

Informatyka 1

Politechnika Białostocka - Wydział Elektryczny
Elektrotechnika, semestr II, studia niestacjonarne I stopnia
Rok akademicki 2017/2018

Wykład nr 3 (16.03.2018)

dr inż. Jarosław Forenc

Plan wykładu nr 3

- Język C
 - instrukcje warunkowa if, operator warunkowy, instrukcja switch
- Kodowanie liczb
 - NKB, BCD, 2 z 5, Graya
- Reprezentacja liczb całkowitych
 - bez znaku, ze znakiem (ZM, U1, U2)

Język C - Pierwiastek kwadratowy

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    float x, y;

    printf("Podaj liczbe: ");
    scanf("%f", &x);

    y = sqrt(x);

    printf("Pierwiastek liczby: %f\n", y);

    return 0;
}
```

```
Podaj liczbe: 15
Pierwiastek liczby: 3.872983
```

```
Podaj liczbe: -15
Pierwiastek liczby: -1.#IND00
```

Język C - Pierwiastek kwadratowy

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    float x, y;

    printf("Podaj liczbe: ");
    scanf("%f", &x);

    if (x >= 0)
    {
        y = sqrt(x);
        printf("Pierwiastek liczby: %f\n", y);
    }
    else
        printf("Blad! Liczba ujemna\n");

    return 0;
}
```

Podaj liczbe: 15
Pierwiastek liczby: 3.872983

Podaj liczbe: -15
Blad! Liczba ujemna

Język C - instrukcja warunkowa if

```
if (wyrażenie)
    instrukcja1
```

- jeśli **wyrażenie** jest prawdziwe, to wykonywana jest **instrukcja1**
- gdy **wyrażenie** jest fałszywe, to **instrukcja1** nie jest wykonywana

```
if (wyrażenie)
    instrukcja1
else
    instrukcja2
```

- jeśli **wyrażenie** jest prawdziwe, to wykonywana jest **instrukcja1**, zaś **instrukcja2** nie jest wykonywana
- gdy **wyrażenie** jest fałszywe, to wykonywana jest **instrukcja2**, zaś **instrukcja1** nie jest wykonywana

■ Wyrażenie w nawiasach:

- **prawdziwe** - gdy jego wartość jest różna od zera
- **fałszywe** - gdy jego wartość jest równa zero

Język C - instrukcja warunkowa if

```
if (wyrażenie)  
    instrukcja
```

■ Instrukcja:

- **prosta** - jedna instrukcja zakończona średnikiem
- **złożona** - jedna lub kilka instrukcji objętych nawiasami klamrowymi

```
if (x>0)  
    printf("inst1");
```

```
if (x>0)  
{  
    printf("inst1");  
    printf("inst2");  
    ...  
}
```

Język C - instrukcja warunkowa if

```
if (wyr)
    instr;
```

```
if (wyr)
    instr;
else
    instr;
```

```
if (wyr)
{
    instr;
    instr;
}
else
    instr;
```

```
if (wyr)
{
    instr;
}
else
{
    instr;
}
```

```
if (wyr)
{
    instr;
    instr;
}
```

```
if (wyr)
{
    instr;
    instr;
}
else
{
    instr;
    instr;
}
```

```
if (wyr)
    instr;
else
{
    instr;
    instr;
}
```

Język C - Operatory relacyjne (porównania)

Operator	Przykład	Znaczenie
>	$a > b$	a większe od b
<	$a < b$	a mniejsze od b
>=	$a >= b$	a większe lub równe b
<=	$a <= b$	a mniejsze lub równe b
==	$a == b$	a równe b
!=	$a != b$	a nierówne b (a różne od b)

- Wynik porównania jest wartością typu **int** i jest równy:
 - **1** - gdy warunek jest prawdziwy
 - **0** - gdy warunek jest fałszywy (nie jest prawdziwy)

Język C - Operatory logiczne

Operator	Znaczenie	Opis
!	NOT, nie	jednoargumentowy operator negacji logicznej - zmienia argument różny od zera na wartość 0, a argument równy zero na wartość 1
&&	AND, i	dwuargumentowy operator koniunkcji, iloczyn logiczny
	OR, lub	dwuargumentowy operator alternatywy, suma logiczna

- Wynikiem zastosowania operatorów logicznych `&&` i `||` jest wartość typu `int` równa 1 (prawda) lub 0 (fałsz)

```
if (x>5 && x<8)
```

```
if (x<=5 || x>8)
```

Język C - Wyrażenia logiczne

- Wyrażenia logiczne mogą zawierać:

- operatory relacyjne
- operatory logiczne
- operatory arytmetyczne
- operatory przypisania
- zmienne
- stałe
- wywołania funkcji
- ...

- Kolejność operacji wynika z **priorytetu operatorów**

Operator	Typ operatora
!	logiczny
* / %	arytmetyczne
+ -	arytmetyczne
> < >= <=	relacyjne
== !=	relacyjne
&&	logiczny
	logiczny
=	przypisania

Język C - Wyrażenia logiczne

```
int x = 0, y = 1, z = 2;
```

```
if ( x == 0 )
```

wynik: 1 (prawda)

```
if ( x = 0 )
```

wynik: 0 (fałsz) (!!!)

```
if ( x != 0 )
```

wynik: 0 (fałsz)

```
if ( x =! 0 )
```

wynik: 1 (prawda) (!!!)

```
if ( z > x + y )
```

wynik: 1 (prawda)

```
if ( z > ( x + y ) )
```

Język C - Wyrażenia logiczne

```
int x = 0, y = 1, z = 2;
```

```
if ( x>2 && x<5 )
```

wynik: 0 (fałsz)

```
if ( (x>2) && (x<5) )
```

- Wyrażenia logiczne obliczane są od strony lewej do prawej
- Proces obliczeń kończy się, gdy wiadomo, jaki będzie wynik całego wyrażenia

```
if ( 2 < x < 5 )
```

wynik: 1 (prawda) (!!!)

