Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

ŠTÁTNY PEDAGOGICKÝ ÚSTAV
NATIONAL INSTITUTE FOR EDUCATION

Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

Počítačové systémy 1

Predmet: Počítačové systémy

Línia: Vlastný odborový kontext informatiky a informatickej výchovy

Operačný program VZDELÁVANIE

EURÓPSKA ÚNIA
Európsky sociálny fond

Európska únia
Európsky sociálny fond

Moderné vzdelávanie pre vedomostnú spoločnosť/Projekt je spolufinancovaný zo zdrojov ES

Počítačové systémy 1

Identifikácia modulu

Aktivita projektu:	1.2 Vzdelávanie nekvalifikovaných učiteľov informatiky na 2. stupni ZŠ a na SŠ
Línia aktivity:	Vlastný odborový kontext informatiky a informatickej výchovy
Predmet:	Počítačové systémy

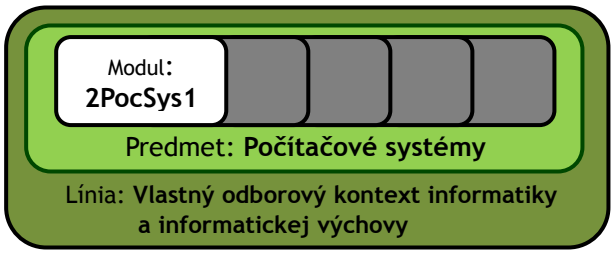
Garant predmetu:

RNDr. Peter Gurský, PhD.
ÚINF PF UPJŠ, Košice
peter.gursky@upjs.sk

Autori:

RNDr. Peter Gurský, PhD.
ÚINF PF UPJŠ, Košice
RNDr. Jozef Jirásek, PhD.
ÚINF PF UPJŠ, Košice
Mgr. Samuel Kupka
ÚINF PF UPJŠ, Košice
RNDr. Veronika Vaneková
ÚINF PF UPJŠ, Košice

Zaradenie modulu



Modul tvorí prvú časť predmetu Počítačové systémy. Účastníci ho spolu s druhou časťou absolvujú v treťom semestri vzdelávania. Zvyšné 3 moduly tohto predmetu sú naplánované na štvrtý semester vzdelávania. Prvý modul predmetu Počítačové systémy čiastočne nadväzuje na predmety Programovanie, Matematika pre učiteľov informatiky a využíva aj poznatky získané v časti Základy hardvérového a softvérového vybavenia počítača predmetu Digitálna gramotnosť učiteľa.

Abstrakt modulu

Prvá časť sa venuje binárnemu zápisu textových znakov, celých a reálnych čísiel a spôsobu počítania v tomto zápise. Sú popísané základné logické funkcie s binárnymi hodnotami a používané binárne operácie.

V druhej časti je ukázaná realizácia jednoduchých logických funkcií pomocou elektronických logických obvodov. Sú popísané aj zložitejšie logické funkcie a spôsob ich realizácie kombinačnými logickými obvodmi z jednoduchých logických členov. Je ukázaná aj konštrukcia registrov a pamäťových obvodov pomocou jednoduchých sekvenčných obvodov.

Obsah

Úvod

Predmet Počítačové systémy sa venuje princípom toho, ako počítače fungujú. Tento predmet sa skladá z 5 modulov. Postupne objasňujú to, ako sú reprezentované znaky a čísla v počítači ako postupnosti jednotiek a núl a aké operácie sa s nimi dajú robiť. Princíp práce počítačov vybudujeme od tých najmenších súčiastok. Začneme tým, ako sa pomocou tranzistorov, základných stavebných jednotiek procesorov a iných hardvérových prvkov, dá pracovať s binárnou informáciou. Ukážeme princíp uchovávania binárnej informácie v registroch a navrhneme jednoduchú pamäť.

Vstupné vedomosti

Požadované prerekvizity

Účastník tohto modulu by mal mať absolvované moduly Programovanie 1 - 9, Matematika pre učiteľov informatiky 1 a 2 a Základy hardvérového a softvérového vybavenia počítača predmetu Digitálna gramotnosť učiteľa.

Predpokladané vstupné vedomosti, skúsenosti a zručnosti

Účastník vie prevádzať medzi zápisom celých čísiel v rôznych sústavách, pozná rôzne primitívne typy premenných. Taktiež má základné poznatky z fyziky.

Preverenie vstupných vedomostí

Aké sú typy premenných v Pascale? Koľko bitov zaberá ich reprezentácia? Ako prevedieme zápis čísla z desiatkovej do dvojkovej a šestnástkovej sústavy?

Morseova abeceda:

A	. –
B	– . . .
C	– . – .
D	– . .
E	.
F	. . – .
G	– – .
H	. . . .
I	. .
J	. – – –
K	– . –
L	. – . .
M	– –
N	– .
O	– – –
P	. – – .
Q	– – . –
R	. – .
S	. . .
T	–
U	. . –
V	. . . –
W	. – –
X	– . . –
Y	– . – –
Z	– – . .
1	. – – – –
2	. . – – –
3	. . . – –
4	. . . . –
5	. . . . .
6	– . . . .
7	– – . . .
8	– – – . .
9	– – – – .
0	– – – – –

Označenie $(.)_x$ budeme používať na vyjadrenie toho, že ide o zápis čísla v x-ovej sústave

1 Reprezentácia znakov a čísiel a operácie s nimi

Všetky bežné počítače reprezentujú údaje, programy, alebo postupnosti inštrukcií, v binárnej forme. Nech je to obrázok, text, video, zvuk, alebo nejaký spustiteľný program, všetko je na tej najnižšej úrovni uložené a spracovávané ako vhodná kombinácia jednotiek a núl.

1.1 Reprezentácia znakov

Reprezentácia písmen a iných znakov prešla, a ešte stále prechádza, zložitým vývojom. Prvé kódovanie znakov vzniklo už v 40. rokoch 19. storočia s príchodom telegrafov. Bola to známa Morseova abeceda, obsahujúca 26 znakov anglickej abecedy a 10 číslic. Medzery medzi znakmi a medzery medzi slovami sa kódovali ako vhodne dlhé odmlky. Morseova abeceda teda používa na zakódovanie textu krátke pípnutie (bodka), dlhé pípnutie (čiarka) a odmlku. Odmlka, ktorá je dlhá ako dlhé pípnutie, predstavuje oddeľovač znakov, a odmlka, ktorá je dlhá ako dve dlhé a jedno krátke pípnutie, predstavuje medzeru medzi slovami. Práve odmlky sú dôvodom, prečo Morseova abeceda nie je považovaná za binárnu ale ternárnu (používa až 3 rôzne značky). Pre použitie na kódovanie znakov v počítačoch je teda nevhodná.

Až v ére prvých počítačov, v roku 1963, sa štandardizovalo binárne kódovanie znakov ASCII (American Standard Code for Information Interchange) pôvodne určené pre ďalekopisy. Hranice medzi znakmi sa jednoducho vyriešili tak, že každý znak má rovnako dlhý 7 miestny binárny kód. Z medzery medzi slovami sa stal obyčajný znak s binárnym kódom 0100000. Binárne kódy môžeme chápať ako zápisy čísiel v dvojkovej, t.j. binárnej sústave. Kód medzery je teda chápaný ako číslo $(100000)_2$ alebo v desiatkovej sústave $(32)_{10}$.

[nul]	0	[syn]	22	,	44	B	66	X	88	n	110
[soh]	1	[etb]	23	-	45	C	67	Y	89	o	111
[stx]	2	[can]	24	.	46	D	68	Z	90	p	112
[etx]	3	[em]	25	/	47	E	69	[	91	q	113
[eot]	4	[sub]	26	0	48	F	70	\	92	r	114
[enq]	5	[esc]	27	1	49	G	71	]	93	s	115
[ack]	6	[fs]	28	2	50	H	72	^	94	t	116
[bel]	7	[gs]	29	3	51	I	73	_	95	u	117
[bs]	8	[rs]	30	4	52	J	74	`	96	v	118
[ht]	9	[us]	31	5	53	K	75	a	97	w	119
[nl]	10		32	6	54	L	76	b	98	x	120
[vt]	11	!	33	7	55	M	77	c	99	y	121
[np]	12	"	34	8	56	N	78	d	100	z	122
[cr]	13	#	35	9	57	O	79	e	101	{	123
[so]	14	$	36	:	58	P	80	f	102	|	124
[si]	15	%	37	;	59	Q	81	g	103	}	125
[dle]	16	&	38	<	60	R	82	h	104	~	126
[dc1]	17	'	39	=	61	S	83	i	105	[del]	127
[dc2]	18	(	40	>	62	T	84	j	106		
[dc3]	19	)	41	?	63	U	85	k	107		
[dc4]	20	*	42	@	64	V	86	l	108		
[nak]	21	+	43	A	65	W	87	m	109		

Obrázok 1: Znaky ASCII kódovania a ich reprezentácia v desiatkovej sústave

Kódovanie ASCII obsahuje okrem znakov, ktoré sa zobrazujú na výstupných zariadeniach aj 33 riadiacich znakov (znaky s kódmi $(0000000)_2=(0)_{10}$ až $(0011111)_2=(31)_{10}$ a znak $(1111111)_2=(127)_{10}$). Tie boli určené pre ovládanie tlačiarní ďalekopisov pripomínajúcich mechanický písací stroj, alebo ako rôzne riadiace dáta určené na ovládanie periférnych zariadení počítača. Mnoho z týchto riadiacich

znakov sa už v súčasnosti nepoužíva.

Kódovanie ASCII sa stalo základom pre rôzne národné kódovania. Na dodefinovanie národných znakov sa využíval ďalší ôsmy bit, čím sa využil celý bajt, ktorý je najmenšou adresovateľnou jednotkou v pamäti počítača. Prvých 128 znakov sa zachovalo z kódovania ASCII a ďalších 128 sa využilo na ďalšie znaky.

V našej oblasti sa využívalo **kódovanie Kamenických** určené primárne pre terminály, kódovanie **ISO-8859-2** (známe aj ako **Latin-2**) a neskôr kódovanie **Windows-1250** (niekedy označované aj ako **CP1250**). Každé z týchto kódovaní obsahovalo všetky slovenské znaky, ale nie vždy s rovnakými kódmi. Napríklad kód $(190)_{10}$ je v kódovaní Windows-1250 kódom pre znak 'ľ', v kódovaní ISO-8859-2 kódom pre znak 'ž' a v kódovaní Kamenických je to kód pre znak rohu tabuľky '╛'. Podobné nezrovnalosti sú aj v ďalších desiatkach iných národných kódovaní.

Rôznorodosť (pluralita) národných kódovaní dodnes spôsobuje mnohé problémy pri výmene textových informácií napr. pri rôznom nastavení kódovania mailových klientov, titulkov, webových stránok, ale napríklad aj názvov súborov. Ďalšou nevýhodou týchto osembitových kódovaní je nemožnosť ich použitia pre východoázijské krajiny s veľkým počtom znakov v abecedách.

Nahradiť pluralitu kódovaní jediným kódovaním má za cieľ univerzálne kódovanie **UTF-8** (Unicode Transformation Format). Pomocou tohto kódovania je možné reprezentovať ľubovoľný znak ľubovoľnej abecedy. Kódovanie UTF-8 má premenlivú dĺžku kódu, čo znamená, že jednotlivé znaky sú podľa štandardu reprezentované 8 až 32 bitmi (1-4 bajtmi) aj keď reálne sa využíva maximálne 24 bitov (3 bajty). Kódovanie UTF-8 preto označujeme ako *kódovanie s premenlivou bitovou dĺžkou*.

Prvých 128 znakov kódovania UTF-8 s kódmi 00000000 až 01111111 sa zhoduje s kódmi kódovania ASCII (podobne ako všetky spomínané národné kódovania). Ak je prvý bit 1, znamená to, že znak má kód dlhší ako jeden bajt. Všetky znaky našej abecedy majú kód dĺžky 1 alebo 2 bajty. Do kódov dvojbajtovej dĺžky sa dostali aj všetky znaky azbuky, gréckej abecedy či arabskej abecedy. Trojbajtové kódy majú hlavne východoázijské znaky. Štvorbajtové kódy sú určené na rôzne historické znakové sady (klinové písmo, hieroglyfy a pod.).

To, koľko bajtov zaberá daný kód znaku je dané povinnými bitmi pre každú dĺžku znaku.

Dĺžka kódu	Povinné bity
1 bajt	0xxxxxxx
2 bajty	110xxxxx 10xxxxxx
3 bajty	1110xxxx 10xxxxxx 10xxxxxx
4 bajty	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Vďaka povinným bitom vieme o ľubovoľnom bajte nejakého textu povedať, či ide o prvý bajt, alebo je to vnútorný bajt nejakého znaku. Príklad kódov niektorých znakov:

Znak	Binárny kód	Hexadecimálna reprezentácia	Dekadická reprezentácia
A	01000001	41	65
á	11000011 10100001	C3 A1	50081
語	11101000 10101010 10011110	E8 AA 9E	15248030
𒀀	11110000 10010000 10001110 10000100	F0 90 8E 84	4036005508

Výhodou používania kódovania s premenlivou bitovou dĺžkou je relatívne malá priemerná dĺžka súborov, napriek tomu že ide o univerzálne kódovanie. V prípade, že dokument pracuje iba s ASCII znakmi, obsahuje iba jednobajtové kódy znakov. Keďže slovenské texty obsahujú relatívne málo národných znakov (tie sú

Kódovanie UTF-16 má dokonca dva varianty - **Little Endian** a **Big Endian**, ktoré určujú ktorý bajt z dvojbajtového resp. štvorbajtového kódu znaku bude prvý. Táto nekompatibilita pramení z rôznej reprezentácie viacbajtových čísiel v rôznych typoch procesorov. Bežné PC používajú Little Endian. Kódovanie UTF-16 vo variante Little Endian sa niekedy nazýva aj **UNICODE**.

Domovská stránka editora **PSPad**: http://www.pspad.com/sk/

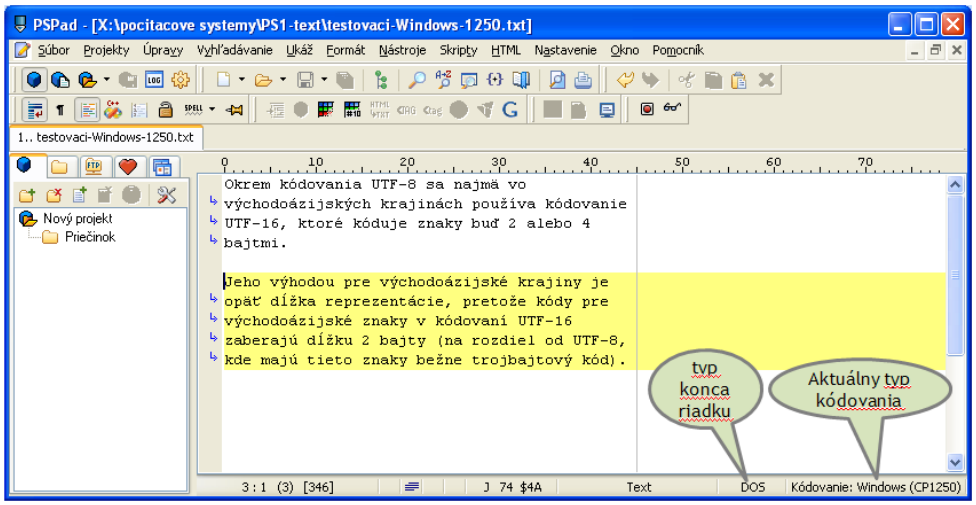
Editor **PSPad** sa dá stiahnuť aj vo verzii, ktorá nepotrebuje inštaláciu a dá sa spúšťať napríklad z USB kľúča.

Ako zaujímavosť môžeme uviesť, že PSPad je celý naprogramovaný v Delphi.

reprezentované 2 bajtmi), priemerná dĺžka kódu znaku v bežnom slovenskom texte s kódovaním UTF-8 je asi 1,1 bajtu.

Okrem kódovania UTF-8 sa najmä vo východoázijských krajinách používa kódovanie **UTF-16**, ktoré kóduje znaky buď 2 alebo 4 bajtmi. Jeho výhodou pre východoázijské krajiny je opäť dĺžka kódov, pretože kódy pre východoázijské znaky v kódovaní UTF-16 zaberajú dĺžku 2 bajty (na rozdiel od UTF-8, kde majú tieto znaky bežne trojbajtový kód).

Konverzia medzi kódovaniami sa dá robiť rôznymi spôsobmi. Jeden z najjednoduchších spôsobov konverzie ponúkajú niektoré textové editory ako napríklad český textový editor PSPad.



Obrázok 1: Editor PSPad

Editor PSPad má zabudovanú podporu práve pre naše národné kódovania. Ak je súbor uložený v kódovaní UTF-8, PSPad to spravidla automaticky odhalí a zobrazí súbor v tomto kódovaní. Ostatné znakové sady skúša (nie vždy úspešne) odhaliť frekvenčnou analýzou znakov v textovom súbore.

Ak PSPad zobrazil súbor v zlom kódovaní, je možné nastaviť správne kódovanie cez menu Formát, kde treba vybrať požadované kódovanie a následne je potrebné znova otvoriť súbor (CTRL-r).

Konverzia zo správne zobrazeného súboru do iného kódovania sa dá urobiť opäť cez menu Formát s vybraním kódovania a následným uložením súboru.

Editor PSPad umožňuje aj meniť reprezentáciu konca riadku, ktorá je rôzna pre Windows (DOS) – dva znaky s kódmi $(13)_{10}$ a $(10)_{10}$, Unix – jeden znak $(10)_{10}$ a MacOS – jeden znak $(13)_{10}$. Zmena reprezentácie konca riadku sa dá vykonať buď kliknutím na označenie typu konca riadku v dolnej lište alebo v menu Formát.

Aktivita 1.	Použite ľubovoľný text s diakritikou a uložte ho rôznymi typmi kódovania znakov. Aký je rozdiel vo veľkosti súborov? Prečo?
Aktivita 2.	Otvorte súbor uložený v kódovaní UTF-8 v nejakom hexaeditore (napríklad program XVI32 - http://www.chmaas.handshake.de/delphi/freeware/xvi32/xvi32.htm) a nájdite znaky, ktoré sú reprezentované viacerými ako 8 bitmi.

