Úvod do jazyka C

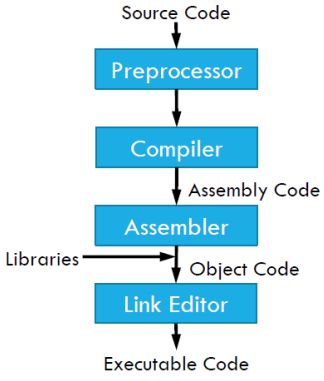
Týždeň	Prednášky	Cvičenia
1.	Úvod do predmetu. Číselné sústavy v elektronike, kódovane čísel. Jazyk C. Úvod do digitálnej elektroniky.	Inštalácia: - IDE *Microchip Studio* - *driver* pre *Arduino kit*
2.	Polovodičové pamäte	Práca s *Microchip Studio*, definovanie projektu
3.	ATmega328, pamäť, generovanie hodinových signálov, manažment napájania, reset	Práca s *Microchip Studio* preklad, simulácia projektu
4.	Vstupy-výstupy	Práca so vstupmi a výstupmi
5.	Watch dog, prerušenia	Práca s prerušeniami Výber zadania na zápočet
6.	Sériové komunikačné prostriedky USART – Tx	Práca s USART – TX. Práca na zadaní
7.	Sériové komunikačné prostriedky USART - Rx	Práca s USART – RX. Práca na zadaní
8.	Časovače – počítadlá	Práca s TC0. Práca na zadaní
9.	Časovače – počítadlá	Práca s TC0. Práca na zadaní
10.	A/D prevodník	Programovanie A/D. Práca na zadaní
11.	EEPROM, *FUSE* bity	Programovanie EEPROM. Práca na zadaní
12.	I2C zbernica	**Zápočet - Praktická práca s MCU**
13.	Predtermín	**Zápočet - Praktická práca s MCU**

Úvod do jazyka C

http://people.tuke.sk/igor.podlubny/C/index.htm

The C Compilation Model

- **The Preprocessor** accepts source code as input and is responsible for
  - Removing comments
  - Interpreting special preprocessor directives denoted by #.
  - Examples: **#include <stdio.h>** , **#define begin {** , **#define end }**
- **The C compiler** translates source to assembly code.
- **The assembler** creates object code.
- **The Link Editor** combines any library functions referenced in the source code with the **main()** function to create an executable file.



Tokens in C

- Keywords
  - These are reserved words of the C language.
  - For example int, float, **if, else, for, while** etc.
- Identifiers
  - An Identifier is a sequence of letters and digits, but must start with a letter.
  - Identifiers are used to name variables, functions etc.
  - Identifiers are case sensitive.
  - Valid: Root, _getchar, __sin, x1, x2, x3, x_1, If
  - Invalid: 324, short, price$, My Name
- Constants
  - 13, 'a', 1.3e-5 etc.
- String Literals
  - A sequence of characters enclosed in double quotes as "…".
  - For example "13" is a string literal and not number 13.
  - 'a' and "a" are different.
- Operators
  - Arithmetic operators: **+, -**, *, / ,%
  - Logical operators: **| |, &&, !**
- White Spaces
  - Spaces, new lines, tabs, comments (A sequence of characters enclosed in /* and */) etc.
  - These are used to separate the adjacent identifiers, keywords and constants.

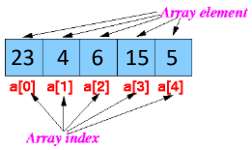
Basic data types

char	Stored as 8 bits. Unsigned 0 to 255. Signed -128 to 127.
short int	Stored as 16 bits. Unsigned 0 to 65535. Signed -32768 to 32767.
int	Same as either short int or long int
long int	Stored as 32 bits. Unsigned 0 to 4294967295. Signed -2147483648 to 2147483647
float	Approximate precision of 6 decimal digits (single precision).
double	Approximate precision of 14 decimal digits (double precision).

Bit (Binary digit) je základnou jednotkou dát v číslicovej elektronike a v informatike. Označuje sa b a môže nadobúdať jednu z dvoch hodnôt 0 alebo 1. Obe hodnoty môžeme chápať aj ako logické hodnoty (true-false, áno-nie), stavy (zapnutý-vypnutý).

