

REV - Aplikace embedded systémů v mechatronice

úvodní proslov

Šimon Řeřucha, Zdeněk Matěj

v1.0 (February 20, 2018)

O čem to bude:

- 1 O předmětu
- 2 "Embedded" systémy a základní koncepty
- 3 Mikrokonotrolery a periferie
- 4 Jazyk C

O předmětu:

- 1 úvodní přednáška (dnes)
- 2 seminář “úvod do programování v jazyce C”
- 3 seminář “úvod do embedded programování”
- 4 samostatný projekt
- 5 klasifikovaný zápočet

Základní koncepty embedded



Embedded systém

Jednouúčelový systém, ve kterém je řídicí počítač zcela zabudován do zařízení, které ovládá. Na rozdíl od univerzálních počítačů, jako jsou osobní počítače, jsou zabudované počítače většinou jednouúčelové, určené pro předem definované činnosti. Vzhledem k tomu, že systém je určen pro konkrétní účel, mohou tvůrci systém při návrhu optimalizovat pro konkrétní aplikaci.

Příklady:

bankomat, kalkulačka, pračka, meteostanice, navigační systém rakety, televize ...

Pozn: lepší, avšak krkolomnější české označení je "vestavěný" či "zabudovaný" systém.

Základní konstrukční prvky digitálních systémů

- procesor obsahuje množství tranzistorů
- tranzistory tvoří tzv. logické členy, které realizují logické funkce
- vhodnou kombinací (zpětná vazba) lze vytvořit paměťový člen
- z primitivních prvků se skládají složitější systémy

Logické členy – hradla

Elementary Elements

Logic Symbol

Truth Table

input A NOT output C

| A | C |
|---|---|
| 0 | 1 |
| 1 | 0 |

A B AND C

| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

A B OR C

| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

A B NAND C

| A | B | C |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

A B NOR C

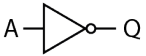
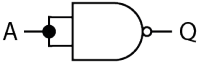

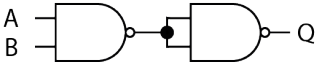

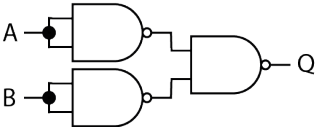
| A | B | C |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

- NOT - negace:
 $C = \neg A = \bar{A}$
- AND - logický součin:
 $C = A \cdot B = AB$
- OR – logický součet:
 $C = A + B$

Funkčně kompletní:

- **NAND – NOT AND** $C = \overline{AB}$
- **NOR – NOT OR** $C = \overline{A + B}$

Funkčně kompletní NAND logika

| Logická funkce | Symbol funkce | Realizace funkce pomocí hradel NAND |
|----------------|---|--|
| NOT |  |  |
| AND |  |  |
| OR |  |  |

Z hradel NAND lze realizovat libovolnou logickou funkci.
Libovolný obvod lze popsat sadou booleovských funkcí.

Booleovská algebra

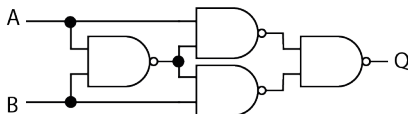
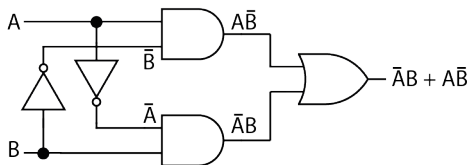
- Komutativnost: $a + b = b + a$; $ab = ba$
- Asociativita: $(a + b) + c = a + (b + c)$
- Absorpce: $a + (ab) = a$; $a(a + b) = a$
- Idempotence: $aa = a$; $a + a = a$
- Komplementarita: $a\bar{a} = 0$; $a + \bar{a} = 1$
- Distributivita: $a(b + c) = ab + ac$; $a + bc = (a + b)(a + c)$
- Ohraničenost: $0a = 0$; $1a = 1$; $0 + a = a$; $1 + a = 1$
- Dvojitá negace: $\overline{\overline{A}} = A$
- De Morganovy zákony: $\overline{A \cdot B} = \bar{A} + \bar{B}$; $\overline{A + B} = \bar{A} \cdot \bar{B}$
V kombinaci pak: $A \cdot B = \overline{\bar{A} + \bar{B}}$; $A + B = \overline{\bar{A} \cdot \bar{B}}$

Příklad: Exkluzivní součet - XOR

Realizace funkce: pravdivostní tabulka, booleovská rovnice, obvod z obecných hradel, obvod z hradel NAND

| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$$Q = A\bar{B} + \bar{A}B$$



Převod mezi intuitivní rovnicí a NAND formou: opakovaně využity De Morganovy zákony.

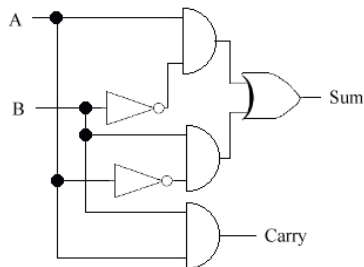
kombinační – výstup je závislý na aktuálním vstupu :

- logické funkce - booleovská logika
- aritmetické funkce - pulsčítačka, sčítačka
- řídicí funkce - multiplexor, dekodér

sekvenční – výstup je závislý na aktuálním vstupu a na celé historii vstupu od iniciálního resetu:

- klopný obvod
- čítač
- stavový automat

Kombinační obvody - pulsčítačka



Vstup:

- jednobitové hodnoty A, B

Výstup:

- součet (Sum): $S = \overline{A}B + A\overline{B}$
- Přenos do vyššího řádu:
 $C = AB$

Kombinační obvody - úplná sčítačka

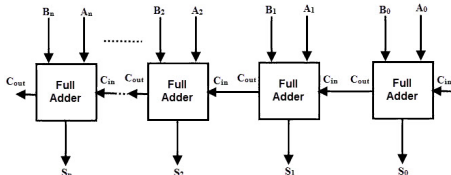
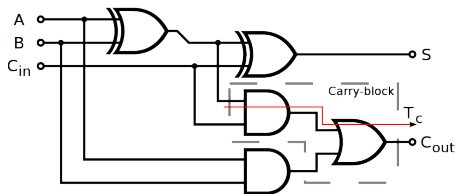
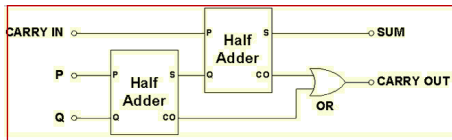
Vstup:

- hodnoty A, B
- přenos z nižšího řádu C_{in}

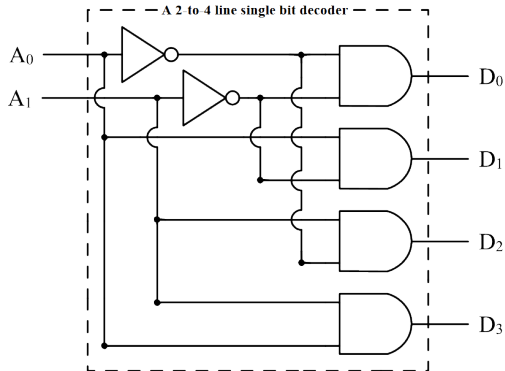
Výstup:

