaky za první pozicí, výstupem bude
    "11.01010"
extractBefore("111.01010", ".") %Vrátí všechny znaky před tečkou, výstupem bude
    "111"
extractBefore("111.01010", 4) %Vrátí všechny znaky před indexem 4-, výstupem bude
    "111"
extractBetween("ABCDEFFEDCBA", 2, 7) %Vrátí všechny znaky s~indexem od 2 do 7, vý
    stupem bude "BCDEFF"
fliplr ('101010') %Výstupem bude "010101"
```

Listing 8.1: Práce se stringy v prostředí Matlab

Znak % se v programovacím prostředí matlab využívá pro komentáře.

Důležitou roli v mé práci hrály cykly while a for, které opakují určitou část kódu, dokud nesplní zadanou podmínku. V případě for cyklu se kód opakuje podle zadaného počtu opakování. To znamená, že je potřeba zadat hodnoty od do. Standardně se používá krok 1, avšak ten si můžeme nastavit také. V případě while cyklu s kód opaku do doby, dokud je zadaná podmínka pravdivá (true), jakmile je nepravdivá (false), tak se cyklus přeruší. [26, 25]

Příklad využití cyklů:

```
n = 10;
while n > 1 %Cyklus se opakuje, dokud proměnná n nebude menší než 1
    n = n-1;
    %Zde bude část kódu, která se má opakovat
end %značí ukončení while cyklu

for c = 1:n %Opakování bude od 1 do n, v~tomto případě tedy od 1 do 10
%Zde bude část kódu, která se má opakovat
end %Značí konec for cyklu
```

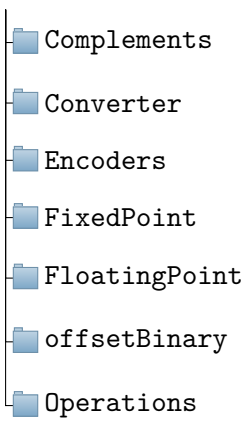
Listing 8.2: Práce se stringy v prostředí Matlab

Při práci se stringy je potřeba myslet na to, že vždy musí mít konec. To znamená, že je potřeba vymyslet rozumnou podmínku, aby se funkce nezacyklila.

8.2.2 Implementace dílčích aplikací

V rámci mé práce jsem vyvíjel aplikace, které odpovídají formátům a aritmetickým operacím v jednotlivých kapitolách. Celkově jsem tak vyvinul 40 aplikací. Některé z aplikací jsou pouze pomocné. Například aplikace na přidávání a mazání nul. Každý název aplikace má prefix "AO" jako Aritmetické Operace, aby se předešlo tomu, že by nějaká aplikace Matlabu mohla mít stejný název. Jelikož je počet aplikací poměrně vysoký, rozhodl jsem se vytvořit adresářovou strukturu tak, aby aplikace měli nějaké logické rozdělení.

Adresářová strukutra vypadá takto:



V kořenovém adresáři `Sources` se nachází aplikace `Main.m`, která mapuje veškeré adresáře, aby bylo možné z této aplikace ovládat aplikace ve všech adresářích. V adresáři Complements se nachází aplikace, které souvisí s doplňky. Jednotlivé aplikace a příklad využití můžeme vidět ve zdrojovém kódu 8.3.

```
%Aplikace pro převod z/do jedničkového doplňku
%Vstupem je číslo v binární soustavě v jednoduchých uvozovkách
AO_BinToOnesComplement('101010')

%Aplikace pro převod z/do dvojkového doplňku
%Vstupem je číslo v binární soustavě v jednoduchých uvozovkách
AO_BinToTwosComplement('101010')

%Aplikace pro převod hodnoty v dekadické soustavě do jedničkového doplňku
%První vstup je dekadická hodnota, druhý vstup je bitový rozsah
AO_DecToOnesComplement(-16, 8)

%Aplikace pro převod hodnoty v dekadické soustavě do dvojkového doplňku
%První vstup je dekadická hodnota, druhý vstup je bitový rozsah
AO_DecToTwosComplement(-16, 8)

%Aplikace pro převod z jedničkového doplňku do dekadické soustavy
%První vstup je hodnota v binární soustavě, druhý vstup je bitový rozsah
AO_OnesComplementToDec('10000111', 8)

%Aplikace pro převod z dvojkového doplňku do dekadické soustavy
%První vstup je hodnota v binární soustavě, druhý vstup je bitový rozsah
AO_TwosComplementToDec('10000111', 8)
```

Listing 8.3: Aplikace pro práci s doplňky a příklady využití

V adresáři Converter se nacházejí aplikace, které slouží pro převod mezi soustavami. Jednotlivé aplikace a příklady využití můžeme vidět ve zdrojovém kódu 8.4

```
%Aplikace pro převod z binární do decimální soustavy
%Vstupem je binární hodnota v jednoduchých uvozovkách
AO_BinToDec('110011')

%Aplikace pro převod z binární do hexadecimální soustavy
%Vstupem je binární hodnota v jednoduchých uvozovkách
```

```
AO_BinToHex('110011')

%Aplikace pro převod z decimální do binární soustavy
%Vstupem je hodnota v desítkové soustavě
AO_DecToBin(10)

%Aplikace pro převod z decimální do hexadecimální soustavy
%Vstupem je hodnota v desítkové soustavě
AO_DecToHex(10)

%Aplikace pro převod z hexadecimální do binární soustavy
%Vstupem je hexadecimální hodnota (bez prefixu 0x) v jednoduchých uvozovkách
AO_HexToBin('ABC')

%Aplikace pro převod z hexadecimální do binární soustavy
%Vstupem je hexadecimální hodnota (bez prefixu 0x) v jednoduchých uvozovkách
AO_HexToBin('ABC')
```

Listing 8.4: Aplikace pro převody mezi soustavami a příklady využití

V adresáři Encoders se nacházejí aplikace, které souvisí s formátem UTF. Jednotlivé aplikace a příklady využití můžeme vidět ve zdrojovém kódu 8.5

```
%Aplikace pro převod z Unicode do UTF-8
%Vstupem je Unicode(bez prefixu U+) v jednoduchých uvozovkách
AO_UTF8('102030')

%Aplikace pro převod z UTF-8 do Unicode
%Prvním vstupem je formát UTF-8 (bez prefixů 0x) v jednoduchých uvozovkách
%Formát prvního vstupu je xx, xx xx, xx xx xx nebo xx xx xx xx
AO_UTF8ToUnicode('F4 82 80 B0')

%Aplikace pro převod z Unicode do UTF-16
%Prvním vstupem je Unicode(bez prefixu U+) v jednoduchých uvozovkách
%Druhým vstupem je endianita zapsaná jako 'le', nebo 'be'
AO_UTF16('102030', 'le')

%Aplikace pro převod z UTF-16 do Unicode
%Prvním vstupem je formát UTF-16 (bez prefixů 0x) v jednoduchých uvozovkách
%Formát prvního vstupu je xxxx nebo xxxx xxxx
%Druhým vstupem je endianita zapsaná jako 'le', nebo 'be'
AO_UTF16ToUnicode('C8DB 30DC', 'le')

%Aplikace pro převod z Unicode do UTF-32
%Prvním vstupem je Unicode(bez prefixu U+) v jednoduchých uvozovkách
%Druhým vstupem je endianita zapsaná jako 'le', nebo 'be'
AO_UTF32('102030', 'be')

%Aplikace pro převod z UTF-32 do Unicode
%Prvním vstupem je formát UTF-32 (bez prefixů 0x) v jednoduchých uvozovkách
%Formát prvního vstupu je xx xx xx xx
AO_UTF32ToUnicode('00 10 20 30')
```

Listing 8.5: Aplikace pro práci s formáty UTF a příklady využití

V adresáři FixedPoint se nacházejí aplikace, které souvisí s čísly v pevné řádové čárce. Jednotlivé aplikace a příklady využití můžeme vidět ve zdrojovém kódu 8.6

```
%Aplikace pro převod do formátu Qm.f
%Prvním vstupem je reálné číslo (s desetinnou tečkou)
%Druhým vstupem je m a třetím vstupem je f
%Aplikace má 2 výstupy a to formát IN a FX
[In, FX] = AO_FixedPoint(-1.33, 1,14)

%Aplikace pro převod z formátu Qm.f
%Prvním vstupem je číslo ve formátu Qm.f IN
%Druhým vstupem je m a třetím vstupem je f
AO_FixedPointToDec('7D70', 1,14)

%Aplikace pro sčítání v pevné řádové čárce
%První dva vstupy jsou reálná čísla (s desetinnou tečkou)
%Třetím vstupem je m a čtvrtým vstupem je f
AO_FixedPointSum(1.35, 0.61, 1, 14)

%Aplikace pro odčítání v pevné řádové čárce
%První dva vstupy jsou reálná čísla (s desetinnou tečkou)
%Třetím vstupem je m a čtvrtým vstupem je f
AO_FixedPointMinus(-1.33, -1.33,1 , 14)
```

Listing 8.6: Aplikace pro práci s čísly v pevné řádové čárce a příklady využití

V adresáři FloatingPoint se nacházejí aplikace, které souvisí s čísly v plovoucí řádové čárce. Jednotlivé aplikace a příklady využití můžeme vidět ve zdrojovém kódu 8.7

```
%Aplikace pro převod do formátů BinaryX
%První vstup je reálné číslo (s desetinnou tečkou)
%Druhý vstup je pro formáty BinaryX, možnosti jsou 16, 32, 64, 128
AO_BinaryX(1.66, 32)

%Aplikace pro převod z formátů BinaryX
%Vstupem je číslo ve formátu BinaryX v jednoduchých uvozovkách (bez mezer)
AO_BinaryXToDec('C0547AE1')

%Aplikace pro sčítání ve formátu BinaryX
%Vstupem jsou čísla ve formátu BinaryX v jednoduchých uvozovkách (bez mezer)
AO_BinaryXSum('BFD47AE1', '4FD47AE1')
```

```
%Aplikace pro odčítání ve formátu BinaryX
%Vstupem jsou čísla ve formátu BinaryX v jednoduchých uvozovkách (bez mezer)
AO_BinaryXMinus('BFD47AE1', '3FD47AE1')
```

Listing 8.7: Aplikace pro práci s čísly v plovoucí řádové čárce a příklady využití

V adresáři OffsetBinary se nacházejí aplikace, které souvisí s čísly v posunutí. Jednotlivé aplikace a příklady využití můžeme vidět ve zdrojovém kódu 8.8

```
%Aplikace pro převod do formátu s posunutím
%Prvním vstupem je kladné, nebo záporné celé číslo
%Druhým vstupem je posunutí
AO_OffsetBinary(50, 127)

%Aplikace pro převod z formátu s posunutím
%Prvním vstupem je nezáporné celé posunuté číslo
%Druhým vstupem je posunutí
AO_OffsetBinaryToDec(136, 127)

%Aplikace sčítání ve formátu s posunutím
%První dva vstupy jsou kladná, nebo záporná celá čísla
%Třetím vstupem je posunutí
AO_OffsetBinarySum(50, 41, 127)

%Aplikace odčítání ve formátu s posunutím
%První dva vstupy jsou kladná, nebo záporná celá čísla
%Třetím vstupem je posunutí
AO_OffsetBinaryMinus(50, 41, 127)
```

Listing 8.8: Aplikace pro práci s čísly v posunutí a příklady využití

V adresáři Operations se nacházejí aplikace, které slouží k aritmetickým operacím v různých soustavách a pomocné funkce. Jednotlivé aplikace a příklady využití můžeme vidět ve zdrojovém kódu 8.9

```
%Binární sčítačka pro odčítání
%První dva vstupy jsou čísla v binární soustavě
%Třetí vstup je bitový rozsah
%Má 5 výstupů, výsledek a 4 příznaky (N, Z, V, C)
[result, N, Z, V, C] = AO_BinaryCounterMinus('1101', '0101', 4)
```

```
%Binární sčítačka
%První dva vstupy jsou čísla v binární soustavě
%Třetí vstup je bitový rozsah
%Má 5 výstupů, výsledek a 4 příznaky (N, Z, V, C)
[result, N, Z, V, C] = AO_BinaryCounterSum('1101', '0101', 4)

%Aplikace pro odčítání v hexadecimální soustavě
%Oba vstupy jsou čísla v hexadecimální soustavě
AO_HexMinus('ABC', 'ABC')

%Aplikace pro sčítání v hexadecimální soustavě
%Oba vstupy jsou čísla v hexadecimální soustavě
AO_HexSum('ABC', 'ABC')

%Aplikace pro odčítání v binární soustavě
%Neřeší příznaky, neumí záporná čísla
%Oba vstupy jsou čísla v binární soustavě
AO_MINUS('11010', '111')

%Aplikace pro násobení v binární soustavě
%Oba vstupy jsou čísla v binární soustavě
AO_Multiplication('111', '101')

%Aplikace pro odčítání v binární soustavě
%Neřeší příznaky ani přetečení, neumí záporná čísla
%Oba vstupy jsou čísla v binární soustavě
AO_SUM('111', '1')
```

Listing 8.9: Aplikace pro práci s čísly v posunutí a příklady využití

8.2.3 Implementace WebMatlabAplikace

V následující části je stručně popsána implementace WebMatlab aplikace Číselné soustavy.

8.2.3.1 Uživatelské rozhraní

HTML kód s uživatelským rozhraním aplikace je zobrazen níže. Stěžejní část je položka tag `select` s identifikátorem `type_miniapp`, v rámci které si uživatel může zvolit požadovanou mini aplikaci.

Po výběru konkrétní aplikace se v tagu `div` s identifikátorem `fieldsetParams` zobrazí položky pro zadání vstupních parametrů. Po vykonání aplikace se vygenerované výsledky (kódované do HTML jazyka) zobrazí v tagu `div` s identifikátorem `resultFromMatlab`.

```
<form method="get" onsubmit="return false" id="formular" class="cmxform" >
      <fieldset>
        <legend>Číselne soustavy</legend>
        <p>
        <label>Aplikace</label>
        <span class='require'> * </span>
        <select name="type_miniapp" id="type_miniapp" onchange='selectMiniapp()'>
          <option value='1'>Převody</option>
          <option value='2'>Doplňky</option>
          <option value='3'>Offset binary</option>
          <option value='4'>Fixed point</option>
          <option value='5'>Floating point</option>
          <option value='6'>UTF</option>
          <option value='7'>Aritmetika binary</option>
          <option value='8'>Aritmetika Offset</option>
          <option value='9'>Aritmetika FX</option>
          <option value='10'>Aritmetika FP</option>
        </select>
        </p>
      </fieldset>
      <fieldset >
        <legend>Parametry</legend>
        <div id='fieldsetParams'></div>
      </fieldset>
        <input type="submit" class="submit" id="submitButton" value="Vypočítat"
              onclick="getResultFromMatlab()">
        </form>
     <!-- Zobrazeni vystupu do DIVu s id resultFromMatlab -->
      <div id="resultFromMatlab" style=""></div>
```

Listing 8.10: Ukázka HTML souboru

Protože je aplikace tvořena dílčími aplikacemi, při zvolení konkrétní aplikace se pomocí Javascriptu vytvoří formulář pro zadání požadovaných vstupních parametrů. V následujícím výpisu je zobrazeno, jakým způsobem je vytvořen formulář pro zadávání vstupních parametrů v případě zvolení miniaplikace UTF.

```
function selectUTF() {
    var str = "";
    str += "<p><label>Vstup</label><input type='text' name='par1' id='par1' value
       ='' required></p>";
    str += "<p><label>Typ vstupu</label>" +
           "<select name='par2' id='par2'>" +
           "<option value='unicode'>UNICODE</option>" +
           "<option value='utf8'>UTF-8</option>" +
           "<option value='utf16'>UTF-16</option>" +
           "<option value='utf32'>UTF-32</option>" +
           "</select>" +
           "</p>";
       $("#fieldsetParams").html(str);
}
```

Listing 8.11: Ukázka javascriptu

8.2.3.2 Servlet

Servlet je rozsáhlý kód implementovaný v Javě. Stěžejní částí je však převzetí vstupních parametrů, zavolání funkce v JAVA komponentě `jComp.miniAppUTF` a vygenerování odpovědi, která se klientovi odešle. [23]

```
input = request.getParameter("par1");
type_in = request.getParameter("par2");
result = jComp.miniAppUTF(1, input, type_in);
printHTMLOutput(request,response,result);
```

Listing 8.12: Ukázka servletu

8.2.3.3 JAVA componenta

JAVA komponenta v sobě zabaluje Matlab kód. V případě mini aplikace UTF obsahuje volaná funkce následující kód. V tomto místě se na základě vstupních parametrů zavolá odpovídající převod. [23]

```
function [message] = miniAppUTF(input, type_in)
LOG = log4ms('debug');

switch type_in
    case char('unicode')
```

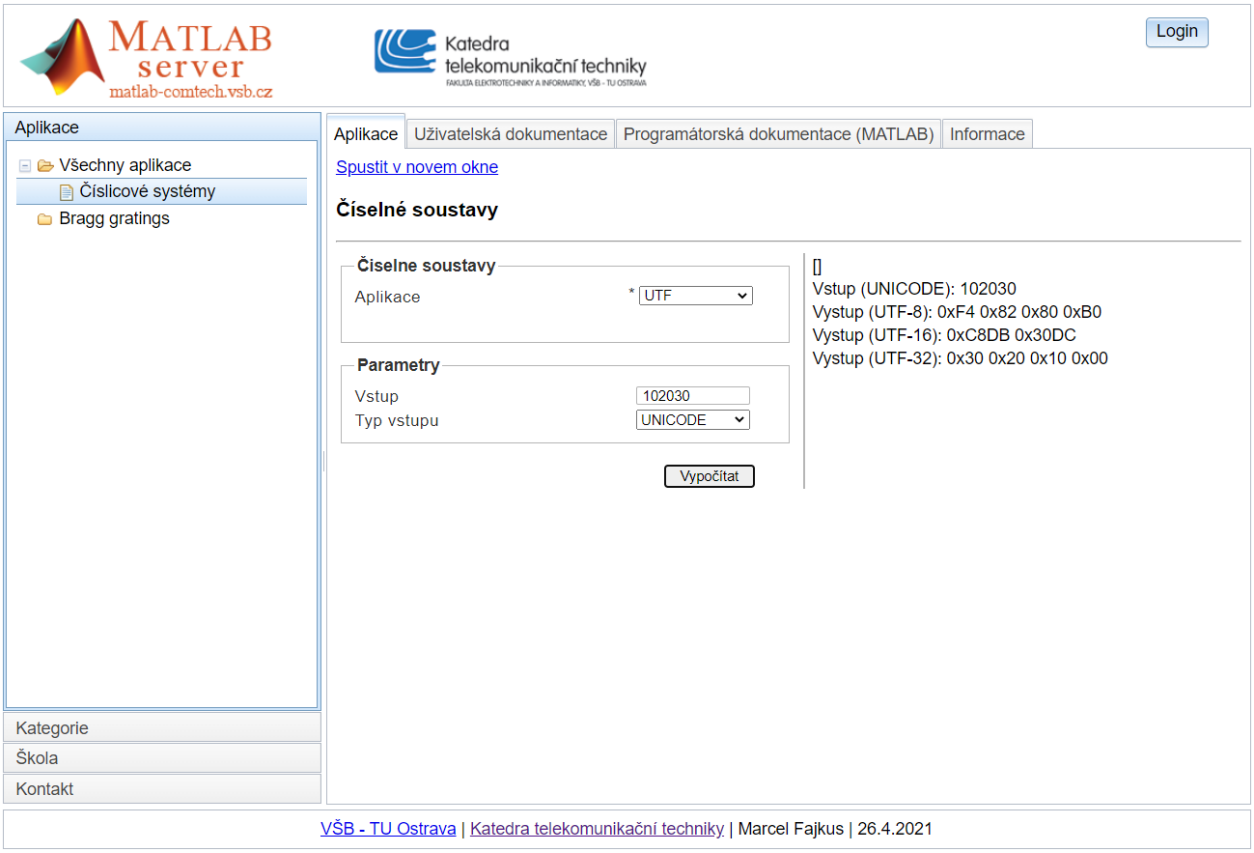
```
        LOG = LOG.debug('Vstup (UNICODE): %s', {input});
        [res, LOG] = AO_UTF8(input, LOG);
        LOG = LOG.debug('Vystup (UTF-8): %s', {res});
        [res, LOG] = AO_UTF16(input, 'LE', LOG);
        LOG = LOG.debug('Vystup (UTF-16): %s', {res});
        [res, LOG] = AO_UTF32(input, 'LE', LOG);
        LOG = LOG.debug('Vystup (UTF-32): %s', {res});
    case char('utf8')
        LOG = LOG.debug('Vstup (UTF-8): %s', {input});
        [res, LOG] = AO_UTF8ToUnicode(input, LOG);
        LOG = LOG.debug('Vystup (UNICODE): %s', {res});
    case char('utf16')
        LOG = LOG.debug('Vstup (UTF-16): %s', {input});
        [res, LOG] = AO_UTF16ToUnicode(input, 'LE', LOG);
        LOG = LOG.debug('Vystup (UNICODE): %s', {res});
    case char('utf32')
        LOG = LOG.debug('Vstup (UTF-32): %s', {input});
        [res, LOG] = AO_UTF32ToUnicode(input, 'LE', LOG);
        LOG = LOG.debug('Vystup (UNICODE): %s', {res});