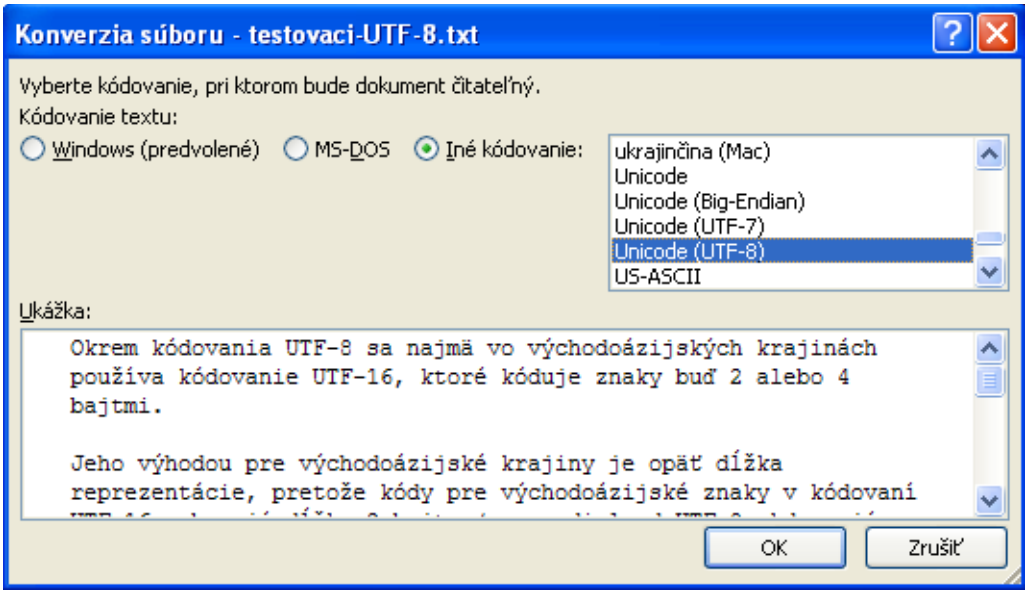
Pluralita kódovaní je problémom aj pri používaní vstavaných textových editorov

systému MS Windows. Program Poznámkový blok (Notepad) dokáže otvárať súbory v kódovaniach UTF-8 a UTF-16. Ak však otvoríte súbor s inou reprezentáciou konca riadku ako $(13)_{10}(10)_{10}$, nedokáže reprezentovať koniec riadka korektne.

Program WordPad, ktorý je tiež súčasťou OS Windows, dokáže správne odhaliť inú reprezentáciu konca riadku, ale nedokáže správne otvoriť všetky súbory s kódovaním UTF-8. WordPad tiež nemá podporu pre konverziu medzi kódovaniami.

Textové súbory sa dajú otvárať aj v programe MS Word. Ten umožňuje výber správneho kódovania ešte pred samotným otvorením súboru.

Program WordPad dokáže správne identifikovať kódovanie UTF-8 iba pre tie súbory, ktoré začínajú špeciálnou, inak nepovinnou, trojbajtovou BOM (Byte Order Mark) sekvenciou $(EFBBBF)_{16}$. Ostatné UTF kódovania majú tiež svoje BOM sekvencie.
BOM sekvenciu pridáva na začiatok súboru pri konverzii do UTF-8 napríklad program NotePad.



Obrázok 2. Dialógové okno MS Word pri otváraní textového súboru

Aktivita 3.

Diskutujte o tom, či je možné niekedy dosiahnuť celosvetové používanie jediného univerzálneho kódovania. Čo považujete za hlavný dôvod zachovania rôznych kódovaní v súčasnosti?

1.2 Reprezentácia nezáporných čísiel

Reprezentácia nezáporných čísiel je veľmi jednoduchá. Zapíšeme číslo do dvojkovej sústavy a výsledok uložíme. V Object Pascal-e sa takto reprezentujú hodnoty dátových typov **`byte`**, **`word`** a **`longword`**. Premenné dátového typu `byte`, ako sa dá už podľa názvu odvodiť, sú schopné uchovávať čísla, ktoré sa dajú reprezentovať vrámci jedného bajtu t.j. 8 bitov. Hodnoty premenných typu `word` sú reprezentované v dvoch bajtoch (16 bitov) a typu `longword` v štyroch bajtoch (32 bitov).

Typ	Binárny rozsah	Dekadický rozsah
byte	0000 0000 až 1111 1111	0 až 255
word	0000 0000 0000 0000 až 1111 1111 1111 1111	0 až 65535
longword	0000 0000 0000 0000 0000 0000 0000 0000 až 1111 1111 1111 1111 1111 1111 1111 1111	0 až 4294967295

Typy, ktorých premenné sú schopné uchovávať iba nezáporné čísla sa nazývajú aj neznamienkové (unsigned) typy.

Aritmetické operácie sa v procesore vykonávajú priamo nad binárnymi kódmi čísiel. Algoritmus na výpočet súčtu a rozdielu sa podobá na postup, ktorý sa učí na

základnej škole. Postupne sa sprava doľava sčítavajú dvojice cifier a prenos zo súčtu predchádzajúcich cifier.

Pri sčítaní čísiel v binárnom tvare máme pri prvých cifrách štyri možnosti, ktoré môžu nastať – každá zo sčítavaných cifier môže mať hodnotu 0 alebo 1. Pre všetky vyššie bity máme celkovo osem možností sčítania, pretože potrebujeme rátať aj s prenosom zo sčítania predchádzajúcich cifier. Môžeme si napísať tabuľku všetkých možností pri binárnom sčítavaní.

$prenos_i$	a_i	b_i	$prenos_i+a_i+b_i$	$prenos_{i+1}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Vyskúšajme si binárne sčítanie čísiel $(118)_{10}=(01110110)_2$ a $(84)_{10}=(01010100)_2$ typu `byte`. Keď sčítavame dve takéto 8-bitové čísla výsledkom je opäť 8-bitové číslo.

```
 01110110    118
+01010100   +84
----------  ----
 11001010    202
 ~~~~ ~
 1111 1
```

Čo sa stane, ak výsledok sčítania presiahne 8 bitov? Ak máme zapnutú kontrolu prekročenia rozsahu, vývojové prostredie nám oznámi chybu. Ak však kontrola prekročenia rozsahu nie je zapnutá, bit, ktorý je mimo rozsah, sa jednoducho ignoruje. Pri 8 bitovom sčítaní sa to deje s deviatym bitom t.j. $(1\ 0000\ 0000)_2=(2^8)_{10}=(256)_{10}$. To znamená, že ak chceme zistiť, aká bude výsledná hodnota sčítania dvoch 8 bitových čísiel v počítači, musíme od matematicky správneho výsledku sčítania odčítať $(256)_{10}$.

```
 11110110    246
+01010100   +84
----------  ----
1:01001010   330
 ~ ~~~~ ~   -256
 1 1111 1   ----
             74
```

Odčítanie čísiel v binárnom tvare je veľmi podobné sčítaniu. Opäť máme celkovo 8 možností, ktoré môžu pri odčítaní dvoch cifier nastať, lebo aj pri odčítaní musíme počítať s prenosom do vyššieho rádu vzniknutom pri odčítaní prechádzajúcich dvoch cifier. Možnosti môžeme opäť vypísať v tabuľke. Zaujímavé je pozorovanie, že 4. stĺpec je zhodný v tabuľke sčítania aj odčítania. Rozdiel je iba v prenose do vyšších rádov.

$prenos_i$	a_i	b_i	$a_i-(prenos_i+b_i)$	$prenos_{i+1}$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

Vyskúšajme si odčítanie čísiel $(118)_{10}$ a $(84)_{10}$ v binárnom tvare. Na rozdiel od ich sčítania, pri ich odčítaní nenastávajú žiadne prenosy.

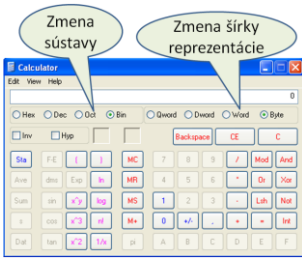
$$\begin{array}{rr} 01110110 & 118 \\ -\,01010100 & -\,84 \\ \hline 00100010 & 34 \end{array}$$

Čo sa však stane, keď vymeníme poradie, teda od $(84)_{10}$ odčítame $(118)_{10}$? Výsledkom v obore celých čísiel je záporné číslo. Ak však pracujeme s nezápornými číslami, a odčítavame podľa pravidiel v tabuľke odčítania, opäť dospejeme k prekročeniu rozsahu ôsmich bitov. V tomto prípade je výsledkom pri vypnutej kontrole prekročenia rozsahu číslo, ktoré si môžeme vypočítať tak, že k výslednému zápornému výsledku pripočítame $(1\ 0000\ 0000)_2=(2^8)_{10}=(256)_{10}$.

$$\begin{array}{rr} 01010100 & 84 \\ -\,01110110 & -\,118 \\ \hline 1\,11011110 & -\,34 \\ \sim\ \sim\sim\sim\sim\sim\sim & +256 \\ 1\ \ 1\ 1\ 1\ 1\ 1\ 1 & \\ \hline & 222 \end{array}$$

Aktivita 1.	Vyskúšajte si v programe Kalkulačka, ktorý je súčasťou OS Windows, konverziu medzi zápismi čísiel v rôznych sústavách. Zmeňte typ kalkulačky na vedeckú. Konverziu vykonáte tak, že zadáte číslo v aktuálnej sústave a prepínaním medzi možnosťami (Hex, Dec, Oct a Bin) robíte konverziu do šestnástkovej, desiatkovej, osmičkovej alebo dvojkovej sústavy.
Aktivita 2.	Vyskúšajte si v programe Kalkulačka, ktorý je súčasťou OS Windows, súčty a rozdiely v binárnej sústave z textu tejto kapitoly. Zmeňte typ kalkulačky na vedeckú. Nastavte si binárnu sústavu (voľba Bin) a typ Byte (šírka reprezentácie je 8 bitov).
Aktivita 3.	V programe Lazarus si vyskúšajte súčet a rozdiel dvoch premenných typu `byte` a overte výsledky sčítaním binárnych reprezentácií týchto čísiel na papieri.

Príklad riešenia aktivity 3	`procedure TForm1.Button1Click(Sender: TObject);`
	`var`
	`  b1,b2,b3:byte;`
	`begin`
	`  b1:=84;`
	`  b2:=118;`
	`  b3:=b1+b2;`
	`  memo1.Append(intTostr(b3));`
	`  b3:=b1-b2;`
	`  memo1.Append(intTostr(b3));`
	`end;`



Obrázok 3. Kalkulačka OS Windows

1.3 Reprezentácia celých čísiel

V prípade celých čísiel je potrebné vedieť reprezentovať kladné aj záporné čísla. Rovnako, ako pri nezáporných číslach, každé celé číslo je reprezentované ako postupnosť daného počtu bitov.

1.3.1 Znamienkový kód

Príklad reprezentácie v znamienkovom kóde pri 8 bitoch:
$(3)_{10}$ má kód 0000 0011
$(-3)_{10}$ má kód 1000 0011

Jedna z najprirodzenejších reprezentácií používa najvyšší bit ako znamienkový. Pri 8 bitovom čísle to znamená, že 7 bitov obsahuje absolútnu hodnotu čísla v dvojkovej sústave a najvyšší ôsmy bit (t.j. bit najviac vľavo) reprezentuje znamienko. 0 predstavuje plus a 1 predstavuje mínus. Siedmimi bitmi vieme reprezentovať čísla $(0)_{10}$ až $(127)_{10}$. Rozsah platných čísiel v znamienkovom kóde je teda $(-127)_{10}$ s kódom 1111 1111 až $(127)_{10}$ s kódom 0111 1111.

Zaujímavé je, že máme dve možnosti na reprezentáciu nuly: (-0) má kód 1000 0000 a (+0) má kód 0000 0000. Táto vlastnosť komplikuje vyhodnocovanie výsledku porovnania hodnôt dvoch výrazov, ktoré sa robí ako vyhodnotenie ich rozdielu. V prípade nejednoznačnej reprezentácie musíme obvykle testovať dvakrát. Znamienkový kód je výpočtovo náročný pri aritmetických operáciách, kde sa mení znamienko.

1.3.2 Posuvný kód

Príklad sčítania dvoch kladných čísiel v 8 bitovom kóde EXCESS-128 kóde:

```
  1000 0011 t.j.(3)10
 +1000 0010 t.j.(2)10
  ----------
±|0000 0101
```

Výsledok predstavuje hodnotu (3+128)+(2+128) = (5+256). Aby výsledok bol tiež v posuvnom kóde musíme od výsledného 8 bitového čísla (už bez bitu pretečenia) odčítať posun 128.

```
  0000 0101
 -1000 0000
  ----------
±|1000 0101 t.j.(5)10
```

Ďalším kódom pre celé čísla je posuvný (excess) kód. Prevod do tohto kódu v 8 bitovej reprezentácii sa vykoná tak, že k reprezentácii čísla v dvojkovej sústave (kladného alebo záporného) prirátame nejaký posun, napr. $(2^7)_{10}$ = $(128)_{10}$ = $(1000\ 0000)_2$. Napríklad, ak chceme v tomto prípade previesť číslo $(-5)_{10}$ = $(-101)_2$ do posuvného kódu, vypočítame $(-5)_{10}$ + $(128)_{10}$ = $(123)_{10}$ = $(0111\ 1011)_2$. Posuvný 8 bitový kód pre $(-5)_{10}$ je teda 0111 1011.

Ak sme posun kódov čísiel vykonali o číslo $(128)_{10}$, označujeme ho EXCESS-128. Posun čísiel môže byť aj iný. Ak chceme napríklad reprezentovať viac kladných ako záporných čísiel, môžeme spraviť menší posun, napríklad o $(64)_{10}$. Rozsah platných čísiel v posuvnom 8 bitovom EXCESS-128 kóde je $(-128)_{10}$ s kódom 0000 0000 až $(127)_{10}$ s kódom 1111 1111.

Aritmetické operácie v posuvnom kóde EXCESS-128 vedú veľmi často k prekročeniu 8 bitov. Stačí, aby sme sčítali ľubovoľné dve kladné čísla a výsledkom sčítania ich dvoch 8 bitových kódov, podľa pravidiel sčítania binárnych čísiel je nejaké 9 bitové číslo. Je to preto, že pri sčítaní dvoch posuvných kódov čísiel x a y vykonávame súčet dvoch posuvných kódov (x+posun) a (y+posun). Výsledkom je (x+y+2·posun). Aby sme mali výsledok opäť v posuvnom kóde potrebujeme od (x+y+2·posun) odčítať posun, aby bol výsledok (x+y+posun). Na korektný súčet v posuvnom kód teda potrebujeme dve operácie. Podobne je na tom odčítanie. Výsledkom rozdielu dvoch kódov (x+posun) a (y+posun) je (x-y). Aby sme dostali výsledok v posuvnom kóde, teda (x-y+posun), potrebujeme ešte pripočítať posun. Opäť potrebujeme dve operácie.

1.3.3 Inverzný kód

Dva príklady sčítania čísiel v inverznom 8 bitovom kóde:

```
  1111 1100 t.j.(-3)10
 +0000 0010 t.j.(2)10
  ----------
  1111 1110 t.j.(-1)10
```

```
  0000 0011 t.j.(3)10
 +1111 1101 t.j.(-2)10
  ----------
1|0000 0000
        +1 prenos
  ----------
  0000 0001 t.j.(1)10
```

Inverzný kód je založený na tom, že záporné číslo -x reprezentujeme tak, že zapíšeme kladné číslo x v dvojkovej sústave a zmeníme v ňom všetky bity na opačné. Táto zmena sa jednoducho vykoná negáciou všetkých bitov reprezentácie čísla v dvojkovej sústave. Napríklad číslo $(6)_{10}$ má kód 0000 0110 a číslo $(-6)_{10}$ má kód 1111 1001. Podobne ako v znamienkovom kóde máme dve nuly (-0) s kódom 1111 1111 a (+0) s kódom 0000 0000. Rozsah platných čísiel inverzného 8 bitového kódu je $(-127)_{10}$ s kódom 1000 0000 až $(127)_{10}$ s kódom 0111 1111.

Práve existencia dvoch kódov pre nulu spôsobuje problémy pri sčítavaní. Ak sčítanie dvoch čísiel spôsobí prekročenie rozsahu, je potrebné k výsledku pripočítať 1 - takzvaný *prenos* (carry bit), aby sme mali správny výsledok v inverznom kóde.

Rozdiel sa realizuje tak, že druhému z čísiel, menšiteľu, zmeníme všetky bity na opačné a namiesto rozdielu urobíme súčet.

1.3.4 Doplnkový kód

Doplnkový kód sa používa na reprezentáciu celých čísiel takmer vo všetkých typoch

počítačov. Vychádza z inverzného kódu. Rozdiel oproti inverznému kódu spočíva v tom, že ku kódom záporných čísiel v inverznom kóde pripočítame jednotku. Napríklad ak chceme reprezentovať číslo $(-6)_{10}$ v doplnkovom 8 bitovom kóde, v binárnej reprezentácii čísla $(6)_{10}$= $(0000\ 0110)_2$ zmeníme bity na opačné t.j. 1111 1001 a pripočítame 1. Dostaneme doplnkový kód čísla $(-6)_{10}$: 1111 1010.

Doplnkový kód obsahuje iba jeden kód pre nulu a to 0000 0000. Obrovskou výhodou doplnkového kódu, na rozdiel od predchádzajúcich kódov pre celé čísla, je to, že po sčítaní alebo odčítaní máme okamžite správny výsledok v doplnkovom kóde. Pokiaľ pri sčítaní alebo odčítaní došlo k prekročeniu rozsahu dĺžky reprezentácie (napr. ak sčítaním dvoch 8 bitových kódov dostaneme 9 bitové číslo), môžeme prenos (carry bit) ignorovať, pretože bez neho máme správny výsledok. Rozsah platných čísiel reprezentovaných doplnkovým 8 bitovom kódom je $(-128)_{10}$ s kódom 1000 0000 až $(127)_{10}$ s kódom 0111 1111.

Ak výsledok aritmetickej operácie sa nedá reprezentovať daným počtom bitov hovoríme o prekročení rozsahu platných čísiel. Táto situácia sa dá zistiť tak, že sčítaním dvoch kódov kladných čísiel nám vznikne kód záporného čísla, respektíve sčítaním kódov dvoch záporných čísiel vznikne kód kladného čísla.

Nasledujúca tabuľka sumarizuje kódy čísiel $(-4)_{10}$ až $(4)_{10}$ pre všetky spomínané typy kódov celých čísiel s rozsahom 8 bitov.

Číslo	Znamienkový kód	Posuvný kód EXCESS-128	Inverzný kód	Doplnkový kód
4	0000 0100	1000 0100	0000 0100	0000 0100
3	0000 0011	1000 0011	0000 0011	0000 0011
2	0000 0010	1000 0010	0000 0010	0000 0010
1	0000 0001	1000 0001	0000 0001	0000 0001
0	0000 0000	1000 0000	0000 0000	0000 0000
	1000 0000		1111 1111	
-1	1000 0001	0111 1111	1111 1110	1111 1111
-2	1000 0010	0111 1110	1111 1101	1111 1110
-3	1000 0011	0111 1101	1111 1100	1111 1101
-4	1000 0100	0111 1100	1111 1011	1111 1100

Na záver kapitoly o reprezentácii celých čísiel si zosumarizujme typy premenných v ObjectPascal-e, ktorých premenné uchovávajú celé čísla.

Typ	Binárny rozsah (doplnkový kód)	Dekadický rozsah
shortint	1000 0000 až 0111 1111	-128 až 127
smallint	1000 0000 0000 0000 až 0111 1111 1111 1111	-32768 až 32767
integer	1000 0000 0000 0000 0000 0000 0000 0000 až 0111 1111 1111 1111 1111 1111 1111 1111	-2147483648 až 2147483647

Typy, ktorých premenné sú schopné uchovávať kladné aj záporné čísla sa nazývajú aj znamienkové (signed) typy.

Úloha 1.	Sčítajte v inverznom kóde čísla $(22)_{10}$ a $(-17)_{10}$.
Úloha 2.	Sčítajte v doplnkovom kóde čísla $(22)_{10}$ a $(-17)_{10}$.
Úloha 3.	Zistite aký typ kódu celých čísiel používa program Kalkulačka v OS Windows.

Dva príklady sčítania čísiel v doplnkovom 8 bitovom kóde:

```
  1111 1101  t.j.(-3)10
+ 0000 0010  t.j.(2)10
  ---------
  1111 1111  t.j.(-1)10


  0000 0011  t.j.(3)10
+ 1111 1110  t.j.(-2)10
  ---------
1 0000 0001  t.j.(1)10
- prenos ignorujeme
```

Príklad riešenia úlohy 3	Zadajte záporné binárne číslo (použitím tlačidla +/-) napríklad -10 a stlačte tlačidlo =.

1.3.5 Násobenie a delenie celých čísiel

Násobenie čísiel v dvojkovej sústave je, v porovnaní s násobením v desiatkovej sústave, veľmi jednoduché. Aj v tomto prípade je postup podobný tomu, ktorý sa učí na základnej škole. Napíšeme si obe násobené čísla pod seba a každú cifru druhého čísla vynásobíme s každou cifrou prvého čísla. Nakoľko v prípade binárnych reprezentácií čísiel násobíme buď jednotkou alebo nulou tak výsledkom násobenia cifry s prvým číslom je buď nula alebo celé prvé číslo bez zmeny iba s príslušným posunom doľava.

Násobenie si ukážeme na príklade násobenia čísiel $(-3)_{10}$ a $(10)_{10}$ v doplnkovom 8 bitovom kóde. V zápise uvádzame aj bity mimo rozsah, tie sa však pri výpočte ignorujú.

