Obsluha přerušení

Digitální elektronika 2

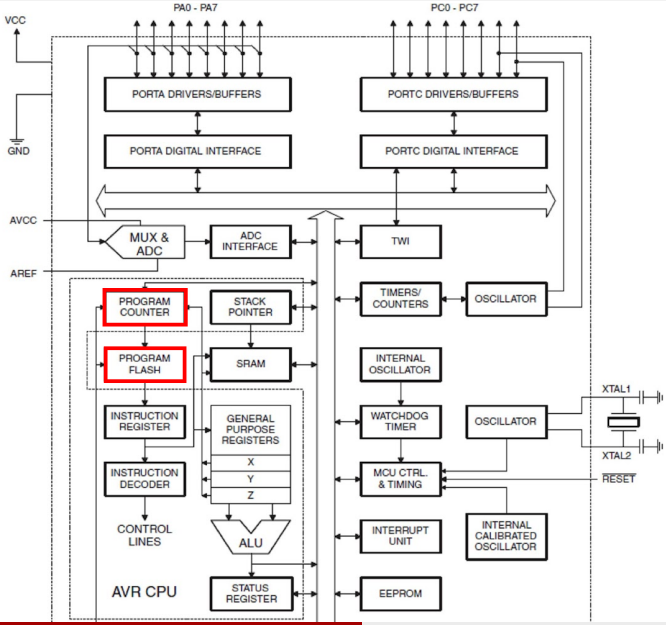
doc. Ing. Tomáš Frýza, Ph.D.

říjen 2020

Obsah přednášky

Obsah přednášky

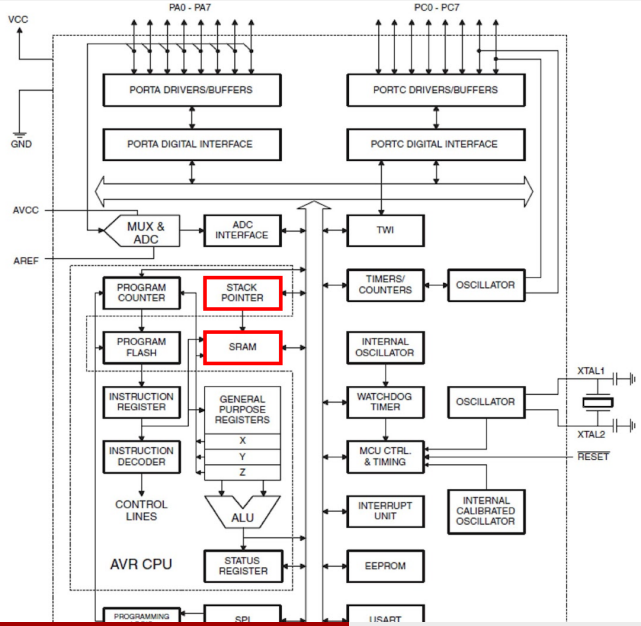
Orientace procesoru v programu: programový čítač



- Mikrokontrolér je řízen instrukcemi uloženými v programové paměti. Např. ATmega8 má kapacitu Flash 8 kB, ATmega16 má 16 kB Flash, Atmega 328P má 32 kB Flash, ...
- Podle von Neumannovy koncepce procesoru jsou instrukce vykonávány sekvenčně, tj. tak jak jsou uloženy v programové paměti
- Adresa instrukce, která se vykonává specifikuje u všech procesorů ukazatel **programový čítač** (Program Counter)
- PC je obecně inkrementován; kromě podmínek, nepodmíněných skoků, volání podprogramů a obsluh přerušení, kdy dochází ke skokové změně PC