end
message = LOG.message;
end
```

Listing 8.13: Ukázka JAVA componenty

Uvedené funkce kromě výsledku vytváří mezivýsledky, které jsou formátovány do proměnné `message`, která ja poté odeslána klientovi a zobrazí se ve webovém prohlížečí.

8.2.3.4 Ukázka miniaplikace UTF

Na obrázku 8.3 můžeme vidět příklad již funkční aplikace. V okénku číselné aplikace si vybereme aplikaci, kterou chceme používat, v nabídce jsou aplikace pro soustavy, doplňky, čísla v posunutí, čísla v pevné řádové čárce, čísla v plovoucí řádové čárce, a nakonec formáty UTF. V okénku parametry už jsou jednotlivé vstupy týkající se vybrané aplikace. V ukázce stačí vyplnit políčko "Vstup" a vybrat o jaký formát se jedná. Když vybereme formát Unicode, tak na pravé straně, která slouží pro výstup, uvidíme převod do všech UTF formátů. V případě, že jako typ vstupu vybereme jeden z formátu, tak se náš vstup převede do Unicode. V případě formátu pro zobrazení reálných čísel je počet vstupů větší, součástí je i rozbalovací menu pro výběr mezi převodem a aritmetickými operacemi.



Obrázek 8.3: Ukázka aplikace pro kódování a dekódování znaků

Kapitola 9

Závěr

V této práci jsem se věnoval některým způsobům ukládání a používání čísel v digitálních přístrojích. Zabýval jsem se převážně převodům a základními aritmetickými operacemi, a to převážně sčítáním a odčítáním v různých formátech, které se používají, nebo se v minulosti používaly v různých zařízeních. Velká část teorie je věnována formátům, které se používají pro zobrazení reálných čísel, protože to jsou čísla, se kterými se pravidelně setkáváme, ale neuvědomujeme si, že v počítačích nejsou reprezentovány tak, jak je známe my.

Jednu kapitolu jsem se věnoval způsobům kódování znaků, které se používají pro reprezentaci znaků, například v programovacích jazycích, operačních systémech nebo webových stránkách.

Stěžejní částí této bakalářské práce bylo vytvoření programů ve vývojovém prostředí MATLAB, které slouží k převodům do jednotlivých formátů, ale také umí základní operace, jako je sčítání a odčítání v daném formátu. Aplikace jsem tvořil podle teoretických znalostí, tudíž jsem nevyužíval žádné funkce, které už v MATLAB existují. To vedlo k tomu, že počet skriptů je 40. Pro lepší manipulaci s jednotlivými programy jsem se rozhodnul vytvořit webové rozhraní, využívající MATLAB serveru, které vychází z diplomové práce pana Ing. Marcela Fajkuse. Zdrojové kódy k aplikacím jsou součástí elektronické přílohy.

Poslední částí mé bakalářské práce jsou studijní materiály, které budou využity při výuce v předmětu Základy číslicových systémů. Studijní materiály byly vytvořeny pro oblasti soustavy, doplňky, čísla v posunutí, čísla v pevné řádové čárce, čísla v plovoucí řádové čárce a formát UTF. Každá oblast je v samostatném materiálu. Součástí je vždy teorie, která odpovídá teoretické části bakalářské práce, převody do formátů, aritmetika a krátký popis mnou vytvořených aplikací, které se věnují danému tématu. Součásti popisu aplikací jsou i příklady k použití. Studijní materiály jsou součástí elektronické přílohy.

Literatura

1. PŘISPĚVATELÉ WIKIPEDIE. *Číselná soustava — Wikipedie: Otevřená encyklopedie* [online]. 2020-12-30 [cit. 2021-04-16]. Dostupné z: `https://cs.wikipedia.org/wiki/%C4%8C%C3%ADseln%C3%A1_soustava`.
2. MDN CONTRIBUTORS. *Endianness - MDN Web Docs Glossary: Definitions of Web-related terms* [online]. 2020-12-30 [cit. 2021-04-16]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Glossary/Endianness`.
3. PŘISPĚVATELÉ WIKIPEDIE. *Desítková soustava— Wikipedie: Otevřená encyklopedie* [online]. 2020-12-15 [cit. 2021-04-16]. Dostupné z: `https://cs.wikipedia.org/wiki/Des%C3%ADtkov%C3%A1_soustava`.
4. WIKIPEDIA CONTRIBUTORS. *Binary number — Wikipedia: The Free Encyclopedia* [online]. 2021-03-04 [cit. 2021-04-16]. Dostupné z: `https://en.wikipedia.org/w/index.php?title=Binary_number`.
5. FOX, Pamela. Hexadecimal numbers | AP CSP (article) | Khan Academy [online]. c2021 [cit. 2021-04-16]. Dostupné z: `https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:digital-information/xcae6f4a7ff015e7d:hexadecimal-numbers/a/hexadecimal-numbers`.
6. TEACH COMPUTER SCIENCE. *Uses of Hexadecimal | Hexadecimal & Character Sets | Computer Science* [online]. c2021 [cit. 2021-04-16]. Dostupné z: `https://teachcomputerscience.com/uses-of-hexadecimal/`.
7. WIKIPEDIA CONTRIBUTORS. *Signed number representations — Wikipedia: The Free Encyclopedia* [online]. 2021-03-17 [cit. 2021-04-18]. Dostupné z: `https://en.wikipedia.org/wiki/Signed_number_representations`.
8. CHMELÍKOVÁ, Zdeňka; ZDRÁLEK, Jaroslav. *Číslicové systémy I pro integrovanou výuku VUT a VŠB-TUO*. Vysoká škola báňská - Technická univerzita, 2014. ISBN 978-80-248-3649-2.
9. JAVATPOINT. *1's Complement in Digital Electronics - Javatpoint* [online]. c2011-2018 [cit. 2021-04-18]. Dostupné z: `https://www.javatpoint.com/1s-complement-in-digital-electronics`.

10. WIKIPEDIA CONTRIBUTORS. *Ones' complement — Wikipedia: The Free Encyclopedia* [online]. 2021-02-01 [cit. 2021-04-18]. Dostupné z: https://en.wikipedia.org/wiki/Ones%27_complement.

11. YADAV, Chandu. *One's Complement* [online]. 2019-02-21 [cit. 2021-04-18]. Dostupné z: https://www.tutorialspoint.com/one-s-complement.

12. JAVATPOINT. *2's Complement in Digital Electronics - Javatpoint* [online]. c2011-2018 [cit. 2021-04-18]. Dostupné z: https://www.javatpoint.com/2s-complement-in-digital-electronics.

13. WIKIPEDIA CONTRIBUTORS. *Offset binary — Wikipedia: The Free Encyclopedia* [online]. 2020-12-21 [cit. 2021-04-18]. Dostupné z: https://en.wikipedia.org/wiki/Offset_binary.

14. TIŠNOVSKÝ, PAVEL. *Fixed point arithmetic - Root.cz* [online]. 2006-05-24 [cit. 2021-04-18]. Dostupné z: https://www.root.cz/clanky/fixed-point-arithmetic/.

15. SO, Hayden. *Introduction to Fixed Point Number Representation* [online]. 2006-06-28 [cit. 2021-04-18]. Dostupné z: https://inst.eecs.berkeley.edu/~cs61c/sp06/handout/fixedpt.html.

16. WIKIPEDIA CONTRIBUTORS. *IEEE 754 — Wikipedia: The Free Encyclopedia* [online]. 2021-04-05 [cit. 2021-04-18]. Dostupné z: https://en.wikipedia.org/wiki/IEEE_754.

17. BESSELING, Johan; RENSTRÖM, Anders. *A comparative study of IEEE 754 32-bit Float and Posit 32-bit floating point format on precision* [online]. 2020 [cit. 2021-04-18]. Dostupné z: https://www.diva-portal.org/smash/get/diva2:1463840/FULLTEXT01.pdf.

18. PARTHASARATHI, Ranjani. Computer Architecture. In: [https://www.cs.umd.edu/~meesh/411/CA-online/chapter/computer-architectureintroduction/index.html]. INFLIBNET Centre, 2018-07-24, chap. 7.

19. UNICODE. *FAQ - UTF-8, UTF-16, UTF-32 & BOM* [online]. 2020-01-24 [cit. 2021-04-18]. Dostupné z: https://unicode.org/faq/utf_bom.html.

20. WIKIPEDIA CONTRIBUTORS. *Unicode — Wikipedia: The Free Encyclopedia* [online]. 2021-04 [cit. 2021-04-18]. Dostupné z: https://en.wikipedia.org/wiki/Unicode.

21. TECHTERMS. *UTF (Unicode Transformation Format) Definition* [online]. 2012-04-20 [cit. 2021-04-18]. Dostupné z: https://techterms.com/definition/utf.

22. WIKIPEDIA CONTRIBUTORS. *UTF-16 — Wikipedia: The Free Encyclopedia* [online]. 2021-04-01 [cit. 2021-04-18]. Dostupné z: https://en.wikipedia.org/wiki/UTF-16.

23. FAJKUS, Marcel. *Návrh MATLAB serveru katedry telekomunikační techniky* [online]. 2011 [cit. 2021-04-20]. Dostupné z: http://hdl.handle.net/10084/87137. Dipl. Vysoká škola báňská - Technická univerzita Ostrava.

24. MATHWORKS. *What Is MATLAB? - MATLAB & Simulink* [online]. c1994-2021 [cit. 2021-04-18]. Dostupné z: https://www.mathworks.com/discovery/what-is-matlab.html.

25. MATHWORKS. *while loop to repeat when condition is true - MATLAB while* [online]. c1994-2021 [cit. 2021-04-18]. Dostupné z: https://www.mathworks.com/help/matlab/ref/while.html.

26. MATHWORKS. *for loop to repeat specified number of times - MATLAB for* [online]. c1994-2021 [cit. 2021-04-18]. Dostupné z: https://www.mathworks.com/help/matlab/ref/for.html?searchHighlight=for&s_tid=srchtitle.

27. DATTA, Dhriti. *Origins of the binary number system* [online]. 2020-12-14 [cit. 2021-04-16]. Dostupné z: https://www.digit.in/features/tech/digit-mag-origins-of-the-binary-number-system-56191.html.

28. PORUBSKÝ, Štefan. *Binary system* [online]. 2007-02-14 [cit. 2021-04-16]. Dostupné z: https://www.cs.cas.cz/portal/AlgoMath/NumberTheory/Arithmetics/NumeralSystems/PositionalNumeralSystems/BinarySystem.htm.

Seznam elektronických příloh

V informačním systému Edison jsou odevzdány přílohy 2021_FOL0141_BP_Příloha.zip. Adresářová struktura přílohy je následující:

sources/	- adresář se zdrojovými kódy
sources/Main.m	- spouštěcí program
sources/Complements	
Sources/Converter	
Sources/Encoders	
Sources/FixedPoint	
Sources/FloatingPoint	
Sources/OffsetBinary	
Sources/Operations	
Complements.pdf	- studijní materiály k dvojkovému doplňku
FP.pdf	- studijní materiály k číslům v pevné řádové čárce
FX.pdf	- studijní materiály k číslům v plovoucí řádové čárce
OB.pdf	- studijní materiály k číslům v posunutí
Systems.pdf	- studijní materiály k soustavám
UTF.pdf	- studijní materiály k formátu UTF

Příloha A

Studijní materiály - Soustavy

Základy číslicových systémů

Soustavy

Datum 22.4.2021
Autor Michael Foltyn, Ing. Marcel Fajkus, Ph.D.,
Kontakt marcel.fajkus@vsb.cz
Předmět Základy číslicových systémů
Studijní materiál vznikl v rámci bakalářské práce studenta Michaela Foltyna.

Číselné soustavy

Existuje spousta způsobů, jak se dá pracovat s čísly. Nejpoužívanější způsob je práce s dekadickými hodnotami. S dekadickou soustavou pracujeme celý život, používáme ji každý den, například k nakupování, měření času, či počítání. V počítačích a číslicových systémech se ale využívá převážně binární příp. hexadecimální soustava.

Endianita

Endianita popisuje, jakým způsobem počítače ukládají čísla do paměti. Čísla, které mají větší hodnotu, než jeden byte musí být rozdělena do menších skupin, označených jako základní jednotky (Atomic Element). Jednou z takových jednotek je například byte, který je nejpoužívanější, ale může to být i slovo, či jiná n-tice. Endianita pak určuje, která hodnota bude uložena v paměti na nejnižší adrese. Základním principem ukládání je Little Endian, který využívá ukládání „Least to most significant“. To znaměná, že první přenášený bude LSB (least significant bit), tedy nejméně podstatný bit a bude na nejnižší adrese. Druhým způsobem ukládání je pak Big Endian. Ten naopak využívá metodu „Most to least significant“, takže první přenášený bude MSB (Most significant bit), tedy nejvíce podstatný bit, a tak bude na nejnižší adrese.

Adresa	a	a+1	a+2	a+3	a+4	a+5	a+6	a+7
Little Endian (byte)	90	78	56	34	12	EF	CD	AB
Big Endian (byte)	AB	CD	EF	12	34	56	78	90
Little Endian (word)	CDAB	12EF	5634	9078				
Big Endian (word)	ABCD	EF12	3456	7890				

Tabulka 1: Ukázka endianity

V tabulce 1 je zobrazeno, jak by se ukládalo 64bitové číslo 0xABCDEF1234567890. V příkladu jsou zvoleny základní jednotky byte a word. Ukládání je zobrazeno pro případ Little Endian i Big Endian. Pokud nebude vysloveno jinak, tak v následujícím textu bude uvažován Big Endian.

Značení

Na začátek je nutno podotknout, že některá čísla, která vypadají stejně mají v různých soustavách, různou hodnotu. V tabulce 2 je zobrazen řetězec 101 ve všech třech soustavách a jeho decimální reprezentace v posledním sloupečku. Je patrné, že při zápisu čísel musíme definovat, v jaké soustavě je číslo zapsáno.

Soustava	Původní hodnota	Decimální hodnota
Decimální	101	101
Binární	101	5
Hexadecimální	101	257

Tabulka 2 Stejné hodnoty v různých soustavách

Při zápisu v desítkové soustavě nepíšeme nic, nebo písmeno D, případně číslo 10 jako dolní index.

$101 = (101)_{10} = (101)_D$

Pro binární soustavu používáme písmeno B, nebo číslo 2 jako dolní index.

$(101)_2 = (101)_B = (5)_D$

Pro Hexadecimální soustavu používáme písmeno H, nebo číslo 16 jako dolní index.

$(101)_{16} = (101)_H = (257)_D$

Nejčastěji využívané je značení písmeny.

Decimální soustava

Decimální soustava, nebo také dekadická, či desítková soustava je soustava, jejíž základem je číslo 10. Čísla se v této soustavě skládají z desíti znaků 0 až 9. Oproti jiným soustavám má výhodu v tom, že nepotřebuje žádné speciální úpravy pro záporná a reálná čísla. Pro záporná čísla použijeme jednoduše znak zvaný mínus „-“ a pro reálná čísla pak desetinnou tečku, nebo desetinnou čárku. Čísla můžeme rozepsat do polynomu a jednotlivé znaky násobit desítkou s určitou váhou a tím „rozebrat“ číslo na jednotlivé znaky. Při sestavování polynomu násobíme číslo základem soustavy umocněným vahou.

$1535{,}23 = 1 * 10^3 + 5 * 10^2 + 3 * 10^1 + 5 * 10^0 + 2 * 10^{-1} + 3 * 10^{-2}$

Využití

Desítková soustava je nejpoužívanější, proto se s ní setkáváme prakticky všude. Používáme ji ve výpočtech, při nákupu, při měření na přístrojích. Možnost využití je prakticky neomezená, takže je pro lidi intuitivní s ní pracovat.

Aritmetické operace

Základní operace, jako je sčítání, odčítání, násobení a dělení jsou všeobecně známé. Jednoduché počty se učíme už v dětství a ve škole se pak naučíme operace, jako je násobení,

dělení, sčítání a odčítání. Samozřejmě je spousta dalších operací, které se s čísly dají dělat. Jako je umocňování, absolutní hodnota a podobně.

Binární soustava

Binární soustava, nebo také dvojková soustava, je soustava jejíž základem je číslo 2. K vyjádření čísel používá dvě hodnoty. Nulu a jedničku. Pokud chceme říct, že číslo, o kterém je řeč je binární, tak ho označíme písmenem B. Čísla psaná v binární soustavě se stejně jako čísla v desítkové soustavě dají zapsat do polynomu. Což pak můžeme využít k převodu na decimální soustavu. Polynom vytvoříme tak, že jednotlivá čísla vynásobíme základem soustavy umocněným váhou.

$(1110.0111)_B = 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} + 1 * 2^{-3} + 1 * 2^{-4}$

Zápis binárních čísel

Protože nosičem informace je jednička, tak v případě, že na první pozici zleva máme nulu, tak jí můžeme jednoduše odstranit. Tento krok můžeme opakovat tak dlouho, dokud první znak zleva bude jednička. Samozřejmě v případě, že potřebujeme, tak si ty nuly na začátek můžeme dopsat. Ale zase je musíme psát zleva. Kdybychom je napsali doprava, tak změníme hodnotu.

$(0011)_B = (11)_B$

Zobrazení reálných čísel

Když pracujeme s binární soustavou, tak se často bavíme o kladných (neznaménkových) celých číslech. Avšak v případě, že chceme dostat například záporné číslo, tak jsou způsoby, jak toho docílit. A to například pomocí doplňku, přímým kódem, nebo kódem s posunutou nulou. Desetinná čísla pak můžeme vyjádřit pomocí formátu s pevnou, nebo plovoucí řádovou čárkou.

Využití

Binární soustava se používá prakticky ve všech digitálních zařízeních, například v telefonech, počítačích, ale i v digitálních měřících přístrojích. Výhodou binární soustavy je, že pomocí jejich dvou znaků, tedy 1/0 můžeme jednoduše vyjádřit stavy vypnuto a zapnuto, nebo také pravda (true) a lež (false).

Aritmetické operace v binární soustavě

Sčítání

K operaci sčítání dvou čísel nám pomůže jednoduchá tabulka 3. První 3 sloupce této tabulky jsou stejné, jako pravdivostní tabulka operace XOR, takže můžeme říct, že sčítání probíhá logickou operací XOR. Důležité je začít sčítat od nejnižší hodnoty, to znamená, že sčítáme stejně jako v desítkové, a to zprava doleva.

Číslo 1	Číslo 2	Výsledek	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabulka 3 Výsledky sčítání s nosnou

Poslední sloupec v tabulce 3 je velmi podstatný pro sčítání, říká nám totiž, že hodnota součtu přešla do vyššího řádu, protože 1 + 1 = 2, ale binární soustava nezná dvojku, tuto hodnotu pak použijeme v dalším sčítaném znaku. V případě, že nastane situace, kdy první i druhé číslo při součtu je 1 a z minulého kroku se nám také přenesla jednička, tak se řídíme pravidlem, které platí i pro součet více než dvou čísel. Když máme sudý počet jedniček, tak je výsledné číslo 0, naopak, když máme lichý počet jedniček, tak je výsledné číslo 1.

Odčítání

Pro operaci odčítání se nejčastěji používá dvojkový doplněk

Násobení

Násobení stejně jako sčítání probíhá dost podobně, jako v desítkové soustavě. Jediný rozdíl je v tom, že používáme jen číslice 0 a 1, ale stejně jako v normálním násobení, když násobíme nulou, tak dostaneme 0.

Číslo 1	Číslo 2	Výsledek
0	0	0
0	1	0
1	0	0
1	1	1

Tabulka 4 Výsledky násobení

Jak můžeme z tabulky 4 vidět, tak nenulový výsledek dostaneme pouze v případě, že obě čísla jsou jedna. Takže se dá říct, že když máme jedničku, tak násobené číslo zopakujeme, a když máme nulu, tak nám to vyjde nulové. Hlavně nesmíme zapomenout, že když násobíme dalším číslem, tak se u výsledku posuneme o jednu pozici doleva. Jakmile máme číslo vynásobeno všemi čísly druhého čísla, tak provádíme součet. Na základě tabulky můžeme říct, že násobení probíhá logickou operací AND.

Hexadecimální soustava

Hexadecimální soustava, nebo také šestnáctková soustava je soustava o základu 16. K vyjádření používá 16 znaků. V první řadě používá znaky 0 až 9, pro hodnoty 10 až 15 používá prvních 6 písmen abecedy, tedy písmena A až F. Je nutno podotknout, že v hexadecimální soustavě se začíná nulou, takže první hodnota je 0 a poslední hodnota je 15 ne 16.

Hodnota dekadicky	Znak hexadecimálně	Hodnota dekadicky	Znak hexadecimálně
0	0	8	8
1	1	9	9
2	2	10	A
3	3	11	B
4	4	12	C
5	5	13	D
6	6	14	E
7	7	15	F

Tabulka 5 Znaky v hexadecimální soustavě

Zápis hexadecimálních čísel

Jak můžeme vidět z tabulky 5 tak k číselným hodnotám nám přibyly znaky z abecedy. Jelikož hexadecimální soustava není case sensitive, tedy nezáleží, jestli jsou písmena malá, nebo velká, tak nezáleží, jestli napíšeme ff, nebo FF. Obojí má stejný význam. Takže jde čistě o preferenci, jaký zápis, komu vyhovuje.

Stejně jako v případě binární soustavy můžeme umazávat, nebo přidávat nuly zleva, dokud nenarazíme na nenulový znak. Opět platí, že pokud umažeme nulu zprava, tak můžeme drasticky změnit hodnotu našeho čísla.

Příklad zkrácení zápisu hexadecimálního čísla:

$(00FACE0)_H = (FACE0)_H$

Využití

Jelikož hexadecimální zápis dokáže dobře zkrátit zápis binárních hodnot, tak se využívá k definování místa v paměti. Velmi často se také používá, když pracujeme s barvami v grafických editorech, nebo třeba na webových stránkách. V počítačových sítích se používá například v IPv6 adresách anebo v MAC adresách zařízení.

Aritmetické operace v hexadecimální soustavě

Součet v hexadecimální soustavě probíhá podobně jako v desítkové soustavě, je tam však ten rozdíl, že když v součtu překročíme číslo 10, tak nezačínáme znova od nuly, ale pokračujeme až do písmena F, tedy do hodnoty 15. V případě, že překročíme hodnotu 15, tak dochází k přenosu do vyššího řádu a je nutné od čísla odečíst základ soustavy. V tomto případě tedy číslo 16.

Odčítání

Odčítání probíhá obdobně jako sčítání, avšak když se dostaneme pod nulu, tak přičteme hodnotu 16 a v tomto případě se carry bude samozřejmě rovná jedné. Ale je tady jedno velké

ale. V případě, že odčítáme od menšího čísla, číslo větší, tak nám vyjde nesmyslný výsledek, a to z toho důvodu, že hexadecimální soustava bez jakýchkoli úprav nezná záporná čísla. Jak pracovat se zápornými čísly bude v další části mé práce.

Násobení

Násobení hexadecimálních čísel už je trošku složitější operace, protože carry může být mnohem vyšší, než 1, protože například F * F, neboli 15 * 15 je rovno 225. Od této hodnoty musíme odečítat číslo 16 tak dlouho, dokud nedostaneme nějakou hodnotu nižší než 16. Po každém jednom odečtení se nám carry zvedne o jedničku.

Převody mezi soustavami

Převod z decimální do binární soustavy

Převod do binární soustavy probíhá tak, že decimální hodnotu dělíme dvěma se zbytkem. Zbytek si zapíšeme do výsledku a hodnotu po dělení využijeme znova stejným způsobem. Dělení provádíme tak dlouho, dokud nám nezbude číslo 1. V programovacích jazycích se pro zbytek po celočíselném dělení používá operace modulo, pro kterou se často používá znak „%“. V posledním kroku je potřeba celý výsledek otočit, tedy psát ho zleva doprava. Alternativou může být, že si výsledek budeme rovnou psát od konce. Důležitá věc je, že když dělíme číslem 2, tak vždy využíváme zaokrouhlování dolů (floor).

Převod z decimální do hexadecimální soustavy

Převod probíhá prakticky stejně, jako převod do binární soustavy, ale v tomto případě využijeme modulo 16, tedy zbytek po celočíselném dělení číslem 16. Na konci je opět potřeba výsledek otočit, nebo ho hned od začátku začít psát od konce.

Na závěr je dobré podotknout, že operace modulo můžeme použít pro převod do jakékoliv soustavy, postup bude vždy stejný, jen se změní číslo, podle základu dané soustavy.

Převod z binární do decimální soustavy

Převod probíhá tak, že si binární číslo rozložíme do polynomu. Tedy vezmeme 1 znak a vynásobíme ho základem umocněným váhou. Váha n-tého bitu zprava je n-1. S každým dalším znakem se pak váha sníží o jedničku. Poté už jen všechna čísla sečteme a máme výsledek v dekadické soustavě.

Převod z binární do hexadecimální soustavy

Pro vytvoření jednoho hexadecimálního čísla potřebujeme celkem 4 bity. To znamená, že si číslo v binární soustavě musíme rozdělit do čtveřic. V našem případě se nejnižší hodnoty binární soustavy nacházejí úplně vpravo, takže je potřeba začít vytvářet čtveřice zprava. V případě, že nedokážeme vytvořit poslední čtveřici, protože nám zbývá méně bitů než 4 tak doplníme nuly zleva. Čtveřice si pak převedeme do dekadické soustavy a přiřadíme znak odpovídající hexadecimální soustavě.

Převod z hexadecimální do decimální soustavy

Postup je prakticky stejný, jako v případě převodu do binární soustavy. Vytvoříme polynom, tedy vezmeme hodnotu hexadecimální soustavy a vynásobíme ho základem soustavy umocněným váhou. Základ soustavy je v tomto případě 16. V případě, že nemáme hodnotu, ale znak, například A, tak ho převedeme na hodnotu podle tabulky 5.

Převod z hexadecimální do binární soustavy

Převod do binární soustavy probíhá tak, že si jednotlivé znaky převedeme do dekadické soustavy a z dekadické soustavy pak můžeme využít převod do binární soustavy. Poté, co převedeme všechny znaky, tak výsledky převodů spojíme do jednoho výsledku.

Řešené příklady

Příklad 1) Převod z decimální soustavy do binární soustavy. Převeďte číslo $(13)_D$ do binární soustavy.

Převod probíhá tak, že číslo dělíme 2 a zbytek si připisujeme do výsledku. Dělíme tak dlouho, dokud nám nezbyde 0. Průběžné výsledky vždy zaokrouhlujeme dolů.

13/2 = 6	Zbytek = 1	Výsledek = 1
6/2 = 3	Zbytek = 0	Výsledek = 10
3/2 = 1	Zbytek = 1	Výsledek = 101
1/2 = 0	Zbytek = 1	Výsledek = 1011

Finální výsledek je pak třeba otočit. Alternativou by bylo, že bychom výsledek rovnou psali od konce.

Po otočení: 1101

Výsledek $(1101)_B$

Příklad 2) Převod z decimální soustavy do hexadecimální soustavy. Převeďte číslo $(263)_D$ do hexadecimální soustavy.

Převod probíhá tak, že číslo dělíme 16 a zbytek připisujeme do výsledku. Dělíme tak dlouho, dokud nám nezbude 0. Průběžné výsledky vždy zaokrouhlujeme dolů.

268/16 = 16	Zbytek = 12	Výsledek = C
16/16 = 1	Zbytek = 0	Výsledek = C0
1/16 = 0	Zbytek = 1	Výsledek = C01

Jak můžeme vidět, tak v prvním kroku máme zbytek 12, to je v binární soustavě písmeno C. Stejně jako u binární soustavy musíme výsledek nakonec otočit, nebo ho rovnou začít psát od konce.

Po otočení: 10C

Výsledek $(10C)_H$

Příklad 3) Převod z binární soustavy do decimální soustavy. Převeďte $(101011)_B$ do decimální soustavy.

Jak můžeme vidět, tak naše číslo má 6 bitů (n = 6). To znamená, že MSB, tedy nejvýznamnější bit (úplně nalevo) bude mít váhu n-1. Každý další pak bude o jedničku nižší. Převod uděláme pomocí polynomu. Tedy vynásobíme číslo základem soustavy umocněným váhou. Jednotlivé složky polynomu pak sečteme.

Polynom: $1 * 2^{n-1} + 0 * 2^{n-2} + 1 * 2^{n-3} + 0 * 2^{n-4} + 2 * 2^{n-5} + 2 * 2^{n-6}$

$1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 2 * 2^1 + 2 * 2^0 = 32 + 0 + 8 + 0 + 2 + 1 = 43$

Výsledek $(43)_D$

Příklad 4) Převod z binární soustavy do hexadecimální soustavy. Převeďte $(11111010100001)_B$ do hexadecimální soustavy.

Prvním krokem je rozdělení si binárního čísla do čtveřic. Máme dvě možnosti, jak to udělat, první je, že čtveřice začneme tvořit od konce a poté převedeme do dekadické soustavy a přiřadíme znak odpovídající hexadecimální soustavě. Nebo si spočítáme počet bitů a přidáme doleva nuly tak, aby počet byl dělitelný 4 beze zbytku.

Jak můžeme vidět, tak číslo má 14 bitů. To znamená, že doleva přidáme 2 nulové bity.

11111010100001 = 0011111010100001

Poté rozdělíme do čtveřic

0011|1110|1010|0001

Jednotlivé čtveřice převedeme do dekadické soustavy a poté přiřadíme znak odpovídající hexadecimální soustavě.

$(0011)_B = (3)_D = (3)_H$
$(1110)_B = (14)_D = (E)_H$
$(1010)_B = (10)_D = (A)_H$
$(0001)_B = (1)_D = (1)_H$

Jakmile máme převedeno, tak je sjednotíme dohromady tak, aby hexadecimální číslo odpovídalo dané čtveřici binárního čísla

Výsledek $(3EA1)_H$

Příklad 5) Převod z hexadecimální soustavy do decimální soustavy. Převeďte $(F1A9)_H$ do decimální soustavy.

Jak můžeme vidět, tak naše číslo má 4 znaky. MSB, tedy nejvíce podstatný znak (úplně nalevo) má váhu n-1. Kde n tentokrát značí počet znaků. Převod uděláme pomocí polynomu, kde jednotlivé znaky převedeme podle tabulky 5 do dekadické soustavy a vynásobíme základem soustavy (16) umocněným váhou.

$(F * 16^{n-1}) + 1 * (16^{n-2}) + A * (16^{n-3}) + 9 * (16^{n-4})$

Z tabulky víme, že písmenu F odpovídá hodnota 15 a písmenu A odpovídá hodnota 10. Tak si dosadíme hodnoty a sečteme.

$(15 * 16^3) + (1 * 16^2) + (10 * 16^1) + (9 * 16^0) = 61440 + 256 + 160 + 9 = (61865)_D$

Výsledek $(61865)_D$

Příklad 6) Převod z hexadecimální soustavy do binární soustavy. Převeďte $(AE3)_H$ do binární soustavy.

Převod probíhá tak, že si podle tabulky 5 vyjádříme jednotlivé hodnoty v dekadické soustavě. Z dekadické soustavy pak uděláme převod do binární soustavy.

A = $(10)_D$ = $(1010)_B$
B = $(14)_D$ = $(1110)_B$
3 = $(3)_D$ = $(11)_B$

Pro převedení binární hodnoty je potřeba mít z každého hexadecimálního znaku 4 bity, jak můžeme vidět, tak číslo 3 má pouze 2 bity, to znamená, že musíme na začátek doplnit 2 nuly.

$(11)_B$ = $(0011)_B$

Nakonec jen čtveřice zřetězíme tak, aby každá čtveřice odpovídala pořadově danému znaku.

Výsledek: 1010 + 1110 + 0011

Znak + je v tomto případě znak pro zřetězení.

Výsledek je $(101011100011)_B$

Příklad 7) Součet v binární soustavě. Sečtěte následující čísla A = 1101, B = 0101 v binární soustavě.

V případě, že by jedno z čísel bylo delší, než druhé, tak můžeme pro lepší výpočet doplnit na začátek nuly tak, aby byla čísla stejně dlouhá.

Součet vyhodnocujeme podle tabulky 3. V každém kroku nesmíme zapomenout přičíst carry, pokud byl v předešlém kroku nějaký přenos do vyššího řádu.

1. krok
Protože je to první krok, tak ještě nemáme žádné carry. Z tabulky víme, že 1 + 1 = 0 a carry = 1

