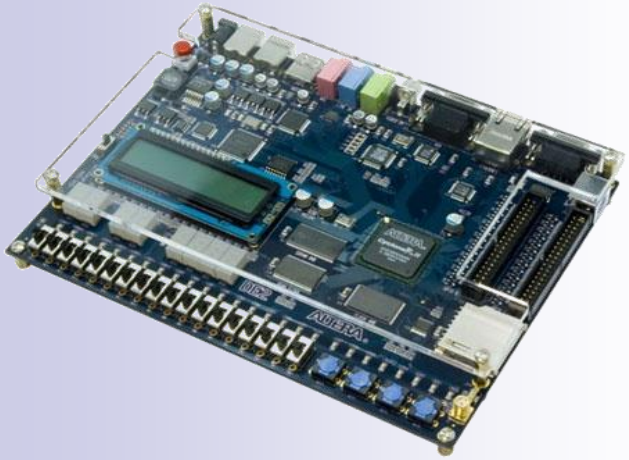
Selection from

Susta:Computer System Structures

& John Loomis: Computer organization

*& M.Mudawar:*Computer Architecture & Assembly Language

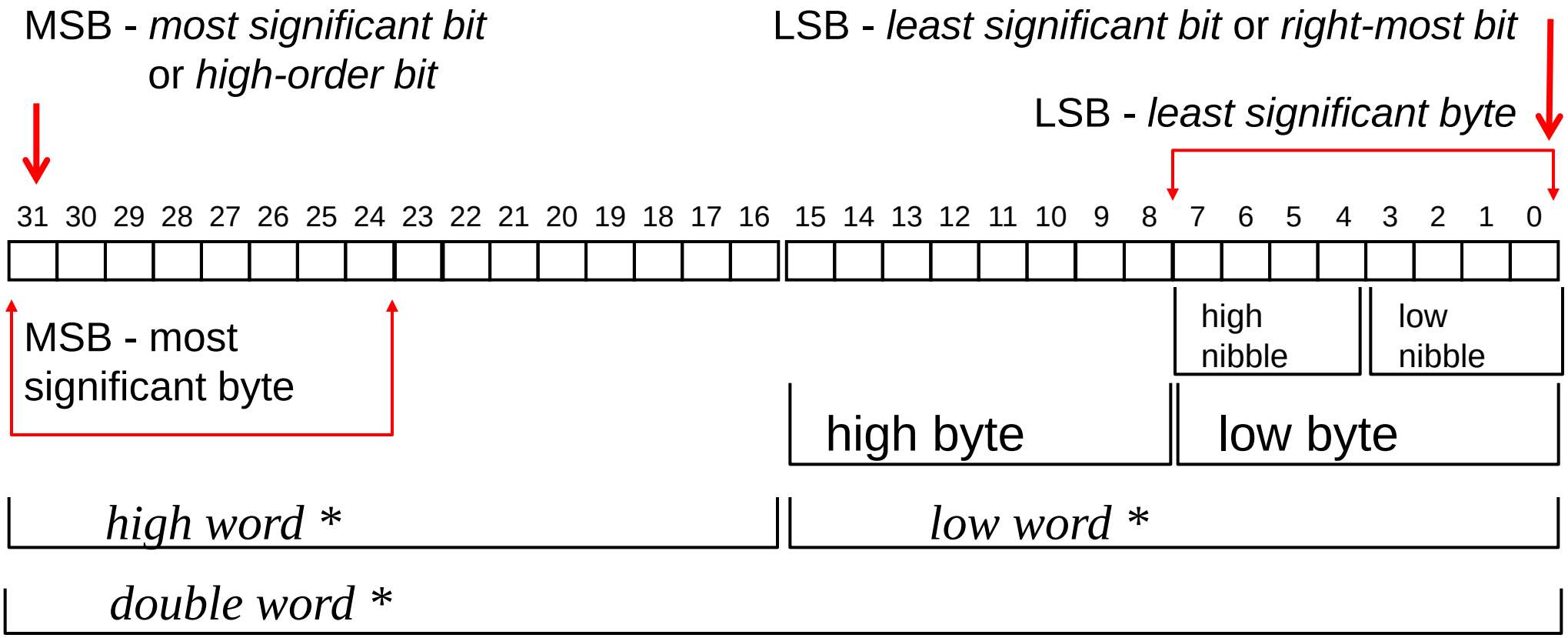
Version: 1.1



Cvičení 1.

