Reprezentácia údajov

V pamäti počítača sú uložené informácie, a to:

- **inštrukcie** (kódy inštrukcií)
- **adresy** (bývajú súčasťou inštrukcie),
  kód určujúci polohu pamäťového miesta, kde je
  - uložený údaj (údaje)
  - uložená inštrukcia, ktorá bude vykonávaná mimo poradia
    (cieľová adresa skokovej inštrukcie, začiatok podprogramu)
- **údaje**

Počítač uchováva údaje (informácie) ako postupnosť binárnych číslic (0,1) v **konečnom počte bajtov** (slov). Interpretácia informácie je závislá od polohy binárnej hodnoty v danej postupnosti.
Spôsob kódovania údajov sa nazýva **formát údajov**. Formát údajov je daný **konvenciou** (dohodou).

Medzi elementárne údajové typy patria
- číslo
- logická informácia
- textová informácia
- obrazová informácia

Komplexnejšie (zložitejšie) informácie sa uchovávajú v tvare vektorov, matíc, štruktúr, únií (unions) a pod.

Čísla

Počítač môže uchovávať len **celé** čísla alebo **racionálne** čísla

Celé čísla

Obsah bajtov môže byť interpretovaný buď

- ako **celé číslo** (číslo so znamienkom – bez prívlastku)
- alebo **celé nezáporné číslo** (číslo bez znamienka – **unsigned**)

Obmedzený rozsah (nemožno zobraziť ľubovolne **malé** alebo **veľké** číslo).

Pozn. 9 223 372 036 854 775 807 cca. **9000 svetelných rokov s rozlíšením 1m**

	Min	Max
1B	-128	127
unsigned	0	255
2B	-32 768	32 767
unsigned	0	65 535 (64Ki)
4B	-2 147 483 648	2 147 483 647
unsigned	0	4 294 967 295 (4Gi)
8B	-9 223 372 036 854 775 808	9 223 372 036 854 775 807
unsigned	0	18 446 744 073 709 551 615

Dĺžky typov v C (Ubuntu64 LLLP64)

Počet bitov, ktoré zaberá dátový typ, pre rôzne dátové modely (3 ks 64 bitových a 2 ks 32 bitových)

Použitý model = f (prekladač, HW, OS, parametre)

V jazyku C platí len pravidlo short <= int <= long <=smerník

http://www.unix.org/version2/whatsnew/lp64_wp.html

Datatype	LP64	ILP64	LLP64	ILP32	LP32
char	8	8	8	8	8
short	16	16	16	16	16
_int32		32			
int	32	64	32	32	16
long	64	64	32	32	32
long long			64		
pointer	64	64	64	32	32

Program na zistenie veľkosti typov

(B vs b, bit vs byte)

```
#include<stdio.h>
int main ()
{
  printf ("size of char:\t\t%ld\n",8*sizeof(char));
   printf ("size of short:\t\t%ld\n",8*sizeof(short));
   printf ("size of int:\t\t%ld\n",8*sizeof(int));
   printf ("size of long:\t\t%ld\n",8*sizeof(long));
   printf ("size of pointer:\t%ld\n",8*sizeof(void *));
   printf ("size of long long:\t%ld\n",8*sizeof(long long));
   //printf ("size of _int32:\t%ld\n",8*sizeof(_int32));
}
```

- Niektoré dátove modely
  - ILP-integer=long=pointer/smerník ,
  - LP-long=pointer/smerník
  - LLLP- long=long long=pointer/smerník.
  - Číslo je veľkosť smerníka v bitoch

Dĺžky typov v C (Ubuntu64 LLLP64)

- **Korektný kód bez varovaní na 32 bit.**

```
#include<stdio.h>

int main ()
{
#ifdef  __i386__
    printf("32 bitov-386 resp. akykolvek 32 bit x86 chip\n");

    printf ("size of char:\t\t%d\n",8*sizeof(char));
    printf ("size of short:\t\t%d\n",8*sizeof(short));
    printf ("size of int:\t\t%d\n",8*sizeof(int));
    printf ("size of long:\t\t%d\n",8*sizeof(long));
    printf ("size of pointer:\t%d\n",8*sizeof(void *));
    printf ("size of long long:\t%d\n",8*sizeof(long long));
    //printf ("size of _int32:\t%d\n",8*sizeof(_int32));
#endif
```

Dĺžky typov v C (Ubuntu64 LLLP64)

- **Korektný kód bez varovaní na 32 bit- pokračovanie**

```
#ifdef __i486__
    printf("32 bitov-486\n");
#endif
#ifdef __i586__
    printf("32 bitov-586\n");
#endif
#ifdef __i686__
    printf("32 bitov-686\n");
#endif
#ifdef __x86_64__
    printf("64 bitov\n");

    printf ("size of char:\t\t%ld\n",8*sizeof(char));
    printf ("size of short:\t\t%ld\n",8*sizeof(short));
    printf ("size of int:\t\t%ld\n",8*sizeof(int));
    printf ("size of long:\t\t%ld\n",8*sizeof(long));
    printf ("size of pointer:\t%ld\n",8*sizeof(void *));
    printf ("size of long long:\t%ld\n",8*sizeof(long long));
    //printf ("size of _int32:\t%ld\n",8*sizeof(_int32));
#endif
}
```

Dĺžky typov v C (Ubuntu64 LLLP64)

Výstup

- **64 bitov**
  - size of char: 8
  - size of short: 16
  - size of int: 32
  - size of long: 64
  - size of pointer: 64
  - size of long long: 64

- **32 bitov**
  - **32 bitov-386 resp. akýkoľvek 32 bit x86 chip**
  - size of char: 8
  - size of short: 16
  - size of int: 32
  - size of long: 32
  - size of pointer: 32
  - size of long long: 64
  - **32 bitov-686** (AMD K6, Intel Pentium Pro a ich modifikácie)

Racionálne čísla

- nepoužíva sa formát racionálnych čísel bez znamienka
- obmedzený rozsah
- presnosť zobrazenia racionálnych čísel je obmedzená
- len podmnožina racionálnych čísel sa dá zobraziť presne
- nie je možné zobraziť presne iracionálne číslo napr. π , $\sqrt{2}$

		min |X|	max |X|
4B	float	$1.18*10^{-38}$	$3.40*10^{38}$
8B	double	$2.23*10^{-308}$	$1.79*10^{308}$
10B	long double	$3.37*10^{-4932}$	$1.18*10^{4932}$

Iný 32 bitový kompilátor
12B long double

Iný 64 bitový kompilátor
16B long double

Logické informácie

Agregácia hodnôt logických premenných do dátových typov, ktoré sú násobkami bajtov

Používajú sa na uchovávanie informácií o stave:

- procesora (stavové slovo)
- vstupno/výstupných obvodov (zariadení)
- procesov alebo zariadení riadených počítačom

Slúžia na riadenie toku programu (postupnosti inštrukcií).
V niektorých prípadoch slúžia na riadenie stavu daného zariadenia alebo procesu.