```
  1101
+ 1101
     0
```

Carry = 1

2. krok
Z minulého kroku už máme carry, to znamená, že ho nesmíme zapomenout přičíst. Z tabulky víme, že 0 + 0 = 0 a 0 + 1 = 1. Takže zapíšeme jedničku. Nedošlo k přenosu do vyššího řádu, takže carry = 0.

```
  1101
  1101
+    1  <- carry
    10
```

Carry = 0

3. krok
Z minulého kroku máme carry = 0. Takže nemusíme nic přičítat a pouze sečteme dvě číslice. Z tabulky víme, že 1 + 1 = 0 a dojde k přenosu do vyššího řádu, tudíž carry = 1.

```
  1101
+ 1101
   010
```

Carry = 1

4. krok
Z minulého kroku máme carry = 1, takže ho nesmíme zapomenout přičíst. Teď nastává situace, kdy sčítáme 3 jedničky. Máme dvě možnosti, jak toto vyřešíme. První možností je postupné sčítání, tedy 1 + 1 = 0 (došlo k přenosu, carry = 1) a potom 0 + 1 = 1. Nebo se můžeme řídit obecným pravidlem, že když sčítáme lichý počet jedniček, tak je výsledek 1.

```
  1101
  1101
+ 1   <- carry
  1010
```

Carry = 1

5. krok
Jak můžeme vidět, už nemáme co sčítat, avšak stále nám zbývá carry = 1. V tomto případě ho akorát dopíšeme na začátek.

Výsledek je $(11010)_B$

Příklad 8) Násobení v binární soustavě. Vynásobte následující čísla A = $(10101010)_B$, B = $(101)_B$ v binární soustavě.

Násobení v binární soustavě je prakticky stejné jako násobení jedničkou a nulou v dekadické soustavě. To znamená, že když máme jedničku, tak násobené číslo zopakujeme a když máme nulu, tak je celý řádek nulový. Hlavně nesmíme s každý číslem posunout o jednu váhu doleva.

1. krok - násobení jedničkou
Násobíme jedničkou, takže celé vrchní číslo zopakujeme.

```
  10101010
×      101
 10101010
```

2. krok – násobení nulou
Při násobení nulou zapíšeme na řádek samé nuly. Hlavně se nesmíme zapomenout o jednu váhu doleva.

```
  10101010
×      101
  10101010
00000000
```

3. krok – násobení jedničkou
Stejně jako v prvním kroku zopakujeme první číslo. Opět se nesmíme zapomenout posunout o jednu váhu doleva.

```
   10101010
×      101
   10101010
  00000000
10101010
```

4. krok součet
Finálním krokem je sečíst jednotlivé mezivýsledky.

```
     10101010
    00000000
+ 10101010
  1101010010
```

Výsledek $(1101010010)_B$

Příklad 9) Sčítání v hexadecimální soustavě. Sečtěte čísla A = $(EF3)_H$, B = $(AF5)_H$ v hexadecimální soustavě.

Součet probíhá prakticky stejně, jako v dekadické soustavě. Akorát používáme i znaky. Ty převedeme do dekadické soustavy a sečteme. V případě, že sečtené číslo překročí hodnotu 15, tak od něj musíme odečíst základ soustavy, tedy číslo 16 a jelikož došlo k přenosu tak carry = 1.

1. krok
V prvním kroku není třeba nic převádět, protože 3 + 5 = 8. Nedošlo k přenosu, takže carry = 0

EF3
+ AF5
8

Carry = 0

2. krok
Z prvního kroku nám zbylo carry = 0, takže nemusíme přičítat. Avšak v tomto kroku sčítáme hodnoty F + F. Hodnota F odpovídá číslu 15, takže sčítáme 15 + 15 = 30. Jelikož číslo překročilo hodnotu 15 musíme odečíst základ soustavy 30 – 16 = 14. Protože číslo překročilo hodnotu 15, tedy došlo k přenosu do vyššího řádu, tak carry = 1. Číslo 14 už odpovídá hodnotě E, tak zapíšeme do mezi výsledku

EF3
+ AF5
E8

Carry = 1

3. krok
Z minulého kroku nám zbylo carry = 1, tak ho nesmíme zapomenou přičíst. V tomto případě sčítáme hodnotu A + E + carry. A odpovídá dekadické hodnotě 10, E odpovídá dekadické hodnotě 14 a carry je 1. Takže 10 + 14 + 1 = 25. Jak vidíme opět došlo k přenosu do vyššího řádu, takže musíme odečíst základ soustavy 25 – 16 = 9 a carry = 1. Číslo 9 zapíšeme do mezivýsledku.

EF3
+ AF5
9E8

Carry = 1

4. krok
Jak můžeme vidět, už nemáme co sečíst, ale z minulého kroku nám zůstalo carry = 1. Tak ho připíšeme na začátek

Výsledek je $(19E8)_H$

Příklady k procvičení

1. Převeďte následující čísla do ostatních soustav

$(10)_D$	Výsledek $(1010)_B$ a $(A)_H$
$(297)_D$	Výsledek $(0001\ 0010\ 1001)_B$ a $(129)_H$
$(35)_D$	Výsledek $(0010\ 0011)_B$ a $(23)_H$
$(81)_D$	Výsledek $(0101\ 0001)_B$ a $(51)_H$
$(255)_D$	Výsledek $(1111\ 1111)_B$ a $(FF)_H$
$(10101111)_B$	Výsledek $(175)_D$ a $(AF)_H$
$(111110111)_B$	Výsledek $(503)_D$ a $(1F7)_H$
$(010111)_B$	Výsledek $(23)_D$ a $(17)_H$
$(10110101)_B$	Výsledek $(181)_D$ a $(B5)_H$
$(11110000)_B$	Výsledek $(240)_D$ a $(F0)_H$
$(ABCD)_H$	Výsledek $(1010\ 1011\ 1100\ 1101)_B$ a $(43981)_D$
$(FACE)_H$	Výsledek $(1111\ 1010\ 1100\ 1110)_B$ a $(64206)_D$
$(EA9)_H$	Výsledek $(1110\ 1010\ 1001)_B$ a $(3753)_D$
$(B199)_H$	Výsledek $(1011\ 0001\ 1001\ 1001)_B$ a $(45465)_D$
$(1234)_H$	Výsledek $(0001\ 0010\ 0011\ 0100)_B$ a $(4660)_D$

2. Sečtěte následující čísla v binární soustavě

A = $(1101)_B$, B = $(1111)_B$	Výsledek $(11100)_B$
A = $(101\ 1010)_B$, B = $(11111)_B$	Výsledek $(1111001)_B$
A = $(111\ 1111)_B$, B = $(1)_B$	Výsledek $(1000\ 0000)_B$
A = $(1110\ 1001)_B$, B = $(1100\ 1111)_B$	Výsledek $(1\ 1011\ 1000)_B$
A = $(11\ 0111)_B$, B = $(11\ 0011)_B$	Výsledek $(110\ 1010)_B$

3. Vynásobte následující čísla v binární soustavě

A = $(1101)_B$, B = $(11)_B$	Výsledek $(100111)_B$
A = $(101\ 1010)_B$, B = $(1001)_B$	Výsledek $(11\ 0010\ 1010)_B$
A = $(111\ 1111)_B$, B = $(10)_B$	Výsledek $(1111\ 1110)_B$
A = $(1110\ 1001)_B$, B = $(111)_B$	Výsledek $(110\ 0101\ 1111)_B$
A = $(11\ 0111)_B$, B = $(1100)_B$	Výsledek $(10\ 1001\ 0100)_B$

4. Sečtěte následující čísla v Hexadecimální soustavě

A = $(AB)_H$, B = $(BA)_H$	Výsledek $(165)_H$
A = $(123)_H$, B = $(FE)_H$	Výsledek $(221)_H$
A = $(AE14)_H$, B = $(ABCD)_H$	Výsledek $(159E1)_H$
A = $(15FE)_H$, B = $(2)_H$	Výsledek $(1600)_H$
A = $(FE80)_H$, B = $(C053)_H$	Výsledek $(1BED3)_H$

Návod k aplikaci

Aplikace pro převod z dekadické do binární soustavy

Aplikace se nazývá AO_DecToBin a má jeden vstup. Vstupem je kladná dekadická hodnota.

Příklady využití aplikace