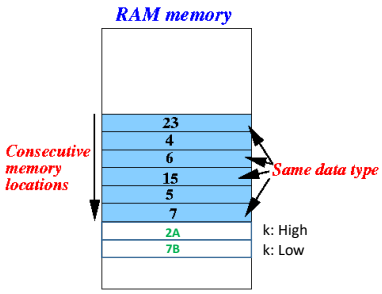
Byte (bajt) je zoskupenie 8 bitov do jedného celku a označuje sa B. Jeden bajt je tvorený ôsmimi bitmi a predstavuje $2^8 = 256$ celých čísel, teda čísla od 0 do 255. Tento interval čísel môžeme zapísať pomocou hexadecimálnych číslic ako 00_H - FF_H alebo (0x00 – 0xFF).

char a[5]={23, 4, 6, 15, 5};



char i = 7;

int k = 0x2A7B;



Constants

- Numerical Constants
  - Constants like 12, 253 are stored as int type (No decimal point).
  - Numbers with a decimal point (21.53) are stored as float or double.
- **Character and string constants**
  - `'c'` , a single character in single quotes are stored as char.
  - Some special character are represented as two characters in single quotes.
    - `'\n'` = newline,
    - `'\t'` = tab,
    - `'\\'` = backlash,
    - `'\"'` = double quotes.
  - A sequence of characters enclosed in double quotes is called a string constant or string literal.
    - For example : "**Hello**"

Variables

- Variable names correspond to locations in the computer's memory
- Every variable has a name, a type, a size and a value
- Naming a Variable
  - Must be a valid identifier
  - Must not be a keyword
  - Names are case sensitive
- Declaring a Variable
  - Each variable used must be declared. Example : `data-type var1, var2,…;`
  - Declaration announces the data type of a variable and allocates appropriate memory location.
  - Initializing value to a variable in the declaration itself: `data-type var = expression;`
  - Examples: `int sum = 0; char newLine = '\n'; float epsilon = 1.0e-6;`

Global and Local variables

- Global Variables
  - These variables are declared outside all functions.
  - Life time of a global variable is the entire execution period of the program.
  - Can be accessed by any function defined below the variable's declaration, in a file.
- Local Variables
  - These variables are declared inside some functions.
  - Life time of a local variable is the entire execution period of the function in which it is defined.
  - Cannot be accessed by any other function.
  - In general variables declared inside a block are accessible only in that block.

Arithmetic Operators

- A = B → Assignment: A gets the value of B
- A + B → Add A and B together
- A – B → Subtract B from A
- A * B → A multiplied by B
- A / B → A divided by B
- A % B → Modulo: Integer remainder of A/B

Example:

```
int A = 11;
int B = 4;
int X = A / B;      // X gets the value 2. Since X is an integer, the fractional part is ignored.
int Y = A % B;      // Y gets the value 3 since A=BX+Y
```

Comparison Operators

- A == B → A is equal to B?
- A != B → A is NOT equal to B?
- A > B → A is greater than B?
- A < B → A is less than B?
- A >= B → A is greater than/equal to B?
- A =< B → A is less than/equal to B?

Logical Operators

Logical Operators map the inputs to either **TRUE** (Logical 1) or **FALSE** (logical 0)

These operators result in a single bit output

- !A → **NOT** A
- A && B → A **AND** B
- A || B → A **OR** B

Example:

```
if (A || (B && C) || !D)
{
  //do something;
}
```

if statement is only satisfied if

- A is logical high **OR**,
- B **AND** C are logical high **OR**,
- D is logical low.

Bitwise Operators

Bitwise operators map input bit vectors to the same sized output bit vector

- ~A → Bitwise complement of A
- A & B → Bitwise AND of A and B
- A | B → Bitwise OR of A and B
- A ^ B → Bitwise XOR of A and B
- A << B → Bitwise left shift A by B positions
- A >> B → Bitwise right shift of A by B positions

Bitwise Operators Examples

Let A = 0b11 and B = 0b01 then

- A represents the bit vector 11
- B represents the bit vector 01

- ~A = 0b00
- A & B = 0b11 & 0b01 = 0b01
- A | B = 0b11 | 0b01 = 0b11
- A ^ B = 0b11 ^ 0b01 = 0b10
- A << B = 0b11 << 0b01 = 0b11 << 1 = 0b10
- A >> B = 0b11 >> 0b01 = 0b11 >> 1 = 0b01

We use bitwise operators frequently to manipulate the register values.