ČVUT-FEL in Prague,

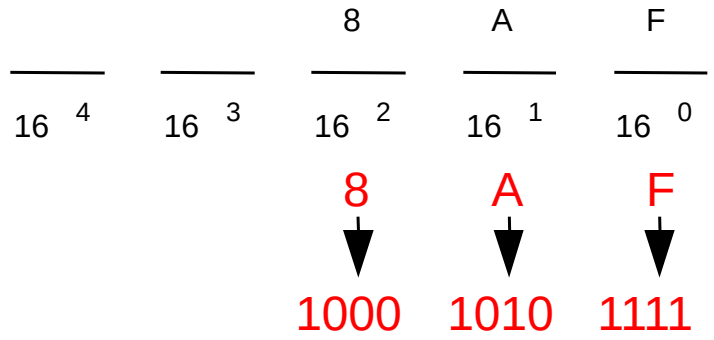
Byte, Nibble, Bit



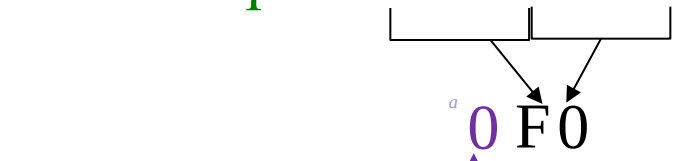
* *Velikost slova "word" není standardem, ale vyjádřením přirozené jednotky architektury počítače;*

Eng: *Hexadecimal* or *Hex* aka Base Sixteen
cz: Hexadecimální čísla

- 4 bity se kódují jako 1 hex číslice pro stručný zápis binárních čísel



Napište 11110000 jako hex



Přidána 0 pro zachování číselného typu

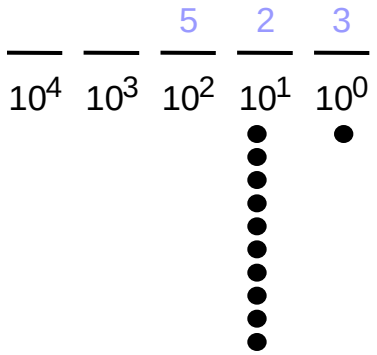
hex	binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

hex	binary
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

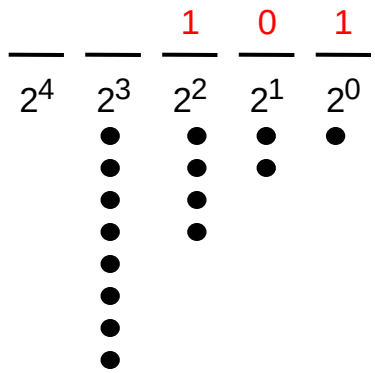
Binary Encoded Unsigned Integers

- Každá pozice reprezentuje kvantitu; symbol na dané pozici udává přírůstek kvantity.

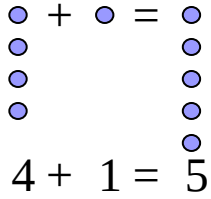
Báze 10 (***decimal***)



Báze 2 (***binary***)



O: Kolik?



4 + 1 = 5

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1,024
11	2,048
12	4,096
13	8,192
14	16,384
15	32,768
16	65,536

Převod čísel bez znaménka

- Desítkové číslo: 13

b) dělením 2

13/2 = 6 zbytek 1 ← LSB

6/2 = 3 zbytek 0

3/2 = 1 zbytek 1

1/2 = 1 zbytek 1 ← MSB

dělením 2 provádíme logický posun čísla doprava

a) postupným odečítáním *mocnin 2 postupně od největší k nejmenší*

32	16	8	4	2	1	
0						$32=2^5$ větší
0	0					$16=2^4$ větší
0	0	1				$8=2^3$ ok, 13-8=5
0	0	1	1			$4=2^2$ =8+4=12 ok, 5-4=1
0	0	1	1			$2=2^1$ větší než 1
0	**0**	**1**	**1**	**0**	**1**	$1=2^0$ =8+4+1=13 1-1=0

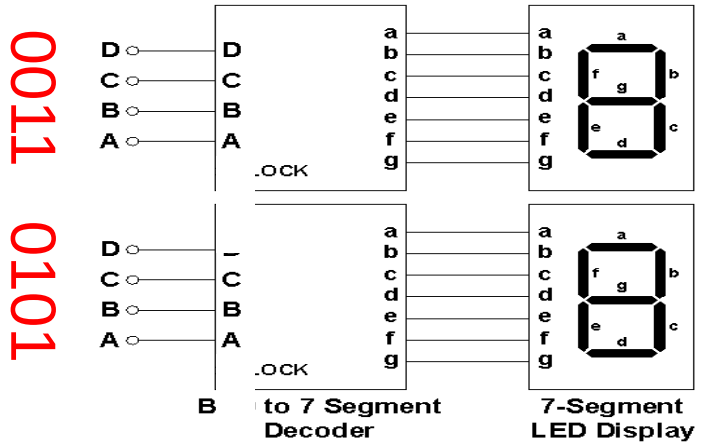
Specialní kódy

Dec.	Binary 8 4 2 1	Octal 421	Hex (VHDL syntax)	BCD binary / hex
0	0000	00	x”0”	0000 0000 = x"00"
1	0001	01	x”1”	0000 0001 = x"01"
2	0010	02	x”2”	0000 0010 = x"02"
3	0011	03	x”3”	0000 0011 = x"03"
4	0100	04	x”4”	0000 0100 = x"04"
5	0101	05	x”5”	0000 0101 = x"05"
6	0110	06	x”6”	0000 0110 = x"06"
7	0111	07	x”7”	0000 0111 = x"07"
8	1000	10	x”8”	0000 1000 = x"08"
9	1001	11	x”9”	0000 1001 = x"09"
10	1010	12	x”A”	0001 0000 = x"10"
11	1011	13	x”B”	0001 0001 = x"11"
12	1100	14	x”C”	0001 0010 = x"12"
13	1101	15	x”D”	0001 0011 = x"13"
14	1110	16	x”E”	0001 0100 = x"14"
15	1111	17	x”F”	0001 0101 = x"15"