```
AO_DecToBin(9999)
```

Aplikace pro převod z dekadické do hexadecimální soustavy

Aplikace se nazývá AO_DecToHex a má jeden vstup. Vstupem je kladná dekadická hodnota.

Příklady využití aplikace

```
AO_DecToHex(9999)
```

Aplikace pro převod z binární do dekadické soustavy

Aplikace se nazývá AO_BinToDec a má jeden vstup. Vstupem je binární hodnota zapsaná v jednoduchých uvozovkách. Mezi hodnotami se nepíšou mezery.

Příklady využití aplikace

```
AO_BinToDec('1010')
AO_BinToDec('101011')
```

Aplikace pro převod z binární do hexadecimální soustavy

Aplikace se nazývá AO_BinToHex a má jeden vstup. Vstupem je binární hodnota zapsaná v jednoduchých uvozovkách. Mezi hodnotami se nepíšou mezery.

Příklady využití aplikace

```
AO_BinToHex('101010')
AO_BinToHex('1010')
```

Aplikace pro převod z hexadecimální do dekadické soustavy

Aplikace se nazývá AO_HexToDec a má jeden vstup. Vstupem je hexadecimální hodnota zapsaná v jednoduchých uvozovkách. Mezi hodnotami se nepíšou mezery.

Příklady využití aplikace

```
AO_HexToDec('D00FACE')
AO_HexToDec('ABCD')
```

Aplikace pro převod z hexadecimální do binární soustavy

Aplikace se nazývá AO_HexToBin a má jeden vstup. Vstupem je hexadecimální hodnota zapsaná v jednoduchých uvozovkách. Mezi hodnotami se nepíšou mezery.

Příklady využití aplikace

```
AO_HexToBin('D00FACE')
AO_HexToBin('ABCD')
```

Aplikace pro binární součet

Aplikace se nazývá AO_SUM a má 2 vstupy. Vstupy jsou binární hodnoty zapsané v jednoduchých uvozovkách.

Příklady využití aplikace

```
AO_SUM('101010', '1010')
A = '1010';
B = '10101010'
AO_SUM(A, B)
```

Aplikace pro binární násobení

Aplikace se nazývá AO_Multiplication a má 2 vstupy. Vstupy jsou binární hodnoty zapsané v jednoduchých uvozovkách.

Příklady využití aplikace

```
AO_Multiplication('101010', '1010')
A = '1010';
B = '10101010'
AO_Multiplication(A, B)
```

Aplikace pro hexadecimální součet

Aplikace se nazývá AO_HexSum a má 2 vstupy. Vstupy jsou hexadecimální hodnoty zapsané v jednoduchých uvozovkách.

```
AO_HexSum('AB', 'BA')
A = 'ABCE';
B = 'D1998'
AO_HexSum(A, B)
```

Aplikace pro hexadecimální odčítání

Aplikace se nazývá AO_HexMinus a má 2 vstupy. Vstupy jsou hexadecimální hodnoty zapsané v jednoduchých uvozovkách.

```
AO_HexMinus('AB', 'BA')
A = 'ABCE';
B = 'D1998'
AO_HexMinus(A, B)
```

Příloha B

Studijní materiály - Doplňky

Základy číslicových systémů

Doplňky

Datum 19.4.2021
Autor Michael Foltyn, Ing. Marcel Fajkus, Ph.D.
Kontakt marcel.fajkus@vsb.cz
Předmět Základy číslicových systémů
Studijní materiál vznikl v rámci bakalářské práce studenta Michaela Foltyna.

Doplňky

Doplňky jsou velmi důležitá součást číslicových systémů. Jejich hlavní využití spočívá ve vyjadřování záporných čísel v binární soustavě.

Jedničkový doplněk

Jedničkový doplněk (neboli inverzní kód) lze vyjádřit logickou unární operací NOT neboli negací. V programovacích jazycích se pro negaci často používá znak „~“ (tilda). Negace binárního čísla je operace, která zaměňuje jedničky za nuly a naopak.

Záporná čísla tedy vyjádříme pomocí jedničkového doplňku tak, že všechny bity v binární reprezentaci znegujeme. Znaménko je pak vyjádřeno pomocí znaménkového bitu (sign), který se nachází na pozici nejvýznamnějšího bit MSB (Most Significant Bit). V případě, že MSB binárního čísla je rovno 1, pak se jedná o číslo záporné v jedničkovém doplňku. V opačném případě se jedná o číslo kladné. Při práci se znaménkovými čísly je nutné dávat pozor na rozsah čísel, se kterými pracujeme.

Jedničkový doplněk můžeme popsat vzorcem:

$^{1}A = \sim A$

kde ^{1}A je označení jedničkového doplňku, A je kladné číslo, pro které je doplněk počítán a ~A je negace čísla A.

Rozsah

Rozsah jedničkového doplňku je menší než rozsah neznaménkových (unsigned) čísel, protože se pro znaménko používá první bit. Rozsah jednotkového doplňku je lze vyjádřit intervalem od $-2^{n-1} - 1$ do $2^{n-1} - 1$, kde n je rozsah v bitech. Pro rozsah $n = 8$ dostaneme dekadický rozsah od -127 do 127.

Využití

Jedničkový doplněk se používá jako hradlo NOT v logických obvodech, jinak už v dnešní době moc využití nemá, má totiž pár vad. Jednou z nich je, že má 2 interpretace nuly. Kladnou a zápornou nulu.

Příklad, rozsah 8 bitů:

$(00000000)_B = +0$

$(11111111)_B = -0$

Dalším problémem je tzv. kruhový přenos. Ten vzniká při operacích sčítání a odčítání. Jde o to, že při práci se zápornými čísly se může stát, že dojde k přenosu jedničky do vyššího řádu. V případě, že se tak stane, je potřeba přičíst k výsledku carry, jinak bude výsledek chybný. Příklad kruhového přenosu můžeme vidět na obrázku níže.

Rozsah: 4 bity

$a = (-1)_D = \sim(0001)_B = (1110)_B$
$b = (-2)_D = \sim(0010)_B = (1101)_B$

$a + b = (-3)_D = (1100)_B$

```
   1110
+  1101
  10011   <-- Došlo k přenosu do vyššího řádu, výsledek je
+     1        špatně, je třeba provést korekci
  1100
```

Obrázek 1: Ukázka kruhového přenosu

Aritmetické operace

Příklad součtu dvou čísel můžeme vidět na obrázku 1. Odčítání probíhá prakticky stejně, jen s tím, že druhé číslo převedeme do jedničkového doplňku a pak provedeme součet. V případě, kdy druhý argument operace odčítání je záporný, tak s tímto argumentem nemusíme nic dělat, protože dvojí negace vrátí původní binární reprezentaci argumentu.

Dvojkový doplněk

Dvojkový doplněk je nejpoužívanější způsob zobrazení záporných čísel v počítačích. Na rozdíl od jedničkového doplňku má pouze jednu interpretaci nuly. Také zde platí, že MSB je znaménkový bit. Stále je však potřeba dávat pozor na rozsah v bitech. Další výhodou dvojkového doplňku je to, že při sčítání a odčítání není třeba dělat žádné korekce.

Dvojkový doplněk můžeme popsat vzorcem:

$^2A = {}^1A + 1 = {\sim}A + 1$

kde ^1A je označení jedničkového doplňku, ^2A je označení dvojkového doplňku, A je kladné číslo, pro které je doplněk tvořen a ~A je negace čísla A. Pro vytvoření dvojkového doplňku tedy použijeme negaci, a k výsledku této operace přičteme jedničku.

Rozsah

Rozsah dekadických čísel ve dvojkovém doplňku je od $-2^{n-1} - 1$ do 2^{n-1}, kde n je rozsah v bitech. Pro rozsah $n = 8$, bude dekadický rozsah od -128 do 127.

Využití

Dvojkový doplněk se používá ve všech počítačových architekturách a programovacích jazycích. Konkrétní využití je také zobrazení reálných čísel ve formátu s pevnou a plovoucí řádovou čárkou.

Aritmetické operace

Jednou z dalších výhod dvojkového doplňku je, že sčítání a odčítání probíhá prakticky stejně jako u neznaménkových čísel. V případě operace odčítání je nutné druhý argument převést do dvojkového doplňku a provést sčítání.

Problémy doplňků

Nevhodným výběrem rozsahu se může jednoduše stát, že po operaci dvou čísel dostaneme hodnotu, která neodpovídá dekadickému rozsahu. To znamená, že dojde k přetečení a výsledek bude chybný. Ke zjištění chybného výsledku existují příznaky (Flag). Ty jsou důležitou součástí procesoru. Na tomto principu je postavená binární sčítačka.

Příznaky:

- **C – Carry** je příznak, který nám říká, že došlo k přenosu do vyššího řádu (C = 1). To znamená, že výsledek překročil bitový rozsah.
- **Z – Zero** je příznak, který se nastaví (Z = 1) v případě, že výsledek bude nulový.
- **N - Negative** je příznak, který se nastaví (N = 1) v případě, že MSB = 1 a výsledek je chápán jako dvojkový doplněk. Příznak N je tedy znaménkový bit.

- **V - Overflow** je příznak, který se nastaví (V = 1) v případě, že došlo k přetečení. To nastane v případě, že není možné zobrazit výsledek ve dvojkovém doplňku. Stane se tak proto, že binární sčítačka má omezený počet bitů.

Chyby, které mohou nastat se dají popsat vztahy:

Sčítání: (+A) + (+B) = -C nebo (-A) + (-B) = +C
Odčítání: (+A) - (-B) = -C nebo (-A) - (+B) = +C

Řešené příklady

Příklad 1) Převod do jedničkového doplňku. Převeďte číslo (-25) do jedničkového doplňku. Při výpočtu použijte rozsah 8 bitů.

a. Převod čísla 25 do binární soustavy
   Prvním krokem je převést kladné číslo do binární soustavy.
   $(25)_D = (11001)_B$

   V případě že chceme zapsat výsledek v předepsaném počtu bitů, pak binární posloupnost doplníme nulami z levé strany tak, aby počet bitů byl roven 8.

   $(11001)_B = (00011001)_B$

b. Převod do jedničkového doplňku
   Teď máme pouze převedené kladné číslo do binární soustavy. Vytvoření jedničkového doplňku probíhá tak, že jedničky vyměníme za nuly a naopak.

   Pro výpočet použijeme vztah:
   $^1A = \sim A$
   $^1A = \sim(00011001)_B = {}^1(11100110)$

   Výsledek $^1(11100110)$

   Poznámka: Do jedničkového doplňku se převádějí pouze záporná čísla.

Příklad 2) Převod z jedničkového doplňku. Převeďte číslo $^1(11110010)$ do dekadické soustavy. Při výpočtu použijte rozsah 8 bitů.

a. Kontrola, že číslo je záporné
   Jak můžeme vidět, číslo má 8 bitů a rozsah je také 8 bitů, není tedy třeba doplňovat nuly. MSB je 1, to nám napovídá, že číslo je záporné. Provedeme tedy negaci a zapamatujeme si, že číslo bylo záporné. V případě, že MSB by byl 0, tak je číslo kladné a negaci neprovádíme.

   Pro převod použijeme vzorec:
   $A = \sim{}^1A$
   $A = \sim{}^1(11110010) = (00001101)_B$

b. Převod do dekadické soustavy
   Z předchozího kroku víme, že číslo je záporné, to znamená, že binární číslo převedeme do dekadické soustavy a přidáme mu znaménko mínus.

   $-(00001101)_B = (-13)_D$

   Výsledek je $(-13)_D$

Příklad 3) Převod do dvojkového Doplňku. Převeďte číslo $(-39)_D$ do dvojkového doplňku. Při výpočtu použijte rozsah 8 bitů.

a. Převod čísla 39 do binární soustavy
   Prvním krokem je převést kladné číslo do binární soustavy

   $(39)_D = (100111)_B$

   Protože je rozsah 8 bitů a naše číslo má pouze 6 bitů, tak doplňujeme nuly zleva (přidáme 2 nuly).

   $(100111)_B = (00100111)_B$

b. Převod do dvojkového doplňku
   Pro vytvoření dvojkového dopňku provedeme negaci a přičteme hodnotu 1.

   Pro výpočet použijeme vztah:
   $^2A = \sim A + 1$
   $^2A = \sim(00100111) + 1 = (11011000) + 1 = ^2(11011001)$

   Výsledek je $^2(11011001)$

Příklad 4) Převod čísla z dvojkového doplňku. Převeďte číslo $^2(10101010)$ do dekadické soustavy. Při výpočtu použijte rozsah 8 bitů.

a. Kontrola, že číslo je záporné
   Jak můžeme vidět, číslo má 8 bitů a rozsah je také 8 bitů, není tedy třeba doplňovat nuly. MSB je 1, to nám napovídá, že číslo je záporné. Provedeme tedy negaci, přičteme jedničku a zapamatujeme si, že číslo bylo záporné. V případě, že MSB by bylo 0, tak je číslo kladné a převod neprovádíme.

   Pro převod použijeme vztah:

   $A = \sim^2A + 1$
   $A = \sim(10101010) + 1 = (01010101) + 1 = (01010110)_B$

b. Převod z dvojkového doplňku
   Z předchozího kroku víme, že číslo je záporné, to znamená, že binární číslo převedeme do dekadické soustavy a přidáme mu znaménko mínus.

   $A = -(01010110)_B = (-86)_D$

   Výsledek je $(-86)_D$

Příklad 5) Binární sčítačka. Sečtěte číslo $A = (96)_D$ a $B = (48)_D$. Při výpočtu použijte rozsah 8 bitů, a vyznačte příznaky C, Z, N, V. Použijte sčítačku pro znaménková čísla.

   V našem případě žádné z čísel není záporné, takže čísla převedeme do binární soustavy, sečteme a vyznačíme příznaky

a. Převod čísel do binární soustavy

$A = (96)_D = (01100000)_B$
$B = (48)_D = (00110000)_B$

b. Binární součet
Čísla sečteme klasickým způsobem.

```
  01100000
+ 00110000
  10010000
```

c. Vyznačení příznaků
Jak můžeme vidět ve výsledku nedošlo k přenosu do vyššího řádu, tudíž C = 0. Číslo netvoří samé nuly, takže Z = 0. Znaménkový bit (MSB) je jedna, to značí, že číslo je záporné, protože používáme znaménkovou sčítačku, tak N = 1, avšak tím, že jsme sčítali kladná čísla, tak víme, že nám nemůže vyjít záporný výsledek, tudíž víme, že došlo k přetečení, takže V = 1. To znamená, že číslo nelze vyjádřit ve dvojkovém doplňku.

Výsledek je $(1001000)_B$. C = 0, Z = 0, N = 1, V = 1

Příklad 6) Binární sčítačka. Odečtěte čísla $A = (35)_D$ a $B = (115)_D$. Při výpočtu použijte rozsah 8 bitů, a vyznačte příznaky C, Z, N, V. Použijte sčítačku pro znaménková čísla.

V tomto případě není ani jedno z čísel záporné, avšak při odčítání se využívá dvojkový doplněk, to znamená, že druhé číslo převedeme dvojkového doplňku, čísla sečteme a vyznačíme příznaky.

a. Převod čísel do binární soustavy
$A = (35)_D = (00100011)_B$
$B = (115)_D = (01110011)_B$

b. Převod čísla B do dvojkového doplňku
$^2B = \sim(01110011) + 1 = (10001100) + 1 = (10001101)$

Poznámka: Kdyby bylo číslo už v zadání záporné, tak nemusíme převádět do dvojkového doplňku, protože když provedeme převod do dvojkového doplňku dvakrát, tak dostaneme výsledek, jako bychom číslo převedli pouze do binární soustavy. Z matematického hlediska je to logické, protože dvakrát mínus rovná se plus.

c. Binární součet

```
  00100011
+ 10001101
  10110000
```

d. Vyznačení příznaků
MSB je 1, takže výsledek je záporný a N = 1. Výsledek není posloupnost samých nul, takže Z = 0. Nedošlo k přenosu do vyššího řádu, takže C = 0. Od menšího čísla jsme odčítali číslo větší, tudíž víme, že výsledek má být záporné a dá se zobrazit ve dvojkovém doplňku, takže nedošlo k přetečení a V = 0.

Výsledek je $^{2}(10110000)$. C = 0, Z = 0, N = 1, V = 0

Poznámka: Výsledek je záporný, tudíž nesmíme zapomenout, že není v binární soustavě, ale že je ve dvojkovém doplňku.

Příklady k procvičení

1. Převeďte následující čísla do jedničkového doplňku. Při výpočtu použijte rozsah 8 bitů.

   $(-15)_D$ Výsledek $^1(11110000)$
   $(0)_D$ Výsledek $^1(11111111$ a $00000000)$
   $(-115)_D$ Výsledek $^1(10001100)$
   $(33)_D$ Výsledek $^1(00100001)$
   $(-127)_D$ Výsledek $^1(10000000)$

2. Převeďte následující čísla z jedničkového doplňku. Při výpočtu použijte rozsah 8 bitů.

   $^1(11110101)$ Výsledek $(-10)_D$
   $^1(10101111)$ Výsledek $(-80)_D$
   $^1(11100000)$ Výsledek $(-31)_D$
   $^1(10001111)$ Výsledek $(-112)_D$
   $^1(1000111)$ Výsledek $(71)_D$

3. Převeďte následující čísla do dvojkového doplňku. Při výpočtu použijte rozsah 8 bitů.

   $(-13)_D$ Výsledek $^2(11110011)$
   $(-128)_D$ Výsledek $^2(10000000)$
   $(-101)_D$ Výsledek $^2(10011011)$
   $(49)_D$ Výsledek $^2(110001)$
   $(-125)_D$ Výsledek $^2(10000011)$

4. Převeďte následující čísla z dvojkového doplňku. Při výpočtu použijte rozsah 8 bitů.

   $^2(11101010)$ Výsledek $(-22)_D$
   $^2(11100000)$ Výsledek $(-32)_D$
   $^2(10010001)$ Výsledek $(-111)_D$
   $^2(10101111)$ Výsledek $(-81)_D$
   $^2(1111111)$ Výsledek $(127)_D$

5. Sečtěte následující čísla. Pro výpočet použijte znaménkovou binární sčítačku, rozsah 8 bitů a vyznačte příznaky.

A = 50, B = 111	Výsledek (10100001), N = 1, Z = 0, V = 1, C = 0
A = -35, B = 59	Výsledek (00011000), N = 0, Z = 0, V = 0, C = 1
A = 111, B = -120	Výsledek (11110111), N = 1, Z = 0, V = 0, C = 1
A = 139, B = 117	Výsledek (00000000), N = 0, Z = 1, V = 0, C = 1
A = 153, B = -86	Výsledek (11010100), N = 1, Z = 0, V = 0, C = 0

6. Odečtěte následující čísla. Pro výpočet použijte znaménkovou binární sčítačku, rozsah 8 bitů a vyznačte příznaky.

A = 128, B = -153	Výsledek (00011001), N = 0, Z = 0, V = 0, C = 1
A = 55, B = 98	Výsledek (11010101), N = 1, Z = 0, V = 0, C = 0
A = -13, B = -141	Výsledek (10000000), N = 1, Z = 0, V = 1, C = 1
A = 68, B = -117	Výsledek (10111001), N = 1, Z = 0, V = 1, C = 0
A = 63, B = 15	Výsledek (11010100), N = 0, Z = 0, V = 0, C = 1

Návod k aplikaci

Aplikace pro převod do jedničkového doplňku

Aplikace se nazývá AO_DecToOnesComplement a má 2 vstupy. První vstup je dekadická hodnota, může být kladná i záporná, v případě kladné hodnot aplikace převede hodnotu do binární soustavy, v případě záporné hodnoty aplikace převede hodnotu do jedničkového doplňku. Druhý vstup je bitový rozsah.

Příklady využití aplikace:

```
vysledek = AO_DecToOnesComplement(-53, 8)
vysledek = AO_DecToOnesComplement(51, 8)

hodnota = '10101010';
hodnota_v_dec = AO_BinToDec(hodnota);
vysledek = AO_DecToOnesComplement(hodnota_v_dec, 8)
vysledek = AO_DecToOnesComplement(AO_BinToDec('10101010'), 8)
```

Aplikace pro převod z jedničkového doplňku

Aplikace se nazývá AO_OnesComplementToDec a má 2 vstupy. Prvním vstupem je hodnota v binární soustavě. Hodnota musí být zapsána jako string v jednoduchých uvozovkách, při použití dvojitých uvozovek program spadne. Druhý vstup je bitový rozsah.

Příklady využití aplikace:

```
bin = '10101010';
AO_OnesComplementToDec(bin, 8)

AO_OnesComplementToDec('10101010', 8)
```

Aplikace pro převod do dvojkového doplňku

Aplikace se nazývá AO_DecToTwosComplement a má 2 vstupy. První vstup je dekadická hodnota, může být kladná i záporná, v případě kladné hodnot aplikace převede hodnotu do binární soustavy, v případě záporné hodnoty aplikace převede hodnotu do dvojkového doplňku. Druhý vstup je bitový rozsah.

Příklady využití aplikace:

```
vysledek = AO_ DecToTwosComplement(-53, 8)
      vysledek = AO_ DecToTwosComplement(51, 8)

      hodnota = '10101010';
      hodnota_v_dec = AO_BinToDec(hodnota);
      vysledek = AO_ DecToTwosComplement(hodnota_v_dec, 8)

      vysledek = AO_ DecToTwosComplement(AO_BinToDec('10101010'), 8)
```

Aplikace pro převod z dvojkového doplňku

Aplikace se nazývá AO_TwosComplementToDec a má 2 vstupy. Prvním vstupem je hodnota v binární soustavě. Hodnota musí být zapsána jako string v jednoduchých uvozovkách, při použití dvojitých uvozovek program spadne. Druhý vstup je bitový rozsah.

Příklady využití aplikace:

```
bin = '10101010';
AO_ TwosComplementToDec(bin, 8)

AO_ TwosComplementToDec('10101010', 8)
```

Binární sčítačka

Aplikace se nazývá AO_BinaryCounterSum má 3 vstupy. První dva vstupy jsou pro čísla, která chceme sečíst v binární soustavě. Hodnota musí být zapsána jako string v jednoduchých uvozovkách, při použití dvojitých uvozovek program spadne. V případě, že chceme jako vstup použít dekadickou hodnotu, můžeme využít funkci DecToTwosComplement, která nám záporná čísla převede do dvojkového doplňku a kladná čísla do binární soustavy. Třetí vstup je bitový rozsah.

Aplikace má celkem 5 výstupních hodnot, první hodnota je pro výsledek, další jsou pro příznaky v pořadí N, Z, V, C. Pořadí se nesmí zaměnit.

Příklady využití aplikace:

```
[result,N,Z,V,C] = AO_BinaryCounterSum('10101010', '10101010', 8)

A = 63;
B = 15;
A_bin = AO_DecToTwosComplement(A, 8);
B_bin = AO_DecToTwosComplement(B, 8);
[result,N,Z,V,C] = AO_BinaryCounterSum(A_bin, B_bin, 8)

[result,N,Z,V,C] = AO_BinaryCounterSum(AO_DecToTwosComplement(63, 8),
AO_DecToTwosComplement(15, 8), 8)
```

Binární sčítačka – odčítání

Aplikace se nazývá AO_BinaryCounterMinus. Funguje naprosto identicky jako Binární sčítačka zmíněná výše, jen místo sčítání dělá operaci odčítání. Čistě teoreticky jí vůbec není třeba, protože když chceme odčítat, stačí nám v obyčejné binární sčítačce převést druhé číslo do dvojkového doplňku.

Příloha C

Studijní materiály - Čísla v posunutí

Základy číslicových systémů

Čísla v posunutí

Datum 17.4.2021
Autor Michael Foltyn, Ing. Marcel Fajkus, Ph.D.,
Kontakt marcel.fajkus@vsb.cz
Předmět Základy číslicových systémů
Studijní materiál vznikl v rámci bakalářské práce studenta Michaela Foltyna.

Kód s posunutou nulou

Kód s posunutou nulou, či anglicky offset binary (má i další názvy), je další ze způsobů, jak zaznamenávat znaménková čísla. Dekadický rozsah zobrazení je od $-b$ (bias, nebo také offset), neboli posunutí do $2^n - 1 - n$, kde n je rozsah v bitech. Bias může být libovolné číslo, avšak při zobrazování čísel v pohyblivé řádové čárce se používá vztah $b = 2^{n-1} - 1$. Narozdíl od doplňků nepoužívá sign bit.

Využití

Kód s posunutou nulou se používá ke zpracování záporných čísel v A/D (analog to digital) a D/A (digital to analog) převodnících, také ve formátu pohyblivé řádové čárky pro zobrazení exponentu. Často se také používá ve zpracování digitálních signálů.

Převod

Převod probíhá tak, že k číslu, které chceme přičteme bias, výsledné číslo nesmí vyjít záporné. V případě zpětného převodu bias odečteme.

Pro převod do formátu s posunutou nulou můžeme použít vztah:

$^{B}A = A + b$

Pro převod z formátu s posunutou nulou můžeme použít vztah:

$A = {}^{B}A - b$

kde ^{B}A je posunuté číslo. Toto číslo musí být přirozené, tedy $^{B}A \geq 0$, A je kladné, nebo záporné celé číslo, pro které posunutí počítáme a b je bias.

Aritmetika s čísly v posunutí

Sčítání a odčítání probíhá tak, že provedeme operaci mezi dvěma čísly a poté je převedeme do kódu s posunutou nulou, tedy přičteme bias. V případě že pracujeme s čísly, která už jsou ve formátu s posunutou nulou, tak v případě sčítání bias odečteme a v případě odčítání bias přičteme. V žádném z těchto případů však nesmí nastat přetečení.

K operaci sčítání můžeme použít vztah:

$^{B}\text{Součet} = (A + B) + b = {}^{B}A + {}^{B}B - b$

K operaci odčítání můžeme použít vztah:

$^{B}\text{Rozdíl} = (A - B) + b = {}^{B}A - {}^{B}B + b$

kde A, B jsou kladná, nebo záporná celá čísla, pro která posunutí počítáme, b je bias a ^{B}A, ^{B}B jsou přirozená čísla s posunutou nulou.

Řešené příklady

Příklad 1) Převod do formátu s posunutou nulou.Převeďte číslo $(-56)_D$ do formátu s posunutou nulou. Při výpočtu použijte rozsah n = 8 bitů a posunutí $b = 2^{n-1} - 1$.

a. Výpočet bias
   V případě, že máme posunutí zadané vzorcem, tak do vzorce dosadíme bitový rozsah.
   $b = 2^{n-1} - 1 = 2^{8-1} - 1 = 127$

b. Převod do formátu s posunutou nulou
   Pro převod do formátu použijeme vzorec ${}^BA = A + b$. Tedy k dekadickému číslu přičteme posunutí (bias).
   ${}^BA = A + b = -56 + 127 = {}^B(71)$

   Výsledek je ${}^B(71)$ nebo ${}^B(01000111)_B$.

Příklad 2) Převod z formátu s posunutou nulou. Převeďte číslo B(93) z formátu s posunutou nulou. Při výpočtu použijte rozsah n = 8 bitů a posunutí b = 2n-1.

a. Výpočet posunutí (bias)
   V případě, že máme posunutí zadané vzorcem, tak do vzorce dosadíme bitový rozsah.
   $b = 2^{n-1} = 2^{8-1} = 128$

b. Převod z formátu s posunutou nulou
   Pro převod do formátu použijeme vzorec $A = {}^BA - b$. Tedy od čísla v posunutí odečteme posunutí (bias).
   $A = 93 - 128 = (-35)_D$

   Výsledek je $(-35)_D$

Příklad 3) Sčítání ve formátu s posunutou nulou. Sečtěte čísla $A = (13)_D$ a $B = (-26)_D$ ve formátu s posunutou nulou. Při výpočtu použijte rozsah n = 8 bitů a posunutí $b = 2^{n-1} - 1$.

Celkově jsou 2 způsoby, jak můžeme čísla ve formátu s posunutou nulou sečíst. A to podle vzorců ${}^BSoučet = (A + B) + b$ a ${}^BSoučet = {}^BA + {}^BB - b$

a. Výpočet bias
   $b = 2^{n-1} - 1 = 2^{8-1} - 1 = 127$

b. Součet podle vzorce (A + B) + b

1. Prvně sečteme čísla bez posunutí a výsledek poté převedeme do formát s posunutou nulou.
   ^{B}Součet = 13 + (-26) = -13 + 127 = ^{B}(114)
2. Výsledek je ^{B}(114) nebo ^{B}(01110010)$_{B}$

2. Součet podle vzorce (^{B}A + ^{B}B) – b
   1. Prvně čísla převedeme do formátu s posunutou nulou.
      ^{B}A = A + b = 13 + 127 = ^{B}(140)
      ^{B}B = B + b = -26 + 127 = ^{B}(101)
   2. Poté čísla sečteme a odečteme bias
      ^{B}Součet = (140 + 101) - 127 = ^{B}(114)
   3. Výsledek je Výsledek je ^{B}(114) nebo ^{B}(01110010)$_{B}$

Příklad 4) Odčítání ve formátu s posunutou nulou. Odečtěte čísla A = (-23)$_{D}$ a B = (-46)$_{D}$ ve formátu s posunutou nulou. Při výpočtu použijte rozsah n = 8 bitů a posunutí b = 2^{n-1}.

Celkově jsou 2 způsoby, jak můžeme čísla ve formátu s posunutou nulou odčítat. A to podle vzorců ^{B}Rozdíl = (A - B) + b a ^{B}Rozdíl = ^{B}A - ^{B}B + b.

a. Výpočet bias
   b = 2^{n-1} = 2^{8-1} = 128
b. Rozdíl podle vzorce (A - B) + b
   1. Čísla odečteme v dekadické soustavě a poté převedeme do formátu s posunutou nulou.
      ^{B}Rozdíl = -23 - (-46) = 23 + 128 = ^{B}(151)
   2. Výsledek je ^{B}(151) nebo ^{B}(10010111)$_{B}$
c. Rozdíl podle vzorce (^{B}A - ^{B}B) + b
   1. Čísla převedeme do formátu s posunutou nulou
      ^{B}A = A + b = -23 + 128 = ^{B}(105)
      ^{B}B = B + b = -46 + 128 = ^{B}(82)
   2. Poté od sebe čísla odečteme a přičteme bias
      ^{B}Rozdíl = 105 - 82 = 23 + 128 = ^{B}(151)
   3. Výsledek je ^{B}(151) nebo ^{B}(10010111)$_{B}$

Příklady k procvičení

1. Převeďte následující čísla do formátu s posunutou nulou. Při výpočtu použijte rozsah n = 8 bitů a posunutí $b = 2^{n-1} - 1$.

$(120)_D$	Výsledek $^B(247)$
$(-111)_D$	Výsledek $^B(16)$
$(43)_D$	Výsledek $^B(170)$
$(-51)_D$	Výsledek $^B(76)$
$(-1)_D$	Výsledek $^B(126)$

2. Převeďte následující čísla z formátu s posunutou nulou. Při výpočtu použijte rozsah n = 8 bitů a posunutí $b = 2^{n-1}$.

$^B(14)$	Výsledek $(-114)_D$
$^B(0)$	Výsledek $(-128)_D$
$^B(222)$	Výsledek $(94)_D$
$^B(33)$	Výsledek $(-95)_D$
$^B(113)$	Výsledek $(-15)_D$

3. Sečtěte následující dvojice ve formátu s posunutou nulou. Při výpočtu použijte posunutí b = 63.

$A = (-10)_D$, $B = (-23)_D$	Výsledek $^B(30)$
$A = (15)_D$, $B = (-46)_D$	Výsledek $^B(32)$
$A = (10)_D$, $B = (-10)_D$	Výsledek $^B(63)$
$A = (-25)_D$, $B = (31)_D$	Výsledek $^B(69)$
$A = (20)_D$, $B = (69)_D$	Výsledek $^B(152)$

4. Odečtěte následující dvojice ve formátu s posunutou nulou. Při výpočtu použijte posunutí b = 256.

$A = (133)_D$, $B = (11)_D$	Výsledek $^B(378)$
$A = (-120)_D$, $B = (-30)_D$	Výsledek $^B(166)$
$A = (77)_D$, $B = (77)_D$	Výsledek $^B(256)$
$A = (-79)_D$, $B = (133)_D$	Výsledek $^B(44)$
$A = (215)_D$, $B = (111)_D$	Výsledek $^B(160)$

Návod k aplikaci

Aplikace pro převod do formátu s posunutou nulou

Aplikace má název AO_OffsetBinary a má 2 vstupy. Prvním vstupem je dekadické číslo, druhým je posunutí. Oba vstupy jsou v dekadické soustavě.

Příklady využití aplikace

```
vysledek = AO_OffsetBinary(215, 256);
n = 8;
vysledek = AO_OffsetBinary(127, 2^(n-1)-1);
```

Aplikace pro zpětný převod

Aplikace má název AO_OffsetBinaryToDec a má 2 vstupy. Prvním vstupem je číslo ve formátu s posunutou nulou a druhým je posunutí v dekadické soustavě.

Příklady využití aplikace

```
vysledek = AO_OffsetBinaryToDec (215, 256);

n = 8;
vysledek = AO_OffsetBinaryToDec(127, 2^(n-1)-1);
```

Aplikace pro sčítání ve formátu s posunutou nulou

Aplikace má název AO_OffsetBinarySum a má 3 vstupy. První dva vstupy jsou čísla v dekadické soustavě, která chceme sečíst a třetí vstup je posunutí.

Příklady využití aplikace

```
vysledek = AO_OffsetBinarySum (1, -1, 128);

n = 8;
vysledek = AO_OffsetBinarySum(13, -23, 2^(n-1)-1);
```

Aplikace pro odčítání ve formátu s posunutou nulou

Aplikace má název AO_OffsetBinaryMinus a má 3 vstupy. První dva vstupy jsou čísla v dekadické soustavě, která chceme odečíst a třetí vstup je posunutí.

Příklady využití aplikace

```
vysledek = AO_OffsetBinaryMinus (1, -1, 128);

