Ústav radioelektroniky
Vysoké učení technické v Brně

Programování mikrokontrolérů

Mikroprocesorová technika a embedded systémy

Přednáška 3

doc. Ing. Tomáš Frýza, Ph.D.

říjen 2012

Obsah přednášky

Typy adresování
- Registrové, přímé, nepřímé, bezprostřední

Větvení programu
- Programový čítač
- Volání podprogramu, zásobník

Obsluha přerušení
- Externí požadavek na přerušení

Vnitřní struktura a využití základních periférií
- Vstupně/výstupní port
- Přímé připojení, maticové, pomocí pomocných obvodů
- Časovač/čítač

Ukázka programu v JSA a v jazyce C pro ATmega16
- Obsluha přerušení

Obsah přednášky

Typy adresování
- Registrové, přímé, nepřímé, bezprostřední

Větvení programu
- Programový čítač
- Volání podprogramu, zásobník

Obsluha přerušení
- Externí požadavek na přerušení

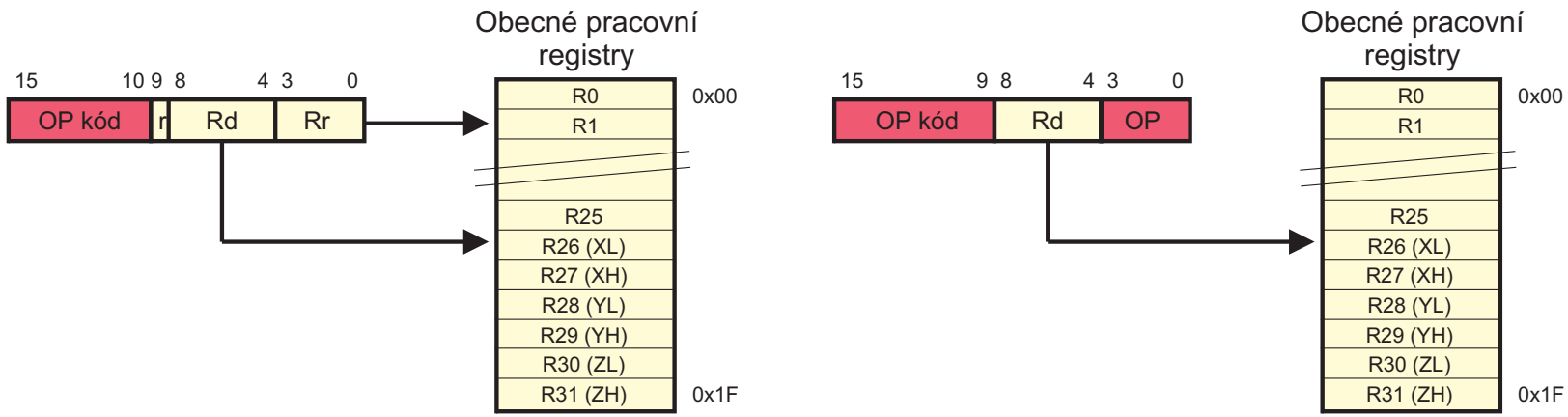
Vnitřní struktura a využití základních periférií
- Vstupně/výstupní port
- Přímé připojení, maticové, pomocí pomocných obvodů
- Časovač/čítač

Ukázka programu v JSA a v jazyce C pro ATmega16
- Obsluha přerušení

Typy adresování

- Proces adresování zpřístupňuje data z programové a datové paměti, příp. ze I/O jednotky.
- Terminologie:
  - přímé adresování – adresa je součástí instrukce. Přímo lze adresovat pracovní registry, řídicí (I/O) registry, datovou i programovou paměť,
  - nepřímé adresování – adresa není součástí instrukce. Adresa je uložena v některém z ukazatelů. Využívá se k opakovanému přístupu k jedné nebo k posloupnosti několika paměťových buněk.
- Mikroprocesorové systémy umožňují celou řadu způsobů adresování operandů. Základní způsoby jsou:

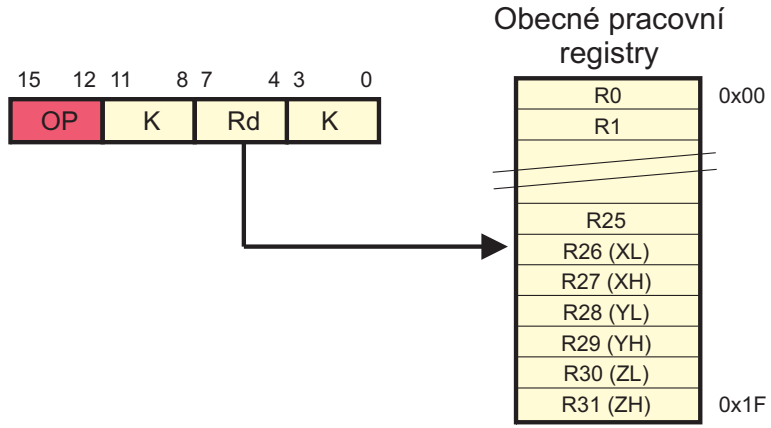
  (1) registrové adresování: zdrojem informace (operand) je registr(y). Např.: `MOV Rd, Rr`.



Obrázek: Adresování dvou (`ADD Rd, Rr`) a jednoho (`DEC Rd`) registru.

Typy adresování, pokračování

- Základní způsoby adresování, pokračování:
  - **(2)** přímé adresování: operandy jsou přímo určeny konkrétní adresou. Např.: `CALL k`,
  - **(3)** nepřímé adresování: adresa operandu je uložena v tzv. ukazateli, např. registrový pár X, Y nebo Z (X – R26:R27; Y – R28:R29; Z – R30:R31). Např. instrukce: `IJMP`, `ST X, Rr`.