Príklady stavových slov procesorov:

Procesor Z80

7	6	5	4	3	2	1	0
S	Z	X	H	X	P/V	N	C

Mikropočítač AVR ATmega16

7	6	5	4	3	2	1	0
I	T	H	S	V	N	Z	C

Signalizujú najmä atribúty výsledku väčšiny aritmeticko-logických operácií vykonaných procesorom. Význam niektorých bitov je rovnaký :

S-znamienko výsledku, **Z**-nulový výsledok, **V**-pretečenie výsledku,

C- prenos do vyššieho rádu

Význam niektorých bitov nemusí byť definovaný **X**

Hodnoty jednotlivých bitov slúžia na „dynamické“ vetvenie programu

Obsah registra sa dá čítať, dá sa programovo meniť

Zápis do bitu **I** program **(programátor)** povoľuje alebo zakazuje spustenie **všetkých špeciálnych podprogramov (obsluhy prerušenia).**

Textové informácie

Elementárnym prvkom textovej informácie je **znak** (symbol).
Postupnosť znakov definovanej dĺžky alebo zakončená dohodnutým znakom tvoria **znakové reťazce (stringy).**

Pod znakom rozumieme: *číslice, písmená, symboly* napr. ***?***, ***!***, ***§***, *a riadiace znaky napr.* ***CR*** (návrat kurzora na začiatok riadku), ***LF*** (posun kurzora o riadok ďalej) atď. z ďalekopisov Line Feed, Carriage return

Textové informácie sa používajú pri komunikácii
- programového vybavenia (operačný systém, aplikácie) s obsluhou
- pri komunikácii medzi procesom (spustený program) s iným procesom (na tom istom, alebo inom počítači resp. zariadení)

Textová informácia môže byť dôležitou zložkou spracovávaných údajov (textové editory, kompilátory, databázy, atď.).

ASCII (podľa American National Standards Institute =ANSI)

(*American Standard Code for Information Interchange*)

Pôvodne 7 bit kód (celkovo 128 znakov)

- kódovanie znakov – základom anglická abeceda
- prvýkrát zverejnené v roku 1967
- posledná úprava v roku 1986
- obsahuje 32 nezobraziteľných znakov (väčšinou už nepoužívaných riadiacich znakov)
- 96 zobraziteľných znakov
- spolu 128 znakov - 7 bitový kód
- najvyšší bit bol rezervovaný pre paritný bit
- protipólom ASCII je EBCDIC kód, ktorý používa vo svojich počítačoch IBM.

Bol neúspešne navrhnutý za štandard amerického NIST=National Institute of Standards and Technology, Escape sekvencie sú v ANSI

ASCII kód:

Riadiace znaky: <0 $_{16}$ až 20 $_{16}$)

CR – Carriage Return
návrat vozíka

LF -Line Feed
posun o riadok

FF – Form Feed
nová stránka

BEL – Bell (zvonček)

Riadiace znaky prenosu, napr:
STX – začiatok textu
ETX – koniec textu, NAK/ACK

Stĺpec / riadok	0	1	2	3	4	5	6	7
0	NUL	DLE		0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	I	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Extended ASCII (high ASCII)

Využitie najvyššieho bitu ďalších 128 znakov (špeciálne symboly, znaky národných abecied, grafické symboly).

128 rozšírených znakov je málo pre „národné“ gramatiky programovo sú prepínané.

Kódové stránky (Code Page)

Na Slovensku sa najčastejšie používali (a ešte stále používajú)

- CP 852 (ISO 8859-2, Latin-2, stredoeurópske jazyky ISO 8859 sú Západné)
- Windows 1250 (ANSI 1250)
- CP 895 (kód bratov Kamenických, nebol štandardizovaný)

Problém, ako zapnúť správnu stránku pre zobrazovanie.

Kódovanie znakov nerieši problém, ako znaky zobrazovať.

Toto rieši výber **Fontu** (príloha).

Problém diakritiky (CZ/SK)

Kódy :

	Č	č	Š	š	Ž	ž
Kamenických	80	87	98	A8	92	91
Latin 2	AC	9F	E6	E7	A6	A7
Windows 1250	C8	E8	8A	9A	8E	9E
Unicode U+	010C	010D	0160	0161	017D	017E

Problém s národnými gramatikami sa snaží riešiť systém UNICODE, kde je znak kódovaný do dvoch bytov, a ktorý predstavuje skoro všetky svetové jazyky.

Unicode

http://www.unicode.org/

Štandard kódovania znakov vyvinutý organizáciou **Unicode Consortium** (1991) (paralelne **ISO 10646** – došlo k zjednoteniu kódovania znakov)

- snaha obsiahnuť takmer všetky znaky používané v jazykoch na svete (japončina, čínština – tisíce znakov)
- prvých 256 znakov je zhodných s rozšíreným ASCII
- pôvodne bol pre každý znak pridelený 16-bitový jedinečný kód
- v súčasnosti 31-bitové, štandard je navrhnutý tak, že všetky možné znaky rozdeľuje do sedemnástich skupín po 65536 znakoch. To znamená môžeme definovať až 1 114 112 (17×2^{16}) znakov.

ISO 10646 – definuje **UCS** (Universal Character Set)

Štandard Unicode sa oproti ISO 10646 navyše zaoberá

- algoritmami pre písmo písané smerom doľava (arabský jazyk)
- podporou obojstranných textov (ako napr. zmes hebrejčiny a latinky)
- algoritmy pre abecedné triedenie a porovnávanie textov

Nevýhodou je dĺžka znakov, prítomnosť kódov v znakoch ako nulový bajt, znak "\", atď.

Vznik systémov kódovania UTF (**Unicode Transformation Format**) ako sú UTF-8, UTF-16 a UTF-32.

Väčšina rozhraní systému Windows používa formu UTF-16. - od Windows NT 5.0 (predávané ako XP, staršie mali CP1250,...)

UTF-8 : Kóduje do 1 až 6 bajtov; Do 1 bajtu kóduje prvých 128 znakov zo znakovej sady US-ASCII (U+0000 až U+007F);

Unicode hodnota	UTF-8 sekvencia
U-00000000 - U-0000007F	0xxxxxxx Znaky s číslami od 0 do 127 sa kódujú ako jeden bajt
U-00000080 - U-000007FF	110xxxxx 10xxxxxx Znaky s číslami od 128 do 2 047 sa kódujú ako dva bajty
U-00000800 - U-0000FFFF	1110xxxx 10xxxxxx 10xxxxxx Znaky s číslami od 2 048 do 65 535 sa kódujú ako tri bajty
U-00010000 - U-001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx Znaky s číslami od 65 536 do 1 114 111 sa kódujú ako štyri bajty
U-00200000 - U-03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
U-04000000 - U-7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Staršie webové prehliadače (**HTML** dokumnety) podporovali len znakovú sadu ASCII,
moderné majú prednastavenú sadu UTF-8 alebo ISO-8859-1, inak musia mať v položke <meta> uvedenú znakovú sadu buď ISO-8858-xx, UTF8 alebo UTF16. (MSIE ignoruje a použije autodetekciu kódovania)
Podobné princípy sú uplatňované v **XML** (eXtensible Markup Language) dokumentoch,
ktoré okrem iného slúžia na výmenu údajov medzi aplikáciami a zverejňovanie dokumentov.