Język C - Wyrażenia logiczne

- W przypadku sprawdzania czy wartość wyrażenia jest równa lub różna od zera można zastosować skrócony zapis
- Zamiast:

```
if ( x == 0 )
```

```
if ( x != 0 )
```

można napisać:

```
if ( !x )
```

```
if ( x )
```

Język C - Operator warunkowy

- Operator warunkowy składa się z dwóch symboli i trzech operandów

```
wyrażenie1 ? wyrażenie2 : wyrażenie3
```

- Najczęściej zastępuje proste instrukcje **if-else**

```
float akcyza, cena, pojemnosc;
```

```
if (pojemnosc <= 2000)
    akcyza = cena*0.031;    /* 3.1% */
else
    akcyza = cena*0.186;    /* 18.6% */
```

```
akcyza = pojemnosc <= 2000 ? cena*0.031 : cena*0.186;
```

Język C - Operator warunkowy

```
if (x < 0)
    y = -x;
else
    y = x;
```

```
y = x < 0 ? -x : x;
```

- obliczenie modułu liczby x

```
if (a > b)
    max = a;
else
    max = b;
```

```
max = a > b ? a : b;
```

- wyznaczenie max z dwóch liczb

- Operator warunkowy ma bardzo niski priorytet
- Niższy priorytet mają tylko operatory przypisania (`=`, `+=`, `-=`, ...) i operator przecinkowy (`,`)

Język C - Operator warunkowy

- x studentów chce dojechać z akademika do biblioteki - ile taksówek powinni zamówić? (jedna taksówka może przewieźć 4 osoby)

```
#include <stdio.h>

int main(void)
{
    int x, taxi;

    printf("Podaj liczbe studentow: ");
    scanf("%d", &x);

    taxi = x / 4 + (x % 4 ? 1 : 0);

    printf("Liczba taxi: %d\n", taxi);

    return 0;
}
```

```
Podaj liczbe studentow: 23
Liczba taxi: 6
```


Język C - Instrukcja switch

- Instrukcja wyboru wielowariantowego **switch**

```
switch (wyrażenie)
{
    case wyrażenie Stała: instrukcje;
    case wyrażenie Stała: instrukcje;
    case wyrażenie Stała: instrukcje;
    ...
    default: instrukcje;
}
```

- **wyrażenie Stała** - wartość typu całkowitego, znana podczas kompilacji
 - stała liczbowa, np. 3, 5, 9
 - znak w apostrofach, np. 'a', 'z', '+'
 - stała zdefiniowana przez **const** lub **#define**

Język C - Instrukcja switch

- Program wyświetlający słownie liczbę z zakresu 1..5 wprowadzoną z klawiatury

```
#include <stdio.h>

int main(void)
{
    int liczba;

    printf("Podaj liczbę (1..5): ");
    scanf("%d", &liczba);
```

Język C - Instrukcja switch

```
switch (liczba)
{
    case 1: printf("Liczba: jeden\n");
            break;
    case 2: printf("Liczba: dwa\n");
            break;
    case 3: printf("Liczba: trzy\n");
            break;
    case 4: printf("Liczba: cztery\n");
            break;
    case 5: printf("Liczba: piec\n");
            break;
    default: printf("Inna liczba\n");
}
}
```

Podaj liczbe: 2
Liczba: dwa

Podaj liczbe: 0
Inna liczba

Język C - Instrukcja switch

```
switch (liczba)
{
    case 1:
    case 3:
    case 5: printf("Liczba nieparzysta\n");
            break;
    case 2:
    case 4: printf("Liczba parzysta\n");
            break;
    default: printf("Inna liczba\n");
}
```

Podaj liczbe: 2
Liczba parzysta

- Te same instrukcje mogą być wykonane dla kilku etykiet **case**

Język C - Instrukcja switch

```
switch (liczba)
{
    case 1: case 3: case 5:
        printf("Liczba nieparzysta\n");
        break;
    case 2: case 4:
        printf("Liczba parzysta\n");
        break;
    default: printf("Inna liczba\n");
}
```

Podaj liczbe: 2
Liczba parzysta

- Etykiety **case** mogą być pisane w jednym wierszu

Język C - Instrukcja switch

```
switch (liczba%2)
{
    case 1: case -1:
        printf("Liczba nieparzysta\n");
        break;
    case 0:
        printf("Liczba parzysta\n");
}
```

Podaj liczbe: 2
Liczba parzysta

- Część domyślna (**default**) może być pominięta

Język C - Instrukcja switch (bez break)

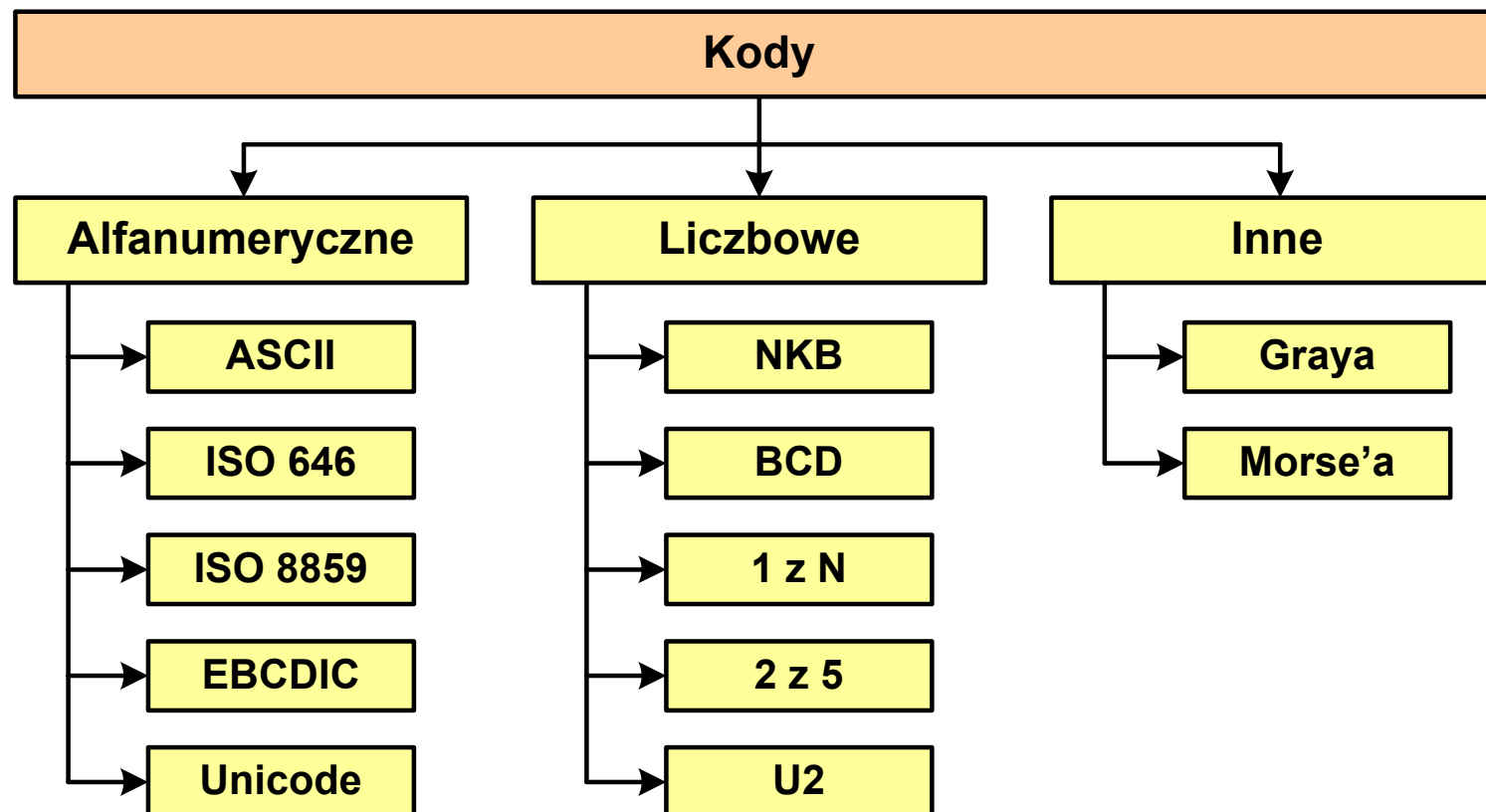
```
switch (liczba)
{
    case 1: printf("Liczba: jeden\n");
    case 2: printf("Liczba: dwa\n");
    case 3: printf("Liczba: trzy\n");
    case 4: printf("Liczba: cztery\n");
    case 5: printf("Liczba: piec\n");
    default: printf("Inna liczba\n");
}
```

```
Podaj liczbe: 2
Liczba: dwa
Liczba: trzy
Liczba: cztery
Liczba: piec
Inna liczba
```