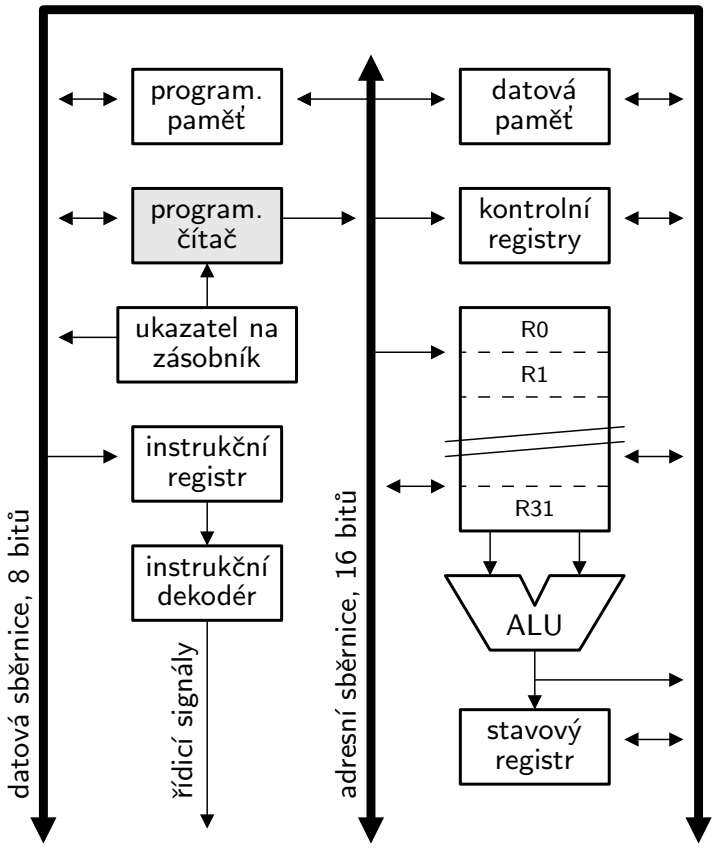


Obrázek: Bezprostřední adresování 8bitové konstanty (`LDI Rd, K`).

- Základní způsoby adresování, dokončení:
  - **(4)** bezprostřední adresování: hodnota operandu je přímo obsažena v instrukci. Např.: `LDI Rd, K`.
- Vícebitové procesory (DSP, ...) obsahují další typické způsoby adresování operandů. Různé způsoby adresování zvyšují možnosti mikroprocesorů.

Obsah přednášky

Větvení programu



Obrázek: Pozice programového čítače v jádře AVR.

- Podle von Neumannovy koncepce procesoru jsou instrukce vykonávány sekvenčně, tj. tak jak jsou uloženy v programové paměti.
- Adresu instrukce, která se bude vykonávat specifikuje programový čítač (PC – Program Counter).
- Při startu systému je hodnota PC vynulována. Dále je zpravidla pouze inkrementována.
- Při podmínkách, nepodmíněných skocích, volání podprogramu, obsluze přerušení dochází ke skokové změně hodnoty PC.

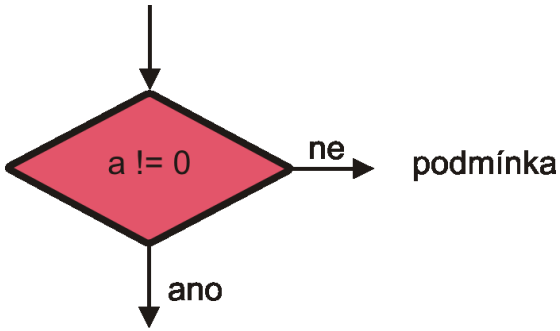
Programový čítač u AVR

15	14	13	12	11	10	9	8	
–	–	–	PC12	PC11	PC10	PC9	PC8	PCH
PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	PCL
7	6	5	4	3	2	1	0	

Obrázek: Struktura programového čítače v AVR – registrový pár PCH:L.

- Program Counter (PC) je u ATmega16 13bitový (PC12:PC0), tj. může adresovat celkem 2^{13}=8 k slov (instrukcí).
- PC je ukazatel na místo v programové paměti, tj. obsahuje adresu právě vykonávané instrukce.
- K hodnotám PC nelze přímo přistupovat (číst, zapisovat), protože tento čítač nelze adresovat!! Změnu hodnoty provádíme pouze nepřímo vykonanými instrukcemi.

Podmíněné skoky v programu



Obrázek: Obecná podmínka.

- Podmíněný skok je reakcí na vyhodnocení podmínky – TRUE/FALSE.
- JSA umožňuje využití jen jednoduchých podmínek.
- Dvě kategorie instrukcí podmíněných skoků:

(1) přeskoč následující instrukci (anglicky: Skip if ...),
`CPSE Rd, Rr` – přeskoč, je-li Rd=Rr (ComPare, Skip if Equal),
`SBIC P, b` – přeskoč, je-li bit "b" v I/O registru "P" nulový (Skip if Bit Is Cleared),
`SBIS P, b` – přeskoč, je-li bit "b" v I/O registru "P" nastaven (Skip if Bit Is Set),
...

(2) Skoč na návěští (anglicky: Branch if ...); houfně se k tomu využívají příznakové bity ze stavového registru,

`BREQ k` – skoč na návěští "k", je-li Z=1 (BRench if EQual),

`BRNE k` – skoč na "k", je-li Z=0 (BRench if Not Equal),

`BRVS k` – skoč na "k", je-li V=1 (BRench if V is Set), V-overflow,

`BRVC k` – skoč na "k", je-li V=0 (BRench if V is Cleared),

`BRIE k` – skoč na "k", je-li I=1 (BRench if I is Enable), I-interrupt,

`BRID k` – skoč na "k", je-li I=0 (BRench if I is Disable),

...

Nepodmíněné skoky v programu

- Instrukce pro nepodmíněné skoky přímo změní hodnotu v programovém čítači (PC) a tím se realizuje skok v programu. (Není uvažována žádna podmínka).
- Trojí typ instrukcí pro nepodmíněný skok u AVR:
  (1) relativní skok – `RJMP k` (Relative JuMP). Pozn.: Může skočit na adresu v rozsahu $-2K \leq k \geq 2K$ ($\Rightarrow$ instrukce je 16bitová, protože obsahuje 12bitovou adresu),
  (2) nepřímý skok – `IJMP` (Indirect JuMP) – nepřímý skok na adresu v registrovém páru "Z". Tj. registrový pár R31:30 je v tomto případě zdrojem adresy,
  (3) přímý skok – `JMP k` (direct JuMP). Pozn.: Může skočit na adresu v rozsahu $0 \leq k \geq 4M$ ($\Rightarrow$ instrukce je 32bitová, protože obsahuje 22bitovou adresu; platí pro systémy s 22bitovým PC).

Otázka

Jaký je rozdíl mezi instrukcemi `RJMP` a `JMP`? (Obsazenost paměti, rychlost výkonu.)