n = 8;
vysledek = AO_OffsetBinaryMinus(13, -23, 2^(n-1)-1);
```

Příloha D

Studijní materiály - Čísla v pevné řádové čárce

Základy číslicových systémů

Čísla v pevné řádové čárce

Datum 22.4.2021
Autor Michael Foltyn, Ing. Marcel Fajkus, Ph.D.,
Kontakt marcel.fajkus@vsb.cz
Předmět Základy číslicových systémů
Studijní materiál vznikl v rámci bakalářské práce studenta Michaela Foltyna.

Formát čísel v pevné řádové čárce

Pevná řádová čárka, anglicky Fixed point (FX), jsou čísla s desetinnou tečkou, která je umístěna v předem definované místě. Z matematického hlediska jsou to racionální čísla, která se vyjadřují jako podíl nebo zlomek. Zlomek pak můžeme zapsat jako reálné číslo krát měřítko (scaling factor). Scalling factor může být celé číslo, nebo zlomek ve tvaru 1/jmenovatel (měřítko).

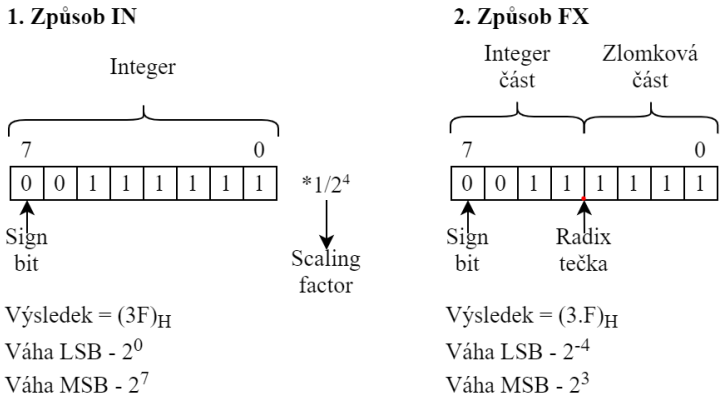
Příklad:

2,38 v měřítku 100/1 je 238

238 v měřítku 1/100 je 2,38

Měřítko se pak vybírá tak, aby čitatel bylo celé číslo, nejčastěji jsou to mocniny čísla 2 nebo 10. V rámci této práce budu využívat binární měřítko $1/2^n$. Fixed point čísla mohou být znaménková i neznaménková, v rámci této práce se budu věnovat pouze znaménkovými. Pro pevnou řádovou čárku je MSB znaménkový bit. Pro záporná čísla používáme dvojkový doplněk. Protože budeme pracovat s binární soustavou, tak se nehodí používat pojem desetinná tečka, nebo desetinná čárka, proto budu používat pojem radix tečka.

Čísla v pevné řádové čárce můžeme chápat dvěma způsoby. Prvním způsobem je integer (IN) číslo neboli celé číslo. Druhým způsobem je zobrazování reálného čísla nebo čísla v pevné řádové čárce (FX). V tomto případě je důležitá pozice radix tečky, která nám určuje celou část a desetinnou část. Pozici radix tečky nám v měřítku $1/2^n$ značí hodnota n. V případě, že budeme používat zobrazení FX, tak se změní i váha LSB, to můžeme vidět na obrázku 1. Pro zobrazení FX je nalevo od radix tečky váha 0 a napravo od radix tečky je váha -1.



Obrázek 1 Příklad zápisu ve formátech IN a FX

Číselný formát ℚ

V matematice je ℚ označení pro racionální čísla. V informatice je to označení fixed point formátu a používají se zápisy jako ℚm.f, ℚf, ℚm.n nebo ℚn, kde písmeno m označuje počet

bitů celočíselné části a písmeno f nebo n počet bitů zlomkové části. Například formát ℚ3.12 má 3 bity pro celou část, 12 bitů pro zlomkovou část a MSB je sign bit. Velikost slova je tedy m+f+1, kde +1 značí sign bit.

Rozsah fixed point čísel je od $(-2^{m+f})/2^f$ do $+(2^{m+f}-1)/2^f$. S přesností $\varepsilon = 1/2^f$. Přesnost zobrazení záleží na vhodném výběru f. V případě, že zlomková část má příliš málo bitů, tak může dojít k zaokrouhlení, v případě, že bude příliš vysoké, tak se může hodnota nepatrně zvýšit.

Využití

Formát fixed point se používá v digitálním zpracování signálů, ta zahrnuje například aplikaci digitálního filtru, digitální zpracování obrazu, převod řeči na text a zpět. Dále se používá pro výpočet komprese JPEG obrázků a obecně v počítačové grafice.

Převod do formátu ℚm.f

Ve formátu ℚm.f písmeno m označuje počet bitů celočíselné části a písmeno f počet bitů zlomkové části. Převod pak probíhá tak, že reálné číslo vynásobíme měřítkem, tedy hodnotou 2^f. Výsledek zaokrouhlíme. V případě, že je číslo záporné, tak formát IN převedeme do dvojkového doplňku a poté převedeme do požadovaného formátu (IN nebo FX). Formát IN značí integer, tedy celé číslo a formát FX je formát s radix tečkou.

Pro lehčí výpočet dvojkového doplňku můžeme využít vzorce:

$$^2A = 2^n - A,$$

kde ^{2}A je číslo ve dvojkovém doplňku, A je číslo, které převádíme na dvojkový doplněk a n je bitový rozsah.

Převod z formátu ℚm.f

Zjistíme hodnotu MSB. Hexadecimální nebo binární hodnotu převedeme na decimální hodnotu. V případě, že je MSB roven jedné, tak číslo převedeme z dvojkového doplňku a přidáme znaménko mínus. Výsledku pak dosáhneme vynásobením měřítkem $1/2^f$.

Aritmetika v pevné řádové čárce

Sčítání a odčítání se provádí pomocí celých čísel, které odpovídají číslu v pevné řádové čárce. Bavíme se tedy o formátu Integer (IN). Pro průběh je potřeba, aby obě čísla byly ve stejném formátu.

Toho dosáhneme pomocí vztahu:

$$\mathbb{Q}m1.f \pm \mathbb{Q}m2.f = \mathbb{Q}\,(max(m1, m2) + 1).f$$

Výsledný formát bude mít o jeden bit v celočíselné části více, než je maximální počet bitů v Integer části obou čísel. V případě záporného čísla nesmíme zapomenout převést na dvojkový doplněk.

Odčítání probíhá úplně stejně, s tím rozdílem, že druhé číslo převedeme do dvojkového doplňku. V případě, že zadané číslo je záporné, tak není třeba převádět do dvojkového doplňku, protože když použijeme doplněk na jedno číslo dvakrát, tak ho převedeme do dvojkového doplňku a zase zpět.

Řešené příklady

Příklad 1) Převod kladného čísla do formátu ℚm.f. Převeďte kladné reálné číslo 1,35 do formátu ℚ2.13. Při výpočtu použijte rozsah 16 bitů.

a. Zjištění parametrů m a f
   Ze zadání si zapíšeme hodnoty m a f.

   m = 2
   f = 13

b. Vynásobení měřítkem a zaokrouhlení
   Dalším krokem je vynásobení měřítkem. Měřítko vytvoříme tak, že základ soustavy umocníme na f. V našem případě bude tedy měřítko 2^{13}. Jakmile máme měřítko, můžeme zadané číslo vynásobit měřítkem. Poté zaokrouhlíme klasickým způsobem k nejbližší hodnotě.

   $1{,}35 * 2^{13} = (11059{,}2)_D$
   Po zaokrouhlení: $(11059)_D$

c. Převod do formátu IN
   Formát IN vytvoříme tak, že zaokrouhlené číslo převedeme do hexadecimální soustavy.

   $(11059)_D = (2B33)_H$

   Výsledek ve formátu IN je $(2B33)_H$

d. Převod do formátu FX
   V případě, že bychom z nějakého důvodu chtěli využít formát s desetinnou tečkou (FX), tak ten uděláme tak, že si zaokrouhlené číslo převedeme do binární soustavy, pro jednodušší práci si číslo v binární soustavě zapíšeme v celém rozsahu. To znamená, že v případě, že výsledné číslo bude mít méně, než 16 bitů, tak doplníme nuly zleva. Poté vyznačíme desetinnou část. V našem případě bude mít desetinná část 13 bitů. MSB je znaménkový bit.

   $(11059)_D = (0010101100110011)_B$

   Vyznačení radix tečky: 001.0101100110011

   Poté je potřeba převést celočíselnou a desetinnou část do hexadecimální soustavy.

   Celočíselná část: $(1)_H$

   Když převádíme desetinnou část do binární soustavy, tak postupujeme naopak než v případě celočíselné části. To znamená, že hexadecimální číslo tvoříme zprava doleva. Pro lepší převod si můžeme doplnit nuly do počtu dělitelného 4. V našem případě tedy 3 nuly doprava.

Doplnění nul: 0.0101100110011000

Převod do hexadecimální soustavy: $(0.0101100110011000)_B = (0.5998)_H$

Spojení celočíselné a desetinné části $(1.5998)_H$

Výsledek ve formát FX je $(1.5998)_H$

Příklad 2) Převod záporného čísla do formátu ℚm.f. Převeďte číslo -1,35 do formátu ℚ2.13. Při výpočtu použijte rozsah 16 bitů.

a. Zjištění parametrů m a f
   Ze zadání si zapíšeme hodnoty m a f.

   m = 2
   f = 13

b. Vynásobení měřítkem a zaokrouhlení
   Dalším krokem je vynásobení měřítkem. Měřítko vytvoříme tak, že základ soustavy umocníme na f. V našem případě bude tedy měřítko 2^{13}. Jakmile máme měřítko, můžeme zadané číslo vynásobit měřítkem. Poté zaokrouhlíme klasickým způsobem k nejbližší hodnotě.

   $-1{,}35 * 2^{13} = (-11059{,}2)_D$
   Po zaokrouhlení: $(-11059)_D$

c. Převod do dvojkového doplňku
   Protože je číslo záporné, je potřeba ho převést do dvojkového doplňku. To uděláme pomocí vzorce $^2A = 2^n - A$, kde n je rozsah a A je naše zaokrouhlené číslo.

   $^2A = 2^n - A = 2^{16} - 11059 = (54447)_D$

   Poznámka: V tomto případě odečítáme od celého rozsahu zaokrouhlené číslo bez znaménka, tudíž neplatí pravidlo, že dvakrát mínus rovná se plus.

d. Převod do formátu IN
   Formát IN vytvoříme tak, že číslo ve dvojkovém doplňku převedeme do hexadecimální soustavy.

   $(54447)_D = (D4AF)_H$

   Výsledek ve formátu IN je $(D4AF)_H$

e. Převod do formátu FX
   V případě, že bychom z nějakého důvodu chtěli využít formát s desetinnou tečkou (FX), tak ten uděláme tak, že si číslo ve dvojkovém doplňku převedeme do binární soustavy, pro jednodušší práci si číslo v binární soustavě zapíšeme v celém rozsahu.

To znamená, že v případě, že výsledné číslo bude mít méně, než 16 bitů, tak doplníme nuly zleva. Poté vyznačíme desetinnou část. V našem případě bude mít desetinná část 13 bitů. MSB je znaménkový bit. V našem případě je jedna, protože se jedná o záporné číslo.

$(11059)_D = (1101010010101111)_B$

Vyznačení radix tečky: 110.1010010101111

Poté je potřeba převést celočíselnou a desetinnou část do hexadecimální soustavy.

Celočíselná část: $(6)_H$

Když převádíme desetinnou část do binární soustavy, tak postupujeme naopak než v případě celočíselné části. To znamená, že hexadecimální číslo tvoříme zprava doleva. Pro lepší převod si můžeme doplnit nuly do počtu dělitelného 4. V našem případě tedy 3 nuly doprava.

Doplnění nul: 110.1010010101111000

Převod do hexadecimální soustavy: $(110.1010010101111000)_B = (0.A578)_H$

Spojení celočíselné a desetinné části $(6.A578)_H$

Výsledek ve formát FX je $(6.A578)_H$

Příklad 3) Převod kladného čísla z formátu ℚm.f. Převeďte číslo $(0AF3)_H$ ve formátu ℚ3.10. Při výpočtu použijte rozsah 16 bitů.

a. Převod do binární soustavy a vyznačení MSB
Prvním krokem je, že si číslo převedeme do binární soustavy a podle MSB bitu zjistíme, zda-li se jedná o kladné, nebo záporné číslo.

$(0AF3)_H = (0000101011110011)_B$

MSB je 0, takže je číslo kladné.

Poznámka: Převod do hexadecimální soustavy není třeba, když si uvědomíme, že záporné číslo bude každé, které na první pozici v hexadecimální soustavě bude mít znaky 8 až F. Převod slouží pouze k vyznačení sign bitu.

b. Převod do decimální soustavy a vydělení měřítkem 2^f
Posledním krokem je převod čísla do decimální soustavy a vydělení měřítkem. V našem případě je měřítko 2^{10}.

$(0AF3)_H = (2803)_D$

$2803 / 2^{10} = 2{,}7373046875$

Výsledek je $(2{,}7373046875)_D$

1. Převod záporného čísla z formátu ℚm.f
   Převeďte číslo $(DAF3)_H$ ve formátu ℚ3.10. Při výpočtu použijte rozsah 16 bitů.

c. Převod do binární soustavy a vyznačení MSB
   Prvním krokem je, že si číslo převedeme do binární soustavy a podle MSB bitu zjistíme, zda-li se jedná o kladné, nebo záporné číslo.

   $(DAF3)_H = (1101101011110011)_B$

   MSB je 1, takže je číslo záporné.

   Poznámka: Převod do hexadecimální soustavy není třeba, když si uvědomíme, že záporné číslo bude každé, které na první pozici v hexadecimální soustavě bude mít znaky 8 až F. Převod slouží pouze k vyznačení sign bitu.

d. Převod z dvojkového doplňku
   Protože je číslo záporné, je potřeba ho převést z dvojkového doplňku. Toho dosáhneme pomocí vzorce $A = -(2^n - {}^2A)$. K převodu je ještě potřeba převést si číslo do dekadické soustavy

   $(DAF3)_H = (56051)_D$

   $A = -(2^n - {}^2A) = -(2^{16} - 56051) = (-9485)_D$

e. Vydělení měřítkem 2^f
   Posledním krokem je vydělení měřítkem. V našem případě je měřítko 2^{10}.

   $-9485 / 2^{10} = -9{,}2626953125$

   Výsledek je $(-9{,}2626953125)_D$

Příklad 4) Sčítání ve formátu ℚm.f. Sečtěte čísla A = 1,25, B = -1,625 ve formátu ℚ1.3. Při výpočtu použijte rozsah 8 bitů.

a. Převod do formátu IN
   Prvním krokem je převod obou čísel do formátu IN.
   $A = 1{,}25 * 2^3 = (10)_D$
   $(10)_D = (0A)_B$

   $B = -1{,}625 * 2^3 = (-13)_D$
   $^2B = 2^8 - 13 = (243)_D$
   $(243)_D = (F3)_H$

b. Součet
   Součet můžeme provést v jakékoliv soustavě. V našem případě jsem zvolil součet v hexadecimální soustavě

0A
+ F3
FD

Výsledek ve formátu IN je $(FD)_H$
Poznámka: Finální formát $\mathbb{Q}$m.f se změnil na Q2.3

2. Odčítání ve formátu $\mathbb{Q}$m.f
Odčítání probíhá stejně jako sčítání. Jediné, co se změní je to, že v případě, že je druhé číslo kladné, tak ho převedeme do dvojkového doplňku. V případě, že je číslo záporné, tak není třeba převádět do dvojkového doplňku, protože když provedeme dvakrát převod do dvojkového doplňku, tak máme kladné číslo.

Příklady k procvičení

1. Převeďte následující čísla do stanoveného formátu ℚm.f. Výsledek zapište ve formátu IN. Při výpočtu použijte rozsah 16 bitů.

$(2,8)_D$	ℚ2.13	Výsledek$(599A)_H$
$(-3.66)_D$	ℚ2.11	Výsledek$(E2B8)_H$
$(-18.339)_D$	ℚ5.10	Výsledek$(B6A5)_H$
$(0.7745)_D$	ℚ0.15	Výsledek$(6323)_H$
$(259.5)_D$	ℚ10.3	Výsledek$(081C)_H$

2. Převeďte následující čísla ze stanoveného formátu ℚm.f. Při výpočtu použijte rozsah 16 bitů.

$(6488)_H$	ℚ2.13	Výsledek$(3.1416015625)_D$
$(17EC)_H$	ℚ2.11	Výsledek$(2.990234375)_D$
$(80FD)_H$	ℚ5.10	Výsledek$(-31.7529296875)_D$
$(D386)_H$	ℚ0.15	Výsledek$(-0.34747314453125)_D$
$(0927)_H$	ℚ10.3	Výsledek$(292.875)_D$

3. Sečtěte následující čísla ve stanoveném formátu. Při výpočtu použijte rozsah 16 bitů.

A = 1,88; B = -1,66	Q1.13	Výsledek$(070A)_H$
A = -7,11; B = -2,933	Q4.10	Výsledek$(D7D7)_H$
A = 2,85; B = 1,25	Q3.11	Výsledek$(20CD)_H$
A = -3,713; B = 8,64	Q4.10	Výsledek$(13B5)_H$
A = 0,98; B = -0,515	Q0.14	Výsledek$(1DC2)_H$

4. Odečtěte následující čísla ve stanoveném formátu. Při výpočtu použijte rozsah 16 bitů.

A = 1,611; B = -1.383	Q1.13	Výsledek$(5FCF)_H$
A = -6,911; B = -3,154	Q4.10	Výsledek$(F0F9)_H$
A = 5,22; B = 1,33	Q3.11	Výsledek$(1F1F)_H$
A = -8,357; B = 8,753	Q4.10	Výsledek$(BB8F)_H$
A = 0,98; B = -0,923	Q0.14	Výsledek$(79CA)_H$

Návod k aplikaci

Aplikace do formátu ℚm.f

Aplikace se nazývá AO_FixedPoint a má 3 vstupy. První vstup je pro reálné číslo, které chceme převést. Číslo musí být zapsáno s desetinnou tečkou. Druhý vstup je pro m a poslední vstup je pro f. Aplikace si sama určí rozsah, maximální rozsah je 32 bitů. Aplikace má 2 výstupy. Jeden výstup je formát IN a druhý výstup je formát FX.

Příklady využití aplikace:

```
[IN, FX] = AO_FixedPoint(0.98354,0,14)
[IN, FX] = AO_FixedPoint(5.974,3,12)
```

Aplikace pro převod z formátu ℚm.f

Aplikace se nazývá AO_FixedPointToDec a má 3 vstupy. První vstup je pro číslo v pevné řádové čárce formátu IN. To musí být zapsáno jako hexadecimální číslo v jednoduchých uvozovkách. Druhý vstup je pro m a poslední vstup je pro f. Aplikace si sama určí rozsah, maximální rozsah je 32 bitů.

Příklady využití aplikace:

```
AO_FixedPointToDec('79CA',0,14)
Vysledek = AO_FixedPointToDec ('5F96',3,12)
```

Aplikace pro součet ve formátu ℚm.f

Aplikace se nazývá AO_FixedPointSum a má 4 vstupy. První dva vstupy jsou pro reálná čísla psaná s desetinnou tečkou. Třetí vstup je pro m a poslední vstup je pro f. Aplikace si sama určí rozsah, maximální rozsah je 32 bitů.

Příklady využití aplikace:

```
AO_FixedPointSum(0.98,-0.923,0,14)
Vysledek = AO_FixedPointSum(0.9165,0.46741,0,14)
```

Aplikace pro rozdíl ve formátu ℚm.f

Aplikace se nazývá AO_FixedPointMinus a má 4 vstupy. První dva vstupy jsou pro reálná čísla psaná s desetinnou tečkou. Třetí vstup je pro m a poslední vstup je pro f. Aplikace si sama určí rozsah, maximální rozsah je 32 bitů.

Příklady využití aplikace:

```
AO_FixedPointMinus(0.98,-0.923,0,14)