- Pominięcie instrukcji **break** spowoduje wykonanie wszystkich instrukcji występujących po danym **case** (do końca **switch**)

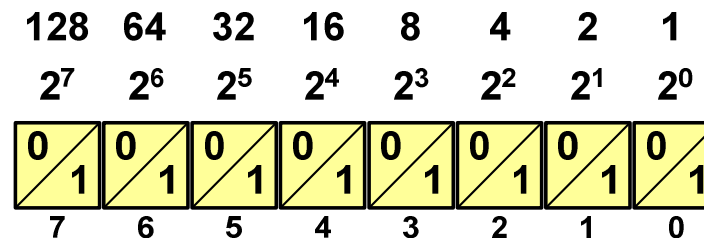
Kodowanie

- **Kodowanie** - proces przekształcania jednego rodzaju postaci informacji na inną postać



Kody liczbowe - Naturalny Kod Binarny (NKB)

- Jeżeli dowolnej liczbie dziesiętnej przypiszemy odpowiadającą jej liczbę binarną, to otrzymamy **naturalny kod binarny** (NKB)



Liczba dziesiętna	Kod NKB
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Liczba dziesiętna	Kod NKB
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Kody liczbowe - Kod BCD

- **B**inary-**C**oded **D**ecimal - dziesiętny zakodowany dwójkowo
- **BCD** - sposób zapisu liczb polegający na zakodowaniu kolejnych cyfr liczby dziesiętnej w 4-bitowym systemie dwójkowym (NKB)

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

- W ogólnym przypadku kodowane są tylko znaki **0 ÷ 9**
- Pozostałe kombinacje bitowe mogą być stosowane do kodowania znaku liczby lub innych znaczników.

Kody liczbowe - Kod BCD

■ Przykład:

$$168_{(10)} = ?_{(BCD)}$$

$$\begin{array}{ccc} \overbrace{0001}^1 & \overbrace{0110}^6 & \overbrace{1000}^8 \end{array}$$

$$168_{(10)} = 000101101000_{(BCD)}$$

$$1001 | 0101 | 0011_{(BCD)} = ?_{(10)}$$

$$\begin{array}{ccc} \underbrace{1001}_9 & \underbrace{0101}_5 & \underbrace{0011}_3 \end{array}$$

$$100101010011_{(BCD)} = 953_{(10)}$$

■ Zastosowania:

- urządzenia elektroniczne z wyświetlaczem cyfrowym (np. kalkulatory, mierniki cyfrowe, kasy sklepowe, wagi)
- przechowywania daty i czasu w BIOSie komputerów (także wczesne modele PlayStation 3)
- zapis części ułamkowych kwot (systemy bankowe).

Kody liczbowe - Kod BCD: przechowywanie liczb

- Użycie 4 najmłodszych bitów jednego bajta, 4 starsze bity są ustawiane na jakąś konkretną wartość:
 - 0000
 - 1111 (np. kod EBCDIC, liczby $F0_{(16)} \div F9_{(16)}$)
 - 0011 (tak jak w ASCII, liczby $30_{(16)} \div 39_{(16)}$)
- Zapis dwóch cyfr w każdym bajcie (starsza na starszej połówce, młodsza na młodszej połówce) - jest to tzw. **spakowane BCD**
 - w przypadku liczby zapisanej na kilku bajtach, najmniej znacząca tetrada (4 bity) używane są jako flaga znaku
 - standardowo przyjmuje się 1100 ($C_{(16)}$) dla znaku plus (+) i 1101 ($D_{(16)}$) dla znaku minus (-), np.

$$127_{(10)} = 0001\ 0010\ 0111\ \mathbf{1100} \quad (127C_{(16)})$$

$$-127_{(10)} = 0001\ 0010\ 0111\ \mathbf{1101} \quad (127D_{(16)})$$

Kody liczbowe - Kod BCD

- Warianty kodu BCD:

Cyfra dziesiętna	BCD 8421	Excess-3	BCD 2421	BCD 84-2-1	IBM 1401 BCD 8421
0	0000	0011	0000	0000	1010
1	0001	0100	0001	0111	0001
2	0010	0101	0010	0110	0010
3	0011	0110	0011	0101	0011
4	0100	0111	0100	0100	0100
5	0101	1000	1011	1011	0101
6	0110	1001	1100	1010	0110
7	0111	1010	1101	1001	0111
8	1000	1011	1110	1000	1000
9	1001	1100	1111	1111	1001

- Podstawowy wariant: **BCD 8421** (**SBCD** - Simple Binary Coded Decimal)

Kody liczbowe - Kod 2 z 5

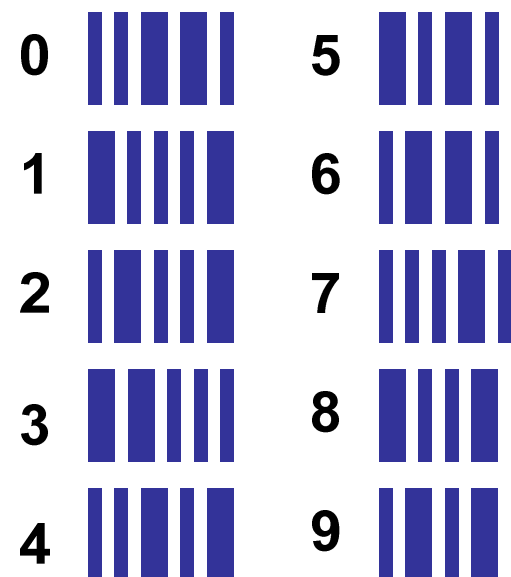
- Kod 5-bitowy: 2 bity zawsze równe 1, a 3 bity zawsze równe 0
- Koduje 10 znaków (cyfry dziesiętne), kody nie są wzajemnie jednoznaczne (ta sama wartość może być zakodowana w różny sposób)

- Kod stałowagowy
- Kod detekcyjny
- Stosowany głównie w kodach kreskowych

Liczba dziesiętna	2 z 5 (01236)	2 z 5 (01234)	2 z 5 (74210)
0	01100	01100	11000
1	11000	11000	00011
2	10100	10100	00101
3	10010	10010	00110
4	01010	01010	01001
5	00110	00110	01010
6	10001	10001	01100
7	01001	01001	10001
8	00101	00101	10010
9	00011	00011	10100

Kody liczbowe - Kod 2 z 5 Industrial (1960 r.)