Obrazová informácia

V počítači sa uchováva a spracováva číslicový (digitálny) obraz.

PIXEL – PICture ELement (najmenšia časť obrazu). Bodová reprezentácia (rastrová reprezentácia obrazu) – uloženie v matici (bit-map)

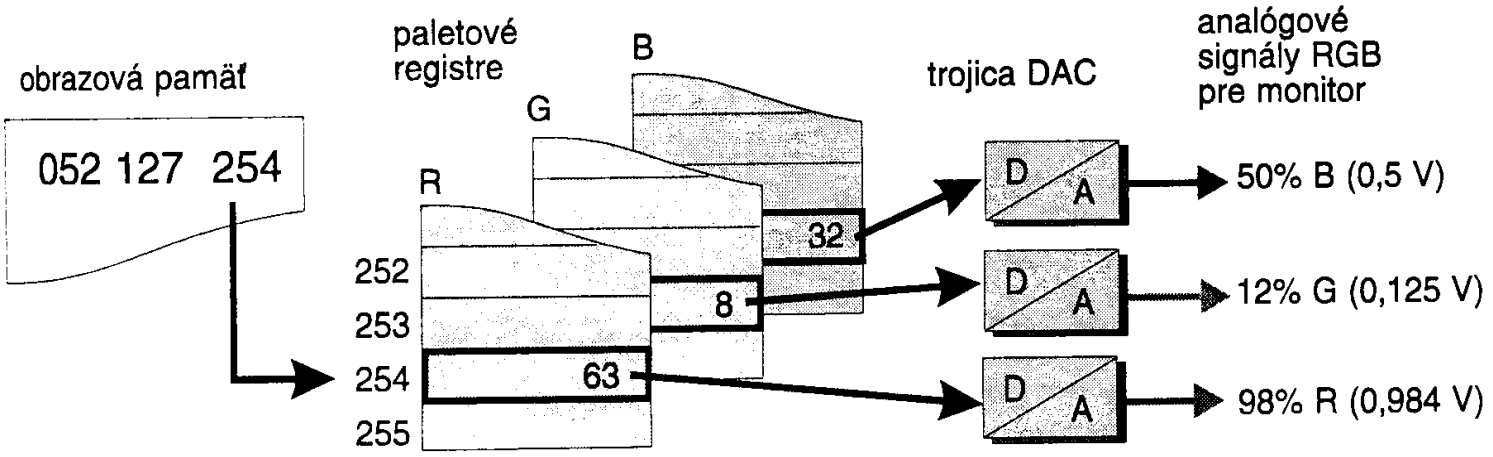
V matici, na uloženie jedného pixel sa používa

- 1 bit - pre čieno-biele obrazy (B/W)
- 1 bajt - pre šedotónové (grayscale) a farebné obrazy, obsah bajtu je indexom do tabuľky 3x256 bajtov (palety), ktorá definuje 256 rôznych farieb, trojica bajtov uchováva úrovne farebných zložiek RGB (Red Green Blue))
- 3 bajty – farebné obrazy tzv. True Color, v bajtoch sú uložené zvlášť hodnoty (R,G,B) – umožňuje uchovávať jednu z 2^{24} = 16 777 216 farieb

Pozn. kvalita šedotónových obrazov pri použití 1 alebo 3 bajtov je rovnaká.

- 4 bajty – (v poslednom období) RGBA (Red Green Blue Alpha) , hodnota zložky A definuje stupeň priehľadnosti (transparencie) pixela

Použitie RGBA znamená kvalitatívnu zmenu pri algoritmoch generovania rastrových obrazov (prechod od používania **logických** operácií nad pixelmi na požitie **aritmetických** operácií), čo vyžaduje podstatne výkonnejší hardvér počítača.



DAC (**D**igital to **A**nalog **C**onverter) – číslicovo-analógový prevodník

Počítačová grafika:

- **rastrová** (kamera, skener a pod.) – spracovanie obrazu (Image Processing)
- **vektorová**
  - **2D** (prvky - elementárne objekty s dvomi súradnicami) – technická dokumentácia, mapy, grafy, a pod.)
  - **3D** (prvky - elementárne objekty s tromi súradnicami) - priestorové simulácie, virtuálna realita, počítačové videnie (Computer Vision),...

3D – grafika (mapovanie priestor->virtuálna kamera, veľmi náročné na výpočet),

-softvérová podpora (knižnice OPEN GL- open Graphics „anguage , Direct3D,...)

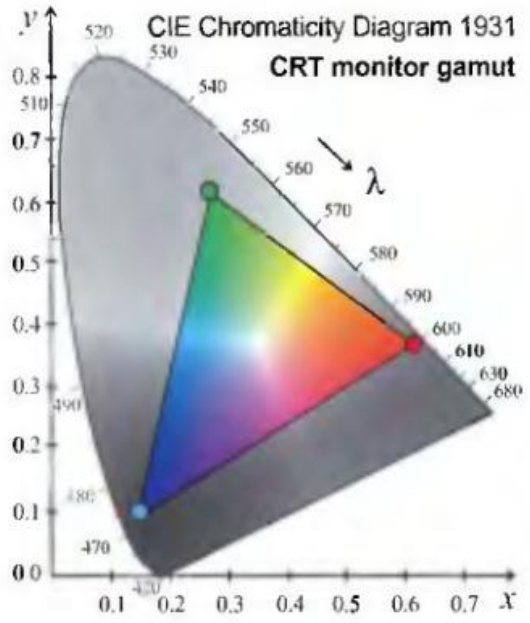
- hardvérová **GPU** (**G**raphics **P**rocessing **U**nit), grafické karty založené na paralelnom výpočte použitím openCL(**open Compute Language**) či **CUDA** (**C**ompute **U**nified **D**evice **A**rchitecture)

(NVIDIA GeForce GTX Titan 2688 CUDA jadier) (250W)(6 GB pamäte)

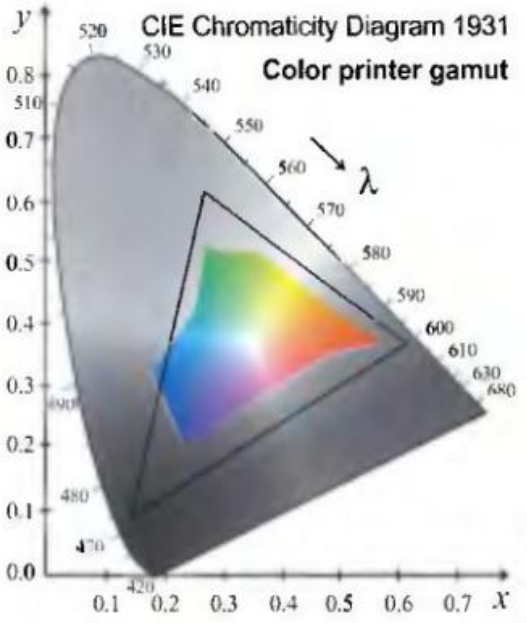
Priemerný človek dokáže rozpoznať

- absolútne (bez porovnávania) 200 farieb
- relatívne s možnosťou porovnať 200000 farieb