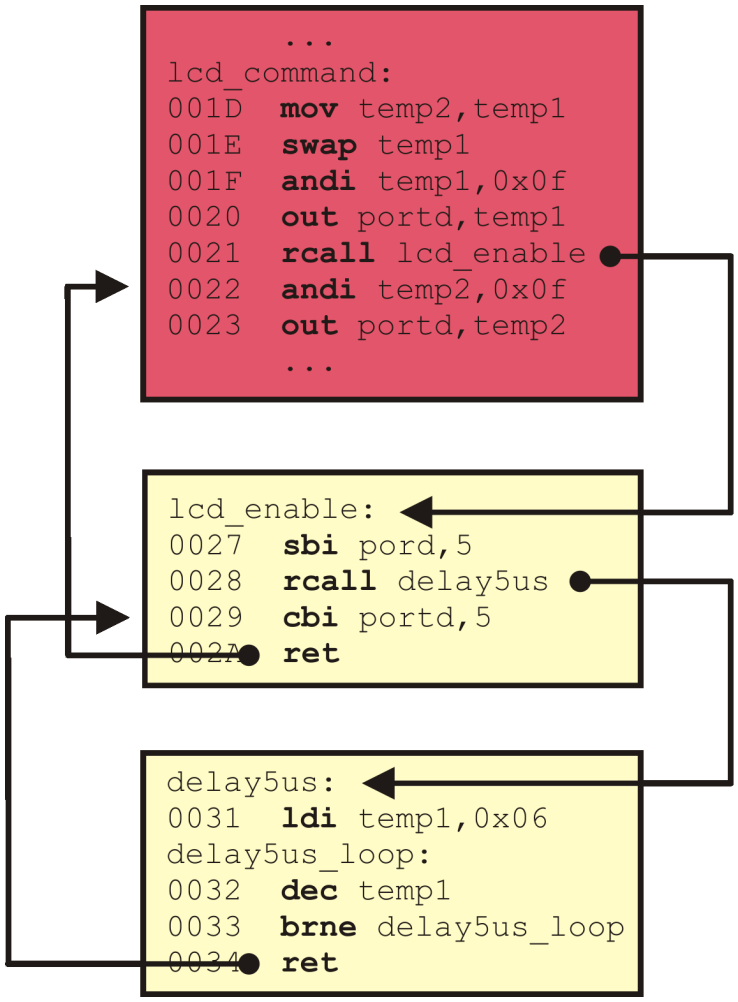
Význam podprogramu, volání podprogramu

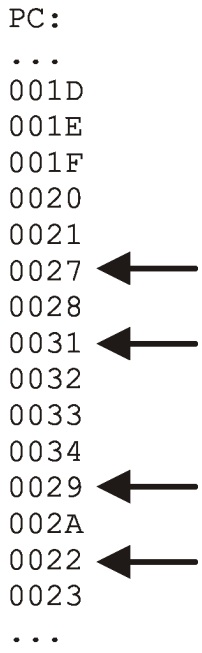
- Podprogram je část kódu, která se často opakuje; do strojového jazyka se přeloží jen jednou. Spouští se zapsáním adresy první instrukce do programového čítače, tj. "skáče" se na něj.
- Trojí typ instrukcí pro volání podprogramu u AVR:
  - **(1)** relativní volání – `RCALL k` (Relative subroutine CALL). Pozn.: Může skočit na adresu v rozsahu $-2K \leq k \geq 2K$ ($\Rightarrow$ instrukce je 16bitová, protože obsahuje 12bitovou adresu),
  - **(2)** nepřímé volání – `ICALL` (Indirect CALL) – nepřímé volání podprogramu; adresa vždy uložena v ukazateli "Z",
  - **(3)** přímé volání – `CALL k` (direct subroutine CALL). Pozn.: Může skočit na adresu v rozsahu $0 \leq k \geq 4M$ ($\Rightarrow$ instrukce je 32bitová, protože obsahuje 22bitovou adresu; platí pro systémy s 22bitovým PC).
- Na rozdíl od (ne)podmíněného skoku dojde po ukončení podprogramu k návrat na instrukci následující po skoku do podprogramu!

Otázka

Jak řídicí jednotka pozná, kam se vrátit??

Funkce zásobníku k uložení návratové adresy





- Návratová adresa musí být uložena před skokem do podprogramu do tzv. zásobníku (anglicky: Stack). Návratová adresa je adresa instrukce následující po instrukci volající podprogram.
- Návrat z podprogramu (vyzvednutí návratové adresy ze zásobníku a její zapsání do PC) zajišťuje instrukce `RET` (subroutine RETurn).

Obrázek: Vnořené volání dvou podprogramů.