Vysledek = AO_FixedPointMinus(0.9165,0.46741,0,14)
```

Příloha E

Studijní materiály - Čísla v plovoucí řádové čárce

Základy číslicových systémů

Čísla v pohyblivé řádové čárce

Datum 21.4.2021
Autor Michael Foltyn, Ing. Marcel Fajkus, Ph.D.,
Kontakt marcel.fajkus@vsb.cz
Předmět Základy číslicových systémů
Studijní materiál vznikl v rámci bakalářské práce studenta Michaela Foltyna.

Formát čísel v pohyblivé řádové čárce

Formát s pohyblivou řádovou čárkou, anglicky Floating Point (FP) je formát, který se používá pro zobrazování reálných čísel. Reálné číslo se skládá z významných číslic zvaných significand a ze základu neboli base umocněného na řád n. Base je číslo, které definuje základ soustavy. Pro desítkovou soustavu je to číslo 10 a pro binární soustavu je to číslo 2.

Pro FP obecně platí vztah:

$$Significand * base^n,$$

kde $Significand$ je významná číslice se znaménkem, $base$ je základ číselné soustavy, n je exponent, který musí být celé číslo a $base^n$ tvoří měřítko.

Velkou výhodou tohoto formátu je, že můžeme jednoduše popsat velmi velké hodnoty, jako jsou například vzdálenosti ve vesmíru, ale také velmi malé hodnoty jako jsou výrobní parametry procesorů a podobně. FP má velký rozsah a maximální přesnost. V informačních technologiích je pak třeba zvážit jaký formát vybrat, aby rozsah nebyl zbytečně velký, popřípadě, aby byla zachována co nejvyšší přesnost.

Způsoby zápisu

Vědecký zápis – formát, který obsahuje significand, který je jakékoliv reálné číslo, základ a exponent. Příklad vědeckého zápisu 134.33×10^{-3}, 11010.11×2^6.

Technický zápis – zápis, který odpovídá prefixům SI soustavy. Exponent je tedy násobkem čísla 3. Příklad technického zápisu 75×10^{-3} m (metrů), neboli 75 mm (milimetrů), 13×10^6 b, neboli 13 Mb.

Normalizovaný vědecký zápis – zápis, ve který má v celočíselné části pouze jednu číslici, která není rovná nule. Příklad normalizovaného zápisu 1.133×10^{13}, 1.1001×2^5.

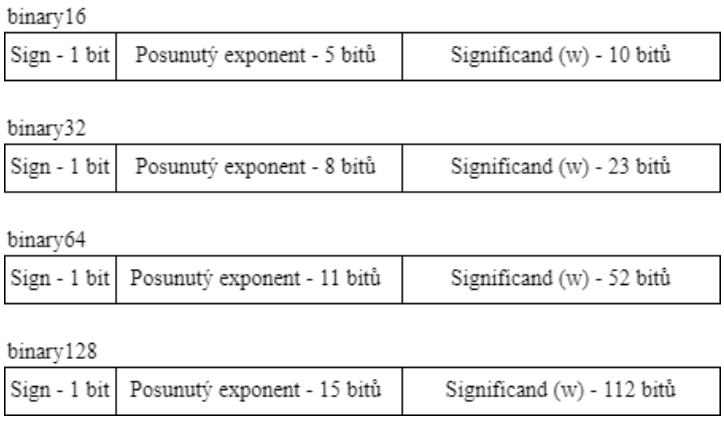
Formáty BinaryX

Formáty BinaryX jsou definovány standardem IEEE 754-2019. Mezi binární standardy patří binary16, binary32 v programování známý pod názvem single precision (jednoduchá přesnost), binary64 známý jako double precision (dvojitá přesnost) a binary128.

Tabulka 1 parametry jednotlivých formátů

Formát	Počet bitů pro significand	Počet bitů pro exponent (n)	Minimální exponent (E min) decimálně	Maximální exponent (E max) decimálně	Bias	Šířka pole significandu (w)
binary16	11	5	-14	15	15	10
binary32	24	8	-126	127	127	23
binary64	53	11	-1 022	1023	1 023	52
binary128	113	15	-16 382	16383	16 383	112

Jak můžeme z tabulky nahoře vidět, tak binary16 má velmi malý rozsah. Co se týče počtu bitů significandu, tak je třeba myslet na to, že první bit je sign bit, takže pro samotné číslo máme o bit méně. Avšak první jedničku v significandu můžeme vynechat, takže se dostaneme zpět na stejný počet bitů. Složení jednotlivých formátů můžeme pozorovat na obrázku 1.



Obrázek 1 Složení jednotlivých formátů BinaryX

Využití

Formát v pohyblivé řádové čárce je vhodný pro zápis velmi velkých, nebo také velmi malých hodnot ve vysoké přesnosti. V technice se používá pro zápis reálných čísel. Můžeme ho také nalézt ve spoustě programovacích jazycích (double, single precision).

Převod do formátu FP

Po převod je nutné znát formát, do kterého převádíme a samozřejmě jeho parametry. Pak postupujeme tak, že převedeme celočíselnou část a desetinnou část do binární soustavy. Převedeme na normalizovaný stav, kde se zaznamená exponent, který použijeme pro výpočet posunutého exponentu a poté už jen podle obrázku 1 sestavíme formát.

Převod z formátu FP

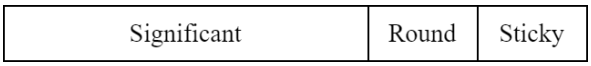
Zpětný převod funguje tak, že si číslo rozdělíme na sign bit, posunutý exponent a Significand (podle obrázku 1). Z posunutého exponentu poté vypočítáme exponent. Poté jsou dvě možnosti, jak převést significand na reálnou hodnotu. První je, že ho převedeme

z normalizovaného tvaru a poté převedeme do decimální soustavy. Druhou možností je, že ho převedeme v normalizovaném tvaru a poté vynásobíme měřítkem. K tomu nám poslouží dříve vypočítaný exponent.

Aritmetika v pohyblivé řádové čárce

Zaokrouhlování

Zaokrouhlování slouží k tomu, aby výsledek ve formátu FP byl co nejpřesnější. Pro zaokrouhlování používáme LSB, Round a sticky bit, které nám určují, zda-li máme zaokrouhlit. LSB bit je nejméně podstatný bit significandu. Sticky bit je vždy logický OR zbývajících nejméně významných bitů. Rozložení jednotlivých bitů můžeme vidět na obrázku 2.



Obrázek 2 Pozice LSB, Round a Sticky bitů

V této práci budu používat pouze zaokrouhlování k nejbližší a sudé, protože to je výchozí způsob zaokrouhlování v FP aritmetice. Avšak existují i další způsoby zaokrouhlování, a to například zaokrouhlení směrem k nule, směrem k plus nekonečnu a směrem k mínus nekonečnu.

V počítačích je zaokrouhlení často prováděno přičtením konstanty k číslu. K tomuto slouží termín ULP (unit in the last place) neboli jednotka na posledním místě.

Rozhodnout o zaokrouhlení můžeme pomocí této tabulky:

Tabulka 2 Tabulka rozhodující o zaokrouhlování

LSB	R	S	Operace
0	0	0	+ 0
0	0	1	+ 0
0	1	0	+ 0
0	1	1	+ 1/2 ulp
1	0	0	+ 0
1	0	1	+ 0
1	1	0	+ 1/2 ulp
1	1	1	+ 1/2 ulp

+1/2 ULP znamená, že na pozici p přičteme číslo 1. Pozice p je první pozice za significandem.

Sčítání

Operace sčítání probíhá tak, že si číslo rozdělíme na sign bit, exponent a significand. Significand si pak zobrazíme v normalizovaném tvaru. Tak to uděláme pro obě čísla. V případě, že čísla mají rozdílný exponent, tak je potřeba jedno z čísel převést na stejný

exponent. Poté je potřeba rozšířit oba significandy o dva bity zleva. Jeden bit je sign bit a druhý pro případ, že by došlo k přenosu do vyššího řádu. Hlavně nesmíme zapomenout na to, že pokud je číslo záporné, je potřeba ho převést na dvojkový doplněk. Provedeme operaci součtu binárních čísel a výsledek převedeme na normalizovaný tvar. V případě, že je výsledek nulový, tak vyznačíme Guard bit. Jedná se o první bit za significandem a v případě, že je vyznačený je potřeba provést logický posun doleva, abychom dostali normalizovaný tvar. Nakonec vyznačíme significand, LSB, Round a Sticky bity. A rozhodneme o zaokrouhlení. Jakmile máme zaokrouhleno, tak výsledek převedeme zpět do požadovaného formátu binaryX a případně do požadované soustavy.

Odčítání

Operace odčítání je identická se sčítáním. Stačí znegovat sign bit u druhého čísla a pak je postup úplně stejný.

Řešené příklady

Příklad 1) Převod do formátu FP. Převeďte číslo 113,625 do formátu binary16.

a. Převod celočíselné části do binární soustavy
Prvním krokem je převést celočíselnou část do binární soustavy.

$(113)_D = (1110001)_B$

b. Převod desetinné části do binární soustavy
Druhým krokem je převedení desetinné části do binární soustavy. Převod je jednoduchý. Číslo násobíme dvěma, když se dostane na hodnotu 1 a výše, tak jedničku odečteme, a do výsledku zapíšeme 1. V případě, že po vynásobení číslo jedničku nepřekročí, zapíšeme do výsledku 0. Tyto kroky opakujeme tak dlouho, dokud po odečtení jedničky nebudeme mít číslo rovné 0.

1. krok 0,625 * 2 = 1,25 – 1 = 0,25	Výsledek = 0.1
2. krok 0,25 * 2 = 0,5	Výsledek = 0.10
3. krok 0,5 * 2 = 1 – 1 = 0	Výsledek = 0.101

Jakmile máme převedenou celočíselnou i desetinnou část, tak oba výsledky spojíme do jednoho.

$(1110001) + (0.101) = (1110001.101)_B$

Poznámka: V případě, že má číslo nějakou nepravidelnou hodnotu, nebo je například periodické, tak tyto kroky provádíme tak dlouho dokud nemáme dostatečně velký výsledek, který by mohl být použit do significandu.

c. Převod na normalizovaný tvar
Dalším krokem je převedení na normalizovaný tvar. V našem případě posouváme desetinnou tečku doleva tak dlouho, dokud nám nezbude jeden nenulový znak.

$(1110001.101) \rightarrow (111000.1101) * 2^1 \rightarrow (11100.01101) * 2^2 \rightarrow \ldots \rightarrow (1.110001101) * 2^6$

Celkově jsme radix tečku posunuli o 6 pozic doleva, to znamená, že exponent e = 6.

d. Výpočet posunutého exponentu
Posunutý exponent se vypočítá pomocí vzorce E = e + bias. Z minulého kroku víme, že e = 6. Z tabulky 1 pak víme, že pro formát Binary16 je posunutí bias = 15.

$E = e + bias = 6 + 15 = 21$

e. Převod do formátu Binary16
Posledním krokem je sestavení formátu. Složení formátu můžeme vidět na obrázku 1. Pro binary16 využíváme 1 bit pro znaménko, 5 bitů pro posunutý exponent a 10 bitů pro Significand. Jak víme, tak převáděli jsme kladné číslo, tudíž sign bit bude 0

S = 0

Posunutý exponent převedeme do binární soustavy $(21)_D = (10101)_B$ a zapíšeme na pozici pro posunutý exponent.

S = 0 | E = 10101

Nakonec přidáme Significand. V tomto je však jeden háček, do formátu se přidává pouze část za radix tečkou, tedy pouze desetinná část, protože na pozici před tečkou bude vždy 1, nikdy tam nebude nic jiného, tak jí můžeme vynechat.

S = 0 | E = 10101 | Significand = 110001101

Jak víme z tabulky 1, tak pro Significand je šířka pole 10, avšak náš Significand má pouze 9 bitů, to znamená, že připisujeme nuly doprava (nakonec), dokud se počty bitů nebudou shodovat.

S = 0 | E = 10101 | Significand = 1100011010

Výsledek ve formátu Binary16 $(0101011100011010)_B$ nebo $(571A)_H$

Poznámka: V případě, že je číslo velmi malé (menší než nula), tak je potřeba posouvat significand doprava tak dlouho, dokud nebudeme mít na celočíselné části jedničku.

Příklad 2) Převod z formátu FP. Převeďte $(D3A0)_H$ na reálné číslo.

Ze znalosti jednotlivých formátů víme, že 4 hexadecimální číslice využívá formát Binary16.

a. Převedení do binární soustavy
Prvním krokem je převedení čísla do binární soustavy.

$(D3A0)_H = (1101001110100000)_B$

b. Rozdělení na Sign bit, posunutý exponent a Significand
Dalším krokem je rozdělení jednotlivých bitů, podle obrázku 1. Z tabulky 1 víme, že první bit je sign, dalších 5 bitů je pro posunutý exponent a poledních 10 bitů je pro Significand.

S = 1 | E = 10100 | Significant = 1110100000

c. Výpočet exponentu
Exponent vypočteme pomocí vztahu e = E – bias. Takže si posunutý exponent E převedeme do dekadické soustavy a odečteme od něj bias, který je pro binary16 bias = 15.

$E = (10100)_B = (20)_D$

e = E – bias = 20 – 15 = 5

d. Vytvoření normalizovaného tvaru
Dalším krokem, je vytvoření normalizovaného tvaru. K tomu použijeme significant a měřítko (základ soustavy umocněný na exponent). U Significandu nesmíme zapomenout na to, že část, kterou ve formátu vidíme je pouze desetinná část, to znamená musíme připsat „skrytý bit" 1.

Significand = 1110100000
Significand se skrytým bitem = 1.1110100000
základ soustavy = 2
exponent e = 5

Normalizovaný tvar = $1.1110100000 * 2^5$

Nesmíme zapomenout na sign bit, v našem případě byl 1, to znamená, že číslo je záporné.

Normalizovaný tvar = $-1.1110100000 * 2^5$

e. Převod na reálné číslo 1. způsob
Pro finální převod můžeme použít 2 způsoby. Jeden způsob je převedení z normalizovaného tvaru.

$-1.1110100000 * 2^5$ -> $-11.110100000 * 2^4$ -> … -> $-111101.00000 * 2^0$

Jak můžeme vidět, tak máme pouze celou část a žádnou desetinnou, takže číslo převedeme do decimální soustavy.

$(-111101.00000)_B = (-61)_D$

f. Převod na reálné číslo 2. způsob
Druhým a používanějším způsobem je, že číslo v normalizovaném tvaru převedeme do dekadické soustavy a vynásobíme měřítkem. Převedení do binární soustavy můžeme udělat pomocí polynomu

$(-1.1110100000)_B = -(1 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2} + 1 * 2^{-3} + 0 * 2^{-4} + 1 * 2^{-5}) =$
$= -(1 + 0{,}5 + 0{,}25 + 0{,}125 + 0 + 0{,}03125) = (-1{,}90625)_D$

$-1{,}90625 * 2^5 = (-61)_D$

Výsledek je $(-61)_D$

Příklad 3) Sčítání ve formátu FP. Sečtěte následující čísla ve formátu binary16 A = $(C310)_H$ a B = $(5456)_H$.

a. Převod do binární soustavy a rozdělení na S | E | Significand
A = (C310) = 1100001100010000 = 1 | 10000 | 1100010000
S_A = 1 | E_A = 10000 | $Significand_A$ = 1100010000

B $(5456)_H = 0101010001010110 = 0 \mid 10101 \mid 0001010110$
$S_B = 0 \mid E_B = 10101 \mid Significand_B = 0001010110$

Podle sign bitu víme, že číslo A je záporné.

b. Převod na normalizovaný tvar
Pro převod na normalizovaný tvar potřebujeme bias a exponent. Z tabulky 1 víme, že bias je pro binary16 bias = 15 a exponent vypočítáme pomocí vzorce e = E – bias. Při vytváření normalizovaného tvaru nesmíme zapomenout na „skrytý bit".

$E_A = (10000)_B = (16)_D$
$e_A = E_A – bias = 16 – 15 = 1$
Normalizovaný tvar: $1.1100010000 * 2^1$

$E_B = (10101)_B = (21)_D$
$e_B = E_B – bias = 21 – 15 = 6$
Normalizovaný tvar: $1.0001010110 * 2^6$

c. Sjednocení exponentů
Abychom mohli provést operaci sčítání, je potřeba mít obě čísla ve stejném měřítku. To znamená, že posuneme A o 5 pozic doprava e + 5.

$A = 1.1100010000 * 2^1 \rightarrow 0.11100010000 * 2^2 \rightarrow \ldots \rightarrow 0.000011100010000 * 2^6$

Protože jsou nuly zprava v desetinné části nepodstatné, tak je můžeme odstranit.

$A = 0.00001110001 * 2^6$

d. Doplnění vedoucích bitů
Protože se může stát, že při součtu dojde k přenosu do vyššího řádu, tak přidáme jeden bezvýznamný bit zleva. Poté přidáme ještě jeden bit pro znaménko.

A = 0.00001110001 = 000.00001110001
B = 1.0001010110 = 001.0001010110

e. Převedení záporných čísel do dvojkového doplňku
Pro součet je potřeba převést všechna záporná čísla do dvojkového doplňku. Jak víme z předchozích kroků, tak číslo A je záporné. Dvojkový doplněk uděláme tak, že číslo znegujeme a přičteme jedničku

$^2A = \sim A + 1 = (111.11110001111)_B$

f. Součet
Provedeme klasický součet binárních čísel. Sčítáme námi upravené significandy.

```
  111.11110001111
+ 001.00010101100
  001.00000111011
```

Bity, které po součtu přetekli do vyššího řádu ignorujeme. Jak můžeme vidět, tak výsledek je kladné číslo, protože sign bit je 0. V případě, že by číslo bylo záporné, je potřeba převést z dvojkového doplňku.

Poznámka: Protože je číslo B o jeden bit menší, tak jsem nakonec (doprava) dopsal jednu nulu.

g. Převod na normalizovaný tvar
Po součtu je potřeba upravit výsledek na normalizovaný tvar. V našem případě pouze odstraníme první dvě nuly zleva. Nemusíme provádět žádný posun. Nesmíme také zapomenout, že naše sjednocené posunutí e = 6.

$001.00000111011 = 1.00000111011 * 2^6$

h. Označení zaokrouhlovacích bitů
Pro co nejpřesnější výsledek potřebujeme rozhodnout, zda-li číslo zaokrouhlit, nebo no. Pro to si potřebujeme označit LSB, Round a sticky bit. Z tabulky 1 víme, že Significand má šířku pole 10 bitů. Proto je potřeba si vyznačit 10 bitů za radix tečkou. Číslo na poslední pozici Significandu je LSB, další za ním je Round bit a všechny ostatní bity jsou Sticky bity.

1.0000011101**1**

Červenou barvou je označený LSB bit a zelenou barvou je označený Round bit, jak můžeme vidět, tak nemáme žádné sticky bity. Sticky bit je vždy logický OR zbývajících nejméně významných bitů za Round bitem.

LSB = 1
R = 1
S = 0

Když se podíváme do tabulky 2, tak zjistíme, že musíme zaokrouhlit. Zaokrouhlení probíhá tak, že k výsledku přičteme + ½ ulp na pozici p. V našem případě přičteme jedničku na pozici round bitu (první pozici za significandem).

```
  1.00000111011
+             1
---------------
  1.00000111100
```

Protože pro Significand máme 10 bitů, ale desetinná část má 11 bitům, můžeme poslední nulu odstranit

1.00000111100 = 1.0000011110

i. Převedení výsledku do formátu Binary16
Nakonec výsledek převedeme do formátu binary16.

Výsledek je kladný
S = 0

e = 6
E = 6 + 15 = $(21)_D$ = $(10101)_B$

Significant = 0000011110

Výsledek je $(0101010000011110)_B$ nebo $(541E)_H$

Příkad 5) Odčítání ve formátu FP

Ssčítání a odčítání je prakticky stejné. Jediné, v čem se odčítání liší je to, že je potřeba znegovat znaménkový bit druhého čísla.

Příklady k procvičení

1. Převeďte následující čísla do stanoveného formátu binaryX. Výsledek zapište v hexadecimálním tvaru.

$(3,5)_D$	Binary16	Výsledek $(4300)_H$
$(-7,1875)_D$	Binary16	Výsledek $(C730)_H$
$(-40,4375)_D$	Binary16	Výsledek $(D10E)_H$
$(108.8125)_D$	Binary32	Výsledek $(42D9\ A000)_H$
$(-0.0625)_D$	Binary32	Výsledek $(BD80\ 0000)_H$

2. Převeďte následující čísla ze stanovených formátů na reálná čísla

$(B880)_H$	Binary16	Výsledek $(-0,5625)_D$
$(5A04)_H$	Binary16	Výsledek $(192,5)_D$
$(CAB0)_H$	Binary16	Výsledek $(-13,375)_D$
$(4380\ E000)_H$	Binary32	Výsledek $(257,75)_D$
$(C337\ F800)_H$	Binary32	Výsledek $(-183,9688)_D$

3. Sečtěte následující čísla ve formátu binary16. Výsledek zapište v hexadecimálním tvaru.

$A = (AAAA)_H$, $B = (BBBB)_H$	Výsledek $(BC13)_H$
$A = (B800)_H$, $B = (3800)_H$	Výsledek $(0000)_H$
$A = (3F00)_H$, $B = (C340)_H$	Výsledek $(BF80)_H$
$A = (BA00)_H$, $B = (B000)_H$	Výsledek $(BB00)_H$
$A = (C820)_H$, $B = (4280)_H$	Výsledek $(C500)_H$

4. Odečtěte následující čísla ve formátu binary16

$A = (BF80)_H$, $B = (3600)_H$	Výsledek $(C080)_H$
$A = (B800)_H$, $B = (3800)_H$	Výsledek $(BC00)_H$
$A = (4810)_H$, $B = (4480)_H$	Výsledek $(4340)_H$
$A = (4A60)_H$, $B = (BE80)_H$	Výsledek $(4B30)_H$
$A = (4D00)_H$, $B = (CD00)_H$	Výsledek $(C500)_H$

Návod k aplikaci

Aplikace pro převod do formátu FP

Aplikace se nazývá AO_BinaryX a má 2 vstupy. Prvním vstupem je reálné číslo s desetinnou tečkou, při použití desetinné čárky program spadne. Druhým vstupem je číslo formátu. Možné formáty jsou 16, 32, 64, 128.

Příklady využití aplikace

```
AO_BinaryX(-20.75, 16)
AO_BinaryX(1.99, 32)
AO_BinaryX(1.4579849, 64)
AO_BinaryX(pi, 128)
```

Aplikace pro převod z formát FP na reálné číslo

Aplikace se nazývá AO_BinaryXToDec a má 1 vstup. Vstupem je hexadecimální číslo psané v jednoduchých uvozovkách. Mezi čísly se nepíše mezera. To znamená, že formát zápisu je 'xxxx', 'xxxxxxxx' a tak dále.

Příklady využití aplikace

```
AO_BinaryXToDec('4B30')
AO_BinaryXToDec('4B300000')
AO_BinaryXToDec('400921FB54442D18')
AO_BinaryXToDec('4000921FB54442D180000000000000000')
```

Aplikace pro sčítání ve formátech BinaryX

Aplikace se nazývá AO_BinaryXSum a má 2 vstupy. Oba vstupy jsou hexadecimální čísla v jednoduchých uvozovkách. Mezi čísly se nepíše mezera. To znamená, že formát zápisu je 'xxxx', 'xxxxxxxx' a tak dále.

Příklady využití aplikace

