

Binární prerekvizita

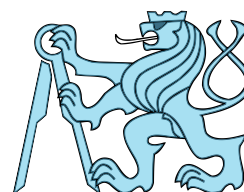
aneb vstupní brána do digitálního světa

Richard Šusta



Katedra řídicí
techniky

ČVUT-FEL v Praze



Verze 1.0 ze dne 18. února 2023

Obsah

1	Souhrn prerekvizity.....	5
2	Kódování čísel	7
2.1	Binární číslo <i>unsigned</i> - bez znaménka	8
2.1.1	Změna bitové délky čísla	8
2.1.2	Logické posuny	8
2.1.3	Převod binárního čísla bez znaménka na dekadické číslo	9
2.1.4	Převod dekadického čísla na binární číslo bez znaménka	10
2.1.5	Aritmetické přetečení při sčítání a odčítání	11
2.2	Binární číslo <i>signed</i> - se znaménkem ve dvojkovém doplňku	13
2.2.1	Významné vlastnosti	14
2.2.2	Aritmetická negace pomocí dvojkového doplňku	14
2.2.3	Převod dekadických čísel na binární.....	14
2.2.4	Převod binárních čísel na dekadická.....	15
2.2.5	Změna délky čísla - sign extension.....	16
2.2.6	Logické a aritmetické posuny	17
2.2.7	Aritmetické přetečení při sčítání a odčítání	19
2.3	Celá čísla se znaménkem v přímém a aditivním kódu	20
2.4	Hexadecimální notace.....	21
2.4.1	Hexadecimální čísla	21
2.4.2	Číselné soustavy.....	22
2.5	BCD - Binary Coded Decimal.....	24
2.5.1	Násobení BCD číslem 2.....	25
2.5.2	Převod binárního čísla unsigned na BCD	26
2.6	Kódování znaků ASCII	26
2.6.1	Extended ASCII	28
2.7	Kolik je 1000?	30
2.8	Test znalostí z kapitoly 2	31
3	Příloha.....	32
3.1	Abecední seznam zkratk a použitých termínů	32
3.2	Řešení testu z kapitoly 2	33

Seznam obrázků

Obrázek 1 - Byte, bit, MSB, LSB	7
Obrázek 2 - Přičítání a odčítání 4bitových čísel bez znaménka	12
Obrázek 3 - 4bitová čísla <i>unsigned</i> a <i>signed</i>	13
Obrázek 4 - Znaménkové rozšíření pro binární čísla <i>signed</i>	16
Obrázek 5 - Logický a aritmetický posun vpravo.....	17
Obrázek 6 - Posun vlevo 8bitového binárního čísla	18
Obrázek 7 - Znaménko a hodnota.....	20
Obrázek 8 - Aditivní kód s $K=8$	20
Obrázek 9 - BCD číslo 35.....	24
Obrázek 10 - ENIAC Electronic Numerical Integrator and Computer	24
Obrázek 11 - Princip extended ASCII	28

Seznam tabulek

Tabulka 1 - Přičítání 1 k 8bitovému číslu bez znaménka	11
Tabulka 2 - Aritmetické přetečení při sčítání 8bitových binárních čísel <i>signed</i>	19
Tabulka 3 - Kdy se objeví příznak <i>overflow</i> u operace s binárními čísly <i>signed</i>	19
Tabulka 4 - Běžně používané základy (radixy) pro číselné soustavy.....	22
Tabulka 5 - ASCII kódování znaků	27

1 Souhrn prerekvizity

Prerekvizita znamená minimální potřebné znalosti pro absolvování určitého předmětu nebo kurzu, které musí být splněné před umožněním jeho studia. [Slovník cizích slov]

Proč vznikla?

Mnozí se zajisté setkali s binárními čísly během studia nebo si téma sami nastudovali, ale jiní se s nimi setkali jen letmo. Když se kdysi přednášky zahájily od úplných základů, znalější studenti pak psali do ankety hodnocení předmětů, že se v prvních hodinách docela nudili. Po reakci na jejich připomínky se výklad zaměřil rychleji na zajímavější otázky, ale méně obeznámení poté vyčítali, že nerozuměli úvodním pasážím.

Napsal jsem tady prerekvizitu pro sjednocení vstupních znalostí, aby se vyhovělo všem. Zahrnul jsem do ní pouze lehké pojmy. Složitosti zůstaly v přednáškách.

Jak ji prerekvizitu studovat?

Přečtěte si určitě celý text. Budete-li mít dojem, že nějaké pasáži rozumíte, nepřeskakujte ji, ale zběžně si ji prohlédněte, zda přece jenom v ní neobjevíte nějaký nový poznatek. Zpomalte však, pokud narazíte na méně známé pojmy či si někde nebudete stoprocentně jisti, a pečlivě si téma prostudujte včetně postupů v řešených příkladech.

Přehled obsahu

Obsah prerekvizity všichni mohou „teoreticky“ znát z programování. Zopakuje binární kódování celých čísel bez znaménka a se znaménkem, tedy typy *unsigned integer* a *signed integer*, a jejich přetečení při sčítání nebo odčítání. Dále si připomeneme logické a aritmetické posuny doleva a doprava, které v logice patří mezi velmi důležité operace. V jazycích C, C# a Java je částečně umí bitové operátory `<<` a `>>`.

Na závěr si zopakujeme hexadecimální notaci, důležitá BCD čísla a ASCII kódování.

Jazyková poznámka

Většina odborné literatury se dnes píše v angličtině, a tak k řadě pojmů neexistují ani ustálené české ekvivalenty. Nevymýšlel jsem si je, akorát bych tím ztížil čtení cizojazyčných publikací. Radši jsem stylisticky hřešil a vkládal jsem anglické termíny psané kurzívou. Zmiňuji i jejich české překlady, pokud se mi podařilo dohledat nějaký trochu zavedený název.

Objeví se žel i podivná spojení typu na Obrázek 7. Kvůli značnému množství obrázků se nevyužil LaTeX, ale MS-Word, který nedovoluje u všech typů odkazů vkládat jenom jejich číslo.

Historie textu

Prerekvizita patřila původně do učebnice APOLOS, jejíž název vznikl spojením zkratk předmětu Architektura počítačů (APO) a Logické obvody a systémy (LOS), který byl předchůdcem dnešních Logických systémů procesorů (LSP). Obsahovala velmi stručný přehled logiky a binární čísla

K předmětu LOS, později LSP, jsem tvořil další učebnice. V roce 2022 jsem dokončil text Logické obvody na FPGA, v němž jsem výrazně rozšířil APOLOS pasáž o logice. Pojednání o binárních číslech jsem pak z APOLOS vyjmul a po drobných adaptacích jsem ho uložil jako Binární prerekvizitu.

Navazující učebnice pro předmět Logické systémy a procesory

Architektura počítačů má k dispozici vynikající učebnici „*David A. Patterson and John L. Hennessy. 2013. Computer Organization and Design*“.

I k logice existuje řada materiálů, avšak často referují k starším technikám návrhům. Nenašlo se nic uceleného, v čemž by se komplexně shrnovaly znalosti potřebné ke správnému návrhu obvodů současnými prostředky. A začínající návrháři přece jenom mohou obtížněji rozlišovat, co dnes pořád platí, jaké konstrukce se stále dají aplikovat po menších úpravách a které zběsilosti se už nesmí na moderních prostředcích zapojovat, ač byly kdysi velmi oblíbené a doporučované, a tak na webech existují četná jejich schémata.

Rozhodl jsem tedy dát průchod své spisovatelské vášni:-) a vytvořit aktualizované učebnice. Psal jsem je v češtině, z té jsem je překládal do angličtiny. Postupně tak vznikala pětice skript:

0. **Binární prerekvizita**, kterou právě čtete, tvoří vstupní portál.
1. **Logické obvody na FPGA** objasní základní logické konstrukce, bez nichž lze těžko cokoliv efektivně navrhovat. Jak název napovídá, zmiňuje se tady sice dost obecných částí, ale uvedou se též verze zapojení při jejich realizaci na moderní programovatelné logice FPGA — zkratka bude vysvětlená v učebnici.
2. Následuje díl o jazyku **VHDL stylu concurrent**. Profesionální návrhy obvodů se dnes píše v jazycích HDL (*Hardware Description Language*), k nimž patří i VHDL a Verilog, dnes nejčastěji používané. K výuce se vybralo VHDL na základě doporučení vývojových firem, které ho označily za vhodnější k počátečním krokům. Nicméně, zná-li někdo striktní VHDL, v němž se dovolí minimum kódových prohřešků, snadno ovládne i frivolnější Verilog, aspoň dle znalců z praxe, což prý opačně většinou neplatí.

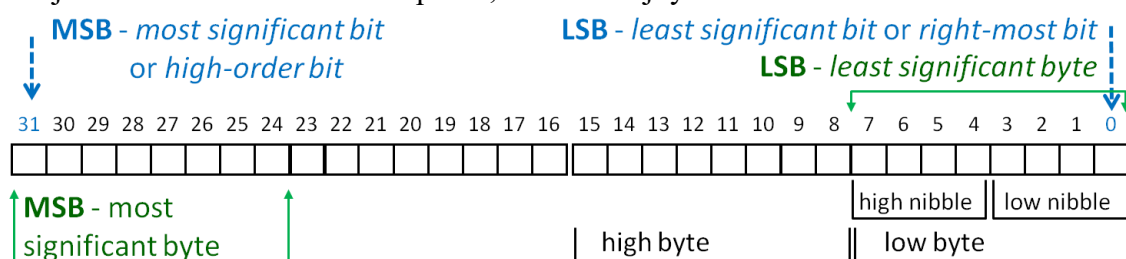
Zdrojový kód VHDL zahrnuje dvě domény. První se nazývá *concurrent* (souběžná) a její příkazy lze přímo převádět na zapojení obvodu. Druhá kódová doména se píše stylem *behavioral* a překladač ji napřed konvertuje na meta-schéma stylu *concurrent*, až z něho sestaví obvod. Třeba tedy napřed vědět, na co se *behavioral* příkazy převádějí.
3. Předposlední učebnice, **VHDL stylem behavioral**, je dosud dopsaná na 75 %, osvětluje druhou kódovou doménu VHDL, v níž se napodobují některé konstrukce programovacích jazyků, avšak musíme je psát v příhodné formě, aby vznikl použitelný obvod.
5. Závěrečná část **Speciální obvody**, z níž zatím existují jen fragmenty, zahrne pokročilejší zapojení jako třeba zpětnovazební posuvné registry, konečné automaty a využití pamětí. Přidá i fázové závěsy, nábojové pumpy a ošetření vstupů a výstupů. Rozšíří také znalosti o CMOS hradlech. Při prvotním studiu ji půjde vynechat bez ztráty souvislostí.

2 Kódování čísel

Možná jste někde už slyšeli následující vtip, či nějakou jeho modifikaci:

*Po autohavárii mi programátor podepsal náhradu škody 1000 Kč.
Zaplatil deset korun s tím, že mi dává dvě koruny navíc.*

Pointa se opírá o fakt, že zápis 1000 znamená 8 dekadicky ve dvojkové (binární) soustavě. Může, ale nemusí. Hodnota závisí na způsobu kódování čísel a bitové délce čísla. Programovací jazyky používají různé délky binárních čísel, ale zpravidla jen násobky délky bytu, tedy 8 bitů. Nejnižší bit se v nich nachází vpravo, zatímco nejvyšší vlevo.



Obrázek 1 - Byte, bit, MSB, LSB

Samotné slovo "bit" pochází angličtiny a znamená trošku, špetku. Někde se používá i pojem "nibble" pro čtveřici bitů, což v angličtině znamená ždíbec, kousíček. Samotný byte vznikl pak jazykovou alternací od anglického "bite", tedy sousto. V literatuře se někdy pro byte používá výjimečně i jiný pojem, a to "octet", neboli osmice.

Binární číslo může mít v logických obvodech libovolnou kladnou délku, neboť v nich není omezení na celý počet bytů. Nejnižší bit binárního čísla může ležet jak vpravo (jako v brázku nahoře), tak vlevo. I když se i v obvodech se upřednostňuje klasické počítačové uspořádání s nejnižším bitem vpravo. Pro označení pořadí se používají zkratky:

MSB ve významu "most significant bit" nebo "high-order bit", tedy nevýznamnější bit.

MSB se používá i pro uspořádání bytů jako "most significant byte".

LSB má opačný význam "least significant bit" nebo "right-most bit", nejméně významný bit. *LSB* se opět používá i pro uspořádání bytů jako "least significant byte".

Rozlišení, zda se *MSB* či *LSB* vztahuje k bitu či bytu musí vyplynout z kontextu popisu.

Pojem word - (slovo) označuje nativní binární formát daného počítače. Na dnešních 32bitových procesorech je *word* 32bitů, na 64bitovém procesoru je 64 bitů. Rozhodně *word* není vždy a všude 16bitové binární číslo, jak se někde mylně uvádí¹. Například při letu na Měsíc měly "Apollo Guidance Computers" 15-bitové *word*.