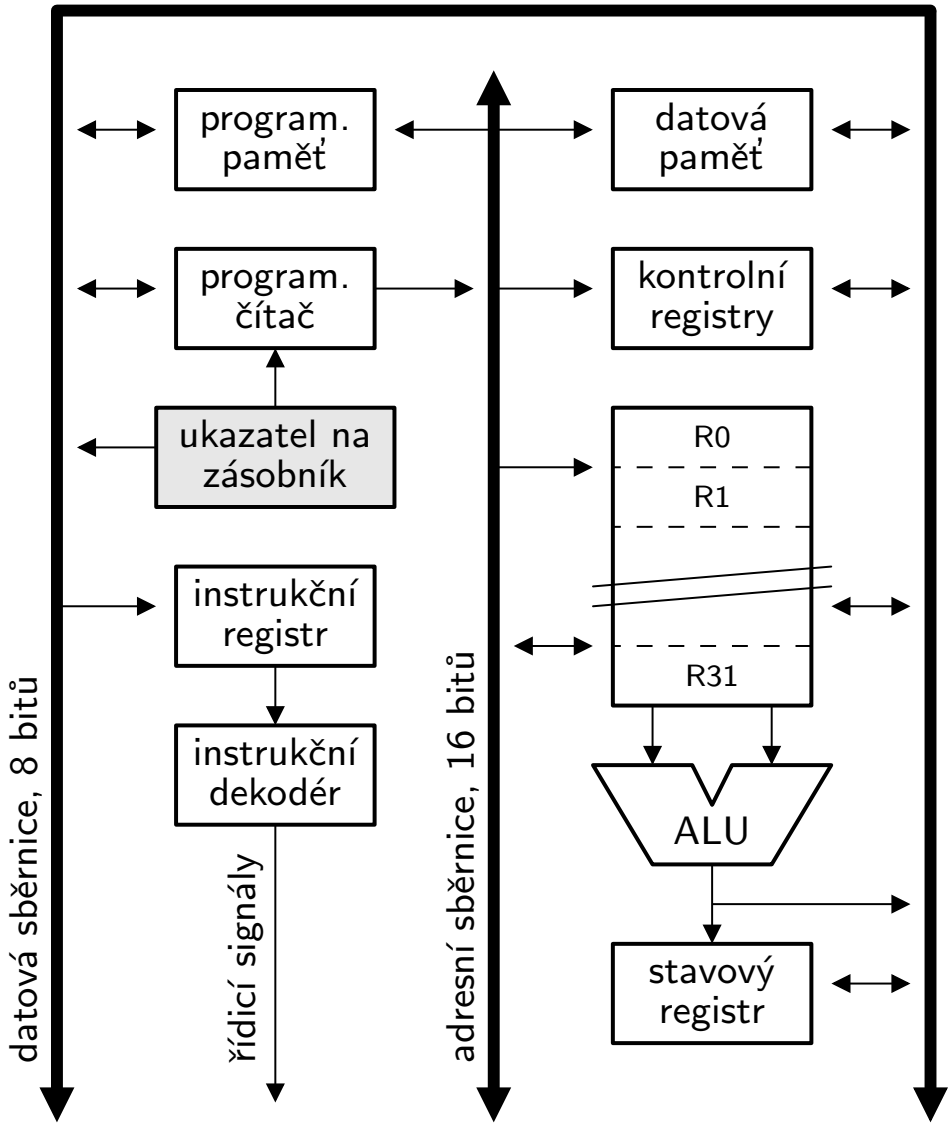
Zásobník, ukazatel na zásobník

- Zásobník je část paměti SRAM (může též sloužit k zálohování obsahu pracovních registrů). Mikrokontroléry, které nemají interní paměť RAM mají hardwarový zásobník s omezenou velikostí (např. tři pozice u ATtiny11).
- Zapisovat/číst lze ze zásobníku pouze do/z jediné pozice. Tu udává tzv. ukazatel na zásobník (anglicky: SP – Stack Pointer).
  - Ukazatel je 16bitová hodnota uložená v registrovém páru SPH:SPL. (Patří mezi I/O registry.)

15	14	13	12	11	10	9	8	
SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
7	6	5	4	3	2	1	0	

Obrázek: Struktura ukazatele na zásobník u AVR – registrový pár SPH:L.

Ukazatel na zásobník



Obrázek: Pozice ukazatele na zásobník v jádře AVR.

Ukazatel na zásobník

- Hodnota ukazatele na zásobník musí být inicializována (tzv. definice zásobníku). Definice se provádí jedenkrát a to hned na začátku programu.
- Ukazatel se naplní adresou poslední paměťové buňky paměťového segmentu určeného pro zásobník (zásobník se z důvodu eliminace nechtěného přepsání dat umísťuje na konec paměti RAM).
- (Konstanta RAMEND obsahuje adresu poslední buňky paměti SRAM – je definována v souboru *.inc.)

```
1  stack_definition:              ; definice zásobníku na konec paměti SRAM
2      LDI   R16, LOW(RAMEND)     ; do R16 ulož nižší byte poslední adresy SRAM
3      OUT   SPL, R16             ; SPL (nižší byte Stack Pointeru) = R16
4      LDI   R16, HIGH(RAMEND)    ; R16 = vyšší byte poslední adresy SRAM
5      OUT   SPH, R16             ; SPH (vyšší byte Stack Pointeru) = R16
```

Otázka

Co je to RAMEND?

Část výpisu definičního souboru `m16def.inc`

```
1  ;***** Specify Device
2  .device ATmega16
3
4  ;***** I/O Register Definitions
5  .equ    SREG    =$3f
6  .equ    SPH     =$3e
7  .equ    SPL     =$3d
8  .equ    OCR0    =$3c
9  .equ    GICR    =$3b          ; New name for GIMSK
10 ...
11 ;***** Bit Definitions
12
13 ; GIMSK / GICR
14 .equ    INT1    =7
15 .equ    INT0    =6
16 .equ    INT2    =5
17 .equ    IVSEL   =1
18 .equ    IVCE    =0
19 ...
20 .equ    RAMEND =$45F
21 ...
```

Proces zápisu 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).

Příklad

Jaká data obsahuje zásobník po volání dvou vnořených podprogramů? Nechť byly volány z adres 0x0021 a 0x0028 instrukcí `RCALL`.

Proces čtení 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 (nutnost maskování 3 nejvýznamnějších bitů, protože PC je pouze 13bitový),
  (3) SP = SP + 1 (ukazatel adresuje předposlední uložené slovo),
  (4) adresovaný nižší byte návratové adresy je přesunut do PC.
- Uložené adresy ve Stacku zůstávají, ale při následném volání podprogramu se přepíší novými daty.

Příklad

Jaká data obsahuje ukazatel na zásobník a programový čítač po návratu ze dvou vnořených podprogramů?

Záloha pracovních registrů do zásobníku; instrukce `PUSH`, `POP`

- Kromě zálohy návratové adresy při volání podprogramu, nebo přerušení, lze zásobník využít také jako dočasné odkladiště obsahu pracovních registrů.
- Použití např. při volání podprogramu – záloha potřebných reg.; na konci podprogramu hodnoty znovu načíst ze zásobníku.
- Fáze uložení obsahu 8bitového registru do zásobníku – `PUSH Rr`, kde $r = 0, 1, \ldots, 31$:
  (1) uložení obsahu registru Rr na adresu definovanou SP,
  (2) SP = SP - 1.
- Fáze načtení 8bitové hodnoty ze zásobníku do registru – `POP Rd`, kde $d = 0, 1, \ldots, 31$:
  (1) SP = SP + 1,
  (2) načtení slova z adresy SP do registru Rd.
- Instrukce pro zálohu registrů mají všechny mikrokontroléry (mohou mít jinou syntaxi, či omezený počet využitelných reg.).

Rozdíl mezi podprogramem a makrem

Příklad

Jaké jsou základní rozdíly mezi podprogramem a makrem?

Obsah přednášky

Obsluha přerušení

- Běh programu (hodnota PC) je také ovlivnitelný pomocí tzv. přerušení (anglicky: Interrupt).
- V případě, že mikroprocesor obdrží žádost o obsluhu povoleného přerušení, musí přerušit vykonávanou činnost (hlavní program) a spustit obslužný program, který je určen výhradně pro zdroj přerušení.
- Každý procesor obsahuje sadu možných přerušení interních i externích; zdroje přerušení závisí na hardwarovém vybavení MCU.
- Obsluha každého přerušení je ovládána tzv. vektorem přerušení v programové paměti; jedná se o adresu od které se začne vykonávat konkrétní obsluha.
- Každé přerušení má svůj povolovací bit; navíc existuje jeden globální povolovací bit (u AVR je to bit I ve stavovém registru).
- Jednotlivé přerušení mají různou prioritu; nižší adresa $\Leftrightarrow$ vyšší priorita (viz další slajd).