Obrázek: Blokové schéma mikrokontrolérů ATmega16/32

Dočasné odkládání dat do datové paměti typu ZÁSOBNÍK



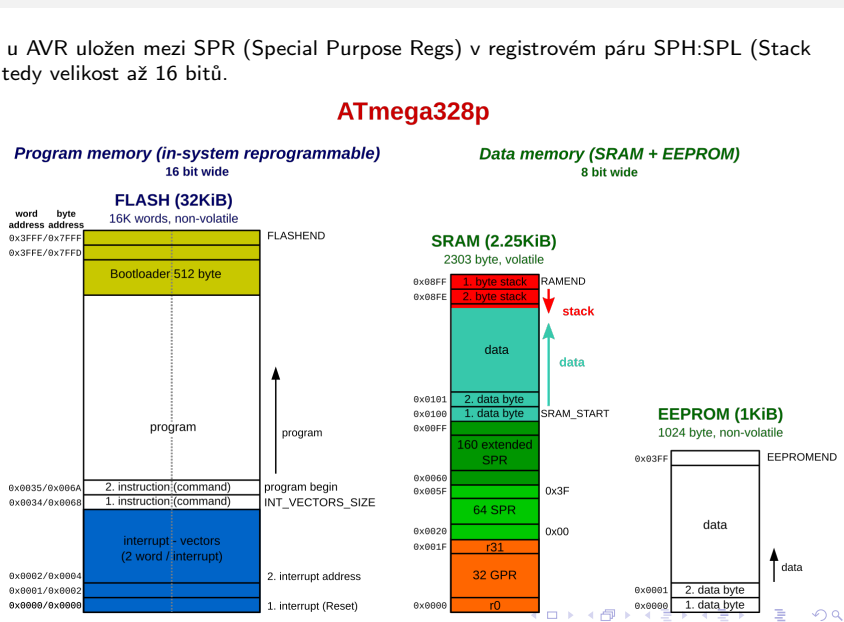
- Zpracovávaná data jsou uložena v pracovních registrech r0–r31. Lze je kdykoliv uložit kdekoliv do datové paměti SRAM nebo dočasně na část datové paměti SRAM, která funguje jako zásobník (Stack).
- Zásobník nemá fixní velikost: při ukládání se zvětšuje, při vyčítání zmenšuje. Pokud není nic uloženo, má nulovou velikost.
- Zásobník má jediný "přístupový bod"k datům: tzv. vrchol zásobníku (ToS, Top-of-Stack). Při ukládání/vyčítání dat se adresa ToS mění a je určována **ukazatelem na zásobník** (Stack Pointer).
- Pozn.: Mikrokontroléry, které nemají interní paměť RAM mají hardwarový zásobník s omezenou velikostí (např. tři pozice u ATtiny11).

Ukazatel na zásobník u AVR

- Ukazatel na zásobník je u AVR uložen mezi SPR (Special Purpose Regs) v registrovém páru SPH:SPL (Stack Pointer High:Low). Má tedy velikost až 16 bitů.

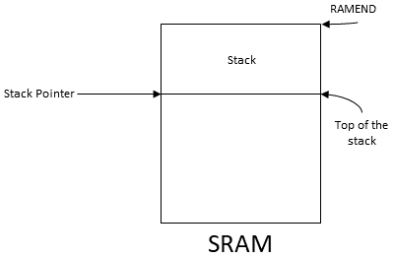
- Hodnota ukazatele na zásobník v SPH:L musí být na začátku aplikace inicializována; používá se adresa RAMEND.

- Protože je zásobník u AVR plněn směrem k nižším adresám, definuje se až na poslední pozici v SRAM; např. 0x08FF u ATmega328P.



Princip zápisu 16bitových dat do zásobníku

- Zásobník funguje jako LIFO systém (Last In First Out – poslední zapsán, první čten)
- Fáze zápisu 16bitové adresy do 8bitového zásobníku:
  (1) Uložení nižšího bytu adresy na pozici adresovanou SP
  (2) SP = SP - 1
  SP tak obsahuje adresu volného bytu, kam je možné dále zapisovat
  (3) Uložení vyššího bytu adresy na pozici adresovanou SP
  (4) SP = SP - 1
  Po ukončení procesu tedy SP vždy obsahuje adresu volného bytu v zásobníku



Example

Jaká data obsahuje zásobník po volání dvou vnořených podprogramů? Nechť byly volány z adres 0x0021 a 0x0029 a zásobník začíná na adrese 0x08FF?