Binární číslo je pouhá posloupnost 1 a 0 a o jeho dekadické hodnotě lze rozhodnout až na základě specifikace způsobu použitého pro zakódování dekadických čísel. Používá se několik různých způsobů. Ty nejčastější rozebereme v následujících kapitolách.

¹ Jistou výjimkou velikosti *word* jsou některé průmyslové programovací jazyky pro PLC (programovatelné automaty), pro něž je šířka *word* definovaná normou IEC 1131-3. V té je pojem WORD zavedený jako 16bitový typ a z toho odvozené DWORD (double word) pro 32bitový typ a LWORD (long word) pro 64bitový typ. Norma se ovšem vztahuje jen k PLC, jinde neplatí.

2.1 Binární číslo *unsigned* - bez znaménka

Binární čísla *unsigned*, celým anglickým názvem: *binary encoded unsigned integers*, představují základní binární čísla. Jejich princip se opírá o matematický fakt, že v řadě 2^N mocnin dvou je součet všech předchozích členů řady vždy o jedničku nižší než následující člen řady. Například součet prvních 4 členů řady je $2^0+2^1+2^2+2^3 = 1+2+4+8 = 15 = 2^4-1$. Obecně platí:

$$2^n - 1 = \sum_{k=0}^{n-1} 2^k \quad (1)$$

Každé celé nezáporné číslo můžeme vyjádřit jako součet vybraných členů řady 2^N . Bude-li nějaký člen příslušné mocniny použitý, zapíšeme bit 1, jinak 0. Řetězec m bitů $x \approx b_{m-1} b_{m-2} \dots b_1 b_0$ nazveme binárním číslem bez znaménka, dále jen binární číslo *unsigned*, a přiřadíme mu hodnotu:

$$x = \sum_{k=0}^{m-1} b_k 2^k \quad (2)$$

Například řetězec 1100100 se jako binární číslo *unsigned* převede na dekadickou hodnotu 100:

$$\begin{array}{ccccccc} 1*2^6 & + & 1*2^5 & + & 0*2^4 & + & 0*2^3 & + & 1*2^2 & + & 0*2^1 & + & 0*2^0 & = \\ \mathbf{64} & & \mathbf{32} & & & & & & \mathbf{4} & & & & & \mathbf{=100} \end{array}$$

Vztah (1) nám zaručuje, že ke každému nezápornému celému číslu existuje právě jedna kombinace členů řady, která dává dané číslo jako svůj součet, přičemž každý člen řady se v součtu vyskytne nejvýše jen jednou. Jinými slovy, kódování je zcela jednoznačné.

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768
16	65536
17	131072
18	262144
19	524288
20	1048576

2.1.1 Změna bitové délky čísla

Hodnotu binárního čísla *unsigned*, bez znaménka, určují pouze jedničkové bity. Před číslo lze vložit libovolný počet nul, aniž se změní jeho hodnota. Například binární čísla *unsigned*: 1100100, 01100100, 001100100, 0001100100, atd. mají stejnou dekadickou hodnotu 100. Zde předpokládáme neomezenou bitovou délku, ale v praxi bývá počet bitů binárního čísla omezený, a tak můžeme samozřejmě přidávat jen tolik nul, kolik se nám vejde.

2.1.2 Logické posuny

Operace posunu doleva, (angl. *logical left shift*), je přidání bitu 0 za binární číslo *unsigned*, tj. na jeho pravou stranu. Například pro u binárního čísla 101, odpovídajícího dekadicky číslu 5 (2^2+2^0), po posunu doprava dostaneme 1010, které má dvojnásobnou dekadickou hodnotu, tj. 10. Každý jedničkový bit se totiž posunul pod následující vyšší člen mocninné řady s dvojnásobnou hodnotou. Podobně binární číslo bez znaménka 10100 s přidanou dvojicí bitů má čtyřnásobnou dekadickou hodnotu, tj. 20, a 101000 má osminásobnou hodnotu 40.

	$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
$8*5=\mathbf{40}$	$=32+8$	1	0	1	0	0
$4*5=\mathbf{20}$	$=16+4$		1	0	1	0
$2*5=\mathbf{10}$	$=8+2$			1	0	1
$\mathbf{5}$	$=4+1$				1	0

Je-li bitová délka čísla omezená, pak hodnota bude dvojnásobkem jen tak dlouho, dokud se nám levý 1 bit nedostane na konec délky pro uložení čísla.

Máme-li například binární číslo 101 uloženo na 8 bitů, tj. jako 00000101, pak jeho hodnota se při prvních 5 posunech vlevo vždy zdvojnásobí, tj. až k hodnotě 10100000, která odpovídá dekadickému číslu $5 * 2^5 = 5*32=160$. Další posun vlevo by v 8bitové délce vedl na 01000000, což jako binární číslo *unsigned* je dekadicky 64. Nejvyšší bit 1 se ztratil díky omezené délce čísla. Došlo zde k aritmetickému přetečení, blíže viz kapitola 2.1.5.

V programovacích jazycích, na bázi jazyka C, se operátor bitového posunu vlevo zapisuje <<, za nímž následuje délka posunu v bitech². Máme-li proměnnou byte $x = 5$; (v jazyce C *unsigned char x=5*;) , pak platí: $2 * x == (x << 1)$ a $4*x == (x << 2)$ až $32*x == (x << 5)$.

Opačná operace logického posunu doprava, (angl. *logical right shift*), pro binární čísla *unsigned* odpovídá operaci celočíselného dělení 2, neboť bity 1 se dostávají k předchozím členům mocninné řady s poloviční hodnotou. Při celočíselném dělení dostaneme výsledek a zbytek po dělení. Posuneme-li binární číslo *unsigned* 101 o jeden bit doprava, dostaneme 10, nejnižší bit 1 nám vypadl, je zbytkem po dělení.

		$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$	
	5	=4+1			1	0	1	
	$5/2 = 2$	=2				1	0	-> 1
	a zbytek po dělení 1							

V jazyce C existuje částečná implementace posunu operátorem >>, který nedává zbytek po dělení, jen podíl. Pro proměnnou byte $x = 5$; platí: $(x \% 2) == 1$ a $x/2 == (x >> 1)$ a $x/2 == 2$.

Překladač programovacích jazyků často využívají posuny k překladač celočíselného násobení a dělení konstantami rovnými mocninám 2. Posuny jsou totiž velmi rychlé operace.

2.1.3 Převod binárního čísla bez znaménka na dekadické číslo

Postup 1: Dekadická hodnota binárního řetězce, který bereme jako binární číslo *unsigned*, je určena přímo součtem odpovídajících členů řady 2^N , kde N je číslo bitu.

Máme-li tedy binární řetězec $X=10011$, který má jedničky na pozicích 4., 1. a 0. bitu, pak můžeme stanovit jeho hodnotu pouhým součtem odpovídajících členů:

$$X=10011 \rightarrow 2^4 + 2^1 + 2^0 = 16 + 2 + 1 = 19$$

Pokud je binární řetězec delší a s více jedničkami, jako například $Q=11111110110$, pak jeho převod na číslo bez znaménka pomocí sčítání by byl trochu náročnější, protože Q má jedenáct bitů a většina z nich jsou jedničky:

bit	10	9	8	7	6	5	4	3	2	1	0
Q	1	1	1	1	1	1	1	0	1	1	0

Výpočet si můžeme zkrátit uplatněním zmíněné vlastnosti řady mocnin dvou, a to, že hodnota následujícího členu řady 2^N je vždy o jedničku vyšší než součet všech předešlých členů. Víme tedy, že

$$2^{11}-1 = 2048-1 = 2047 = 2^{10}+2^9+2^8+2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0$$

Číslo Q vede na posloupnost sčítanců z řady 2^N , ve které nejsou jen dva členy 2^3 a 2^0 , neboť binární řetězec Q má nulové bity na 3. a 0. pozici. Z toho plyne

$$Q = 11111110110 \rightarrow 2047 - 2^3 - 2^0 = 2047-8-1 = 2038$$

² V jazyce C++ bývají operátory << a >> přetížené, např. vložení `iostream.h`, a používají se pro čtení a zápis z/do datových proudů (angl. *streams*). Pro číselné operátory se i tak pořád chovají jako posuny.

Postup 2: Pro převod můžeme využít i operaci logického posunu doleva, jímž provádíme výpočet polynomu podle Hornerova schématu (pod tímto názvem k nalezení na Wikipedii).

Postupujeme od nejvyššího bitu. Vezmeme ho a zapíšeme jako výsledek, tedy 1 nebo 0. Pokud za ním existuje další bit, vynásobíme výsledek dvěma a přičteme k němu hodnotu dalšího bitu, tedy 1 nebo 0. Postup opakujeme, dokud se nedostaneme k nejnižšímu bitu.

$$10011$$

$$1 \rightarrow 1*2+0=2 \rightarrow 2*2+0=4 \rightarrow 4*2+1=9 \rightarrow 9*2+1=19$$

Jiný příklad již ve zkráceném zápisu

$$11111110110$$

$$1 \rightarrow 2+1=3 \rightarrow 6+1=7 \rightarrow 14+1=15 \rightarrow 30+1=31 \rightarrow 62+1=63 \rightarrow 126+1=127 \dots$$

$$\rightarrow 254+0=254 \rightarrow 508+1 \rightarrow 1018+1 = 1019 \rightarrow 2038+0=2038$$

Postup 2 nedoporučujeme pro ruční výpočet, a to na základě zkušeností. Střídají se při něm operace násobení a sčítání, takže není zcela mechanický a snadno se v něm udělá početní chyba. Hodí se však pro algoritmizaci převodu, více dále u BCD čísel, kapitola 2.5.2, str. 26.

2.1.4 Převod dekadického čísla na binární číslo bez znaménka

Pro jednoduché převody lze použít buď postupné odčítání, nebo dělení 2.

2.1.4.1 Postupné odčítání

Při postupném odčítání najdeme v řadě mocnin 2 největší člen řady, který je stále menší než převáděné číslo. Například při převodu čísla 35, vybereme 2^5 , tedy 32. Od něho začneme postupné odčítání, dokud nedostaneme nulu.

číslo	odečítaný člen řady	výsledek	bit	
35	-32	3	1	MSB
3	-16	nelze	0	
3	-8	nelze	0	
3	-4	nelze	0	
3	-2	1	1	
1	-1	0	1	LSB

Pokud je člen řady menší, tak zapíšeme bit 1, hodnotu členu odečteme, v opačném případě píšeme 0 a zkusíme další člen. Výsledkem převodu dekadického čísla 35 na binární číslo bez znaménka bude binární číslo 100011.

Postupné odčítání vyžaduje znalost členů řady 2^N . Několik prvních členů řady není těžké si zapamatovat, ale větší dekadická čísla se pohodlněji převedou postupným dělením.

2.1.4.2 Převod dělením 2

Princip převodu vychází z posunu doprava, viz kapitola 2.1.3 na str. 9. Číslo postupně celočíselně dělíme 2, dokud nedostaneme 0, a zapisujeme zbytky po dělení jako binární číslo. Ty píšeme od nejnižšího bitu k vyššímu, protože při posunu doprava nám vypadávají nejnižší bity. Při převodu čísla většího než 0, dostaneme nakonec vždy 1 celočíselně děleno 2, a to s výsledkem 0, a zbytkem po celočíselném dělení 1.

$35 / 2 = 17$ zbytek po celočíselném dělení **1** - nejnižší bit
 $17 / 2 = 8$ zbytek po celočíselném dělení **1**
 $8 / 2 = 4$ zbytek po celočíselném dělení **0**
 $4 / 2 = 2$ zbytek po celočíselném dělení **0**
 $2 / 2 = 1$ zbytek po celočíselném dělení **0**
 $1 / 2 = 0$ zbytek po celočíselném dělení **1** - nejvyšší bit

Dělení není nutné tak podrobně rozepisovat, stačí nám pořad celočíselně dělit 2 a zbytky po dělení zpětně stanovit podle toho, které dílčí podíly byly liché - ty měly zbytek po dělení 1.

Převedeme například dekadické číslo 1000 na binární číslo *unsigned*. V následující řádce \rightarrow značí, že další číslo bylo odvozeno celočíselným dělením 2:

1000 \rightarrow **500** \rightarrow **250** \rightarrow **125** \rightarrow **62** \rightarrow **31** \rightarrow **15** \rightarrow **7** \rightarrow **3** \rightarrow **1** \rightarrow **0**

Zapišeme-li liché podíly jako bity 1 a ostatní jako 0, pak dostaneme binární číslo **1111101000**. Bity jsme řadili od nejnižšího, čili v opačném pořadí, než je řada podílů dělení. Všimněte si, že by vůbec nevadilo, kdybychom zahrnuly i poslední $\rightarrow 0$, protože bychom dostali binární číslo **01111101000**, které má stejnou hodnotu, viz odstavec 2.1.1.