```
      1111 1101 t.j. (-3)2
     ·0000 1010 t.j. (10)2
      ---------
      0000 0000
+   1|1111 1010
+  00|0000 0000
+111|1110 1000
      ---------

      1110 0010 t.j. (-30)2
```

Samotné počítanie súčtov sa deje postupne po dvojiciach. Reálne pri násobení dvoch 8 bitových čísiel potrebujeme maximálne 7 posunov prvého čísla doľava a 7 sčítaní.

Aj na delenie sa používa algoritmus podobný tomu zo základnej školy. Delenie prevádzame na nezáporných číslach. Pri výpočte zisťujeme koľkokrát sa v pomocnom čísle, predstavujúcom na začiatku delenia úvodnú časť delenca, nachádza deliteľ a takú cifru zapisujeme na koniec výsledného čísla. Pri binárnom delení sa deliteľ nachádza v časti delenca nula krát alebo raz. Ak sa nachádza raz, odpočítame deliteľa od tohto pomocného čísla, ak nula krát, nerobíme nič, a nakoniec pridáme k pomocnému číslu ďalšiu cifru z delenca.

Vydeľme čísla $(29)_{10}=(11101)_2$ a $(5)_{10}=(00101)_2$ ako dve 5 bitové čísla.

```
     11101 : 00101 = 00101 = (5)10 = (29 div 5) v jazyku Pascal
 00001
-00000
------
  00011
 -00000
 ------
   00111
  -00101
  ------
    00100
   -00000
   ------
     01001
    -00101
    ------
     00100 = (4)10 zvyšok = (29 mod 5) v jazyku Pascal
```

Popíšme si, ako sme postupovali.

- Vzali sme pomocné číslo nula a prvú cifru delenca sme nasunuli do tohto čísla sprava, čím sme získali kód 00001.
- Porovnali sme pomocné číslo 00001 s deliteľom 00101 a zistili, že deliteľ je väčší. Do výsledku sme si zapísali ako prvú cifru nula.
- Pridali sme do pomocného čísla 00001 sprava ďalšiu cifru delenca a získali kód 00011.
- Porovnali sme pomocné číslo 00011 s deliteľom 00101 a zistili, že deliteľ je väčší. Do výsledku sme si zapísali ako druhú cifru nula.
- Pridali sme do pomocného čísla 00011 sprava ďalšiu cifru delenca a získali kód 00111.
- Porovnali sme pomocné číslo 00111 s deliteľom 00101 a zistili, že deliteľ je menší. Do výsledku delenia sme si zapísali ako tretiu cifru jedna. Vykonali sme rozdiel čísiel 00111 a 00101, čím sme dostali nové pomocné číslo 00010.
- Pridali sme do pomocného čísla 00010 sprava ďalšiu cifru delenca a získali kód 00100.
- Porovnali sme číslo 00100 s deliteľom 00101 a zistili, že deliteľ je väčší. Do výsledku sme si zapísali ako štvrtú cifru nula.
- Pridali sme do pomocného čísla 00100 sprava poslednú cifru delenca a získali kód 01001.
- Porovnali sme pomocné číslo 01001 s deliteľom 00101 a zistili, že deliteľ je menší. Do výsledku delenia sme si zapísali ako poslednú piatu cifru jedna. Vykonali sme rozdiel čísiel 01001 a 00101 čím sme dostali pomocné číslo 00100, ktoré predstavuje zvyšok po delení.

Určite by ste vedeli naprogramovať podľa tohto postupu delenie aj na ľubovoľný počet desatinných miest.

Pri delení potrebujeme pre každú cifru 3 až 4 operácie: posun jednej cifry z delenca do pomocného čísla, porovnanie pomocného čísla a deliteľa, posun jednej cifry do výsledku a prípadný rozdiel dvoch čísiel.

1.4 Reprezentácia čísiel s pohyblivou rádovou čiarkou

Doteraz ste prevody z desiatkovej do dvojkovej sústavy počítali pre celé čísla. V prípade racionálnych a reálnych čísiel je potrebné robiť aj prevod časti čísla za desatinnou čiarkou.

Celé časti ste najčastejšie počítali postupným delením dvojkou. Celé číslo v desiatkovej sústave prevedieme do dvojkovej sústavy tak, že toto číslo postupne celočíselne delíme dvomi a zapisujeme si zvyšky po delení sprava doľava až pokiaľ výsledok delenia nie je nula.

29 : 2 = 14 zvyšok 1
14 : 2 = 7 zvyšok 0
7 : 2 = 3 zvyšok 1
3 : 2 = 1 zvyšok 1
1: 2 = 0 zvyšok 1
Z toho $(29)_{10} = (11101)_2$

Ak potrebujeme spraviť prevod aj v časti čísla za desatinnou čiarkou, namiesto delenia dvojkou potrebujeme dvojkou násobiť – teraz však už v obore reálnych čísiel. Ak po násobení dostaneme číslo väčšie ako 1, zapíšeme si na koniec binárneho čísla číslo 1 a túto jednotku od čísla odrátame. Ak po násobení dostaneme číslo menšie ako 1, iba si zapíšeme 0 na koniec binárneho čísla. Výpočet robíme dovtedy, pokiaľ násobením dvojkou nedostaneme výsledok presne 1. V tomto prípade už iba dopíšeme túto jednotku na koniec binárneho čísla.

0,15625 · 2 = **0**,3125
0,3125 · 2 = **0**,625
0,625 · 2 = **1**,25
0,25 · 2 = **0**,5
0,5 · 2 = **1**
Z toho $(0{,}15625)_{10} = (0{,}00101)_2$

Na rozdiel od prevodu zápisu celej časti čísla v desiatkovej sústave do zápisu v dvojkovej sústave, algoritmus prevodu časti za desatinnou čiarkou nemusí vždy skončiť. Napríklad, ak prevádzame do dvojkovej sústavy číslo $(0,1)_{10}$, zistíme, že v dvojkovej sústave má toto číslo nekonečný rozvoj. Z toho napríklad vyplýva, že číslo $(0,1)_{10}$ nevieme v dvojkovej sústave reprezentovať presne.

0,1 · 2 = **0**,2
0,2 · 2 = **0**,4
0,4 · 2 = **0**,8
0,8 · 2 = **1**,6
0,6 · 2 = **1**,**2**
0,2 · 2 = **0**,**4**
…
Z toho $(0,1)_{10} =$
$(0{,}0\ 0011\ 0011\ 0011\ \ldots)_2$

Reálne čísla sa v počítačoch najčastejšie reprezentujú vo forme čísiel s pohyblivou rádovou čiarkou v tvare $m \cdot 2^e$, kde m sa nazýva **mantisa** a e **exponent**. Napríklad číslo $(5{,}375)_{10} = (101{,}011)_2$ sa môže reprezentovať ako $101{,}011 \cdot 2^0$, alebo ako $101011 \cdot 2^3$. V praxi sa na reprezentáciu používa **normalizovaný** (kánonický) tvar v tvare $1{,}xxx\ldots x \cdot 2^e$ kde xxx…x je nejaká postupnosť jednotiek a núl. Číslo $(101{,}011)_2$ má normalizovaný tvar $1{,}01011 \cdot 2^2$.

Podľa normy IEEE-754-2008, ktorá je podporovaná mnohými programovacími jazykmi (aj Object Pascalom), ale hlavne aj hardvérom, sa reprezentácia čísiel s pohyblivou rádovou čiarkou skladá z troch častí. Prvá časť je znamienkový bit, určujúci kladnosť čísla. Druhá časť predstavuje exponent (ktorý môže byť kladný alebo záporný) v posuvnom kóde. Poslednou časťou reprezentácie, teda mantisou, je desatinná časť normalizovaného tvaru čísla, t.j. jednotka pred desatinnou čiarkou sa z reprezentácie vynecháva.

V Object Pascale, ako aj v mnohých iných jazykoch, je najmenšia šírka reprezentácie čísla s pohyblivou rádovou čiarkou 4 bajty, teda 32 bitov. Je to typ **`single`**. Prvý bit je znamienkový. Ďalších 8 bitov je exponent v posuvnom EXCESS-127 kóde. Posledných 23 bitov obsadzuje mantisa.

Napíšme si, ako bude vyzerať reprezentácia čísla $(5{,}375)_{10} = (101{,}011)_2$. Jeho normalizovaný tvar je $1{,}01011 \cdot 2^2$. Je to kladné číslo takže znamienkový bit bude 0. Ďalej potrebujeme zapísať $(2)_{10}$ v posuvnom EXCESS-127 kóde. K jeho binárnej reprezentácii $(10)_2$ pripočítame posun $(127)_{10}=(0111\ 1111)_2$, čím získame kód 1000 0001. Na záver opíšeme cifry za desatinnou čiarkou z normalizovaného tvaru a zvyšok doplníme nulami.

0	1000 0001	01011 00000 00000 00000 000

Nasledujúca tabuľka obsahuje rôzne extrémne prípady čísiel typu single.

Prípad	Binárna reprezentácia	Približná hodnota v desiatkovej sústave
najväčšie konečné	0|1111 1110|11111 11111 11111 11111 111	$3.4028235 \cdot 10^{38}$
najmenšie konečné	1|1111 1110|11111 11111 11111 11111 111	$-3{,}4028235 \cdot 10^{38}$
najmenšie kladné	0|0000 0001|00000 00000 00000 00000 000	$1{,}4012985 \cdot 10^{-45}$
najväčšie záporné	1|0000 0001|00000 00000 00000 00000 000	$-1{,}4012985 \cdot 10^{-45}$
kladná nula	0|0000 0000|00000 00000 00000 00000 000	+0
záporná nula	1|0000 0000|00000 00000 00000 00000 000	-0
plus nekonečno	0|1111 1111|00000 00000 00000 00000 000	dostaneme napríklad výpočtom 1 / 0
mínus nekonečno	1|1111 1111|00000 00000 00000 00000 000	dostaneme napríklad výpočtom -1 / 0
Nie je číslo (NaN)	x|1111 1111|1xxxx xxxxx xxxxx xxxxx xxx kde hodnoty na pozíciách x môžu byť ľubovoľné	dostaneme napríklad výpočtom 0 / 0

Object Pascal pozná ešte typ `real`, ktorý sa v závislosti od hosťovského počítača berie buď ako `single` alebo ako `double`. Na súčasných počítačoch je to väčšinou `double`.

Okrem typu **`single`** pozná Object Pascal na reprezentáciu čísiel s pohyblivou rádovou čiarkou ešte typ **`double`** so šírkou 8 bajtov teda 64 bitov. Ten sa tiež skladá zo znamienkového bitu, nasleduje 11 bitov na exponent v posuvnom kóde EXCESS-1023 a z 52 bitov na mantisu.

Norma IEEE-754-2008 definuje aj 128 bitové a 16 bitové reprezentácie čísiel s pohyblivou rádovou čiarkou, ale mnohé jazyky ich nepodporujú. Dôvodom je pravdepodobne to, že v súčasnosti má väčšina počítačov 32 alebo 64 bitové architektúry a aj samotné procesory majú pre operácie s týmito veľkosťami čísiel s pohyblivou rádovou čiarkou špeciálnu podporu.

Pri číslach s pohyblivou rádovou čiarkou je dôležité si uvedomiť to, že aj keď vieme

reprezentovať čísla od veľmi veľkých po veľmi malé (pri type `double` to je zhruba 10^{308} až 10^{-324}), nevieme reprezentovať číslo, ktoré je zároveň veľké a zároveň presné za desatinnou čiarkou. Mantisa má iba obmedzenú veľkosť s presnosťou na 23 (typ `single`) resp. 52 (typ `double`) binárnych cifier. Keďže na zápis toho istého čísla v dvojkovej sústave potrebujeme viac bitov ako v desiatkovej sústave, tak v desiatkovej sústave máme k dispozícii iba zhruba 7 (typ `single`) resp. 16 (typ `double`) platných cifier.

Najväčšie číslo, ktoré vieme reprezentovať na 23 binárnych pozíciách je 2^{23}-1= $(8\ 388\ 607)_{10}$, ktoré má 7 cifier.

1.4.1 Aritmetické operácie s číslami s pohyblivou rádovou čiarkou

Pri súčte a rozdiele dvoch čísiel uložených v svojich normalizovaných tvaroch musíme pred samotnou operáciou súčtu resp. rozdielu dostať obe čísla na rovnaký exponent, aby jednotlivé pozície mantís oboch čísiel zodpovedali tým istým pozíciám pred alebo za desatinnou čiarkou.

Pred súčtom alebo rozdielom prevedieme takzvanú denormalizáciu, to znamená, že číslo s nižším exponentom zmeníme tak, aby obe čísla mali rovnaký exponent. Naviac spravíme aj to, že v oboch mantisách zobrazíme aj tú jednotku pred desatinou čiarkou, ktorú sme do reprezentácie v normalizovanom tvare nezahrnuli.

Sčítajme dve čísla $(45{,}36328125)_{10}$ = $(101101{,}01011101)_2$ a $(2{,}708984375)_{10}$ = $(10{,}101101011)_2$. Ich reprezentácia v single kódovaní je nasledovná:

V dvojkovej sústave	Normalizovane	Single kódovanie
101101,01011101	1,01101 01011 101·2^5	0|1000 0100|01101 01011 10100 00000 000
10,101101011	1,01011 01011 ·2^1	0|1000 0000|01011 01011 00000 00000 000

Obe čísla teraz musíme normalizovať. Prvému číslu, ktoré má väčší exponent dodáme na začiatok mantisy 1 a zvyšok mantisy posunieme doprava. Druhému číslu zvyšujeme exponent o $(4)_{10}$, to znamená, že po pridaní 1 na začiatok mantisy musíme túto mantisu ešte posunúť o 4 miesta doprava (posúvame desatinnú čiarku o 4 miesta doľava). Stratíme tak informáciu o posledných 5 binárnych cifrách.

Čo by sa stalo ak by sme sčítali čísla 1,0·2^{15} a 1,0·2^{-15}?

Po denormalizácii vykonáme súčet mantís a výsledok opäť normalizujeme, t.j. odstránime z ľavej strany mantisy jednotku a hodnoty v mantise posunieme doľava a úplne napravo dodáme nulu.

V dvojkovej sústave	Denormalizovane	Single kódovanie po denormalizácii
101101,01011101	1,01101 01011 1010 ·2^5	0|1000 0100|10110 10101 11010 00000 000
10,101101011	0,00010 10110 1011 ·2^5	0|1000 0100|00001 01011 01011 00000 000
Súčet:	1,10000 00010 0101 ·2^5	0|1000 0100|11000 00001 00101 00000 000
Normalizácia:		0|1000 0100|10000 00010 01010 00000 000

Pri súčte mantís by sa mohlo stať, že nastane pretečenie rozsahu mantís. V tomto prípade sa táto jednotka pripočíta k exponentu.

Rozdiel dvoch čísiel sa vykonáva úplne analogicky ako súčet.

Pri výpočte súčinu a podielu nepotrebujeme obe čísla dostať na rovnaký exponent. Ak chceme urobiť súčin čísiel m·2^e a n·2^f tak výsledok má tvar (m·n)·2^{e+f}. Stačí ak vykonáme súčet exponentov v posuvnom kóde a vynásobíme medzi sebou mantisy. Samozrejme potrebujeme ešte upraviť znamienko, t.j. prvý bit, na čo použijeme logickú funkciu XOR. Samotné násobenie a sčítanie sa už vykonáva v obore celých a nezáporných čísiel.

Všimnite si, že XOR zodpovedá pravidlám úprav znamienok pri násobení alebo delení.

0 XOR 0 = 0 + · + > +
1 XOR 0 = 1 — · + > —
0 XOR 1 = 1 + · — > —
1 XOR 1 = 0 — · — > +

Podiel čísiel m·2^e a n·2^f sa vypočíta ako (m:n)·2^{e-f}, takže urobíme rozdiel exponentov v obore celých čísiel a podiel mantís v obore nezáporných čísiel. Úprava výsledného znamienka sa riadi rovnakými pravidlami ako pri násobení.

Úloha 1.	Preveďte číslo $(43{,}625)_{10}$ do dvojkovej sústavy.
Úloha 2.	Zapíšte číslo $(43{,}625)_{10}$ v single kódovaní.
Úloha 3.	Vyskúšajte si, ako bude prebiehať súčet čísiel $(43{,}625)_{10}$ a $(1024)_{10}$ typu single .

1.4.2 Nepresnosť pri počítaní s číslami s pohyblivou rádovou čiarkou

Pri operáciách s číslami typu `double` alebo `single` treba mať stále na pamäti, že pri mnohých situáciách dochádza k zaokrúhľovaniu či odrezaniu časti mantisy. Ako sme si ukázali, už pri prevode zápisu niektorých čísiel s malým počtom desatinných miest z desiatkovej do dvojkovej sústavy musí nutne dôjsť k strate presnosti na toľko rádových miest binárneho čísla, aká je veľkosť mantisy.

Keď vykonávame aritmetické operácie s takýmito nepresnými číslami musíme počítať s tým, že aj výsledok môže nepresný. K ďalšiemu skresleniu môže dôjsť pri prevode naspäť do desiatkovej sústavy, v ktorej obvykle zobrazujeme výsledok.

K najvýraznejšej strate presnosti dochádza pri súčte a rozdiele dvoch čísiel s rôznymi exponentmi. Ak je rozdiel exponentov väčší ako šírka mantisy, menšie číslo je pri denormalizácii zaokrúhlené na nulu.

Aktivita 1

Vyskúšajte si spustiť nasledujúcu procedúru, ktorá demonštruje chyby pri sčítaní zaokrúhlených čísiel. V tejto procedúre vykonávame súčet $\sum_{i=1}^{n} \frac{1}{n}$, ktorý je v obore racionálnych čísiel (bez zaokrúhľovania) pre ľubovoľné n>0 rovný jednej. Test robíme pre všetky $n \in \langle 1, pocetTestov \rangle$.