- Jednowymiarowy kod kreskowy kodujący cyfry: 0 ÷ 9
- Znak to 5 pasków: 2 szerokie i 3 wąskie
- Szeroki pasek jest wielokrotnością wąskiego, szerokości muszą być takie same dla całego kodu
- Struktura kodu:
 - start: 11011010
 - numer
 - stop: 11010110



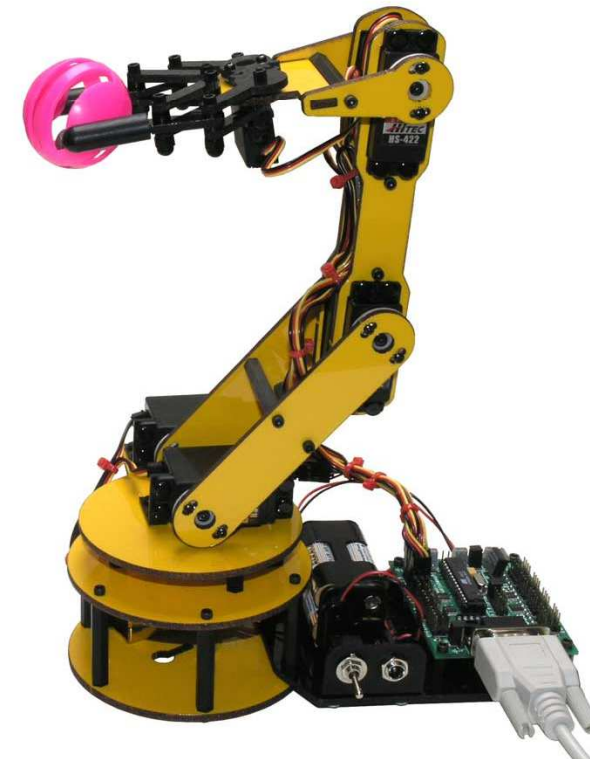
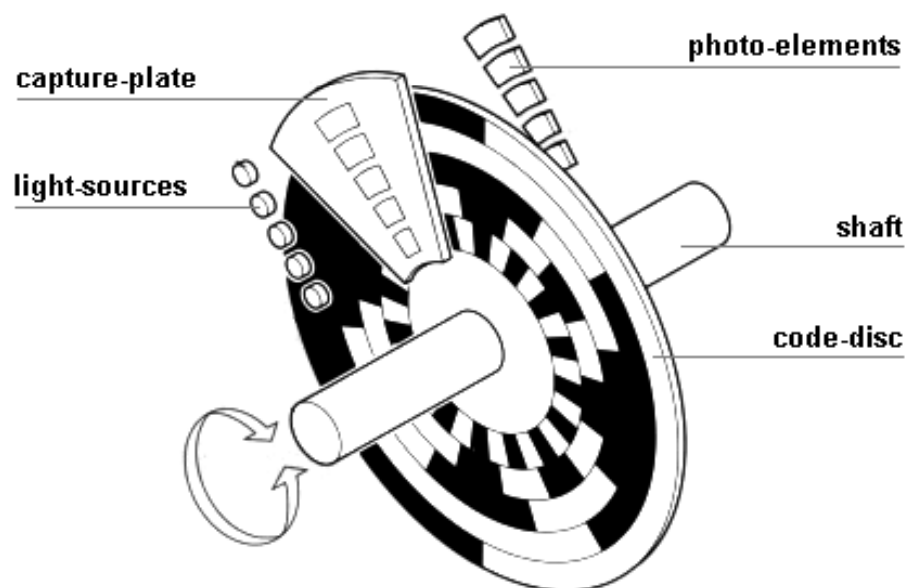
Kod Graya (refleksyjny)

- Kod dwójkowy, bezwagowy, niepozycyjny
- Dwa kolejne słowa kodowe różnią się stanem jednego bitu
- Kod cykliczny - ostatni i pierwszy wyraz również różnią się stanem jednego bitu

<u>kod 1-bitowy</u>	<u>kod 2-bitowy</u>	<u>kod 3-bitowy</u>
0	00	000
1	01	001
	<u>11</u>	011
	10	<u>010</u>
		110
		111
		101
		100