2.1.5 Aritmetické přetečení při sčítání a odčítání

V počítačích i logických obvodech se vždy ukládá jen konečný počet bitů. Pokud k číslu přičítáme 1, pak časem dosáhneme maximální hodnoty, při níž číslo bez znaménka je reprezentováno samými bity 1, v počtu zvolené délky pro uložení čísla.

	Carry	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
254		1	1	1	1	1	1	1	0
+1									1
255		1	1	1	1	1	1	1	1
+1									1
0	1	0	0	0	0	0	0	0	0
+1									1
1		0	0	0	0	0	0	0	1

Tabulka 1 - Přičítání 1 k 8bitovému číslu bez znaménka

Maximální 8bitové binární číslo *unsigned* je 11111111, což je dekadicky 255. Když k němu přičteme 1, dostaneme binární číslo *unsigned* = 100000000, které správně odpovídá 2^8 tedy 256, ale má devět bitů. Do 8bitového čísla se nám uloží jediné jeho spodních osm nulových bitů. Nejvyšší 9. bit se musí zahodit, takže náš výsledek bude ve skutečnosti 0.

Na 8bitové aritmetice binárních čísel *unsigned* bude tedy $255+1 = 0$, ale samozřejmě, kdybychom použili delší binární číslo, pak by platilo $255+1=256$.

Ztracený nejvyšší bit se nazývá Carry, neboli přenos do vyššího řádu. Pro aritmetiku binárních čísel *unsigned* ohlašuje chybu aritmetického přetečení, tedy překročení maximální zobrazitelné hodnoty pro zvolenou bitovou délku.

K přetečení dojde i při odečítání 1, což si můžeme představit jako postup odspodu nahoru v Tabulka 1. Pak operace odečtení 1 od 0 zde dává výsledek 255. V logických obvodech se někdy tento opačný směr, přetečení z 0 na maximum, nazývá *Borrow*, protože si při něm půjčujeme bit z vyššího řádu.³ V mnohé literatuře se však oba směry nazývají *Carry*.

³ Většina procesorů nerozlišuje směr přetečení a jejich ALU v obou případech generuje *Carry*. Zda šlo o *Carry* či *Borrow* se stanoví jen dle právě provádění instrukce assembleru. Sčítání - *Carry*, odčítání - *Borrow*.

Odečteme-li od 0 číslo 1, pak vždy dojde k přetečení a celé číslo se vyplní bity 1, což je maximální hodnota. Bitová délka výsledku určí jedinečně to, jakou dekadickou hodnotu bude chybný výsledek operace 0-1 představovat, například:

pro 8bitové číslo bez znaménka bude $(0-1) = 2^8-1 = 255$,

pro 9bitové číslo bez znaménka bude $(0-1) = 2^9-1 = 511$

pro 16bitové číslo bez znaménka bude $(0-1) = 2^{16}-1=65535$, atd.

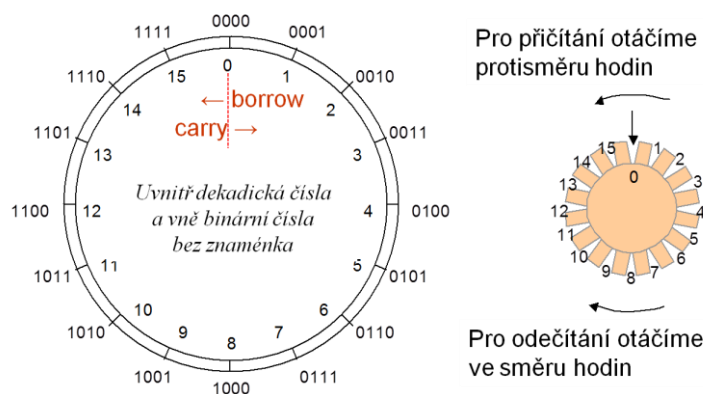
Výsledek odčítání 0-1 je při běžném počítání -1. Hodnota výsledku operace s binárními čísly *unsigned* omezené délky se stanovila pomocí korekce 2^m , kde m je počet bitů zvolený pro uložení čísla. Při výsledku menším než nula připočítáváme 2^m , dokud nedostaneme kladné číslo. Pokud je výsledek naopak větší než 2^m , pak odečítáme 2^m .

Otázka 1: V 4bitové aritmetice čísel *unsigned*, jaká bude dekadická hodnota výsledku operace dvou dekadických čísel při jejich sečtení 14+4 a při odečtení 4-14 ?

Výpočet: Vypočteme $14+4 = 18$. Výsledek je větší než $2^4=16$, a tak korekce je $18-16=2$.

Vypočteme $4-14 = -10$. Výsledek je menší než 0, a tak korekce je $-10+16 = 6$.

Předchozí výpočet si můžeme názorně zobrazit, pokud si binární čísla *unsigned* představíme jako kolo opatřené číslicemi, viz Obrázek 2. Operace sčítání odpovídá otáčení kola proti směru hodinových ručiček a odčítání zase otáčení kola ve směru hodinových ručiček. Počet zubů, o něž kolo otočíme, je dán hodnotou čísla, které přičítáme, či odčítáme. Na obrázku vidíme, že číslo 14 leží o 4 pozice proti směru hodinových ručiček od čísla 2 ($14+4=2$) a číslo 4 leží o 14 pozic ve směru hodinových ručiček od čísla 6 ($4-14=6$).



Obrázek 2 - Přičítání a odčítání 4bitových čísel bez znaménka

Otázka 2: V 8bitové aritmetice binárních čísel *unsigned*, jaká bude dekadická hodnota výsledku sčítání dvou dekadických čísel 200 a 100?

Výpočet: Sečtu obě dekadická čísla, tedy $200+100 = 300$. Výsledek je větší než $2^8 = 256$. Odečtu korekci 2^8 : $300-256=44$. Výsledek operace bude tedy 44.

Otázka 3: V 10bitové aritmetice binárních čísel *unsigned*, jaká bude dekadická hodnota výsledku operace odčítání dvou dekadických čísel 1000-1500?

Výpočet: Vypočtu rozdíl dekadických čísel $1000-1500 = -500$. Výsledek je menší než 0, a tak k němu přičtu $2^{10} = 1024$. Tedy $-500+1024=524$. Výsledek bude tedy 524.

Otázka 4: V 5bitové aritmetice binárních čísel *unsigned*, jaká bude dekadická hodnota výsledku sčítání dvou dekadických čísel 10 a 20?

Výpočet: Vypočtu $10+20=30$. Výsledek je kladný a menší než $2^5=32$. Korekce není třeba.

2.2 Binární číslo *signed* - se znaménkem ve dvojkovém doplňku

Pro celá čísla se znaménkem existuje několik různých kódování, z nichž se nejvíce používá dvojkový doplněk, založený na aritmetickém přetečení, viz kapitola 2.1.5.

Pokud k binárnímu číslu x uloženému na m bitů vytvoříme **jedničkový doplněk** χ jako negaci všech jeho bitů, tak součet $x + \chi$ bude binární číslo se všemi bity v hodnotě 1, neboť χ má bity 1 jediné tam, kde x má bity 0.

Součet $x + \chi = 2^m - 1$ je maximální binární číslo *unsigned*. Přičteme-li 1 k χ , pak dostaneme $x + (\chi + 1) = 2^m$. Výsledek 2^m je binární číslo *unsigned* o délce $m+1$ bitů. Do m bitů se uloží jen jeho nulové spodní bity. Pro m -bitová čísla *unsigned* tedy platí, že $x + (\chi + 1) = 0$.

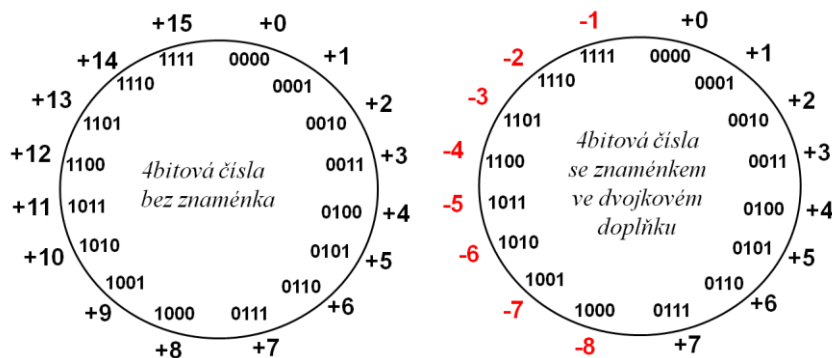
Hodnota $(\chi + 1)$ nazývá **dvojkový doplněk k číslu x** (*two's complement of x*), tedy podle výše zmíněné vlastnosti.

Například, u binárních čísel délky 4 bity je dekadické číslo 4 uloženo jako binární číslo *unsigned* 0100. Jeho jedničkový doplněk (bitová negace) je 1011 (χ). Přičteme 1 (binárně 0001), dostaneme 1100 ($\chi + 1$), což je dvojkový doplněk 4bitového binárního čísla 0100. Součet $0100 + 1100 = 10000$. Výsledek 10000 vzatý jako binární číslo *unsigned* se dekadicky rovná 16, avšak 10000 má 5bitovou délku. Do 4bitového binárního čísla se nám uloží jeho spodní 4 bity, takže dostaneme výsledek 0000. Došlo zde k aritmetickému přetečení.

Čísla se znaménkem v dvojkovém doplňku, dále jako typ *signed*, zavedeme takto:

- dvojkový doplněk binárního čísla prohlásíme za negaci původního čísla,
- dále stanovíme, že binární číslo délky m bitů, které má bit 1 ve svém nejvyšším bitu (tj. v bitu s váhou 2^{m-1}), bude obrazem záporného dekadického čísla.

Pro 4bitovou aritmetiku jsou čísla *signed* uvedena na Obrázek 3 vpravo. Binární číslo 1000, mající dekadickou hodnotu -8, má výlučné postavení. I k němu existuje dvojkový doplněk, ale jím se binární číslo 1000 samo. Jelikož to má 1 v nejvyšším bitu, reprezentuje dekadické číslo -8, ale už zde nemá kladný dekadický protějšek. Tato nesymetrie je jediným nedostatkem zakódování binárních čísel *signed* (se znaménkem ve dvojkovém doplňku).



Obrázek 3 - 4bitová čísla *unsigned* a *signed*

Zobrazení čísel *signed* jinak nabízí samé výhody. Operace sčítání a odčítání se počítají jako v případě čísel *unsigned* (binárních čísel bez znaménka). Můžeme tedy používat stejnou výpočetní jednotku pro obě reprezentace čísel. Závisí jen na nás, zda interpretujeme výsledek operace jako binární číslo *unsigned* nebo *signed*. Navíc kladná čísla se zobrazí stejně jako čísla bez znaménka, pouze pro záporná čísla se používá dvojkový doplněk.

Pro výše uvedené výhody jsou čísla typu *signed* (se znaménkem ve dvojkovém doplňku) hlavním způsobem pro uložení celých čísel se znaménkem (typ *signed integer*).

2.2.1 Významné vlastnosti

Binární čísla typu *signed* mají významné vlastnosti, které stojí za zapamatování.

- Číslo 0 se zobrazí jako samé bity 0.
- Označme bitovou délku m , pak dekadická čísla v rozsahu -2^{m-1} do $2^{m-1} - 1$ lze převést, ostatní čísla leží mimo rozsah. Například pro $m=4$ je to -2^3 až 2^3-1 , tedy -8 až 7.
- Binární číslo *signed* mající 1 a za ní $m-1$ bitů 0 je vždy nejmenším číslem a jeho hodnota se rovná -2^{m-1} . Toto číslo je současně jedinou anomálií — neexistuje k němu kladné číslo. Například, máme-li tedy 8bitová čísla, pak nejmenší binární číslo je 10000000 a jeho dekadická hodnota je $-2^{8-1} = -2^7 = -128$.
- Binární číslo *signed* mající 0 následovanou $m-1$ jedničkami je vždy největším číslem a jeho hodnota se rovná $2^{m-1} - 1$. Například, máme-li tedy 8bitová čísla, pak největší binární číslo *signed* je 01111111 a jeho dekadická hodnota je $2^{8-1} - 1 = 2^7 - 1 = 127$. *Poznámka: Dekadické číslo -127 bude převedeno na 10000001 - je o 1 vyšší než -128.*
- Binární číslo *signed* mající samé bity 1, tedy m bitů 1, se vždy rovná dekadickému číslu -1. Například pro 8bitová čísla je to 11111111. *Poznámka: Dekadické číslo -2 bude převedeno na 11111110, je o 1 menší než -1.*

2.2.2 Aritmetická negace pomocí dvojkového doplňku

Mějme binární číslo *signed* (se znaménkem ve dvojkovém doplňku) o známé bitové délce m . K němu najdeme záporné číslo (aritmetická negace) **algoritmem dvojkového doplňku**:

- a) logicky negujeme všechny bity binárního čísla, tzv. jedničkový doplněk.
- b) k jedničkovému doplňku přičteme binárně 1, čímž obdržíme dvojkový doplněk původního čísla.