```
AO_BinaryXSum('4B30', '4B30')
AO_BinaryXSum('4B300000', '4B300000')
AO_BinaryXSum('400921FB54442D18', '400921FB54442D18')
AO_BinaryXSum('4000921FB54442D180000000000000000',
4000921FB54442D180000000000000000')
```

Aplikace pro odčítání ve formátech BinaryX

Aplikace se nazývá AO_BinaryXMinus a má 2 vstupy. Oba vstupy jsou hexadecimální čísla v jednoduchých uvozovkách. Mezi čísly se nepíše mezera. To znamená, že formát zápisu je 'xxxx', 'xxxxxxxx' a tak dále.

Příklady využití aplikace

```
AO_BinaryXMinus('4B30', '4B30')
AO_BinaryXMinus('4B300000', '4B300000')
AO_BinaryXMinus('400921FB54442D18', '400921FB54442D18')
AO_BinaryXMinus('4000921FB54442D180000000000000000',
4000921FB54442D18000000000000000')
```

Příloha F

Studijní materiály - UTF

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA TELEKOMUNIKAČNÍ TECHNIKY

Základy číslicových systémů

Unicode Transformation Format

Datum 20.4.2021
Autor Michael Foltyn, Ing. Marcel Fajkus, Ph.D.,
Kontakt marcel.fajkus@vsb.cz
Předmět Základy číslicových systémů
Studijní materiál vznikl v rámci bakalářské práce studenta Michaela Foltyna.

Unicode Tranformation Format

Unicode je technická norma definující jednotnou znakovou sadu pro reprezentaci a zpracování textů. Nejnovější verze Unicode obsahuje více než 142 000 znaků a momentálně pokrývá 154 moderních a historických písem, ale také mnoho sad symbolů a smajlíků. UTF je pak způsob kódování jakým ukládáme řetězce složené z Unicode znaků do paměti počítače. Momentálně jsou 3 formáty a to UTF-8, UTF-16 a UTF-32.

UTF-8

Jeden z nejpoužívanějších formátů, protože je prostorově úsporný. Je odolný proti chybám a je zpětné kompatibilní s ASCII. Setkáme se s ním na většině webových stránkách, ale často i v některých operačních systémech. Pro kódování znaků používá 1 – 4 Byty podle jejich kódu v Unicode. V tabulce níže můžeme vidět princip tvoření formátu UTF-8. Formát je rozdělený do čtyř skupin a každá skupina má svá pravidla, respektive prefix pro vytvoření formátu.

Významové bity	První Unicode pozice	Poslední Unicode pozice	Vedoucí Byte 1 binárně	Byte 2 binárně	Byte 3 binárně	Byte 4 binárně
7	U+0000	U+007F	0xxx xxxx			
11	U+0080	U+07FF	110x xxxx	10xx xxxx		
16	U+0800	U+FFFF	1110 xxxx	10xx xxxx	10xx xxxx	
21	U+1 0000	U+1F FFFF	1111 0xxx	10xx xxxx	10xx xxxx	10xx xxxx

Převod do UTF-8

Převod do formátu probíhá tak, že Unicode znak převedeme do binární soustavy a poté rozdělíme do skupin podle tabulky nahoře. Například, když budeme mít Unicode do 007F, tak použijeme všech 7 bitů pro vytvoření UTF-8. Vedoucí neboli leading Byte využívá při každém formátu jiný binární prefix, a tedy jiný počet bytů, který můžeme doplnit. U dalších bytů pak vždy doplňujeme k binárnímu prefixu 10 šest znaků.

UTF-16

UTF-16 používá až dvě 16 bitové dvojice, aby pokryl celý 21 bitový prostor. Pokud je využita pouze jedna z dvojic, tedy v případě hodnot do U+FFFF, tak se hodnoty kódují, tak jak jsou bez žádné změny. V případě, že je Unicode větší než U+FFFF je potřeba využít obě dvojice. Pro tento případ existuje speciální dvojice takzvaná surrogate pair. První z dvojic, zvaná lead surrogate je v rozsahu 0xD800 až 0xDBFF. Druhá z dvojic, zvaná trail surrogate je v rozsahu 0xDC00 až 0xDFFF. UTF-16 se používá ve všech podporovaných operačních systémech od společnosti Microsoft. Společnost Apple využívá tento formát pro SMS (Short Massage Service).

Převod do UTF-16

V případě, že máme Unicode hodnoty do U+FFFF, tak se vytvoří formát s hodnotou tak jak je. Například pro U+1234 je UTF16 formát roven 0x1234. V případě, že potřebujeme vyjádřit Unicode hodnotu, která je vyšší než U+FFFF, tak musíme využít lead a trail surrogate. Samotný převod pak funguje tak, že od Unicodu odečeteme číslo 0x01 0000. Poté převedeme do binární soustavy a rozdělíme do dvou dvojic po 10 bitech zprava ty pak převedeme zpět do Hexadecimální soustavy. Nižších 10 bitů (bity s nižší váhou) náleží trail surrogate, vyšších 10 bitů (bity s vyšší váhou) náleží lead surrogate. K lead surrogate poté přičteme 0xD800 a k trail surrogate přičteme 0xDC00.

UTF-32

Tento formát využívá pro slovo 32 bitů. Jelikož Unicode má maximálně 21 bitů, tak při převodu do UTF-32 se doplní nuly do 32 bitů. Tento formát jinak odpovídá Unicodu.

Převod do UTF-32

Převod funguje tak, že vezmeme Unicode tak jak je a doplníme nuly do 4 bytů (32 bitů). Převod z formátu UTF-32 pak funguje tak, že nuly odstraníme. Jediné, na co je třeba dávat pozor je endianita.

Řešené příklad

Příklad 1) Převod do formátu UTF-8. Převeďte U+15698 do formátu UTF-8.

a. Převod Unicode do binární soustavy.
Hodnota Unicode je v hexadecimální soustavě, takže ji klasickým způsobem převedeme do binární soustavy.

$(1)_H = (0001)_B$
$(5)_H = (0101)_B$
$(6)_H = (0110)_B$
$(9)_H = (1001)_B$
$(8)_H = (1000)_B$

Jak můžeme vidět, máme 20 bitů, to znamená, že z tabulky si vybereme poslední řádek a podle něj budeme tvořit formát. Jednotlivé byty zřetězíme do jednoho 20 bitového čísla.

Zřetězené 20bitové číslo - 00010101011010011000

b. Přiřazení k bytům
V tabulce si můžeme všimnout, že pro vedoucí byte (leading) se používá prefix „1111 0xxx“, pro každý další byte je prefix „10xx xxxx“ a celkově máme 21 významových bitů. Jak víme z předchozího kroku, tak máme pouze 20bitové číslo, ale máme 21 významových bitů, to znamená, že na začátek musíme přidat 1 bezvýznamný bit, tedy 0.

21bitové číslo – 000010101011010011000

Jakmile máme hotovo, tak si bity rozdělíme do skupin. Jak už víme, tak leading byte má 3 významové bity a každý další byte má 6 významových bitů. Takže první skupina bude mít 3 bity a další budou mít 6.

Skupina1 = 000
Skupina2 = 010101
Skupina3 = 011010
Skupina4 = 011000

Jakmile máme vytvořené skupiny, tak je přiřadíme k bytům

1. byte = 1111 0xxx = 1111 0000
2. byte = 10xx xxxx = 1001 0101
3. byte = 10xx xxxx = 1001 1010
4. byte = 10xx xxxx = 1001 1000

c. Převod do hexadecimální soustavy
Jakmile máme vytvořené byty, tak je můžeme převést do hexadecimální soustavy

1. byte = $(11110000)_B = (F0)_H$
2. byte = $(10010101)_B = (95)_H$
3. byte = $(10011010)_B = (9A)_H$
4. byte = $(10011000)_B = (98)_H$

Výsledek v UTF-8 je 0xF0 0x95 0x9A 0x98

Příklad 2) Převod z formátu UTF-8. Převeďte 0xF0 0x9F 0xAA 0xB3 ve formátu UTF-8 do Unicode.

a. Převedení do binární soustavy
Prvním krokem je, že si jednotlivé byty převedeme do binární soustavy.

1. byte = $(F0)_H = (11110000)_B$
2. byte = $(9F)_H = (10011111)_B$
3. byte = $(AA)_H = (10101010)_B$
4. byte = $(B3)_H = (10110011)_B$

b. Odstranění prefixů
Jak můžeme vidět máme 4 byty, to znamená, že se podíváme do tabulky nahoře, na řádek, kde se využívají 4 byty a zjistíme, jaký je prefix pro první byte a jaký je prefix pro každý další byte. Jak můžeme vidět, tak pro první byte je prefix „1111 0xxx“ a pro každý další „10xx xxxx“. Tyto prefixy pak odstraníme z našich bytů.

1. byte = (1111 0000) – (1111 0xxx) = 000
2. byte = (1001 1111) – (10xx xxxx) = 011111
3. byte = (1010 1010) – (10xx xxxx) = 101010
4. byte = (1011 0011) – (10xx xxxx) = 110011

Znak mínus v tomto případě používám pro odstranění prefixů.

c. Zřetězení a převod do hexadecimální soustavy
Nakonec to, co nám zbylo po odebrání prefixů zřetězíme a převedeme do hexadecimální soutavy.

Zřetězené číslo = 000 + 011111 + 101010 + 110011 = $(00001111101010110011)_B$ =
= $(1FAB3)_H$

Výsledek v Unicode je U+1 FAB3

Příklad 3) Převod do formátu UTF-16. Převeďte U+2 FACE do formátu UTF-16 Big endian i Little endian.

a. Odečtení 0x1 0000
Při převodu do formátu UTF-16 je vždy prvním krokem odečtení hodnoty 0x1 0000 v případě, že máme Unicode hodnotu větší, než FFFF.

2FACE – 10000 = 1FACE

b. Převod do binární soustavy a rozdělení do skupiny po 10 bitech
Převod do binární soustavy se dělá proto, aby se číslo lépe rozdělilo do dvojic.
$(1FACE)_H = (00011111101011001110)$

Formát UTF-16 se skládá ze dvou párů. Lead surrogate a Trail surrogate, proto je potřeba binární hodnotu rozdělit do 2 skupin po 10 bitech.

Lead = (0001111110)
Trail = (1011001110)

Po rozdělení do skupin převedeme obě skupiny zpět do hexadecimální soustavy

$Lead = (0001111110)_B = (07E)_H$
$Trail = (1011001110)_B = (2CE)_H$

c. Přičtení 0xD800 a 0xDC00
Finálním krokem je přičtení 0xD800 k Lead části a 0xDC00 k Trail části.

D800 + 07E = D87E
DC00 + 2CE = DECE

Výsledek ve formátu UTF-16 Big endian 0xD87E 0xDECE
Výsledek ve formátu UTF-16 Little endian 0x7ED8 0xCEDE

Příklad 4) Převod z formátu UTF-16. Převeďte 0xDB80 0xDF13 ve formátu UTF-16 Big endian do Unicode.

Protože používáme Big Endian, tak není třeba žádné úpravy, ale v případě, že bychom měli Little endia, tak musíme převést na Big Endian.

a. Odečtení 0xD800 a 0xDC00
Prvním krokem je vždy odečtení hodnot 0xD800 od Lead části a 0xDC00 od Trail části.

$DB80 – D800 = (380)_H$
$DF13 – DC00 = (313)_H$

b. Převod do binární soustavy a zřetězení
Pro jednodušší zřetězení je potřeba oba páry převést do binární soustavy

$(380)_H = (001110000000)_B$
$(313)_H = (001100010011)_B$

Před zřetězením ještě musíme brát v potaz to, že když jsme převedli čísla do binární soustavy, tak má každé číslo 12 bitů, ale my pro sestavení unicode potřebujeme pouze 10 bitů, to znamená, že první dvě nuly odstraníme.

$(001110000000)_B = (1110000000)_B$
$(001100010011)_B = (1100010011)_B$

Zřetězení = 1110000000 + 1100010011 = $(11100000001100010011)_B$

c. Převod do hexadecimální soustavy
$(11100000001100010011)_B = (E0313)_H$

d. Přičtení 0x1 0000
E0313 + 1 0000 = F0313

Výsledek je U+F 0313

Příklad 5) Převod do UTF-32. Převeďte U+12345 do formátu UTF-32 Big endian i Little endian.

Převod do formátu UTF-32 je nejjednodušší, protože nevyžaduje žádné úpravy, sčítání ani odčítání. Pouze k číslu doplníme nuly do 32 bitů. Protože víme, že naše číslo má 20 bitů, tak doplníme 12 bitů, tedy 3 nuly hexadecimálně.

12345 = 00012345

Výsledek je 0x00 0x01 0x23 0x45 v Big endian
Výsledek je 0x45 0x23 0x01 0x00 v Little endian

Příklad 6) Převod z UTF-32. Převeďte 0xEF 0xBE 0x01 0x00 ve formátu UTF-32 Little endian do Unicode.

Zpětný převod funguje tak, že číslo převedeme do Big endian a odstraníme nuly.

Převedení na BE - 0x00 0x01 0xBE 0xEF
odstranění nul - 0001BEEF = 1BEEF

Výsledek je U+1BEEF

Příklady k procvičení

Poznámka: Dolní index určuje endianitu.

1. Převeďte Unicode do UTF-8

U+FAFA	Výsledek (0xEF 0xAB 0xBA)
U+78	Výsledek (0x78)
U+DA5CE	Výsledek (0xF3 0x9A 0x97 0x8E)
U+F5	Výsledek (0xC3 0xB5)
U+1F00D	Výsledek (0xF0 0x9F 0x80 0x8D)

2. Převeďte UTF-8 do Unicode

0xF2 0xBF 0xB3 0x91	Výsledek (U+BFCD1)
0xF2 0xAB 0xB3 0x9E	Výsledek (U+ABCDE)
0xEF 0x85 0x95	Výsledek (U+F158)
0xC3 0x9A	Výsledek (U+DA)
0x15	Výsledek (U+15)

3. Převeďte Unicode do UTF-16 BE a LE

U+3DEAD	Výsledek $(0xD8B7\ 0xDEAD)_{BE}$ a $(0xB7D8\ 0xADDE)_{LE}$
U+A321A	Výsledek $(0xDA4C\ 0xDE1A)_{BE}$ a $(0x4CDA\ 0x1ADE)_{LE}$
U+7FFAD	Výsledek $(D9BF\ 0xDFAD)_{BE}$ a $(0xBFD9\ 0xADDF)_{LE}$
U+FEDCB	Výsledek $(0xDBBB\ 0xDDCB)_{BE}$ a $(0xBBDB\ 0xCBDD)_{LE}$
U+AD00D	Výsledek $(0xDA74\ 0xDC0D)_{BE}$ a $(0x74DA\ 0x0DDC)_{LE}$

4. Převeďte UTF-16 do Unicode

$(0xABDB\ 0xE3DD)_{LE}$	Výsledek	(U+FADE3)
$(0xDB07\ 0xDFE3)_{BE}$	Výsledek	(U+D1FE3)
$(0xD9A6\ 0xDD8F)_{BE}$	Výsledek	(U+7998F)
$(0x76DA\ 0xDADE)_{LE}$	Výsledek	(U+ADADA)
$(0x6FDB\ 0xBEDE)_{LE}$	Výsledek (U+EBEBE)	

5. Převeďte Unicode do UTF-32 BE a LE

U+DEDA	Výsledek $(0x00\ 0x00\ 0xDE\ 0xDA)_{BE}$ a $(0xDA\ 0xDE\ 0x00\ 0x00)_{LE}$
U+1F1DA	Výsledek $(0x00\ 0x01\ 0xF1\ 0xDA)_{BE}$ a $(0xDA\ 0xF1\ 0x01\ 0x00)_{LE}$
U+5F5F5	Výsledek $(0x00\ 0x05\ 0xF5\ 0xF5)_{BE}$ a $(0xF5\ 0xF5\ 0x05\ 0x00)_{LE}$
U+13	Výsledek $(0x00\ 0x00\ 0x00\ 0x13)_{BE}$ a $(0x13\ 0x00\ 0x00\ 0x00)_{LE}$
U+DAD1	Výsledek $(0x00\ 0x00\ 0xDA\ 0xD1)_{BE}$ a $(0xD1\ 0xDA\ 0x00\ 0x00)_{LE}$

6. Převeďte UTF-32 do Unicode

$(0x00\ 0x01\ 0xAB\ 0xCD)_{BE}$	Výsledek (U+1ABCD)
$(0x88\ 0x13\ 0x00\ 0x00)_{LE}$	Výsledek (U+1388)

$(0x45\ 0x19\ 0x07\ 0x00)_{LE}$	Výsledek (U+71945)
$(0x00\ 0x00\ 0x00\ 0x01)_{BE}$	Výsledek (U+1)
$(0xFF\ 0xAA\ 0x0B\ 0x00)_{LE}$	Výsledek (U+BAAFF)

Návod k aplikaci

Aplikace na převod do UTF-8

Aplikace se nazývá AO_UTF8 a má 1 vstup. Vstupem je Unicode hodnota v podobě stringu s jednoduchými uvozovkami. Unicode se píše bez „U+“, píše se pouze hodnota. Aplikace není case sensitive.

Příklady využití aplikace:

```
AO_UTF8('15')
AO_UTF8('aba')
AO_UTF8('DEAD')
```

Aplikace na převod z UTF-8 do Unicode

Aplikace se nazývá AO_UTF8ToUnicode a má 1 vstup. Vstupem je UTF-8 ve formátech 'xx', 'xx xx', 'xx xx xx' a 'xx xx xx xx'. UTF-8 se píše bez „0x“, píše se pouze hodnota. Aplikace není case sensitive.

Příklady využití aplikace:

```
AO_UTF8ToUnicode('EF AB BA')
AO_UTF8ToUnicode('78')
AO_UTF8ToUnicode('f0 9f 80 8D')
```

Aplikace na převod do UTF-16

Aplikace se nazývá AO_UTF16 a má 2 vstupy. Prvním vstupem je Unicode hodnota v podobě stringu s jednoduchými uvozovkami. Unicode se píše bez „U+“, píše se pouze hodnota. Druhým vstupem je endian v podobě 'le', nebo 'be'. Aplikace není case sensitive.

Příklady využití aplikace:

```
AO_UTF16('15', 'be')
AO_UTF16('aba', 'LE')
AO_UTF16('DEAD', 'BE')
```

Aplikace na převod z UTF-16 do Unicode

Aplikace se nazývá AO_UTF16ToUnicode a má 2 vstupy. Vstupem je UTF-16 ve formátech 'xxxx' a 'xxxx xxxx'. UTF-16 se píše bez „0x“, píše se pouze hodnota. Druhým vstupem je endian v podobě 'le', nebo 'be'. Aplikace není case sensitive.

Příklady využití aplikace:

```
AO_UTF16ToUnicode('D8B7 DEAD', 'be')
AO_UTF16ToUnicode('FFFF', 'be')
AO_UTF16ToUnicode('B7D8 AddE', 'le')
```

Aplikace na převod do UTF-32

Aplikace se nazývá AO_UTF32 a má 2 vstupy. Prvním vstupem je Unicode hodnota v podobě stringu s jednoduchými uvozovkami. Unicode se píše bez „U+“, píše se pouze hodnota. Druhým vstupem je endian v podobě 'le', nebo 'be'. Aplikace není case sensitive.

Příklady využití aplikace:

```
AO_UTF32('15', 'be')
AO_UTF32('aba', 'BE')
AO_UTF32('DEAD', 'le')
```

Aplikace na převod UTF-32 do Unicode

Aplikace se nazývá AO_UTF32ToUnicode a má 2 vstupy. Vstupem je UTF-32 ve formátu 'xx xx xx xx'. UTF-32 se píše bez „0x“, píše se pouze hodnota. Druhým vstupem je endian v podobě 'le', nebo 'be'. Aplikace není case sensitive.

Příklady využití aplikace:

```
AO_UTF32ToUnicode('00 0b aa FF', 'be')
AO_UTF32ToUnicode('00 0B AA FF', 'be')
AO_UTF32ToUnicode('FF AA 0B 00', 'le')
```