Prefix & Postfix Increment/Decrement

- ++A → The value of A is incremented before assigning it to variable A
- --A → The value of A is decremented before assigning it to variable A
- A++ → The value is incremented after assigning it to the variable A
- A-- → The value is decremented after assigning it to the variable A

Pre/Post Increment Examples

```
int x = 0;
while(++x < 5)
{
   printf("%d ", x);
}
```

- This prints 1, 2, 3, 4
- x is incremented BEFORE the comparison. Since 1 is less than 5, a '1' is printed. This is repeated until x = 4.
- Then the condition for the while loop fails, since x will be assigned a value of 5 before the values are compared.

```
int x = 0;
while(x++ < 5)
{
   printf("%d ", x);
}
```

- This prints 1, 2, 3, 4, 5.
- x is incremented AFTER the comparison, therefore, it meets the criteria of the while loop until x = 5.

Compound Assignments

- A += B → A = A + B
- A -= B → A = A – B
- A *= B → A = A * B
- A /= B → A = A/B
- A %= B → A = A%B
- A &= B → A = A&B
- A |= B → A= A|B
- A <<= B → A = A<<B
- A >>= B → A = A>>B

- Číselná sústava je systém, ktorým sa zapisujú čísla pomocou určitých symbolov. Poznáme dva typy číselných sústav:
  - Pozičná – je definovaná základom, čo je maximálny počet používaných číslic v sústave. Význam číslice v čísle závisí od jej pozície. (Dvojková, Osmičková, Desiatková, Šestnástková).
  - Nepozičná - význam číslice (alebo symbolu) nie je určený jej pozíciou, ale je daný samotným symbolom a jeho usporiadaním. Typickým príkladom sú rímske číslice:
    (MCMXLVI = 1946 = 1000 + 1000-100 + 50-10 + 5 + 1).

- Dvojková sústava má základ 2, čo znamená, že používa iba dve číslice 0 a 1. Hodnotu binárneho čísla s číslicami x_0, x_1,..., x_k vypočítame:

$$x = \sum_{i=0}^{k} x_i * 2^{k-i}$$

Ak, k=7 a x <0, 1>, i - pozícia číslice, potom dekadická hodnota binárneho čísla 11010110 sa vypočíta:

$$(11010110)_2 = \sum_{i=0}^{7} x_i * 2^{7-i} = 1*2^7 + 1*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0$$
$$= 128 + 64 + 16 + 4 + 2 = (214)_{10}$$

Číselné sústavy

- Šestnástková sústava (hexadecimálna) používa základ 16. Hexadecimálne čísla sa zapisujú pomocou číslic: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 a písmen A, B, C, D, E, F. Písmena A až F reprezentujú čísla s hodnotou 10 až 15.

- Čísla zapísané v šestnástkovej sústave sa obvykle označujú písmenom H pripojeným k číslu v dolnom indexe. Napríklad hexadecimálne číslo $03F7_H$ predstavuje hodnotu 1015_{10} v desiatkovej sústave.

$$x = \sum_{i=0}^{k} x_i * 16^{k-i}$$

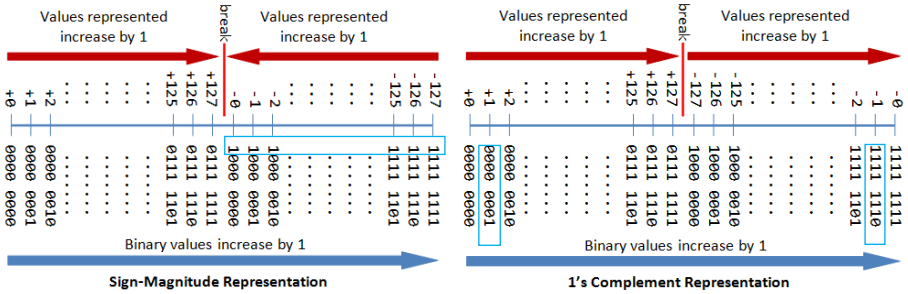
k=3; x <0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F>; i - pozícia číslice

$$4B0E_H = \sum_{i=0}^{3} x_i * 16^{3-i} = 4 * 16^3 + 11 * 16^2 + 0 * 16^1 + 14 * 16^0 = 16384 + 2816 + 0 + 14$$
$$= 19214_{10}$$

Kódovanie binárnych čísel

Kódovanie binárnych čísel na vyjadrenie kladných a záporných čísel je možné vykonať:

- znamienkovým bitom,
- jednotkovým doplnkom,
- dvojkovým doplnkom.



Nevýhoda oboch spôsobov kódovania je, že číslo nula je reprezentované dvakrát.