Příklad 1: Pro 8bitové binární číslo *signed* 01100100, mající dekadickou hodnotu 100, vypočítejte jeho aritmetickou negaci.

Výpočet: Spočteme jedničkový doplněk negací bitů 01100100 → 10011011. Přičteme k němu 1, tedy $10011011 + 00000001 = \mathbf{10011100}$.

Příklad 2: Proveďte aritmetickou negaci pro 10bitové číslo *signed* 1000000000, (dekadicky -512).

Výpočet: Zadáni je chyták. Správná odpověď je: **nelze**, viz významné vlastnosti výše.

2.2.3 Převod dekadických čísel na binární

Pro převod dekadických čísel musíme vždy znát bitovou délku požadovaného binárního čísla *signed*. Tu opět označme jako m .

- Kladná čísla vyhovující rozsahu čísel, viz odstavec 2.2.1, převádíme stejně jako čísla bez znaménka.
- Záporná dekadická čísla konvertujeme jedním z následujících postupů, v nichž označíme vstupní záporné dekadické číslo jako $-x$
 - a) převedeme absolutní hodnotu $-x$, tedy $|-x|$, jako číslo bez znaménka a k němu vypočteme dvojkový doplněk. Nevýhodou tohoto způsobu je nutnost binárně přičíst 1 při výpočtu dvojkového doplňku.

- b) binárnímu sčítání se vyhneme, pokud převedeme absolutní hodnotu dekadického čísla zmenšenou o 1, tedy převedeme $|-x|-1$ na číslo bez znaménka a pro něj uděláme jen jedničkový doplněk, logickou negaci bitů. Výsledek bude rovný binárnímu číslu $-x$.
- c) alternativně můžeme převést $(2^m - x)$ na číslo bez znaménka. Výsledek bude rovný číslu $-x$. Tento způsob využívá přímo způsobu, jakým byla čísla se znaménkem ve dvojkovém doplňku zavedena.

Jak si ověříme, že si postup pamatujeme? Stačí znát obraz aspoň jednoho čísla, třeba dekadické -1 převáděné na samé bity 1. Poté si napřed zkuste převést zvoleným postupem ono známé číslo. Pokud dostanete správný výsledek, je i způsob převodu správný.

Příklad: Převed'te číslo -12 na 8bitové binární číslo *signed* (binární číslo se znaménkem ve dvojkovém doplňku).

- Výpočet podle a) Absolutní hodnota $|-12| = 12$. Převedeme 12 jako 8bitové binární číslo *unsigned* na 00001100. Vypočteme jeho jedničkový doplněk 11110011 a k němu přičteme 1, tedy $11110011 + 00000001 = \mathbf{11110100}$.
- Výpočet podle b): $|-12|-1 = 11$. Dekadické 11 číslo bude jako 8bitové binární číslo *unsigned* = 00001011. Od něho jedničkový doplněk (bitová negace) je 00001011 $\rightarrow \mathbf{11110100}$.
- Výpočet podle c): $2^8 - 12 = 256 - 12 = 244$. Dekadické číslo 244 jako 8bitové binární číslo *signed* je 11110100. A toto binární číslo **11110100** má dekadickou hodnotu, pokud ho bereme jako binární číslo *signed*, rovnou -12 .

2.2.4 Převod binárních čísel na dekadická

Pro převod binárních čísel musíme opět znát bitovou délku požadovaného binárního čísla *signed*. Tu si znovu označme jako m .

- Binární čísla mající 0 v nejvyšším bitu, tedy kladná, se opět převádí na kladná dekadická čísla stejně jako čísla bez znaménka.
- Záporná binární čísla, tj. mající 1 v nejvyšším bitu, konvertujeme dále uvedenými postupy, které jsou opačnými k předchozím metodám v odstavci 2.2.3.
 - a) vypočteme dvojkový doplněk binárního čísla a ten převedeme jako binární číslo *unsigned* na hodnotu, kterou označíme x . U ní změním znaménko na mínus, tedy na $-x$.
 - b) můžeme také udělat jen jedničkový doplněk, logickou negaci bitů, a výsledek převést na číslo bez znaménka, které označíme y . Hledané dekadické číslo $-x$ se rovná: $-x = -y - 1$.
 - c) alternativně můžeme převést celé binární číslo na číslo jako číslo bez znaménka, výsledek označíme z . Hledané číslo $-x = z - 2^m$.

Příklad: Převed'te 000111000 jako 9bitové binární číslo *signed*.

Výpočet: Nejvyšší bit je 0, takže binární číslo konvertujeme jako číslo bez znaménka, třeba sečtením vah jedničkových bitů na $2^5 + 2^4 + 2^3 = 32 + 16 + 8 = 56$.

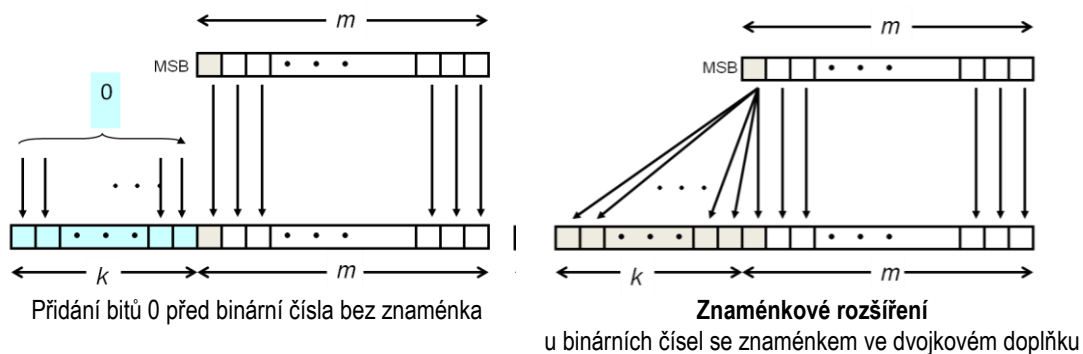
Příklad: Převeďte 8bitové binární číslo 11001100 jako binární číslo *signed*.

Výpočet: Nejvyšší bit je 1, takže převádíme metodami pro záporná čísla.

- Výpočet podle a) K binárnímu číslo 11001100 vypočteme jeho dvojkový doplněk $00110011 + 00000001 = 00110100$. Ten převedeme jako binární číslo *unsigned* na 52. Hledané číslo je pak jeho negací, tedy **-52**.
- Výpočet podle b): Provedeme bitovou negaci (jedničkový doplněk) čísla 11001100 $\rightarrow 00110011$ a výsledek převedeme jako binární číslo *unsigned* $00110011 \rightarrow 51$. Hledaný výsledek je $-51 - 1 = -52$.
- Výpočet podle c): Číslo 11001100 převedeme jako číslo bez znaménka třeba sčítání vah bitů jako $128 + 64 + 8 + 4 = 204$. Hledané číslo je $204 - 2^8 = 204 - 256 = -52$.

2.2.5 Změna délky čísla - sign extension

Před binární číslo *unsigned* lze přidat bity 0, aniž se změní jeho hodnota, viz 2.1.1. Pro zachování hodnoty čísla *signed* se musí použít znaménkové rozšíření (angl. *sign extension*), při němž se zachovává nejvyšší bit určující, zda je číslo kladné nebo záporné.



Obrázek 4 - Znaménkové rozšíření pro binární čísla *signed*

Obrázek 4 ukazuje rozdíly mezi oběma způsoby uložení binárních čísel. Zatímco u neznaménkových čísel se vždy přidávají bity 0, u binárních čísel *signed* se do přidaných bitů kopíruje nejvyšší bit čísla, tj. *MSB*.

Naopak, zmenšujeme-li binární číslo *unsigned*, lze odstranit všechny levé bity 0, ale u binárních čísel *signed* lze rušit jen nejvyšší bity 0 u kladných čísel a bity 1 u záporných, ale výhradně jen v takové délce, aby se nezměnila kladnost/zápornost čísla, tj. po změně délky musí být hodnota nejvyššího bitu nového čísla shodná s nejvyšším bitem původního čísla.

Příklad 1: Rozšiřte 4bitové číslo *signed* 0111 na 8 bitů.

Řešení: Nejvyšší bit, tj. 0, se kopíruje do bitů vlevo, tedy výsledek 0000 0111.

Příklad 2: Rozšiřte 8bitové číslo *signed* 1000 0010 na 16 bitů.

Řešení: Opět kopírujeme $MSB=1$ do přidaných bitů, tedy výsledek 1111 1111 1000 0010.

Příklad 3: Jaký je nejmenší možný počet bitů pro 8bitové číslo *signed* 1110 0100?

Řešení: Nejmenší možná bitová délka pro jeho uložení je 6 bitů, tedy 10 0100.

Příklad 4: Jaký je nejmenší možný počet bitů pro 8bitové číslo *signed* 0000 0100?

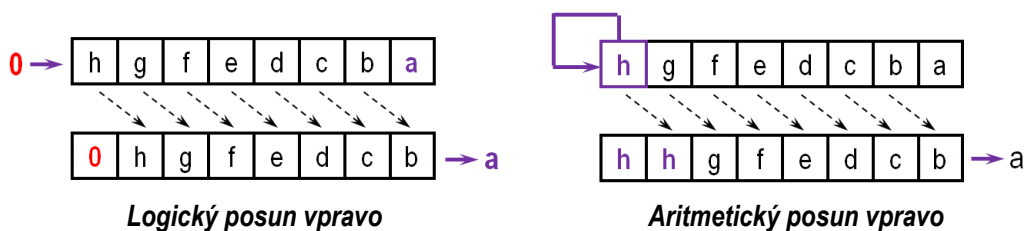
Řešení: Nejmenší možná bitová délka pro jeho uložení jsou 4 bity, tedy 0100.

Poznámka: Typ použitého čísla určuje, zda se musí provést znaménkové rozšíření. Ve strojovém kódu procesorů kvůli tomu existují odlišné instrukce pro nahrání dat menší velikosti do

větší. Například procesor MIPS nahrává neznaménkový byte do 32bitového registru instrukcí LBU, ale pro byte obsahující číslo se znaménkem ve dvojkovém doplňku má instrukci LB, která provádí znaménkové rozšíření. Procesory typu x86 zas používají instrukce MOV a MOVSX. Překladače vyšších jazyků vybírají strojové instrukce podle typů převáděných čísel.

2.2.6 Logické a aritmetické posuny

Nutnost zachovat znaménkový bit si vyžaduje i různé operace posunů (angl. *shift*) s binárními čísly. Rozlišují se logické posuny doleva a doprava, kdy se obsah posouvá jako binární číslo *unsigned*, a aritmetické posuny doleva a doprava, které respektují znaménkový bit.



Obrázek 5 - Logický a aritmetický posun vpravo

Posuny vpravo jsou naznačené na Obrázek 5 pro 8bitové binární číslo "hgfedcba", kde 'a' je nejnižším bitem a 'h' nejvyšším bitem. Pokud máme v binárním čísle formát *unsigned*, pak provádíme posun vždy s přidáváním 0. Bereme-li binární číslo jako *signed*, pak provedeme aritmetický posun, při němž zůstává jeho nejvyšší bit pevně namístě, zde 'h'.

Například řetězec 8 bitů 11101011, s nejvyšším bitem 1, se po logickém posunu změní na 01110101, zatímco po aritmetickém posunu na 11110101. Naproti tomu u 8bitového řetězce 01101010, jehož nejvyšší bit je rovný 0, logický i aritmetický posun dávají stejné výsledky 00110101.

Vstup	Hodnota jako	Posun vpravo	Výsledek posunu	Hodnota jako
11101011	<i>unsigned</i> = 235	logický	01110101	<i>unsigned</i> = 117
	<i>signed</i> = -21	aritmetický	11110101	<i>signed</i> = -11
01101010	<i>unsigned</i> = 106	logický	00110101	<i>unsigned</i> = 53
	<i>signed</i> = 106	aritmetický	00110101	<i>signed</i> = 53

Z tabulky nahoře můžeme odvodit, že logický posun doprava je vhodné použít pro binární čísla *unsigned*, u nichž odpovídá dělení 2 s tím, že nejnižší bit ztracený operací posunu doprava je zbytkem po dělení.

Aritmetický posun doprava je nutné použít u binárních čísel *signed*, aby se nám zachoval znaménkový bit. Pro kladná čísla je výsledek rovný dělení 2, a to opět se zbytkem po dělení, kterým je ztracený nejnižší bit. U záporných čísel je však podíl zaokrouhlený směrem k nižšímu číslu, tedy se provádí operace $-21/2 = \text{floor}(-10.5) = -11$, kde *floor()* označuje zaokrouhlení na celé číslo směrem dolů.⁴ Aritmetické posuny doprava interpretované jako dělení 2 dávají paradoxně třeba pro číslo -1 výsledek $-1/2 = -1$.