- součet (Sum):
$$S_h = \overline{A}B + A\overline{B}$$
$$S = \overline{S_h}C_{in} + S_h\overline{C_{in}}$$
- Přenos do vyššího řádu:
$$C_{out} = AB + S_hC_{in}$$

Bloky zřetězeny pro součet vícebitových čísel:



Řídicí obvody – dekodér



Truth Table

| A_1 | A_0 | D_3 | D_2 | D_1 | D_0 |
|-------|-------|-------|-------|-------|-------|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |

Minterm Equations

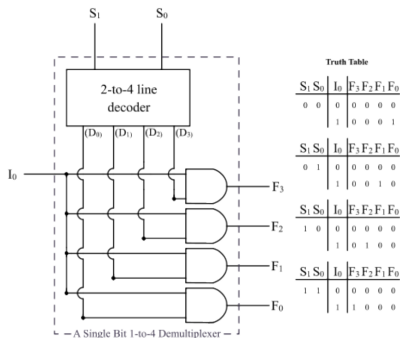
$$D_0 = \overline{A_1} \cdot \overline{A_0}$$

$$D_1 = \overline{A_1} \cdot A_0$$

$$D_2 = A_1 \cdot \overline{A_0}$$

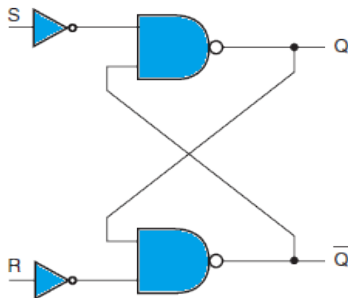
$$D_3 = A_1 \cdot A_0$$

Řídicí obvody – multiplexing



Klopné obvody - SR Latch

Díky zpětné vazbě lze vytvořit paměťový obvod:

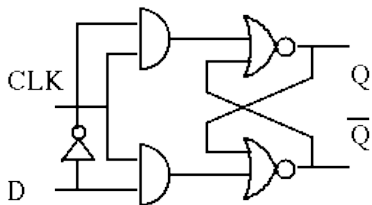
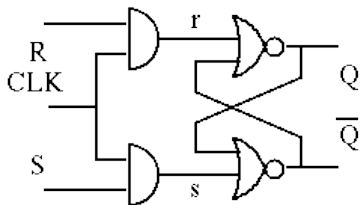


| Operation Mode | S | R | Q_{n+1} |
|----------------|---|---|-----------|
| No change | 0 | 0 | Q_n |
| SET | 1 | 0 | 1 |
| RESET | 0 | 1 | 0 |
| Forbidden | 1 | 1 | — |

Vnitřní stav obvodu se změní přivedením 1 na vstup R/S a tento stav "přetrvává" do dalšího podnětu.

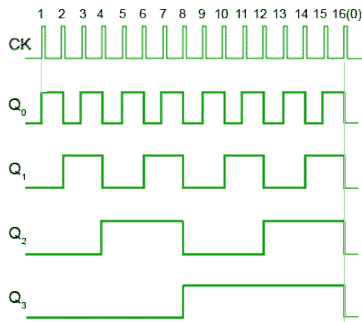
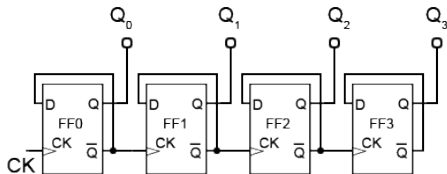
Klopné obvody - odvozené typy

U klopného obvodu typu RS (zde z hradel NOR) lze jednoduchou úpravou dosáhnout synchronizace na hodinový signál:



Klopný obvod typu D navíc odvozuje svůj stav od jednoho vstupu a eliminuje nedefinovaný stav vstupu.

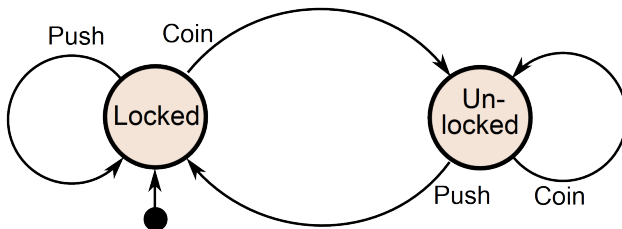
Zřetěžením několika klopných obvodů typu D vznikne čítač:



Stavový automat – Finite State Machine

- FSM klíčový koncept pro embedded, nezávisle na platformě
- konečná množina stavů, vstup a výstup, iniciální stav
- přechodová funkce: změna stavu na základě vstupu
- vystupní funkce: změna výstupu na základě stavu, případně vstupu

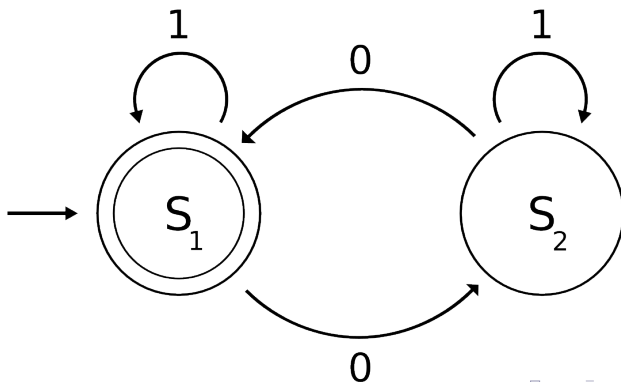
Příklad: stavový diagram "turniket na mince":



Stavový automat typu "Acceptor"

- rozpoznává vstupní posloupnost
- neprázdná množina koncových stavů
- výstupní funkce: akceptace/zamítnutí

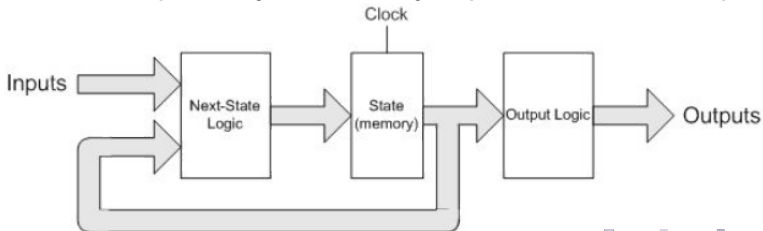
Příklad: stavový diagram "parita":



Stavový automat typu "Transducer"

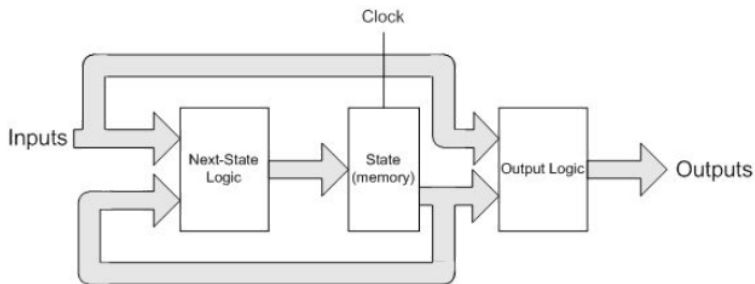
- generuje výstup na základě změn stavu
- dva základní koncepty: Moore, Mealy
- oba využívají tři základní části:
 - ▶ *current state*: realizuje aktuální stav (sekvenční)
 - ▶ *next state*: vyhodnocuje příští stav (kombinační) – realizuje přechodovou funkci
 - ▶ *output logic*: vyhodnocuje výstup (kombinační) – realizuje výstupní funkci