Převod binární na BCD

- BCD (Binary-coded decimal) - zjednodušuje číselné displeje

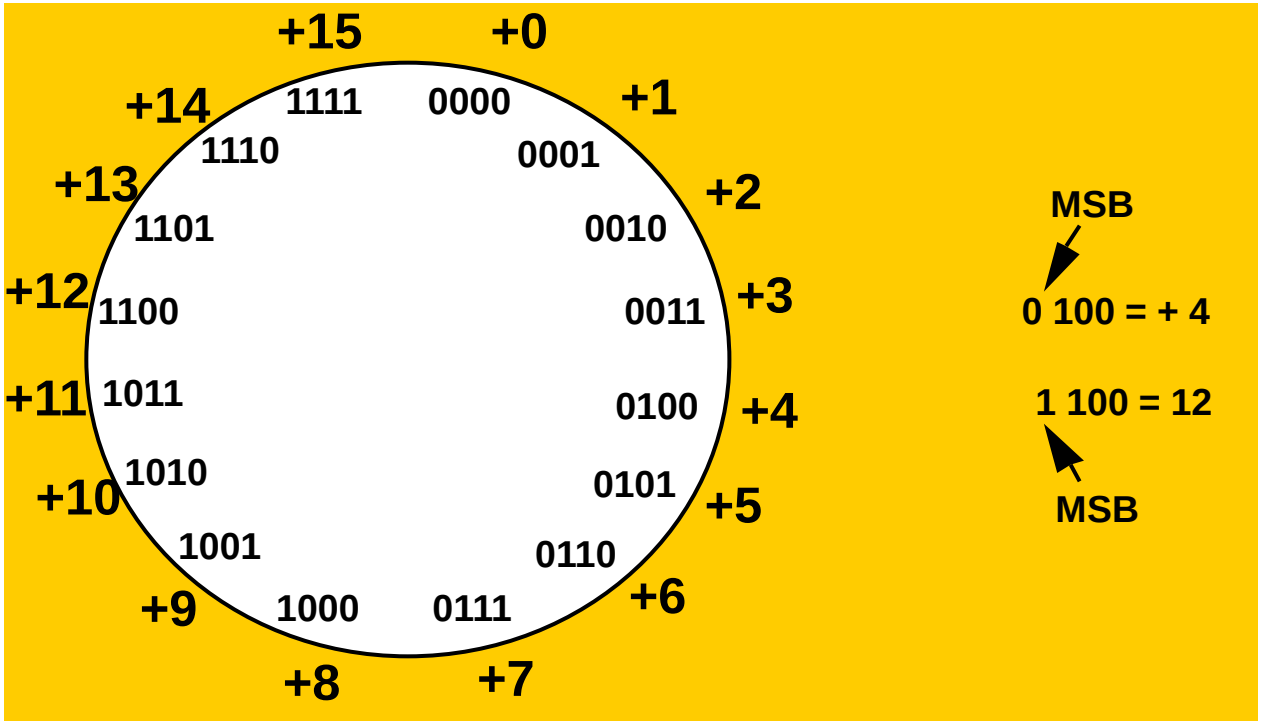
35 jako BCD = x"3 5"
=0011 0101



35 binárně= x"23"
= 10 0011

Předpokládejme 4bitový počítač

Unsigned 4-bit numbers - 4bitové číslo bez znaménka



- Nepřehledné odčítání

[Seungryoul Maeng:Digital Systems]