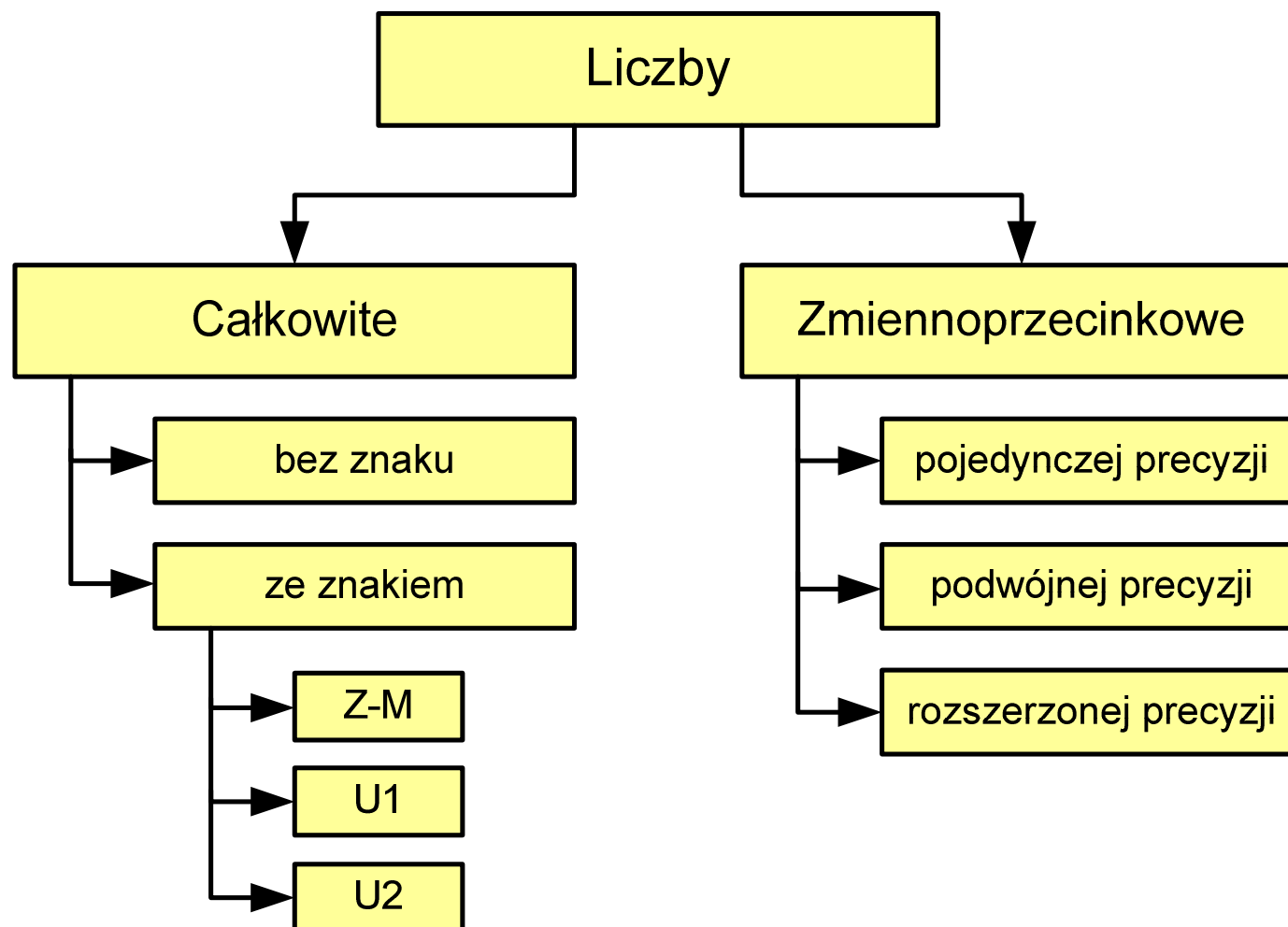
Kod Graya

- Stosowany w przetwornikach analogowo-cyfrowych, do cyfrowego pomiaru analogowych wielkości mechanicznych (np. kąt obrotu)



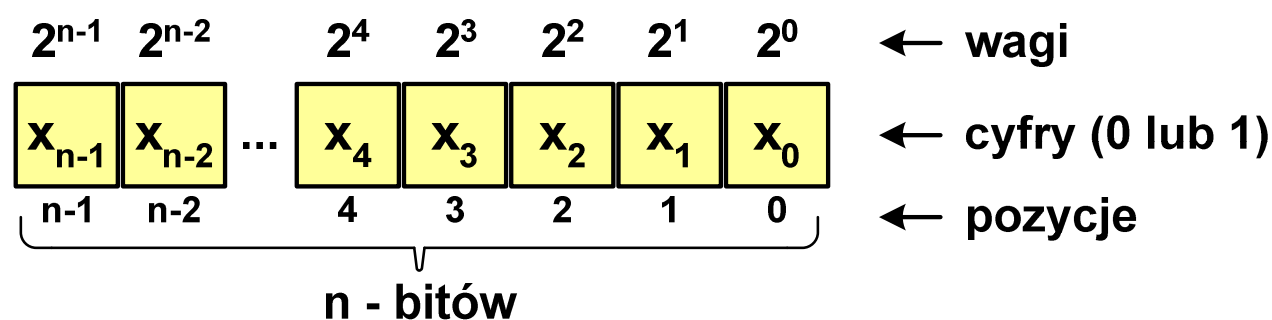
<http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/10-gates/15-graycode/dual2gray.html>

Reprezentacja liczb w systemach komputerowych



Liczby całkowite bez znaku

- Zapis liczby w systemie dwójkowym:



- Używając **n-bitów** można zapisać liczbę z zakresu:

$$X_{(2)} = \langle 0, 2^n - 1 \rangle$$

8-bitów 0 ... 255

16-bitów 0 ... 65 535

32-bity 0 ... 4 294 967 295

64-bity 0 ... 18 446 744 073 709 551 615

18 trylionów 446 miliardów 744 biliony 73 miliardy 709 milionów 551 tysięcy 615

Liczby całkowite bez znaku w języku C

- Typy zmiennych całkowitych bez znaku stosowane w języku C:

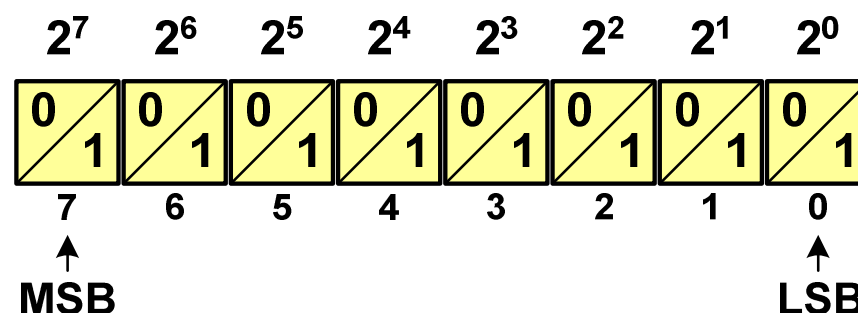
<u>Nazwa typu</u>	<u>Rozmiar (bajty)</u>	<u>Zakres wartości</u>
unsigned char	1 bajt	0 ... 255
unsigned short int	2 bajty	0 ... 65 535
unsigned int	4 bajty	0 ... 4 294 967 295
unsigned long int	4 bajty	0 ... 4 294 967 295
unsigned long long int	8 bajtów	0 ... 18 446 744 073 709 551 615

- W nazwach typów **short** i **long** można pominąć słowo **int**:

unsigned short int	→	unsigned short
unsigned long int	→	unsigned long
unsigned long long int	→	unsigned long long

Liczby całkowite bez znaku w języku C

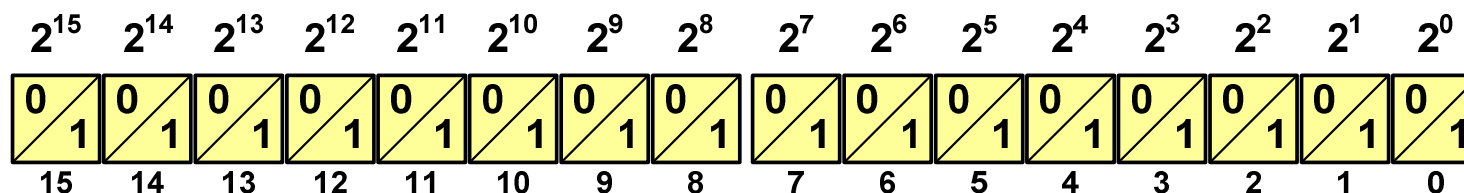
- Typ **unsigned char** (1 bajt):



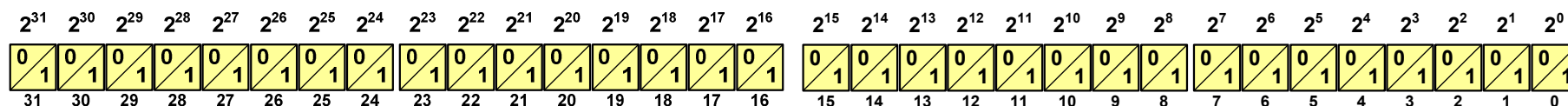
- **MSB** (Most Significant Bit) - najbardziej znaczący bit, najstarszy bit, największa waga
 - **LSB** (Least Significant Bit) - najmniej znaczący bit, najmłodszy bit, najmniejsza waga
- Zakres wartości:
 - dolna granica: $0000\ 0000_{(2)} = 00_{(16)} = 0_{(10)}$
 - górna granica: $1111\ 1111_{(2)} = FF_{(16)} = 255_{(10)}$