```
Procedure TestSuctu;
 var
  i, n, pocetTestov : Integer;
  s : Single;
  d : Double;
 begin
  pocetTestov := 100;
  for n := 1 to pocetTestov do
    begin
     s := 0;
     d := 0;
     for i := 1 to n do
       begin
        s := s + 1 / n;
        d := d + 1 / n;
       end;
     Form1.Memo1.Append(IntToStr(i) + ': ' +
       FloatToStr(s) + ' ' + FloatToStr(d));
    end;
 end;
```

Poznámka: funkcia `FloatToStr` v Object Pascale vracia výsledok pre typ `single` s presnosťou na 9 desatinných miest a pre typ `double` s presnosťou na 15 desatinných miest. Povedali sme, že počet platných desatinných miest pre typ `single` je zhruba 7 a pre typ `double` zhruba 16. Z toho dôvodu má typ `single` vo výpise častejšie skreslenie.

1.5 Logické a posuvné operácie

S logickými operáciami sa bežne stretávame v matematickej logike a ako programátori pri vyhodnocovaní podmienok v podmienených príkazoch (If) a v podmienených cykloch (While). Kľúčovým pojmami s ktorými pracujeme sú **pravda** a **nepravda**.

Mnoho programovacích jazykov obsahuje špeciálny typ premenných Boolean v ktorých môžeme uchovávať buď hodnotu True (pravda) alebo False (nepravda). Nakoľko najmenšia adresovateľná jednotka v pamäti počítača je jeden bajt, aj tieto premenné zaberajú v pamäti jeden bajt, aj keď na reprezentáciu tohto rozsahu hodnôt by nám stačil jeden bit.

V programovacích jazykoch sú obvykle na prácu s premennými typu Boolean k dispozícii 4 základné logické spojky (operátory) AND, OR, NOT a XOR (vylučujúci OR). Ak potrebujeme iné logické spojky, dokážeme ich z týchto základných štyroch odvodiť. Napríklad implikáciu môžeme vyjadriť kombináciou spojok OR a NOT, lebo vieme, že platí: $(A \Rightarrow B) \Leftrightarrow (\neg A \vee B)$.

Pri vyhodnocovaní logických operácií nad hodnotami typu Boolean stačí pracovať s informáciou dĺžky jeden bit. V mnohých situáciách sa využíva, že v procesore môžeme vykonávať logické operácie paralelne nad viacerými bitmi. V bežných počítačoch je to 32 alebo až 64 bitov. Ako premenné, v ktorých vieme uchovať viac pravdivostných hodnôt súčasne, sa používajú obyčajné celočíselné premenné. Napríklad v premennej typu `byte` vieme uložiť osem pravdivostných hodnôt, kde na príslušnej pozícii 1 znamená pravda a 0 znamená nepravda.

Logické spojky AND, OR, NOT a XOR môžeme použiť aj ako operátory vo výrazoch s číslami. Napríklad logické operácia $(24)_{10}$ AND $(15)_{10}$ dá ako výsledok $(8)_{10}$, lebo vykonáme logický AND pre každú bitovú pozíciu zvlášť. Ukážme si aký výsledok pre tieto dve čísla dajú všetky tri binárne operátory:

```
   (11000)2 = (24)10      (11000)2 = (24)10      (11000)2 = (24)10
AND (1111)2 = (15)10   OR (1111)2 = (15)10  XOR (1111)2 = (15)10
    -----                  -----                  -----
   (01000)2 = (8)10       (11111)2 = (31)10      (10111)2 = (23)10
```

Operátor NOT je unárny a mení všetky bity svojho operandu na opačné. Napríklad NOT $(24)_{10}$, kde 24 je hodnota premennej typu byte, teda má kód 0001 1000, vráti $(1110\ 0111)_2=(231)_{10}$. Ak by však táto premenná bola typu shortint (v doplnkovom kóde), výsledok 1110 0111 by predstavoval hodnotu $(-25)_{10}$.

K týmto 4 základným logickým operátorom máme v prípade viacbitových logických operácií k dispozícii ešte operátory posunu SHR (shift right) - posun doprava a SHL (shift left) - posun doľava. Oba operátory posunú všetky binárne cifry príslušným smerom o daný počet pozícií. Cifry, ktoré sa posunom dostanú mimo rozsah premennej sa ignorujú. Na voľné miesta na druhej strane bitového zápisu , uvoľnené posunom, sa zapíšu nuly.

Operácia	Pred posunom	Po posune
25 SHL 1	0001 1001	0011 0010 t.j. $(50)_{10}$
25 SHL 4	0001 1001	1001 0000 t.j. $(144)_{10}$
25 SHR 2	0001 1001	0000 0110 t.j. $(6)_{10}$

1.5.1 Použitie logických a posuvných operácií

Logické a posuvné operácie sa používajú obvykle v situáciách, kde je potrebné urýchliť nejakú operáciu, ktorú je potrebné vykonať veľmi veľakrát v krátkom čase.

Jednoduchým príkladom je použitie posunu o 1 vľavo v prípade, že potrebujeme násobiť dvomi. Napríklad platí, že (41 SHL 1) = 82. Je to rovnaká „finta“, ako keď

False má kód 0000 0000, a True je obvykle reprezentované ako 0000 0001. Mnohé jazyky však pokladajú za True akýkoľvek kód veľkosti jeden bajt rôzny od 0000 0000.

Prehľadná tabuľka pravdivostných hodnôt všetkých binárnych logických spojok a ich obvyklé označenie:

A	0	0	1	1
B	0	1	0	1
False	0	0	0	0
A AND B	0	0	0	1
¬(A⇒B)	0	0	1	0
A	0	0	1	1
¬(B⇒A)	0	1	0	0
B	0	1	0	1
A XOR B	0	1	1	0
A OR B	0	1	1	1
A NOR B	1	0	0	0
A⇔B	1	0	0	1
¬B	1	0	1	0
B⇒A	1	0	1	1
¬A	1	1	0	0
A⇒B	1	1	0	1
A NAND B	1	1	1	0
True	1	1	1	1

v desiatkovej sústave pri násobení desiatimi jednoducho dopíšeme na koniec čísla nulu. Napríklad, určite bez zaváhania viete prehlásiť, že platí (41 · 10) = 410.

Podobným spôsobom v desiatkovej sústave delíme desiatimi. Jednoducho posunieme desatinnú čiarku doľava. V prípade, že poslednú cifru jednoducho škrtneme spravíme vlastne celočíselné delenie desiatimi, napríklad (41 div 10) = 4. Ak použijeme bitový posun o jedno miesto doprava, urobíme tak celočíselné delenie dvomi, napríklad (41 SHR 1) = 20. Ak si spomeniete, že za normálnych okolností si celočíselné delenie vyžaduje pre každú cifru 4 operácie, tak v prípade premennej typu `longword`, ktorá má 32 bitov potrebujeme na delenie až (32·4)=128 operácií. Pomocou posunu vykonáme celočíselné delenie dvomi iba jednou operáciou. Cifra, ktorú posunutím stratíme, predstavuje zvyšok po delení.

Zvyšok po delení dvomi môžeme vypočítať cez logickú operáciu AND s číslom 1