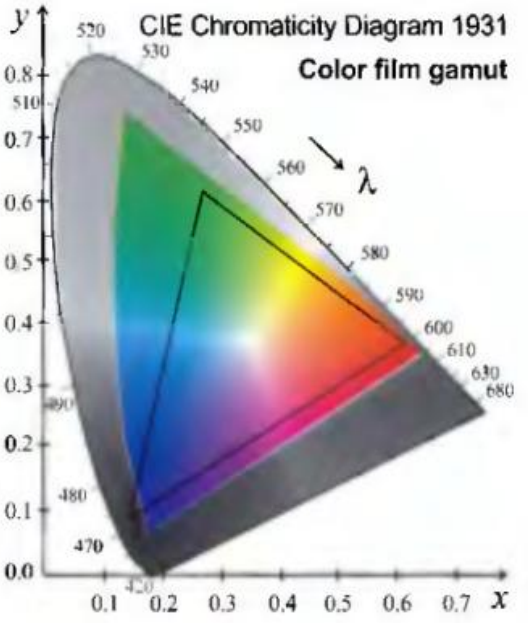
Technické zariadenia (monitor, tlačiareň) nedokážu reprodukovať všetky farby – schopnosť charakterizuje **gamut**



(a) CRT monitor



(b) printer



(c) film

Ukladanie viacbajtových údajov do pamäte

•Problém ukladania informácií je, ak veľkosť údajového typu nie je zhodná s veľkosťou strojového slova.
Ukladanie a čítanie z pamäti v strojových slovách
(dané šírkou dátovej zbernice 16b = 2B, 32b = 4B, 64b = 8B)
•Poradie ukladania bytov údajového typu do pamäte.

Slovom ENDIAN chceme povedať v akom poradí sú byty (resp. bity) usporiadané, ak chceme preniesť komplexnejšiu informáciu po menších častiach.

Niekedy sa tento problém nazýva aj NUXI problém .
(reťazec „UNIX“ môže počítač s opačnou orientáciou zobraziť ako „NUXI“)

little endian orientation

$$(305\ 419\ 896)_{10} = (12\ 34\ 56\ 78)_{H}$$

3 2 1 0

Adresa	BE	LE	ME	
			2301	1032
0x1000	0x12 (MSB)	0x78	0x34	0x56
0x1001	0x34	0x56	0x12	0x78
0x1002	0x56	0x34	0x78	0x12
0x1003	0x78 (LSB)	0x12	0x56	0x34

Stredný endian – poradie bajtov 2301 (PDP11)

V architektúrach počítačov sa využívajú nasledujúce ukladacie endiány :

- len malý endian (Intel i386)
- len veľký endian (MC 68030)
- nastaviteľný: - špeciálnym signálom pri Reset (MIPS 2000)
  - nastaviteľný inštrukciou (Intel i860, i486)

Musí byť riešená konverzia rôznych endianov, ak niektorý podsystém počítača pracuje v inom ukladacom režime.

Dátum

US : middle - endian

Month, Day, Year (May, 24 th,2006 = 5/24/2006)

Europe : little - endian

Day, Month, Year (24 th, May, 2006 = 24/ 5/2006)

China, Japan & ISO 8601: big - endian

Year, Month, Day (2006, May, 24 th = 2006-05-24)

Poznámky k endianom:

Nedá sa povedať, že niektorý endian je výhodnejší voči inému, len ak zapíšeme dátum v big endian – ľahšie sa triedia položky.

Ak prenášame súbory medzi počítačmi s rôznymi endianmi, treba robiť vykonať transformáciu.

Text “hello” zapísaný pomocou ASCII kódu je:
0x68, 0x65, 0x6c, 0x6c, 0x6f

Ak tento istý text zapíšeme pomocou **UTF-16** treba rozlišovať zápis:

Big endian:
0xfe 0xff 0x00 0x68 0x00 0x65 0x00 0x6c 0x00 0x6c 0x00 0x6f

Little endian:
0xff 0xfe 0x68 0x00 0x65 0x00 0x6c 0x00 0x6c 0x00 0x6f 0x00

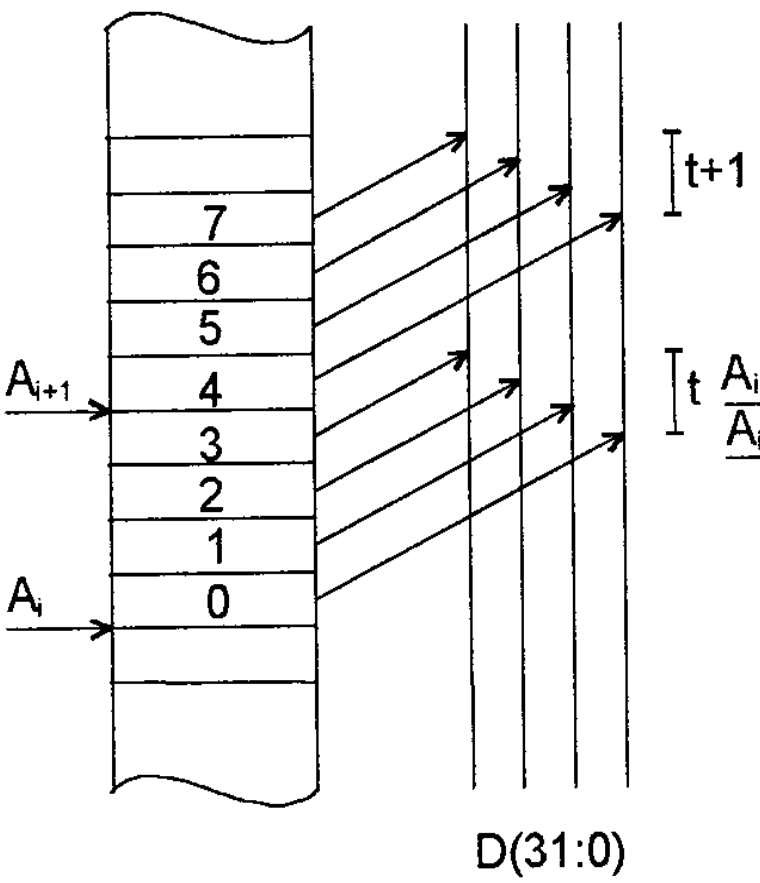
Zarovnávanie bajtov

Sprístupňovanie (čítanie a zápis objektov) jednoduchých alebo viacbajtových sa môže uskutočňovať v režime:

- **zarovnávaného sprístupňovania** bajtov (používa sa v arch. **RISC**)
  - údaje sa ukladajú na adresy, ktoré sú celočíselným násobkom strojového slova (napríklad pre 32-bitové slovo, sa ukladajú na adresy, ktoré sú násobkom čísla 4). V prípade ukladania údajov, ktorých veľkosť nie je celočíselným násobkom strojového slova, nie sú využité niektoré bajty v pamäti.
  - jednoduchší hardvér, jednoduchšie sprístupňovanie