Způsoby uložení vícebytových čísel v paměti

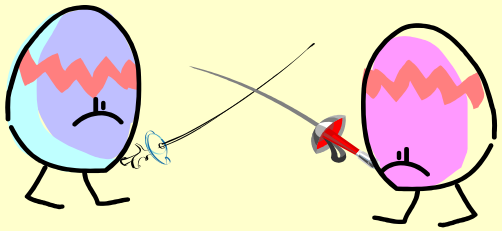
- Hex číslo: 1234567

Big Endian - downto

		0x100	0x101	0x102	0x103		
		01	23	45	67		

Little Endian - to

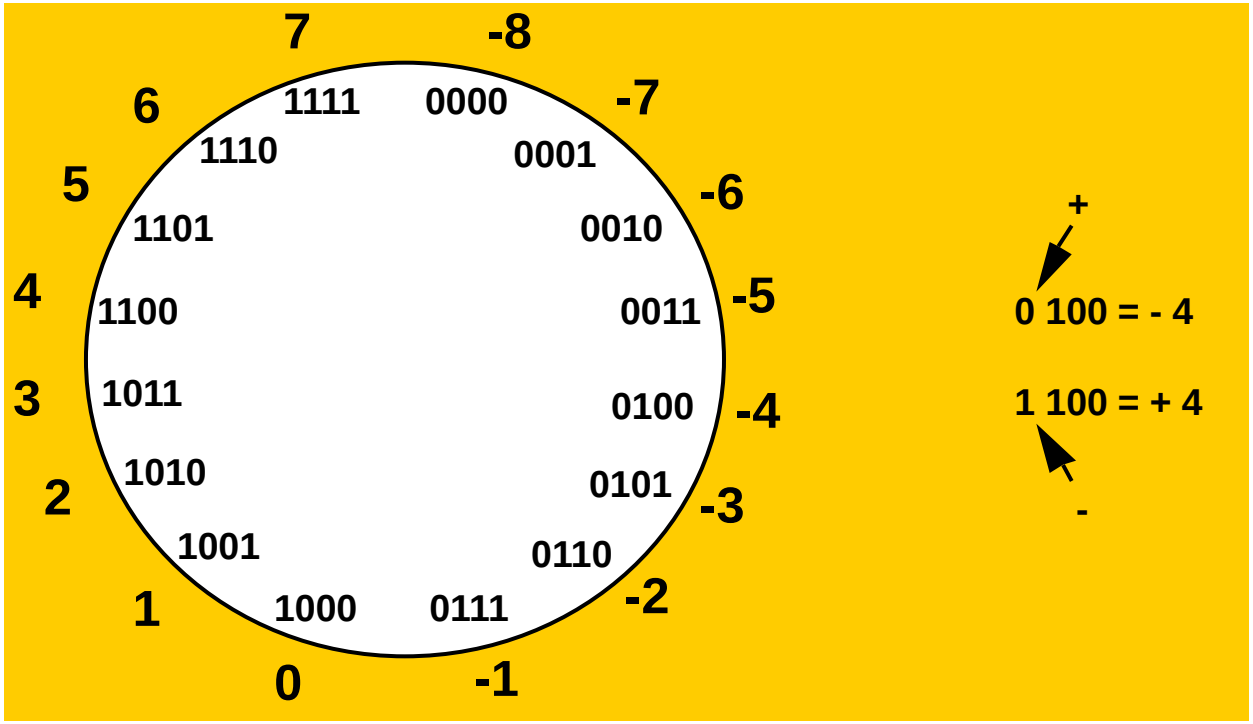
		0x100	0x101	0x102	0x103		
		67	45	23	01		



Little-Endien *pochází z knihy Gulliverovy cesty, Jonathon Swift 1726, v níž označovalo jednu ze dvou znepřátelených frakcí Lilliputů. Její stoupenci jedli vajíčka od užšího konce k širšímu, zatímco*
Big Endien *postupovali opačně. A válka nedala na sebe dlouho čekat…*

Pamatujete si, jak válka skončila?

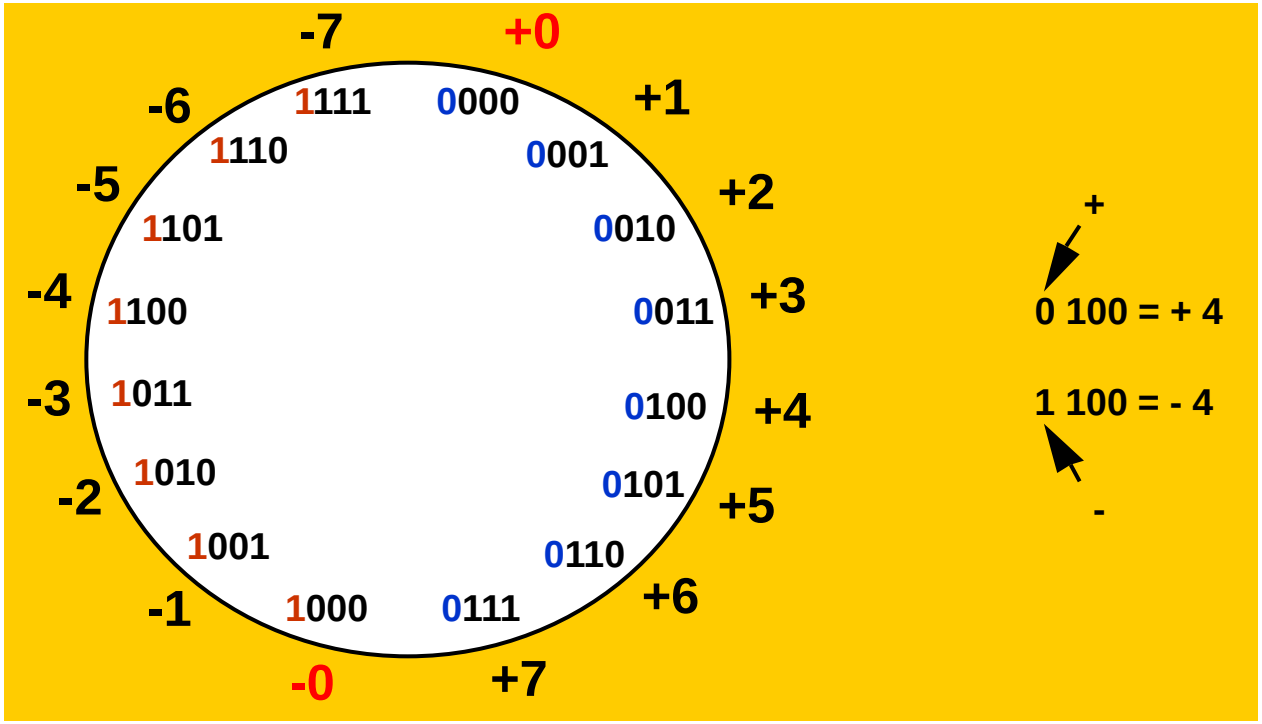
Excess-K, offset binary or biased representation
(cz: Aditivní kód, kód s posunutou nulou)