```
     0010 1001 = (41)10
AND  0000 0001 = (1)10
     ---------
     0000 0001 = (1)10
```

Zmenou druhého argumentu posunu môžeme vykonať násobenie alebo celočíselné delenie aj s inými násobkami dvojky.

Známym využitím viacbitových logických operácií je práca s internetovými adresami. Pri nastavovaní internetovej adresy počítača nastavujeme IP adresu a masku siete. IP adresa aj maska siete sa skladajú zo štyroch osembitových čísiel, ktoré sa zapisujú vedľa seba oddelené bodkami. Predstavme si, že máme IP adresu 158.197.31.4 s maskou 255.255.255.0. V dvojkovej sústave táto maska vyzerá nasledovne: 11111111.11111111.11111111.00000000. Ak chceme z počítača s touto adresou komunikovať s iným počítačom, napríklad 158.197.31.66, potrebujeme z určitých dôvodov vedieť, či je tento druhý počítač v rovnakej sieti alebo nie. Test urobíme tak, že urobíme logický AND masky s oboma číslami. Ak je výsledok týchto operácií rovnaký, počítače sú v rovnakej sieti. V tomto prípade je výsledkom oboch operácií 158.197.31.0, teda IP adresy 158.197.31.4 a 158.197.31.66 sú v rovnakej sieti.

Úloha 1

Vytvorte program, ktorý prekonvertuje ľubovoľné číslo typu `word` na pole dĺžky 16 s prvkami typu `boolean`, v ktorom každý prvok zodpovedá príslušnej bitovej hodnote v binárnej reprezentácii tohto čísla. Použite na to logické a posuvné operácie.

Operácie posunu a iné logické operácie sa využívajú aj pri násobení a delení celých čísiel a pri všetkých operáciách s číslami s pohyblivou rádovou čiarkou.

Logické operácie sa používajú napríklad aj pri indexovaní v databázach, výpočte hašovacích a kryptografických funkcií, ale to už je mimo rozsah tohto materiálu.

Čo sme sa naučili

Kódovaní znakov v počítači je veľmi veľa. Popri ASCII kódovaní a národných kódovaniach poznáme aj univerzálne kódovania UTF-8 a UTF-16.

Kódy nezáporných čísiel sa tvoria jednoducho ako binárne zápisy čísiel. Pri celých číslach už máme viac možností. Pre pohodlnosť aritmetických operácií sa presadil doplnkový kód.

Reálne čísla sa reprezentujú ako čísla s pohyblivou rádovou čiarkou v tvare $\text{mantisa} \cdot 2^{\text{exponent}}$. Ukázali sme si, prečo si treba dať pozor na nepresnosť pri počítaní s takými číslami.

Veľmi často sa využíva, že logické operácie sa dajú v počítačoch vykonávať paralelne. Logické operácie sa využívajú aj pri výpočte aritmetických operácií.

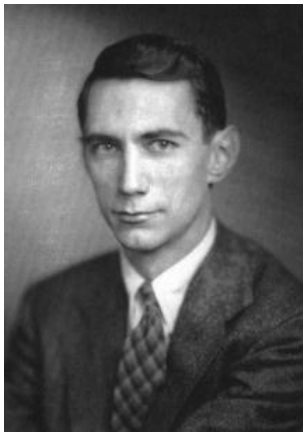
2. Logické obvody

2.1 História a vývoj technológií

Už oddávna sa ľudia snažili zautomatizovať rutinné výpočty, ktorých neustále pribúdalo. V 18. a 19. storočí sa objavili prvé mechanické zariadenia, uľahčujúce zložité numerické výpočty (mechanické kalkulátory. Babbageov diferenčný a analytický stroj). Základnými prvkami týchto zariadení boli väčšinou ozubené kolieska (prevzaté z hodinových strojov), prevody, posuvné mechanizmy, páky ...

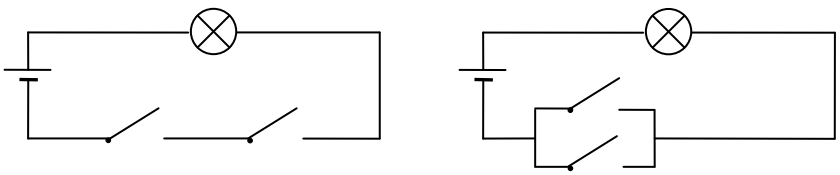
Časti diferenčného stroja (Charles Babbage)

Prelom v konštrukcii mechanických výpočtových zariadení (a v konečnom dôsledku definitívny prechod od analógových zariadení k digitálnym) spôsobili v prvej polovici 20. storočia prevratné objavy v elektrotechnike a prechod k binárnej reprezentácii spracovávaných údajov. V roku 1937 opísal 21-ročný Claude Shannon (jeden z duchovných otcov informatiky) v svojej diplomovej práci A Symbolic Analysis of Relay and Switching Circuits[1] spôsob vyjadrenia ľubovoľnej logickej funkcie pomocou základných logických funkcií AND, OR a NOT.

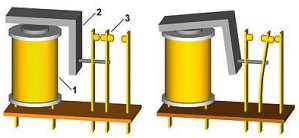


Claude Elwood Shannon (1916-2001) – budovateľ teoretických základov informatiky

Ukázal, že na realizáciu týchto základných funkcií nám potom postačia obyčajné spínače. V zapojení na ľavom obrábzku môže obvodom prechádzať prúd, ktorý rozsvieti žiarovku, len vtedy, keď zapneme súčasne obidva spínače. V zapojení vpravo postačí, keď zapneme ľubovoľný zo spínačov (a samozrejme, aj keď zapneme oba). Stav žiarovky v ľavom obvode závisí od stavov spínačov podobne, ako hodnota logickej funkcie AND a stav žiarovky v pravom obvode tak, ako hodnota logickej funkcie OR. Hovoríme, že obvod **realizuje** logickú funkciu, ak jeho výstupný stav závisí od vstupov tak, ako príslušná logická funkcia. Obvod vľavo je teda realizáciou funkcie AND a obvod vpravo realizáciou funkcie OR.



Shannon použil vo svojej práci namiesto výstupnej žiarovky elektromagnetické relé. Prechodom prúdu cez cievku (1) elektromagnetického relé sa zmagnetizuje jej oceľové jadro, ktoré pritiahne kotvu (2). Tým sa zopne kontakt (3). Ak elektrický prúd prestane cievkou prechádzať, kotva odpadne a kontakt (z pružného materiálu) sa vráti na pôvodné miesto, čím sa spojenie preruší. Relé namiesto žiarovky v našich AND a OR zapojeniach na základe stavu spínačov v našom obvode zopne spínače v iných obvodoch. Umožní nám skladať jednoduché obvody do väčších celkov, realizujúcich ľubovoľne zložité logické funkcie. Napríklad tiež aritmetické operácie (ako bolo ukázané v predchádzajúcej časti).



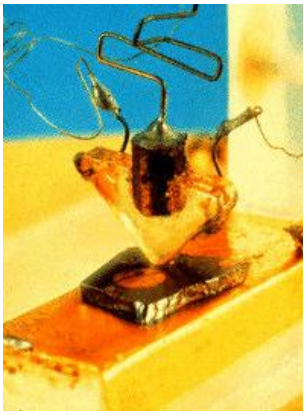
Elektromagnetické relé

Elektromagnetické relé sa aj skutočne v prvých počítačoch používali. Vzniknuté zariadenia však boli konštrukčne náročné, pomalé a spotrebovali veľa energie (nehľadiac na hluk, ktorý relé produkovali častým spínaním kontaktov).

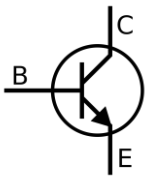
Elektrónka

V období 2. svetovej vojny sa začali konštruovať počítače tiež z elektrónok. Elektrónka je tiež jednoduché zariadenie, ktoré v správnom zapojení funguje ako spínač. Pokiaľ si spomínate, v elektrónke zahrievaním katódy vznikajú vo vákuu

[1] http://dspace.mit.edu/bitstream/handle/1721.1/11173/34541425.pdf?sequence=1
Táto Shannonova práca je považovaná za najlepšiu diplomovú prácu, ktorá bola kedy napísaná.

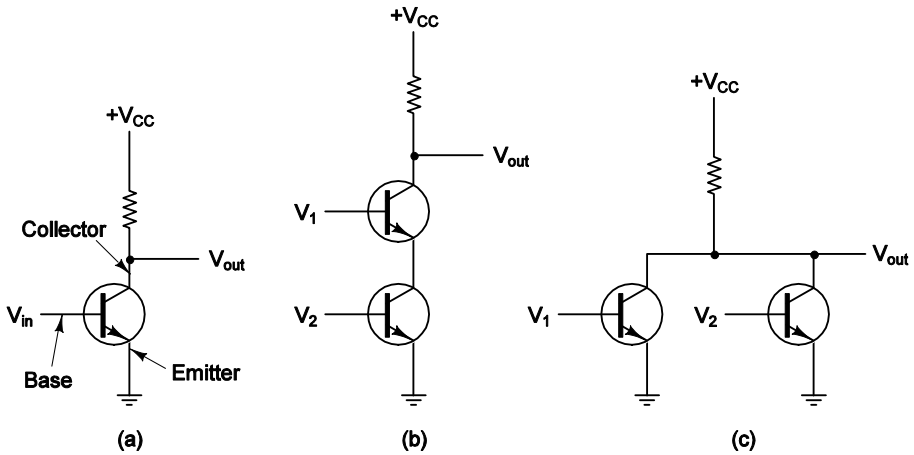


Bodový tranzistor (Shockley, Bardeen, Brattain - 1947)



voľné elektróny, ktoré sú priťahované ku kladne nabitej anóde. Mriežka, ktorá je medzi nimi, uzatvára alebo otvára tento prúd elektrónov a pôsobí ako vodovodný kohútik (podobne pracovali aj CRT (Cathode Ray Tube) monitory – mriežka vychyľovala postupne lúč elektrónov k fosforeskujúcim bodom na matnici obrazovky). Problémy s elektrónkovými zariadeniami boli hlavne v nespoľahlivosti, energetickej náročnosti, nutnosti chladenia. Výpočty boli síce rýchlejšie, ale rozmery sa podstatne nezmenšili.

Rozmach elektroniky a špeciálne aj digitálnej techniky nastal objavením tranzistorového efektu (1947). Prejavuje sa podobne ako v elektrónke tým, že prechod prúdu medzi kolektorom (C) a emitorom (E) je ovládaný napätím na báze (B). Tranzistor teda môžeme tiež použiť ako spínač. Navyše na vyvolanie efektu stačí niekoľko málo molekúl zlúčenín kremíka.



V_{in}	V_{out}
nízke	vysoké
vysoké	nízke

zapojenie (a) - NOT

V_1	V_2	V_{out}
0	0	V_{CC}
0	V_{CC}	V_{CC}
V_{CC}	0	V_{CC}
V_{CC}	V_{CC}	0

zapojenie (b) - NAND

V_1	V_2	V_{out}
0	0	V_{CC}
0	V_{CC}	0
V_{CC}	0	0
V_{CC}	V_{CC}	0

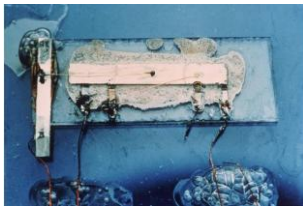
zapojenie (c) - NOR

Uvažujme zapojenie (a). Ak vstupné napätie na báze (V_{in}) je nízke (pod kritickou hodnotou), tranzistor je zatvorený a prúd medzi emitorom a kolektorom neprechádza. Tranzistor sa správa ako veľký odpor a výstupné napätie (V_{out}) bude blízke V_{CC} – napájaciemu napätiu. Keď vstupné napätie zvýšime nad kritickú hodnotu, tranzistor sa otvorí a začne sa správať medzi kolektorom a emitorom ako vodič. Výstupné napätie klesne k nule. V tomto jednoduchom zapojení tak dostaneme elektronický obvod, ktorý funguje ako *invertor* (logická funkcia NOT)- pokiaľ V_{in} je malé, V_{out} je veľké a naopak, pokiaľ V_{in} bude veľké, V_{out} bude malé.

V zapojení dvoch tranzistorov (b) bude pri nízkych napätiach na ktoromkoľvek zo vstupov V_1 a V_2 aspoň jeden z tranzistorov zatvorený, bude klásť prúdu veľmi veľký odpor, čo sa prejaví vysokým napätím na výstupe V_{out}. Ak ale privedieme na oba vstupy zvýšené napätie, tranzistory sa otvoria a prúd vyrovná rozdiel medzi V_{out} a uzemnením. Na výstupe V_{out} bude nízke napätie, blízke nule. Výstupné napätie sa zníži len keď sa otvoria všetky tranzistory - to odpovedá chovaniu logickej funkcie negácie konjunkcie - NAND.

Keď v zapojení (c) otvoríme ktorýkoľvek z tranzistorov (zvýšime vstupné napätie V_1 alebo V_2 nad kritickú hodnotu), začne ním pretekať prúd a na výstupe V_{out} dostaneme nízke hodnoty napätia. Pokiaľ budú obidva tranzistory zatvorené (vstupné napätia nízke), bude výstupné napätie blízke napájaciemu V_{CC}. Čo je ale to isté, ak by sme negovali výsledky obvodu OR – teda zapojenie je realizáciou funkcie NOR.

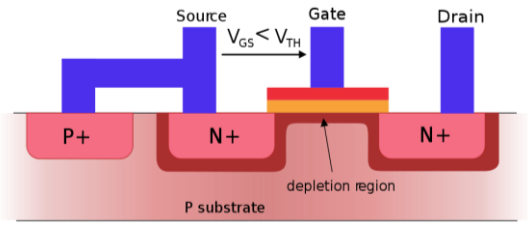
Podobne ako v prípade elektromagnetických relé môžem tieto jednoduché obvody komponovať do väčších celkov, výstupy jedných pripájať na vstupy iných a takto (podľa Shannona) vytvoriť obvody, realizujúce akékoľvek funkcie, ktoré pracujú s binárnymi vstupmi.



Prvý funkčný integrovaný obvod (Jack Kilby - 1958)

Ďalším veľkým technologickým krokom, ktorý umožnil súčasný rozmach výpočtovej techniky, bol objav vytvárania tranzistorov priamo na kremíkovej doske (J. Kilby, R. Noyce 1958). Pre technológie integrovaných obvodov boli vynájdené špeciálne typy unipolárnych komplementárnych tranzistorov (bez odporov a iných pasívnych prvkov)

typu MOSFET (Metal-Oxide-Semiconductor field-effect transistor), kde polovodičom (Semiconductor) je tenká kremíková doska, na ktorú sa nanášajú vrstvy kysličníka kremíka (Oxide), tvoriace elektródy typu Source (emitor) a Drain (kolektor).

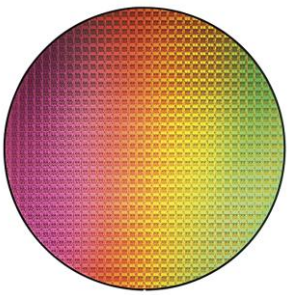


Prierez MOSFET tranzistorom

Vodivosť medzi elektródami Source a Drain je riadená elektrickým poľom, ktoré vznikne v okolí elektródy Gate (báza), tvorenej tenkou kovovou vrstvou (Metal) a dielektrikom. Výhodou je, že tranzistorový efekt (ovplyvňovanie, či v našom prípade zapínanie a vypínanie prúdu napätím na báze) nastáva už na veľmi malej ploche. Technologickým problémom ostáva naniesť na kremíkovú dosku príslušné prvky a navzájom ich na čo najmenšej ploche aj poprepájať.

Umelý kremíkový kryštál

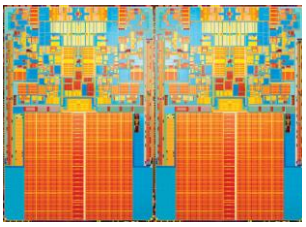
V súčasnosti prebieha výroba integrovaných obvodov tohoto typu tak, že sa z čistého (syntetického) kremíkového kryštálu odreže tenká kremíková doska (priemeru cca 30 cm) a na ňu sa postupne nanáša viacero vrstiev spojov pre niekoľko stovák integrovaných obvodov s veľmi vysokou integráciou (väčšinou rovnakého typu) metódami, podobnými tlačovej litografii. Nanášanie pozostáva z viacerých krokov, po každom sa doska testuje. Ak počet chybných obvodov presiahne určenú hranicu a ďalší postup nanášania sa už nevyplatí, celá doska sa zahodí. Pokiaľ chybných obvodov nie je veľa (vyťažiteľnosť je väčšinou lepšia ako 80% - ale závisí aj od použitej technológie a jej zvládnutia), doska sa nareže na jednotlivé obvody (chybné sa vyhodia), ktoré sú potom zapuzdrené. Samozrejme všetko musí prebiehať pri extrémnej čistote výrobného prostredia.



Kremíková doska (wafer) s procesormi technológie Intel Westmere (32 nm)

Súčasne používané technológie sú schopné vytvoriť na kremíkovej doske prepojenia so šírkou 32 nm. Pripravujú a testujú sa výrobné procesy, pracujúce až na rozlíšení 22 nm (pričom samotný atóm kremíka má rozmer 0,24 nm). Integrovaný obvod je možné súčasnými technológiami vybaviť až niekoľkými miliardami tranzistorov na ploche štvorcového centimetra (najhustejšie sa dajú integrovať obvody s pravidelnou štruktúrou – napríklad pamäte).

Nevýhodou polovodičov snáď je len to, že sú citlivé na prudké elektromagnetické poruchy (na rozdiel od elektrónok, ktoré by snáď prežili aj jadrovú vojnu). Exkurziu do fyziky a techniky ukončíme konštatovaním, že extrémna výkonnosť a kapacita súčasných počítačov je dôsledkom rozvoja technológií nanášania tenkých vrstiev a myšlienky realizácie zložitých logických funkcií pomocou jednoduchých obvodov.



820 miliónov tranzistorov štvorjadrového procesora Intel Core i7

Čo sme sa naučili

Každú zložitú logickú funkciu (ktorá počíta čokoľvek s binárnou reprezentáciou údajov) je možné vyjadriť pomocou jednoduchých logických funkcií. Pokiaľ vieme automaticky vyhodnocovať tieto jednoduché funkcie, vieme automaticky vykonávať aj akékoľvek zložité operácie s binárnymi údajmi.

Jednoduché logické funkcie môžeme realizovať (a automaticky vyhodnocovať) pomocou jednoduchých obvodov so spínačmi. Na realizáciu zložitých logických funkcií postačia tiež poprepájané spínače, no musíme ich mať dostatočné množstvo.

Súčasné technológie výroby integrovaných obvodov dokážu vyrobiť niekoľko miliárd spínačov (založených na princípe tranzistorového javu) a ich prepojení na ploche jedného štvorcového centimetra.

2.2 Jednoduché logické obvody

Elektronické zariadenia sú tvorené komponentmi, vzájomne prepojenými pomocou vodičov. Vodiče tvoria akési komunikačné kanály, cez ktoré je možné medzi komponentmi komunikovať. Komunikácia sa uskutočňuje zmenami elektrických parametrov prepojenia – väčšinou zmenami napätia. Jeden komponent na vodivom spoji parameter (napätie) nastaví, druhý môže na toto nastavenie patrične reagovať. Hodnota parametrov na spoji v čase sa teda mení – nazývame ju **signál**.