Liczby całkowite bez znaku w języku C

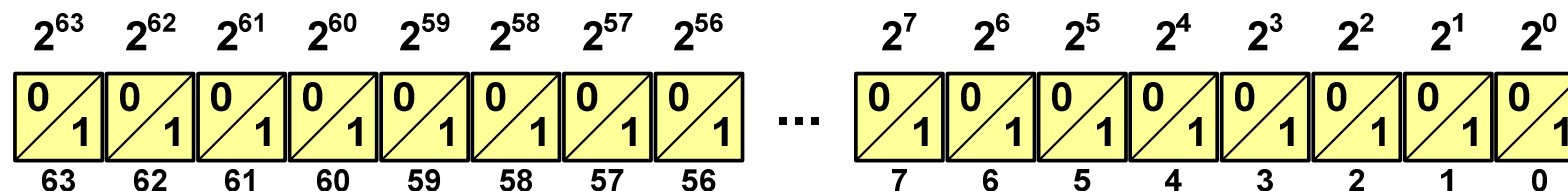
- Typ **unsigned short int** (2 bajty):



- Typy **unsigned int** (4 bajty) i **unsigned long int** (4 bajty):



- Typ **unsigned long long int** (8 bajtów):



Liczby całkowite bez znaku w języku C

```
unsigned short int:      65535 0 1
unsigned int:           4294967295 0 1
unsigned long int:     4294967295 0 1
unsigned long long int: 18446744073709551615 0 1
```

```
#include <stdio.h>

int main() /* przepełnienie zmiennej, ang. integer overflow */
{
    unsigned short int    usi = 65535;
    unsigned int          ui  = 4294967295;
    unsigned long int     uli  = 4294967295;
    unsigned long long int ulli = 18446744073709551615;

    printf("unsigned short int:      %hu %hu %hu\n", usi, usi+1, usi+2);
    printf("unsigned int:           %u %u %u\n", ui, ui+1, ui+2);
    printf("unsigned long int:      %lu %lu %lu\n", uli, uli+1, uli+2);
    printf("unsigned long long int: %llu %llu %llu\n",
           ulli, ulli+1, ulli+2);

    return 0;
}
```

Liczby całkowite bez znaku w języku C

```
unsigned short int:      1 0 65535
unsigned int:            1 0 4294967295
unsigned long int:      1 0 4294967295
unsigned long long int: 1 0 18446744073709551615
```

```
#include <stdio.h>

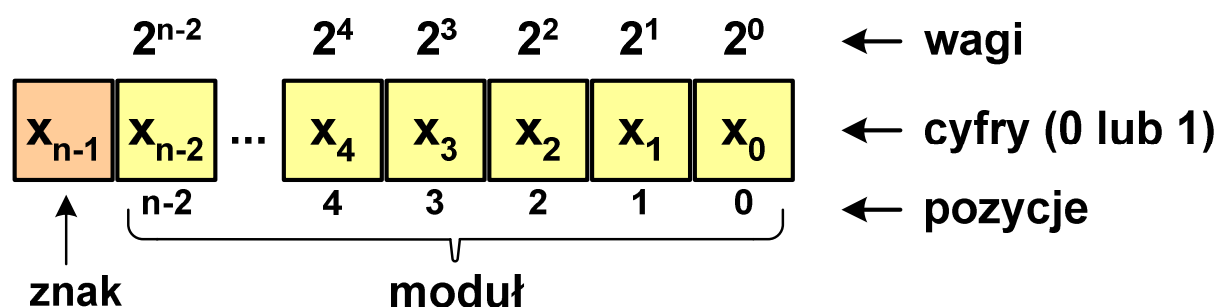
int main() /* przepełnienie zmiennej, ang. integer overflow */
{
    unsigned short int    usi = 1;
    unsigned int          ui = 1;
    unsigned long int     uli = 1;
    unsigned long long int ulli = 1;

    printf("unsigned short int:      %hu %hu %hu\n", usi, usi-1, usi-2);
    printf("unsigned int:           %u %u %u\n", ui, ui-1, ui-2);
    printf("unsigned long int:      %lu %lu %lu\n", uli, uli-1, uli-2);
    printf("unsigned long long int: %llu %llu %llu\n",
           ulli, ulli-1, ulli-2);

    return 0;
}
```


Liczby całkowite ze znakiem - kod znak-moduł

- Inne nazwy: **ZM**, **Z-M**, **SM (Signed Magnitude)**, **S+M**
- Najstarszy bit jest bitem znaku liczby: 0 - dodatnia, 1 - ujemna
- Pozostałe bity mają takie same znaczenie jak w **NKB**



- Wartość liczby:

$$X_{(10)} = \underbrace{(x_0 \cdot 2^0 + x_1 \cdot 2^1 + x_2 \cdot 2^2 + \dots + x_{n-2} \cdot 2^{n-2})}_{\text{moduł}} \cdot \underbrace{(-1)^{x_{n-1}}}_{\text{znak}} = (-1)^{x_{n-1}} \cdot \sum_{i=0}^{n-2} x_i \cdot 2^i$$

Liczby całkowite ze znakiem - kod znak-moduł

- Liczby **4-bitowe** (1 bit - znak, 3 bity - moduł) w kodzie **Z-M**:

Z-M	dziesiętnie	Z-M	dziesiętnie
0000	+0	1000	-0
0001	1	1001	-1
0010	2	1010	-2
0011	3	1011	-3
0100	4	1100	-4
0101	5	1101	-5
0110	6	1110	-6
0111	7	1111	-7

- dwie reprezentacje zera

+ 0 (0000_{ZM})

- 0 (1000_{ZM})

- Zakres liczb dla **n-bitów**:

$$X_{(10)} = \langle -2^{n-1} + 1, 2^{n-1} - 1 \rangle$$

dla 8 bitów : $\langle -127 \dots 127 \rangle$

dla 16 bitów : $\langle -32767 \dots 32767 \rangle$

Liczby całkowite ze znakiem - kod znak-moduł

■ Zamiana liczby dziesiętnej na kod Z-M:

- liczba dodatnia

$$93_{(10)} = ?_{(ZM)}$$

- zamieniamy liczbę na NKB

$$93_{(10)} = 1011101_{(NKB)}$$

- dodajemy bit znaku

$$93_{(10)} = 01011101_{(ZM)}$$

- liczba ujemna

$$-93_{(10)} = ?_{(ZM)}$$

- zamieniamy **moduł** liczby na NKB

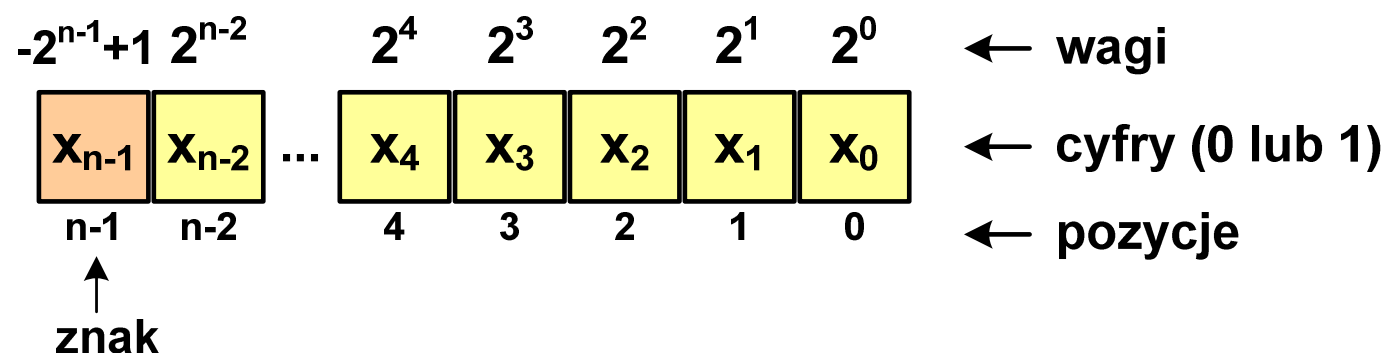
$$|-93_{(10)}| = 93_{(10)} = 1011101_{(NKB)}$$

- dodajemy bit znaku

$$-93_{(10)} = 11011101_{(ZM)}$$

Liczby całkowite ze znakiem - kod U1

- Inne nazwy: **U1, ZU1, uzupełnień do jedności**
- Najstarszy bit jest bitem znaku liczby: 0 - dodatnia, 1 - ujemna
- Wszystkie bity liczby posiadają takie same wagi jak w NKB, oprócz pierwszego bitu, który ma wagę $-2^{n-1} + 1$



- Wartość liczby:

$$X_{(10)} = x_0 \cdot 2^0 + x_1 \cdot 2^1 + x_2 \cdot 2^2 + \dots + x_{n-2} \cdot 2^{n-2} + x_{n-1} \cdot (-2^{n-1} + 1)$$

Liczby całkowite ze znakiem - kod U1

- Liczby **4-bitowe** (1 bit - znak, 3 bity - moduł) w kodzie **U1**:

U1	dziesiętnie	U1	dziesiętnie
0000	+0	1111	-0
0001	1	1110	-1
0010	2	1101	-2
0011	3	1100	-3
0100	4	1011	-4
0101	5	1010	-5
0110	6	1001	-6
0111	7	1000	-7

- liczby dodatnie zapisywane są tak samo jak w NKB
- liczby ujemne otrzymywane są poprzez bitową negację
- dwie reprezentacje zera

- Zakres liczb dla **n-bitów**:

$$X_{(10)} = \langle -2^{n-1} + 1, 2^{n-1} - 1 \rangle$$

dla 8 bitów : $\langle -127 \dots 127 \rangle$

dla 16 bitów : $\langle -32767 \dots 32767 \rangle$

Liczby całkowite ze znakiem - kod U1

- Zamiana liczby dziesiętnej na kod **U1**:

- liczba dodatnia

$$93_{(10)} = ?_{(U1)}$$

- zamieniamy liczbę na NKB

$$93_{(10)} = 1011101_{(NKB)}$$

- dodajemy bit znaku: 0

$$93_{(10)} = \mathbf{0}1011101_{(U1)}$$

- liczba ujemna

$$-93_{(10)} = ?_{(U1)}$$

- zamieniamy **moduł** liczby na U1

$$|-93_{(10)}| = 93_{(10)} = 01011101_{(U1)}$$

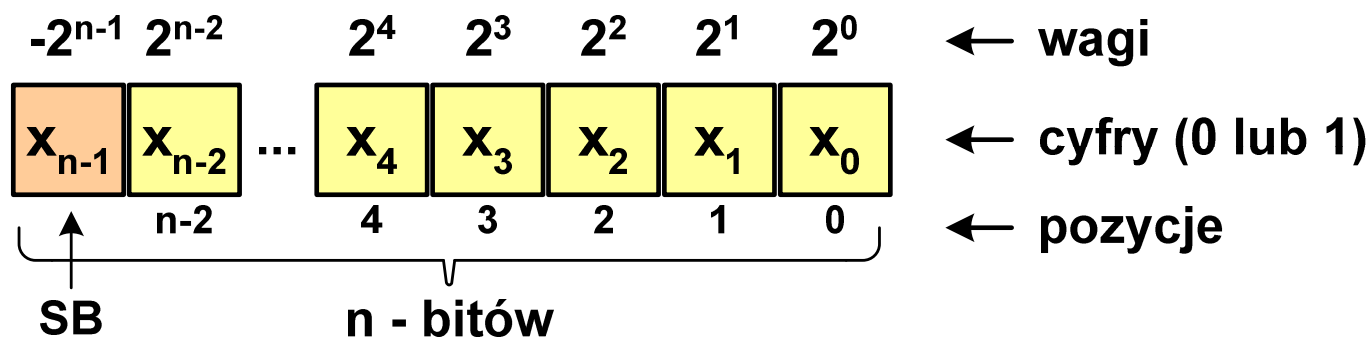
- negujemy wszystkie bity

$$-93_{(10)} = 10100010_{(U1)}$$

↖ bit znaku

Liczby całkowite ze znakiem - kod U2

- Inne nazwy: **ZU2**, **uzupełnień do dwóch**, **two's complement**
- Najstarszy bit jest bitem znaku liczby: 0 - dodatnia, 1 - ujemna



- Wartość liczby:

$$X_{(10)} = x_0 \cdot 2^0 + x_1 \cdot 2^1 + x_2 \cdot 2^2 + \dots + x_{n-2} \cdot 2^{n-2} + x_{n-1} \cdot (-2^{n-1})$$

- Kod **U2** jest obecnie powszechnie stosowany w informatyce

Liczby całkowite ze znakiem - kod U2

- Liczby **4-bitowe** (1 bit - znak, 3 bity - moduł) w kodzie **U2**:

U2	dziesiętnie	U2	dziesiętnie
0000	0	1111	-1
0001	1	1110	-2
0010	2	1101	-3
0011	3	1100	-4
0100	4	1011	-5
0101	5	1010	-6
0110	6	1001	-7
0111	7	1000	-8

- brak podwójnej reprezentacji zera
- liczb ujemnych jest o jeden więcej niż dodatnich
- **00...000** zawsze oznacza $0_{(10)}$
11...111 zawsze oznacza $-1_{(10)}$