Moore automat: přísně synchronní, výstup asociován se vstupem



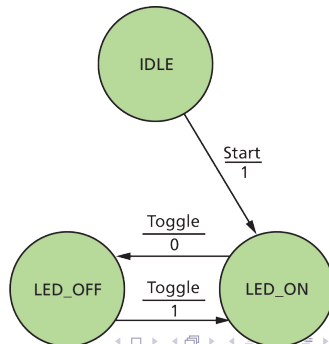
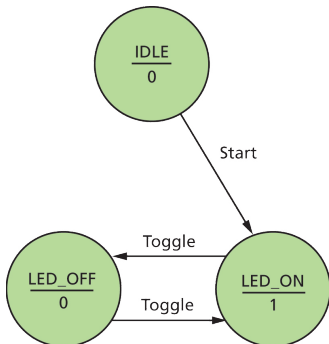
Mealy automat

- výstup na základě aktuálního stavu a vstupu vstupů, tj. výstup asociován s přechodem
- rychlejší díky asynchronní reakci, riziko hazardních stavů
- menší množství stavů než Moore

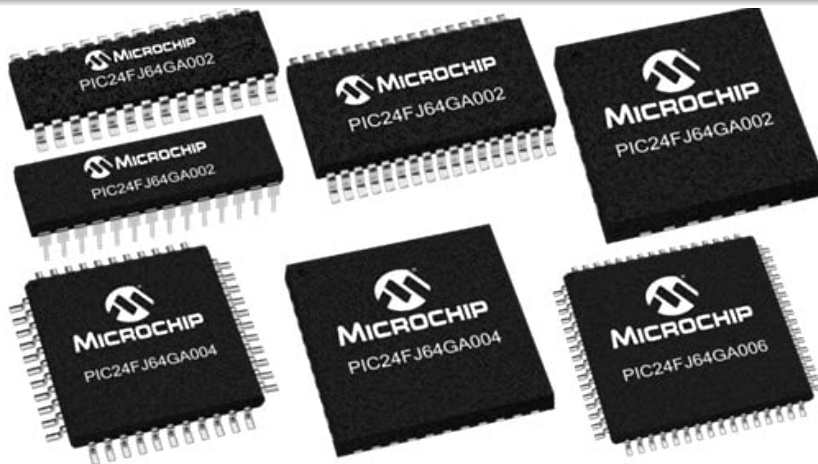


Příklad FSM: LED lampička

- vstup: tlačítko *toggle*
- výstup: LED
- množina stavů: ON, OFF
- iniciální stav: ON
- přechodová/výstupní funkce viz diagram:

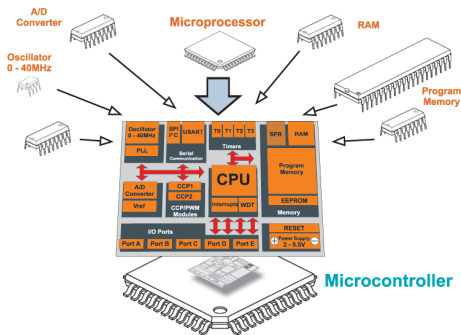


Mikrokontrolery a periferie



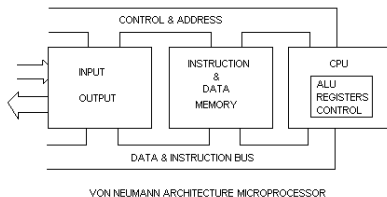
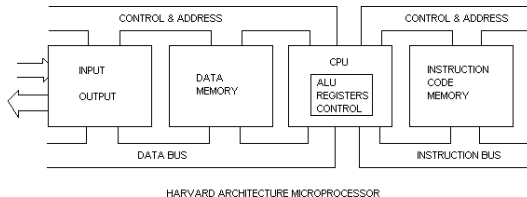
Jednočipový počítač alias mikrokontroler, MCU, uC

- jádrem většiny embedded systémů, zpravidla monolitický integrovaný obvod
- jádro procesoru + paměti pro data a program + obsahuje podpůrné obvody, umožňující samostatnou funkčnost + fixní množství fyzických vstupů a výstupů + periferie



Typické užití: řízení a regulace, komunikační rozhraní, uživatelské rozhraní – analogie zmenšeného stolního PC

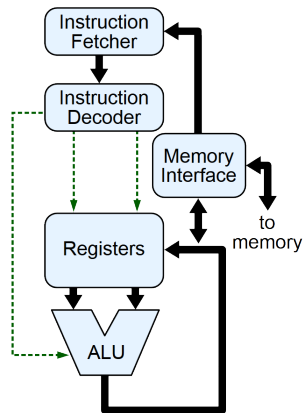
Architektura procesoru



Von Neumann vs. Harvard – společná či oddělená paměť/sběrnice pro data a program

Jádro procesoru

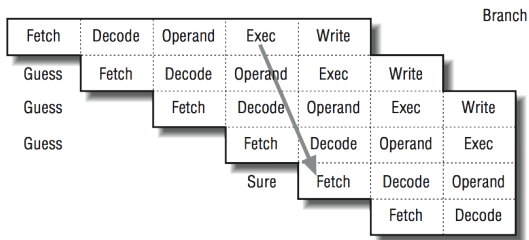
- řadič instrukcí - sekvenční automat (IF + ID)
- registry - pracovní paměť
- aritmeticko-logická jednotka - vykonává operace
- řadič sběrnic(e) - umožňuje přístup k datové a programové paměti (memory interface)



Instrukční cyklus

- Fetch - zpracovávána instrukce načtena z paměti programu
- Decode - rozpoznán typ instrukce a operandy
- Operand - načteny operandy z datové paměti do registrů
- Execute - vykonána instrukce + zápis výsledku (Write)

Příklad: řetězení zpracování (pipelining)

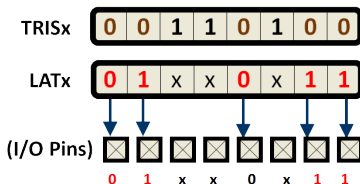


Podpůrné obvody

- datová paměť – energeticky závislá
- paměť programu (Flash)
- zásobník – HW nebo SW
- hodinový signál – reprezentuje diskrétní čas
- resetovací obvody
- **speciální funkční registry** – v adresovém prostoru datové paměti, slouží pro řízení periférií
- konfigurační pojistky – konfigurace obvodů, nutných pro start MCU
- sběrnice

Paralelní I/O (General-Purpose I/O)

- řízení fyzického výstupu (pinu): jednotlivě či sdružené n-bitové slovo
- výstup: GND / Vcc – logická 0 / 1
- třístavová logika – přepínání vstupu a výstupu
- volitelně: pull-up, open-drain
- SFR TRISx pro směr
- SFR PORTx (ev. LATx) pro čtení/zápis hodnoty

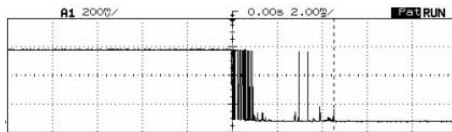
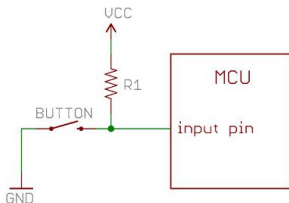