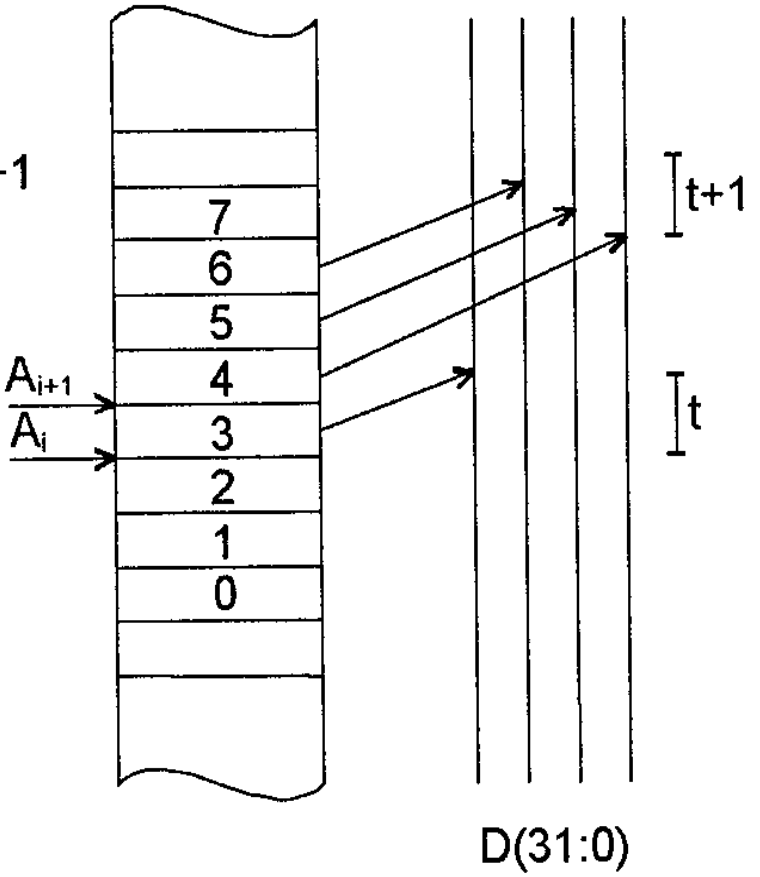
- **nezarovnávaného sprístupňovania** bajtov (používa sa v arch. **CISC**)
  - zložitejší hardvér (najmä pri stránkovaní pamäte)
  - zložitejšie sprístupňovanie (viacej strojových cyklov pri zápise a čítaní, pomalšie čítanie a zápis do pamäte)
  - pohodlnejšie programovanie
  - údaje sa ukladajú „úsporne“ bezprostredne za sebou (úspora pamäte)

Zarovnaný výber



Slovo (0,1,2,3) sa sprístupní
za jeden adresovací prístup

Nezarovnaný výber



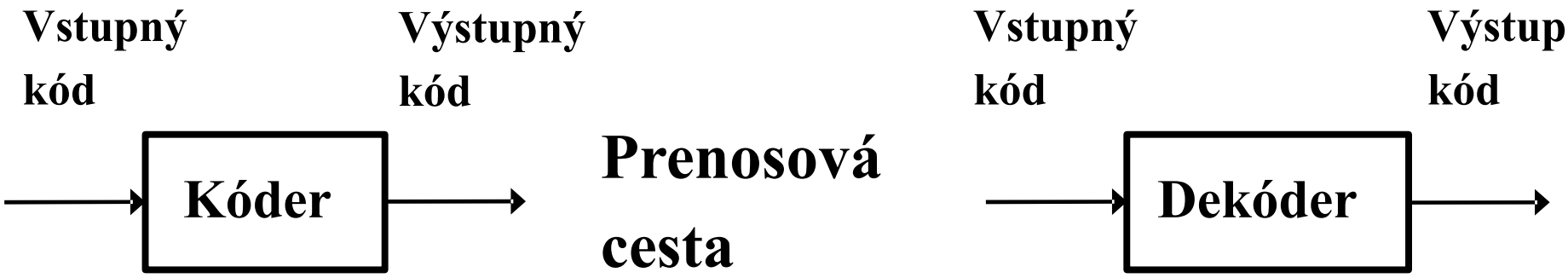
Slovo (3,4,5,6) sa sprístupní
za dva adresovacie prístupy

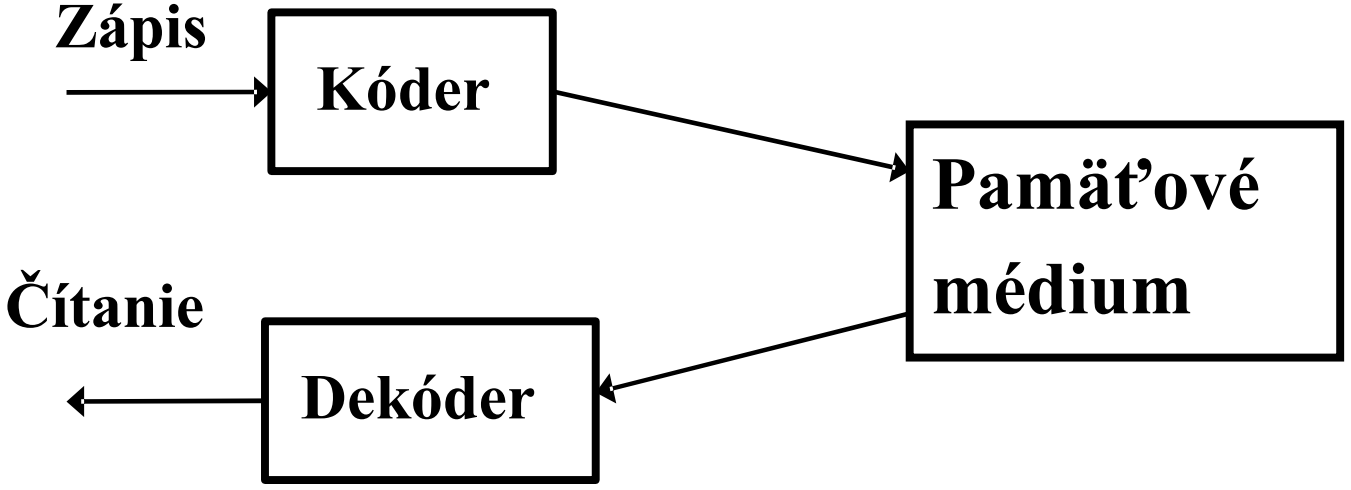
Kódovanie údajov

Kódovanie údajov (binárnych slov) = **mapovanie** množiny **vstupných slov** (kódov) do množiny **výstupných slov** (kódov)

Cieľ':

- **zabrániť vzniku chyby** pri prenose a uchovávaní informácie
- **komprimácia dĺžky informácie** (zrýchlenie prenosu informácie, lepšie využitie pamäťových médií)
- **prispôsobenie sa konvenciám** pri prenose dát (protokol)
- **ochrana údajov** (šifrovanie),
- počítač – binárne čísla, operátor – dekadické čísla, atď.