- Jedna reprezentace 0, možno zvolit si ofset, a tím i počet záporných čísel - často užívaný při zpracování signálů
- Běžné aritmetické jednotky neumí s kódem počítat

Reprezentace čísel se znaménkem

Znaménko a hodnota, tzv. přímý kód.
"Sign and Magnitude Representation"

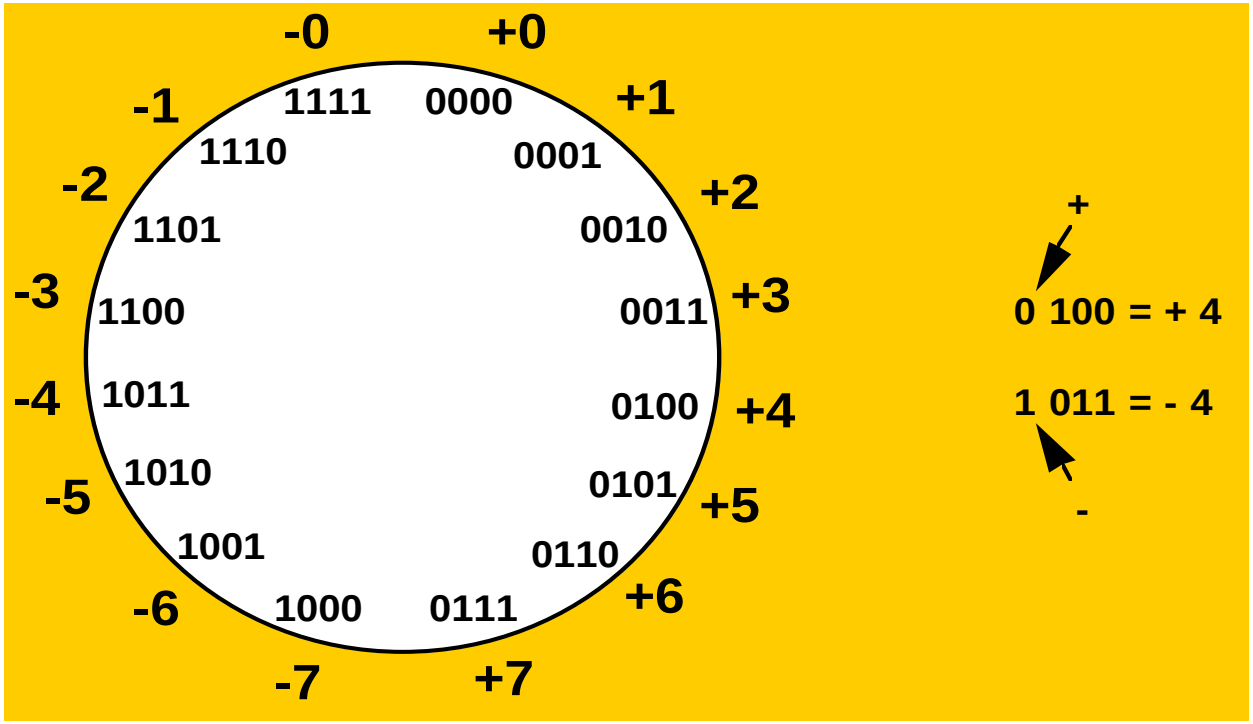


- Nevýhoda: při aritmetických se jinak pracuje se znaménkovým bitem a jinak s hodnotou.
- 2 různá vyjádření nuly.

[Seungryoul Maeng:Digital Systems]