Řešení

Adresa SRAM	Stack before	Stack after
0x08FF	*0xff*	*0x22*
0x08FE	*0xff*	*0x00*
0x08Fd	*0xff*	*0x2a*
0x08FC	*0xff*	*0x00*
0x08Fb	*0xff*	*0xff*
	...	...

Proces čtení 16bit. dat ze zásobníku

- Fáze čtení adresy ze zásobníku:
  - (1) SP = SP + 1
    Ukazatel tak obsahuje adresu posledního bytu uloženého v zásobníku
  - (2) Adresovaný vyšší byte návratové adresy je přesunut do PC
  - (3) SP = SP + 1
    Ukazatel adresuje předposlední uložené slovo
  - (4) Adresovaný nižší byte návratové adresy je přesunut do PC

Pozn.: Uložená data ve Stacku zůstávají, ale při následném zápisu dat se přepíší

Example

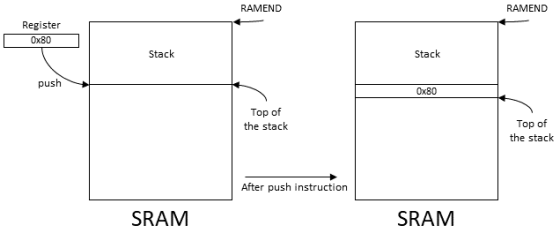
Jaká data obsahuje ukazatel na zásobník a programový čítač po návratu ze dvou vnořených podprogramů (viz předchozí příklad)?

Řešení

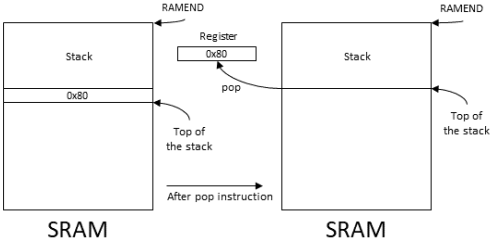
Byl-li proveden návrat ze všech podprogramů, obsahuje ukazatel na zásobník původní hodnotu danou při definování zásobníku (např. $08FF) a programový čítač "pokračuje" adresou 0x0022.

Práce se zásobníkem, ukládání pracovních registrů

- Kromě zálohy 16bit. adres (typicky návratové adresy při volání podprogramu nebo přerušení) lze zásobník využít také jako dočasné odkladiště obsahu pracovních registrů r0 až r31. Slouží k tomu instrukce `push` a `pop`.
- Toho je často používáno na začátku podprogramu, nebo obsluhy přerušení. Na konci (před návratem) jsou hodnoty opět překopírovány do původních registrů, tj. dojde k obnově původního stavu registrů. (Při programování v C toto zajišťuje kompilátor!)



Obrázek: Použití instrukce `push` pro uložení registru do Stacku

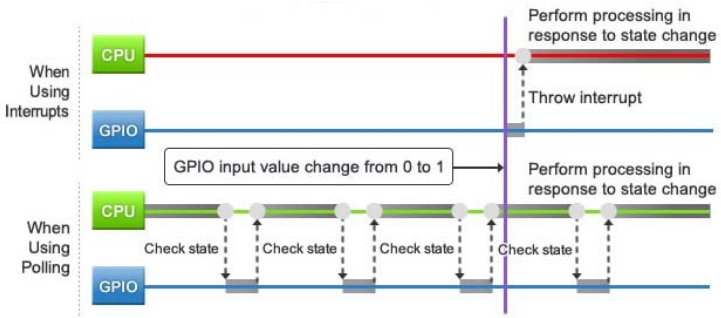


Obrázek: Použití instrukce `pop` pro vyzvednutí bytu ze Stacku do registru

Obsah přednášky

Obsluha přerušení

- Stav nepřetržitého monitorování libovolného parametru se nazývá **polling** (dotazování). Mikrokontrolér stále kontroluje stav ostatních zařízení a přitom neprovádí jinou operaci.
- Je to jednoduchý způsob, jak zkontrolovat změny stavu. Pokud je ale interval kontroly příliš dlouhý, může dojít k dlouhému zpoždění mezi výskytem, případně může úplně uniknout změna, pokud se stav před kontrolou změní zpět. Kratší interval získá rychlejší a spolehlivější detekci, ale také spotřebuje mnohem více času a energie na zpracování, protože mnohem více kontrol je "zbytečných".
- Alternativním přístupem je využití **interrupt** (přerušení). Zde změna stavu generuje signál přerušení, který způsobí, že CPU pozastaví svou aktuální operaci (a uloží svůj aktuální stav), poté provede zpracování spojené s přerušením a následně obnoví svůj předchozí stav a pokračuje tam, kde přestal.