Vstupný kód
Výstupný kód
Kóder
Prenosová cesta
Vstupný kód
Výstupný kód
Dekóder


Zápis
Kóder
Pamäťové médium
Čítanie
Dekóder

BCD kód, (kód s váhami 8 4 2 1)

0000 – 0
0001 – 1
0010 – 2
0011 – 3
0100 – 4
0101 – 5
0110 – 6
0111 – 7
1000 – 8
1001 – 9
1010 – x
1011 – x
1100 – x
1101 – x
1110 – x
1111 – x

Nevyužité

Prevod 1357 do BCD kódu

1	3	5	7
0001	0011	0101	0111

Kódovanie na zabránenie vzniku chyby Pri prenose a uchovávaní informácie

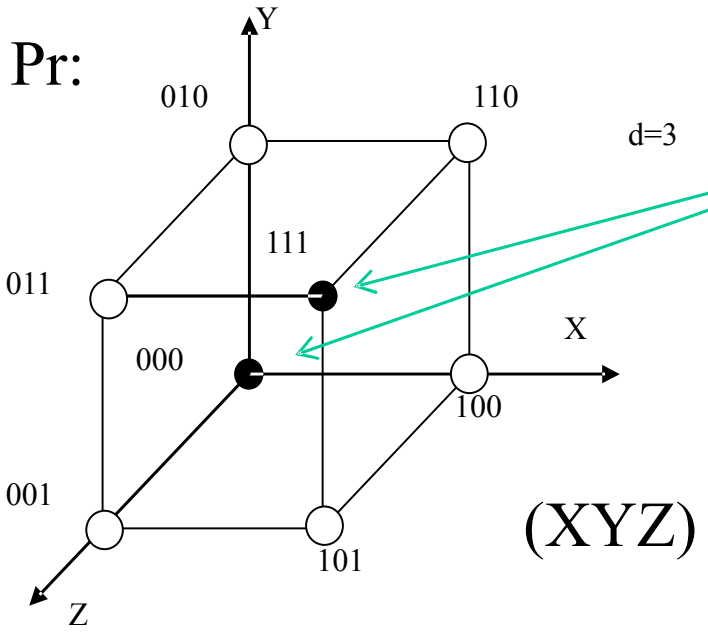
Hammingova vzdialenosť medzi **dvomi** binárnymi slovami (kódmi) **sa rovná počtu bitov**, v ktorých sa **nezhodujú** dané slová

Napríklad medzi štvorbitovými slovami

$(0111)_2=(7)_{10}$ $(1000)_2=(8)_{10}$

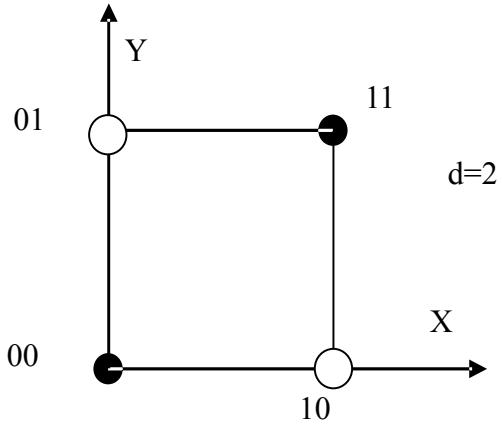
je Hammingova vzdialenosť 4.

Pr:

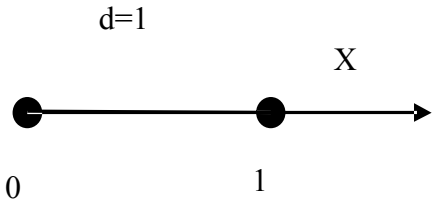


(XYZ)

- n **informačných bitov**
  (kód pôvodnej informácie)
- k **kontrolných bitov**
  (redundantné bity)



(XY)



(X)

d – **kódová vzdialenosť**

α – **počet detekovateľných chýb**

β – **počet opraviteľných chýb**

$$d \geq \alpha + 1$$

$$d \geq 2\beta + 1$$

Grayov kód:

Hammingova vzdialenosť medzi kódmi **= 1** (príloha)

Detekčné kódy (EDC)

sú schopné detekovať definovaný počet chýb

Skladajú sa z :

- **n informačných bitov** (kód pôvodnej informácie) a
- **k kontrolných bitov (** nenesú žiadnu novú informáciu, redundantné bity)

Najväčší dôraz sa kladie na odhalenie jednej chyby (jej pravdepodobnosť je najväčšia z možných chýb).

Princíp je založený na tom, že z 2^{n+k} možných prijatých slov je len 2^{n} správnych (neobsahujúcich definovaný počet chýb)

Paritný kód (detekčný)

- používa sa **jeden** kontrolný bit (paritný bit)
- parita môže byť – **párna** (even) alebo **nepárna** (odd)

Stanovenie parity :

$$\left(\sum_{i=0}^{m-1} a_i\right)(mod\,2) = a_{m-1} \oplus a_{m-2} \oplus \ldots \oplus a_1 \oplus a_0 = C,$$

$$kde\,(a+b)(mod\,2) = a \oplus b$$

Kde:

C=1 - kódové zabezpečenie **nepárnou paritou**,

C=0 - kódové zabezpečenie **párnou paritou**.

Príklad : Prenášame informáciu zakódovanú v 7 informačných bitoch

p	1	0	0	1	1	1	0

potom

Pre 11001110 *bude*

$$(1\oplus 1)\oplus(0\oplus 0)\oplus(1\oplus 1)\oplus(1\oplus 0)=(0\oplus 0)\oplus(0\oplus 1)=1$$

pre 01001110 *bude*

$$0\oplus 1\oplus 0\oplus 0\oplus 1\oplus 1\oplus 1\oplus 0=1\oplus 0\oplus 0\oplus 1=1\oplus 1=0$$

- pri párnej parite vysielame: **0100 1110**

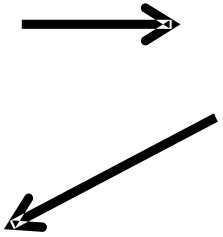
- pri nepárnej parite vysielame: **1100 1110** .

Ak nastane **nepárny počet chýb** pri prenášaní ľubovolného bitu, **paritný bit** prijatého slova nebude **správny**).

Blokový detekčný kód (maticový detekčný kód)

- pri **prenose bloku** viacerých informačných slov,
- použitie **priečnej** a **pozdĺžnej parity**

Bit	W_1	W_2	W_3	W_4	W_5	W_6
D_0	0	1	1	0	1	0
D_1	1	0	0	1	0	1
D_2	1	1	0	1	1	0



Bit	W_1	W_2	W_3	W_4	W_5	W_6	W_7
D_0	0	1	1	0	1	0	1
D_1	1	0	0	1	0	1	1
D_2	1	1	0	1	1	0	0
D_3	0	0	1	0	0	1	0

Bit	W_1	W_2	W_3	W_4	W_5	W_6	W_7
D_0	0	1	1	0	1	0	1
D_1	1	0	1	1	0	1	1
D_2	1	1	0	1	1	0	0
D_3	0	0	1	0	0	1	0

- detekuje viacero chýb
- opraviť je možné len jednu chybu

Samoopravné kódy (ECC)

Hammingove kódy

Patria medzi najjednoduchšiu skupinu **detekčných** a zároveň **samoopravných kódov**.