Dvojkový doplnok je spôsob kódovania celých čísel v dvojkovej sústave, ktorý umožňuje vykonávať sčítanie a odčítanie pri číslach so znamienkom rovnako ako pri číslach bez znamienka. Rozdiel je len v interpretácií pretečenia. Pri tejto metóde sa záporné čísla kódujú tak, že sa inverzne otočia všetky bity kladného čísla a pripočíta sa 1. Tento spôsob kódovania zjednodušuje aritmetické operácie, pretože rovnaký algoritmus sčítania a odčítania funguje pre kladné aj záporné čísla.

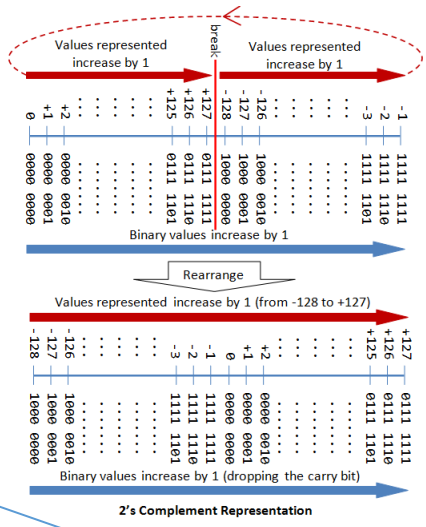
Kódovanie binárnych čísel

Ak 00001101 je binárne vyjadrenie čísla 13, potom číslo −13 sa vypočíta ako:

NOT $(00001101_2) + 1_2 = 11110010_2 + 1_2 = 11110011_2$

Odčítanie je možné realizovať ako sčítanie prvého operandu s dvojkovým doplnkom druhého operandu.

Ak sa takto vyjadrené záporné číslo spočíta s iným záporným alebo väčším kladným číslom, môže dôjsť k pretečeniu rozsahu. Kódovanie dvojkovým doplnkom je navrhnuté tak, že po odrezaní pretečeného bitu zostane správny výsledok.

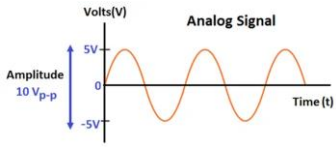


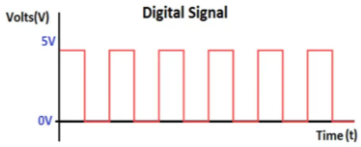
char	Stored as 8 bits. Unsigned 0 to 255. Signed -128 to 127.

$$45_{10} - 13_{10} = 45_{10} + (-13)_{10} = 00101101_2 + 11110011_2 = 1\ 00100000_2 = (32)_{10}$$
$$\approx \quad = \quad 2D_H \quad + \quad F3_H \quad = 1\ 20_H \quad = (32)_{10}$$

Úvod do problematiky digitálnej elektroniky

Analógový a digitálny signál





- Analógový signál označuje signál, ktorý v priebehu času neustále mení svoju hodnotu.
- Digitálny signál je charakterizovaný sekvenciou diskrétnych hodnôt, môže nadobudnúť iba jednu z konečného počtu hodnôt.

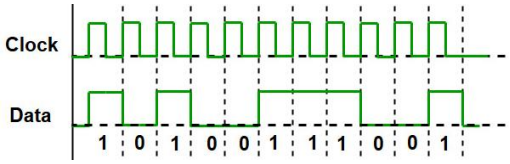
Binárne číslice a logické úrovne

- Väčšina digitálnych obvodov používa binárny systém, ktorý pozná iba dve číslice 1 a 0.
- Napätie používané na vyjadrenie 1 alebo 0 sa nazýva logická úroveň.
- Číslici 0 sa často priradzuje nižšie napätie a označuje sa ako úroveň LOW, zatiaľ čo číslici 1 sa priradzuje vyššia napäťová úroveň HIGH. V takomto prípade hovoríme o pozitívnej logike.
- Ak 0 sa priradí úroveň HIGH a 1 sa priradí úroveň LOW, hovoríme o negatívnej logike.

