

Informatyka 1 (EZ1F1002)

Politechnika Białostocka - Wydział Elektryczny
Elektrotechnika, semestr I, studia niestacjonarne I stopnia
Rok akademicki 2023/2024

Wykład nr 3 (28.10.2023)

dr inż. Jarosław Forenc

Plan wykładu nr 3

- Systemy liczbowe
 - konwersje między systemami liczbowymi
- Jednostki informacji cyfrowej
 - bit, bajt
- Język C
 - instrukcja if, operatory relacyjne i logiczne, wyrażenia logiczne
- Kodowanie znaków i liczb
 - ASCII, ISO 8859, Unicode
 - NKB, BCD, kod Graya
- Reprezentacja liczb całkowitych
 - liczby bez znaku, liczby ze znakiem (ZM, U1)

Konwersja na system dziesiętny

- $p = 4, D = \{0, 1, 2, 3\}$

4^4	4^3	4^2	4^1	4^0
2	1	3	0	2

$$21302_{(4)} = ?_{(10)}$$

$$21302_{(4)} = 2 \cdot 4^0 + 0 \cdot 4^1 + 3 \cdot 4^2 + 1 \cdot 4^3 + 2 \cdot 4^4$$

$$21302_{(4)} = 2 \cdot 1 + 0 \cdot 4 + 3 \cdot 16 + 1 \cdot 64 + 2 \cdot 256$$

$$21302_{(4)} = 2 + 0 + 48 + 64 + 512 = 626_{(10)}$$

- $p = 17, D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G\}$

17^3	17^2	17^1	17^0
A	C	2	4

$$AC24_{(17)} = ?_{(10)}$$

$$AC24_{(17)} = 4 \cdot 17^0 + 2 \cdot 17^1 + 12 \cdot 17^2 + 10 \cdot 17^3$$

$$AC24_{(17)} = 4 \cdot 1 + 2 \cdot 17 + 12 \cdot 289 + 10 \cdot 4913$$

$$AC24_{(17)} = 4 + 34 + 3468 + 49130 = 52636_{(10)}$$

Konwersja z systemu dziesiętnego na dowolny

- zamiana liczby z systemu $p = 10$ na system $p = 2$

$$626_{(10)} = ?_{(2)}$$

$$626_{(10)} = 1001110010_{(2)}$$

$626/2 = 313$	<i>reszta</i>	0
$313/2 = 156$	<i>reszta</i>	1
$156/2 = 78$	<i>reszta</i>	0
$78/2 = 39$	<i>reszta</i>	0
$39/2 = 19$	<i>reszta</i>	1
$19/2 = 9$	<i>reszta</i>	1
$9/2 = 4$	<i>reszta</i>	1
$4/2 = 2$	<i>reszta</i>	0
$2/2 = 1$	<i>reszta</i>	0
$1/2 = 0$	<i>reszta</i>	1

kolejność odczytywania
cyfr liczby w systemie
dwójkowym

kończymy, gdy liczba dziesiętna ma wartość 0

Konwersja z systemu dziesiętnego na dowolny

- zamiana liczby z systemu $p = 10$ na system $p = 7$

$$626_{(10)} = ?_{(7)} \qquad 626_{(10)} = 1553_{(7)}$$

$626/7 = 89$	<i>reszta</i>	3	↑
$89/7 = 12$	<i>reszta</i>	5	
$12/7 = 1$	<i>reszta</i>	5	
$1/7 = 0$	<i>reszta</i>	1	

- zamiana liczby z systemu $p = 10$ na system $p = 14$

$$626_{(10)} = ?_{(14)} \qquad 626_{(10)} = 32A_{(14)}$$

$626/14 = 44$	<i>reszta</i>	10	→ A	↑
$44/14 = 3$	<i>reszta</i>	2		
$3/14 = 0$	<i>reszta</i>	3		

Szybkie konwersje: $2 \rightarrow 4, 8, 16$ $4, 8, 16 \rightarrow 2$

$2 \rightarrow 4$

$$110110011_{(2)} = ?_{(4)}$$
$$\underbrace{01}_1 \mid \underbrace{10}_2 \mid \underbrace{11}_3 \mid \underbrace{00}_0 \mid \underbrace{11}_3$$
$$110110011_{(2)} = 12303_{(4)}$$

$2 \rightarrow 8$

$$10110011_{(2)} = ?_{(8)}$$
$$\underbrace{010}_2 \mid \underbrace{110}_6 \mid \underbrace{011}_3$$
$$10110011_{(2)} = 263_{(8)}$$

$2 \rightarrow 16$

$$1011010_{(2)} = ?_{(16)}$$
$$\underbrace{0101}_5 \mid \underbrace{1010}_A$$
$$1011010_{(2)} = 5A_{(16)}$$

$4 \rightarrow 2$

$$12303_{(4)} = ?_{(2)}$$
$$\underbrace{01}_1 \mid \underbrace{10}_2 \mid \underbrace{11}_3 \mid \underbrace{00}_0 \mid \underbrace{11}_3$$
$$12303_{(4)} = 110110011_{(2)}$$

$8 \rightarrow 2$

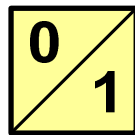
$$263_{(8)} = ?_{(2)}$$
$$\underbrace{010}_2 \mid \underbrace{110}_6 \mid \underbrace{011}_3$$
$$263_{(8)} = 10110011_{(2)}$$

$16 \rightarrow 2$

$$5A_{(16)} = ?_{(2)}$$
$$\underbrace{0101}_5 \mid \underbrace{1010}_A$$
$$5A_{(16)} = 1011010_{(2)}$$

Jednostki informacji - bit

- **Bit** (ang. **binary digit**) - podstawowa jednostka informacji stosowana w informatyce i telekomunikacji
- Określa najmniejszą ilość informacji potrzebną do stwierdzenia, który z dwóch możliwych stanów przyjął układ
- Bit przyjmuje jedną z dwóch wartości:
 - 0 (zero)
 - 1 (jeden)
- Bit jest tożsamy z cyfrą w systemie dwójkowym
- Oznaczenia bitów:
 - standard IEEE 1541 (2002) - mała litera „b”
 - standard IEC 60027 - „bit”



Jednostki informacji - bit

■ Wielokrotności bitów:

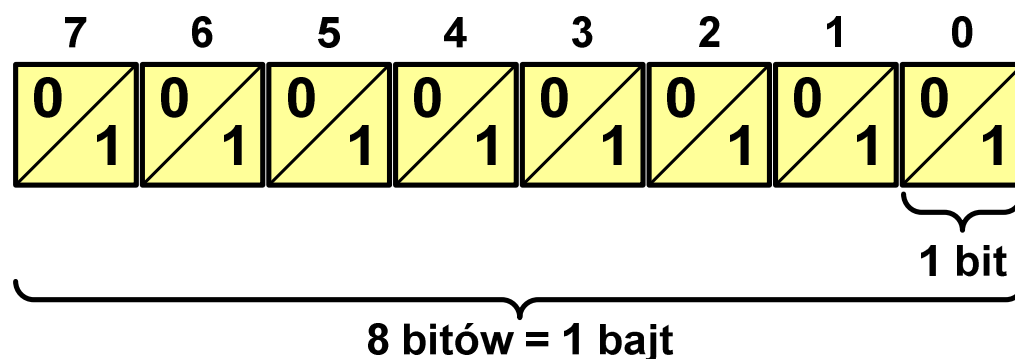
Przedrostki dziesiętne (układ SI)		
Nazwa	Symbol	Mnożnik
bit	b	---
kilobit	kb	$10^3 = 1000^1$
megabit	Mb	$10^6 = 1000^2$
gigabit	Gb	$10^9 = 1000^3$
terabit	Tb	$10^{12} = 1000^4$
petabit	Pb	$10^{15} = 1000^5$
eksabit	Eb	$10^{18} = 1000^6$
zettabit	Zb	$10^{21} = 1000^7$
jottabit	Yb	$10^{24} = 1000^8$