Ones Complement

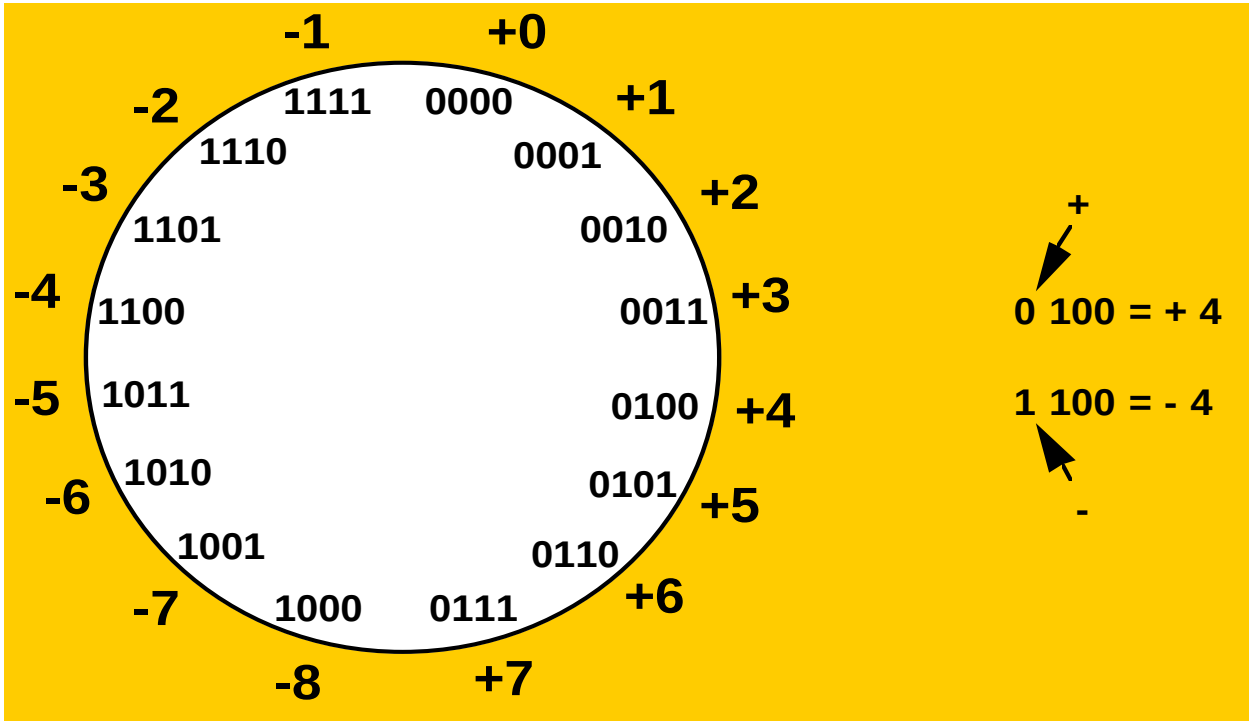
(cz: Jedničkový doplněk, inverzní kód)