V minulosti boli konštruované aj analógové počítače. Umožňovali analógovo integrovať a derivovať vstupné signály, čo sa dalo využívať na riešenie diferenciálnych rovníc a podobne.

Signál je teda fyzikálna veličina a môže nadobúdať spojito všetky možné hodnoty. Elektronické zariadenie tak môže spojito reagovať na podnety a hneď ich aj patrične spracovať (napr. zosilňovač zvuku, chladič riadený otáčkami motora). Elektronické obvody, ktoré pracujú so spojitými signálmi, sa zvyknú označovať ako **analógové obvody**.

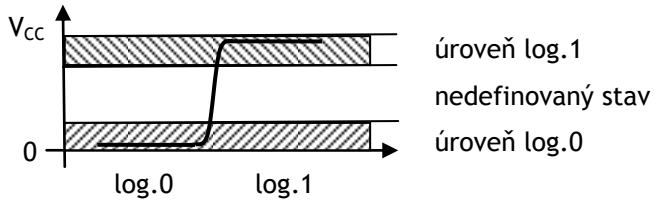
Aj spotrebná elektronika prechádza od analógového spracovania (gramofóny, magnetofóny, film) k číslicovému. Namiesto slovného spojenia číslicová technika sa (v duchu anglikanizácie) zvykne v súčasnosti používať spojenie digitálna technika.

(Možno presnejšie by malo znieť binárna technika, pretože v drvivej väčšine sa spracúva informácia v binárnom tvare)

Pre spracovanie údajov v číselnej forme nás nebudú zaujímať všetky možné hodnoty parametrov signálu, ale len hodnoty v určitých okamžikoch, navyše (z dôvodov odstránenia možných nepresností) nás nebudú zaujímať konkrétne veličiny, ale len či tieto sú v stanovených rozsahoch. Rozsahov býva obyčajne len ohraničený počet. Obvody, pracujúce so signálmi s konečným počtom rozsahov (úrovní) nazývame **číslicové (digitálne) obvody**.

Pre spracovanie binárnych údajov sú zaujímavé špeciálne číslicové obvody, ktoré rozlišujú len dve hodnoty parametrov signálu, ktoré môžeme interpretovať ako logické hodnoty **pravda** (true) a **nepravda** (false). Logickú úroveň pravda budeme označovať **log.1** a logickú úroveň nepravda budeme označovať **log.0** (prípadne len 1 a 0). Obvody, rozlišujúce len dve úrovne parametrov signálu nazývame **logické obvody** (niekedy tiež binárne obvody).

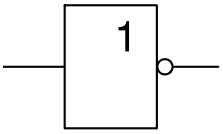
V predchádzajúcej kapitole sme ukázali, ako je možné využiť tranzistorové prvky na realizáciu jednoduchých logických funkcií NOT, NAND a NOR. Pri popise sme používali výrazy nízke napätie, vysoké napätie, kritická hodnota. Tie odpovedajú digitálnemu (binárnemu) pohľadu na parametre signálu.



Pre potreby binárnej logiky nám bude stačiť považovať nízke napätia za logickú 0 a napätia blízke hodnote napájacieho napätia V_{CC} za logickú 1. Prechody od nízkych napätí k vysokým sú ale spojité, signál sa v čase prechodu dostáva do nedefinovaného stavu. Snahou je, aby čas zotrvania v nedefinovanom stave bol čo najkratší. Úroveň signálu môžeme vyhodnocovať len v okamžikoch, keď nie je v nedefinovanom stave.

Jednoduché logické obvody NOT, NAND, NOR, uvedené v úvode kapitoly, sa nazývajú **logické členy** (niekedy tiež aj hradlá (angl. gates)). Na označenie logických členov v schémach zapojenia používame schematické značky. V publikovaných schémach sa môžeme stretnúť s dvomi typmi značiek. Tradičný zápis pomocou zaoblených symbolov bol súčasťou amerických noriem ANSI v minulosti, nájdeme ho teda v mnohých starších materiáloch. V súčasnosti sa podľa medzinárodnej normy ISO/IEC (International Electrotechnical Commission) používajú obdĺžnikové symboly s označením typu logickej operácie. Budeme používať štandardizovaný zápis a pre úplnosť uvedieme na okraji aj americké ekvivalenty.

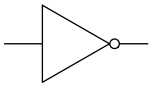
V úvode kapitoly sme ukazovali realizáciu obvodu **NOT** (invertora).



Schematická značka pre invertor (NOT)

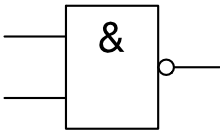
A	¬A
0	1
1	0

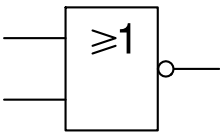
Tabuľka pravdivostných hodnôt



Tradičná schematická značka pre invertor

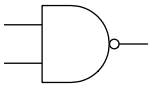
Tiež sme ukazovali, ako je možné realizovať zapojenia, ktoré sa správajú podľa logických funkcií **NAND** a **NOR**. Tu sú ich schematické značky a tabuľky pravdivostných hodnôt :



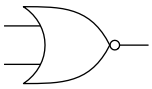


A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0



Tradičná schematická značka pre obvod NAND

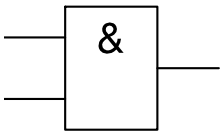


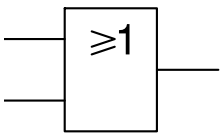
Tradičná schematická značka pre obvod NOR

Nápisy na značkách pripomínajú logickú funkciu, ktorú obvod realizuje. Vstupy sú kreslené spravidla zľava, výstup vpravo označený krúžkom hovorí, že funkcia, uvedená na značke sa na výstupe ešte invertuje (neguje). Krúžok sa niekedy v schémach kreslí aj pred vstupmi, vtedy to analogický znamená, že signál sa invertuje ešte pred vstupom do logického obvodu.

Všimnime si, že k navrhnutým zapojeniam tranzistorov je možné jednoducho pripájať ďalšie vstupy. Rozšírené viacvstupové obvody sa budú správať tak, že v prípade NAND je výstup log.0 práve vtedy, keď je na všetkých vstupoch úroveň log.1. Pre obvod NOR je naopak výstup log.1 práve vtedy, keď všetky vstupy majú úroveň log.0 (teda výstup je log.0 (pred invertovaním log.1) práve keď aspoň jeden zo vstupov má hodnotu log.1, čo naznačuje symbol ≥ na značke).

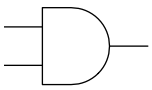
Medzi logické členy patria tiež obvody, realizujúce logické funkcie **AND** a **OR**:





A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1



Tradičná značka pre logický člen AND

Tradičná značka pre logický člen OR

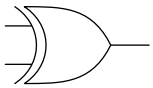
Aj členy AND a OR bývajú s viacerými vstupmi. Výstup členu AND je log.1 práve vtedy, ak sú všetky vstupy log.1, výstup členu OR je log.1 práve vtedy, ak je na vstupe aspoň jedna log.1.

Úloha 1

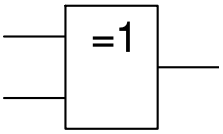
Navrhnite schému, ktorá pomocou viacerých dvojvstupových logických členov AND realizuje osemvstupový obvod AND.

Logický člen XOR

Logický člen **XOR** (XOR gate, skratka z eXclusive OR) je jednoduchý logický obvod, realizujúci logickú funkciu negácie ekvivalencie (resp. funkciu alebo s vylúčením súčasnej pravdivosti obidvoch vstupov). Výstup je log.1 práve vtedy, ak na vstupe je práve jedna hodnota úrovne log.1 (symbolizuje to na značke hodnota =1). V prípade dvoch vstupov je teda výstup 1 vtedy, keď sú vstupy rôzne. Výstup sa dá interpretovať tiež ako výsledok sčítania modulo 2. Využíva sa pri realizácii aritmetických výpočtov.



Tradičná schematická značka pre XOR



A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

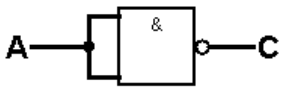
Úloha 2

Navrhnite obvod, ktorého výstup je log.1 práve vtedy, ak je vstup log.1 na nepárnom počte vstupov. Takýto obvod sa dá využiť ako generátor parity. Parita sa v počítačovej technike využíva ako spôsob kontroly integrity prenášaných údajov.

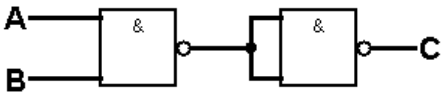
Realizácia všetkých obvodov logickými členmi jedného typu

Použitím viacerých logických členov NAND môžeme vytvoriť logické obvody, ktoré realizujú ktorúkoľvek zo šestnástich možných dvojvstupových logických funkcií. Často je preto člen NAND označovaný ako **univerzálny logický člen**.

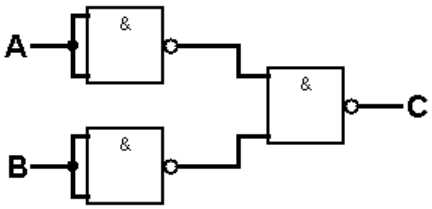
Prepojením viacerých členov NOR je tiež možné vytvoriť logický obvod, ktorý počíta ktorúkoľvek zo šestnástich možných dvojvstupových logických funkcií. NOR je tiež univerzálnym logickým členom. Dá sa ukázať, že už žiadne iné členy nie sú univerzálne.



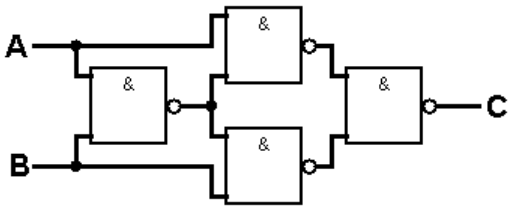
Zapojenie pre logickú funkciu NOT



Obvod, realizujúci funkciu AND



Zapojenie pre logickú funkciu OR



Zapojenie pre funkciu XOR

Pri zámene členov OR a AND môžeme využívať De Morganove pravidlá (negácia disjunkcie je konjunkciou negácií jej členov resp. negácia konjunkcie je disjunkciou negácií jej členov).

Úloha 3

Navrhnite obvody zložené výhradne z logických členov typu NOR, ktoré budú realizovať logické funkcie NOT, AND, OR a XOR.

2.3 Simulačný program Logisim

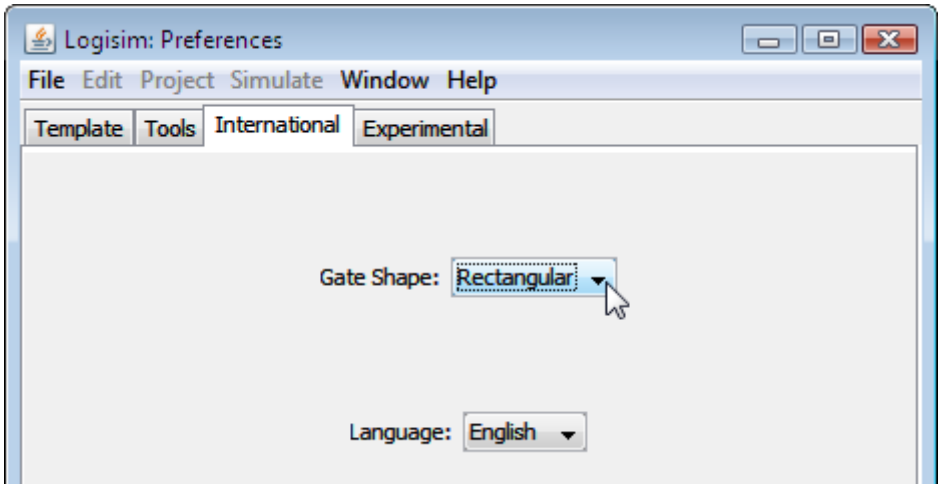
V súčasnosti sa už kreslené schémy používajú len výnimočne. Na zápis zložitejších obvodov sa využívajú špeciálne popisovacie jazyky ako VHDL (very-high-speed-integrated-circuit hardware description language) alebo Verilog. Zápis v štandardizovanom jazyku umožňuje prenos schém medzi rôznymi vývojovými prostrediami a medzi rôznymi nástrojmi na simuláciu a verifikáciu schém.

Pre naše potreby sú tieto nástroje príliš komplikované. Pre simuláciu činnosti a zložitejšie zapojenia logických obvodov budeme používať jednoduchý program Carla Burcha Logisim (môžete ho nájsť v e-learningovej časti systému Moodle spolu s dopredu pripravenými obvodmi podľa tejto učebnice, alebo ho môžete bezplatne získať na adrese http://www.cburch.com/logisim/). Praktické skúšanie obvodov v simulačnom programe určite pomôže k lepšiemu pochopeniu ich fungovania.

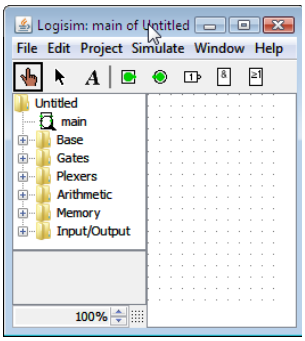
Možnosti programu ďaleko presahujú požiadavky kladené týmto modulom a preto sa zameriame v jeho popise len na časti, ktoré bude potrebné priamo využiť.

Program Logisim je voľne dostupný pod slobodnou licenciou GNU GPL verzie 2. Naprogramovaný je v jazyku Java, čo znamená, že funguje vo všetkých operačných systémoch, ktoré podporujú SUN JRE (teda GNU/Linux, Microsoft Windows, Mac OS X a iné). Z jazykov podporuje momentálne (verzia 2.3.2) žiaľ len angličtinu a španielčinu, takže návod bude v prvom spomínanom.

Po zapnutí program používa oblé, tradičné značky. V prvom kroku je vhodné (pre kompatibilitu s týmto materiálom) nastaviť zobrazovanie značiek podľa štandardu IEC. Z hlavnej ponuky vyberte záložku *File* a v nej voľbu *Preferences*. Zobrazí sa okno s nastaveniami programu. Zvoľte záložku *International* a v nej nastavte *Gate Shape* na *Rectangular*.

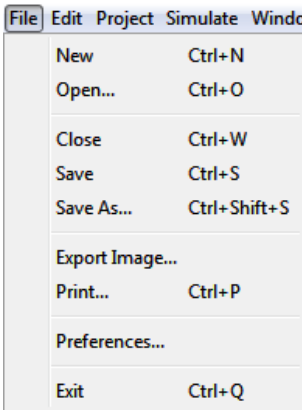


Nastavenia programu



Hlavné okno programu Logisim

V ponuke *File* nájdete aj ďalšie nástroje, ktoré poznáte z iných programov. Funkcie *New, Open, Close, Save* a *Save As* slúžia na prácu s aktuálnym projektom. Funkcia *Export Image* Vám umožní uložiť projekt vo forme obrázku. Funkcia *Print* ho vytlačí. Funkcia *Exit* ukončí program.



Ponuka File

Voľba *Open* umožní otvoriť uložený výsledok predchádzajúcej práce alebo pripojí k štandardným funkciám ďalšie, umiestnené v externých knižniciach. Voľbou *Save* resp. *Save As* uložíte aktuálny stav projektu, prípadne ho môžete aj premenovať.

Hneď po zapnutí programu alebo po kliknutí na voľbu *New* máte pripravenú prázdnu pracovnú plochu na návrh obvodu s názvnom *main*. Je vhodné tento názov zmeniť, aby sa zabezpečila prehľadnosť pri používaní viacerých obvodov.

Najjednoduchší spôsob zmeny názvu obvodu je kliknutie pravým tlačidlom myši na názov obvodu a voľba funkcie *Rename Circuit*.

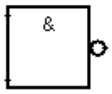
Označenie logických členov v programe Logisim

Niektoré logické členy majú v programe Logisim iné značky, ako sú uvedené v tomto texte.

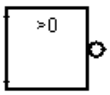
Logický člen NOT

Logický člen AND

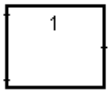
Logický člen OR



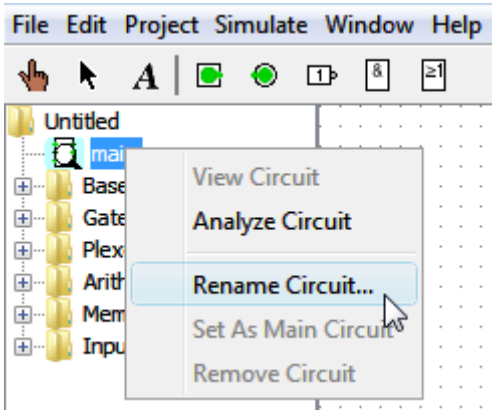
Logický člen NAND



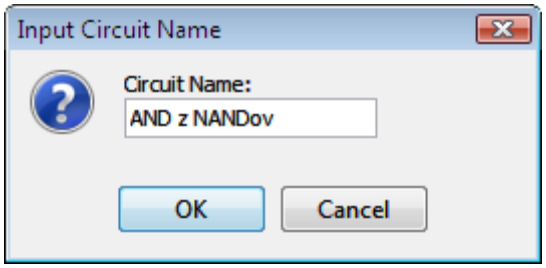
Logický člen NOR



Logický člen XOR

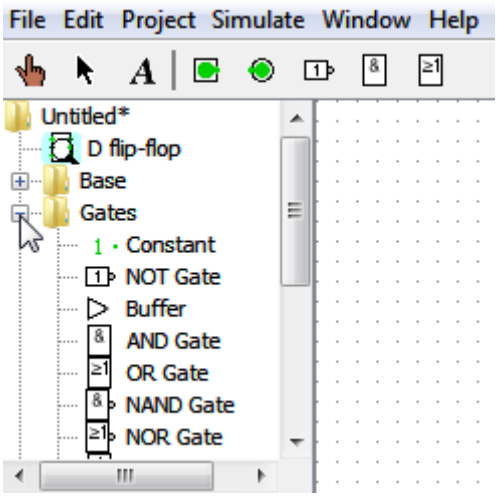


Zmena názvu obvodu



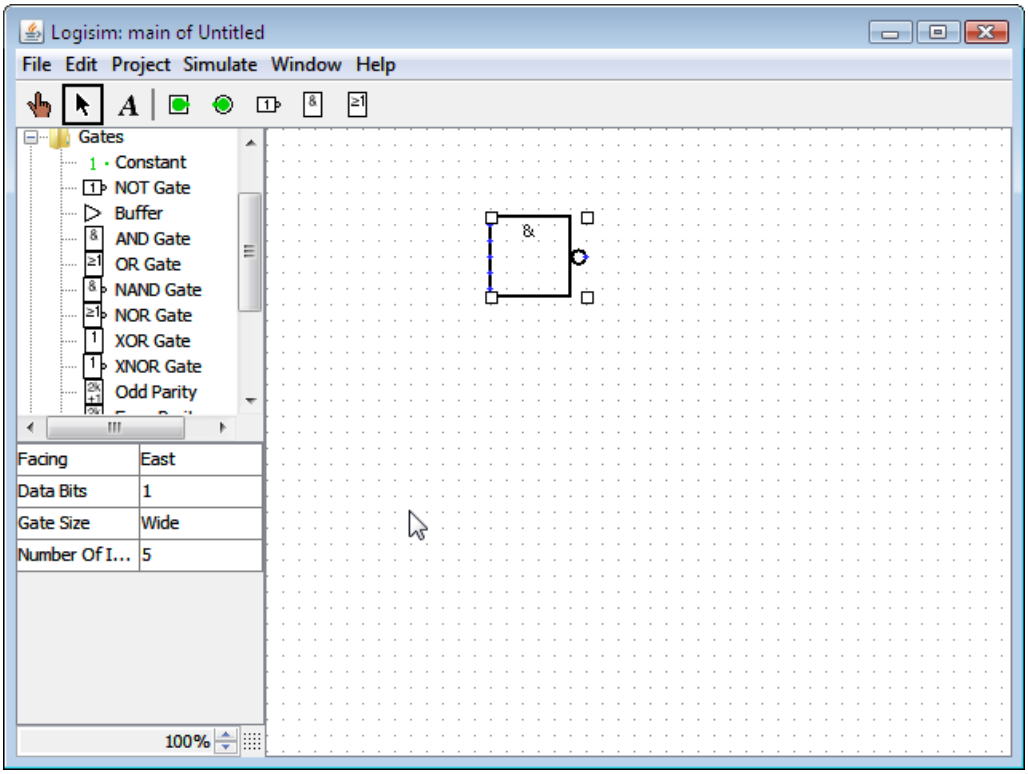
Dialógové okno s novým názvom

V príklade budeme navrhovať logický obvod AND zložený z logických členov NAND, preto zvolíme príslušný názov. Môžeme pokračovať výberom logických členov.



Zoznam dostupných logických členov

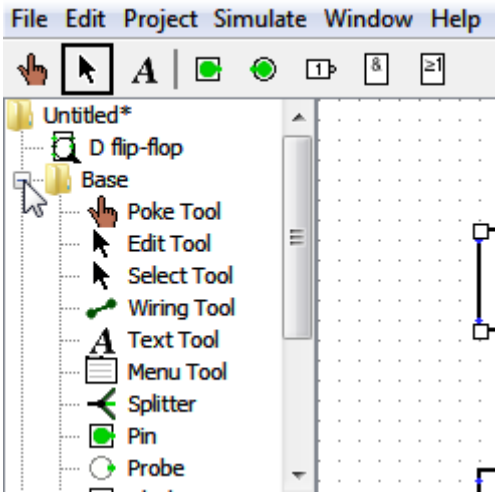
Zoznam dostupných logických členov získame po rozbalení položky *Gates*. Nie všetky z dostupných logických členov sú spomínané v tomto module. Na AND potrebujeme dva logické členy NAND, takže v ponuke zvolíme NAND. Kurzor nad pracovnou plochou sa následne zmení na sivý obrys zvoleného logického členu a kliknutím ho umiestnime na požadovanú pozíciu.



Logický člen NAND umiestnený na pracovnej ploche

V ľavom stĺpci sa zobrazila ponuka s možnosťami nastavení vybraného prvku, v našom prípade logického členu NAND. Môžeme si zvoliť orientáciu obvodu (*Facing*), počet bitov, ktoré paralelne spracováva (*Data Bits*), veľkosť schematickej značky (*Gate Size*) a počet vstupov (*Number Of Inputs*). Pre iné súčiastky je možné meniť iné nastavenia.

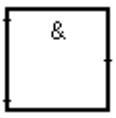
Po umiestnení súčiastok na pracovnú plochu je potrebné vytvoriť medzi nimi prepojenia. Nástroj na túto činnosť nájdete v ponuke *Base*.



Zoznam nástrojov

Na prepojenie súčiastok slúži nástroj *Wire Tool* (angl. wire – spoj, drôť). Medzi ďalšie dôležité nástroje patria *Poke tool* (angl. poke - šťuchnúť), ktorý slúži na prepínanie stavov na vstupe, *Edit tool* (angl. edit - upraviť), ktorým je možné upravovať existujúce spoje a súčiastky, *Select tool* (angl. select - výber), ktorý slúži na označovanie súčiastok a *Pin* (angl. pin – kolík, vývod, pin) slúžiaci ako vstup a výstup signálu.

Orientácia obvodov (*Facing*)



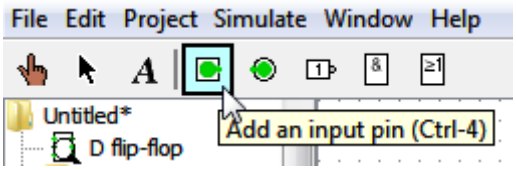
East - východ

West - západ

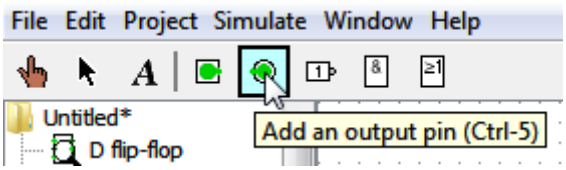
North - sever

South - juh

Samotný pin, rozlíšený už na vstupnú a výstupnú verziu, nájdete aj na nástrojovej lište.

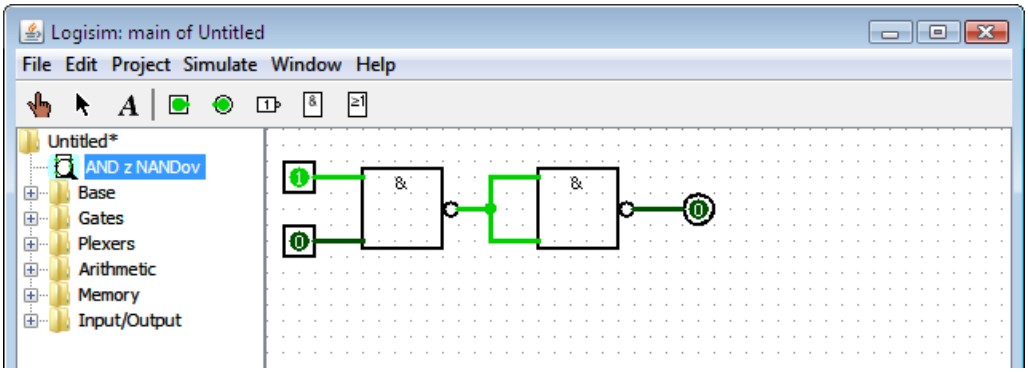