Zdroje a vektory přerušení ATmega328 a ATmega328P

- Každý procesor obsahuje sadu možných přerušení interních i externích; zdroje přerušení závisí na hardwarovém vybavení konkrétního MCU.
- V případě, že mikroprocesor obdrží žádost o přerušení, přeruší vykonávanou činnost (hlavní program) a spustí "obsluhu přerušení", která je určena výhradně pro jeden zdroj přerušení.
- Obsluha každého přerušení je dána tzv. vektorem přerušení v programové paměti; jedná se o adresu, od které se začne vykonávat konkrétní obsluha.

Č.	Adresa	Zdroj přerušení	Popis přerušení
1	0x0000	RESET	Externí pin, Power-on reset, Brown-out reset, Watchdog reset
2	0x0002	INT0	Externí požadavek na přerušení 0
3	0x0004	INT1	Externí požadavek na přerušení 1
4	0x0006	PCINT0	Externí požadavek při změně pinu 0
5	0x0008	PCINT1	Externí požadavek při změně pinu 1
6	0x000A	PCINT2	Externí požadavek při změně pinu 2
7	0x000C	WDT	Přetečení Watchdog časovače
8	0x000E	TIMER2 COMPA	Časovač/čítač 2 – shoda s komparátorem A
9	0x0010	TIMER2 COMPB	Časovač/čítač 2 – shoda s komparátorem B
10	0x0012	TIMER2 OVF	Časovač/čítač 2 – přetečení
11	0x0014	TIMER1 CAPT	Časovač/čítač 1 – událost zachycení
12	0x0016	TIMER1 COMPA	Časovač/čítač 1 – shoda s komparátorem A
13	0x0018	TIMER1 COMPB	Časovač/čítač 1 – shoda s komparátorem B
14	0x001A	TIMER1 OVF	Časovač/čítač 1 – přetečení
15	0x001C	TIMER0 COMPA	Časovač/čítač 0 – shoda s komparátorem A
16	0x001E	TIMER0 COMPB	Časovač/čítač 0 – shoda s komparátorem B
17	0x0020	TIMER0 OVF	Časovač/čítač 0 – přetečení

Obsluha přerušení

Jednotlivé fáze obsluhy přerušení:

(1) dokončí se výkon právě vykonávané instrukce

(2) do zásobníku se uloží adresa následující instrukce v programové paměti (zásobník musí být definován)

(3) podle zdroje přerušení se do PC načte vektor přerušení (např.: PC=0x0002 pro INT0)

(4) vykoná se obsluha přerušení, tj. konkrétní posloupnost instrukcí; do oblasti vektorů přerušení se běžně umísťují pouze skoky na obslužnou funkci

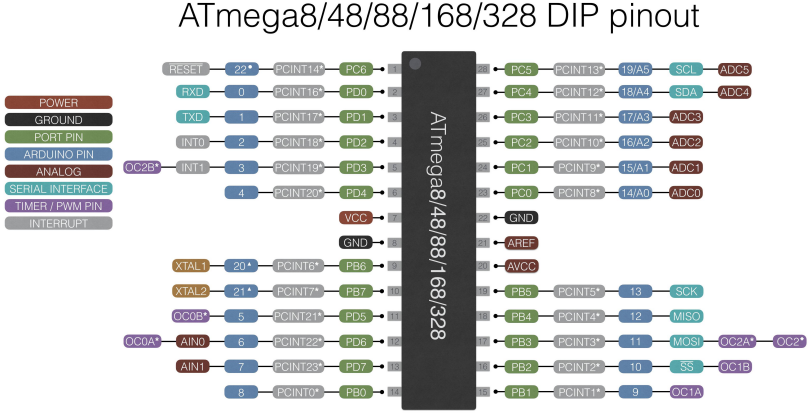
(5) obsluha přerušení se ukončí instrukcí `reti` (RETurn Interrupt); analogie s ukončením podprogramu