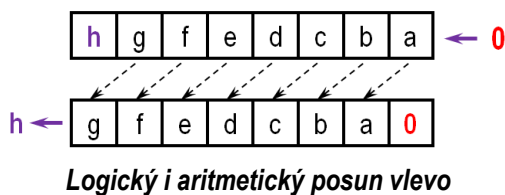
Chceme-li aritmetické posuny doprava použít místo dělení 2 pro binární čísla *signed*, pak musíme pro záporná čísla provést někdy korekci výsledku, abychom obdrželi námi oče-

⁴ V jazyce C je *floor(...)* funkce, v Java je *floor(...)* metoda, v C# zná metodu *Math.Floor(...)*

kávané číslo. Korekce je však velmi jednoduchá, přičíst 1 ve vybraných případech.⁵ Nutno si však pamatovat, že obecně nelze aritmetický posun doprava pro záporná binární čísla *signed* brát automaticky jako přesnou analogii celočíselného dělení 2.

Posuny vlevo binárních čísel jsou shodné — logický posun se provádí stejně jako aritmetický, viz Obrázek 6, kde posun je opět naznačený pro 8bitové binární číslo "hgfedcba", v němž 'a' je nejnižším bitem a 'h' nejvyšším bitem, který se při posunu vlevo ztrácí.

Pokud nedojde při posunu k aritmetickému přetečení, pak posun vlevo odpovídá násobení 2, jak pro binární čísla *unsigned*, tak pro binární čísla *signed*.



Obrázek 6 - Posun vlevo 8bitového binárního čísla

Jediný rozdíl u obou reprezentací čísel je podmínka, kdy při posunu vlevo dojde k aritmetickému přetečení, po němž hodnota výsledku už neodpovídá původnímu číslu vynásobenému 2. U binárního čísla *unsigned* nastane aritmetické přetečení v okamžiku, kdy se jeho nejvyšší bit, který se při posunu ztratil, rovnal 1. U binárních čísel *signed* ale nastane aritmetické přetečení v tom okamžiku, kdy se změní hodnota nejvyššího bitu.

Například, máme-li 8bitové binární číslo *signed* 11101011 (dekadicky -21), pak po jeho prvním posunu vlevo dostaneme 11010110 (-42) a po druhém 10101100 (-84). Provedeme-li třetí posun vlevo, pak obdržíme 01011000 (dekadicky 88), neboť pro binární číslo *signed* došlo k přetečení.

Jiný příklad: mějme 8bitové číslo *signed* 00110010 (dekadicky 50). První posun vlevo dá výsledek 01100100 (100). Další posun dá binární číslo 11001000, které má hodnotu jako 8bitové číslo *signed* -56, čili máme přetečení. Všimněte si, že při stejném vstupu, ale značícím tentokrát binární číslo *unsigned*, by výsledek byl stále dvojnásobkem předchozí hodnoty, a to 200. K přetečení by u binárního čísla *unsigned* došlo až při dalším posunu vlevo, jehož výsledek by byl 10010000. Ten má jako binární číslo *unsigned* hodnotu 144.

Poznámka: Všimněte si, že pojem aritmetického přetečení závisí na tom, jak interpretujeme binární číslo. Pokud ho bude brát jako obyčejný řetězec bitů, bez stanovené číselné hodnoty, pak posuny budou pouhou změnou pozic bitů, jakousi analogií dopravníkového pásu, který posune o jednu pozici každý bit a bit ležící na konci pásu přitom vypadne ven.

⁵ Obecně možno říct, že binární čísla *unsigned* i *signed* mohou v procesorech používat stejnou aritmetickou jednotku, což je hlavní jejich výhodou. Pouze v některých případech vyžaduje manipulace s binárními čísly *signed* nepatrně odlišné instrukce, třeba u operací násobení nebo dělení, když jsou jejich operandy záporná binární čísla *signed*. Pak se musí provést nenáročná korekce výsledku. Existují i speciální násobící algoritmy pro čísla *signed* jako "Boothův algoritmus", ale dnes se příliš nepoužívají. Našly se již lepší metody. Blíže se násobičky proberou v navazující učebnici Logické obvody na FPGA.

2.2.7 Aritmetické přetečení při sčítání a odčítání

V kapitole 2.1.5 na straně 11, kde jsme rozebírali sčítání a odčítání binárních čísel *unsigned*, jsme detekovali aritmetického přetečení pomocí *Carry*, tj. přenosu z nejvyššího řádu. *Carry* ale nemá žádný praktický význam pro binární čísla *signed*, neboť se generuje běžně; čísla sama jsou na něm založená, viz popis dvojkového doplňku v kapitole 2.2 na straně 13.

U binárních čísel *signed* dochází k aritmetickému přetečení při překročení hranice mezi jejich největším a nejmenším číslem. Například, máme-li 8bitová čísla *signed*, pak jejich největší číslo 127 je uloženo 0111 1111 a pokud k němu přičteme 1, tj. 0000 0001, pak dostaneme 1000 0000, jejich nejmenší číslo -128. Obdržíme paradoxně záporný výsledek po součtu dvou kladných čísel. Analogicky bychom při odčítání 1 od -128 dostali zase kladný výsledek 127.

	Overflow	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰
126		0	1	1	1	1	1	1	0
+1		0	0	0	0	0	0	0	1
127	0	0	1	1	1	1	1	1	1
+1		0	0	0	0	0	0	0	1
-128	1	1	0	0	0	0	0	0	0
+1		0	0	0	0	0	0	0	1
-127	0	1	0	0	0	0	0	0	1

Tabulka 2 - Aritmetické přetečení při sčítání 8bitových binárních čísel *signed*

ALU procesorů detekují podobné situace na základě tabulky znaménkových bitů operandů a výsledku. Pokud je výsledek evidentně nesprávný generují příznak (*flag*) *Overflow*.

operand 1	operace	operand 2	výsledek
kladný	+	kladný	záporný
záporný	+	záporný	kladný
záporný	-	kladný	kladný
kladný	-	záporný	záporný

Tabulka 3 - Kdy se objeví příznak *overflow* u operace s binárními čísly *signed*

Přesněji ALU nastaví po každém sčítání a odčítání binárních čísel nejméně čtyři základní aritmetické příznaky⁶ na 0 nebo na 1, podle toho, zda situace nastala:

- *Carry* - příznak přenosu z nejvyššího řádu binárního čísla, tj. ztracený nejvyšší bit⁷;
- *Overflow* - příznak přetečení pro binární čísla *signed*;
- *Sign* - kopie nejvyššího bitu čísla;
- *Zero* - pokud výsledek obsahuje samé bity 0.

Sčítání i odčítání binárních čísel *signed* a *unsigned* se provádí úplně stejně, a tak většina ALU vždy nastaví všechny zmíněné příznaky. Program si sám musí otestovat příznak, na kterém mu záleží, a to podle toho, jaký formát uvažoval u vstupních operandů.

Zmíněné hlavní aritmetické příznaky se samozřejmě generují i pro další aritmeticko-logické operace, ale u nich platí jiné podmínky; ty jsou vždy specifikované v popisu jednotlivých instrukcí procesoru.

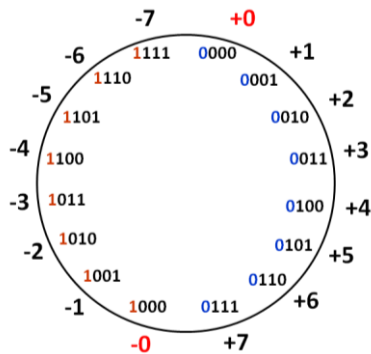
⁶ U procesoru bývají příznaky uložené registru nazývaném "*flag register*" či "*status register*" spolu s dalšími jinými příznaky.

⁷ ALU nastavuje *Carry* i jinde, například i u operací logických posunů (*shift*), zmíněných v předchozí kapitole. U posunu doprava je pak *Carry* nejnižším bitem, který se po posunu ztrácí.

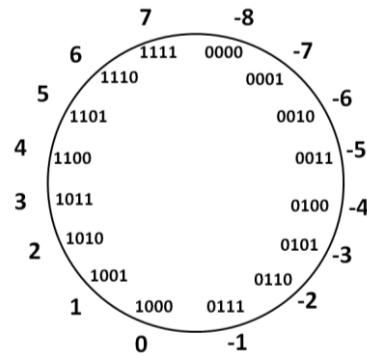
2.3 Celá čísla se znaménkem v přímém a aditivním kódu

Pro čísla se znaménkem existují i další způsoby kódování. Pro úplnost se zmíníme o dvou velmi rozšířených metodách, a to přímém a aditivním kódu.

Číslo se znaménkem můžeme uložit tak, že zakódujeme absolutní hodnotu čísla způsobem binárního čísla *unsigned* a k ní přidáme znaménkový bit. Tento způsob se nazývá přímý binární kód (angl. *straight binary*) nebo kód znaménko a hodnota (angl. *Sign and magnitude representation* či zkráceně jen *sign-magnitude*).



Obrázek 7 - Znaménko a hodnota



Obrázek 8 - Aditivní kód s $K=8$

Přímý kód pro 4bitová binární čísla ukazuje Obrázek 7. V kódu existují dvě nuly, záporná a kladná. Pokud se nám například během iterací výsledek limitně přiblížil k nule, známe směr, ze kterého se k ní dostal. Další předností kódu je velmi rychlá aritmetická negace, kterou tvoříme pouze změnou nejvyššího bitu.

Příklad: Zakódujte číslo -15 v 8bitovém přímém kódu.

Řešení: 8bitový přímý kód má jeden bit znaménka a sedm bitů absolutní hodnoty čísla. Zakódujeme tedy $|-15|$ na 7 bitů jako binární číslo *unsigned* 000 1111 a před něj doplníme bit znaménka, zde 1, protože číslo je záporné. Dostaneme výsledek 1000 1111.

Další zakódování čísel se znaménkem se v češtině označuje jako **aditivní kód** nebo jako kód s posunutou nulou. V angličtině pro něj existují názvy *Excess-K* nebo *offset binary* či *biased representation*. V něm se celá čísla znaménkem napřed převedou na čísla nezáporná, a to přičtením pevně dané konstanty K , zvolené tak, aby výsledek byl vždy kladné číslo. To pak zakódujeme jako binární číslo *unsigned*.

Při zvolené bitové délce binárního čísla m a konstantě K dostaneme rozsah zobrazitelných celých čísel od $-K$ do 2^m-1-K .

Například pro 4bitová čísla a hodnotu $K=8$, dostaneme rozsah od -8 do 7, viz Obrázek 8. Rozsah zobrazitelných záporných čísel můžeme nastavit dle potřeby volbou hodnoty K .

Zvolíme-li například $K=30$, pak pro 4bitová binární čísla dostaneme zobrazitelný rozsah dekadických od -30 do -15.

Příklad: Zakódujte na -15 jako 5 bitové binární číslo v aditivním kódu +16.

Řešení: $-15 + 16 = 1$. Dekadické číslo 1 jako 5bitové binární číslo je 00001.

Aditivní kód místo čísel x a y provádí aritmetické operace s čísly $(x+K)$ a $(y+K)$, takže například výsledek součtu čísel $x+y$ je číslo $(x+y)+2*K$, což znamená, že se výsledek musí vždy ko-

rigovat odečtením K. Korekce násobení budou ještě složitější a s čísly v aditivním vůbec nelze přímo provést dělení.

Přímé a aditivní kódy se obecně nehodí pro bezprostřední výpočty, protože běžné procesory s nimi neumí pracovat. Používají se však pro přenos čísel, jako vnitřní kódy nebo pro složených čísel, jako například u čísel v pohyblivé řádové čárce zakódovaných podle normy IEEE 754. Tu používají skoro všechny moderní procesory pro typy *float*, *double* i *extended*. IEEE 754 ukládá mantisu čísla v přímém kódu a exponent čísla v aditivním kódu⁸.

2.4 Hexadecimální notace

Hexadecimální notace je způsob zkráceného zápisu binárních řetězců. Každou jejich čtveřici bitů zakódujeme jako 4bitové binární číslo *unsigned*. Pro zachování zápisu čtveřic jedním znakem se pro hodnoty 10 až 15 používají písmena A až F.

Příklad 1: Jak je binární řetězec 10100111 v hex-notaci?

Řešení: Řetězec rozdělíme na čtveřice 1010 0111 a každou z nich zakódujeme, takže dostaneme **A7**

Příklad 2: Řetězec 11100110101011 převed'te na hex-notaci:

Řešení: Řetězec rozdělíme opět na čtveřice, a to od jeho pravé strany na 11 1001 1010 1011. Levou skupinu, která má jen 2 bity, doplníme nulami, tedy na 0011 1001 1010 1011, což zakódujeme na **39AB**

Příklad 3: Hexadecimální notaci 1F přepište na 6bitový binární řetězec.