Vstupný pin



Výstupný pin

Pomocou týchto nástrojov a dostupných súčiastok je možné zostrojiť všetky obvody, ktoré sú spomínané v tomto module.



Hotový obvod AND zložený z dvoch členov NAND

Okrem návrhu obvodov umožňuje program Logisim aj ich simuláciu. Tá je zapnutá hneď po štarte a prebieha aj počas samotného navrhovania obvodu. Logické úrovne na vstupných pinoch môžete meniť pomocou *Poke tool* a výsledok sa okamžite prejaví na výstupnom pine. Zelená farba spoja označuje logickú úroveň log.1, čierna farba úroveň log.0.

Podrobná príručka v anglickom jazyku je dostupná po kliknutí na *Help* a následne *User's Guide*.

Úloha 1	V programe Logisim odsimulujte funkcie logických členov NOT, AND, OR, NAND, NOR, XOR.
Úloha 2	Prepíšte do programu Logisim výsledky predchádzajúcich úloh a overte ich správnosť – zapojenia osemvstupových obvodov AND a OR pomocou dvojvstupových a generátora parity.

2.4 Ďalšie kombinačné logické obvody

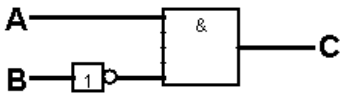
Kombinačné logické obvody sú logické obvody, ktorých výstup závisí výhradne od logických hodnôt (signálov) privedených na vstup. Po určitom čase (v závislosti od počtu logických členov, ktoré sú na najdlhšej ceste medzi vstupom a výstupom) sa výstupný signál ustáli na úrovni, odpovedajúcej logickej hodnote realizovanej logickej funkcie.

Dá sa ukázať, že logické členy každého kombinačného obvodu je možné očíslovať prirodzenými číslami tak, že každé prepojenie smeruje od výstupu člena s nižším číslom k vstupu člena s vyšším číslom (v zapojení neexistujú cyklické prepojenia).

Spomenieme ďalej niektoré obvody, ktoré budeme potrebovať v nasledujúcich moduloch.

Dekóder

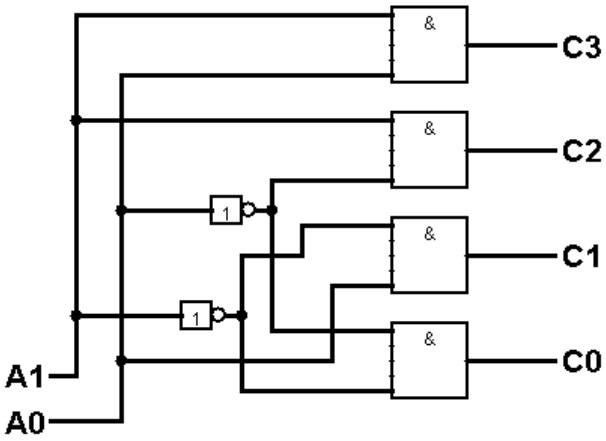
Dekodér je kombinačný logický obvod, ktorý zisťuje prítomnosť jedného alebo viacerých špecifických vstupov a jeho prítomnosť signalizuje na výstupe log.1. Pre akýkoľvek iný vstup je stav na výstupe log.0.



Dekóder detekujúci vstup A = log.1, B = log.0

Dekóder na obrázku má na výstupe C stav log.1 len v prípade, ak je na vstup A privedený stav log.1 a na vstup B stav log.0. Vo všetkých iných prípadoch je na výstupe C stav log.0.

Dekóder však môže reagovať aj na viacero vstupných kombinácií.



Dekóder binárneho kódu na kód jedna zo štyroch

Dekóder na obrázku reaguje na ľubovoľný dvojbitový binárny kód na vstupe stavom log.1 na výstupe s číslom, odpovedajúcim tomuto kódu. Všetky ostatné výstupy budú na úrovni log.0. Typ dekódera, ktorý ma stav log.1 vždy práve na jednom výstupe, označujeme 1 z N, v tomto konkrétnom prípade 1 zo 4.

Koľko vstupov bude mať dekóder 1 z 8 ?

Tabuľka pravdivostných hodnôt logických funkcií dekódera dvojbitového binárneho kódu na kód jedna zo štyroch:

A0	A1	¬A0	¬A1	C0 ¬A0 AND ¬A1	C1 ¬A0 AND A1	C2 A0 AND ¬A1	C3 A0 AND A1
0	0	1	1	**1**	0	0	0
0	1	1	0	0	**1**	0	0
1	0	0	1	0	0	**1**	0
1	1	0	0	0	0	0	**1**

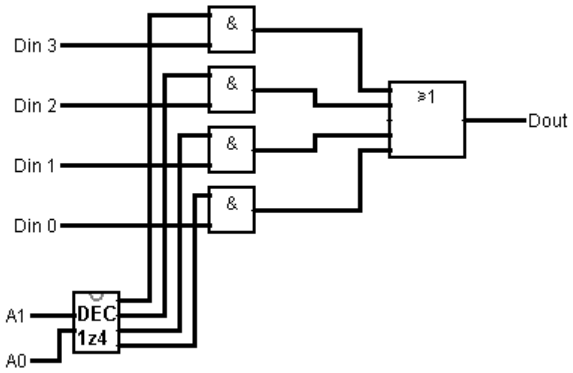
Takýto dekóder sa nachádza v moderných počítačoch často a svoje využitie nachádza napríklad pri adresovaní pamäťových buniek.

Úloha 1	Opačnou úlohou k dekódovaniu je kódovanie. Navrhnite a v prostredí Logisim odskúšajte kóder so štyrmi vstupmi a tromi výstupmi. Výstup V (valid) signalizuje, že práve jeden zo vstupov je v úrovni log.1, ostatné dva výstupy signalizujú v binárnom trvare číslo vstupu, ktorého úroveň je log.1. Pokiaľ na vstupe nie je práve jedna úroveň log.1, je na výstupe V úroveň log.0 a na ostatných výstupoch nezáleží.
Úloha 2	Niekedy sa využíva jednoduchší variant kódera – tzv. prioritný kóder. Výstup V bude log.0 práve v prípade, že na vstupe sú samé log.0. Inokedy bude výstup V log.1 a ostatné dva výstupy budú v binárnom tvare signalizovať najvyššie číslo zo vstupov, ktoré sú v úrovni log.1. Zostavte taký kóder v prostredí Logisim. Prioritný kóder sa napr. používa na výber zariadenia, ktoré treba akceptovať s najvyššou prioritou.

Multiplexor a demultiplexor

Multiplexor je kombinačný logický obvod, ktorý vyberá jeden (alebo jednu skupinu) z viacerých vstupných signálov a prepúšťa ho na výstup. **Demultiplexor** je kombinačný logický obvod, ktorý prepúšťa signál (alebo skupinu signálov) z jediného vstupu na jeden z viacerých výstupov.

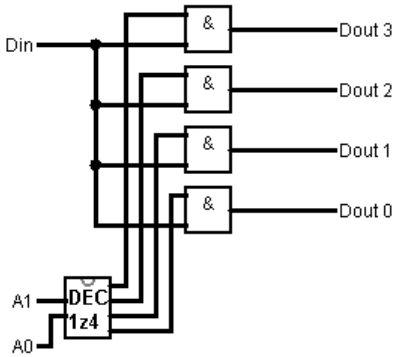
Pomocou multiplexora vieme tiež zrealizovať každú logickú funkciu dvoch vstupov. Stačí nastaviť na vstupoch Din výsledný stĺpec tabuľky pravdivostných hodnôt (v úrovniach log.0 resp. log.1).



4 do 1 jednobitový digitálny multiplexor

Logickú funkciu je možné realizovať rôznymi spôsobmi. Pre efektívnu implementáciu je výhodné nájsť riešenie s čo najmenším počtom logických členov (čo v konečnom dôsledku znamená menší počet tranzistorov a jednoduchšiu a lacnejšiu výrobu).

Táto úloha však nie je ľahká. Optimalizácia návrhu logických obvodov tvorí dynamicky sa rozvíjajúcu a aplikačne zaujímavú časť informatiky.



1 do 4 jednobitový digitálny demultiplexor

Výber vstupného signálu v multiplexore (alebo výstupného v demultiplexore) sa uskutočňuje na základe riadiaceho signálu, privedeného na vstup binárneho dekódera 1 z N. Z jeho N výstupov je vždy práve jeden log.1 a ostatné log.0.

Tabuľka popisujúca výstup 4 do 1 digitálneho multiplexora:

A0	A1	Dout
0	0	**Din0**
0	1	**Din1**
1	0	**Din2**
1	1	**Din3**

Tabuľka popisujúca výstup 1 do 4 digitálneho demultiplexora:

A0	A1	Dout0	Dout1	Dout2	Dout3
0	0	**Din**	0	0	0
0	1	0	**Din**	0	0
1	0	0	0	**Din**	0
1	1	0	0	0	**Din**

Čo sme sa naučili

Kombinačné logické obvody realizujú ľubovoľne zložité logické funkcie pomocou jednoduchých logických členov. Spoznali sme činnosť kódera/dekódera a multiplexora/demultiplexora.

Logické funkcie môžeme realizovať viacerými spôsobmi. Problémom môže byť nájdenie realizácie s najmenším počtom logických členov určeného typu, prípadne s najmenším množstvom prepájacích vodičov.

Naučili sme sa navrhovať a testovať jednoduché obvody v prostredí Logisim.

2.5 Sekvenčné logické obvody

Pokiaľ umožníme v zapojení logického obvodu tzv. spätné väzby (pripojíme výstup logického člena na vstup niektorého z predchádzajúcich členov – obvod sa už nebude dať nakresliť tak, aby signály od výstupov jedných obvodov k vstupom iných smerovali stále vpravo), dostaneme zapojenia, ktoré nemožno popísať pomocou jednoduchých logických funkcií.

Ich výstupné úrovne signálov sa menia v závislosti na aktuálnych, ale tiež aj minulých stavoch vstupných signálov (minulé stavy sa prejavia práve prostredníctvom spätnej väzby z minulých výstupov na aktuálne vstupy obvodu). Obvody nazývame tiež **sekvenčné obvody**, pretože výstupné signály závisia na sekvenciách – postupnostiach – vstupných signálov

Pretože výstupy závisia od stavov, ktoré už pominuli, majú sekvenčné obvody teda určité pamäťové vlastnosti, ktoré môžeme šikovne využiť. Obvody sa môžu dostať do nestabilných stavov (pozri poznámku na okraji), alebo do viacerých stabilných stavov. Pre potreby číslicovej techniky sú zaujímavé obvody s dvomi stabilnými stavmi, ktoré tiež nazývame **bistabilné preklápacie obvody**.

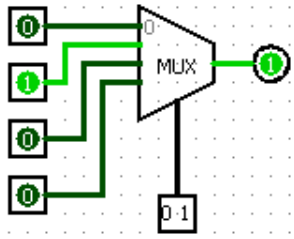
Preklápacie obvody

Preklápací obvod je sekvenčný logický obvod, ktorý sa dokáže na základe vstupu a vnútorného stavu (spätnej väzby) prepínať medzi niekoľkými stabilnými alebo nestabilnými stavmi. Podľa spôsobu reakcie na vstupné signály ich delíme na

- Asynchrónne (angl. Latch alebo Transparent latch) - záchytné obvody - preklopia sa ihneď po zmene úrovne na niektorom vstupe
- Synchrónne (angl. Gated latch, Clocked latch, alebo Flip-flop podľa typu synchronizácie) - preklopia sa len v súčinnosti so synchronizačným

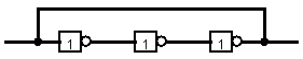
Multiplexor v programe Logisim

Program Logisim ponúka už hotový multiplexor. Na jeho vstupe môže byť od dvoch až po šestnásť kanálov, podľa toho, koľko bitov ho ovláda. Príklad zapojenia:



Multiplexor 4 do 1 ovládaný dvoma bitmi

Program Logisim umožňuje nastaviť pre jedno prepojenie jeho „šírku“, teda počet bitov, ktoré dokáže preniesť. Na obrázku je ovládací signál privedený zdola a má šírku dva bity.



Skúmajme jednoduché zapojenie so spätnou väzbou na obrázku. Prenesenie signálu zo vstupu na výstup trvá invertoru určitý čas. Výstup obvodu bude teda neustále prechádzať z log.0 na log.1 a naopak (pokiaľ by sme vedeli presne nastaviť konštrukčné parametre, mohli vy sme ho použiť ako generátor pravidelných časových impulzov) – obvod sa zvykne nazývať aj astabilný multivibrátor.

(hodinovým, taktovacím) vstupom, ktorý (úrovňou alebo zmenou úrovne) povoľuje reakciu obvodu na vstupy. Rozoznávame tieto typy synchronizácie:

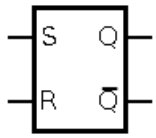
- synchronizácia úrovňou hodinového signálu (úrovňová alebo hladinová synchronizácia) (angl. Gated latch alebo Clocked latch)
- synchronizácia nábežnou hranou hodinového signálu (derivačná synchronizácia) (angl. Positive edge triggered flip-flop)
- synchronizácia zostupnou hranou hodinového signálu (derivačná synchronizácia) (angl. Negative edge triggered flip-flop)

Keďže rýchlosť šírenia sa signálu je ohraničená, preklápaciemu obvodu môže trvať určitý časový interval, kým sa dostane do výsledného, požadovaného stavu. Tento časový interval sa nazýva **hazard** a počas neho sa obvod prepína medzi definovanými logickými stavmi, alebo sa môže nachádzať aj v nedefinovaných medzistavoch.

Činnosť sekvenčných obvodov so stabilnými stavmi popisujeme **prechodovou tabuľkou**, ktorá je rozšírením tabuľky pravdivostných hodnôt o údaje o týchto stavoch. Na vstupnej strane obsahuje okrem všetkých kombinácií hodnôt vstupov aj všetky možnosti stavov pred preklopením, výstupná strana obsahuje výstupné hodnoty a stavy po preklopení. Charakteristiku prechodov medzi stavmi je tiež možné ilustrovať **časovým diagramom**, v ktorom môžeme zachytiť reakcie výstupov obvodu na rôzne postupnosti kombinácií jeho vstupov.

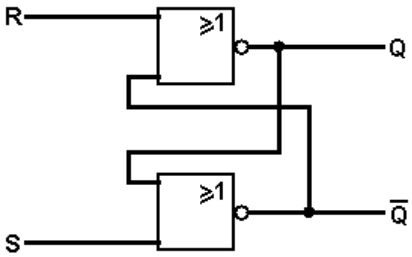
Preklápací obvod RS

Preklápací obvod RS (angl. SR latch) je najjednoduchší asynchrónny bistabilný preklápací obvod. Na vstupe má signály R (z angl. Reset – nulovanie) a S (z angl. Set – nastavenie). Uložená hodnota je k dispozícii na výstupe Q. Obvykle je k dispozícii tiež negovaný výstup ¬Q, označovaný aj ako Q'.



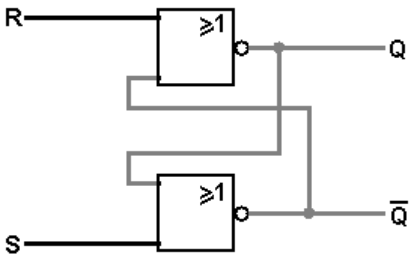
Schematická značka preklápacieho obvodu RS

Obvod je možné zostrojiť pomocou dvoch logických členov NOR.



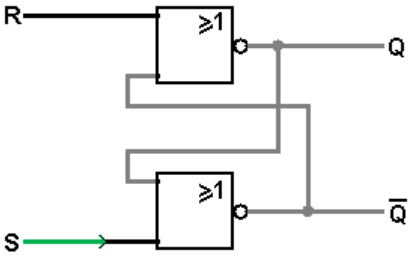
Preklápací obvod RS

Základný stav oboch vstupov je log.0. V tomto režime si obvod pamätá naposledy nastavenú hodnotu. Na obrázku je tento stav označený sivou farbou.

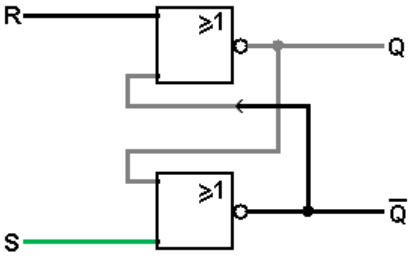


Vstupy R a S v stave log.0

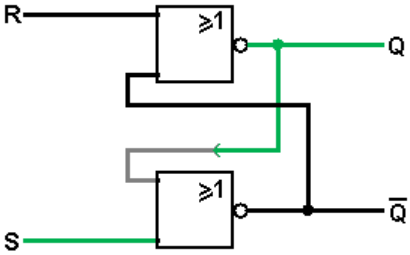
Po privedení stavu log.1 na vstup S sa výstup spodného logického člena NOR prepne do stavu log.0 a signál sa začne šíriť k hornému logickému členu. Keďže obidva vstupy horného NOR sú v stave log.0, výstup sa prepne do stavu log.1 a začne sa šíriť k spodnému NOR.



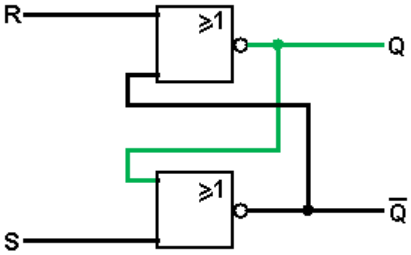
Stav log.1 privedený na vstup S



Log.0 sa šíri z výstupu spodného NOR



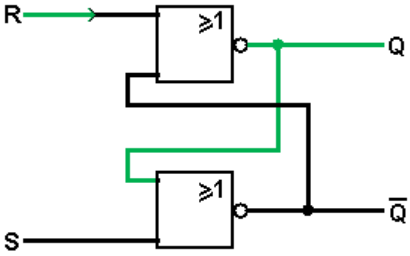
Šírenie log.1 z výstupu horného NOR



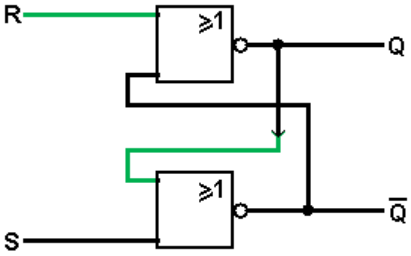
Obvod ostáva v stabilnom stave aj po privedení log.0 na vstup S

Obvod sa stabilizuje v stave log.1 na výstupe Q. V takomto stave ostane aj po privedení log.0 na vstup S, keďže jeden zo vstupov spodného logického člena NOR je v stave log.1 a teda výstup ostane v stave log.0 aj naďalej.

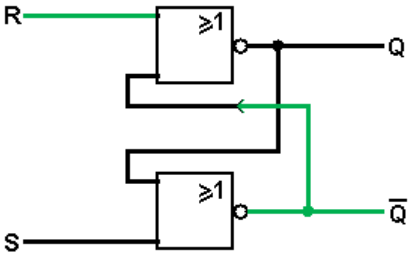
Po privedení stavu log.1 na vstup R sa výstup horného logického člena NOR prepne do stavu log.0 a signál sa začne šíriť k spodnému NOR. Keďže obidva vstupy spodného logického člena NOR sú v stave log.0, výstup sa prepne do stavu log.1 a začne sa šíriť k hornému logického členu.



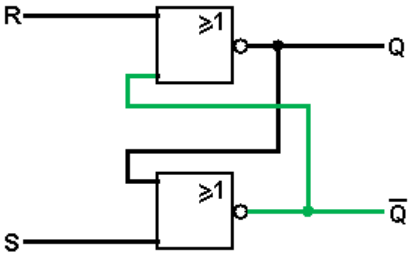
Stav log.1 privedený na vstup R



Šírenie log.0 z výstupu horného NOR

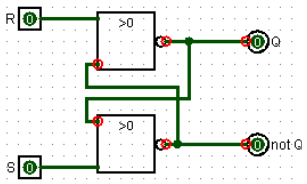


Šírenie log.1 z výstupu spodného NOR



Obvod ostáva v stabilnom stave aj po privedení log.0 na vstup R

Oscilácia v Logisime

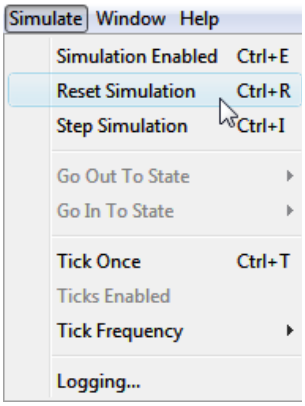


Obvod RS s osciláciou v programe Logisim

Pri simulácii sa občas môže stať, že v obvode nastane oscilácia. To znamená, že jedna alebo viac súčiastok neustále, bez zmeny vonkajších podmienok, cyklicky prechádza medzi dvoma alebo viacerými stavmi. Často ide o neželaný stav.

Nápis upozorňujúci na osciláciu

V takomto prípade je potrebné simuláciu prerušiť, odstrániť zdroj oscilácie alebo mu zabrániť vhodným nastavením počiatočných signálov (pomocou nástroja *Poke Tool*) a simuláciu opäť spustiť.



Prerušenie simulácie

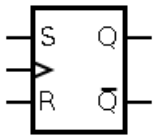
Obvod sa stabilizuje v stave log.0 na výstupe Q. V takomto stave ostane aj po privedení log.0 na vstup R, keďže jeden zo vstupov horného logického člena NOR je v stave log.1 a teda výstup ostane v stave log.0 aj naďalej.

Prípad, kedy R a S sú oba log.1 je zakázaný (alebo tiež nestabilný) stav, pretože pri ňom nie je definované v akom stave zostane obvod po návrate R a S na log.0.