(6) do PC se načte uložená návratová adresa ze zásobníku

(7) pokračuje se ve výkonu hlavního programu

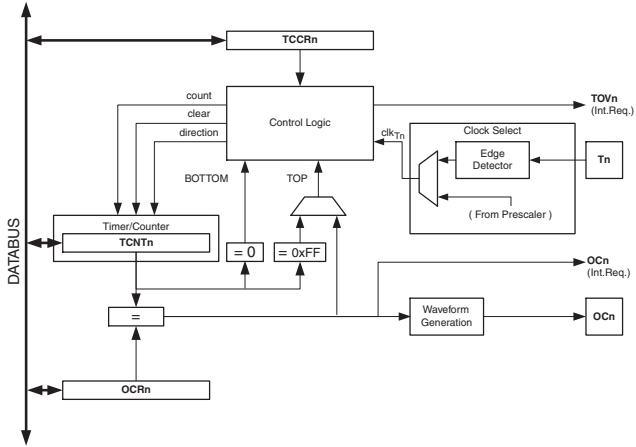
Příklad obsluhy přerušení, externí zdroj přerušení

- Přerušení se vykoná (obslouží) v případě že: (1) je povoleno konkrétní přerušení a (2) je povoleno globální přerušení instrukcí `sei`
- ATmega328P umožňuje dva typy externích přerušení: *External Interrupts* (INT0, INT1: lze aktivovat při sestupné hraně, nástupné hraně, libovolné změně a při trvající nízké úrovni) a *Pin Change Interrupts*, u kterých jsou I/O piny sdruženy do tři skupin (PCINTx: lze aktivovat změnou stavu pinu).



Obsah přednášky

Funkce interního časovače/čítače



Obrázek: Blokové schéma 8bitového časovače/čítače (označení "n" udává číslo čítače)

- Každý mikrokontrolér obsahuje periférii umožňující odečítat čas (časovač, anglicky: Timer), příp. načítat vstupní pulsy (čítač, anglicky: Counter)
- Toho lze využít k jednoduchým aplikacím:
  - měření krátkých intervalů, periodické spouštění funkce,
  - konstrukce jednoduchých frekvenčních čítačů,
  - ...
- Podstatou obou funkcí je inkrementace interního datového registru pomocí hodinového, příp. externího signálu
- ATmega328P obsahuje dva 8bitové časovače (interní označení Timer/Counter 0 a 2) a jeden 16bitový časovač (Timer/Counter 1)

Blokové schéma 8bitového časovače/čítače 0, viz manuál ATmega328P

- Časovač/čítač 0 je 8bitový obsahuje tři datové/kontrolní registry pomocí kterých lze periférii využívat:
  - TCNT0 (Timer/Counter Registr) – datová hodnota časovače/čítače 0
  - OCR0A (Output Compare Register A) – hodnota pro neustálé porovnávání s datovým registrem
  - OCR0B (Output Compare Register B) – druhá hodnota pro neustálé porovnávání s datovým registrem
  - TCCR0A (Timer/Counter Control Register A) – řídicí registr, výběr čítacího módu
  - TCCR0B (Timer/Counter Control Register B) – řídicí registr, nastavení předděličky hodinového signálu, výběr čítacího módu
  - TIMSK0 (Timer/Counter Interrupt Mask Register) – povolení přerušení
  - TIFR0 (Timer/Counter Interrupt Flag Register) – příznakové bity událostí, tj. že nastala událost, která může vyvolat přerušení
- Periférie (její řídicí logika) má schopnost generovat přerušení při události:
  - přetečení časovače/čítače, tj. při dosažení maximální hodnoty a znovu návrat k minimální hodnotě
  - rovnosti hodnot datového a komparačního registru

Hodinový signál časovače/čítače 0

- Periférie je řízena vnitřním zdrojem hodinového signálu, nebo externím signálem z pinu T0. Interní signál lze odvodit od hodinového signálu jádra mikrokontroléru a zpomalit interní děličkou.
- Dělička ATmega328P obsahuje pět konkrétních hodnot, kterými lze frekvenci dělit: 1, 8, 64, 256, 1024. Pozn.: Timer/Counter2 jich umožňuje nastavit 7: navíc ještě 32 a 128.

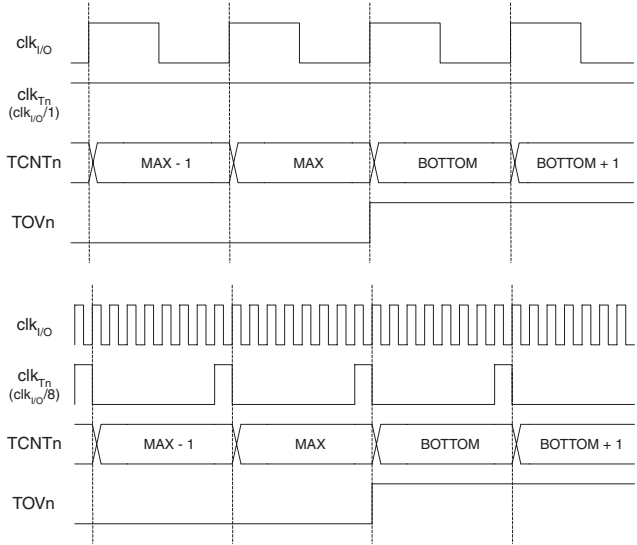
Tabulka: Nastavení děličky 8bitového časovače/čítače 0 v kontrolním registru TCCR0B, viz manuál

CS02:0	Popis funkce
0 0 0	Hodinový signál odpojen, tj. časovač/čítač 0 nepracuje
0 0 1	Bez děličky: f_{CPU}
0 1 0	Dělička 8: $f_{CPU}/8$
0 1 1	Dělička 64: $f_{CPU}/64$
1 0 0	Dělička 256: $f_{CPU}/256$
1 0 1	Dělička 1 024: $f_{CPU}/1\,024$
1 1 0	Externí zdroj řídicího signálu, reakce na sestupnou hranu
1 1 1	Externí zdroj řídicího signálu, reakce na náběžnou hranu

Example

Za jak dlouho dojde k přetečení 8bitového časovače/čítače 0 ATmega328, jsou-li bity CS02:CS00=010 a $f_{clk} = 1\,\text{MHz}$?

Doba přetečení časovače/čítače



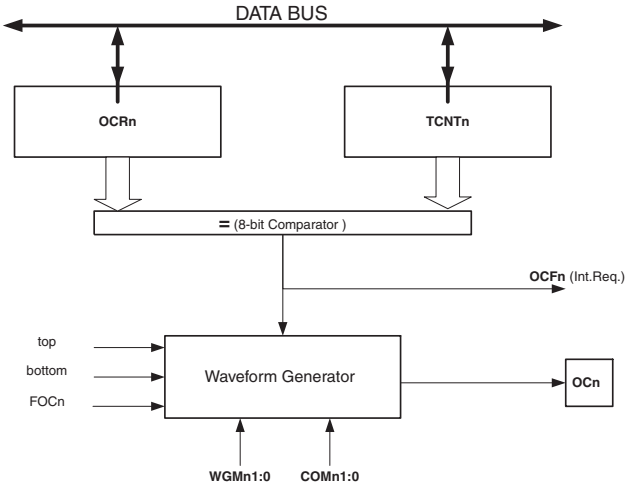
- Doba přetečení *n* bitového časovače t_{OVF}, kde f_{CPU} je frekvence hodinového signálu jádra mikrokontroléru a N je hodnota děličky:

$$t_{OVF} = \frac{1}{f_{CPU}} \cdot N \cdot 2^n \qquad (1)$$

- Doba přetečení *n* bitového časovače s nenulovou počáteční hodnotou $init$ v datovém registru TCNT0:

$$t_{OVF} = \frac{1}{f_{CPU}} \cdot N \cdot (2^n - init) \qquad (2)$$

Funkce porovnávání (komparace) časovače/čítače 0

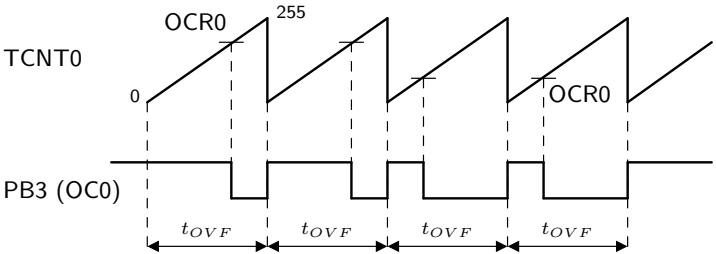


Obrázek: Princip porovnávání komparační a datové hodnoty časovače/čítače AVR

- Při každé změně datového registru časovače TCNTn je jeho hodnota porovnávána s komparačním registrem OCRn.
- Jestliže se obsahy obou registrů rovnají, lze generovat přerušení (pokud je řádně povoleno) a/nebo generovat PWM signál (Pulse Width Modulation) na výstupním pinu.
- Lze konfigurovat různé módy PWM:
  - Normal mode,
  - Clear Timer on Compare mode (CTC),
  - Fast PWM mode a
  - Phase Correct PWM Mode.

Časovač/čítač 0, funkce generováni PWM

- Informace je PWM signálem přenášena proměnnou střídou obdélníkového signálu s konstantní periodou
- Princip činnosti Fast PWM módu:
  - přetečení časovače: nastaví se vysoká úroveň generovaného signálu (tj. začátek periody)
  - při rovnosti datového a komparačního registru: nastaví se nízká úroveň



Obrázek: Symbolické znázornění inkrementované hodnoty datového registru TCNT0 a princip generování PWM signálu pomocí změny hodnoty komparačního registru OCR0

Obsah přednášky

Obsluha přerušení v jazyce C pro AVR

- Viz User Manual: `https://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html`
- Pro využití přerušení v jazyce C je nutné vložit hlavičkový soubor `interrupt.h`
- Není nutné definovat zásobník; respektive udělá to překladač. Rovněž není nutné znát konkrétní ADRESY vektorů přerušení
- Obsluhy přerušení představují vždy makra `ISR` (anglicky: Interrupt Service Routine) se vstupním parametrem, který identifikuje zdroj přerušení, např.:
  - INT0_vect – externí přerušení
  - ADC_vect – přerušení od A/D převodníku
  - ...
- POZOR: Všechny informace a ukázky využívají překladače GCC s knihovnou avr-libc. Pro jiné překladače bude syntaxe odlišná!

Obsluha přerušení v jazyce C pro AVR

- Základní struktura aplikace pro AVR s hlavní funkcí `main()` a obsluhou přerušení `ISR()`
- Vložit knihovnu `avr/interrupt.h`
- Povolení dílčích přerušení
- Globální povolení přerušení pomocí `sei()`
- Nekonečná smyčka často neobsahuje žádné příkazy. CPU zde "čeká" na přerušení

```
#include <avr/io.h>
#include <avr/interrupt.h>

int main(void)
{
    /* Set output pin 13 (PB5) */
    DDRB |= (1<<PB5);
    /* Turn LED off */
    PORTB &= ~(1<<PB5);

    /* Clock prescaler 1024
     * t_ovf = 1/f_cpu * 2^n * N
     * t_ovf = 1/16e6 * 256 * 1024 = 16 ms */
    TCCR0B |= (1<<CS02) | (1<<CS00);

    /* Overflow interrupt enable */
    TIMSK0 |= (1<<TOIE0);

    sei();                  // Enable interrupts
    while (1);              // Empty forever loop

    return 0;
}

ISR(TIMER0_OVF_vect)    // Interrupt service routine
{
    /* Toggle LED */
    PORTB ^= (1<<PB5);
}
```