Vektory přerušení mikrokontroléru ATmega16

Č.	Adresa	Zdroj přerušení	Popis přerušení
1	0x0000	RESET	Externí reset, připojení napájení.
2	0x0002	INT0	Externí požadavek na přerušení 0.
3	0x0004	INT1	Externí požadavek na přerušení 1.
4	0x0006	TIMER2 COMP	Časovač/čítač 2 – shoda komparace.
5	0x0008	TIMER2 OVF	Časovač/čítač 2 – přetečení.
6	0x000A	TIMER1 CAPT	Časovač/čítač 1 – zachycení.
7	0x000C	TIMER1 COMPA	Časovač/čítač 1 – shoda s komparátorem A.
8	0x000E	TIMER1 COMPB	Časovač/čítač 1 – shoda s komparátorem B.
9	0x0010	TIMER1 OVF	Časovač/čítač 1 – přetečení.
10	0x0012	TIMER0 OVF	Časovač/čítač 0 – přetečení.
11	0x0014	SPI, STC	Dokončení sériového přenosu SPI.
12	0x0016	USART, RXC	USART – kompletní příjem dat.
13	0x0018	USART, UDRE	USART – prázdný datový registr.
14	0x001A	USART, TXC	USART – kompletní vyslání dat.
15	0x001C	ADC	ADC – dokončení A/D převodu.
16	0x001E	EE_RDY	EEPROM – komunikace připravena.
17	0x0020	ANA_COMP	Změna výstupu analogového komparátoru.
18	0x0022	TWI	Událost na sériové sběrnici I2C.
19	0x0024	INT2	Externí požadavek na přerušení 2.
20	0x0026	TIMER0 COMP	Časovač/čítač 0 – shoda komparace.
21	0x0028	SPM_RDY	Uložení do programové paměti připraveno.

Vektory přerušení mikrokontroléru ATtiny11

Č.	Adresa	Zdroj přerušení	Popis přerušení
1	0x000	RESET	Externí reset, připojení napájení.
2	0x001	INT0	Externí požadavek na přerušení 0.
3	0x002	I/O piny	Změna stavu I/O pinu.
4	0x003	TIMER0 OVF	Časovač/čítač 0 – přetečení
5	0x004	ANA_COMP	Změna výstupu analogového komparátoru.

- U ATmega16 je mezi jednotlivými vektory přerušení volných 32 bitů (tj. dvě paměťové pozice). U ATtiny11 jen 16 bitů $\Rightarrow$ pro samotnou obsluhu to nestačí.

Pozn.: ATtiny11 neobsahuje datovou paměť, proto není možné definovat softwarový zásobník pro uložení návratových adres. Obsahuje ale hardwarový zásobník, schopný uchování 3 adres (ukazatel na zásobník je 9bitový @ 1kB Flash).

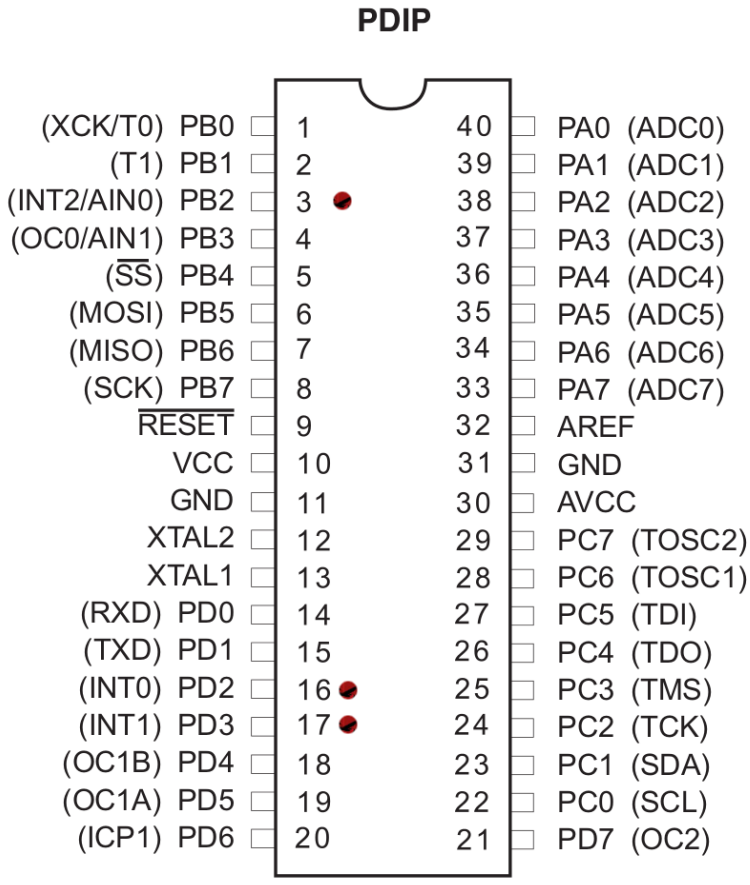
Otázka

Proč vektory přerušení u ATmega obsahují dvě adresové pozice a u ATtiny jen jednu?

Obsluha přerušení

- Jednotlivé fáze obsluhy přerušení:
  1. dokončí se výkon právě vykonávané instrukce,
  2. zakáže se obsluha případných následujících přerušení,
  3. do zásobníku se uloží adresa následující instrukce v programové paměti (zásobník musí být definován),
  4. podle zdroje přerušení se do PC načte vektor přerušení (např.: PC=0x0002 pro INT0 u ATmega16),
  5. vykoná se obsluha přerušení, tj. konkrétní posloupnost instrukcí; do oblasti vektorů přerušení se běžně umísťují pouze skoky na obslužnou funkci,
  6. obsluha přerušení se ukončí instrukcí `RETI` (RETurn Interrupt); analogie s ukončením podprogramu,
  7. povolí se obsluha dalších přerušení,
  8. do PC se načte uložená návratová adresa ze zásobníku,
  9. pokračuje se ve výkonu hlavního programu.

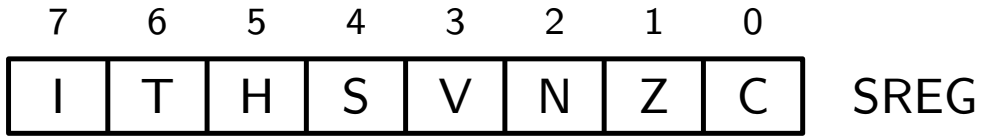
Příklad obsluhy přerušení – externí zdroj přerušení



Obrázek: Zdroje externího přerušení u mikrokontroléru ATmega16.