Generujú kód zložený z **m informačných bitov** vstupného slova a **k kontrolných bitov**, dĺžka výstupného kódu bude **n = m + k**. Kód sa označuje **H(n,m)**

Pre **H(7,4)**

Poloha bitu	7	6	5	4	3	2	1
Kód bitu	I_4	I_3	I_2	C_3	I_1	C_2	C_1

I_i - hodnota zdrojového bitu

C_i - hodnota kontrolného bitu

Kontrolné bity sa generujú podľa vzťahov

$$C_3 = I_2 \oplus I_3 \oplus I_4$$
$$C_2 = I_1 \oplus I_3 \oplus I_4$$
$$C_1 = I_1 \oplus I_2 \oplus I_4$$

Nech $I_4 I_3 I_2 I_1 = 1101$

potom
$$C_3 = 0 \oplus 1 \oplus 1 = 0$$
$$C_2 = 1 \oplus 1 \oplus 1 = 1$$
$$C_1 = 1 \oplus 0 \oplus 1 = 0$$

potom

$$I_4 I_3 I_2 C_3 I_1 C_2 C_1 = 1100110$$

Nech prijatý kód je $I_4 I_3 I_2 C_3 I_1 C_2 C_1$ 1 0 00110 (1 1 00110)

potom $C_3 = I_2 \oplus I_3 \oplus I_4 = 0 \oplus 0 \oplus 1 = 1$

$C_2 = I_1 \oplus I_3 \oplus I_4 = 1 \oplus 0 \oplus 1 = 0$

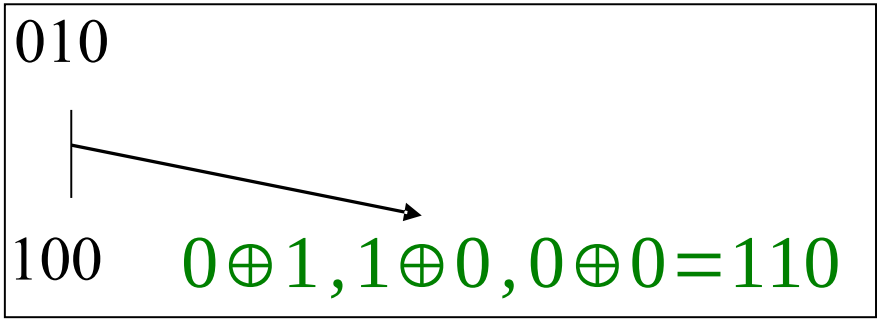
$C_1 = I_1 \oplus I_2 \oplus I_4 = 1 \oplus 0 \oplus 1 = 0$

chyba

pri prenose nastala chyba

Nová trojica kontrolných bitov je **100** namiesto **010**.

Teraz po dvojiciach určíme súčet modulo 2 pre kontrolné bity



Výsledok operácie nad kontrolnými bitmi dáva binárnu kombináciu, ktorej dekadická hodnota určuje polohu bitu, v ktorom treba opraviť hodnotu bitu. V tomto prípade je to hodnota $\mathbf{110_2 = 6}$, t.j. je to 5. bit ktorý obsahuje informáciu o $\mathbf{I_3}$. Prijatý byte: 1000110

Teraz predpokladajme, že pri prenose bude poškodený kontrolný bit, napr. $\mathbf{C}_2$

1100100

Nová trojica kontrolných bitov **000** namiesto **010**. Určíme polohu poškodeného bitu

010

000 $0 \oplus 0, 1 \oplus 0, 0 \oplus 0 = 010$

V tomto prípade je to hodnota $\mathbf{010}_2 = \mathbf{2}$, t.j. je to 1. bit, ktorý obsahuje informáciu o $\mathbf{C}_2$.

Hammingov kód je schopný opraviť chybu v prijatom bajte aj pre hodnoty kontrolných bitov.

Komprimácia dĺžky informácie

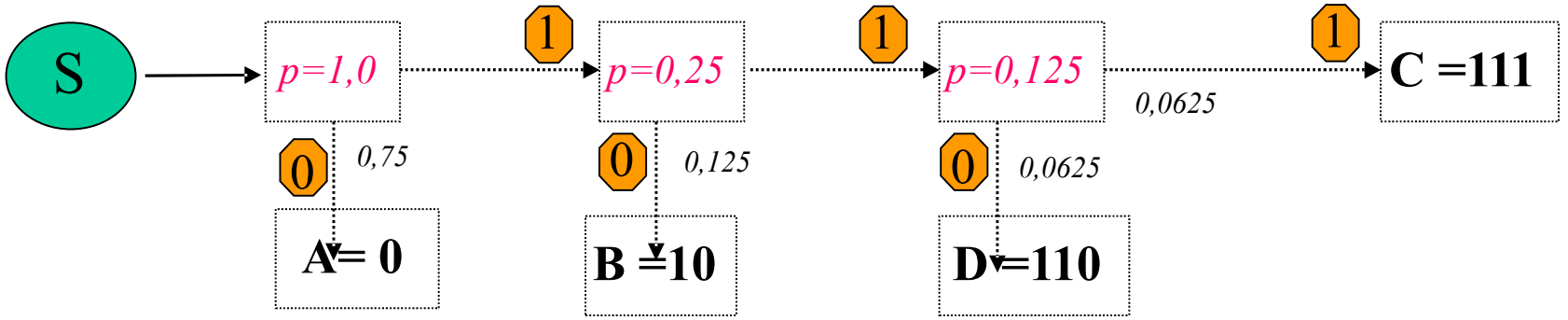
Huffmanov kód

- komprimačný kód
- premenlivá dĺžka výstupného kódu
- vychádza zo štatistiky (z pravdepodobností výskytu kódovaného slova, napr. písmen abecedy v textoch určitého typu), štatistickými metódami sa určuje kódovacia tabuľka (strom)
- tabuľky sú pevné a známe na strane kódera aj dekódera
- nevýhodou je sériové generovanie aj spracovanie kódu po bitoch
- používa sa v kódovaní faxových správ, v kódovaní rastrových obrazov (TIFF)

Príklad: Majme danú množinu 4 symbolov (znakov) s pravdepodobnosťami ich výskytu v prenášaných správach -dĺžka kódu 2 bity

Znak	Pravdepodobnosť výskytu (početnosť)	Kód
A	*0,75*	**00**
B	*0,125*	**01**
C	*0,0625*	**10**
D	*0,0625*	**11**

Vytvoríme kódovací strom, tak aby v koncových uzloch kratších vetiev končili kódy s väčšou pravdepodobnosťou :



Potom kódy jednotlivých znakov budú :