Řešení: Přímým přepisem dostaneme 0001 1111 a vezmeme posledních 6 bitů, tedy **01 1111**

Binární řetězec	Znak	Hodnota <i>unsigned</i>
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Samotný zápis v hexadecimální notaci vede implicitně na počet bitů dělitelný čtyřmi, avšak je možné jím zaspát i menší délku, pokud přidáme specifikaci bitové délky.

2.4.1 Hexadecimální čísla

Hexadecimální číslo znamená, že bitový řetězec zapsaný v hexadecimální notaci interpretujeme jako binární číslo *unsigned*. Pro zápis hexadecimálního čísla existuje zhruba dvacet různých způsobů. Některé z nich si ukážeme na hodnotách A7 a 39AB z příkladu 1 a 2 nahore:

- a) 0A7H , 39ABH Hexadecimální notace se zakončí příponou H a pokud začíná písmenem, přidá se prefix 0 pro zdůraznění číselné hodnoty.
- b) 0xA7, 0x39AB Před číslo se připojí prefix 0x.
- c) X"A7", X"39AB" Zápis v jazyce VHDL.
- d) 16#A7, 16#39AB Zápis v jazyce PostScript.

Přehled podává Wikipedie, anglické heslo [Hexadecimal](#), část *Distinguishing from decimal*.

⁸ S čísly IEEE 754 se většinou nepočítá přímo, ale před aritmetickými operacemi se vždy rozloží na mantisu a exponent, které se zpracují zvlášť, a výsledek se znovu zakóduje. Některé operace lze však provést přímo se zakódovaným číslem. Uložení mantisy v přímém kódu dovoluje rychlou aritmetickou negaci čísla, a to pouhou změnou jednoho bitu. Rovněž lze dvě čísla IEEE 754 vzájemně porovnat přímo v zakódovaném tvaru, tedy stejně tak rychle jako dvě čísla integer.

V programovacích jazycích se konstanta (*literal*) zapsaná jako hexadecimální číslo může i brát jako binární číslo *signed*, pokud ji přiřazujeme do proměnné typu *signed*. Situaci si ukážeme na programu v jazyce C++, přeloženém tak, že proměnná typu *int* má délku 4 byty.⁹

```
int intsize = sizeof(int);      // intsize = 4 (byty)
unsigned char uc = 0xFF;       // uc = 255
char sc = 0xFF;                // sc = -1
unsigned short int usi = 0xFFFF; // usi = 65535
short int ssi = 0xFFFF;       // ssi = -1
unsigned int ui = 0xFFFFFFFF;  // ui = 4294967295
int si = 0xFFFFFFFF;          // si = -1
```

Jak vidíte z ukázky, konstanta 0xFF se nemusí vždy rovnat 255, viz proměnná *sc*.

2.4.2 Číselné soustavy

Hexadecimální čísla se často zavádějí jako čísla o základu (*radix*) 16. Taková definice všem přímo indukuje, že číslo je typu *unsigned*, a kvůli tomu jsme se jí zatím vyhnuli.

$$x_{16} = \sum_{k=0}^{m-1} a_k 16^k; \quad 16 > a_k \geq 0 \quad (3)$$

Hodnotu hexadecimálního čísla $x_{16}=0xA7$ podle (3) vypočteme jako $x_{16}=10*16^1+7*16^0 = 167$. Z matematického hlediska lze jako základ zvolit libovolné celé číslo větší než jedna. Zvolíme-li například $r=10$, dostaneme dekadická čísla. Označme-li obecný nenulový základ r , pak číslo x_r je v soustavě se základem r dáno vzorcem:

$$x_r = \sum_{k=0}^{m-1} a_k r^k; \quad r > a_k \geq 0, \quad r > 1 \quad (4)$$

Hodnoty a_k jsou čísla, ale nahrazují se jedním znakem, podobně jako v hexadecimální notaci. V obvodech a počítačích se upřednostňují čísla s radixy $r=2^m$ dovolující rychlé převody na binární čísla a efektivní uložení v paměti. Exponent m určuje délku skupiny bitů, která se kóduje jedním znakem. Běžně používané soustavy jsou v tabulce dole:

Název čísla	Základ (radix) r	Počet bitů skupiny
Binární	2	1
Oktalové	8	3
Hexadecimální	16	4
Base32	32	5
Base64 nebo Radix64	64	6
<i>Dekadické</i>	<i>10</i>	<i>- nelze převádět po bitových skupinách-</i>

Tabulka 4 - Běžně používané základy (radixy) pro číselné soustavy

V následujících odstavcích se krátce zmíníme o dosud neprobraných kódech.

2.4.2.1 Oktalová čísla

Zvolíme ve vzorci (4) hodnotu $r=8$, pak dostaneme kódování ve skupinách po třech bitech, které se nazývá oktalová (nebo též oktální) čísla, v angličtině "*octal numeral system*" nebo zkráceně "*oct*",

⁹ Velikost *int* závisí na překladači. V 64bitovém prostředí by sice mohla být i 8 bytů, tj. 64 bitů, ale mnohé překladače volí ve výchozím nastavení i zde *int* jako 4byty, tj. 32 bitů, z důvodů zpětné kompatibility

Například hexadecimální číslo 0xA7, s dekadickou hodnotou 167, je zakódováno jako binární řetězec 1010 0111. Ten rozdělíme odprava na trojice 10 100 111 a každou zakódujeme jako binární číslo *unsigned*. Výsledek bude oktalové číslo 247. Oktalová čísla se někdy zapisují s příponou Q, aby se zdůraznil jejich charakter, tedy 247Q.

Oktalová čísla se dříve hodně používala v telekomunikační technice, ale dnes se vyskytují pouze výjimečně. Prakticky jediné častější uplatnění jsou příkazy *chown* a *chmod* používané v implementacích Unixu (zkratky pro *change owner* a *change mode*), kde se jejich argumenty zadávají jako oktalová čísla (bez přípony Q).

2.4.2.2 **Base32 a Base64**

Při volbě základu 32 dostaneme kódování po pěticích a při základu 64 po šesticích bitů. Jde o velmi hojně používané způsoby pro úsporný textový přenos binárních řetězců, například šifrovacích klíčů mezi počítači. Často se v nich zadávají i kódy pro aktivaci programů.

Kódování Base32 a Base64 nejsou tak průhledná jako hexadecimální čísla, protože málokdy reprezentují nějakou číselnou hodnotu podle vzorce (4). Kódované skupiny o délce 5, či respektive 6 bitů, nejsou soudělné s běžnými velikostmi čísel, a to 1, 2, 4 a 8 bytů, takže se nevyplatí kódovat oddělená čísla, protože by se nedosáhla významná úspora. Skupiny čísel se proto často spojují v jeden dlouhý binární řetězec, který se zakóduje jako celek. Při dekódování se zase rozdělí na řadu čísel.

Dělení velmi dlouhých binárních řetězců začíná zpravidla odleva, a lze užít různé přiřazení znaků číslům a_k ze vzorce (4). Existuje několik kódovacích tabulek, zavedených jednotlivými výrobci, podle kterých se převádějí hodnoty od 0 do 31 pro Base32 a od 0 do 63 pro Base64 na alfanumerické znaky. Tabulky pro kódování jsou navrženy tak, aby se vyloučila záměna podobných znaků, jako třeba malého písmena l a číslice 1. Base32 využívá jen číslice a písmena a malé i velké písmeno hodnotí stejně. Base64 již rozlišuje mezi malými a velkými písmeny, malému písmenu je přiřazena jiná hodnota než velkému.

Na konec kódů Base32 a Base64 se připojuje sekvence =, což se nazývá *pad* (oddělení), slouží pro specifikaci délky a udává i konec, aby se textový řetězec mohl poslat ve více textových rádcích. Blíže viz Wikipedie, hesla Base32 a Base64.

Srovnání jednoho možného zakódování pro Base32 a Base64 s jinými kódy.:

Dekadické číslo 1234567890

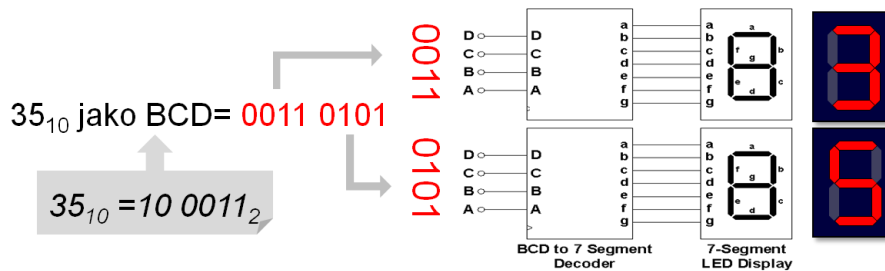
zapsáno jako	zakódované jako řetězec
• dekadické	1234567890
• binární (<i>unsigned</i>)	0100 1001 1001 0110 0000 0010 1101 0010
• hexadecimální	499602D2
• oktalové	11145401322
• jeho dělení na bity	01 001 001 100 101 100 000 001 011 010 010

v bitovém řetězci doplněném na celý počet skupin

- Base32 RFC4648 **JGLAFUQ=**
- bity **01001 00110 01011 00000 00101 10100 10+000**
- Base64 original **SZYC0g==**
- bity **010010 011001 011000 000010 110100 10+0000**

2.5 BCD - Binary Coded Decimal

BCD kódování čísel patří k nejstaršímu používanému způsobu. Jde o přímý zápis dekadického čísla, protože každou jeho číslici zakódujeme jako binární číslo *unsigned* a uložíme do čtyř bitů. Například dekadické číslo 35 uložíme jako 0011 0101.



Obrázek 9 - BCD číslo 35

Výhodou BCD kódování je velká čitelnost binárního čísla, protože jsme schopni přímo z jeho binárního kódování vidět dekadické číslo. Převody na BCD tvar se používají s výhodou u zobrazovacích jednotek, např. chceme-li číslo zobrazit na 7segmentovém displeji, musíme ho napřed převést na BCD tvar, viz Obrázek 9. Stejně tak při tisku čísla, například pomocí C funkce `printf()`, se číslo převede na BCD tvar a ten na znaky číslic.

BCD čísla kódují číslice do čtveřic bitů, stejně jako hexadecimální čísla, však na rozdíl od nich nevyužívají celý rozsah 4bitových čísel *unsigned* od 0 do 15, ale pouze hodnoty 0 až 9. Pokud by se ve čtveřici bitů ocitla hodnota mimo tento rozsah, nejde o platné BCD číslo. Při zakódování dekadického čísla 9876543210 bude výsledné BCD číslo mít $4 \cdot 10$ bitů, tedy 40 bitů:

Dekadické číslo	9	8	7	6	5	4	3	2	1	0
BCD číslo	1001	1000	0111	0110	0101	0100	0011	0010	0001	0000

První počítače pracovaly v BCD pro snadnou čitelnost hodnoty pro lidské operátory, jako třeba vůbec první elektronický počítač (*Electronic Numerical Integrator and Computer*) vyrobený v roce 1946, viz Obrázek 10 [fotografie z Wikipedie, heslo ENIAC].



ENIAC vážil přes 27 tun, potřeboval příkon 150 kW a měl celkovou paměť 20 čísel o 10 číslicích, interně uložených v BCD.

Obrázek 10 - ENIAC Electronic Numerical Integrator and Computer

Z hlediska uložení BCD čísel v počítači se rozlišují dva tvary BCD čísel. *Unpacked* BCD ukládá každou BCD číslici v samostatném bytu a hodí se pro numerické operace. *Packed* BCD ukládá v jednom bytu 2 BCD číslice a slouží pro úsporné uložení čísel. Mikroprocesory dovedou většinou provádět aritmetické operace jen s *unpacked* BCD čísly.

Dnes se přímé počítání s BCD čísly provádí pouze výjimečně, protože je zhruba 2krát až 3krát pomalejší než s binárními čísly a navíc vyžaduje více přístupů do paměti. Používá se však převod na BCD, který potřebujeme před každým tiskem čísel.

Binární číslo lze samozřejmě převést na BCD pomocí dělení deseti a počítání zbytků po dělení, ale takový postup je zbytečně pomalý. Existuje mnohem rychlejší metoda, která se dá na úrovni assembleru velmi rychle algoritmizovat, a to převod pomocí posunů binárního doleva pomocí Hornerova schématu, viz Postup 2 v kapitole 2.1.3 začínající na str. 9. Metoda se dá navíc provádět přímo s *packed* BCD čísly a i snadno paralelizovat na úrovni obvodů. Potřebujeme pro ni jediné operaci násobení BCD číslem 2.