Příklad obsluhy přerušení – externí zdroj přerušení

- Přerušení se vykoná (obslouží) v případě že:
  (1) je povoleno konkrétní přerušení,
  (2) je povoleno globální přerušení (tj. bit I=1 v registru SREG).



Obrázek: Struktura stavového registru SREG (Status Register).

- Příklad povolení externího přerušení (External Interrupt) u AVR:
  - řídicí registr GICR (General Interrupt Control Register) obsahuje povolovací bity INT1, INT0, INT2 pro externí zdroj přerušení 1, 0 a 2,
  - pro povolení je nutné nastavit příslušný(é) bit(y) na 1,
  - povolit globální přerušení v registru SREG instrukcí `SEI` (SEt Interrupt).
- Konkrétní nastavení (náběžná hrana, sestupná, ...) lze definovat v registru MCUCR (MCU Control Register).
- Požadavek na přerušení je pro CPU signalizován příznakovým bitem v registru GIFR (General Interrupt Flag Register).

Řídicí registry externího zdroje přerušení

7	6	5	4	3	2	1	0	
INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR

Obrázek: Struktura řídicího registru GICR (General Interrupt Control Register).

7	6	5	4	3	2	1	0	
SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR

Obrázek: Struktura řídicího registru MCUCR (MCU Control Register).

7	6	5	4	3	2	1	0	
INTF1	INTF0	INTF2	–	–	–	–	–	GIFR

Obrázek: Struktura řídicího registru GIFR (General Interrupt Flag Register).

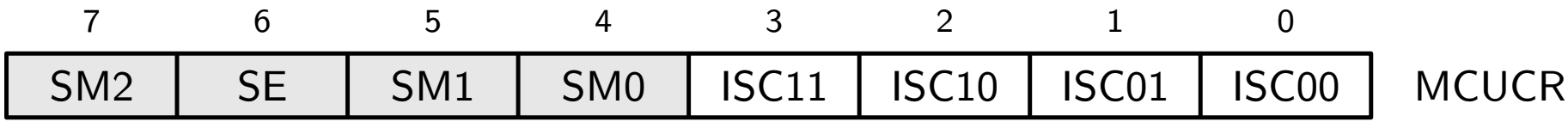
Možnosti nastavení externího přerušení

Tabulka: Způsob spouštění externích požadavků na přerušení INTn prostřednictvím kontrolního registru MCUCR.

ISCn1:0	Popis funkce
0b00	Nízká úroveň (log. 0) na pinu INTn generuje požadavek.
0b01	Jakákoliv změna logické úrovně na pinu INTn generuje požadavek.
0b10	Sestupná hrana na pinu INTn generuje požadavek.
0b11	Vzestupná hrana na pinu INTn generuje požadavek.

Příklad

Jakou hodnotu bude obsahovat registr MCUCR, požadujeme-li generování externího přerušení INT1 při přechodu vstupního signálu z vysoké do nízké úrovně?



Obrázek: Struktura řídicího registru MCUCR (MCU Control Register).

Obsah přednášky

Typy adresování
- Registrové, přímé, nepřímé, bezprostřední

Větvení programu
- Programový čítač
- Volání podprogramu, zásobník

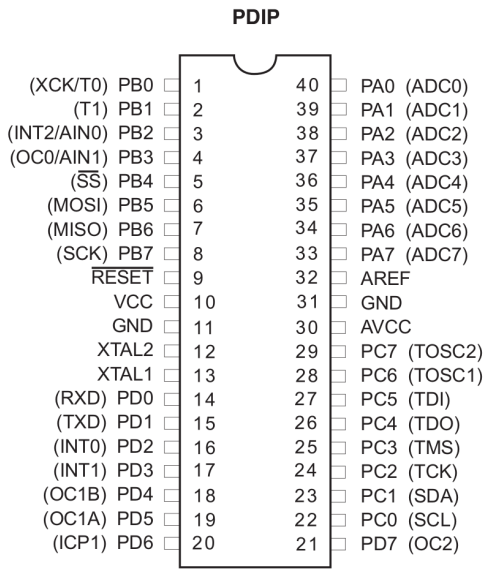
Obsluha přerušení
- Externí požadavek na přerušení

Vnitřní struktura a využití základních periférií
- Vstupně/výstupní port
- Přímé připojení, maticové, pomocí pomocných obvodů
- Časovač/čítač