The pin level is driven to the value of
LATx for each bit set to 0 in TRISx

Typické využití: LED, displeje, tlačítka, řízení I²C

Paralelní I/O – tlačítka

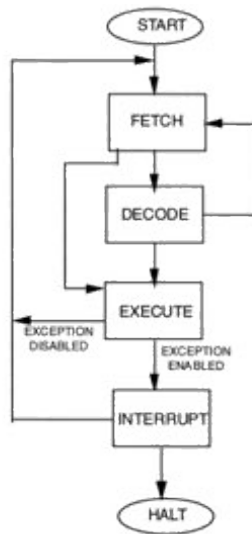
- v rozepnutém stavu náhodná hodnota na vstupu
- pull-up odpor zavede definovanou hodnotu (log. 1)
- invertovaná sémantika



- přechodové jevy při stisknutí tlačítka
- nutno číst vstup opakovaně
- rozestup typ. ≈ 10 ms

Přerušení (interrupts)

- mechanismus pro zpracování asynchronních událostí
- řadič přerušení – vyhodnotí zdroj, nastaví příznak
- tabulka vektorů přerušení (IVT)
- procesor přeruší běh programu, uloží stav procesoru a provede obsluhu přerušení
- vymazání příznaku, návrat z přerušení
- atomické operace a maskování přerušení
- priority přerušení
- souběh přerušení – odložení, vynechání
- latence přerušení



Konfigurace a obsluha přerušení

Definice obsluhy přerušení – různé metody, platformově závislé:

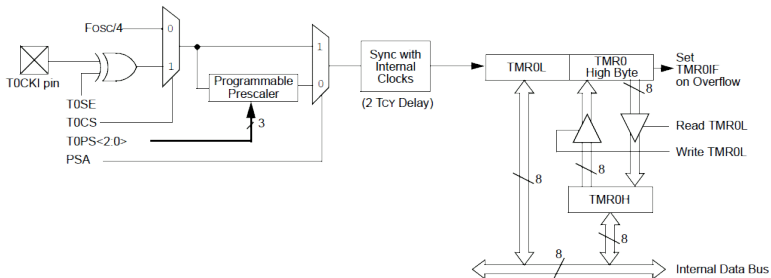
- jedna funkce + multiplexing
- předdefinované názvy procedur
- plnění tabulky vektorů přerušení

SFR:

- `intIE` (interrupt enable) – povolí konkrétní přerušení
- `intIF` (interrupt flag) – indikuje příchod přerušení, příznak nutno vymazat v rámci obsluhy
- `GIE` (global interrupt enable) – povolení funkce řadiče přerušení
- `PEIE` (peripheral interrupt enable) – povolení funkce třídy přerušení

Časovač/Čítač

- binární čítač, definovaná šířka (8, 16, 32bit)
- zdroj: systémové hodiny, externí hodiny, externí signál
- (a-)synchronní čítání
- pre/post-scaler: čítání každé n -té události
- automatický reset čítače, generování přerušení



Obsluha Časovače/Čítače

Obvyklé SFR:

- TMR_x / TMR_xH + TMR_xL: pracovní registr čítače
- TMR_xIE / TMR_xIF: povolení / příznak přerušení
- PR_x (Period Register): hodnota se porovnává s pracovním registrem, v případě shody se pracovní registr nuluje a vyvolá se přerušení
- TMR_xPS, TMR_xCKPS, TMR_xOUTPS: nastavení pre/post-scaleru, zpravidla bitový kód
- TMR_xCS (clock source): nastavení pre/post-scaleru, zpravidla bitový kód
- TMR_xON: povolení časovače

MCU obvykle obsahuje několik časovačů s různou funkcí; zároveň mohou tyto realizovat časovou základnu pro další periferie (např. PWM).

Hlídací pes (WDT)

- bezpečnostní obvod, hlídá korektní běh procesoru
- čítač se zpravidla dedikovaným oscilátorem
- program MCU musí čítač pravidelně programově nulovat
- pokud čítač dosáhne stanovené hodnoty, resetuje MCU
- předpokládá se, že se MCU někde zapoměl
- konfigurace typicky pomocí pojistek

EEPROM paměť slouží k uložení konfiguračních dat.

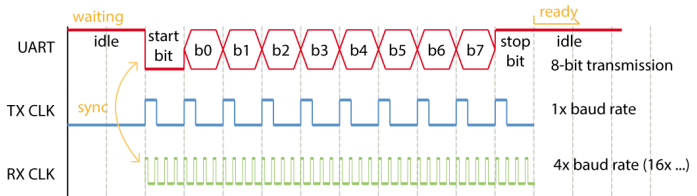
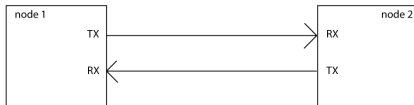
Obsluha pomocí SFR:

- **čtení:** zápisem log. 1 do položky RD registru EECON1 se hodnota na adrese EEADR uloží do EEDATA
- **zápis:** zápisem log. 1 do položky WR registru EECON1 se hodnota EEDATA uloží do paměti na adresu EEADR
- zabezpečení zápisu – před zápisem je třeba:
 - ▶ explicitně zápis povolit zápisem log. 1 do položky WREN registru EECON1
 - ▶ zapsat sekvenci 0x55, 0xAA do registru EECON2.

Podobně lze u některých MCU zapisovat do Flash paměti program nebo konfigurační pojistky.

Komunikační protokol UART

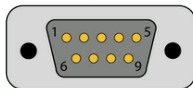
- Universal Asynchronous Receiver-Transmitter
- volitelná délka datového slova (7,8,9 bitů)
- volitelně detekční kód – parita
- volitelná délka stop-bitu
- uzly domluvené na taktování a parametrech přenosu



Rozhraní RS-232

- dříve běžná součást PC jako "sériový port"
- využívá UART
- definuje fyzické rozhraní a napěťové úrovně
- standardní datové rychlosti
- definuje metodu synchronizace a dekódování na RX straně

DB9M Connector

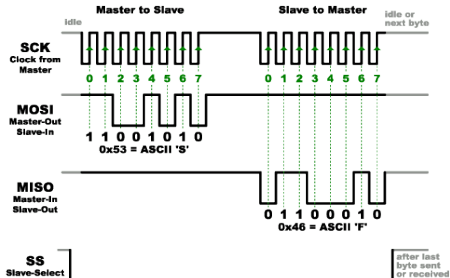
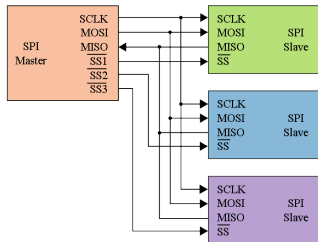


RS232 Pin Out

| Pin # | Signal |
|-------|--------|
| 1 | DCD |
| 2 | RX |
| 3 | TX |
| 4 | DTR |
| 5 | GND |
| 6 | DSR |
| 7 | RTS |
| 8 | CTS |
| 9 | RI |

Sběrnice SPI

- master-slave sériová sběrnice
- hodinový signál
- dva datové signály:
master in - slave out (MISO)
master out - slave in (MOSI)
- slave-select signály