2.5.1 Násobení BCD číslem 2

Pokud číslo BCD obsahuje číslici 0 až 4, pak ji snadno vynásobíme 2 stejně jako binární číslo pomocí posunu doleva o jeden bit, protože výsledek bude 0 až 8, což je platný BCD rozsah. Potíž nastane při násobení dvěma BCD číslice v rozsahu 5 až 9, protože výsledek bude 10 až 18, což již není BCD číslice.

Můžeme si však pomoci tak, že k BCD kódu číslice o hodnotě 5 až 9 před posunem připočteme 3.

Proč přičítáme 3? Jde o polovinu délky rozsahu, který chybí v BCD kódování. To používá pouze hodnoty 0 až 9 z celého rozsahu od 0 do 15 pro 4bitová *unsigned binary* čísel. Chybí v něm tedy 6 hodnot, a to do 10 do 15. Kvůli tomu se před posunem doleva (násobením 2) připočítává korekce +3, tedy polovina délky chybějícího rozsahu, ke všem číslicím větším než 4. Při posunu doleva (násobení 2) pak dojde u korigovaných BCD číslic automaticky k přeskočení 6 chybějících čísel, a tak dostaneme správný výsledek.

BCD	Korekce	Před posunem doleva	Po posunu
0000 0000 [0 0]		0000 0000 [0 0]	0000 0000 [0 0]
0000 0001 [0 1]		0000 0001 [0 1]	0000 0010 [0 2]
0000 0010 [0 2]		0000 0010 [0 2]	0000 0100 [0 4]
0000 0011 [0 3]		0000 0011 [0 3]	0000 0110 [0 6]
0000 0100 [0 4]		0000 0100 [0 4]	0001 0100 [0 8]
0000 0101 [0 5]	+ [0 3]	0000 1000 [0 8]	0001 0000 [1 0]
0000 0110 [0 6]	+ [0 3]	0000 1001 [0 9]	0001 0010 [1 2]
0000 0110 [0 7]	+ [0 3]	0000 1010 [0 10]	0001 0100 [1 4]
0000 0110 [0 8]	+ [0 3]	0000 1011 [0 11]	0001 0110 [1 6]
0000 0110 [0 9]	+ [0 3]	0000 1100 [0 12]	0001 1000 [1 8]

Korekce může způsobit dočasný vznik neplatných hodnot pro BCD číslice, v tabulce nahore 1010, 1011 a 1100, viz poslední řádky, ale ty se vzápětí logickým posunem doleva převedou na správné hodnoty.

2.5.2 Převod binárního čísla unsigned na BCD

Algoritmizaci převodu si ukážeme 8-bitovém binárním číslu *unsigned* 01110011, hex 0x73, s dekadickou hodnotou 115.

- Na začátku každého kroku se napřed projdou všechny jednotlivé 4bitové BCD číslice. Ke každé, která je větší než číslo 4 (0100), se připočte binárně 3 (0011). Přičítání se provádí izolovaně pro každou BCD číslici, tedy jako operace se 4bitovým číslem *unsigned*. Výsledek může po přičtení být větší než 9 (1001) - to se však ihned zkoriguje v kroku b).
- Dále se provede logický posun vlevo celého BCD čísla a převáděného binárního čísla jako jednoho celistvého řetězce bitů.

Pro 8bitové číslo se kroky a) a b) opakují v cyklu celkem 8krát.

Operace	packed BCD		Binární číslo
Inicializace	[0 0 0]	0000 0000 0000	01110011
po 1. společném posunu BCD a bin. čísla	[0 0 0]	0000 0000 0000	11100110
po 2. společném posunu BCD a bin. čísla	[0 0 1]	0000 0000 0001	11001100
po 3. společném posunu BCD a bin. čísla	[0 0 3]	0000 0000 0011	10011000
po 4. společném posunu BCD a bin. čísla	[0 0 7]	0000 0000 0111	00110000
Skupina 0111 > 0100	+ [0 0 3]	+0000 0000 0011	
výsledek korekce	[0 0 10]	0000 0000 1010	00110000
po 5. společném posunu BCD a bin. čísla	[0 1 4]	0000 0001 0100	01100000
po 6. společném posunu BCD a bin. čísla	[0 2 8]	0000 0010 1000	11000000
Skupina 1000 > 0100 → korekce	+ [0 0 3]	+0000 0000 0011	
výsledek korekce	[0 2 11]	0000 0010 1011	11000000
po 7. společném posunu BCD a bin. čísla	[0 5 7]	0000 0101 0111	10000000
Skupiny 0101 a 0111 >= 0100 → korekce	+ [0 3 3]	+0000 0011 0011	
výsledek korekce	[0 8 10]	0000 1000 1010	10000000
po 8. společném posunu BCD a bin. čísla	[1 1 5]	0001 0001 0101	00000000
konec - BCD udává výsledek			

Algoritmus převodu lze ho rozšířit i na delší čísla. Bude blíže probraný v navazující učebnici **Logické obvody na FPGA**, kde si ukážeme jak jeho C-kód tak i obvod, který ho realizuje.

2.6 Kódování znaků ASCII

Kódování znaků zvané ASCII (*American Standard Code for Information Interchange*) vzniklo už v roce 1963, poté prošlo revizemi. Po poslední v roce 1986 se ustálila tabulka používaná dodnes a současně tvořící i základ pro novější kódy UTF8 nebo Unicode. Ty pro zpětnou kompatibilitu zachovávají číselnou hodnotu znaků definovaných v ASCII.

ASCII obsahuje 128 platných znaků o dekadické hodnotě 0 až 127, viz Tabulka 5 na str. 27. Tento rozsah lze uložit do 8bitového čísla jak typu *unsigned*, tak i *signed*.

Příklad: Převeďte řetězec "Hello, Logic!" na ASCII byty.

Řešení: Vyhledáme symboly v ASCII tabulce a zapíšeme jejich ASCII kódy třeba jako:

Znak	H	e	l	l	o	,		L	o	g	i	c	!
Hexadecimálně	48	65	6c	6c	6f	2c	20	4c	6f	67	69	63	21
Dekadicky	72	101	108	108	111	44	32	76	111	103	105	99	33

Tabulka 5 - ASCII kódování znaků

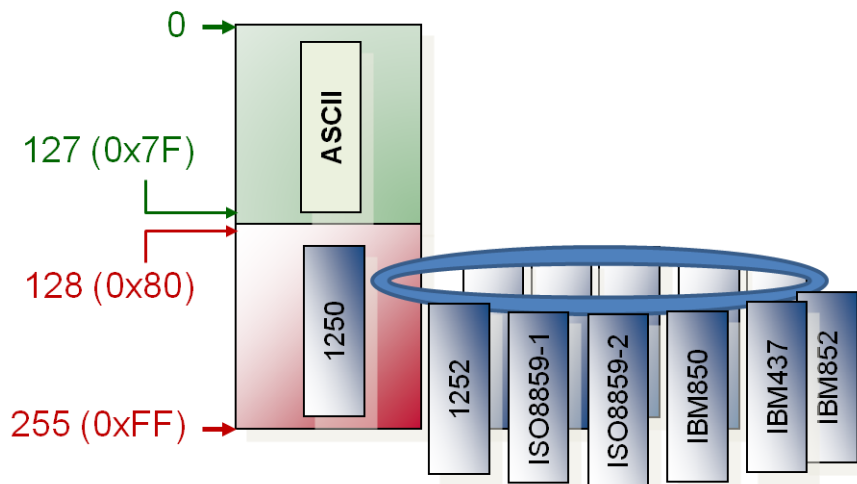
ASCII	Hex	Znak	ASCII	Hex	Znak	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	0x0	NUL	32	0x20	(mezera)	64	0x40	@	96	0x60	`
1	0x1	SOH	33	0x21	!	65	0x41	A	97	0x61	a
2	0x2	STX	34	0x22	"	66	0x42	B	98	0x62	b
3	0x3	ETX	35	0x23	#	67	0x43	C	99	0x63	c
4	0x4	EOT	36	0x24	\$	68	0x44	D	100	0x64	d
5	0x5	ENQ	37	0x25	%	69	0x45	E	101	0x65	e
6	0x6	ACK	38	0x26	&	70	0x46	F	102	0x66	f
7	0x7	\a BEL	39	0x27	'	71	0x47	G	103	0x67	g
8	0x8	\b BS	40	0x28	(72	0x48	H	104	0x68	h
9	0x9	\t TAB	41	0x29)	73	0x49	I	105	0x69	i
10	0xA	\n LF	42	0x2A	*	74	0x4A	J	106	0x6A	j
11	0xB	\v VT	43	0x2B	+	75	0x4B	K	107	0x6B	k
12	0xC	\f FF	44	0x2C	,	76	0x4C	L	108	0x6C	l
13	0xD	\r CR	45	0x2D	-	77	0x4D	M	109	0x6D	m
14	0xE	SO	46	0x2E	.	78	0x4E	N	110	0x6E	n
15	0xF	SI	47	0x2F	/	79	0x4F	O	111	0x6F	o
16	0x10	DLE	48	0x30	0	80	0x50	P	112	0x70	p
17	0x11	DC1	49	0x31	1	81	0x51	Q	113	0x71	q
18	0x12	DC2	50	0x32	2	82	0x52	R	114	0x72	r
19	0x13	DC3	51	0x33	3	83	0x53	S	115	0x73	s
20	0x14	DC4	52	0x34	4	84	0x54	T	116	0x74	t
21	0x15	NAK	53	0x35	5	85	0x55	U	117	0x75	u
22	0x16	SYN	54	0x36	6	86	0x56	V	118	0x76	v
23	0x17	ETB	55	0x37	7	87	0x57	W	119	0x77	w
24	0x18	CAN	56	0x38	8	88	0x58	X	120	0x78	x
25	0x19	EM	57	0x39	9	89	0x59	Y	121	0x79	y
26	0x1A	SUB	58	0x3A	:	90	0x5A	Z	122	0x7A	z
27	0x1B	ESC	59	0x3B	;	91	0x5B	[123	0x7B	{
28	0x1C	FS	60	0x3C	<	92	0x5C	\	124	0x7C	
29	0x1D	GS	61	0x3D	=	93	0x5D]	125	0x7D	}
30	0x1E	RS	62	0x3E	>	94	0x5E	^	126	0x7E	~
31	0x1F	US	63	0x3F	?	95	0x5F	_	127	0x7F	

Důležité vlastnosti ASCII

- Řídící znaky - znaky s kódy 0 až 31 mají význam řídicích kódů, které byly původně určeny pro synchronizaci dálkopisů. Pro hlavní řídicí kódy, dnes stále používané, s hodnotami 7 až 13, existují i *escape* kódy v jazyce C, psané se zpětným lomítkem. Tabulka 5 je má vyznačené červeně. Zde se zmíníme jen BS - *backspace* ('\b'), TAB - tabulátor ('\t'), LF - *linefeed* - posun o řádek ('\n') a CR - *carriage return* - přesun na začátek řádku ('\r'). Úplný přehled řídicích znaků lze najít na Wikipedii, pod heslem [ASCII](#).
- Čísla 0 až 9 mají kódy dekadicky 48 až 57, hexadecimálně 0x30 až 0x39. Převod mezi číslicemi a jejich číselnou hodnotou je proto snadný, stačí odečíst či přičíst 48 (0x30) od hodnoty znaku číslice, tedy ASCII hodnotu znaku '0'.
- Písmena - písmena jsou uložena v souvislých blocích v abecedním pořadí. Velká písmena od 65 do 90 (0x41 do 0x5A) a malá od 97 až 122 (0x61 do 0x7A), takže se snadno dá testovat, zda znak je písmeno, i abecedně třídit.
- Malé a velké písmeno mají kódy vůči sobě vzájemně posunuté vždy o 32 dekadicky, 0x20 hexadecimálně, takže převody mezi malými a velkými písmeny jsou rovněž velmi rychlé, stačí přičíst, resp. odečíst 32 (0x20) od hodnoty kódu znaku.

2.6.1 Extended ASCII

Základní kód ASCII končil u 127 (0x7f). Nepoužité hodnoty od 0x80 do 0xFF se později začaly využívat pro rozšíření ASCII (*extended ASCII*) o národní znaky. Pro češtinu šlo hlavně o znaky s diakritikou.



Obrázek 11 - Princip extended ASCII

Princip rozšíření ukazuje Obrázek 11. Zatímco v části od znaku 0 do 127 zůstávala kódovací tabulka neměnná, definovaná ASCII normou, tak v horní části od 128 do 255 se tabulka kódů měnila podle národní potřeby.

Zde existovalo mnoho různých kódových možností. Specifikace IBM OEM jich zahrnovala 81, podrobnější IANA (*Internet Assigned Numbers Authority*) jich registrovala 257, a ještě zdaleka neobsahovala všechny používané kódové stránky. Chyběla tam třeba česká kódová stránka 895 (bratří Kameničtí), dříve u nás velmi oblíbená. IANA ale obsahuje kódovou

stránkou 1250 používanou českými Windows a českou kódovou stránku 852, v níž se často ukládalo v MS-DOSu.