Zjednodušená tabuľka prechodov medzi stavmi obvodu RS:

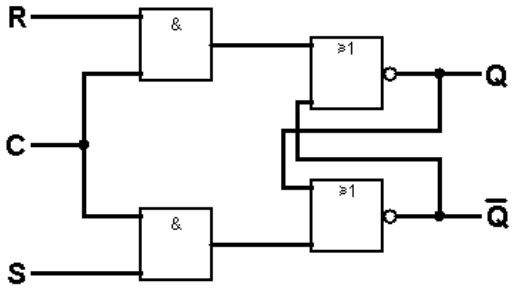
R	S	Q_{n+1}	Popis
0	0	Q_n	Zachovanie stavu
0	1	1	Nastavenie
1	0	0	Nulovanie
1	1	?	Zakázaný stav

Preklápací obvod RST



Schematická značka preklápacieho obvodu RST

Preklápací obvod RST (angl. Gated SR latch) je synchrónny variant obvodu RS. Princíp zostáva zachovaný, avšak k nastaveniu alebo nulovaniu výstupu dochádza len v konkrétnych prípadoch, v závislosti od hodnoty signálu na hodinovom vstupe C (z angl. Clock – hodiny , občas označovaný T, prípadne v schematickom zápise trojuholníkom).

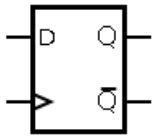


Preklápací obvod RST

Tabuľka prechodov stavov pre obvod RST:

R	S	C	Q_{n+1}	Popis
0/1	0/1	0	Q_n	Zachovanie stavu
0	0	1	Q_n	Zachovanie stavu
0	1	1	1	Nastavenie
1	0	1	0	Nulovanie
1	1	1	?	Zakázaný stav

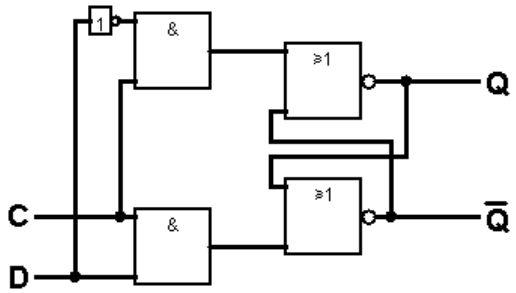
Preklápací obvod D



Schematická značka preklápacieho obvodu D

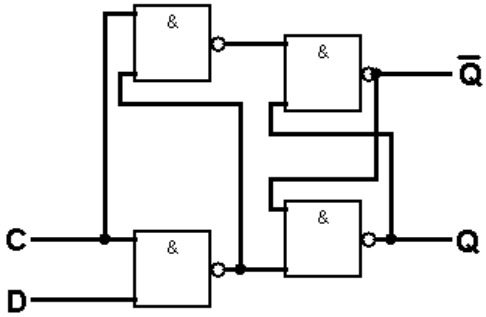
Preklápací obvod D (z angl. Delay – zdržanie) je synchrónny bistabilný preklápací obvod so vstupom D (z angl. Data - údaj) a hodinovým vstupom C (z angl. Clock - hodiny). Pri nábežnej hrane hodinového signálu sa momentálna hodnota vstupu D skopíruje do vnútorného stavu a na výstup, kde zostane zachovaná až do nasledujúcej nábežnej hrany hodinového signálu. Obvod je teda schopný vo svojom vnútornom stave určitú dobu uchovať zaznamenanú úroveň vstupného signálu.

Jednoduchý preklápací obvod D je možné zostaviť z obvodu RST tak, že na vstup S privedieme priamo hodnotu vstupu D a na vstup R jeho negovanú hodnotu. Obvod sa potom pri log.1 na vstupe D nastaví a naopak pri log.0 vynuluje.

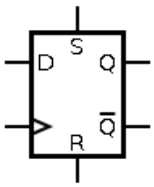


Preklápací obvod D vytvorený z obvodu RST

V praxi sa ale preklápací obvod D realizuje pomocou štyroch členov typu NAND.



Preklápací obvod D vytvorený z logických členov NAND



Schematická značka preklápacieho obvodu D so vstupmi R a S

Tabuľka prechodov stavov obvodu D:

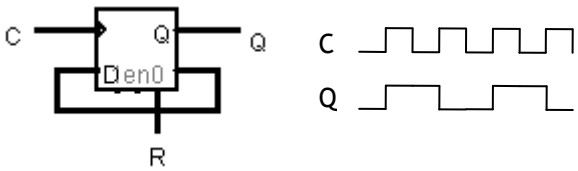
D	C	Q_{n+1}	Popis
0/1	0	Q_n	Zachovanie stavu
0	1	0	Nastavenie
1	1	1	Nastavenie

V programe Logisim sú vstupy R a S vyvedené na spodnej strane obvodu a označené 0 a 1.

Preklápacie obvody D mávajú tiež často okrem vstupu D vyvedené aj vstupy R a S, umožňujúce nastavenie a nulovanie obvodu v asynchrónnom režime.

Počítadlo

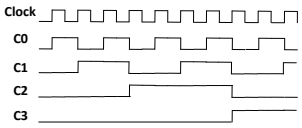
Uvažujme zapojenie, obvodu D tak, že prepojíme negovaný výstup Q so vstupom D. Po resetovaní bude výstup Q v stave log.0, na vstupe D bude log.1 z negovaného výstupu Q.



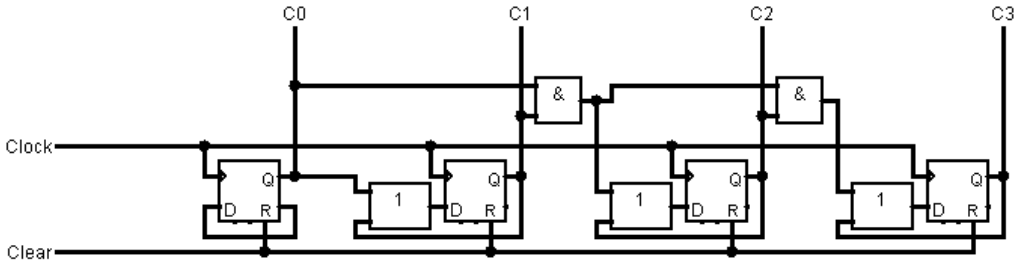
Časový diagram závislosti výstupu Q od vstupu C

Zmena signálu C z log.0 na log.1 spôsobí preklopenie obvodu D do stavu s výstupom Q na úrovni log.1. Tento stav ostane zachovaný, aj keď vstupný signál klesne na úroveň log.0. Na vstupe D už ale teraz máme log.0. Pri ďalšom nábehu signálu C z log.0 na log.1 dôjde opäť k preklopeniu obvodu (vstup D je 0) do stavu s výstupom Q v úrovni log.0. Čo zasa ostane zachované po ďalší nábeh signálu C.

Závislosť stavu výstupu Q od vstupu C môžeme vysledovať aj na časovom diagrame. Výsledok môžeme interpretovať aj ako paritu počtu nábehov signálu C od resetovania obvodu a využijeme ho pre zostavenie počítadla.



Časový diagram 4-bitového počítadla



Zapojenie na tomto obrázku realizuje 4-bitové synchrónne počítadlo. Podľa časového diagramu môžete vysledovať (a pomocou programu odsimulovať) činnosť tohto obvodu. Po resetovaní prechádzajú výstupy C0, C1, C2 a C3 pri každej nábehovej hrane signálu Clock do stavov, odpovedajúcich binárnemu zápisu čísel 0 – 15 (výstup C3 zodpovedá najvyššiemu rádu. Obvod môžeme použiť na počítanie vstupných pulzov, ale tiež ako delič kmitočtu (ak na vstupe sa budú striedať úrovne s frekvenciou f, dostaneme na výstupe C0 striedanie úrovní s frekvenciou f/2, na výstupe C1 s frekvenciou f/4, na C2 s frekvenciou f/8 a na C3 s frekvenciou f/16).

Uvedené zapojenie môže spôsobovať hazardy pri súčasných zmenách vstupov C a D. Vedeli by ste navrhnúť obvod, kde by tieto hazardy nastať nemohli ?

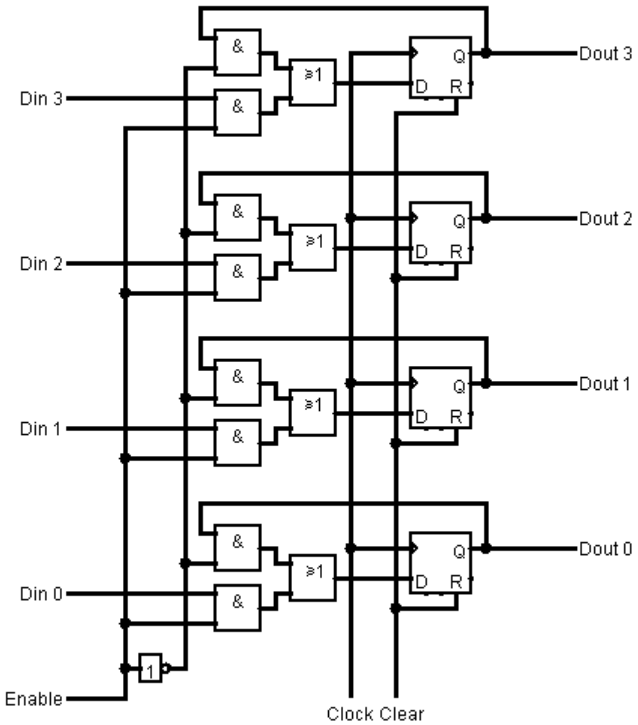
Úloha 1

Vysledujte v prostredí Logisim činnosť 4-bitového počítadla a zamyslite sa nad spôsobom prípravy vstupu D pre jednotlivé rády výpočtu. Čo sa stane, keď počet nábehov signálu Clock dosiahne 16 ?

Register

Register je sekvenčný logický obvod, ktorý slúži na zaznamenanie úrovní viacerých vstupných signálov naraz. Je zložený z viacerých preklápacích obvodov (väčšinou typu D), ktoré sú riadené spoločnými signálmi.

Register slúži procesoru ako najrýchlejšia pamäť. Je ich len niekoľko desiatok a zabezpečujú pohotový prístup k údajom, s ktorými sa aktuálne pracuje. Podľa veľkosti použitých registrov rozoznávame 8 bitové, 16 bitové, 32 bitové resp. 64 bitové procesory.



Štvorbitový register

Vstupný signál Clock (z angl. clock – hodiny) je privedený na vstup C a vstupný signál Clear (z angl. clear – čistý, čistiť) je privedený na asynchrónny vstup R všetkých preklápacích obvodov D. Vstupný signál Load (z angl. load – načítať) určuje, či sa na vstupy jednotlivé D privedie signál z ich vlastného výstupu Q alebo signál zo vstupov

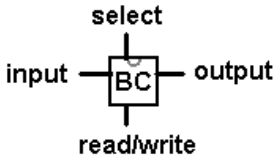
Din0 až Din3. Ak je na vstupe Load log.1, na vstupy D jednotlivých preklápacích obvodov sú privedené signály Din0 až Din3. Pri najbližšom hodinovom impulze na vstupe Clock sú tie prekopírované do vnútorného stavu preklápacích obvodov. Ak je signál na Load log.0, na vstup D je privedený signál z výstupu Q a aj pri hodinových impulzoch na Clock bude teda zachovaný.

Po privedení signálu log.1 na vstup Clear budú vnútorné stavy všetkých preklápacích obvodov okamžite zmenené na log.0.

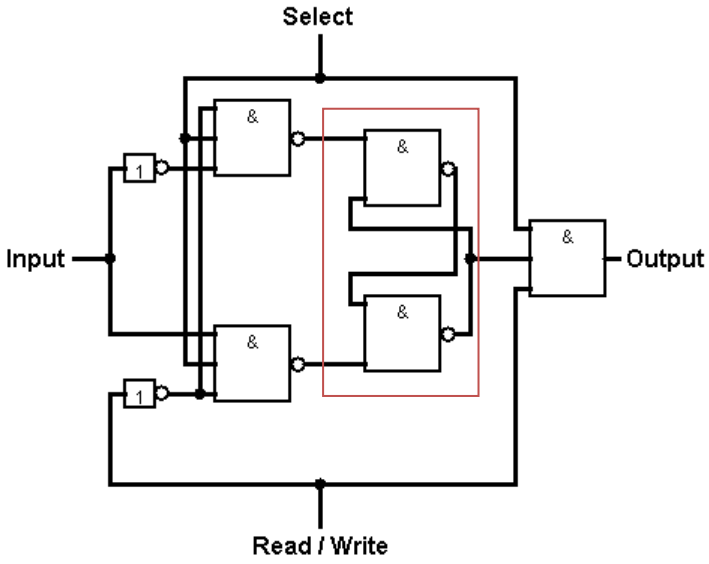
Pamäť

Základnou stavebnou jednotkou pamäte je pamäťová bunka.

Pamäťová bunka (BC, z angl. Binary Cell – binárna bunka) je sekvenčný logický obvod slúžiaci na uloženie jedného bitu informácie. Jej základ tvorí asynchrónny preklápací obvod RS (na obrázku zložený z dvoch NAND hradiel – v červenom obdĺžniku), ktorý je riadený troma signálmi – input (z angl. input - vstup), select (z angl. select - výber) a read/write (z angl. read/write – čítanie/zápis).



Pamäťová bunka



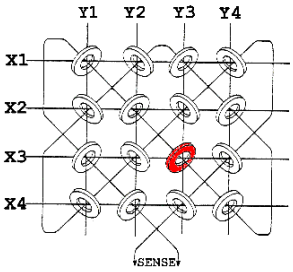
Pamäťová bunka

Signál Select slúži na vyberanie aktívnej bunky. Ak je signál na vstupe select log.0, bunka neprijíma žiadne iné signály na ostatných vstupoch a jej výstupný stav je log.0. Uvažujme teda ďalej, že vstup select je nastavený na log.1.

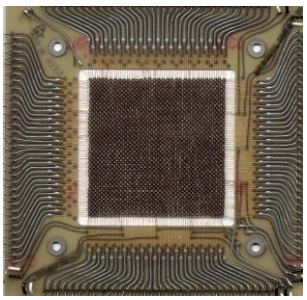
Signál read/write slúži na prepínanie medzi zápisom a čítanim z bunky. Ak je tento signál nastavený na log.0, vnútorný stav preklápacieho obvodu RS sa nastaví na hodnotu signálu na vstupe input. Na výstupe output je log.0. Ak je vstup read/write nastavený na log.1, vstup input je ignorovaný a na výstupe output je vnútorný stav preklápacieho obvodu RS.

Pamäťové bunky je možné vzájomne prepojiť a vytvoriť **pamäť**. Z praktického dôvodu sa používa zapojenie, kedy je možné čítať a zapisovať len skupinu pamäťových buniek a nie konkrétne bunky. Tiež čítanie a zapisovanie súčasne nie je povolené (dochádzalo by ku kolíziám v prístupe do buniek). V jednom okamihu sa môže len čítať, alebo len zapisovať.

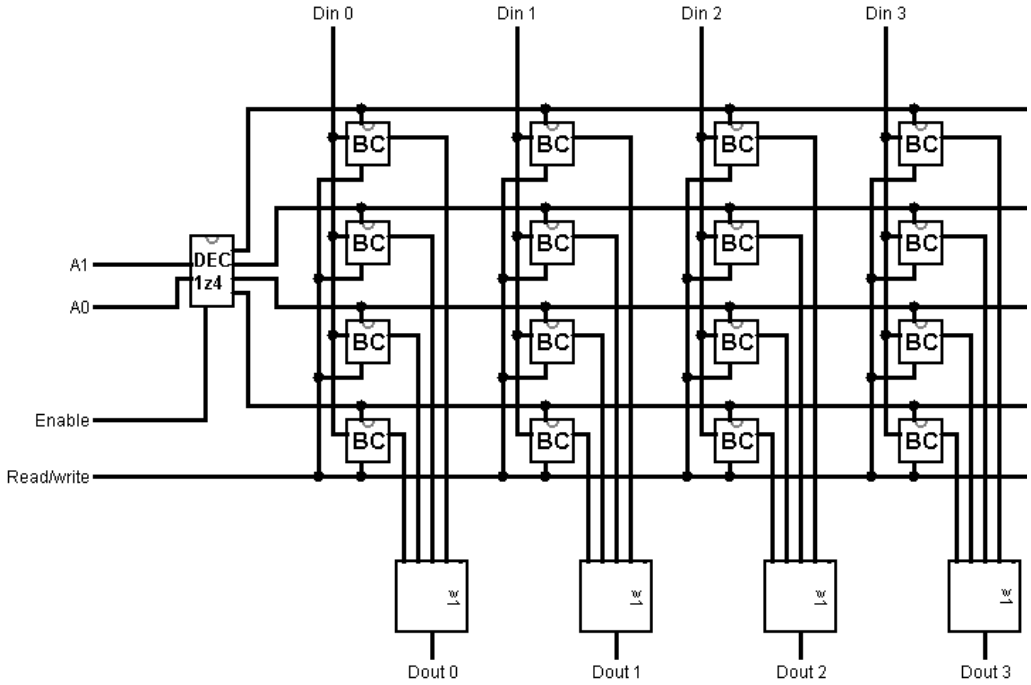
Použijeme v príklade usporiadanie po štyroch bunkách, teda po štyroch bitoch. Každá takáto skupina buniek je uložená v jednom riadku, číslo riadku tvorí jej jednoznačnú adresu. Pri práci s pamäťou potom vyberáme adresovými signálmi (A0 a A1) jeden riadok, s ktorým budeme pracovať. Výber konkrétneho riadku zabezpečuje binárny dekóder typu 1 z N. Pre adresáciu pamäte so štyrmi riadkami potrebujeme dvojbitový binárny dekóder.



V počiatkoch počítačov sa vo funkcii pamäťových buniek používali feritové jadierka, prevlečené (ručne!) elektrickými vodičmi. Pri zápise sa pustil elektrický prúd v práve jednom riadkovom a jednom stĺpcovom adresnom vodiči. Len jedným jadierkom tak pretekalo dvojnásobné množstvo prúdu, ktoré stačilo na jeho zmagnetizovanie (polovičná dávka nestačila). Čítanie sa robilo deštrukčným spôsobom – jadierko sa zmagnetizovalo (ako pri zápise) a na čítacom vodiči (prevlečenom cez všetky jadierka) sa zaznamenalo, či niekde k zmene magnetizácie došlo. Mohli sme tak poznať, či k jeho zmagnetizovaniu došlo až pri čítaní, alebo tomu tak bolo už predtým. Po čítaní bolo treba bit opäť zapísať.



Pamäť 32x32 bitov (rozmerov 10x10 cm) s magnetickými feritovými jadierkami



Pamäť so štyrmi bitmi v štyroch riadkoch, spolu 16 bitov

Vstupy A0 a A1 dekódera 1 z 4 určujú číslo aktívneho riadku. Vstup Select (z angl. select – výber) slúži na aktivovanie celého bloku pamäte. Ak je na vstup Select privedená úroveň log.0, všetky výstupy z dekódera su nastavené na log.0. Ak je na vstupe Select log.1, aktívny je práve jeden z výstupov z dekódera.

Vstup Read/write celej pamäte je privedený na vstup read/write jednotlivých pamäťových buniek. Vstupy Din0 až Din4 sú privedené na vstupy input buniek v jednotlivých stĺpcoch. Vstup Din0 je teda privedený na vstupy všetkých buniek v prvom stĺpci, vstup Din1 na bunky v druhom stĺpci, Din2 na bunky v treťom a Din3 na bunky vo štvrtom stĺpci. Rovnako po stĺpcoch sú privedené aj výstupy jednotlivých buniek na spoločné viacvstupové logické členy OR, z ktorých vychádzajú výstupy Dout0 až Dout3. Do takejto pamäte je možné zapísať alebo prečítať súčasne štyri bity.

Jednotlivé pamäte je možné spolu spájať do väčších celkov.

Úloha 2

Navrhnite a v prostredí Logisim realizujte spojenie štyroch vytvorených modulov 16 bitových pamätí do väčšej pamäte s kapacitou osem osembitových záznamov.

Čo sme sa naučili

Pokiaľ chceme, aby si logický obvod niečo pamätal (aby jeho výstup nezávisel len na okamžitom stave vstupov, ale aj na predchádzajúcich stavoch), musíme v obvode pripustiť spätné väzby. Spätné väzby môžu v obvode spôsobovať kolízie, preto treba obvod pred použitím riadne analyzovať.

Poznali sme niekoľko základných bistabilných sekvenčných obvodov, ktoré slúžia ako základné kamene k práci s binárnymi údajmi. Dajú sa využiť ako pohotové registre s možnosťou snímania a zaznamenania okamžitej situácie na vstupoch. Môžeme ich využiť aj v pamäťových maticiach, ktoré umožňujú lepšiu manipuláciu so skupinami bitov, ku ktorým môžeme priamo pristupovať podľa jednoznačnej adresy.

Čo sme sa naučili v tomto module

Zhrnutie

V module sa účastníci oboznámili s rôznymi formami kódovania znakov, nezáporných, celých a reálnych čísiel v počítači. Ukázali sa niekoľké spôsoby vykonávania aritmetických a logických operácií.

Účastníci získali prehľad o jednoduchých logických a sekvenčných obvodoch. Vedia, ako sa dá spracovať a uchovávať binárna informácia.

Preverenie výstupných vedomostí

Účastník vie v programe Logisim navrhnúť osembitový register.

Literatúra a použité zdroje

[1] Bernard, M.J., Hugon, J. (1986) *Od logických obvodů k mikroprocesorům*, SNTL Praha, 1986

[2] Gvozdjak, L., Hanulová, Ľ., Jankovičová, A., Molnár, Ľ. (1987) *Počítače a programovanie*, Alfa Bratislava, 1987

[3] Hillis, W.D. (1998) *The Pattern on the Stone*, Basic Books, New York 1998 (slov. preklad *Obrazce v kameni*, Kalligram, 2002, ISBN 80-71490-59-8)

[4] Jirovský, V. (2000) *Principy počítačů*, Matfyzpress Praha, 2000, ISBN 80-85863-51-0

[5] Patterson, D.A. (2007) *Computer Organization and Design*, 3. ed., Morgan Kaufmann, 2007, ISBN 978-0-12-370606-5

[6] Predko, M., (2008) *Digitální elektronika (bez předchozích znalostí)*, CPress, 2008, ISBN: 978-80-251-2124-5

[7] Stallings, W. (2005) *Computer Organization and Architecture: Designing for Performance*, 7. ed., Prentice Hall, 2005, ISBN 978-0-13-185644-8

[8] Simulačný program Logisim dostupný pod licenciou GNU GPL: http://ozark.hendrix.edu/~burch/logisim/index.html

[9] Tanenbaum, A.S. (2005) *Structured Computer Organization*, 5. ed., Prentice Hall, 2005, ISBN 978-0-13-148521-1

[10] White, R.; Downs, T. (2002) *How Computers Work*, 6. ed., Pearson, 2002 (čes. preklad *Jak fungují počítače*, SoftPress, 2003, ISBN 80-86497-48-8)

Tento študijný materiál vznikol ako súčasť národného projektu Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika v rámci Aktivity „Vzdelávanie nekvalifikovaných učiteľov informatiky na 2. stupni ZŠ a na SŠ“.

Autori © RNDr. Peter Gurský, PhD.
RNDr. Jozef Jirásek, PhD.
Mgr. Samuel Kupka
RNDr. Veronika Vaneková

Názov Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

Podnázov Počítačové systémy 1

Študijný materiál prešiel recenzným pokračovaním.

Recenzenti doc. Ing. Peter Fabián, PhD.
doc. Ing. Ľudovít Trajteľ, PhD.

Počet strán 40

Náklad 300 ks

Prvé vydanie, Bratislava 2010

Vydal Štátny pedagogický ústav, Pluhová 8, 830 00 Bratislava, v súčinnosti s Univerzitou Pavla Jozefa Šafárika v Košiciach, Univerzitou Komenského v Bratislave, Univerzitou Konštantína Filozofa v Nitre, Univerzitou Mateja Bela v Banskej Bystrici a Žilinskou univerzitou v Žiline

Vytlačil BRATIA SABOVCI, s r.o., Zvolen

ISBN 978-80-8118-033-0