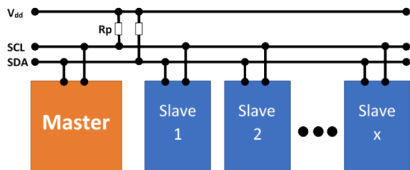


- duplexní režim možný
- korektní nastavení fáze a polarity
- libovolné napěťové úrovně i délka slova
- taktování $> 10 \text{ MHz}$

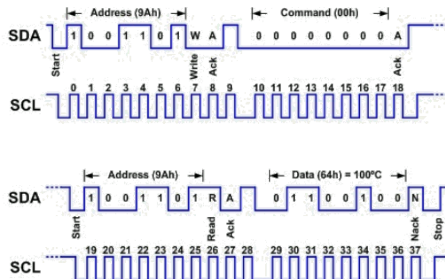
Typické užití: komunikace mezi IC nebo mezi blízkými komponenty

Sběrnice I2C

- master-slave sériová sběrnice
- hodinový signál SCL
- datový signál SDA
- externí pull-up pro střídání zdroje signálu
- uzly se mohou v roli Master střídát



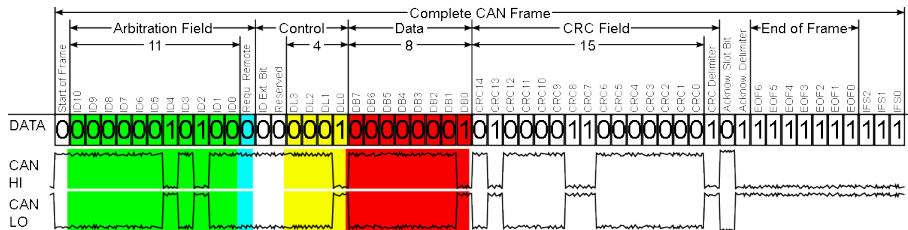
Typické užití: komunikace mezi IC



- součástí datového paketu adresa Slave uzlu
- potvrzování / timeouty
- taktování 400 KHz

CAN-bus (Controller Area Network)

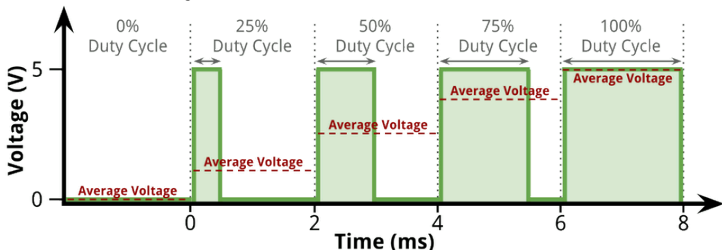
- peer-to-peer sériová sběrnice, 1 diferenciální pár
- spolehlivé doručení, prioritizace provozu
- 11-bit (28-bit) CAN-id, 8 bajtů data
- rychlost až 1 Mbit/s, robustní provoz, optické oddělení uzlů



Typické užití: automotive aplikace

Pulsně-šířková modulace (PWM)

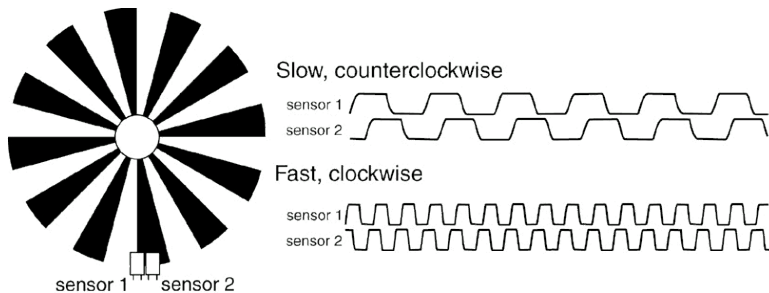
Modulační technika, která umožňuje binárně generovat spojitou veličinu změnou délky aktivní doby pulzu v rámci periody
Zpravidla řízeno čítačem, který je porovnáván s registrem periody a registrem aktivní doby.



Typické užití: motory, světelné zdroje, výkonové zátěže obecně

Kvadrurní (rotační) enkodér

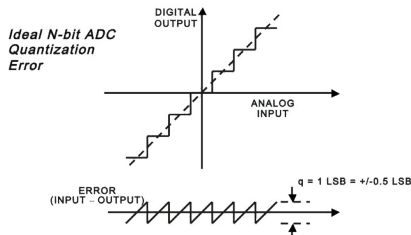
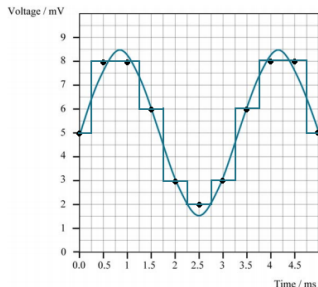
- elektromechanický/ elektrooptický systém
- převádí změnu úhlu na kvadrurní signál
- rozlišení: pulzy na otáčku
- vzájemná fáze dvojice signálů udává směr pohybu
- ev. doplněn 3. signálem pro indikaci absolutní polohy
- řízeno přerušením (A xor B)



Typické užití: mechanické ovládací prvky, úhlové a lineární délkové senzory

Převodník analog-číslo (ADC)

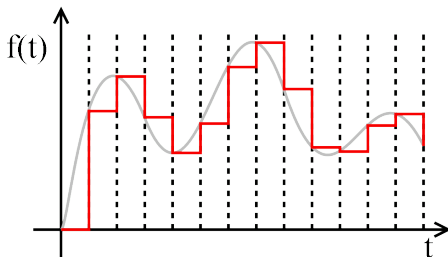
- převod spojité veličiny na digitální reprezentace
- diskrétní vzorkování
- diskrétní kvantizace
- přenosová funkce



- kvantizační chyba
- vzorkovací rychlost
- vzorkovací teorém
- filtr pro anti-aliasing

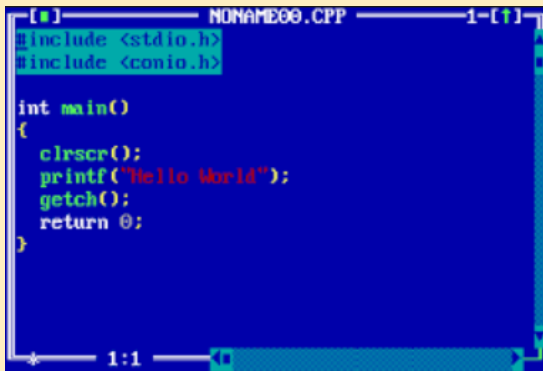
Převodník číslo-analog(DAC)

- převádí digitálně reprezentovanou veličinu na analogovou
- vzorkování, kvantizace, přenos analogické opačným směru
- rekonstrukční filtr



Rychlý úvod do jazyka C

...vše, co jste potřebovali k programování a báli jste se zeptat.