- Dvě reprezentace 0!
- Obtížnější sčítání

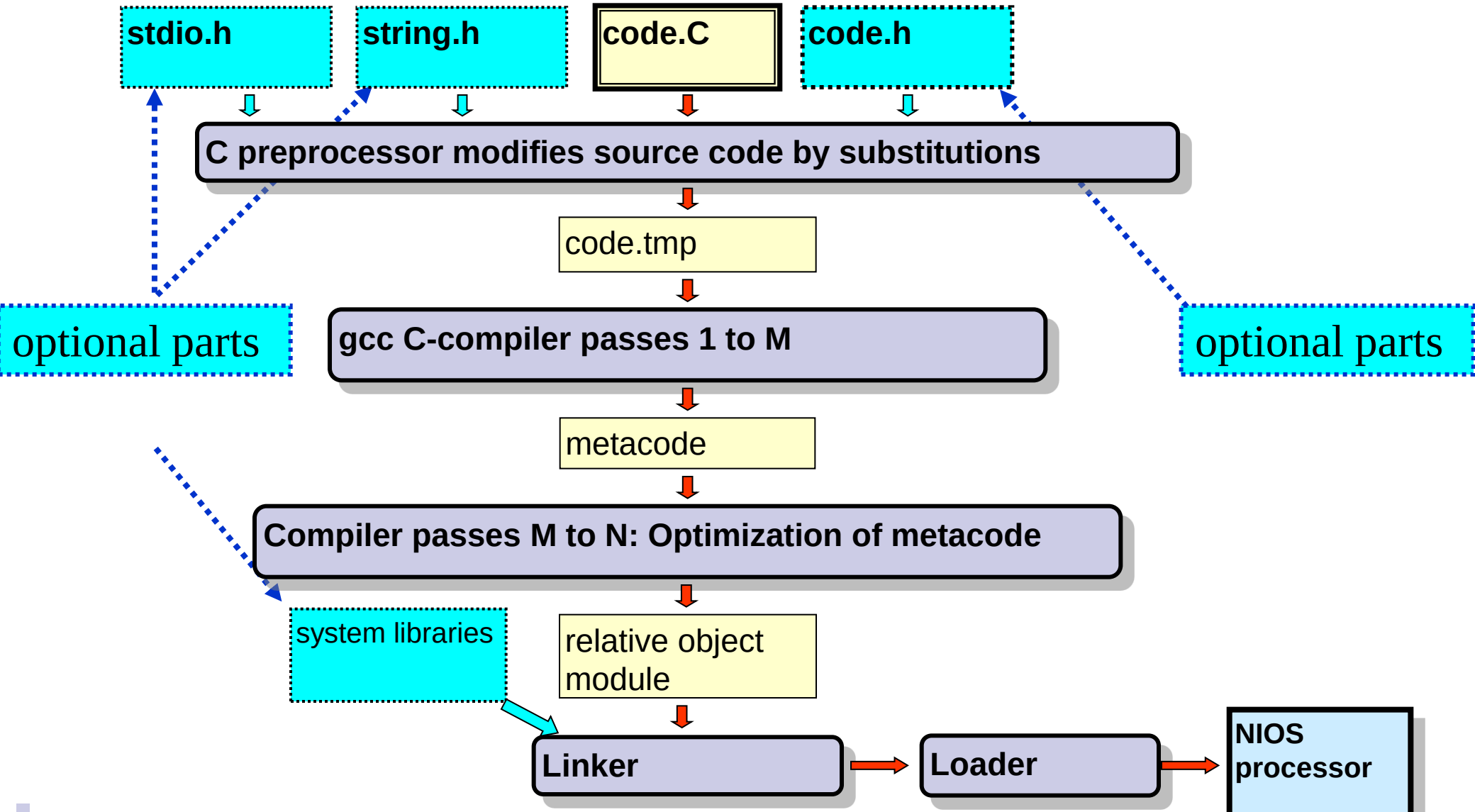
Twos Complement

(cz: Dvojkový doplněk, doplňkový kód)



- Jedna reprezentace 0, stejná aritmetická jednotka jako pro čísla bez znaménka
- Záporných čísel máme o jedno více

Basic Steps of C Compiler



C primitive types

Size	Java	C	C alternative	Range
1	boolean	any integer, true if !=0	BOOL(1	0 to !=0
8	byte	char(2	signed char	–128 to +127
8		unsigned char	BYTE(1	0 to 255
16	short	int	signed short	–32768 to +32767
16		unsigned short		0 to + 65535
32	int	int	signed int	-2^31 to 2^31-1
32		unsigned int	DWORD(1	0 to 2^32-1
64	long	long	long int	-2^63 to 2^63-1
64		unsigned long	LWORD(1	0 to 2^64-1

1) In many implementations, it is not a standard C datatype, but only common custom for user's "#define" macro definitions, see next slides

2) Default is signed but it is best to specify.

Definition of BYTE and BOOL

// by substitution rule no ; and no type check

- #define BYTE unsigned char
- #define BOOL int

// by introducing new type, ending ; is required

- typedef unsigned char BYTE;
- typedef int BOOL;

C language has no strict type checking #define ~ typedef,
but typedef is usually better integrated into compiler.

Defining a Parameterized Macro

#define PRINT_MEM(a) print_mem((unsigned char*)&(a), sizeof(a))

- Similar to a C function, preprocessor macros can be defined with a parameter list; parameters are without data types.
- Syntax:

  `#define MACRONAME(`*`parameter_list`*`) text`

- No white space before (.

Examples:

```
#define MAXVAL(A,B) ((A) > (B)) ? (A) : (B)

#define PRINT(e1,e2) printf("%c\t%d\n",(e1),(e2));

#define putchar(x) putc(x, stdout)
```

#define PRINT_MEM(a) print_mem((unsigned char*)&(a), sizeof(a))

Side-effects!!!

Example:

```
#define PROD1(A,B) A * B
```

Wrong result:

```
PROD1(1+3,2) → 1+3 * 2
```

Improved example with ()

```
#define PROD2(A,B) (A) * (B)

PROD2(1+3,2) → (1+3) * (2)
```

Sign Extension Example in C

```
short int x =  15213;
int       ix = (int) x;
short int y = -15213;
int       iy = (int) y;
```

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
ix	15213	00 00 C4 92	00000000 00000000 00111011 01101101
y	-15213	C4 93	11000100 10010011
iy	-15213	FF FF C4 93	11111111 11111111 11000100 10010011

Bitwise Logic Instructions

		Examples:
■ AND	&	n = n & 0xF0;
■ OR	|	n = n | 0xFF00;
■ XOR	^	n = n ^ 0x80;
■ left shift	<<	n = 0xFF << 4;
■ right shift	>>	n = n >> 4
■ NOT	~	n = ~0xFF;

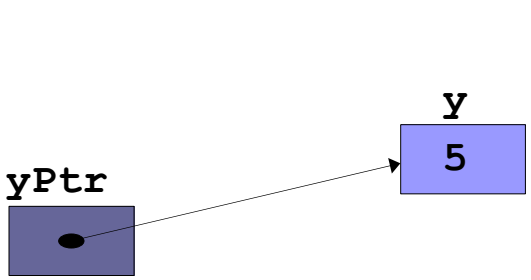
In C, operator << usually behaves as logical for unsigned operand and as arithmetic shift for signed operands.

Pointer Operators

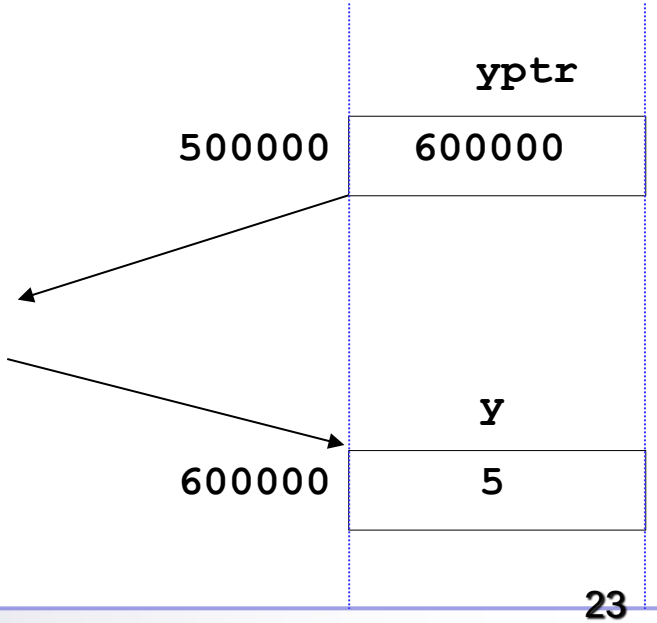
- **&** (address operator)
  - Returns the address of its operand
  - Example

```
int y = 5;
int *yPtr;
yPtr = &y;     // yPtr gets address of y
```