Przedrostki binarne (IEC 60027-2)		
Nazwa	Symbol	Mnożnik
bit	b	---
kibibit	Kib	$2^{10} = 1024^1$
mebibit	Mib	$2^{20} = 1024^2$
gibibit	Gib	$2^{30} = 1024^3$
tebibit	Tib	$2^{40} = 1024^4$
pebibit	Pib	$2^{50} = 1024^5$
eksbibit	Eib	$2^{60} = 1024^6$
zebibit	Zib	$2^{70} = 1024^7$
jobibit	Yib	$2^{80} = 1024^8$

- **Przedrostki binarne** - wprowadzone w 1999 roku w celu odróżnienia przedrostków o mnożniku 1000 (10^3) od przedrostków o mnożniku 1024 (2^{10})

Jednostki informacji - bajt

- **Bajt** (ang. byte) - najmniejsza adresowalna jednostka informacji pamięci komputerowej składająca się z bitów
- W praktyce przyjmuje się, że jeden bajt to 8 bitów

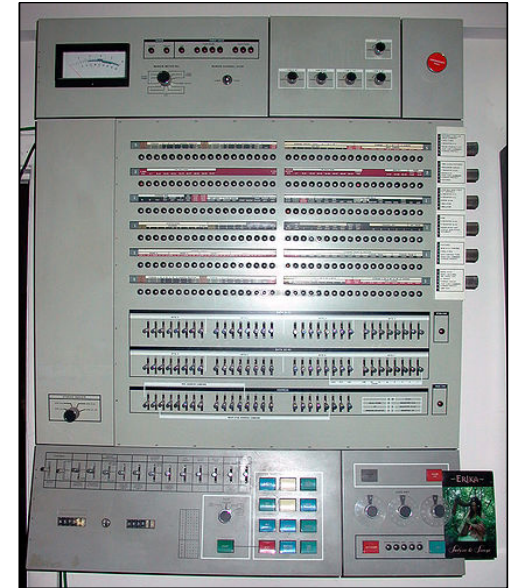


- Za pomocą jednego bajtu można zapisać $2^8 = 256$ różnych wartości:

0000 0000	→	0
0000 0001	→	1	1111 1101	→ 253
0000 0010	→	2	1111 1110	→ 254
...		...	1111 1111	→ 255

Jednostki informacji - bajt

- W pierwszych komputerach bajt mógł mieć inną liczbę bitów: 4, 6, 7, 9, 12
- 8-bitowy bajt:
 - koniec 1956 r. - pierwsze zastosowanie
 - 1964 r. - uznanie za standard (IBM System/360)
- Inna nazwa 8-bitowego bajtu - **oktet**
- Najczęściej stosowanym skrótem dla bajtu jest wielka litera „**B**”
 - „**B**” używane jest także do oznaczania **bela** - jednostki miary wielkości ilorazowych
 - zamiast bela częściej używa się jednostki podwielokrotnej - **decybela (dB)** więc nie ma problemu z rozróżnieniem obu jednostek



Jednostki informacji - bajt

■ Wielokrotności bajtów:

Przedrostki dziesiętne (układ SI)		
Nazwa	Symbol	Mnożnik
bajt	B	---
kilobajt	kB	$10^3 = 1000^1$
megabajt	MB	$10^6 = 1000^2$
gigabajt	GB	$10^9 = 1000^3$
terabajt	TB	$10^{12} = 1000^4$
petabajt	PB	$10^{15} = 1000^5$
eksabajt	EB	$10^{18} = 1000^6$
zettabajt	ZB	$10^{21} = 1000^7$
jottabajt	YB	$10^{24} = 1000^8$

Przedrostki binarne (IEC 60027-2)		
Nazwa	Symbol	Mnożnik
bajt	B	---
kibibajt	KiB	$2^{10} = 1024^1$
mebibajt	MiB	$2^{20} = 1024^2$
gibibajt	GiB	$2^{30} = 1024^3$
tebibajt	TiB	$2^{40} = 1024^4$
pebibajt	PiB	$2^{50} = 1024^5$
eksbibajt	EiB	$2^{60} = 1024^6$
zebibajt	ZiB	$2^{70} = 1024^7$
jobibajt	YiB	$2^{80} = 1024^8$

Jednostki informacji - bajt

- Przedrostki binarne (dwójkowe) nie zostały przyjęte przez wszystkie środowiska zajmujące się informatyką
- Producenci nośników pamięci korzystają z przedrostków dziesiętnych

Prefiks	Nazwa	System SI	System binarny	Różnica
k	kilo	$10^3 = 1000$	$2^{10} = 1024$	2,40%
M	mega	$10^6 = 1000000$	$2^{20} = 1048576$	4,86%
G	giga	$10^9 = 1000000000$	$2^{30} = 1073741824$	7,37%
T	tera	$10^{12} = 1000000000000$	$2^{40} = 1099511627776$	9,95%

- Z ulotki „Dysk Desktop HDD - zestawienie danych”, Seagate:
 - w przypadku oznaczania pojemności dysków, jeden gigabajt (oznaczany także jako „GB”) jest równy jednemu miliardowi bajtów, a jeden terabajt (oznaczany także jako „TB”) jest równy jednemu bilionowi bajtów

Jednostki informacji - bajt

- Seagate ST1000DM003 (1 TB)
- Drive specification:
 - formatted capacity: 1000 GB (1 TB)
 - guaranteed sectors: 1,953,525,168
 - bytes per sector: 4096
(4K physical emulated at 512-byte sectors)
- Pojemność dysku:
 - $1.953.525.168 \times 512 = 1.000.204.886.016$ bajtów
 - $1.000.204.886.016 / (1024) = 976.762.584$ kB
 - $1.000.204.886.016 / (1024 \times 1024) = 953.870$ MB
 - $1.000.204.886.016 / (1024 \times 1024 \times 1024) = 931,5$ GB



Przykład: 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

Przykład: 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
>	<code>a > b</code>	<code>a</code> większe od <code>b</code>
<	<code>a < b</code>	<code>a</code> mniejsze od <code>b</code>
>=	<code>a >= b</code>	<code>a</code> większe lub równe <code>b</code>
<=	<code>a <= b</code>	<code>a</code> mniejsze lub równe <code>b</code>
==	<code>a == b</code>	<code>a</code> równe <code>b</code>
!=	<code>a != b</code>	<code>a</code> nierówne <code>b</code> (<code>a</code> różne od <code>b</code>)