A screenshot of a text editor window showing a C program. The title bar at the top reads "[.] NONAME00.CPP 1-[↑]". The code is as follows:

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

int main()
{
    clrscr();
    printf("Hello World");
    getch();
    return 0;
}
```

The code is displayed on a dark blue background with a light blue border. The first two lines are highlighted in light blue. The function name 'main' in the 'int main()' declaration is highlighted in red. The 'printf' function call is also highlighted in red. The editor has a status bar at the bottom left showing '1:1' and a cursor icon.

Programovací jazyk C

- programovací jazyk pro obecné použití (general-purpose)
- imperativní paradigma, strukturovaný jazyk
- jednoduchá struktura, zaměřená na algoritmizaci
- pro smysluplné využití vyžaduje standardní knihovnu funkcí
- původně vznikl pro vývoj systémového SW v Un*xových operačních systémech
- v současnosti využíván zejména pro:
 - ▶ systémový software (např. Linux Kernel)
 - ▶ embedded systems
 - ▶ stále i aplikační software (doplněno GUI knihovnou)

Charakteristika jazyka C I.

- free-form jazyk, case-sensitive
- málo klíčových slov, mnoho aritmeticko-logických operátorů

Proměnné

- jednoduché typy - znak, celé číslo, reálné číslo
- složené typy heterogenní - struktury
- složené typy homogenní - pole, řetězce znaků
- ukazatele - odkaz na proměnou
- uživatelsky definované typy

Charakteristika jazyka C II.

Funkce:

- C program se skládá z procedur/funkcí – základní modularita
- deklarace = funkční prototyp
udává název funkce, vstupní argumenty a návratovou hodnotu
- definice obsahuje hlavičku a posloupnost příkazů
- funkce vždy definována vně jiné funkce

Na funkce i proměnné se vztahuje tzv. lexikální viditelnost: obé lze použít pouze pokud je již definováno nebo deklarováno.

Modularita:

- funkce podobného účelu jsou sdružovány do modulů.
- deklarace umožňuje "export" funkcí mezi moduly

Textový preprocesor jazyka C:

- zpracování zdrojového textu před kompilací
- možnost spojení zdrojových textů
- podpora textových maker
- podmíněná/modulární kompilace

Hello world!

```
#include<stdio.h>

void main(void){
    int i;
    int k = 5;

    printf("Ahoj svete!\n");

    if(k > 1){
        for (i = 0; i < k; i++){
            printf("...%d\n", i);
        }
    }

    return;
}
```

Základní syntaxe C

- příkazy ukončené středníkem: `a = 42;`
- bloky příkazů ve složených závorkách: `{...}`
- jednořádkové komentáře: `// komentar do konce radku`
- víceřádkové komentáře: `/* komentar od .. do */`
- identifikátory (názvy proměnných a funkcí)
 - ▶ skládají se z alfanumerických znaků a podtržítka
 - ▶ jsou case-sensitive
 - ▶ nesmí začínat číslem
 - ▶ identifikátory začínající podtržítkem rezervované pro systémové knihovny

Jednoduché proměnné

Proměnná = identifikátor + datový typ + platnost.

Celá čísla (Integers) a racionální č. (Floats):

- `int` – celé číslo
- `unsigned int` – přirozené číslo (vč nuly)
- `short int`, `long int` – menší/větší rozsah
- `char` – znak, reprezentován ordinálním číslem (viz ASCII tabulka)
- `float`, `double` – reálná čísla, liší se v rozsahu/přesnosti

Speciální typ: `void` – tzv. prázdný datový typ

Uživatelské datové typy – kl. slovo `typedef`

Rozsah hodnot mají proměnné vždy pevně daný
(překladačem+platformou)

Abeceda jazyka C a konstanty

- **celé číslo**: soustava dle prefixu implicitně typu `int`:
`123 = 0173 = 0x7B = 0b01111011`
- **racionální číslo**: implicitně typu `double`
`1.0, 3.14, 5e6, .84`
- u konstant lze explicitně zadat typ: `123U /*unsigned*/, 123L /*long*/, 3.14f /*float*/`
- **znak**, v apostrofech: `'A' = '\101' = '\x041' = 65`
- speciální znaky:
 - `'\0' /*nulovy znak/,`
 - `'\n' '\r' /*konec radku/,`
 - `'\'' /* apostrof/,`
 - `'\\' /* zpetne lomitko/,`

Řetězcové konstanty

- uvádí se v uvozovkách, konstanty se implicitně spojují:
`"kus textu" = "kus" " textu" // spojení konstant`
- pokud má obsahovat uvozovku, nutno ošetřit zpětným lomítkem
`"kdyz \"ano\" znamená \"ne\"" // escape sequence`
- může obsahovat netisknutelné znaky
`" radek1 \n radek dva" // na dva radky`
- může obsahovat i zpetne lomítka, apostrofy jsou OK
`"neplet lomítko '/' a backslash!'\\' "`

Příklady

```
int i = 0;           // same as "signed int i"
unsigned int ui;     // "unsigned ui" also valid
long int a = 1L;     // "long a" also valid
long long int u;      // c99 only
char c = 'A';        // signed, range -128..127!
unsigned char uc;     // unsigned, range 0..255
float f = 1.0f;       // single precision
double d = 0.5;       // double precision
long double xxx;      // extended precision
typedef unsigned long int my_type; // uživatelsky datový
my_type udaj;         // "udaj" typu unsigned long
```

V rámci definice lze proměnnou i tzv. inicializovat.

Jazyk C je bohatý na **operátory**:

- přiřazení: `=`
- aritmetické: `+`, `-`, `*`, `/`, `%`,
- kombinované:
`+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `~=`, `<<=`, `>>=`
- relační: `==`, `!=`, `<`, `>`, `<=`, `>=`
- logické: `!`, `&&`, `||`

Logické a relační operátory pracují s typem `int`:

logická 0 = FALSE = hodnota 0

logická 1 = TRUE = nenulová hodnota

Pozor: nutno si neplést přiřazení a porovnání

Operátory II.

- bitové: `~, &, |, ^`
- bitový posun: `<<, >>`
- inkrement/dekrement: `++, --`
- podvýraz: `()`
- podmínkový: `? :`
- volání funkce: `()`
- (de)referenční: `*, &, []`
- sekvenční: `,`
- typová konverze: `(type)`
- adresační: `., ->`
- pomocné: `sizeof()`

Operátory III.

- je definována priorita operátorů
- líné vyhodnocení – pokud je po vyhodnocení části výrazu známý výsledek, zbytek podvýrazu se nevyhodnocuje
- implicitní typová konverze při přiřazení: ořezání, přetečení
- implicitní typová konverze při vyhodnocení:
`int -> unsigned -> long -> float -> double`
`int i = 2; float f = i / 2;`
- explicitní TK: `float f=3.14; int i = (int)f;`

- větvení programu (branching) – if, switch
- cykly (branching) – while, switch

Dvoucestné větvení - IF

Syntaxe:

```
if(condition) then expr1; or  
if(condition) then expr1; else expr2;
```

Příklady:

```
if (a == 1) b = 2;
```

```
if (a == 1) { b = 2; }
```

```
if (a == 1) {  
    b = 2;  
};
```

```
(a == 1) ? b = 2 : ;
```

```
if (a = 1) b = 2; // !!!
```

```
if (a == 1) b = 2;  
else      b = 1;
```

```
b = (a == 1) ? 2 : 1;
```

```
if (a == 1){  
    b = 2;  
    c = 1;  
}else{  
    a = 1;  
}
```

Vícecestné větvení - SWITCH/CASE

Konstrukce `switch()` přijímá celočíselný parametr a vyhodnotí `case`, jehož konstanta odpovídá hodnotě. Pokud takový není definován, vyhodnotí se větev `default`.