  - **yPtr** "points to" **y**



address of y is value of yptr



Pointer Operators

- **`&`** (address operator)
  - Returns the address of its operand
- * dereference address
  - Get operand stored in address location
- **`*`** and **`&`** are inverses
  (*though not always applicable*)
  - Cancel each other out

```
*&myVar == myVar
```

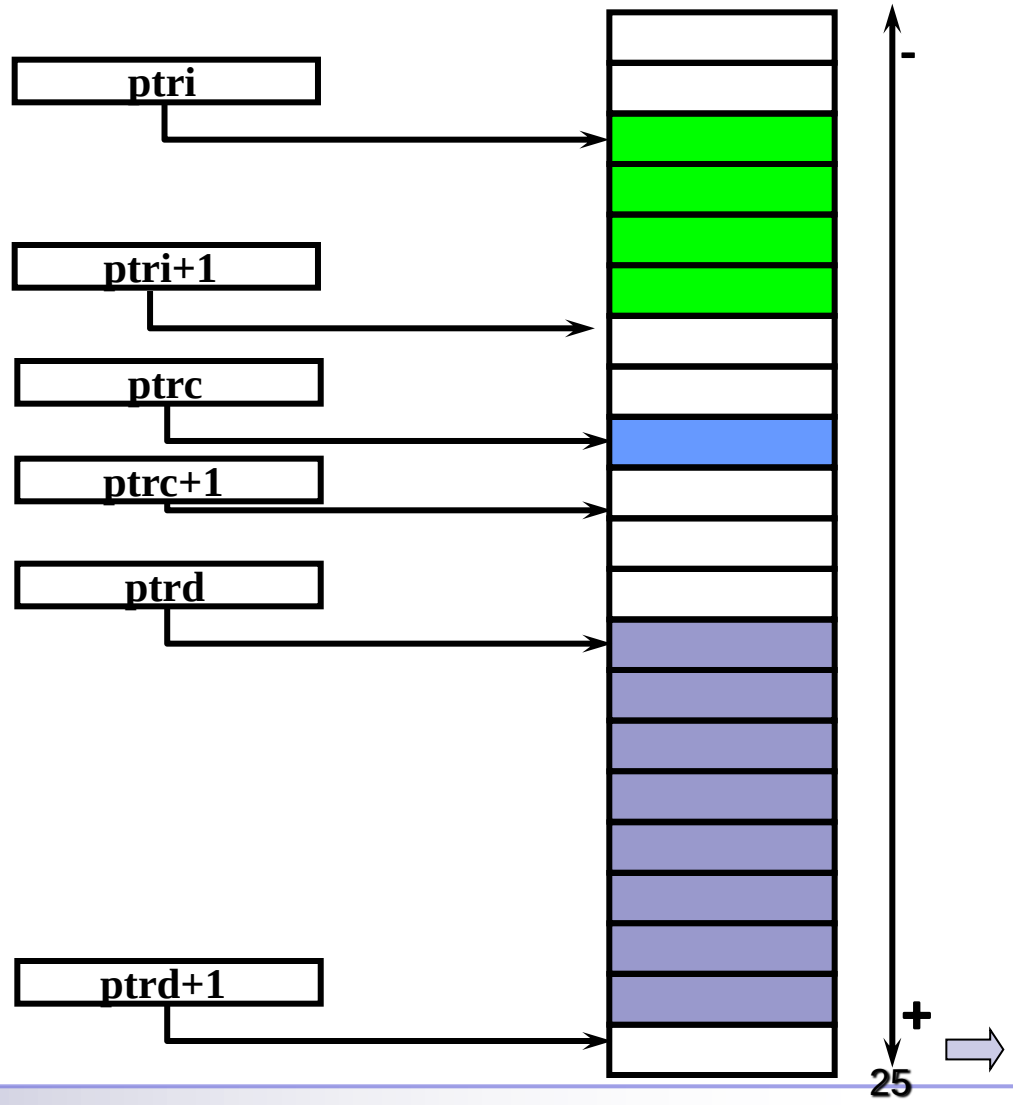
and

```
&*yPtr == yPtr
```

Size of Pointer in C-kod

```
int * ptri;
char * ptrc;
double * ptrd;
```

*ptrx ≡ ptrx[0]
*(ptrx+1) ≡ ptrx[1]
*(ptrx+n) ≡ ptrx[n]
*(ptrx-n) ≡ ptrx[-n]



nr1 = sizeof (double);
nr2 = sizeof (double*);

nr1 != nr2