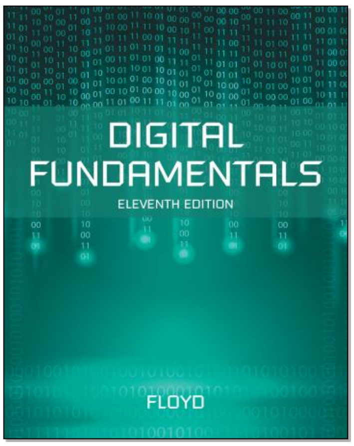
Digital Fundamentals

ELEVENTH EDITION



Rozdział 2

Systemy liczbowe,
operacje i kody

>>> Jakie znamy systemy liczbowe?

Pozycyjno-wagowe systemy liczbowe

2 3 4

$$2 \times 10^2 + 3 \times 10^1 + 3 \times 10^0$$

System dziesiętny

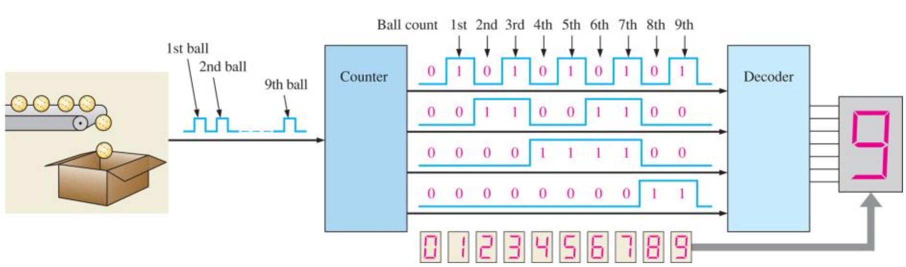
Symbole: 0, 1, …, 9
Podstawa: 10
Wagi: 10^5 10^4 10^3 10^2 10^1 10^0
Wagi dla ułamka dziesiętnego: 10^2 10^1 10^0, 10^{-1} 10^{-2} 10^{-3}

separator części ułamkowej
(w Polsce: przecinek)

TABLE 2–1

Decimal Number	Binary Number			
0				0
1				1
2			1	0
3			1	1
4		1	0	0
5		1	0	1
6		1	1	0
7		1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Rys 2-1 Ilustracja prostej aplikacji licznika binarnego



System binarny

Liczba binarna: 1 0 1

Jej wartość dziesiętna: $1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$

Symbole: 0, 1
Podstawa: 2
Wagi: $2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$
Wagi dla ułamka binarnego: $2^2\ 2^1\ 2^0,\ 2^{-1}\ 2^{-2}\ 2^{-3}$

separator części ułamkowej
(w Polsce: przecinek)

Tabela 2–2 Wagi w systemie binarnym

TABLE 2–2
Binary weights.

Positive Powers of Two (Whole Numbers)									Negative Powers of Two (Fractional Number)					
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
256	128	64	32	16	8	4	2	1	1/2 0.5	1/4 0.25	1/8 0.125	1/16 0.625	1/32 0.03125	1/64 0.015625

Konwersja liczby binarnej na dziesiętną

$1101101_2 = 2^6 + 2^5 + 2^3 + 2^2 + 2^0 = 64 + 32 + 8 + 4 + 1 = 109$

$0{,}1011_2 = 2^{-1} + 2^{-3} + 2^{-4} = 0{,}5 + 0{,}25 + 0{,}625 = 0{,}6875$

Konwersja liczby dziesiętnej na binarną

1. Przez rozbicie na sumę wag, np.:

   $9 = 8 + 1 = 2^3 + 2^0 = 1001_2$

   $0{,}625 = 0{,}5 + 0{,}125 = 2^{-1} + 2^{-3} = 0{,}101_2$

2. Metodą dzielenia (z resztą) przez 2 (dla liczby całkowitej)

12 / 2 = 6 reszta 0
6 / 2 = 3 reszta 0
3 / 2 = 1 reszta 1
1 / 2 = 0 reszta 1

1 1 0 0

LSB
(least significant bit)
bit o najmniejszej wadze

MSB
(most significant bit)
bit o największej wadze

reszta = ang. *reminder*

Konwersja liczby dziesiętnej na binarną, cd.

3. Metodą mnożenie przez dwa (dla ułamka binarnego)

MSB **LSB**

0, 0 1 0 1

0,3125 * 2 = 0,625 przeniesienie = 0
0,625 * 2 = 1,25 przeniesienie = 1
0,25 * 2 = 0,5 przeniesienie = 0
0,5 * 2 = 1,0 przeniesienie = 1

przeniesienie = ang. *carry*

>>> Operacje na bitach

Operacje na liczbach binarnych

takie jak: dodawanie, odejmowanie, mnożenie, dzielenie

są przeprowadzane w sposób analogiczny do operacji w systemie dziesiętnym

0 + 0 = ? 0 - 0 = ? 0 * 0 = ?

0 + 1 = ? 0 - 1 = ? 0 * 1 = ?

1 + 0 = ? 1 - 0 = ? 1 * 0 = ?

1 + 1 = ? 1 - 1 = ? 1 * 1 = ?

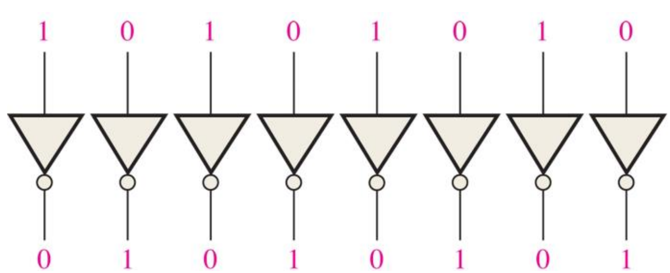
1 + 1 + 1 = ?

1 + 1 + 1 + 1 = ?

1010 + 101 =? 1010 * 101 =?

1010 - 101 =? 1010 / 100 =?

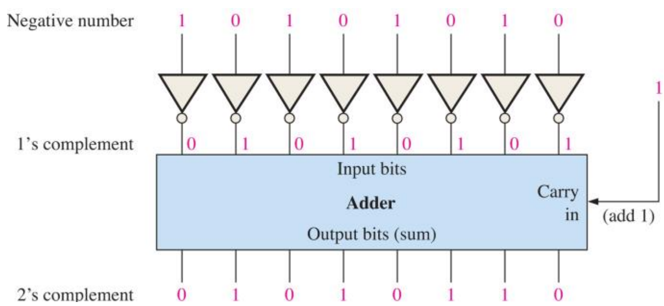
Rys 2-2 Przykład inwerterów używanych do uzyskania uzupełnienia do 1 dla liczby 8-bitowej.



Uzupełnieniem liczby binarnej do 1 nazywamy liczbę, której wszystkie bity mają wartości odwrotne.

(ang. 1's complement of the binary number)

Rys. 2-3 Przykład uzupełnienia liczby binarnej do 1 i do 2



Uzupełnieniem liczby binarnej do 2 nazywamy liczbę, która powstaje po zwiększeniu o jeden uzupełnienia liczby do 1.

(ang. 2's complement of the binary number)

Formy kodowania liczb binarnych ze znakiem (ang. signed binary numbers)

Pierwszy bit liczby (MSB) przeznaczamy na znak – jest to tzw. bit znaku (po którym rozpoznajemy, czy liczba jest ujemna (b_z=1), czy dodatnia (b_z=0))

$$b_Z \underbrace{b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0}$$

bit znaku
(ang. sign bit)

moduł liczby (wartość)
(ang. magnitude)

Kody znak-moduł (ZM)
- pierwszy bit liczby określa znak (0 dla liczb dodatnich,1 dla ujemnych), pozostałe to wartość liczby

Kod uzupełnień do 1 (U1)
- liczba ujemna jest uzupełnieniem do 1 liczby dodatniej

Kod uzupełnień do 2 (U2)
- liczba ujemna jest uzupełnieniem do 2 liczby dodatniej

Wartości dziesiętne liczb binarnych ze znakiem

ZM	+/-	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	1	0	0	1	0	1	0	1

10010101_{ZM} = - (16 + 4 + 1) = -21

U1	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	0	0	0	1	0	1	1	1

0001_0111_{U1} = 16 + 4 + 2 + 1 = 23

1110_1000_{U1} = (-128 + 64 + 32 + 8) **+ 1** = - 23

Wartości dziesiętne liczb binarnych ze znakiem, cd.

U2	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	0	0	0	1	0	1	1	1

0001_0111_{U2} = 16 + 4 + 2 + 1 = 23

1110_1001_{U2} = (-128 + 64 + 32 + 8 + 1) = - 23

Zaleta U2: liczby dodanie i ujemne przelicza się na wartość dziesiętną w ten sam sposób

>>> Po co nam liczby rzeczywiste? Jak je kodować?

Liczby zmiennoprzecinkowe (ang. floating point numbers)

32 bity		
Znak (sign **S**)	Wykładnik (cecha) (exponent **E**)	Mantysa (ułamek) (mantissa, fraction **F**)
1 bit	8 bitów	23 bity

- Wykładnik (cecha) jest spolaryzowany (zapisywany w kodzie z nadmiarem), tzn. zapisujemy wartość zwiększoną o 127 (**bias** = 127)

- Mantysa jest liczbą z zakresu [1,2), przy czym wiodącą jedynkę pomijamy – dla liczby 32-bitowej mantysa ma więc efektywnie 24 bity

Obliczanie wartości 32-bitowej liczby zmiennoprzecinkowej

$$FP = (-1)^S (1+F)(2^{E-bias})$$

32 bity		
Znak (sign **S**)	Wykładnik (cecha) (exponent **E**)	Mantysa (ułamek) (mantissa, fraction **F**)
1 bit	8 bitów	23 bity

$$FP = (-1)^S (1+F)(2^{E-bias})$$
$$bias = 127$$

Przykład zadania: Zakoduj podaną liczbę dziesiętną do formatu 32-bitowej liczby zmiennoprzecinkowej

$3{,}248 \times 10^4 = 32'480 =$

$111\ 1110\ 1110\ 0000_2 =$

$1{,}11\ 1110\ 1110\ 0000_2 \times 2^{14}$

S = 0 (liczba jest dodatnia)

$E = 14 + 127 = 1000\ 1101_2$

$F = 11\ 1110\ 1110\ 0000_2$

0	**1000 1101**	**111 1101 1100 0000 0000 0000**

Przykład zadania: oblicz wartość binarną i dziesiętną liczby zmienno przecinkowej:
0 1001_1000 100_0010_0010_0011_1000_0000

Przykład zadania: zapisz w postaci 32-bitowej liczby zmiennoprzecinkowej liczbę dziesiętną 132,6125.

Standard dla operacji na liczbach zmiennoprzecinkowych

IEEE Std 754^{TM}-2008 - IEEE Standard for Floating-Point Arithmetic

parametr	binary16	binary32	binary64	binary128
całkowita liczba bitów	16	32	64	128
bit znaku (S)	1	1	1	1
liczba bitów wykładnika (E)	5	8	11	15
liczba bitów mantysy (F)	10	23	52	112
polaryzacja wykładnika (bias)	15	127	1023	16383

Prosto opisane: http://eduinf.waw.pl/inf/alg/006_bin/0022.php

Kodowanie liczb zmiennoprzecinkowych wg IEEE Std. 754

32 bity			
Znak (sign **S**)	Wykładnik (cecha) (exponent **E**)	Mantysa (ułamek) (mantissa, fraction **F**)	Wartość / znaczenie
1 bit	8 bitów	23 bity	
S	1111_1111	F > 0	qNaN | sNaN*
S	1111_1111	F = 0	$(-1)^S \times (+\infty)$
S	$1 \le E \le 2^8 - 2$	F	$(-1)^S \times (1+F) \times 2^{(E-bias)}$
S	0000_0000	F > 0	$(-1)^S \times (0+F) \times 2^{(E-bias)}$ liczba zdenormalizowana (ang. subnormal number)
S	0000_0000	F = 0	$(-1)^S \times (+0)$

* quiet | signaling Not-A-Number
qNaN ma MSB mantysy równy 1, np. 0 1111_1111 1010_1000_...
sNaN ma MSB mantysy równy 0, np. 1 1111_1111 0010_0100_...
bit znaku jest ignorowany

>>> Jakie dodajemy liczby binarne ze znakiem?

Operacje arytmetyczne na liczbach ze znakiem

Dodawanie; suma = dodajna + dodajnik (składniki)
Addition; sum = addend + augend

```
   7        0000_0111
+  4      + 0000_0100
  11        0000_1011
```

dodanie dwóch liczb dodatnich daje wynik dodatni

```
  15        0000_1111
+ -6      + 1111_1010
   9      1 0000_1001
```

dodanie mniejszej liczby ujemnej daje wynik dodatni
(przeniesienie ignorujemy)

```
  16        0001_0000
+-24      + 1110_1010
  -8        1111_1000
```

dodanie większej liczby ujemnej daje wynik ujemny zakodowany w U2

Operacje arytmetyczne na liczbach ze znakiem, cd.

```
  -5        1111_1011
+ -9      + 1111_0111
 -14      1 1111_0010
```

dodanie dwóch liczb ujemnych daje wynik ujemny
(przeniesienie ignorujemy)

```
 125        0111_1101
+ 58      + 0011_1010
 183        1011_0111
```

UWAGA:
przekroczenie zakresu
(ang. overflow) – rozpoznawane
przez niepoprawny bit znaku;
wartość jest również niepoprawna

Przykładowe zadanie. Wykonaj działania na liczbach całkowitych ze znakiem:

0011_0011 + 1011_1111 = ?

1011_1111 + 0110_0011 = ?

0011_0011 + 1011_1111 + 0110_0011 = ?

Operacje arytmetyczne na liczbach ze znakiem, cd.

Odejmowanie; różnica = odjemna – odjemnik
Substraction; difference = minuend - substrahend

Odejmowanie liczb binarnych ze znakiem polega na dodaniu liczby przeciwnej w kodzie U2.

Dlatego tak lubimy U2. 

Wykonaj działanie:

0100_0111 – 0101_1000 = ?

Kiedy przy odejmowaniu wystąpi przekroczenia zakresu?

UWAGA: znak podkreślenia jest stosowany wyłącznie w celu poprawienia czytelności liczb binarnych! Nie jest to żaden standard, choć niektóre języki opisu sprzętu HDL pozwalają na taki zapis (np. Verilog)

>>> Jakie znamy metody mnożenia?

Operacje arytmetyczne na liczbach ze znakiem, cd.

Mnożenie; iloczyn = mnożna × mnożnik (czynniki)
Multiplication; product = multiplicand × multiplier

1. Metoda bezpośredniego dodawania (ang. direct_addition)

   Iloczyn powstaje poprzez dodanie pierwszego czynnika odpowiednią liczbę razy

2. Metoda iloczynów częściowych (ang. partial product)

```
      239
x     123
---------
      717
   + 478
  + 239
---------
    29397
```

Metoda iloczynów częściowych (U2)

$$\begin{array}{rr} & 01010011_{U2} \\ \times & 11000101_{U2} \\ \hline \end{array}$$

1. Określ znaki czynników i iloczynu
2. Liczby ujemne przekonwertuj na dodatnie
3. Wykonaj mnożenie
4. Jeżeli znak iloczynu ma być ujemny, przekonwertuj wynik na liczbę przeciwną (liczba bitów wyniku jest za zwyczaj dwa razy większa od liczby bitów czynników)

01010011_{U2} x 11000101_{U2}

1. Znaki czynników różne -> iloczyn ujemny

2. 11000101 -> 00111011

3. Obliczenia

```
      1010011
   x  0111011
   ----------
      1010011    #1 iloczyn częściowy
   + 1010011     #2 iloczyn częściowy
   ----------
     11111001    suma #1 i #2
  + 0000000      #3 iloczyn częściowy
   ----------
     11111001    suma #1,#2, #3
 + 1010011       #4 iloczyn częściowy
 ..................
 -------------
 1001100100001    iloczyn
```

4. 0001 0011 0010 0001 -> 1110 1100 1101 1111

Podobne zadanie może być na sprawdzianie wstępnym do laboratorium i egzaminie.

Metoda mnożenia rosyjskich chłopów

Wynik przepołowienia pierwszego czynnika	Podwajane sumy drugiego czynnika	Wielokrotności drugiego czynnika
57	**384**	$= 1 \cdot 384$
28	~~768~~	$= 2 \cdot 384$
14	~~1536~~	$= 4 \cdot 384$
7	3072	$= 8 \cdot 384$
3	6144	$= 16 \cdot 384$
1	12288	$= 32 \cdot 384$
	21888	

Opisany w tym punkcie algorytm mnożenia dwóch liczb jest znany pod nazwą metody rosyjskich chłopów, gdyż odwiedzający Rosję w XIX wieku spotykali się tam z jego powszechnym stosowaniem. Znany był już jednak egipskim matematykom 1800 lat p.n.e. Ciekawe jest to, że w działaniach w nim wykonywanych korzysta się (niejawnie jednak) z przedstawienia jednej z liczb w systemie binarnym. Zatem idea takiego rozkładu liczby w obliczeniach pojawiła się w sposób naturalny na długo przed wykorzystaniem jej w arytmetyce komputerowej.

http://mmsyslo.pl/Materialy/Ksiazki-i-podreczniki/Ksiazki/Ksiazka-Piramidy-szyszki-i

Operacje arytmetyczne na liczbach ze znakiem, cd.

Dzielenie; iloraz = dzielna / dzielnik
Division; quotient = dividend / divisor

Dzielenie może być w najprostszych algorytmach realizowane jako wielokrotne odejmowanie dzielnika o dzielnej (i sprawdzanie, czy wynik jest większy od zera) lub jako odejmowanie i przesuwanie dzielnika (jak w dzieleniu „pod kreskę”).

Kroki przy dzieleniu liczb ze znakiem kodowanych w U2 są analogiczne do stosowych przy mnożeniu.

Dzielenie jest najbardziej złożonym z podstawowych działań arytmetycznych

>>> System szesnastkowy.

TABLE 2–3

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

System szesnastkowy (heksadecymalny)

Liczba szesnastkowa: B 9

Jej wartość dziesiętna: $11 \times 16^1 + 9 \times 16^0 = 185$

Symbole: 0, 1, …, 9, A, B, C, D, E, F

Podstawa: 16

Wagi: $16^5\ 16^4\ 16^3\ 16^2\ 16^1\ 16^0$

Wagi dla ułamka: $16^2\ 16^1\ 16^0, 16^{-1}\ 16^{-2}\ 16^{-3}$

separator części ułamkowej
(w Polsce: przecinek)

Konwersja liczby binarnej na szesnastkową i odwrotnie

$$\underbrace{1100}_{\mathrm{C}}\underbrace{1010}_{\mathrm{A}}\underbrace{0101}_{5}\underbrace{0111}_{7}{}_{2} = \mathrm{CA57}_{16}$$

Tradycyjny zapis w językach programowania: 0xCA57

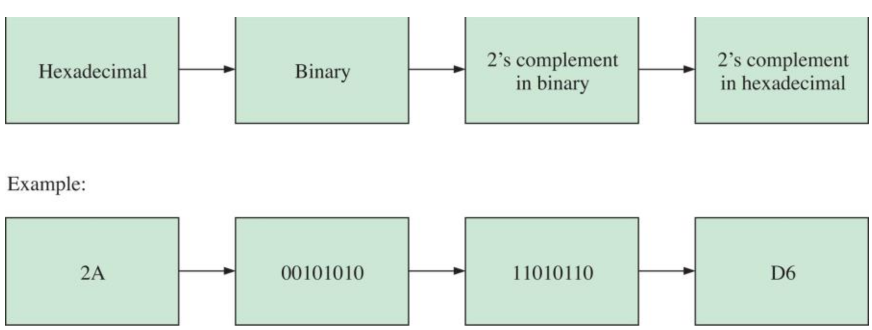
Uzupełnienie do dwóch liczby szesnastkowej:
HEX -> BIN -> U2 -> HEX

Przykładowe zadania:

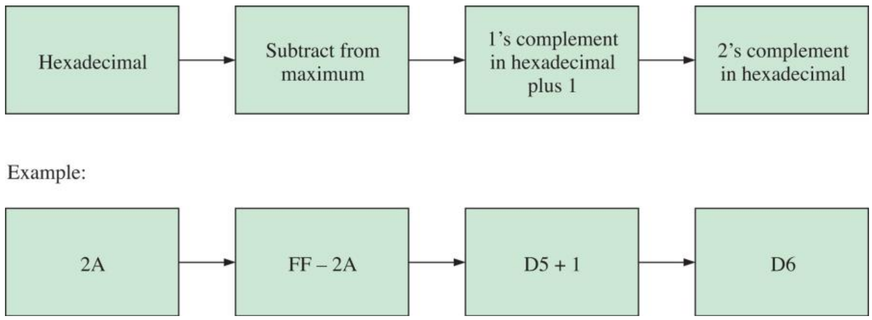
Przekształć liczbę szesnastkową 0xCA58 na system dziesiątkowy i binarny.

Zapisz 16-bitową liczbę -124 w systemie szesnastkowym w kodzie uzupełnień do 2.

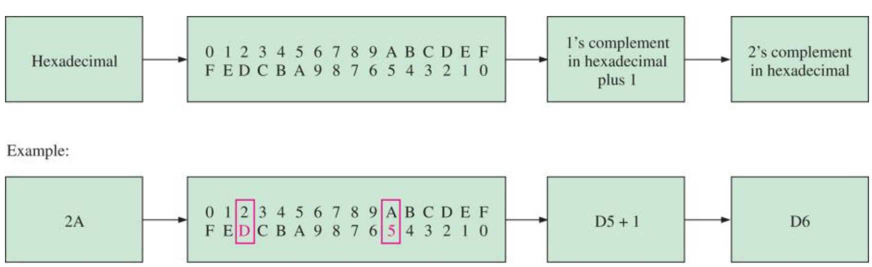
Rys. 2-4 Uzupełnienie do 2 liczby szesnastkowej, metoda pierwsza.



Rys. 2-5 Uzupełnienie do 2 liczby szesnastkowej, metoda druga.



Rys. 2-6 Uzupełnienie do 2 liczby szesnastkowej, metoda trzecia.



System ósemkowy (oktalny)

Liczba ósemkowa: $\boxed{7}\ \boxed{6}$

Jej wartość dziesiętna: $\boxed{7 \times 8^1} + \boxed{6 \times 8^0} = 62$

Symbole: 0, 1, …, 7
Podstawa: 8
Wagi: $8^5\ 8^4\ 8^3\ 8^2\ 8^1\ 8^0$
Wagi dla ułamka ósemkowego: $8^2\ 8^1\ 8^0,\ 8^{-1}\ 8^{-2}\ 8^{-3}$

separator części ułamkowej
(w Polsce: przecinek)

>>> Gdzie jest używany system ósemkowy?

TABLE 2–4

Octal/binary conversion.

Octal Digit	0	1	2	3	4	5	6	7
Binary	000	001	010	011	100	101	110	111

Kod BCD – system dziesiętny zakodowany dwójkowo (ang. Binary Coded Decimal)

TABLE 2–5

Decimal/BCD conversion.

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

$35 = 0011\_0101_{BCD}$ $98 = 1001\_1000_{BCD}$

Kody binarne od 1010 do 1111 - zabronione

Packed BCD – dwie cyfry w jednym bajcie (8 bitów)

Zadanie: zapisz w kodzie BCD liczbę dziesiętną 120

>>> Czy są kody inne niż pozycyjno-wagowe?

Kod Gray'a

TABLE 2–6

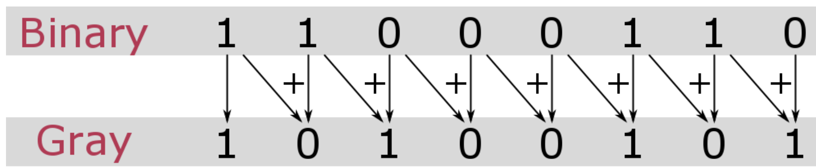
Four-bit Gray code.

Decimal	Binary	Gray Code	Decimal	Binary	Gray Code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Każda kolejna pozycja w kodzie Gray'a różni się od poprzedniej **tylko jednym bitem**

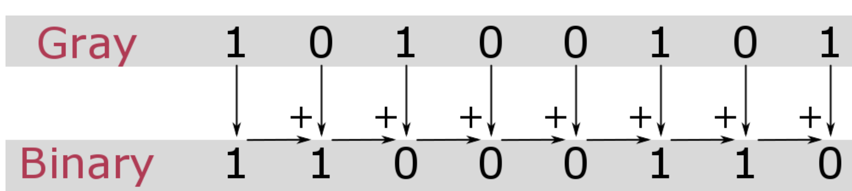
Kod **Gray'a nie jest kodem ważonym**, pozycje bitów nie mają odpowiadających im wag.

Konwersja liczby binarnej do kodu Gray'a



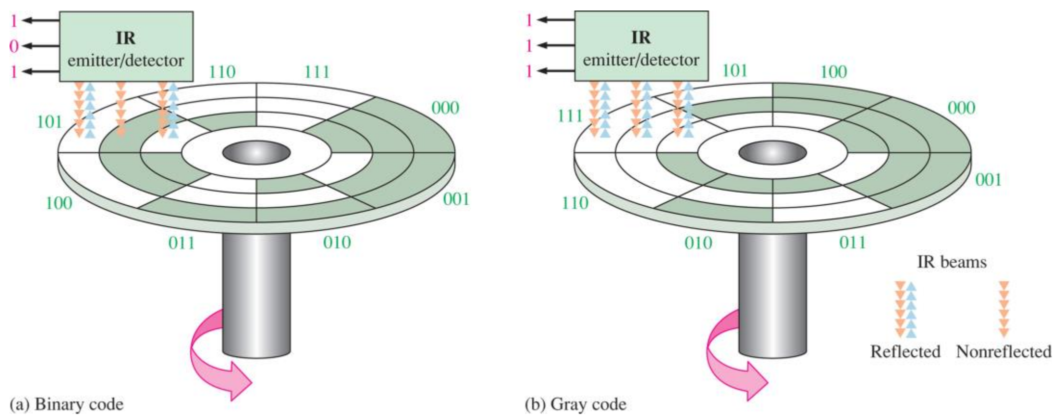
1. Przepisz MSB
2. Dodaj każdy bit liczby w postaci binarnej do swojego sąsiada. Zignoruj bity przeniesienia

Konwersja liczby w kodzie Gray'a do postaci binarnej



1. Przepisz MSB
2. Dodaj kolejny bit liczby w kodzie Gray'a do otrzymanego wcześniej wyniku. Zignoruj przeniesienia. Powtórz dla kolejnych bitów.

Rys. 2-7 Uproszczona ilustracja pokazująca, jak zastosowanie kodu Gray’a rozwiązuje problem błędów w enkoderach pozycyjnych. Koncepcja jest przedstawiona na przykładzie trzech bitów; większość enkoderów używa ponad 10 bitów w celu osiągnięcia lepszej rozdzielczości.



American Standard Code for Information Interchange (ASCII).

TABLE 2–7

American Standard Code for Information Interchange (ASCII).

Control Characters				Graphic Symbols											
Name	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex
NUL	0	0000000	00	space	32	0100000	20	@	64	1000000	40	‘	96	1100000	60
SOH	1	0000001	01	!	33	0100001	21	A	65	1000001	41	a	97	1100001	61
STX	2	0000010	02	”	34	0100010	22	B	66	1000010	42	b	98	1100010	62
ETX	3	0000011	03	#	35	0100011	23	C	67	1000011	43	c	99	1100011	63
EOT	4	0000100	04	$	36	0100100	24	D	68	1000100	44	d	100	1100100	64
ENQ	5	0000101	05	%	37	0100101	25	E	69	1000101	45	e	101	1100101	65
ACK	6	0000110	06	&	38	0100110	26	F	70	1000110	46	f	102	1100110	66
BEL	7	0000111	07	’	39	0100111	27	G	71	1000111	47	g	103	1100111	67
BS	8	0001000	08	(	40	0101000	28	H	72	1001000	48	h	104	1101000	68
HT	9	0001001	09	)	41	0101001	29	I	73	1001001	49	i	105	1101001	69
LF	10	0001010	0A	*	42	0101010	2A	J	74	1001010	4A	j	106	1101010	6A
VT	11	0001011	0B	+	43	0101011	2B	K	75	1001011	4B	k	107	1101011	6B
FF	12	0001100	0C	,	44	0101100	2C	L	76	1001100	4C	l	108	1101100	6C
CR	13	0001101	0D	–	45	0101101	2D	M	77	1001101	4D	m	109	1101101	6D
SO	14	0001110	0E	.	46	0101110	2E	N	78	1001110	4E	n	110	1101110	6E
SI	15	0001111	0F	/	47	0101111	2F	O	79	1001111	4F	o	111	1101111	6F
DLE	16	0010000	10	0	48	0110000	30	P	80	1010000	50	p	112	1110000	70
DC1	17	0010001	11	1	49	0110001	31	Q	81	1010001	51	q	113	1110001	71
DC2	18	0010010	12	2	50	0110010	32	R	82	1010010	52	r	114	1110010	72
DC3	19	0010011	13	3	51	0110011	33	S	83	1010011	53	s	115	1110011	73
DC4	20	0010100	14	4	52	0110100	34	T	84	1010100	54	t	116	1110100	74
NAK	21	0010101	15	5	53	0110101	35	U	85	1010101	55	u	117	1110101	75
SYN	22	0010110	16	6	54	0110110	36	V	86	1010110	56	v	118	1110110	76
ETB	23	0010111	17	7	55	0110111	37	W	87	1010111	57	w	119	1110111	77
CAN	24	0011000	18	8	56	0111000	38	X	88	1011000	58	x	120	1111000	78
EM	25	0011001	19	9	57	0111001	39	Y	89	1011001	59	y	121	1111001	79
SUB	26	0011010	1A	:	58	0111010	3A	Z	90	1011010	5A	z	122	1111010	7A
ESC	27	0011011	1B	;	59	0111011	3B	[	91	1011011	5B	{	123	1111011	7B
FS	28	0011100	1C	<	60	0111100	3C	\	92	1011100	5C	|	124	1111100	7C
GS	29	0011101	1D	=	61	0111101	3D	]	93	1011101	5D	}	125	1111101	7D
RS	30	0011110	1E	>	62	0111110	3E	^	94	1011110	5E	~	126	1111110	7E
US	31	0011111	1F	?	63	0111111	3F	_	95	1011111	5F	Del	127	1111111	7F

Kody detekcji błędów (ang. error codes)

TABLE 2–8

The BCD code with parity bits.

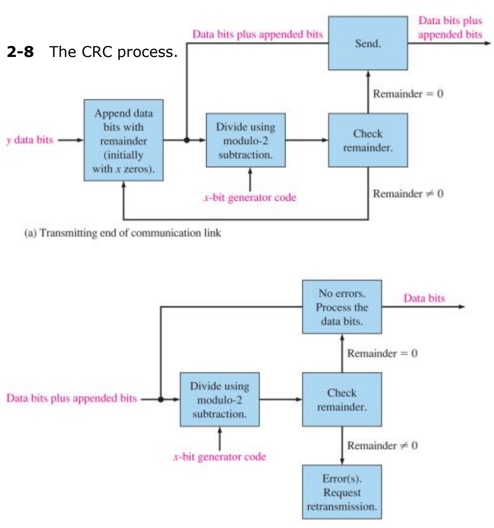
Even Parity		Odd Parity	
P	BCD	*P*	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

Kod BCD z bitami parzystości

Dodanie bitu parzystości umożliwia wykrycie pojedynczego błędu w bajcie (np. w podczas transmisji danych)

CRC – cykliczny kod nadmiarowy (ang. Cyclic Redundancy Check)

Rys. 2-8 The CRC process.



Umożliwia detekcję błędów wielobitowych w transmisji danych

Kod Hamminga – detekcja i korekcja błędów