Ukázka programu v JSA a v jazyce C pro ATmega16
- Obsluha přerušení

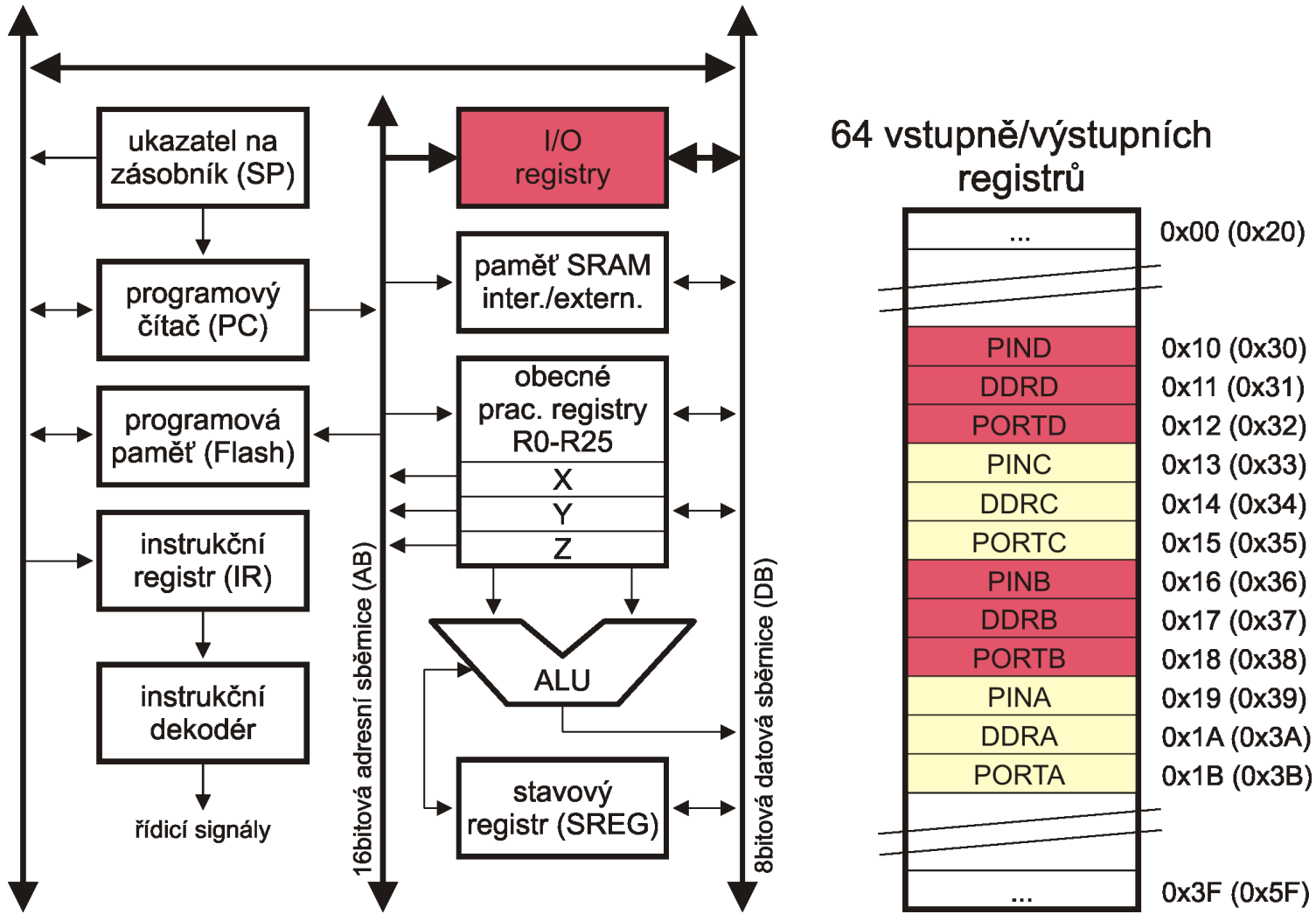
I/O (paralelní) port

- I/O port obsahuje vstupně/výstupní piny, které zajišťují komunikaci s externím "světem".
- Jsou to nejvšestrannější I/O zařízení mikrokontroléru.
- Počty pinů korespondují s použitím mikrokontroléru: 4 I/O piny (RS08KA, Freescale), 6 I/O pinů (ATtiny12, Atmel), 32 I/O pinů (ATmega16, Atmel), 48 I/O pinů (ATmega103, Atmel), ...
- U mikrokontrolérů AVR jsou ke každému obousměrnému I/O portu asociovány 3 I/O (řídicí) registry:
  - směrový registr DDRx (Data Direction Register),
  - datový (výstupní) registr PORTx (Data Register),
  - vstupní piny PINx (Port Input Pins),



Obrázek: Schématická značka mikrokontroléru ATmega16.

Řídicí registry I/O portů



Obrázek: Řídicí registry I/O portů mikrokontrolérů AVR.

Ovládání I/O portu

- Směrový I/O registr DDRxn (x=A,B,C,D; n=0,1,. . .,7):
  - hodnota 0 ↔ pin je definován jako vstupní,
  - hodnota 1 ↔ pin je definován jako výstupní,
  - v každém I/O portu je možné libovolně kombinovat vstupní/výstupní piny. Směr komunikace pinu lze bezprostředně měnit bez ohledu na ostatní.

7	6	5	4	3	2	1	0	
DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDR

Obrázek: Struktura směrového registru DDRC (Port C Data Direction Register).

- Výstupní I/O registr PORTxn :
  - v případě definovaného výstupního pinu, udává hodnota bitu v registru logickou úroveň na pinu. Hodnota 1 (DDRxn=1) ↔ pin je na vysoké úrovni, apod.,
  - v případě vstupního pinu, udává hodnota bitu v registru aktivaci/deaktivaci pull-up rezistoru.

Tabulka: Definice vstupně/výstupního portu.

DDRxn	PORTxn	I/O	Popis funkce
0	0	Vstupní	Vysoká impedance.
0	1	Vstupní	Aktivace pull-up rezistoru.
1	0	Výstupní	Nízká úroveň.
1	1	Výstupní	Vysoká úroveň.

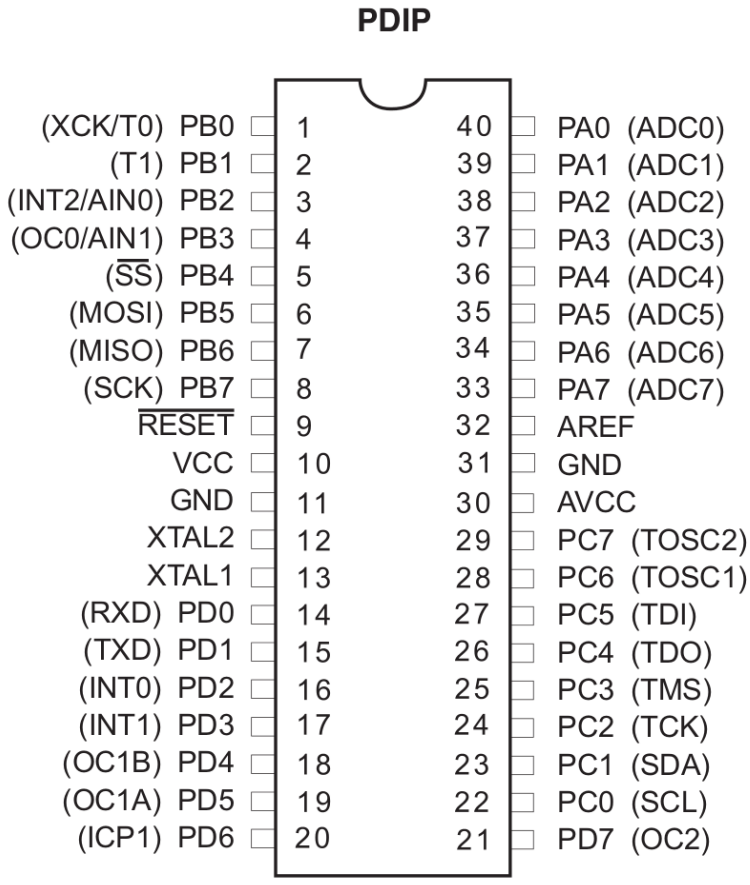
Ovládání I/O portu, pokračování

- Vstupní I/O registr PINxn :
  - hodnota 0 nebo 1 korespondující s úrovní signálu na vstupu,
  - kopírují se zde také data zapsaná do výstupního registru PORTxn !
- Každý pin obsahuje přepěťovou ochranu (omezující diody) a možnost připojení pull-up rezistoru.
- Většina I/O pinů umožňuje kromě klasické I/O komunikace také alternativní funkci. Konkrétní piny jsou tak využívány interními periferiemi MCU.