- 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 )
```

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

wynik: 0 (fałsz)

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

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

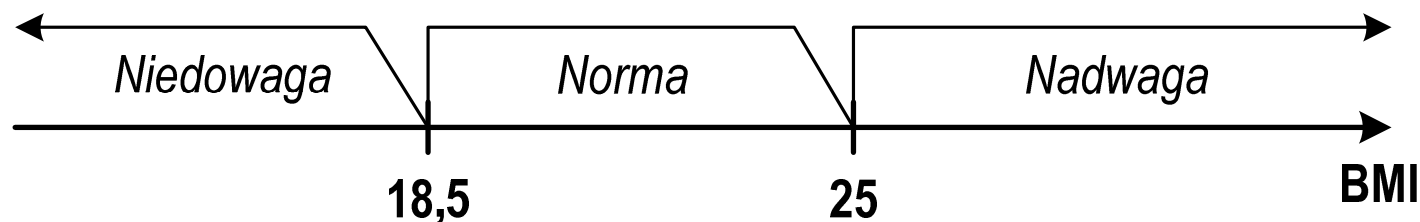
wynik: 1 (prawda) (!!!)

Przykład: obliczanie BMI (Body Mass Index)

- **BMI** - współczynnik powstały przez podzielenie **masy** ciała podanej w kilogramach przez **kwadrat wzrostu** podanego w metrach

$$BMI = \frac{masa}{wzrost^2}$$

- Dla osób dorosłych:
 - BMI < 18,5 - wskazuje na niedowagę
 - BMI ≥ 18,5 i BMI < 25 - wskazuje na prawidłową masę ciała
 - BMI ≥ 25 - wskazuje na nadwagę



Przykład: obliczanie BMI (Body Mass Index)

```
#include <stdio.h>

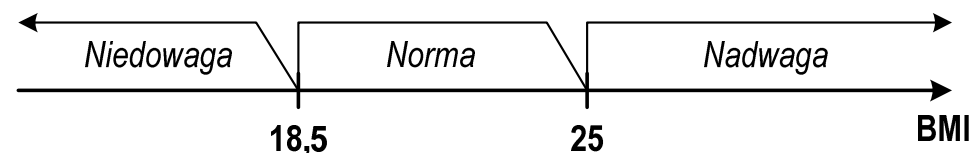
int main(void)
{
    double masa, wzrost, bmi;

    printf("Podaj mase [kg]: "); scanf("%lf", &masa);
    printf("Podaj wzrost [m]: "); scanf("%lf", &wzrost);
    bmi = masa / (wzrost*wzrost);
    printf("bmi: %.2f\n", bmi);

    if (bmi < 18.5)
        printf("Niedowaga\n");
    if (bmi >= 18.5 && bmi < 25)
        printf("Norma\n");
    if (bmi >= 25)
        printf("Nadwaga\n");

    return 0;
}
```

```
Podaj mase [kg]: 84
Podaj wzrost [m]: 1.85
bmi: 24.54
Norma
```



Przykład: obliczanie BMI (Body Mass Index)

- Zamiast trzech instrukcji `if`:

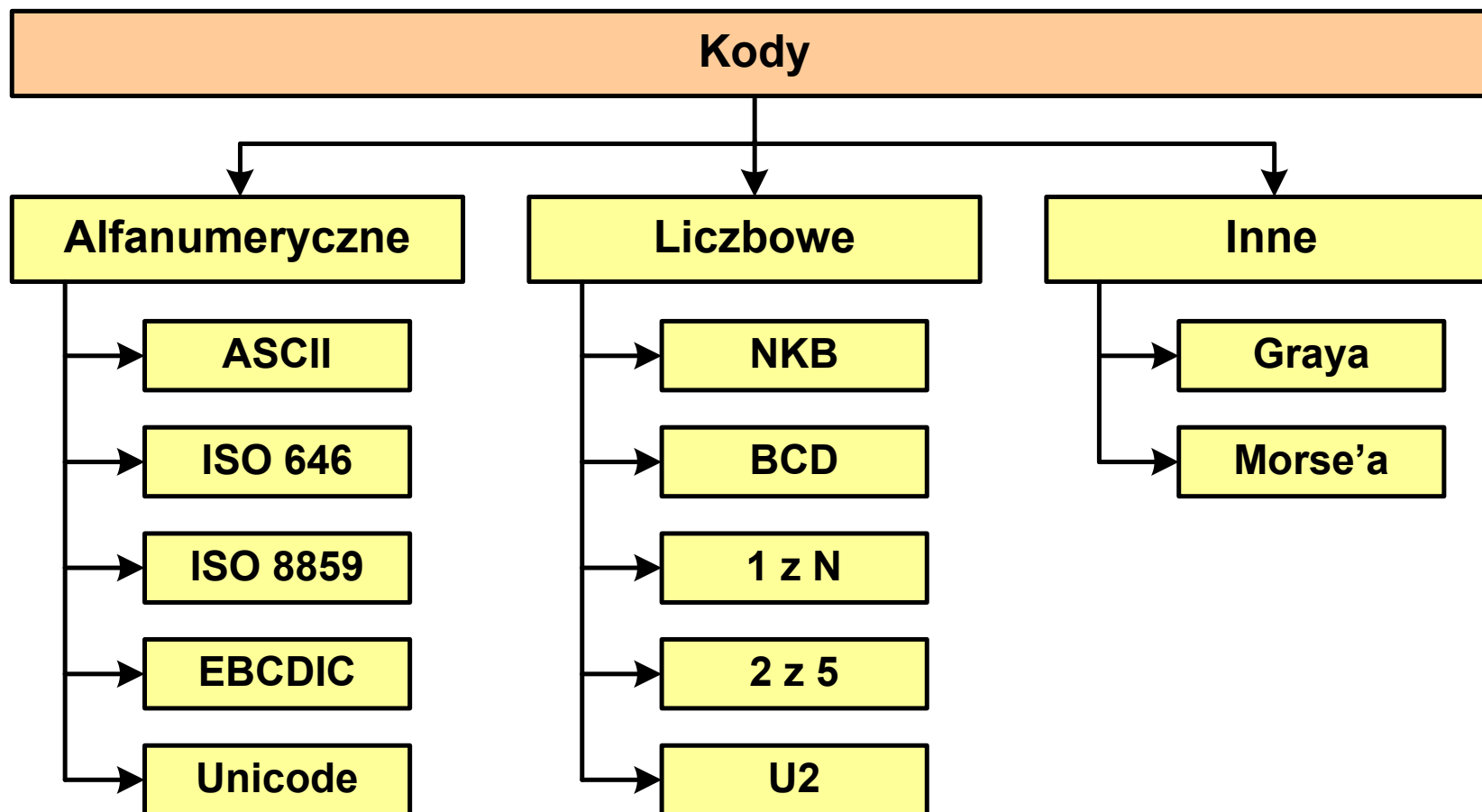
```
if (bmi<18.5)
    printf("Niedowaga\n");
if (bmi>=18.5 && bmi<25)
    printf("Norma\n");
if (bmi>=25)
    printf("Nadwaga\n");
```

można zastosować tylko dwie:

```
if (bmi<18.5)
    printf("Niedowaga\n");
else
    if (bmi<25)
        printf("Norma\n");
    else
        printf("Nadwaga\n");
```

Kodowanie

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



Kod ASCII

■ ASCII - American Standard Code for Information Interchange

- 7-bitowy kod przypisujący liczby z zakresu 0-127:
 - literom (alfabet angielski)
 - cyfrom
 - znakom przestankowym
 - innym symbolom
 - poleceniom sterującym.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	NUL	32	20	Space	64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	\$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	\	71	47	G	103	67	g
8	8	BS	40	28	(72	48	H	104	68	h
9	9	TAB	41	29)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	
29	1D	GS	61	3D	=	93	5D]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Kod ASCII - Kody sterujące

- Kody sterujące - 33 kody, o numerach: 0-31, 127

Dec	Hex	Char	Dec	Hex	Char
0	0	NUL (null)	16	10	DLE (data link escape)
1	1	SOH (start of heading)	17	11	DC1 (device control 1)
2	2	STX (start of text)	18	12	DC2 (device control 2)
3	3	ETX (end of text)	19	13	DC3 (device control 3)
4	4	EOT (end of transmission)	20	14	DC4 (device control 4)
5	5	ENQ (enquiry)	21	15	NAK (negative acknowledge)
6	6	ACK (acknowledge)	22	16	SYN (synchronous idle)
7	7	BEL (bell)	23	17	ETB (end of trans. block)
8	8	BS (backspace)	24	18	CAN (cancel)
9	9	TAB (horizontal tab)	25	19	EM (end of medium)
10	A	LF (NL line feed, new line)	26	1A	SUB (substitute)
11	B	VT (vertical tab)	27	1B	ESC (escape)
12	C	FF (NP form feed, new page)	28	1C	FS (file separator)
13	D	CR (carriage return)	29	1D	GS (group separator)
14	E	SO (shift out)	30	1E	RS (record separator)
15	F	SI (shift in)	31	1F	US (unit separator)
			127	7F	DEL

- W języku C:

0 (NULL) - `\0`

7 (BEL) - `\a`

8 (BS) - `\b`

9 (TAB) - `\t`

10 (LF) - `\n`

13 (CR) - `\r`

Kod ASCII - Pliki tekstowe

- Elementami pliku tekstowego są **wiersze**, mogą one mieć różną długość
- W systemie Windows każdy wiersz pliku zakończony jest parą znaków:
 - **CR**, ang. carriage return - powrót karetki, kod ASCII - $13_{(10)} = 0D_{(16)}$
 - **LF**, ang. line feed - przesunięcie o wiersz, kod ASCII - $10_{(10)} = 0A_{(16)}$

- Załóżmy, że plik tekstowy ma postać:

```
Pierwszy wiersz pliku
Drugi wiersz pliku
Trzeci wiersz pliku
```

- Rzeczywista zawartość pliku jest następująca:

```
00000000: 50 69 65 72 77 73 7A 79|20 77 69 65 72 73 7A 20 | Pierwszy wiersz
00000010: 70 6C 69 6B 75 0D 0A 44|72 75 67 69 20 77 69 65 | pliku■■Drugi wie
00000020: 72 73 7A 20 70 6C 69 6B|75 0D 0A 54 72 7A 65 63 | rsz pliku■■Trzec
00000030: 69 20 77 69 65 72 73 7A|20 70 6C 69 6B 75 0D 0A | i wiersz pliku■■
```

- Wydruk zawiera:
 - przesunięcie od początku pliku (szesnastkowo)
 - wartości poszczególnych bajtów pliku (szesnastkowo)
 - znaki odpowiadające bajtom pliku (traktując bajty jako kody ASCII)

Kod ASCII - Pliki tekstowe

- W systemie Linux znakiem końca wiersza jest tylko **LF** o kodzie ASCII - $10_{(10)} = 0A_{(16)}$

- Założmy, że plik tekstowy ma postać:

```
Pierwszy wiersz pliku
Drugi wiersz pliku
Trzeci wiersz pliku
```

- Rzeczywista zawartość pliku jest następująca:

```
00000000: 50 69 65 72 77 73 7A 79|20 77 69 65 72 73 7A 20 | Pierwszy wiersz
00000010: 70 6C 69 6B 75 0A 44 72|75 67 69 20 77 69 65 72 | plikuDrugi wier
00000020: 73 7A 20 70 6C 69 6B 75|0A 54 72 7A 65 63 69 20 | sz plikuTrzeci
00000030: 77 69 65 72 73 7A 20 70|6C 69 6B 75 0A | wiersz pliku
```

- Podczas przesyłania pliku tekstowego (np. przez protokół **ftp**) z systemu Linux do systemu Windows pojedynczy znak **LF** zamieniany jest automatycznie na parę znaków **CR** i **LF**
- Błędne przesłanie pliku tekstowego (w trybie binarnym) powoduje nieprawidłowe jego wyświetlanie:

```
Pierwszy wiersz plikuDrugi wiersz plikuTrzeci wiersz pliku
```

ISO/IEC 8859

- **ISO/IEC 8859** - zestaw standardów służących do kodowania znaków za pomocą 8-bitów
- Wszystkie zestawy ISO 8859 mają znaki $0_{(10)}-127_{(10)}$ ($00_{(16)}-7F_{(16)}$) takie same jak w kodzie ASCII
- Pozycjom $128_{(10)}-159_{(10)}$ ($80_{(16)}-9F_{(16)}$) przypisane są dodatkowe kody sterujące, tzw. C1 (obecnie nie są używane)
- Od czerwca 2004 roku ISO 8859 nie jest rozwijane.

ISO/IEC 8859

■ Stosowane standardy ISO 8859:

- ISO 8859-1 (Latin-1) - alfabet łaciński dla Europy zachodniej
- ISO 8859-2 (Latin-2) - łaciński dla Europy środkowej i wschodniej
- ISO 8859-3 (Latin-3) - łaciński dla Europy południowej
- ISO 8859-4 (Latin-4) - łaciński dla Europy północnej
- ISO 8859-5 (Cyrillic) - dla cyrylicy
- ISO 8859-6 (Arabic) - dla alfabetu arabskiego
- ISO 8859-7 (Greek) - dla alfabetu greckiego
- ISO 8859-8 (Hebrew) - dla alfabetu hebrajskiego
- ISO 8859-9 (Latin-5)
- ISO 8859-10 (Latin-6)
- ISO 8859-11 (Thai) - dla alfabetu tajskiego
- ISO 8859-12 - brak
- ISO 8859-13 (Latin-7)
- ISO 8859-14 (Latin-8) - zawiera polskie litery
- ISO 8859-15 (Latin-9)
- ISO 8859-16 (Latin-10) - łaciński dla Europy środkowej, zawiera polskie litery

ISO/IEC 8859-2

- ISO/IEC 8859-2, Latin-2 („środkowo”, „wschodnioeuropejskie”)
- dostępne języki: bośniacki, chorwacki, czeski, węgierski, polski, rumuński, serbski, serbsko-chorwacki, słowacki, słoweński, górno- i dolnołużycki
- możliwość przedstawienia znaków w języku niemieckim i angielskim
- 191 znaków łacińskiego pisma
- do 02.11.2015 kodowanie to było zgodne z **Polską Normą**

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	<i>Znaki kontrolne</i>															
10																
20	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
80	<i>Nie używane</i>															
90																
A0	NB SP	À	Á	Â	Ã	Ä	Å	Æ	Ç	Ð	Ñ	Ò	Ó	Ô	Õ	Ö
B0	°	à	á	â	ã	ä	å	æ	ç	ð	ñ	ò	ó	ô	õ	ö
C0	í	ä	å	æ	ç	ð	ñ	ò	ó	ô	õ	ö	÷	ř	ů	ú
D0	đ	ń	ň	ó	ô	õ	ö	÷	ř	ů	ú	ű	ü	ý	ț	·
E0	í	ä	å	æ	ç	ð	ñ	ò	ó	ô	õ	ö	÷	ř	ů	ú
F0	đ	ń	ň	ó	ô	õ	ö	÷	ř	ů	ú	ű	ü	ý	ț	·

SP - spacja
NBSP - twarda spacja
SHY - miękki dywiz (myślnik)

ISO/IEC 8859-2 - Litery diakrytyczne w j. polskim

■ 18 liter:

- Ā - ā
- Ć - ć
- Ę - ę
- Ł - ł
- Ń - ń
- Ó - ó
- Ś - ś
- Ź - ź
- Ż - ż

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	<i>Znaki kontrolne</i>															
10																
20																
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
80	<i>Nie używane</i>															
90																
A0	NB SP	Ā	˘	Ł	ł	Ĺ	Ś	ś	˙	Š	š	Ť	Ž	SHY	Ž	Ž
B0	°	ā	˘	ł	ł	ĺ	ś	ś	˙	š	š	ť	ž	ˆ	ž	ž
C0	Ř	Á	Â	Ă	Ä	Í	Ć	Ç	Č	É	Ę	Ë	Ě	Í	Î	Ď
D0	Đ	Ń	Ň	Ó	Ô	Õ	Ö	×	Ř	Ů	Ú	Ů	Ü	Ý	Ŧ	ß
E0	ř	á	â	ă	ä	í	ć	ç	č	é	ę	ë	ě	í	î	ď
F0	đ	ń	ň	ó	ô	õ	ö	÷	ř	ů	ú	ů	ü	ý	ț	·



Unicode (Unikod)

- Komputerowy zestaw znaków mający obejmować wszystkie pisma i inne znaki (symbole techniczne, wymowy) używane na świecie