Tvar digitálneho signálu

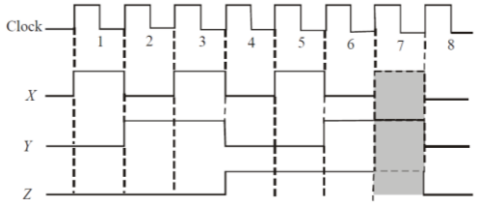
Úvod do problematiky digitálnej elektroniky

Časový priebeh digitálneho signálu



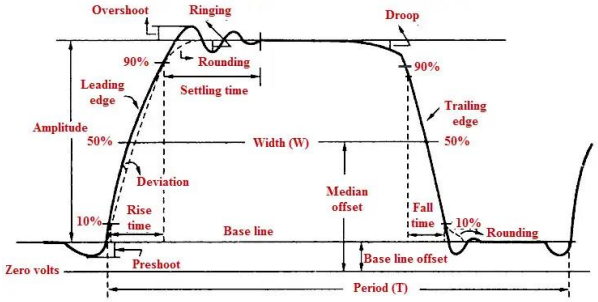
- Binárne informácie, s ktorými pracujú digitálne systémy, sa prejavujú ako časové zmeny digitálneho signálu. V digitálnych systémoch sú priebehy signálu synchronizované so základným časovým priebehom nazývaným taktovacie hodiny. Hodinový signál je periodická postupnosť impulzov, ktorých perióda je rovná trvaniu jedného bitu.

Časový diagram - je graf digitálnych priebehov, ktorý ukazuje časový vzťah všetkých priebehov a ako sa každý priebeh mení vo vzťahu k ostatným.



Úvod do problematiky digitálnej elektroniky

Charakteristika digitálneho impulzu



Rise time - čas vzostupu (nábehu) je čas prechodu impulzu z LOW na HIGH úroveň. V čas nábehu sa meria od 10 % do 90 % amplitúdy impulzu.

Fall time - čas zostupu (pokles) je čas prechodu impulzu z HIGH na LOW úroveň. (od 90 % do 10 %).

Pulse width – šírka impulzu je mierou trvania impulzu a definuje sa ako časový interval medzi 50 % bodmi na vzostupnej a zostupnej hrane.

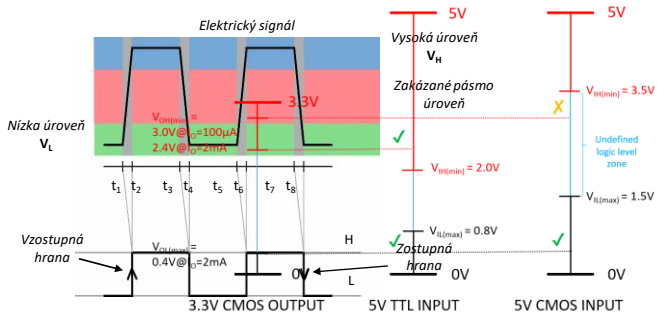
Overshoot – Prekročenie napäťovej úrovne. Môže byť pozitívne aj negatívne.

Ringing – zvlnenie je prekročenie napäťovej úrovne. Môže byť pozitívne aj negatívne.

Úvod do problematiky digitálnej elektroniky

Základné parametre digitálnych obvodov

- Presné hodnoty napätí jednotlivých úrovni sa líšia podľa typu použitých logických obvodov



	Vstupná úroveň V_I		Výstupná úroveň V_O	
Technológia	nízka (V_{IL})	vysoká (V_{IH})	nízka (V_{OL})	vysoká (V_{OH})
TTL 5V	≤ 0,8	≥ 2,0	≤ 0,4	≥ 2,4
CMOS 5V	≤ 1,5	≥ 3,5	≤ 0,5	≥ 4,44
LVTTL 3,3V	≤ 0,8	≥ 2,0	≤ 0,4	≥ 2,4
CMOS 2,5 V	≤ 0,7	≥ 1,7	≤ 0,2	≥ 2,3
CMOS 1,8 V	≤ 0,7	≥ 1,17	≤ 0,45	≥ 1,2

Úvod do problematiky digitálnej elektroniky

Základné parametre digitálnych obvodov

- Presné hodnoty napätí jednotlivých úrovni sa líšia podľa typu použitých logických obvodov

	Vstupná úroveň V_I		Výstupná úroveň V_O	
Technológia	nízka (V_{IL})	vysoká (V_{IH})	nízka (V_{OL})	vysoká (V_{OH})
TTL 5V	≤ 0,8	≥ 2,0	≤ 0,4	≥ 2,4
CMOS 5V	≤ 1,5	≥ 3,5	≤ 0,5	≥ 4,44
LVTTL 3,3V	≤ 0,8	≥ 2,0	≤ 0,4	≥ 2,4
CMOS 2,5 V	≤ 0,7	≥ 1,7	≤ 0,2	≥ 2,3
CMOS 1,8 V	≤ 0,7	≥ 1,17	≤ 0,45	≥ 1,2