- Zakres liczb dla **n-bitów**:

$$X_{(10)} = \langle -2^{n-1}, 2^{n-1} - 1 \rangle$$

dla 8 bitów : $\langle -128 \dots 127 \rangle$

dla 16 bitów : $\langle -32768 \dots 32767 \rangle$

Liczby całkowite ze znakiem - kod U2

■ Zamiana liczby dziesiętnej na kod U2:

- liczba dodatnia

$$75_{(10)} = ?_{(U2)}$$

- zamieniamy liczbę na NKB

$$75_{(10)} = 1001011_{(NKB)}$$

- dodajemy bit znaku: 0

$$75_{(10)} = \mathbf{0}1001011_{(U2)}$$

- liczba ujemna

$$-75_{(10)} = ?_{(U2)}$$

- zamieniamy **moduł** liczby na U2

$$|-75_{(10)}| = 75_{(10)} = 01001011_{(U2)}$$

- negujemy wszystkie bity i dodajemy 1

$$\begin{array}{r} 01001011 \\ \text{negacja: } 10110100 \\ +1: \qquad \qquad 1 \\ \hline -75_{(10)} = 10110101_{(U2)} \end{array}$$

Liczby całkowite ze znakiem - kod U2 w języku C

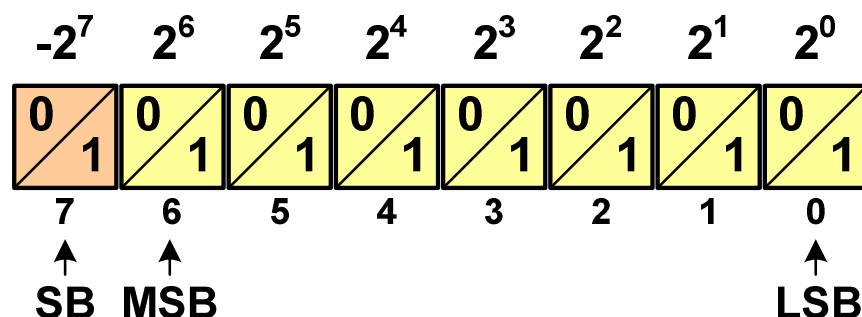
- Typy zmiennych całkowitych ze znakiem stosowane w języku C:

<u>Nazwa typu</u>	<u>Rozmiar (bajty)</u>	<u>Zakres wartości</u>
char	1 bajt	-128 ... 127
short int	2 bajty	-32 768 ... 32 767
int	4 bajty	-2 147 483 648 ... 2 147 483 647
long int	4 bajty	-2 147 483 648 ... 2 147 483 647
long long int	8 bajtów	-9 223 372 036 854 775 808 ... 9 223 372 036 854 775 807

- Przed nazwą każdego z powyższych typów można dodać **signed**
signed char, **signed short int**, **signed int** ...
- W nazwach typów **short** i **long** można pominąć słowo **int**:
short int → **short**, **long int** → **long**, **long long int** → **long long**

Liczby całkowite ze znakiem - kod U2 w języku C

- Typ `char` / `signed char` (1 bajt):

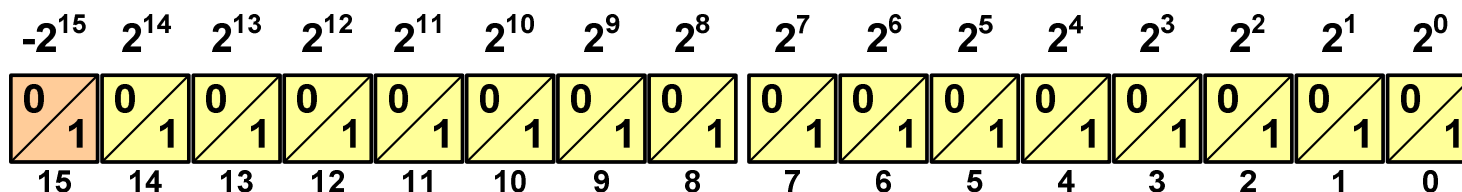


- Zakres wartości:

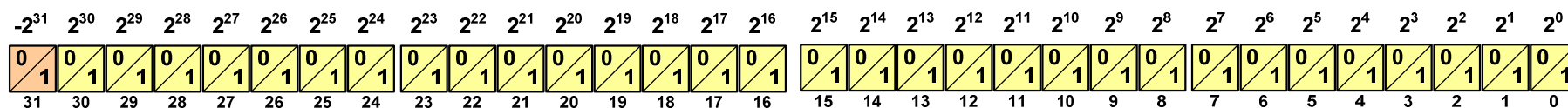
- dolna granica: $1000\ 0000_{(2)} = -128_{(10)}$
- górna granica: $0111\ 1111_{(2)} = 127_{(10)}$
- inne wartości: $1111\ 1111_{(2)} = -1_{(10)}$
 $0000\ 0000_{(2)} = 0_{(10)}$

Liczby całkowite bez znaku w języku C

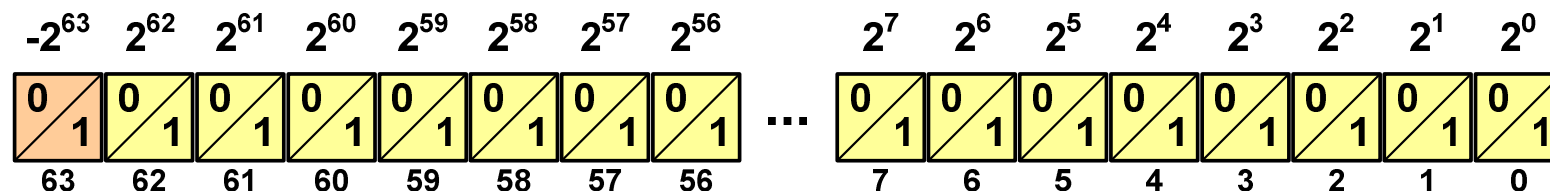
- Typ **short / signed short int** (2 bajty):



- Typy **int / signed int** (4 bajty) i **long / signed long int** (4 bajty):



- Typ **long long int / signed long long int** (8 bajtów):



Liczby całkowite ze znakiem - kod U2 w języku C

```
short int:      32767 -32768 -32767
int:            2147483647 -2147483648 -2147483647
long int:       2147483647 -2147483648 -2147483647
long long int:  9223372036854775807 -9223372036854775808
```

```
#include <stdio.h>

int main() /* przepełnienie zmiennej, ang. integer overflow */
{
    short int    si = 32767;
    int          i  = 2147483647;
    long int     li = 2147483647;
    long long int lli = 9223372036854775807;

    printf("short int:      %hd %hd %hd\n", si, si+1, si+2);
    printf("int:          %d %d %d\n", i, i+1, i+2);
    printf("long int:     %ld %ld %ld\n", li, li+1, li+2);
    printf("long long int: %lld %lld\n", lli, lli+1);

    return 0;
}
```

Koniec wykładu nr 3

Dziękuję za uwagę!
(następny wykład: 23.03.2018)