- Unicode przypisuje unikalny numer każdemu znakowi, niezależny od używanej platformy, programu czy języka
- Konsorcjum: <http://www.unicode.org>
- Pierwsza wersja: **Unicode 1.0** (październik 1991)
- Ostatnia wersja: **Unicode 15.1.0** (12 września 2023)
 - The Unicode Consortium. The Unicode Standard, Version 15.1.0, (South San Francisco, CA: The Unicode Consortium, 2023. ISBN 978-1-936213-33-7)
 - <https://www.unicode.org/versions/Unicode15.1.0/>
 - Koduje 149.813 znaków



Unicode - Zakresy

<u>Zakres:</u>	<u>Znaczenie:</u>
U+0000 - U+007F	Basic Latin (to samo co w ASCII)
U+0080 - U+00FF	Latin-1 Supplement (to samo co w ISO/IEC 8859-1)
U+0100 - U+017F	Latin Extended-A
U+0180 - U+024F	Latin Extended-B
U+0250 - U+02AF	IPA Extensions
U+02B0 - U+02FF	Spacing Modifiers Letters
...	
U+0370 - U+03FF	Greek
U+0400 - U+04FF	Cyrillic
...	
U+1D00 - U+1D7F	Phonetic Extensions
U+1D80 - U+1DBF	Phonetic Extensions Supplement
U+1E00 - U+1EFF	Latin Extended Additional
U+1F00 - U+1FFF	Greek Extended
...	



Unicode

- Standard Unicode definiuje nie tylko kody numeryczne przypisane poszczególnym znakom, ale także określa sposób bajtowego **kodowania** znaków
- Kodowanie określa sposób w jaki znaki ze zbioru mają być zapisane w **postaci binarnej**
- Istnieją trzy podstawowe metody kodowania:
 - 32-bitowe: UTF-32
 - 16-bitowe: UTF-16
 - 8-bitowe: UTF-8
- **UTF** - UCS Transformation Format, **UCS** - Universal Character Set
- Metody kodowania różnią się liczbą bajtów przeznaczonych do opisanie kodu znaku
- Wszystkie metody obejmują wszystkie kodowane znaki w Unicode.



Unicode - kodowanie UTF-32

- **UTF-32** - sposób kodowania standardu Unicode wymagający użycia 32-bitowych słów

A	Ω	語	卍
00000041	000003A9	00008A9E	00010384

UTF-32

- Kod znaku ma zawsze stałą długość 4 bajtów i przedstawia numer znaku w tabeli Unikodu
- Kody obejmują zakres od 0 do 0x10FFFF (od 0 do 1 114 111)
- Kodowanie to jest jednak bardzo nieefektywne - zakodowane ciągi znaków są 2-4 razy dłuższe niż ciągi tych samych znaków zapisanych w innych kodowaniach.



Unicode - kodowanie UTF-16

- **UTF-16** - sposób kodowania standardu Unicode wymagający użycia 16-bitowych słów

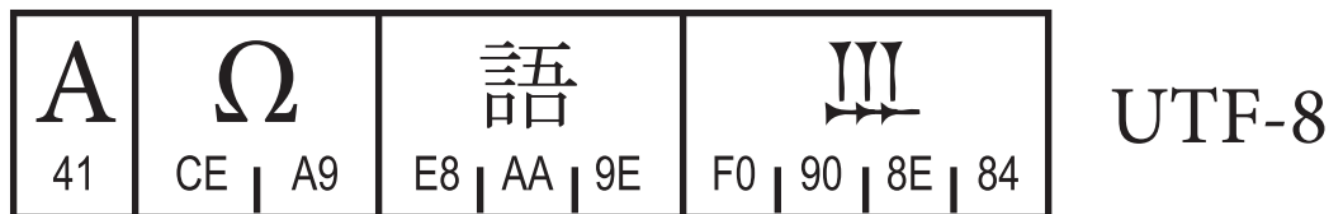


- Dla znaków z przedziału od **U+0000** do **U+FFFF** używane jest jedno słowo, którego wartość jest jednocześnie kodem znaku w Unicode
- Dla znaków z wyższych pozycji używa się dwóch słów:
 - pierwsze słowo należy do przedziału: **U+D800 - U+DBFF**
 - drugie słowo należy do przedziału: **U+DC00 - U+DFFF**.



Unicode - kodowanie UTF-8

- **UTF-8** - kodowanie ze zmienną długością reprezentacji znaku wymagające użycia 8-bitowych słów



- Znaki Unikodu są mapowane na ciągi bajtów
 - 0x00 do 0x7F - bity 0xxxxxxx
 - 0x80 do 0x7FF - bity 110xxxxx 10xxxxxx
 - 0x800 do 0xFFFF - bity 1110xxxx 10xxxxxx 10xxxxxx