```
int c;
switch(c){
    case 1: ...
        break;
    case 2: ...
        break;
    default: ...
        break;
}
```

```
char c;
switch(c){
    case 'f': ... //
    case 'F': ...
        break;
    default: ...
        break;
}
```

Poznámka: pokud vynecháte `break`, vyhodnocení pokračuje v další větvi.

Cyklus WHILE

Syntaxe: `while(condition) expr1;`

- cyklus probíhá dokud platí *condition*
- *condition* je vyhodnocena dříve než *expr*

```
a = 123; int b;                                // prazdny cyklus
                                                while(a < 0);

// jednoduchy cyklus
while (a > 0){                                    // nekonecny cyklus
    // spocitej
    b += calculate(a);                            while (1){
                                                b += calculate(a++);
                                                }

    // odedti 1 od a
    a--;                                          // neprobehne nikdy
}                                                while(0 || a < 0);
```


Cyklus DO .. WHILE

Syntaxe: `do expr1 while(condition);`

- cyklus probíhá dokud platí *condition*
- *expr* je vyhodnocena dříve než *condition*, *expr* proběhne alespoň jednou

```
a = 32;
```

```
do{  
    b += calculate(a);  
}while (a > 0);
```

Cyklus FOR

Syntaxe: `for(c_start; c_end; c_iter) expr;`

- 1 na začátku se jednou vyhodnotí *c_start*
- 2 poté se vyhodnotí *c_end*:
 - ▶ pokud platí, vyhodnotí se *expr* (proběhne iterace)
 - ▶ pokud ne, cyklus končí
- 3 na vyhodnotí se *c_iter*, pokračuje se bodem 2.

```
// iterate through 0 to 9
for (i = 0; i < 10; i++){
    b = cool_fn(i);
}
```

Řízení cyklu

Příkaz `continue` skočí na
konec cyklu

Příkaz `break` ukončí
provádění cyklu

```
while (a > 0){  
    // preskoc cisla  
    // delitelna 5  
    if(a % 5 == 0) continue;  
  
    // spocitej a odedti 1 od a  
    b += calculate(a--);  
  
    // pokud vychazi soucet  
    // zaporny, ukonci  
    if(b < 0) break;  
}
```

- izolovaná posloupnost příkazů
- fixní nebo variabilní počet argumentů
- návratová hodnota
- definice = funkční prototyp + tělo funkce
- deklarace = pouze funkční prototyp

Funkce - příklad

```
double pi_times (int num); // declaration - fn prototype
double circ (double r);
```

```
void main (void){ // funkce main
    double dia;
    dia = circ(3); // volani fce circ()
    ...
}
```

```
double circ (double r){ // definice fce circ()
    return pi_krat( 2 * r ) ;
}
```

```
double pi_krat (int num){ // definice fce pi_krat()
    return num * 3.14;
}
```

Modularita jazyka C

- skupiny funkcí se sdružují do modulů
- jeden modul = soubor *.c obsahuje:
 - ▶ definice funkcí (vnitřních i veřejných)
 - ▶ definice globálních proměnných (dtto)
 - ▶ definice lokálních maker, datových typů
- příslušný hlavičkový soubor *.h definuje vnější rozhraní modulu:
 - ▶ deklarace vnějších datových struktur a maker
 - ▶ deklarace vnějších proměnných
 - ▶ deklarace vnějších funkcí

Právě hlavičkové soubory programátorovi zprostředkovávají přístup ke funkcionalita standardní knihovny jazyka C.

Modularita - příklad

```
// circ.h
double pi_times (int num);
double circ (double r);
```

```
// circ.c
#include "circ.h"

double circ (double r){
    return pi_krat(2 * r);
}
```

```
double pi_krat (int num){
    return num * 3.14;
}
```

```
// main.c
#include "circ.h"

void main (void){
    double dia;
    dia = circ(3);
    ...
}
```

Preprocesor jazyka C

- 1. krok procesu překladač
- příkazy (direktivy) preprocesoru začínají znakem #
- zpracuje zdrojový text před překladem, odstraní komentáře
`#include <soubor.h>`, `#include "muj.h"`
- provede náhradu maker - symbolické konstanty a makra s parametry
`#define`, `#undef`
- podmíněný překlad
`#if` `#elif` `#else` `#endif`
`#ifdef` `#ifndef`
- řízení překladače
`#pragma`

Preprocesor: příklady

```
#define DELKADAT 34    // symbolická konstanta, bez ';'!  
int pole[DELKADAT];    // stejne jako pole[34]  
  
#define FLOAT_DATA 1  
#ifndef FLOAT_DATA    // podle toho, zda je konstatna definovana  
float data[DELKADAT];  
typedef my_data_type float;  
#else  
int data[DELKADAT];  
typedef my_data_type int;  
#endif  
  
#include <stdio.h>
```

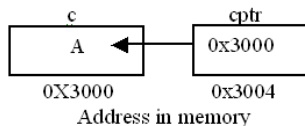
Složené proměnné a jiní příbuzní

- ukazatel – nenese hodnotu, ale odkaz na jinou proměnnou
- pole – složený typ, indexovaná posloupnost (od nuly) položek stejného datového typu
- řetězec (znaků) – realizován jako pole typu `char`, ukončené nulovým znakem `'\0'`
- uživatelsky definované datové typy
- `struct` – složený typ, sdružuje více položek (pojmenovaných) různých typů
- `enum` – celočíselný výčet, pojmenované konstanty
- `union` – umožňuje přístup k jednomu paměťovému místu jako k různým datovým typům

Ukazatele (pointers)

- **Ukazatel** je proměnná, která nese odkaz (adresu) jiné proměnné (struktury, funkce)
- typované: "ukazatel na (typ) XXX"
- lze je přetypovat
- rozsah odpovídá architektuře platformy

```
char c = 'A';  
char *cptr;  
cptr=&c;
```



Ukazatele jsou klíčovým prvkem jazyka C; jejich pochopení je branou k pochopení celého konceptu jazyka!

Ukazatele a operátory

- **referenční operátor** &: &var znamená adresu proměnné var
- **dereferenční operátor** *: *ptr znamená hodnotu, uloženou na adrese ptr

```
int i = 1, j;    // i a j jsou typu int
int *p_i;        // pi je ukazatel na typ int
p_i = &i;        // uloz adresu i do p_i
*p_i = 5;        // to stejne jako i = 5;
```

Poznámka: hvězdička v definici ukazatele a dereferenční operátor jsou dvě různé věci!

Práce s ukazateli

S ukazateli lze počítat:

- porovnávat (`==`, `!=`, `<`, `>`): `if(p1 == p2) { ...`
- odčítat (`p1 - p2`)
- přičítat celé číslo integer: `*(p + n)`, `p++`, `p--`
Pro `int *p` \Rightarrow `*(p+n)` ukazuje na n -tý prvek typu `int` za adresou n .

Ukazatel může být argumentem/návratovou hodnotou funkce:

```
int prohod(int *a, int* b); //volani odkazem
int* vetsi(int *a, int* b);
```

Ukazatel může být **prázdný** a/nebo **prázdného typu**:

```
void *ptr;          // prazdny typ
int *pi = NULL;     // specielni nulova hodnota ukazatele
```

Pole

- homogenní posloupnost prvků stejného typu
- dobře uspořádaná, indexovaná od nuly
- prvky jsou uloženy v paměti v pořadí za sebou, meze nejsou hlídány překladačem
- hranaté závorky pro definici i pro přístup k prvku pole