Pokud se neznala informace o kódové stránce textu, vznikaly četné problémy s kompatibilitou. Nicméně dodnes se *extended ASCII* používá, protože má nezanedbatelnou výhodu, že každý znak má délku pouze 1 byte.

Pokud pracujeme v programu se znaky v *extended ASCII*, musíme mít na paměti, že rozšiřující kódová stránka používá hodnoty od 128 do 255 (od 0x80 do 0xFF). Ty se v 8bitovém čísle typu *signed* zobrazí jako hodnoty od -128 do -1. A v jazyce C se typ *char* bere právě jako *signed char* (*signed* je zde výchozí), tedy 8bitové binární číslo *signed*.

Velmi častou chybou bývá přeskočení bílých znaků (*whitespaces*), o hodnotách 0x8 až 0x20 pomocí porovnání se znakem mezera ' ' = 0x20.

Následující program byl přeložený v C++, kde `sizeof(char)=1` (1 byte):

```
char * line = " \n\t šípek";  
// wrong program for skipping of whitespaces  
int i=0; while (line[i]!=0 && line[i] <= ' ') i++;  
char c = line[i]; // c='p'
```

Program přeskočil nejen mezery, znak nové řádky '\n' (0xA) a tabulátor '\t' (0x9), ale i znaky s diakritikou, protože v rozšířené ASCII tabulce mají záporné hodnoty.

Kód opravíme třeba použitím *unsigned char* pro *line*. Znaky s diakritikou z rozsahu 0x80 až 0xFF v rozšířeném ASCII se pak převedou na hodnoty 128 až 255, takže jediné bílé znaky budou menší nebo rovné mezeře.

```
unsigned char * line = (unsigned char*)" \n\t šípek";  
int i=0; while (line[i]!=0 && line[i] <= ' ') i++;  
char c = line[i]; // c='š'
```

Kódování *extended ASCII* se dnes při programování nejčastěji nahrazuje Unicodem, který může mít ve verzi Basic 16bitové znaky. V plné verzi má znaky v rozsahu 0 až 0x10FFFF, jimiž se pokryjí všechny světové jazyky a používané značky. Znaky Unicodu s hodnotou 0 až 0x7f jsou shodné s ASCII.

Pro ukládání textů a webové stránky se dnes zase stále častěji volí UTF8 (*Unicode Transformation Format*) mající maximální kompatibilitu s ASCII, protože ukládá text po bytech a jeho kódy 0 až 0x7f jsou shodné s ASCII. Kódy 0x80 až 0xFF se v UTF8 používají pro indikaci více bytových sekvencí pro vložení znaků Unicodu. Jejich délka může být až 6 bytů, avšak v UTF-8 se dnes často uplatňuje norma RFC 3629 omezující sekvence na 4 byty.

Výše uvedený program pro vynechání bílých znaků (*whitespaces*) se však rozhodně ne-zjednoduší, použijeme-li znaky v Unicodu, například v jazyce C typ *wchar_t*. Pak nám totiž nestačí testovat jen 6 bílých znaků (v ASCII tabulce menších nebo rovných mezeře), ale správně bychom měli doplnit další testy rozeznávající nejméně 19 nových znaků¹⁰ přidaných Unicodem pro různé typografické mezery a řádkování. Obvykle se však na ně v programech pro zpracování textu zapomíná, buď na všechny, či na jejich část, a tiše☺ se předpokládá, že vstupní text je neobsahuje.

V hardwaru a jednoduchých zobrazovacích jednotkách však ASCII stále zůstává hlavním použitým kódováním.

¹⁰ Celkem Unicode přidává 25 bílých znaků, ale 6 z nich se používá málokdy. Kompletní seznam najdete na Wikipedii pod heslem "[Whitespace character](#)".

2.7 Kolik je 1000?

Kapitolu jsme zahájili vtípem:

*Po autohavárii mi programátor podepsal náhradu škody 1000 Kč.
Zaplatil deset korun s tím, že mi dává dvě koruny navíc.*

Kolik je tedy 1000 v různých soustavách? Vše závisí na použitém zobrazení :-)

Zápis 1000 se převede na dekadické číslo

- = **-8** ze 4-bitového binárního *signed*,
- = **-0** ze *sign-magnitude*,
- = **8** z binárního *unsigned* či binárního *signed* delšího než 4 bity,
- = **512** z oktalového čísla,
- = **1000**, bereme-li zápis přímo dekadicky,
- = **4096** z hexadecimálního čísla; vychytralý programátor si zde připlatí☺
- = **na libovolné celé číslo** z aditivního kódu, a to v závislosti na offsetu, který se v něm používá. Někdo se asi nedoplatí...

2.8 Test znalostí z kapitoly 2

Zkuste odpovědět bez pomůcek na otázky. Výsledky najdete v příloze.

Otázka 1 - doplňte chybějící hodnoty v tabulkách

Dekadické číslo	8 bitové číslo unsigned		BCD číslo	
	Binárně	Hexadecimálně	Binárně	Hexadecimálně
100	0110 0100	64	0001 0000 0000	100
150				
50				
300				

Dekadické číslo	10 bitové číslo signed		Přímý kód - znaménko hodnota	
	Binárně	Hexadecimálně	Binárně	Hexadecimálně
-100	11 1001 1100	39C	10 0110 0100	264
-10				
	11 1111 1110			
		200		
			10 0000 0100	
511				

Otázka 2 - Mějme výraz s operandy zapsanými dekadicky. Doplňte dekadickou hodnotu, kterou bude mít výsledek součtu či rozdílu, budou-li čísla i výsledek uložena v daném formátu

Formát čísla	Délka v bitech	Operace s čísly s dekadickými hodnotami	dá výsledek s dekadickou hodnotu
unsigned	8	100+200=	44
unsigned	10	100+200=	300
unsigned	8	200+200=	
unsigned	9	200+200=	
unsigned	8	127+1=	
signed	8	127+1=	
signed:	8	100-150=	
signed:	12	100-150=	

Otázka 3 - Mějme 8-bitové operandy v binárním tvaru. Napište výsledky operací s nimi.

bitový řetězec	posun vlevo o 1 bit		posun vpravo o 1 bit	
	aritmetický	logický	aritmetický	logický
1000 0001				
1111 1111				
0101 0101				
1010 1010				

Otázka 4 - Jaká bude výsledná hodnota proměnných `iresult` a `creresult` programu v jazyce C?

```
char c1 = 'A';
char c2 = 'b';
int iresult = c2 - c1; // iresult =.....
char creresult = '0' + 5; // creresult =.....
```

3 Příloha

3.1 Abecední seznam zkratk a použitých termínů

Aditivní kód - způsob uložení čísel se znaménkem, viz kapitola 2.5 na str. 24.

ALU *Arithmetic Logic Unit* - aritmeticko-logická jednotka procesoru je základní komponentou procesoru, která provádí všechny aritmetické a logické operace.

ASCII *American Standard Code for Information Interchange* - základní kódování znaků, viz kapitola 2.6 na str. 26.

BCD *Binary Coded Decimal* - způsob zakódování čísel. Všechna čísla se na něj musí převést před jejich zobrazením v dekadickém tvaru, viz kapitola 2.5 na str. 24.

biased representation - - anglický název pro aditivní kód, viz kapitola 2.3 na straně 20.

Borrow přenos z nejvyššího řádu binárního čísla směrem dolů, ke kterému dochází při odčítání, popsáno v kapitole 2.1.5 na straně 11. V řadě publikací se však směr přenosu nerozlišuje a pro oba směry přetečení se používá *Carry*,

Carry přenos z nejvyššího řádu binárního čísla, viz kapitola 2.1.5 na straně 11. *Carry* je současně jedním z hlavních příznaků (*flags*), které generují ALU procesorů.

Excess-K anglický název pro aditivní kód, viz kapitola 2.3 na straně 20.

Extended ASCII - rozšíření ASCII kódu o národní tabulky, viz Kapitola 2.6.1 na str. 28.

LSB má význam "*least significant bit*" nebo "*right-most bit*", nejméně významný bit. LSB se používá i pro uspořádání bytů jako "*least significant byte*". Zda se LSB vztahuje k bitu či bytu, musí vyplynout z kontextu, viz Obrázek 1 na straně 7.

MSB má význam "*most significant bit*" nebo "*high-order bit*", tedy nejméně významný bit. MSB se používá i pro uspořádání bytů jako "*most significant byte*". Zda se MSB vztahuje k bitu či bytu záleží na kontextu popisu, viz Obrázek 4 na straně 7.

offset binary - anglický název pro aditivní kód, viz kapitola 2.3 na straně 20.

overflow termín obecně znamená přetečení, ale v počítačové logice se tento pojem nejčastěji používá pro příznak (*flag*) přetečení u aritmetické operace s binárními čísly *signed*. Příznak *overflow* znamená, že výsledek operace je nesmyslný — jako třeba záporný výsledek sčítání dvou kladných čísel. Dojde k tomu kvůli omezené délce binárního čísla. *Overflow* je současně jedním z hlavních příznaků, které generují ALU všech běžnějších procesorů.

přímý kód způsob uložení čísel se znaménkem, viz kapitola 2.3 na straně 20.

Sign–magnitude - anglický název pro přímý kód, viz kapitola 4.3 na straně 20.

signed (binární číslo) - zkrácený název používaný pro zakódování celých čísel se znaménkem ve dvojkovém doplňku, popsáno v kapitole 2.2 na straně 13.

straight binary - anglický název pro přímý kód, viz kapitola 2.3 na straně 20.

unsigned (binární číslo) - zkrácený název používaný pro zakódování celého nezáporného čísla jako binárního čísla bez znaménka, popsáno v kapitole 2.1 na straně 8.

3.2 Řešení testu z kapitoly 2

Otázka 1 - doplňte chybějící hodnoty v tabulkách

Dekadické číslo	8 bitové číslo unsigned		BCD číslo	
	Binárně	Hexadecimálně	Binárně	Hexadecimálně
100	0110 0100	64	0001 0000 0000	100
150	1001 0110	96	0001 0101 0000	150
50	0011 0010	32	0000 0101 0000	050
300	nelze	nelze	0011 0000 0000	300

Dekadické číslo	10 bitové číslo signed		Prímý kód - znaménko hodnota	
	Binárně	Hexadecimálně	Binárně	Hexadecimálně
-100	11 1001 1100	39C	10 0110 0100	264
-10	11 1111 0110	3F6	10 0000 1010	20A
-2	11 1111 1110	3FE	10 0000 0010	202
-512	10 0000 0000	200	mimo rozsah	mimo rozsah
-4	11 1111 1100	3FC	10 0000 0100	204
511	01 1111 1111	1FF	01 1111 1111	1FF

Otázka 2 - Mějme výraz s operandy zapsanými dekadicky. Doplňte dekadickou hodnotu, kterou bude mít výsledek součtu či rozdílu, budou-li čísla i výsledek uložena v daném formátu

Formát čísla	Délka v bitech	Operace s čísly s dekadickými hodnotami	dá výsledek s dekadickou hodnotu
unsigned	8	100+200=	44
unsigned	10	100+200=	300
unsigned	8	200+200=	144
unsigned	9	200+200=	400
unsigned	8	127+1=	128
signed	8	127+1=	-128
signed:	8	100-150=	nelze, 150 mimo rozsah
signed:	12	100-150=	-50

Otázka 3 - Mějme 8-bitové operandy v binárním tvaru. Napište výsledky operací s nimi

bitový řetězec	posun vlevo o 1 bit		posun vpravo o 1 bit	
	aritmetický	logický	aritmetický	logický
1000 0001	0000 0010	0000 0010	1100 0000	0100 0000
1111 1111	1111 1110	1111 1110	1111 1111	0111 1111
0101 0101	1010 1010	1010 1010	0010 1010	0010 1010
1010 1010	0101 0100	0101 0100	1101 0101	0101 0101

Otázka 4 - Jaká bude výsledná hodnota proměnných `iresult` a `cresult` programu v jazyce C?

```
char c1 = 'A';
char c2 = 'b';
int iresult = c2 - c1; // iresult = 33 (0x21)
char cresult = '0' + 5; // cresult = '5'
```

Autor: Richard Šusta,
<https://control.fel.cvut.cz/ing-richard-susta-phd>
nebo <https://susta.cz/>

Obrázky: * Obrázek 10 - ENIAC Electronic Numerical Integrator and Computer
převzatý z Wikipedie,
* ostatní obrázky Richard Šusta

Vydavatel: Katedra řídicí techniky ČVUT-FEL v Praze,
Technická 2, 166 00 Praha 6
<https://control.fel.cvut.cz/>

Datum vydání: únor 2023, září 2016

Rozsah: 29 stran

Domovská stránka dokumentu:
<https://dcenet.fel.cvut.cz/edu/fpga/doc/BinarniPrerekvizita.pdf>