 - 0x10000 do 0x1FFFFF - bity 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
 - 0x200000 do 0x3FFFFFFF - bity 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
 - 0x4000000 do 0x7FFFFFFF - bity 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx



Unicode

	010	011	012	013	014	015	016	017
0	Ā 0100	Ð 0110	Ġ 0120	İ 0130	ı 0140	Ō 0150	Š 0160	Ū 0170
1	ā 0101	đ 0111	ġ 0121	ı 0131	Ł 0141	ō 0151	š 0161	ū 0171
2	Ǻ 0102	Ē 0112	Ɔ 0122	IJ 0132	ł 0142	Œ 0152	Ț 0162	Ț 0172
3	ǻ 0103	ē 0113	ġ 0123	ij 0133	Ń 0143	œ 0153	ț 0163	Ț 0173
4	Ą 0104	Ě 0114	Ĥ 0124	Ĵ 0134	ń 0144	Ŕ 0154	Ť 0164	Ŵ 0174
5	ą 0105	ě 0115	ĥ 0125	ĵ 0135	Ń 0145	ŕ 0155	ť 0165	ŵ 0175
6	Ć 0106	Ę 0116	Ħ 0126	Ɔ 0136	ņ 0146	Ŗ 0156	Ʀ 0166	Ŷ 0176
7	ć 0107	ę 0117	ħ 0127	Ɔ 0137	Ņ 0147	ŗ 0157	ƣ 0167	ŷ 0177

European Latin

- 0100 Ā LATIN CAPITAL LETTER A WITH MACRON
≡ 0041 A 0304 ̄
- 0101 ā LATIN SMALL LETTER A WITH MACRON
• Latvian, Latin, ...
≡ 0061 a 0304 ̄
- 0102 Ă LATIN CAPITAL LETTER A WITH BREVE
≡ 0041 A 0306 ̆
- 0103 ă LATIN SMALL LETTER A WITH BREVE
• Romanian, Vietnamese, Latin, ...
≡ 0061 a 0306 ̆
- 0104 Ą LATIN CAPITAL LETTER A WITH OGONEK
≡ 0041 A 0328 ̇
- 0105 ą LATIN SMALL LETTER A WITH OGONEK
• Polish, Lithuanian, ...
≡ 0061 a 0328 ̇
- 0106 Ć LATIN CAPITAL LETTER C WITH ACUTE
≡ 0043 C 0301 ́
- 0107 ć LATIN SMALL LETTER C WITH ACUTE
• Polish, Croatian, ...
→ 045B ħ cyrillic small letter tshe
≡ 0063 c 0301 ́

Unicode



27308

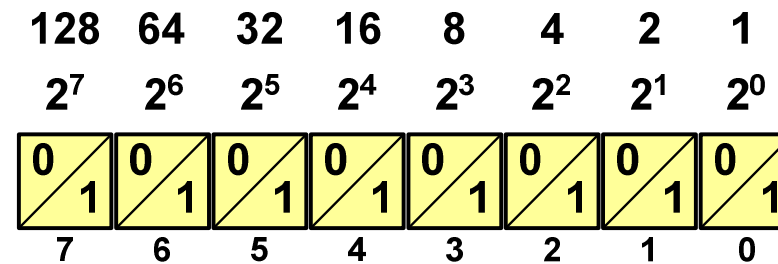
CJK Unified Ideographs Extension B

27342

27308 虫 142.8	𧈧	𧈧	𧈧	2731B 虫 142.8	𧈧	𧈧	𧈧	2732F 虫 142.8	𧈧	𧈧
	UCS2003	GKX-1086.03	T4-4721		UCS2003	GKX-1088.15	T6-617B		UCS2003	GHC
27309 虫 142.8	𧈩	𧈩	𧈩	2731C 虫 142.8	𧈩	𧈩	𧈩	27330 虫 142.9	𧈩	𧈩
	UCS2003	GKX-1086.05	T5-4955		UCS2003	GKX-1088.16	T6-6221		UCS2003	GHC
2730A 虫 142.8	𧈪	𧈪	𧈪	2731D 虫 142.8	𧈪	𧈪	𧈪	27331 虫 142.8	𧈪	𧈪
	UCS2003	GKX-1086.08	T4-467D		UCS2003	GKX-1088.17	T5-4960		UCS2003	G4K
2730B 虫 142.8	𧈫	𧈫	𧈫	2731E 虫 142.7	𧈫	𧈫		27332 虫 142.8	𧈫	𧈫
	UCS2003	GKX-1086.10	T6-6223		UCS2003	GKX-1088.18			UCS2003	GHC
2730C 虫 142.8	𧈬	𧈬	𧈬	2731F 虫 142.8	𧈬	𧈬	𧈬	27333 虫 142.8	𧈬	𧈬
	UCS2003	GKX-1086.12	T5-495F		UCS2003	GKX-1088.19	T6-6174		UCS2003	GHC
2730D 虫 142.8	𧈭	𧈭	𧈭	27320 虫 142.8	𧈭	𧈭	𧈭	27334 虫 142.8	𧈭	𧈭
	UCS2003	GKX-1086.22	T4-4677		UCS2003	GKX-1088.20	T6-617D		UCS2003	T5-4953

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 $0 \div 9$
- Pozostałe kombinacje bitowe mogą być stosowane do kodowania znaku liczby lub innych znaczników.

Kody liczbowe - Kod BCD

■ Przykład:

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

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

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

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

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

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

■ 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)

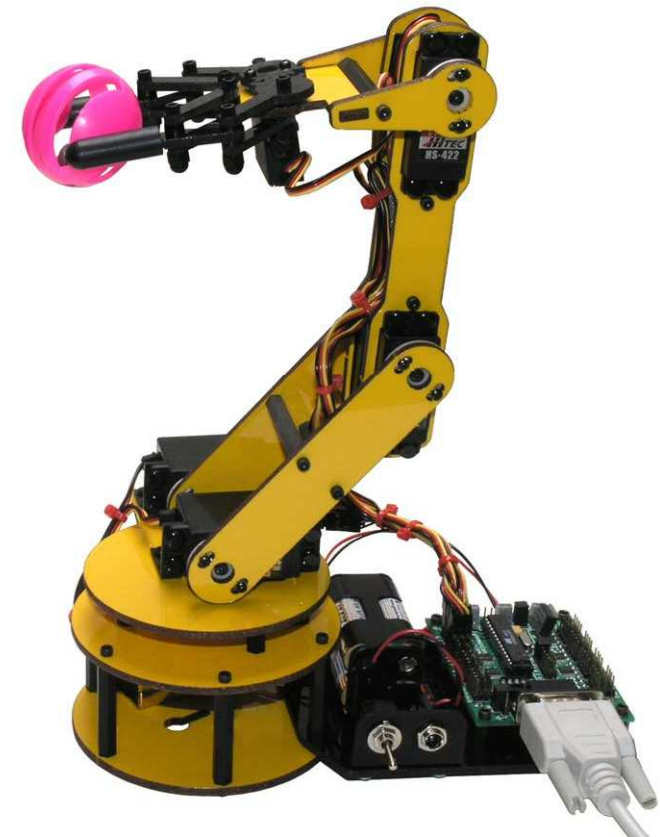
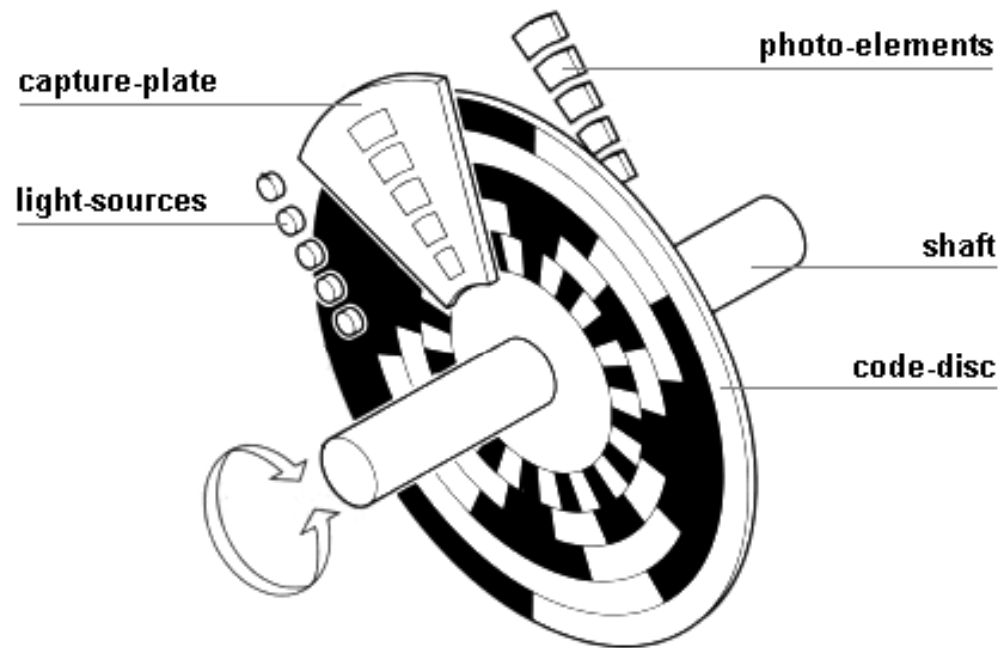
Kod Graya (refleksyjny)

- 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

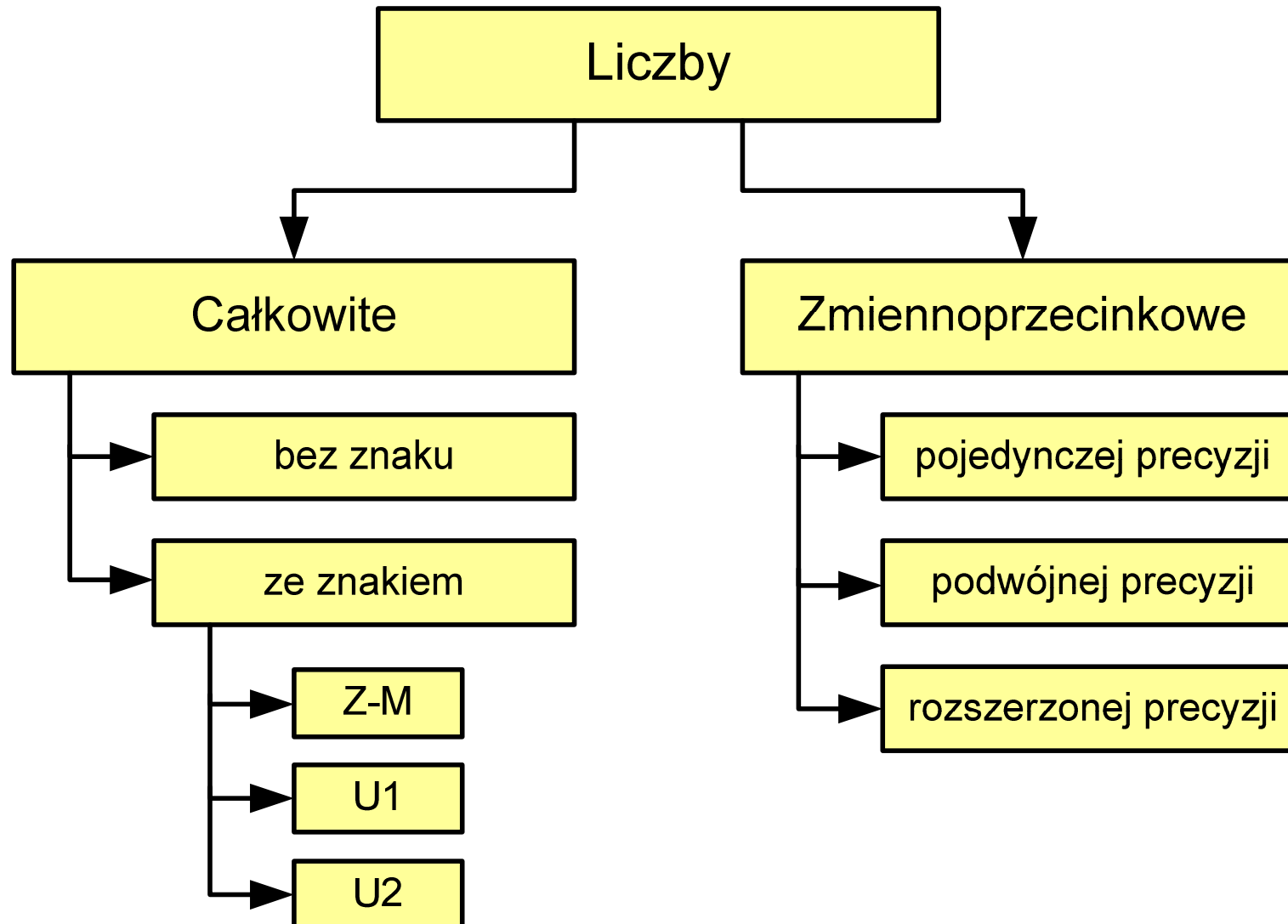
<u>kod 1-bitowy</u>	<u>kod 2-bitowy</u>	<u>kod 3-bitowy</u>
0	00	000
1	01	001
	<u>11</u>	011
	10	010
		<u>110</u>
		111
		101
		100

Kod Graya

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

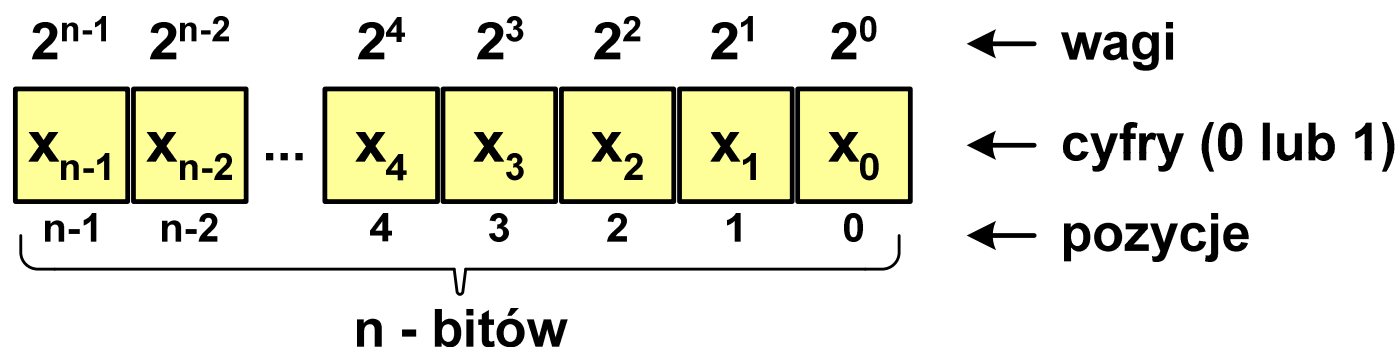


Reprezentacja liczb w systemach komputerowych



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 biliardów 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:

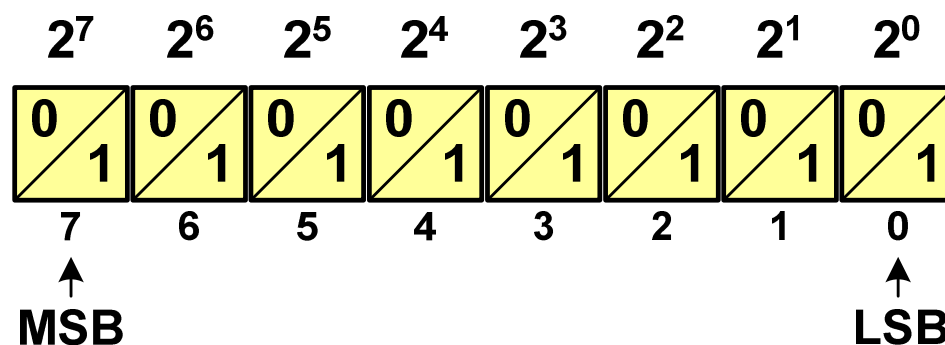
<u>Nazwa typu</u>	<u>Rozmiar (bajty)</u>	<u>Zakres wartości</u>
<code>unsigned char</code>	1 bajt	0 ... 255
<code>unsigned short int</code>	2 bajty	0 ... 65 535
<code>unsigned int</code>	4 bajty	0 ... 4 294 967 295
<code>unsigned long int</code>	4 bajty	0 ... 4 294 967 295
<code>unsigned long long int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

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

<code>unsigned short int</code>	→	<code>unsigned short</code>
<code>unsigned long int</code>	→	<code>unsigned long</code>
<code>unsigned long long int</code>	→	<code>unsigned long long</code>

Liczby całkowite bez znaku w języku C

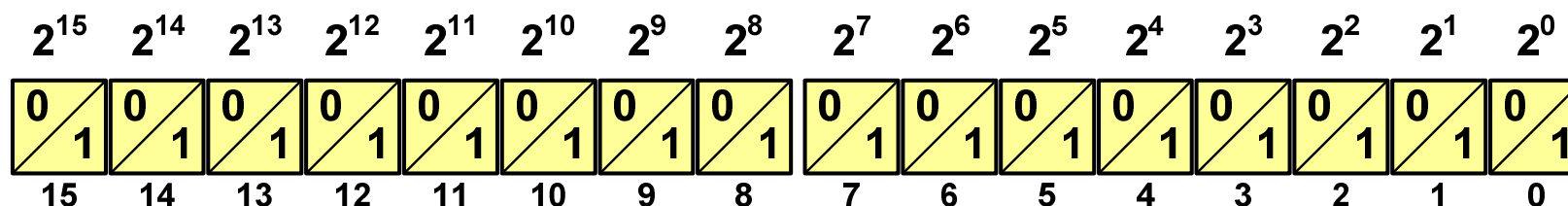
- Typ **unsigned char** (1 bajt):



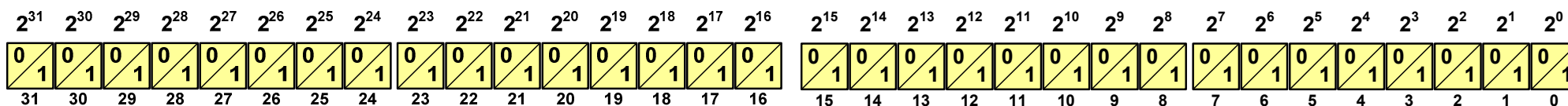
- **MSB** (Most Significant Bit) - najbardziej znaczący bit, najstarszy bit, największa waga
 - **LSB** (Least Significant Bit) - najmniej znaczący bit, najmłodszy bit, najmniejsza waga
- Zakres wartości:
 - dolna granica: $0000\ 0000_{(2)} = 00_{(16)} = 0_{(10)}$
 - górna granica: $1111\ 1111_{(2)}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

Liczby całkowite bez znaku w języku C

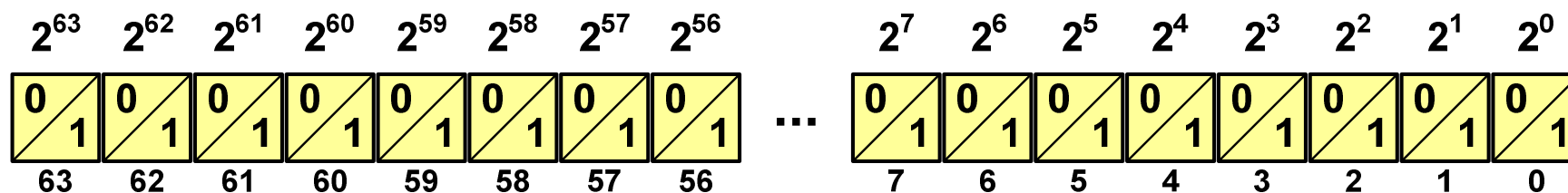
- Typ **unsigned short int** (2 bajty):



- 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>

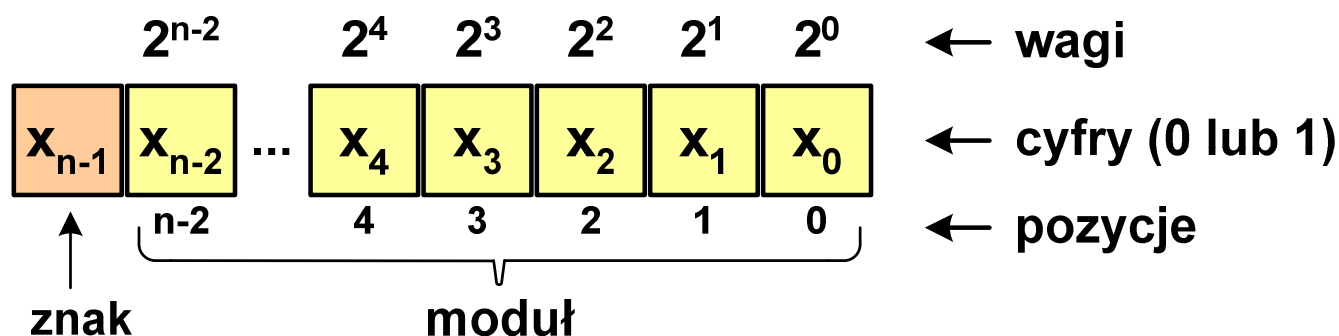
int main() /* przepełnienie zmiennej, ang. integer overflow */
{
    unsigned short int    usi = 1;
    unsigned int          ui = 1;
    unsigned long int     uli = 1;
    unsigned long long int ulli = 1;

    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

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

- liczba ujemna

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

- zamieniamy **moduł** liczby na NKB

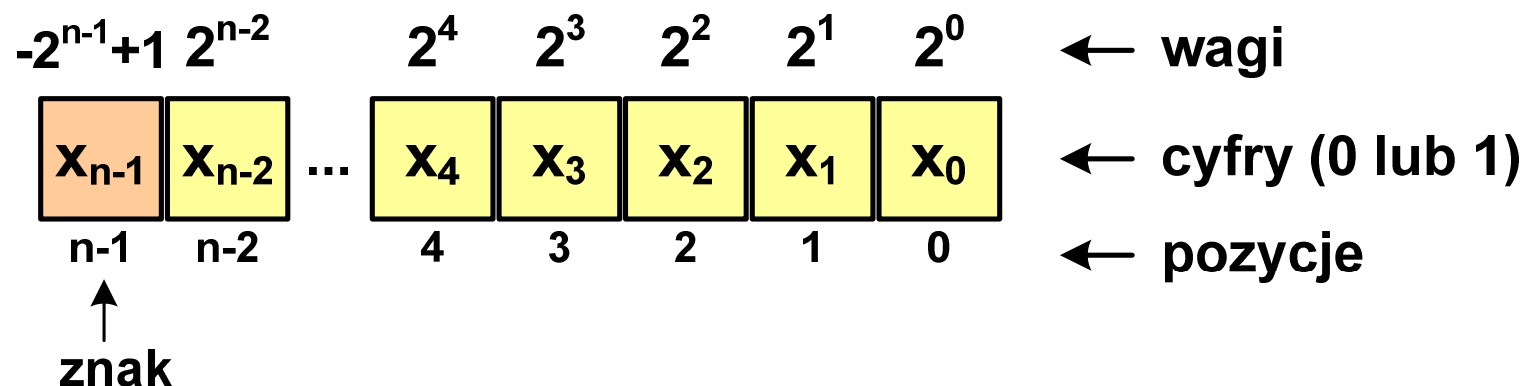
$$|-93_{(10)}| = 93_{(10)} = 1011101_{(NKB)}$$

- dodajemy bit znaku

$$-93_{(10)} = \mathbf{1}1011101_{(ZM)}$$

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

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

- 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

Koniec wykładu nr 3

Dziękuję za uwagę!