Znak	Pravdepodobnosť výskytu (početnosť)	Kód
A	*0,75*	**0**
B	*0,125*	**10**
C	*0,0625*	**111**
D	*0,0625*	**110**

Stredná dĺžka kódu bude :

$$1\frac{3}{4}+2\frac{1}{8}+3\frac{1}{16}+3\frac{1}{16}=\frac{12+4+3+3}{16}=\frac{22}{16}=\frac{11}{8}=1+\frac{3}{8}=1{,}375$$

bez tohto kódu je dĺžka 2

Literatúra :

[1] Jelšina, M.: Architektúry počítačových systémov,

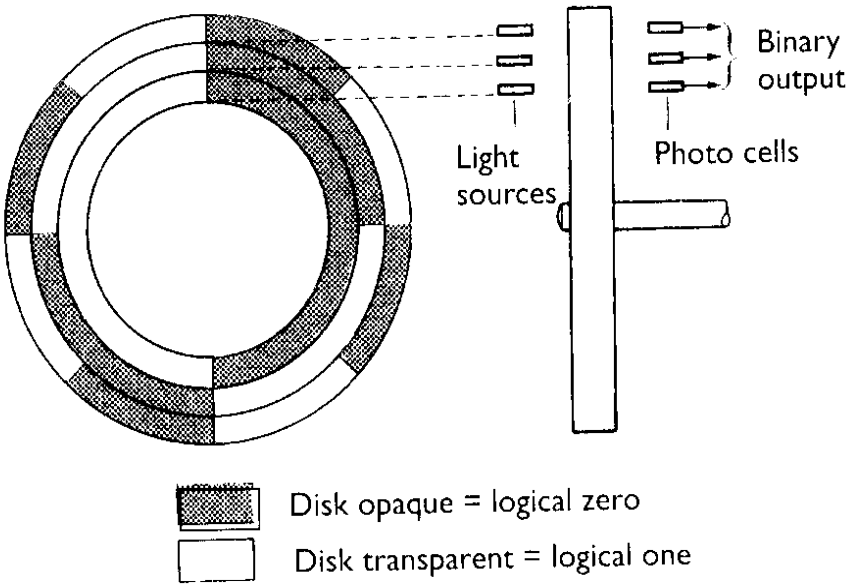
Princípy, ... ELFA 2002

[2] Clements,A: The Principles of Computer Hardware,
Oxford

2. Grayov kód

Problém správneho dekódovania polohy kódového kotúča

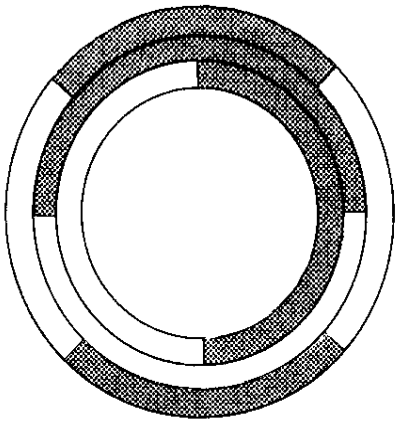
Sector	Angle	Binary code
0	0–45	0 0 0
1	45–90	0 0 1
2	90–135	0 1 0
3	135–180	0 1 1
4	180–225	1 0 0
5	225–270	1 0 1
6	270–315	1 1 0
7	315–360	1 1 1



Problém „presnej“ realizácie prechodov medzi priehľadnými a nepriehľadnými sektormi.

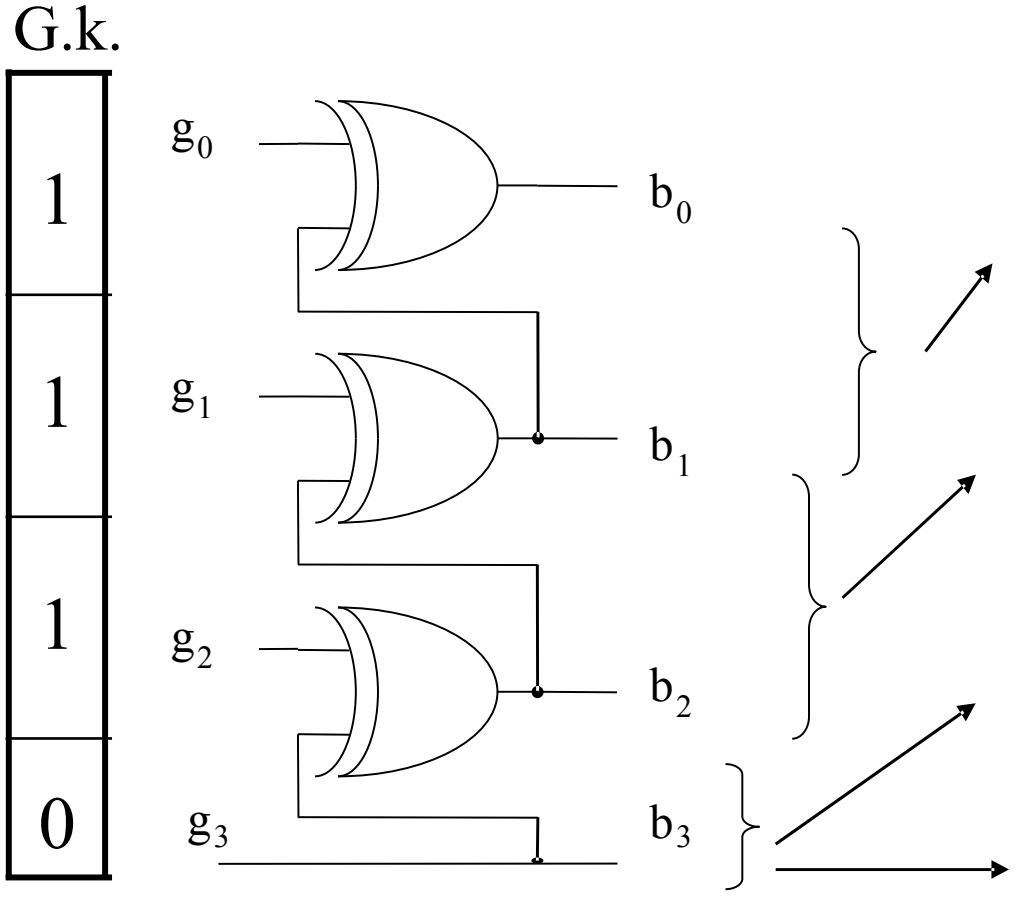
Riešením je kódovanie sektorov tak, aby pri zmene sektora sa menil kód len v jednom $bite \Rightarrow Hammingova$ vzdialenosť medzi kódmi susedných sektorov bola 1.

Sector	Angle	Gray code
0	0–45	0 0 0
1	45–90	0 0 1
2	90–135	0 1 1
3	135–180	0 1 0
4	180–225	1 1 0
5	225–270	1 1 1
6	270–315	1 0 1
7	315–360	1 0 0



Dĺžka kódového slova sa pre Grayov kód nezmení.

Pr.: Dekóder:



Kóder: