

# Wprowadzenie do technologii cyfrowych

**Roman Simiński**

roman.siminski@us.edu.pl

roman@siminskionline.pl

**Cyfrowe reprezentacje**

**Liczby, i nie tylko one**



# System liczenia intuicyjnie stosowany przez większość z nas

- Na co dzień posługujemy się dziesiętnym, pozycyjnym systemem liczenia.
- Wykorzystujemy dziesięć cyfr: 0 1 2 3 4 5 6 7 8 9
- Z nich budujemy liczby.
- Każda pozycja liczby ma wagę, będącą potęgą wartości 10.

$$\begin{array}{r} 10^2 \quad 10^1 \quad 10^0 \\ 100 \quad 10 \quad 1 \\ \mathbf{1} \quad \mathbf{2} \quad \mathbf{5} \\ \begin{array}{r} \longrightarrow 5 \cdot 1 = 5 \\ \longrightarrow 2 \cdot 10 = 20 \\ \longrightarrow 1 \cdot 100 = 100 \\ + \\ \hline \mathbf{125} \end{array} \end{array}$$

## Ogólnie – pozycyjny zapis liczb

- System pozycyjny ma pewną wybraną wartość nazywaną *podstawą systemu*, oznaczaną zwykle symbolem  $p$ .
- Do zapisu liczb stosuje się cyfry.
- Dla systemu o podstawie  $p$  występować będzie  $p$  cyfr:  $0, 1, \dots, p-1$ .
- Cyfry występują na określonych *pozycjach*, każda z pozycji ma przypisaną *wagę*.
- Wagi kolejnych pozycji to kolejne potęgi podstawy  $p$ .
- Cyfra zapisana na konkretnej pozycji stanowi jej mnożnik – mówi nam ile razy waga danej pozycji uczestniczy w wartości liczby.
- Wartość liczby wyznacza się sumując iloczyny cyfry na danej pozycji przez wagę pozycji.

$$25642_{(10)} = 2 \cdot 10^4 + 5 \cdot 10^3 + 6 \cdot 10^2 + 4 \cdot 10^1 + 2 \cdot 10^0$$

# Ogólnie – pozycyjny zapis liczb

- Dla systemu o podstawie  $p > 1$  mamy  $p$  cyfr, oznaczonych literą  $c$ .
- Załóżmy, że używać będziemy  $n$  cyfr, inaczej mówiąc liczba będzie miała  $n$  pozycji:

$$c_{n-1} c_{n-2} \dots c_2 c_1 c_0$$

- Wagi poszczególnych pozycji:

$$p^{n-1} p^{n-2} \dots p^2 p^1 p^0$$

- Wartość liczby:

$$c_{n-1} c_{n-2} \dots c_2 c_1 c_0 = c_0 \cdot p^0 + c_1 \cdot p^1 + c_2 \cdot p^2 + \dots + c_{n-2} \cdot p^{n-2} + c_{n-1} \cdot p^{n-1} = \sum_{i=0}^{n-1} c_i \cdot p^i$$

## Ciekawostka – schemat Hornera

- Potęgowanie jest sensie obliczeniowym jest czasochłonne.
- Lepsze jest dodawanie i mnożenie.
- Formułę obliczania wartości liczby można sprowadzić do schematu Hornera:

$$C_{n-1} C_{n-2} \dots C_2 C_2 C_0 = c_0 \cdot p^0 + c_1 \cdot p^1 + c_2 \cdot p^2 + \dots + c_{n-3} \cdot p^{n-3} + c_{n-2} \cdot p^{n-2} + c_{n-1} \cdot p^{n-1}$$

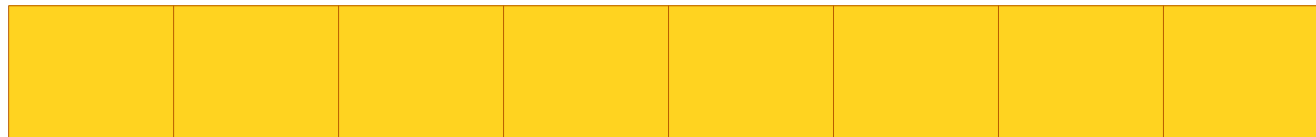


$$C_{n-1} C_{n-2} \dots C_2 C_2 C_0 = c_0 + p \cdot (c_1 + p \cdot (c_2 + \dots + p \cdot (c_{n-3} + p \cdot (c_{n-2} + c_{n-1} \cdot p)) \dots))$$

# System pozycyjny o podstawie 2 – system binarny

- Niech podstawą liczenia będzie **2**, taki system liczenia nazywamy binarnym.
- Ile będzie cyfr? Dwie. Jakie to będą cyfry? **0 1**
- Z nich budujemy liczby binarne, założmy, że używamy liczb mających **osiem cyfr**.

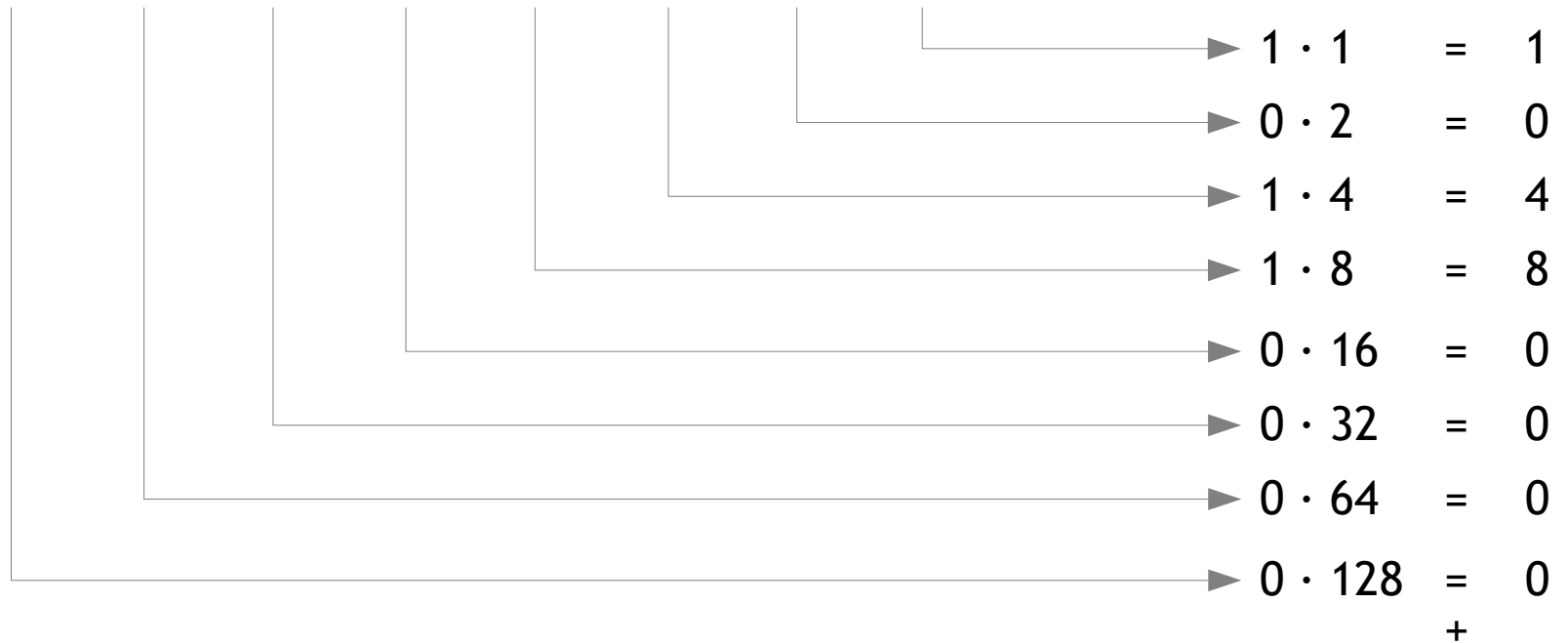
Nr cyfry:	7	6	5	4	3	2	1	0
Potęga 2:	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
Waga:	128	64	32	16	8	4	2	1



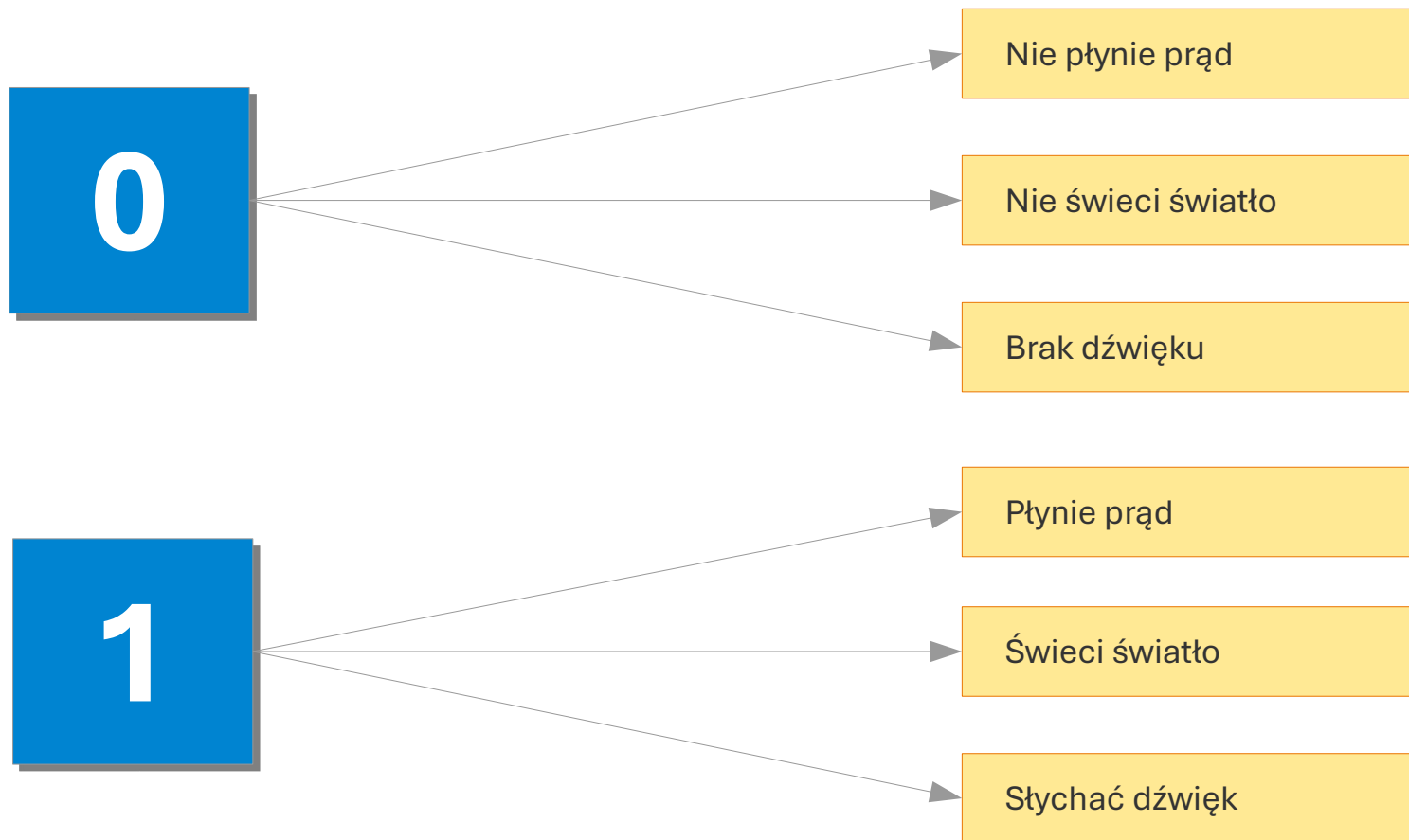
# Jak liczymy w systemie binarnym?

Nr bitu:	7	6	5	4	3	2	1	0
Potęga 2:	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
Waga:	128	64	32	16	8	4	2	1

0	0	0	0	1	1	0	1
---	---	---	---	---	---	---	---



# Cyfry binarne mogą być przesyłane poprzez różne media transmisyjne



- Wartości 0 i 1 często określa się odpowiednio stanem wysokim i stanem niskim, co jest powiązane z fizyczną reprezentacją.
- Cyfry binarne nazywamy bitami:

**bit = binary digit**

- Bit to zatem wielkość, który może wystąpić tylko w dwóch stanach, oznaczanych symbolami 0 i 1. Bit to jednostka miary dla informacji.
- O tym jaka wielkość fizyczna reprezentuje bit w urządzeniach fizycznych oraz jakie są wartości tej wielkości decydują ustalone konwencje, założenia konstrukcyjne i konkretna realizacja.
- Słowo „bit” zaproponował John W. Tukey z Bell Labs w 1947 roku, rozpowszechniło się wraz z publikacjami Claude E. Shannona (1948, „A Mathematical Theory of Communication”).

- Z bitów tworzymy **słowa**.
- Liczba bitów tworzących słowo jest zależna od kontekstu, potrzeb i zastosowania.
- We „wczesnej” informatyce słowa bywały 18, 24, 36 a nawet 60 bitowe.
- Wraz z pojawieniem się **IBM 360** zaczęto powszechnie posługiwać się ósemkami bitów, zwanych *oktetami*.
- Aktualnie oktet bitów nazywa się **bajtem**.
- Uwaga – historycznie nie zawsze bajt liczył osiem bitów, gdy nazwa bajt pojawiła się w 1956 roku, bajt był 4 bitowy.
- Słowo **bajt** (ang. byte) wywodzi się angielskiego *bite* (kęs), ponieważ jest to najmniejsza porcja danych, którą komputer może „ugryźć” za jednym razem (czyli pobrać, zapisać, przetworzyć).
- Aktualnie bajt oznacza *najmniejszą adresowalną jednostkę informacji zapisywanej w pamięci komputera*.

- Z bitów tworzymy **słowa**.
- Liczba bitów tworzących słowo jest zależna od kontekstu, potrzeb i zastosowania.
- W „wczesnej” informatyce słowa bywały 18, 24, 36 a nawet 60 bitowe.

• W

zw **Ważne**

- Ak Bajt to ósemka bitów, bajt zwykle oznacza się literą **B**, bit literą **b**

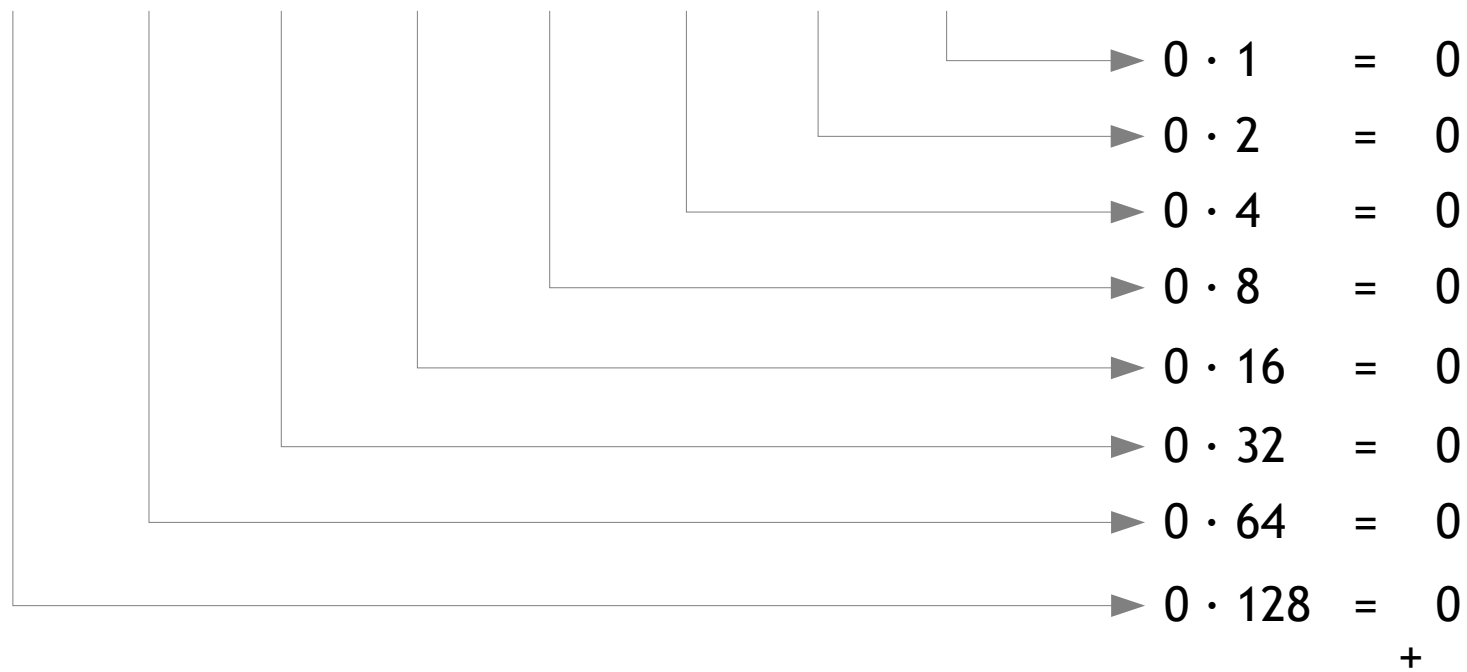
• Uw

ro

- Słowo **bajt** (ang. byte) wywodzi się angielskiego *bite* (kęs), ponieważ jest to najmniejsza porcja danych, którą komputer może „ugryźć” za jednym razem (czyli pobrać, zapisać, przetworzyć).
- Aktualnie bajt oznacza *najmniejszą adresowalną jednostkę informacji zapisywanej w pamięci komputera*.

# Najmniejsza nieujemna liczba zapisywana na jednym bajcie

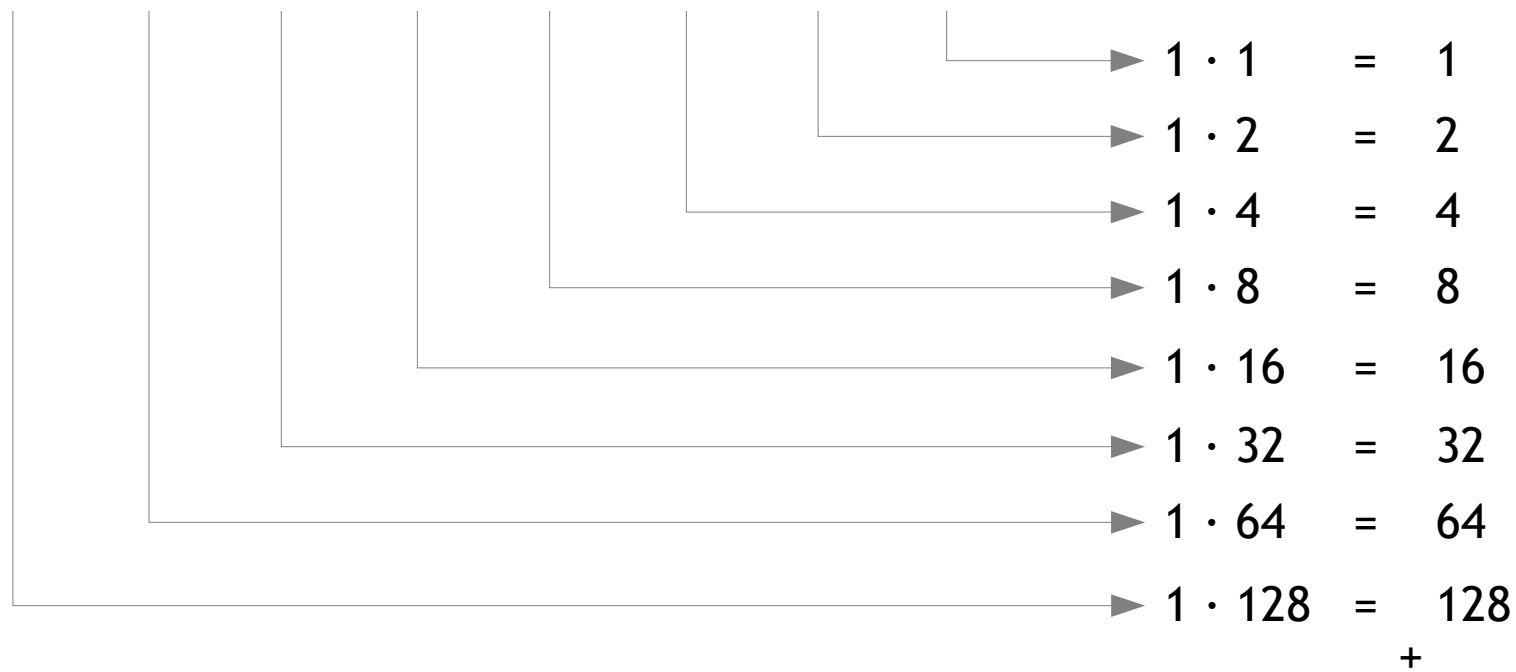
Nr bitu:	7	6	5	4	3	2	1	0
Potęga 2:	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
Waga:	128	64	32	16	8	4	2	1



0

# Największa nieujemna liczba zapisywana na jednym bajcie

Nr bitu:	7	6	5	4	3	2	1	0
Potęga 2:	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
Waga:	128	64	32	16	8	4	2	1



255

# Do czego tak naprawdę służą liczby w technice cyfrowej?

$$13_{(10)} = 1101_{(2)}$$

0	0	0	0	1	1	0	1
---	---	---	---	---	---	---	---

Liczby służą do  
reprezentowania liczb

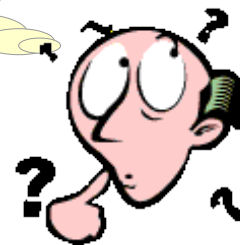
# Do czego tak naprawdę służą liczby w technice cyfrowej?

$$13_{(10)} = 1101_{(2)}$$

0	0	0	0	1	1	0	1
---	---	---	---	---	---	---	---

Liczby służą do  
reprezentowania liczb

Trochę się to  
wydaje głupie, ale...



# Liczby służą do reprezentowania informacji numerycznych różnego rodzaju

$$13_{(10)} = 1101_{(2)}$$

0	0	0	0	1	1	0	1
---	---	---	---	---	---	---	---

Liczby służą do  
reprezentowania liczb

- Liczby całkowite bez znaku
- Liczby całkowite ze znakiem
- Liczby rzeczywiste
- Liczby do obliczeń walutowych
- Liczby zespolone ...

# Liczby służą do zakodowania informacji o charakterze nienumerycznym

$$13_{(10)} = 1101_{(2)}$$

0 0 0 0 1 1 0 1

Liczby służą do  
reprezentowania liczb

Liczby całkowite bez znaku  
Liczby całkowite ze znakiem  
Liczby rzeczywiste  
Liczby do obliczeń walutowych  
Liczby zespolone ...

Liczby służą do reprezentowania  
informacji nienumerycznych

Znaki  
Teksty  
Kolory → grafika i wideo  
Dźwięk  
... i wszystko co analogowe, a ma być komputerowe

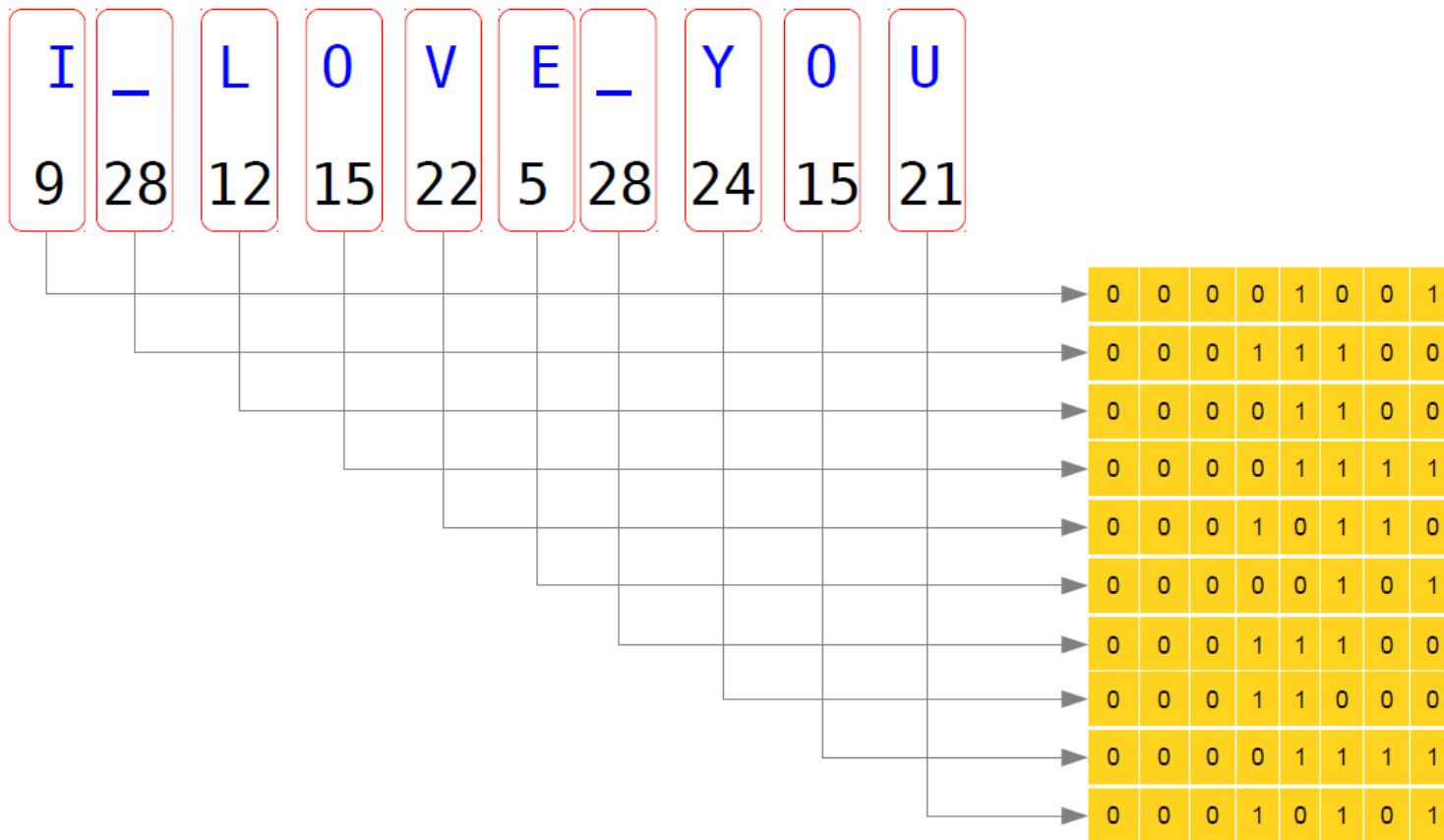
# Ten przykład już znamy...

A	—	1	N	—	14	.	—	27
B	—	2	O	—	15	_	—	28
C	—	3	P	—	16			
D	—	4	Q	—	17			
E	—	5	R	—	18			
F	—	6	S	—	19			
G	—	7	T	—	20			
H	—	8	U	—	21			
I	—	9	V	—	22			
J	—	10	W	—	23			
K	—	11	X	—	24			
L	—	12	Y	—	25			
M	—	13	Z	—	26			

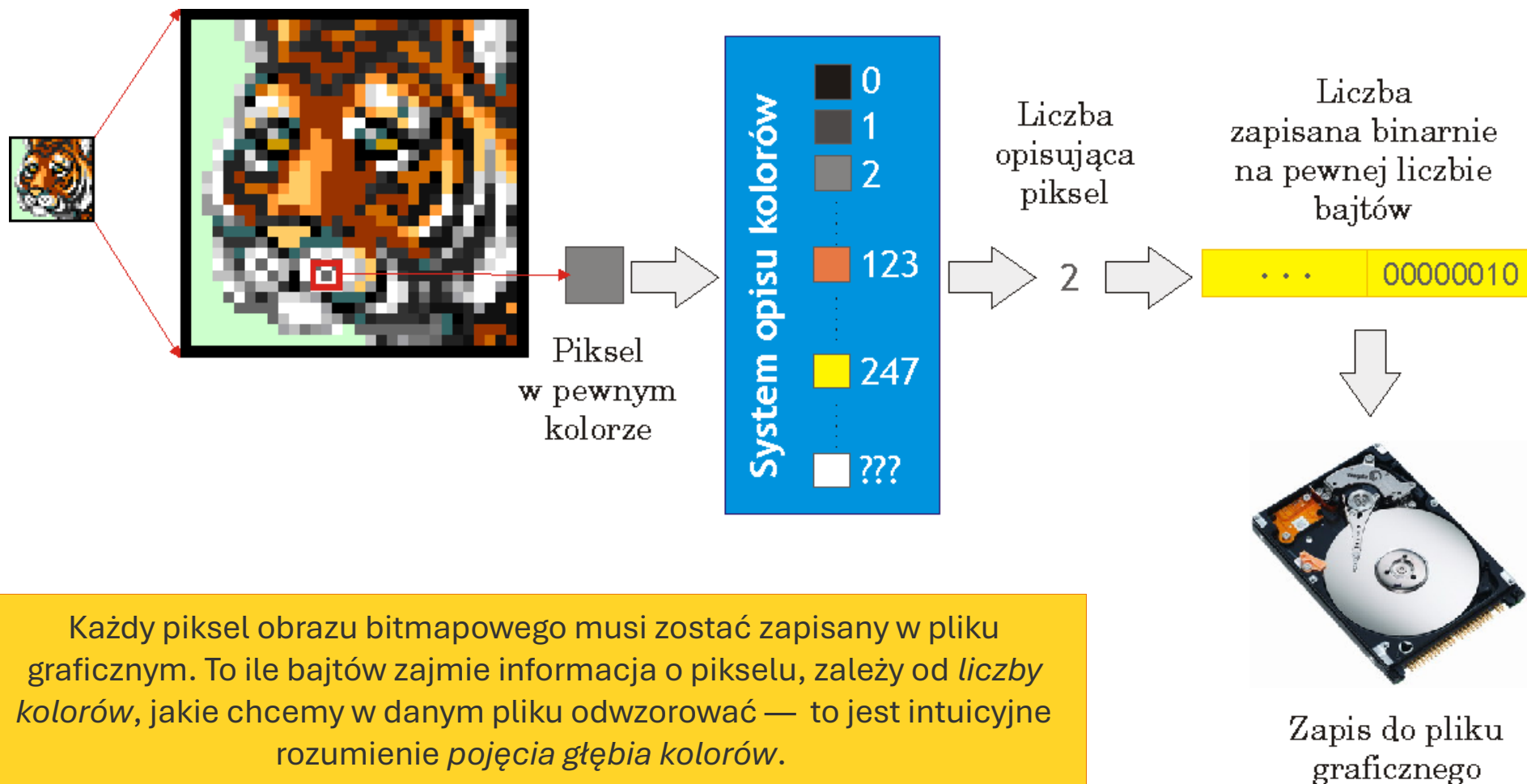


I	_	L	O	V	E	_	Y	O	U
9	28	12	15	22	5	28	24	15	21

# Zakodowany liczbowo komunikat w reprezentacji binarnej



# Kodowanie kolorów pikseli w grafice rastrowej

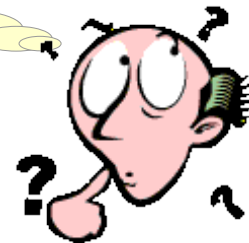


# Jaki jest potencjał bajtu jako środka kodowania informacji?

- Jeden bajt może reprezentować liczby z zakresu od **0** do **255**.
- Razem tych liczb jest **256**.
- Matematyczne uzasadnienie: *tylko jest ośmioelementowych wariacji z powtórzeniami dla zbioru dwuelementowego:  $2^8$ .*
- Jeżeli chcemy wykorzystać bajtową liczbę binarną do *ponumerowania* czegoś, to jesteśmy w stanie ponumerować **256** elementów, przydzielając im numery od **0** do **255**.



Na jednym bajcie ponumeruję  
256 foteli w kinie, a co gdy  
ich będzie więcej?



# Gdy bajt to za mało...

Rozmiar tabliczki = 1 bajt:

10101010



Maksymalnie 256

Rozmiar tabliczki = 2 bajty:

1010101010101010



Maksymalnie 65536

Rozmiar tabliczki = 4 bajty:

10



Maksymalnie 4294967296

# Gdy bajt to za mało...

Rozmiar tabliczki = 1 bajt:

10101010



Maksymalnie 256

Jeżeli chcę ponumerować **coś**, to muszę wybrać liczbę binarną *wystarczająco dużą*, aby mogła *pomieścić największą liczbę z zakresu*, który interesuje.

Maksymalnie 65536

Rozmiar tabliczki = 4 bajty:

10



Maksymalnie 4294967296

# Liczba symboli/liczb binarnych zależy od liczby dostępnych bitów

Liczba bitów	Liczba symboli	Potęga podstawy 2
1	2	$2^1$
2	4	$2^2$
3	8	$2^3$
4	16	$2^4$
5	32	$2^5$
6	64	$2^6$
7	128	$2^7$
8	256	$2^8$
...	...	...
$n$	$2^n$	$2^n$

# Pewna ważna prawidłowość

Liczba bitów	Liczba symboli	Zakres liczb
1	2	0 .. 1
2	4	0 .. 3
3	8	0 .. 7
4	16	0 .. 15
5	32	0 .. 31
6	64	0 .. 63
7	128	0 .. 127
8	256	0 .. 255
...	...	...
$n$	$2^n$	0 .. $2^n - 1$

# Pewna ważna prawidłowość

Liczba bitów

Liczba symboli

Zakres liczb

## Ważne

W naturalnym kodzie binarnym, dla liczby składającej się z  $n$  bitów można zapisać liczby z zakresu:

$$0 .. 2^n - 1$$

0

255

0 .. 255

...

...

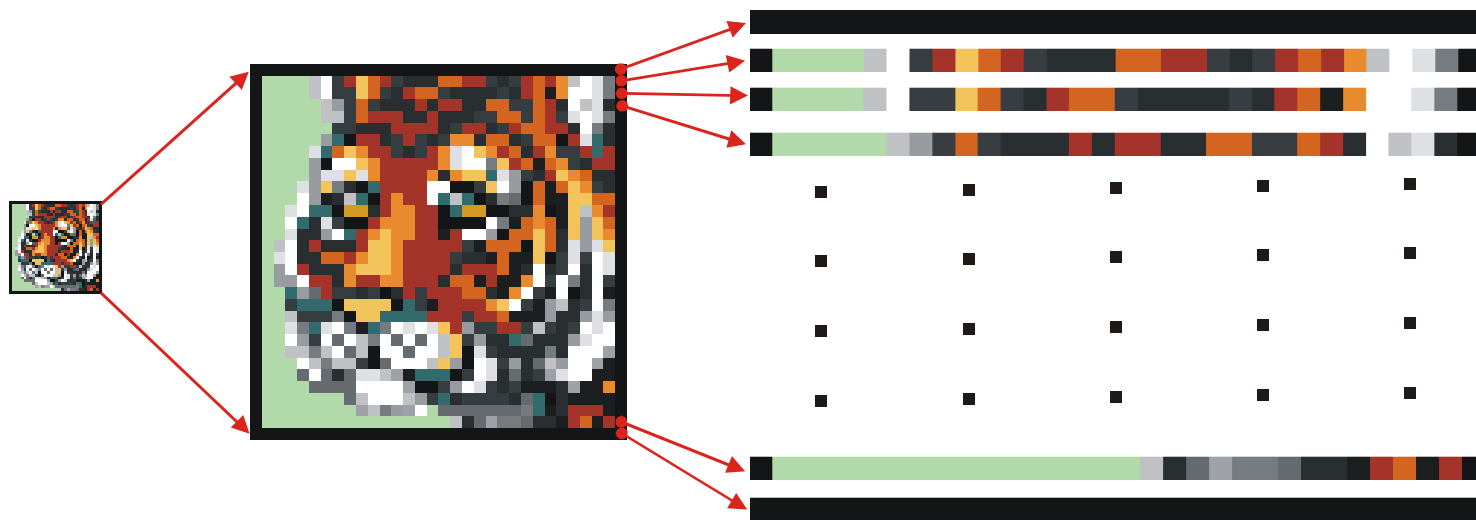
...

$n$

$2^n$

$0 .. 2^n - 1$

# Wróćmy jeszcze do przykładu grafiki...



Ciąg pikseli



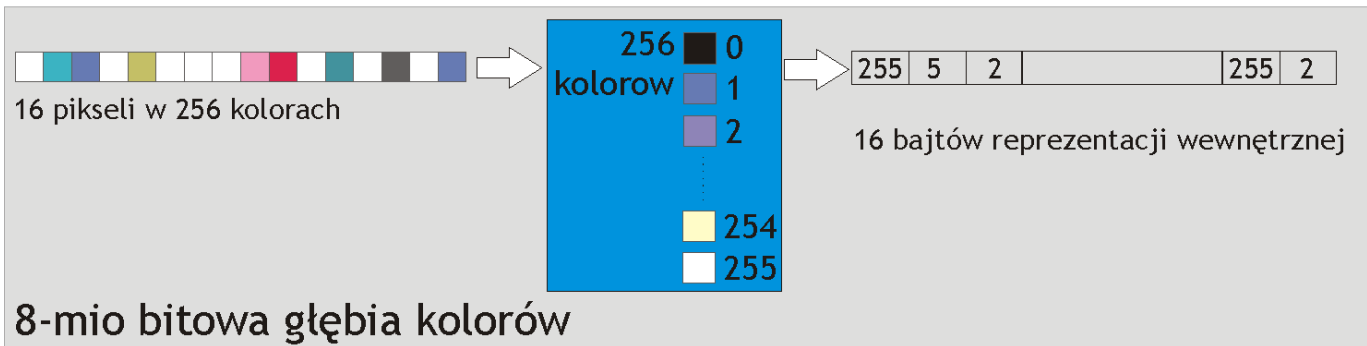
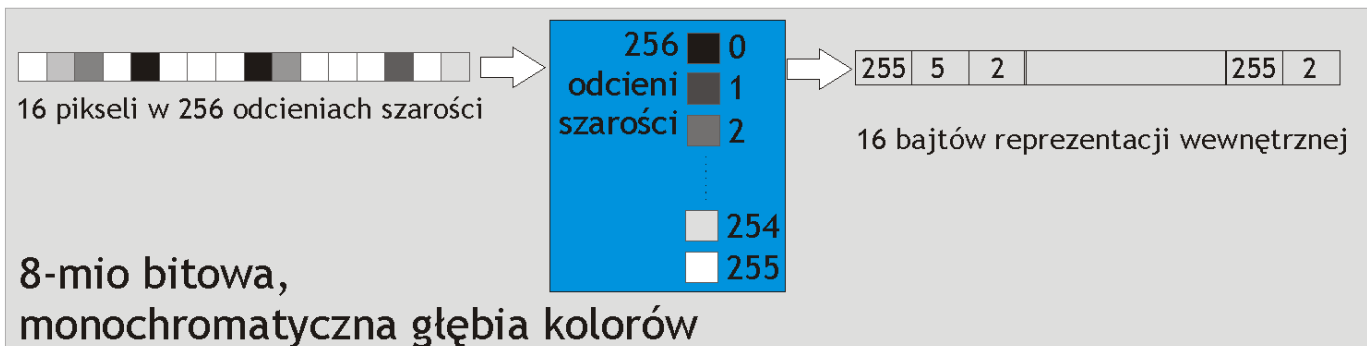
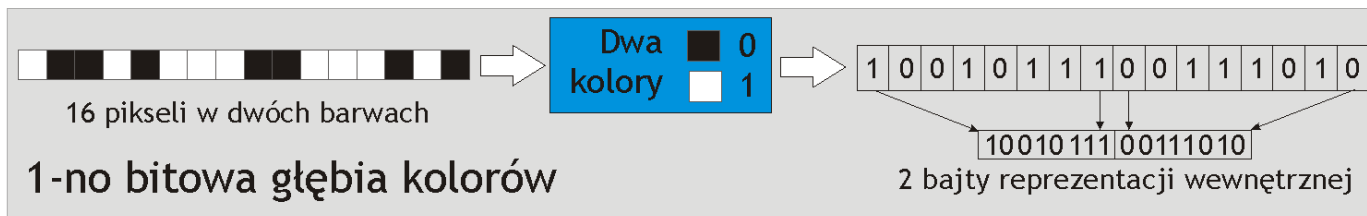
**System opisu kolorów**



Liczby kodujące  
kolory

0	2	2	2	2	14	15	.....	15	14	7	0
---	---	---	---	---	----	----	-------	----	----	---	---

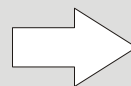
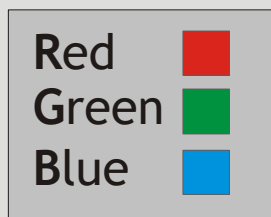
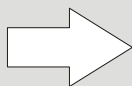
# Różne podejścia do numerowania kolorów



# Odwzorowanie dużej liczby kolorów



3 piksele, każdy  
w jednym z 16.7 mln kolorów

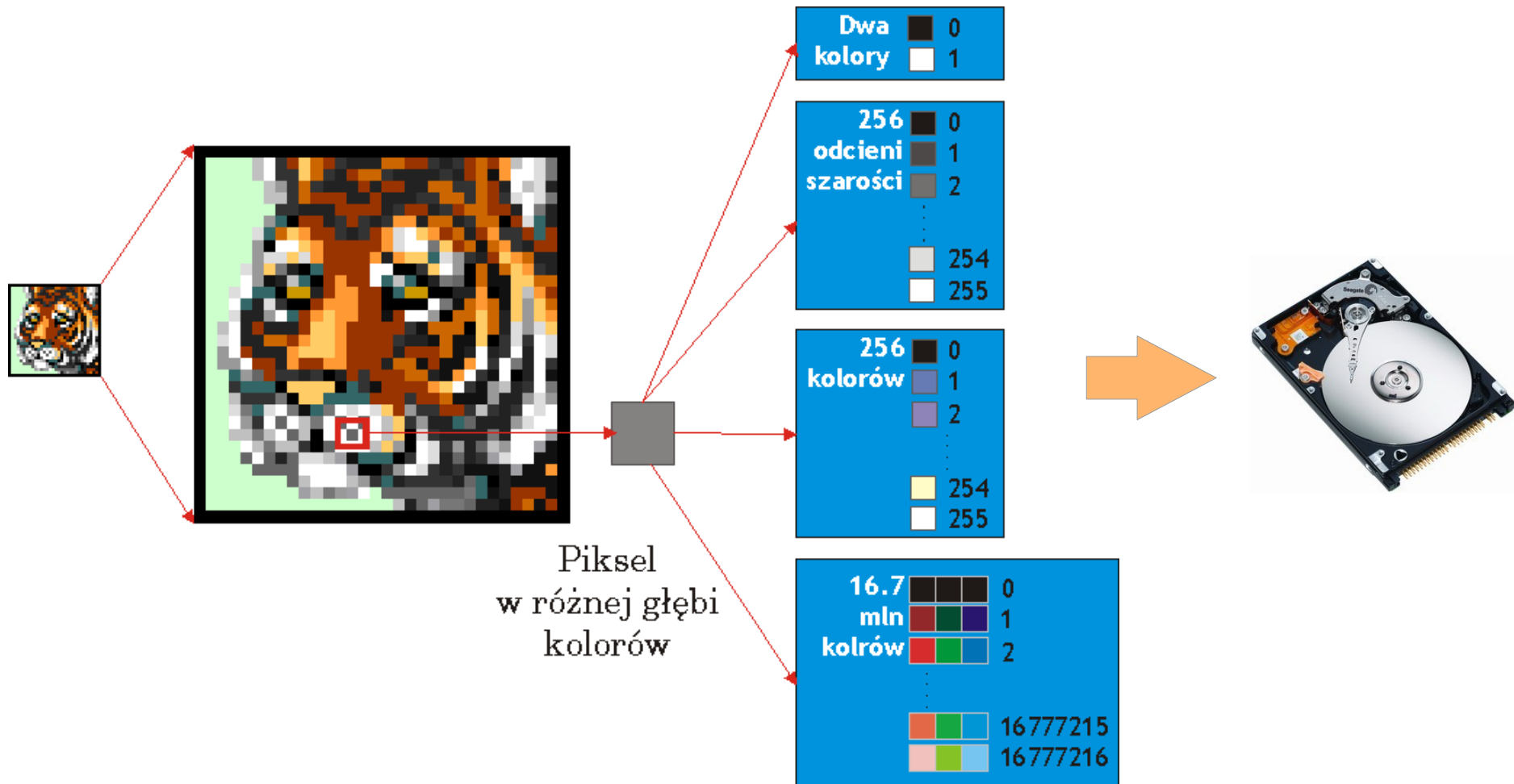


68	255	231
146	252	120
132	200	68

24-ro bitowa głębokość kolorów  
tzw. True color

9 bajtów reprezentacji wewnętrznej  
przy 8-miu bitach na każdy kanał  
pojedynczego piksela

# Różne systemy numerowania, różne zapotrzebowanie na zasoby komputera



# Różne systemy numerowania, różne zapotrzebowanie na zasoby komputera

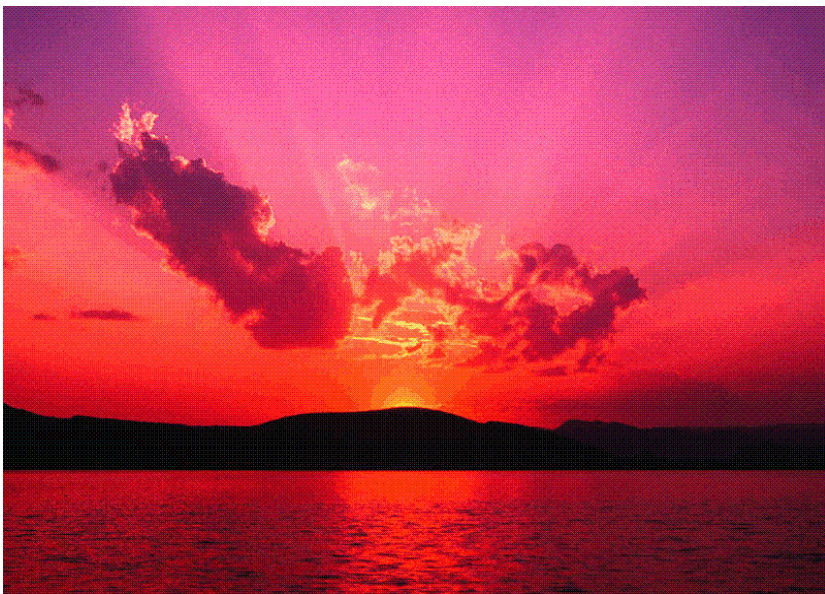


Kolor 24 bitowy  
rozmiar pliku bez kompresji 1,37 MB

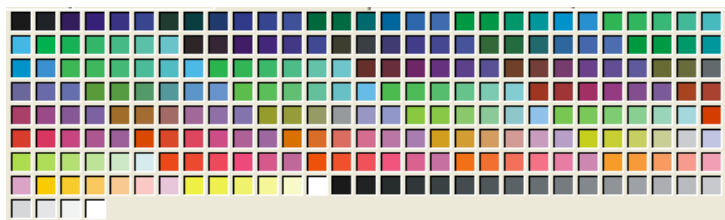


Odcienie szarości  
rozmiar pliku bez kompresji 467 kB

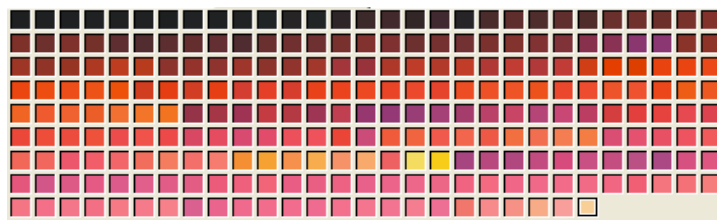
# Różne systemy numerowania, różne zapotrzebowanie na zasoby komputera



Standardowa paleta 256 kolorów  
rozmiar pliku bez kompresji 469 kB



Zoptymalizowana paleta 256 kolorów  
rozmiar pliku bez kompresji 469 kB



# Różne systemy numerowania, różne zapotrzebowanie na zasoby komputera



Czarno-biały

rozmiar pliku bez kompresji 58 kB



Duotone

rozmiar pliku bez kompresji 620 kB

# Pewne charakterystyczne wartości

Liczba bitów	Liczba symboli	Potęga podstawy 2
3	8	$2^3$
4	16	$2^4$
7	128	$2^7$
8	256	$2^8$
16	65536	$2^{16}$
32	4294967296	$2^{32}$
64	18 446 744 073 709 551 616	$2^{64}$

# Wielokrotności bajtu

- Podstawowe wielokrotności (mnożniki) jednostek podstawowych w fizyce i analogowym życiu:

kilo	= 1000	= $10^3$	
mega	= 1000000	= $10^6$	= kilo · 1000
giga	= 1000000000	= $10^9$	= mega · 1000
tera	= 1000000000000	= $10^{12}$	= giga · 1000

- Podstawowe wielokrotności (mnożniki) dla danych binarnych:

Kilo	= 1024	= $2^{10}$	
Mega	= 1048576	= $2^{20}$	= Kilo · 1024
Giga	= 1073741824	= $2^{30}$	= Mega · 1024
Tera	= 1099511627776	= $2^{40}$	= Giga · 1024

# Wielokrotności bajtu

- Podstawowe wielokrotności (mnożniki) jednostek podstawowych w fizyce i analogowym życiu:

kilo = 1000 =  $10^3$

Ważne

Podstawą mnożników binarnych jest 2 a nie 10, dlatego stosujemy:

**1024 bo  $2^{10}$**

- a nie:

**1000 bo  $10^3$**

Giga = 1073741824 =  $2^{30}$  = Mega · 1024

Tera = 1099511627776 =  $2^{40}$  = Giga · 1024

# Wielokrotności bajtu – problemy z nazwami

- Podobieństwo nazw mnożników w układzie dziesiętnym SI i binarnym powodowało problemy.
- Uwzględniono to w normie [IEC 60027-2 Units](#), niezłe objaśnienie dostępne [tutaj](#).

Binary Prefixes				Decimal Prefixes (SI)		
Name	Symbol	Power	Origin	Name	Symbol	Power
kibi	Ki	$(2^{10})^1$	Kilobinary	kilo	(k)	$(10^3)^1$
mebi	Mi	$(2^{10})^2$	Megabinary	mega	(M)	$(10^3)^2$
gibi	Gi	$(2^{10})^3$	Gigabinary	giga	(G)	$(10^3)^3$
tebi	Ti	$(2^{10})^4$	Terabinary	tera	(T)	$(10^3)^4$
pebi	Pi	$(2^{10})^5$	Petabinary	peta	(P)	$(10^3)^5$
exbi	Ei	$(2^{10})^6$	Exabinary	exa	(E)	$(10^3)^6$
zebi	Zi	$(2^{10})^7$	Zettabinary	zetta	(Z)	$(10^3)^7$
yobi	Yi	$(2^{10})^8$	Yottabinary	yotta	(Y)	$(10^3)^8$

# Wielokrotności bajtu – problemy z nazwami

- Podobieństwo nazw mnożników w układzie dziesiętnym SI i binarnym powodowało problemu.
- Uwzględniono to w normie [IEC 60027-2 Units](#), niezłe objaśnienie dostępne [tutaj](#).

Binary Prefixes				Decimal Prefixes (SI)		
Name	Symbol	Power	Origin	Name	Symbol	Power
kibi	Ki	$(2^{10})^1$	Kilobinary	kilo	(k)	$(10^3)^1$
mebi	Mi	$(2^{10})^2$	Megabinary	mega	(M)	$(10^3)^2$
gibi	Gi	$(2^{10})^3$	Gigabinary	giga	(G)	$(10^3)^3$
tebi	Ti	$(2^{10})^4$	Terabinary	tera	(T)	$(10^3)^4$
pebi	Pi	$(2^{10})^5$	Petabinary	peta	(P)	$(10^3)^5$
exbi	Ei	$(2^{10})^6$	Exabinary	exa	(E)	$(10^3)^6$
zebi	Zi	$(2^{10})^7$	Zettabinary	zetta	(Z)	$(10^3)^7$
yobi	Yi	$(2^{10})^8$	Yottabinary	yotta	(Y)	$(10^3)^8$

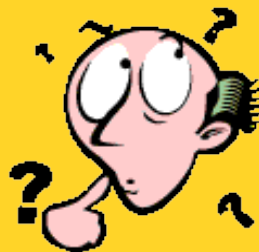
1024

1000

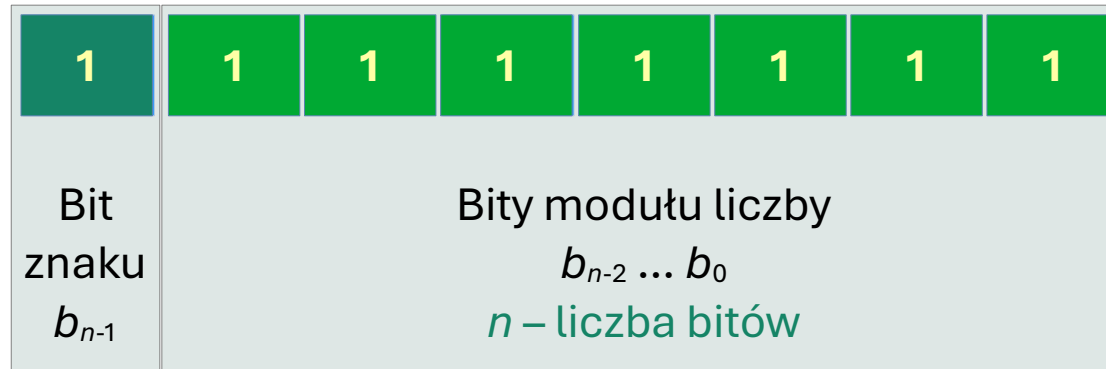
# Naturalny kod binarny – tylko całkowite wartości dodatnie

Liczba bitów	Liczba symboli	Zakres liczb
1	2	0..1
2	4	0..3
3	8	0..7
4	16	0..15
5	32	0..31

Co z wartościami ujemnymi?

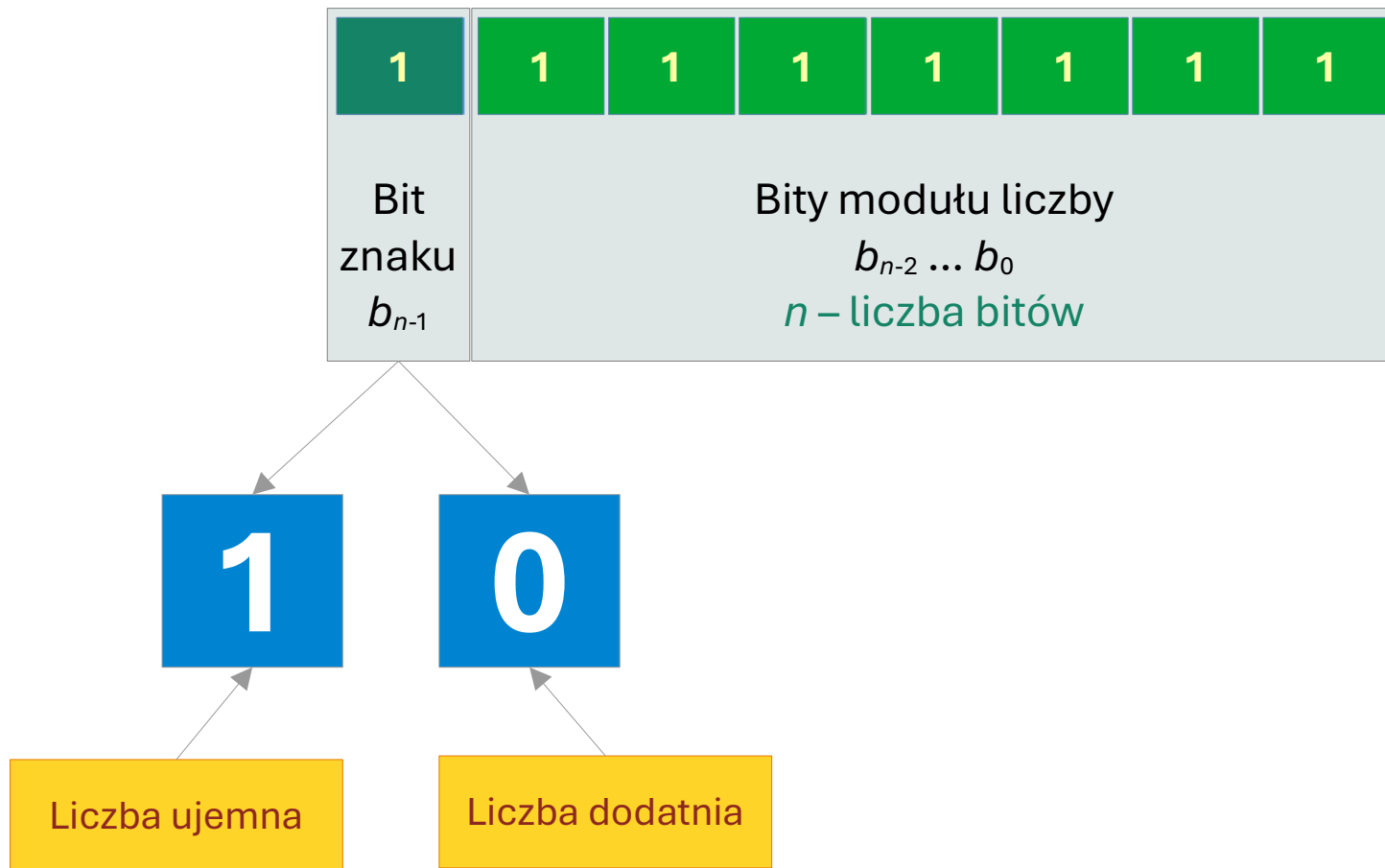


# Zapis znak-moduł, koncepcja

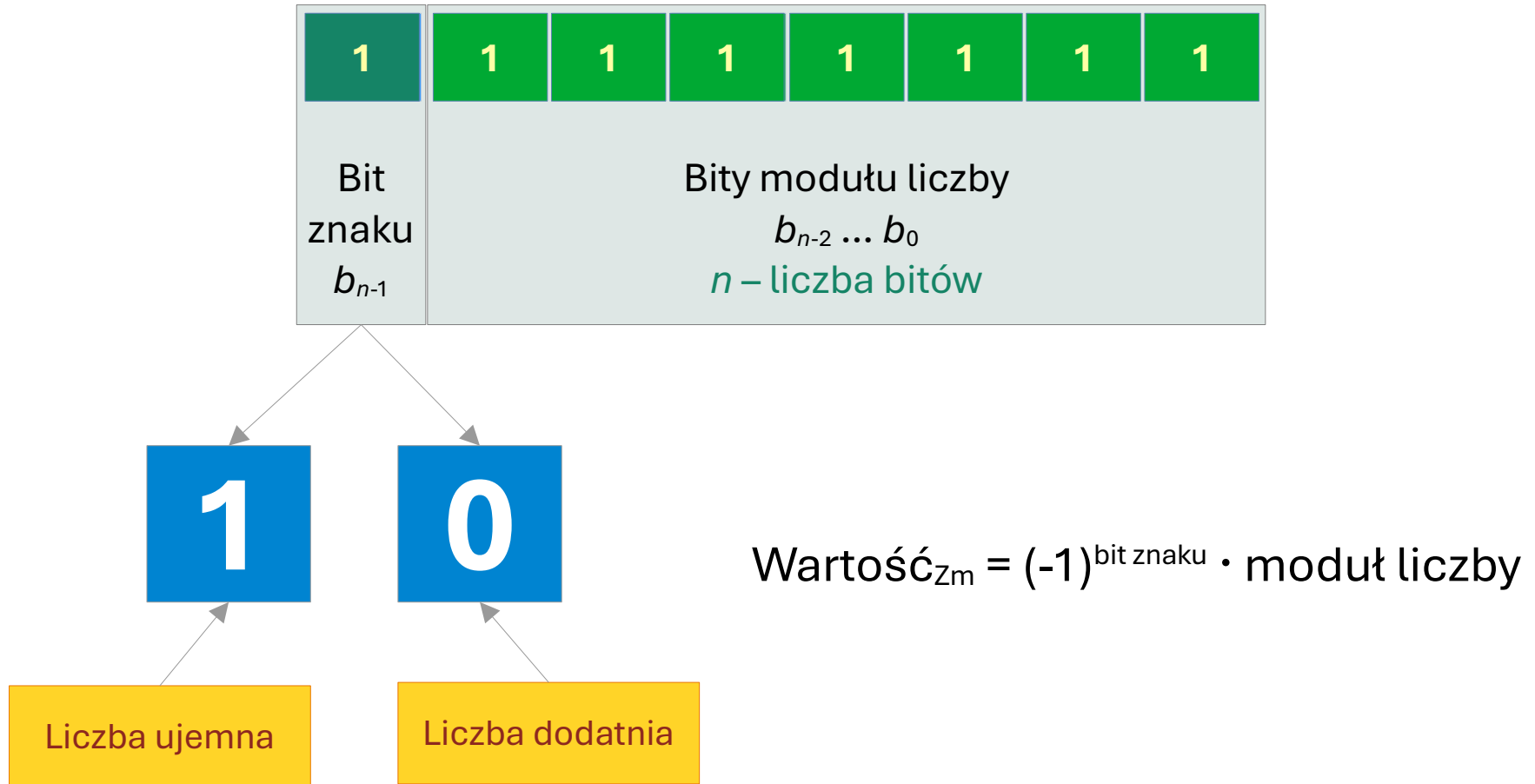


- Kod *znak-moduł* to sposób zapisu liczb całkowitych oznaczany jako *ZM* lub *SM* (ang. *sign-magnitude*).
- Wszystkie bity poza najstarszym mają takie samo znaczenie jak w *naturalnym* kodzie binarnym.

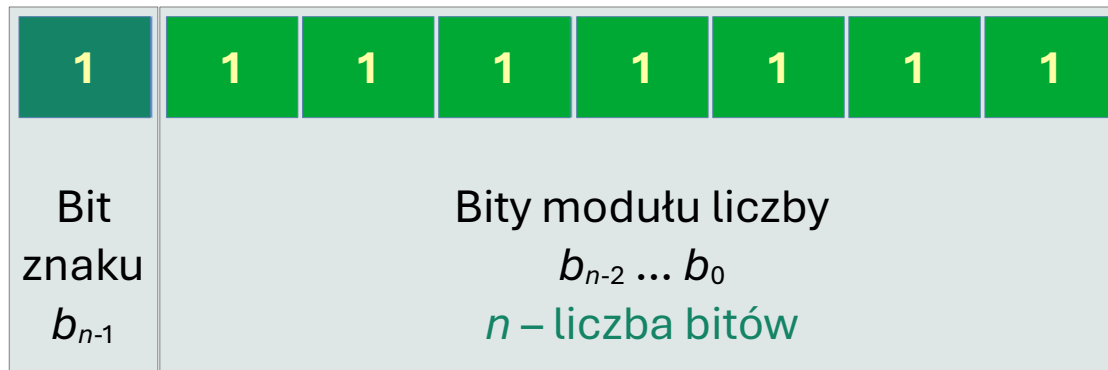
# Zapis znak-moduł, bit znaku



# Zapis znak-moduł, wartość liczby



## Zapis znak-moduł, wartość liczby, nieco formalniej



Wartość<sub>Zm</sub> =  $(-1)^{\text{bit znaku}} \cdot \text{moduł liczby}$

$$b_{n-1} b_{n-2} \dots b_2 b_1 b_0 = (-1)^{b_{n-1}} \cdot (b_{n-2} \cdot 2^{n-2} + \dots + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0)$$

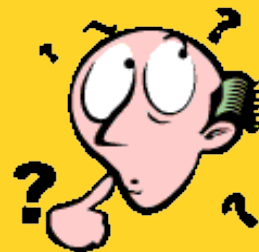
## Zapis znak-moduł, kilka przykładowych wartości

Liczba	Rozwinięcie	Wartość	Liczba	Rozwinięcie	Wartość
0000	$(-1)^0 \cdot 0$	0	1000	$(-1)^1 \cdot 0$	0
0001	$(-1)^0 \cdot (2^0)$	1	1001	$(-1)^1 \cdot (2^0)$	-1
0010	$(-1)^0 \cdot (2^1)$	2	1010	$(-1)^1 \cdot (2^1)$	-2
0011	$(-1)^0 \cdot (2^1 + 2^0)$	3	1011	$(-1)^1 \cdot (2^1 + 2^0)$	-3
0100	$(-1)^0 \cdot (2^2)$	4	1100	$(-1)^1 \cdot (2^2)$	-4
0101	$(-1)^0 \cdot (2^2 + 2^0)$	5	1101	$(-1)^1 \cdot (2^2 + 2^0)$	-5
0110	$(-1)^0 \cdot (2^2 + 2^1)$	6	1110	$(-1)^1 \cdot (2^2 + 2^1)$	-6
0111	$(-1)^0 \cdot (2^2 + 2^1 + 2^0)$	7	1111	$(-1)^1 \cdot (2^2 + 2^1 + 2^0)$	-7

# Zapis znak-moduł, kilka przykładowych wartości

Liczba	Rozwinięcie	Wartość	Liczba	Rozwinięcie	Wartość
0000	$(-1)^0 \cdot 0$	0	1000	$(-1)^1 \cdot 0$	0
0001	$(-1)^0 \cdot (2^0)$	1	1001	$(-1)^1 \cdot (2^0)$	-1
0010	$(-1)^0 \cdot (2^1)$	2	1010	$(-1)^1 \cdot (2^1)$	-2
0011	$(-1)^0 \cdot (2^1 + 2^0)$	3	1011	$(-1)^1 \cdot (2^1 + 2^0)$	-3
0100	$(-1)^1 \cdot 0$	-0	1100	$(-1)^1 \cdot 0$	-0
0101	$(-1)^1 \cdot (2^0)$	-1	1101	$(-1)^1 \cdot (2^0)$	-1
0110	$(-1)^1 \cdot (2^1)$	-2	1110	$(-1)^1 \cdot (2^1)$	-2
0111	$(-1)^1 \cdot (2^1 + 2^0)$	-3	1111	$(-1)^1 \cdot (2^1 + 2^0)$	-3

Co można zauważyć?



# Zapis znak-moduł, kilka przykładowych wartości

Liczba	Rozwinięcie	Wartość	Liczba	Rozwinięcie	Wartość
0000	$(-1)^0 \cdot 0$	0	1000	$(-1)^1 \cdot 0$	0
0001	$(-1)^0 \cdot (2^0)$	1	1001	$(-1)^1 \cdot (2^0)$	-1
0010	$(-1)^0 \cdot (2^1)$	2	1010	$(-1)^1 \cdot (2^1)$	-2
0011	$(-1)^0 \cdot (2^1 + 2^0)$	3	1011	$(-1)^1 \cdot (2^1 + 2^0)$	-3
0100	$(-1)^0 \cdot (2^2)$	4			-4
0101	$(-1)^0 \cdot (2^2 + 2^0)$	5		$2^0$	-5
0110	$(-1)^0 \cdot (2^2 + 2^1)$	6		$2^1$	-6
0111	$(-1)^0 \cdot (2^2 + 2^1 + 2^0)$	7		$2^1 + 2^0$	-7

Dwie wartości dla **0**

Zakres wartości:

$$-(2^{n-1} - 1) .. (2^{n-1} - 1)$$

# Zapis znak moduł sprawia problemy przeliczeniach

- Kodowanie ZM sprawia problemy przy obliczeniach automatycznych – bit znaku należy traktować specjalnie.
- Opracowano dwa inne sposoby wykorzystujące nieco inaczej traktowany bit znaku.
- W przypadku liczb dodatnich wszystkie sposoby zapisu liczby takie traktują tak samo.
- Te dwa sposoby to kodu **U1 – uzupełnienie do jeden** i **U2 – uzupełnienie do 2**.
- Więcej informacji:

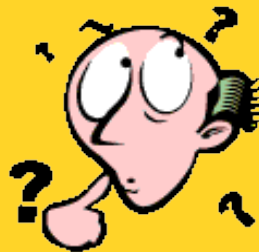
## Wykład: Teoretyczne podstawy informatyki

- Tutaj dostępne są przystępne i świetnie opracowane materiały na temat arytmetyki binarnej wraz z wieloma przykładami. Polecam.
- Że co, że te materiały są dla liceum? Nie szkodzi, są dobre i wystarczająco zaawansowane.

# Jakie systemy oprócz binarnego i dziesiętnego są wykorzystywane?

- Zapis liczb w postaci binarnej jest potrzebny i często stosowany.
- Zapis liczb w postaci długich ciągów zer i jedynek jest niewygodny i sprzyja popełnieniu przez człowieka błędów.
- Do zwięzłego zapisywania liczb binarnych stosuje się system **ósemkowy** i **szesnastkowy**.

Dlaczego akurat te systemy?



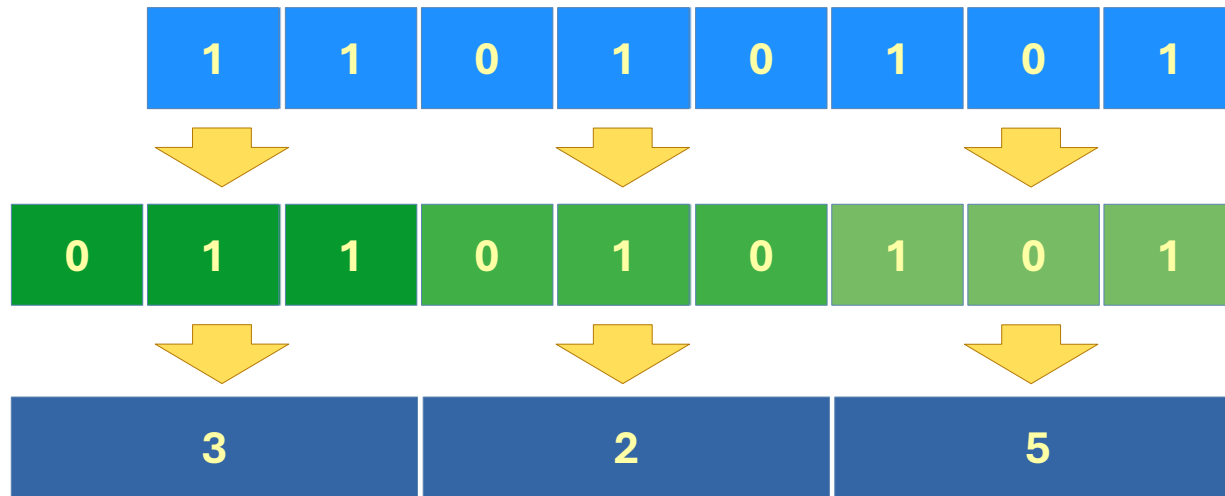
# Z systemy binarnego na ósemkowy

- Podstawą systemu ósemkowego jest – jak łatwo się domyślić – wartość 8.
- System ten będzie miał zatem osiem cyfr: 0 1 2 3 4 5 6 7
- Binarne rozwinięcia cyfr ósemkowych:

Cyfra ósemkowa	Wartość dwójkowa
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

## Z systemu binarnego na ósemkowy

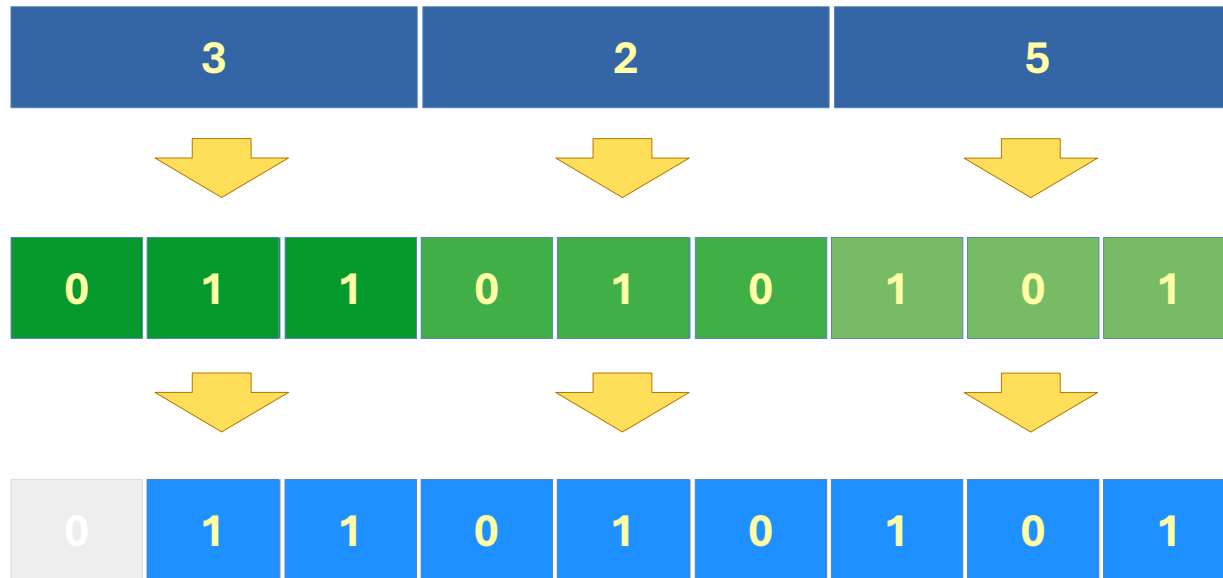
- Dzielimy liczbę binarną na grupy liczące 3 bity, zaczynamy od strony prawej.
- Jeżeli ostatnia grupa nie jest pełna, uzupełniamy ją zerami od strony lewej.
- Zastępujemy każdą 3 bitową grupę odpowiednią cyfrą ósemkową.



$$11010101_{(2)} = 325_{(8)}$$

# Z systemu ósemkowego na binarny

- Dla każdej cyfry ósemkowej znajdujemy jej binarny odpowiednik.
- Składamy znalezione odpowiedniki w liczbę binarną.



$$325_{(8)} = 11010101_{(2)}$$

# Z systemy binarnego na szesnastkowy

- Podstawą systemu szesnastkowego jest – jak również wiadomo – wartość 16.
- System ten będzie miał zatem szesnaście cyfr:

0 1 2 3 4 5 6 7 8 9 A B C D E F      lub   0 1 2 3 4 5 6 7 8 9 a b c d e f

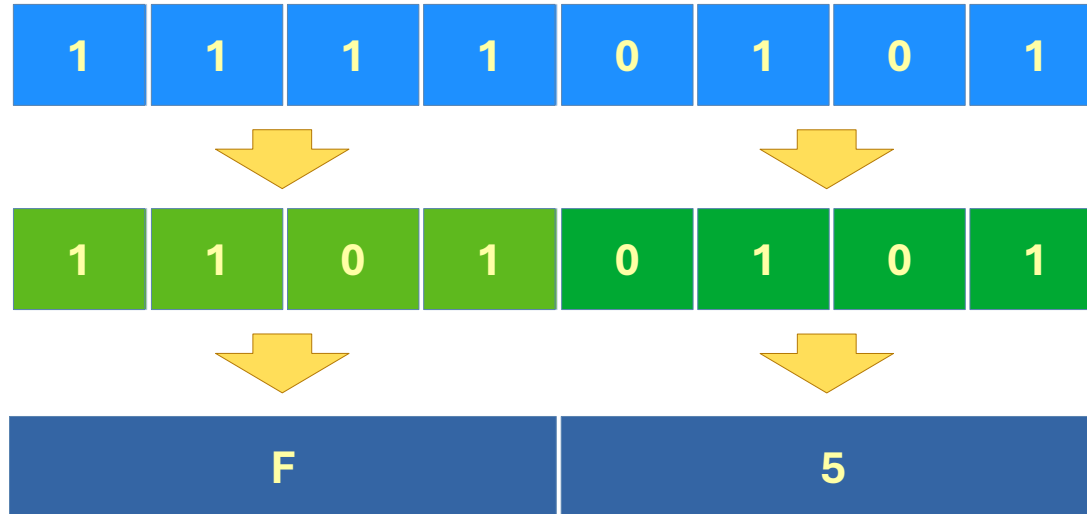
Binarne rozwinięcia cyfr szesnastkowych:

Cyfra szesnastkowa	Wartość dwójkowa
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Cyfra szesnastkowa	Wartość dwójkowa
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

## Z systemu binarnego na szesnastkowy

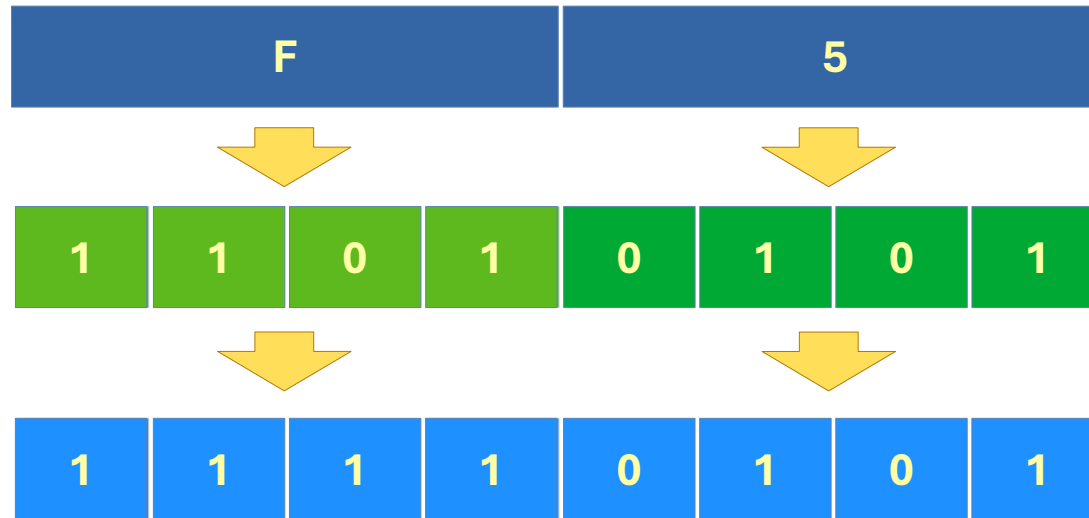
- Dzielimy liczbę binarną na grupy liczące 4 bity, zaczynamy od strony prawej.
- Jeżeli ostatnia grupa nie jest pełna, uzupełniamy ją zerami od strony lewej.
- Zastępujemy każdą 4 bitową grupę odpowiednią cyfrą szesnastkową.



$$11110101_{(2)} = F5_{(16)}$$

# Z systemu szesnastkowego na binarny

- Dla każdej cyfry szesnastkowej znajdujemy jej binarny odpowiednik.
- Składamy znalezione odpowiedniki w liczbę binarną.



$$F5_{(16)} = 11110101_{(2)}$$

# Przykład wykorzystania literałów szesnastkowych

Fragment pliku `qglobal.h` (Qt)

```
enum WinVersion
{
    WV_32s      = 0x0001,
    WV_95       = 0x0002,
    WV_98       = 0x0003,
    WV_Me       = 0x0004,
    WV_DOS_based = 0x000f,

    /* codenames */
    WV_NT       = 0x0010,
    WV_2000     = 0x0020,
    WV_XP       = 0x0030,
    WV_2003     = 0x0040,
    WV_VISTA    = 0x0080,
    WV_WINDOWS7 = 0x0090,
    WV_NT_based = 0x00f0,

    . . .
};
```

Prefiks `0x` – liczba szesnastkowa

Prefiks `0` – liczba ósemkowa

**Dziękuję za uwagę**

**roman.siminski@us.edu.pl**

**roman@siminskionline.pl**