```
int a[10]; // definujeme pole o 10 prvcích
a[1] = 6;  // prirad hodnotu druhého prvku
          // (první is a[0])
a[10] = -1; // tzv. off-by-one error
int prime[3] = {5, 7, 11}; // pole vc. inicializace
```

Pozor! Pole **nej**sou primární datatyp, nelze je přiřazovat:

```
pole1 = pole2 // no f***ing way!!!
```

Pole a ukazatele (!!!)

Pro `int a[3] = 3,5,6; int *pa = &(a[0]);` platí následující:

- `a[0] == *pa` – hodnota 1. prvku
- `a[1] == *(pa + 1)` – hodnota 2. prvku
- `&(a[n]) == pa + n` – hodnota n . prvku

Proč? Konstrukce `a[0]` se vskutečnosti skládá z:

- konstantního **ukazatele** `a`
- **operátoru** pro přístup k prvku pole `[]`

Proto dále platí:

- hodnota `a[0] == *a == *pa == pa[0]`
- hodnota `a[n] == *(a + n) == *(pa + n) == pa[n]`
- ukazatel `&(a[n]) == a + n == pa + n`

Pole a funkce

- pole může být argumentem i návratovou hodnotou funkce
- v obou případech je předává **pouze ukazatel** (!)
- demonstrace, proč C **nekontroluje velikost polí** – principiálně ani nemůže

```
int max(int *in, int pocet){
    int i, max = in[0];
    for (i = 1; i < pocet; i++)
        if(in[i] > max) max = in[i];
    return max;
}
```

```
int mojedata[7] = {1,5,388,5,3,5,7};
int nejveci = max(mojedata, 7);
```


V jazyce C jsou řetězce realizovány jako pole typu `char`, ukončené nulovým znakem `'\0'`.

```
char pozdrav[6] = "Ahoj!"; // znak navíc kvůli '\0'
char pozdrav2[] = "Zdar!"; // prekladace dosadí delku
char pozdrav3[] = {'c','a','u','\0'}; // po znacích
```

```
pozdrav[2] = 'g'; // pozdrav obsahuje "Agoj!"
```

```
int delka(char *str){ // funkce pro práci s řetězcem
    int i = 0, len = 0;
    while(str[i++] != '\0') len++;
    return len;
}
```

Struktury I.

- obsahuje množinu pojmenovaných položek, obecně různého typu
- **Syntaxe:**

```
struct struct_name { entries } var_name(s) ;  
typedef struct { entries } type_name ;
```

```
struct { // anonymní struktura, def. promenne  
    int vyska;  
    float vaha;  
    int vek;  
} anna, bara, dana;
```

```
struct gf{ // pojmenovaná struktura  
    int vyska;  
    float vaha;  
    int vek; };  
struct gf anna, bara, dana;
```

Struktury II.

```
// define new datatype
typedef struct gf{    // definovany datovy typ
    int vyska;
    float vaha;
    int vek;
    char prezdivka[25];
} t_gf;

t_gf anna, bara, dana;    // define variables
t_gf *bff;    // define pointer

// init
t_gf simona = {185, 75.8, 22, "Simi"};
t_gf market = {.prezdivka = "Dracice", .vyska = 168,
               .vaha = 55.2, .vek = 20 };
```

Struktury a operátory

```
// op pro pristup k prvkum:  
anna.vek = 21; // staticky  
bara.vyska = 170;  
  
bff = &dana;      // pomoci ukazatele  
bff->weight = 55;  
(*bff).age = 22;  
  
anna = bara;      // strukturu lze prirazovat  
  
t_gf cizinky[15]; // pole struktur  
cizinky[12].height = 145;  
  
t_gf *bff_cizi;  
bff_cizi = &foreigners[10];  
bff_cizi->age = 14;
```

Struktury a funkce

```
// volani hodnotou
void print_age (t_gf kamoska){
printf("Vek je %d\\d",kamoska.vek);
}
print_age(bara);

// volani odkazem
void print_age (t_gf *kamoska){
printf("Vek je %d\\d",kamoska->vek);
}
print_age(&market);
}
```

Uniony, výčtové typy

Výčtové typy: celá čísla s pěknými jmény

```
enum semafor { red, green, blue}; // values 0, 1, 2
enum semafor t1;    // promenna t1
t1 = red;           // prirazeni
int a = red;        // tez mozno, slabe typovano
```

Uniony: podobné strukturám, uschovává pouze jeden element

```
union vyska {
    long simpleDate;
    double perciseDate;
} mojevyska ;
mojevyska = 10;    // long
mojevyska = 10.34; // double
```

Typové modifikátory

```
const int ci = 5; // nelze priradit jinou hodnotu
volatile int a;   // asynchronni pristup k promenne

a = sizeof(struct gf); // operator urci delku v bytech

// funkce se timto zavazuje nemenit predavanou hodnotu
int fce (const char *str){
static int i; // static ve fci udrzuje hodnotu
...

// globalni static nebude viditelny vne modulu
static int globalData;
```

Standardní knihovna

Sada knihoven pro typické úlohy:

- `stdio.h` – core input and output functions
- `stdlib.h` – numeric conversion functions, pseudo-random numbers generation functions, memory allocation, process control functions
- `string.h` – string handling functions.
- `math.h` – common mathematical functions.

```
#include <stdio.h>
printf("Hello world!");
```

```
#include <math.h>
float a = sin (1.5 * M_PI);
```


Standardní knihovna: příklady

```
#include <stdio.h>
int getchar(void); // read character from standard input
int putchar(int c); // put character from standard input
int puts(const char *str); //put string
    int printf(const char *format, ...); // formatovany vystup

#include <string.h>
// copy src to dest (aka dest = src)
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src, size_t n); // secure
size_t strlen(const char *s); // vrati delku retezce

#include <stdlib.h>
int atoi(const char *nptr); // prevod retezce na cislo int
long atol(const char *nptr); // prevod retezce na long
void *malloc(size_t size); // dynamicka alokace pameti
void free(void *ptr); // a její uvolnění
```

Na co zatím nedošlo

- detailní specifikace operátorů: bitové
- doporučená struktura hlavičkových a zdrojových souborů
- štábní kultura
- práce s pamětí
- statická/dynamická 2D pole
- proces kompilace
- nedefinované chování
- ukazatele na funkce

- <https://www.studytonight.com/c/overview-of-c.php>
- <https://www.studytonight.com/c/programs/>
- Herout, P: Učebnice jazyka C
- Doplnkové materiály k předmětu

Podpora C pro embedded I.

- překladač vždy pro konkrétní platformu (vendor-specific), IDE zpravidla taktéž – podpora datových typů
- standardní knihovna implementuje část standardních knihoven – např. chybí vlákna, práce s časem
- standardní funkce často příliš náročné – např. `printf()`
- některé standardní funkce částeční DIY – standardní I/O
- doplněny funkce pro řízení periférií – EEPROM, PWM, SPI, UART, ...
- jazykové konstrukce pro konfigurační pojistky – `#pragma WDT_ON`
- triviální názvy pro SFR

Kde hledat informace

- Logické obvody, syntaxe jazyka C, standardní knihovna C – ST*G
- Podpora C pro MCU, knihovny fce pro MCU – manuál k překladači
- Architektura a periferie MCU – datasheet MCU, family datasheet
- Zapojení periférií v konkrétním HW – manuál ke kitu