Tabulka: Některé alternativní funkce I/O pinů.

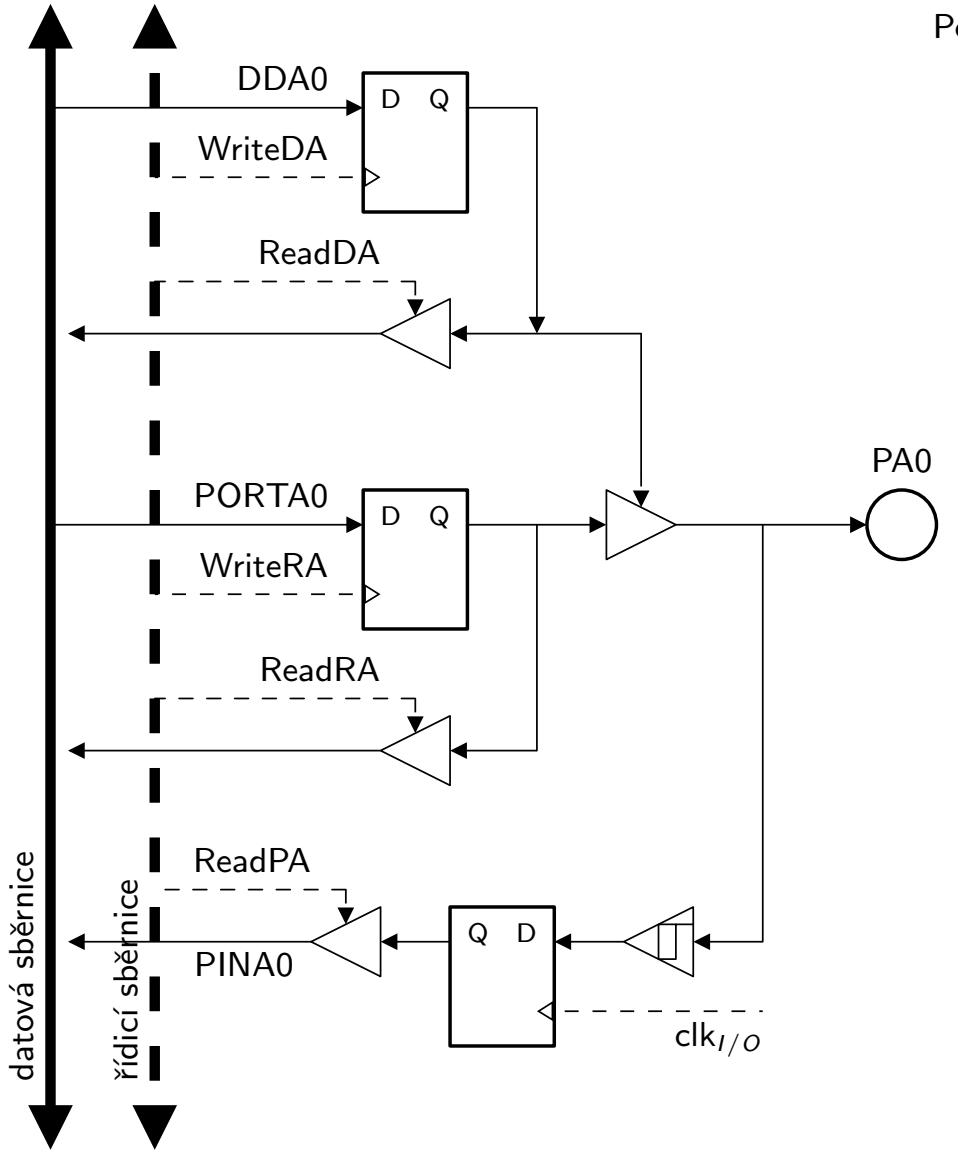
Pin	Alternativa	Popis funkce
PA7:0	ADC	Vstup 10bitového A/D převodníku, tj. 8 kanálů.
PB7:4	SPI	Sériová komunikace pomocí SPI.
PC5:2	JTAG	Programovací/ladicí rozhraní JTAG.
PC1:0	TWI	Dvouvodičová sériová sběrnice I2C.
PD2	INT0	Zdroj externího přerušení 0.
PD1:0	USART	Vstup/výstup sériové komunikace USART.

Alternativní funkce I/O pinů



Obrázek: Alternativní funkce u mikrokontroléru ATmega16.

Zapojení jednoho I/O pinu



Použité řídicí signály:

WriteDA – zápis do registru DDRA
ReadDA – čtení obsahu registru DDRA

WriteRA – zápis do registru PORTA
ReadRA – čtení obsahu registru PORTA

ReadPA – čtení obsahu registru PINA

Obrázek: Funkční schéma jednoho I/O pinu (není zakreslen pull-up, přepěťová ochrana, alternativní funkce, synchronizace vstupu).

Způsoby nastavení jednotlivých I/O pinů v JSA

(1) Pomocí kombinace instrukcí `LDI` (Load immediate) a `OUT` (Out port):

```
1  .include <m16def.inc>      ; definiční soubor mikrokontroléru ATmega16
2      ...
3      LDI   R16, 0b00011000  ; R16 <- 24
4      OUT   DDRB, R16        ; definování výstupních pinů PB4 a PB3
5      ...
6      LDI   R16, (1<<DDB4)|(1<<DDB3)   ; R16 <- 24
7      OUT   DDRB, R16        ; definování výstupních pinů PB4 a PB3
8                             ; operace << realizuje bitový posun doleva
9                             ; operace | realizuje logický součet
10                            ; konst. DDB4 = 4 a DDB3 = 3 definovány v m16def.inc
11                            ; 1<<4 = 16   1<<3 = 8
12     ...
13     OUT   PORTB, R16       ; nastavení výstupních hodnot portu B
```

(2) Pomocí instrukcí `SBI` (Set bit in I/O register) a `CBI` (Clear bit in I/O register):

```
1      ...
2      SBI   DDRB, 0x03       ; definování výstupního pinu PB3
3      SBI   DDRB, 4          ; definování výstupního pinu PB4
4      CBI   DDRB, 3          ; definování vstupního pinu PB3
5      ...
6      IN    R16, PINB        ; načtení hodnot vstupního portu B
```

Způsoby nastavení jednotlivých I/O pinů v jazyce C

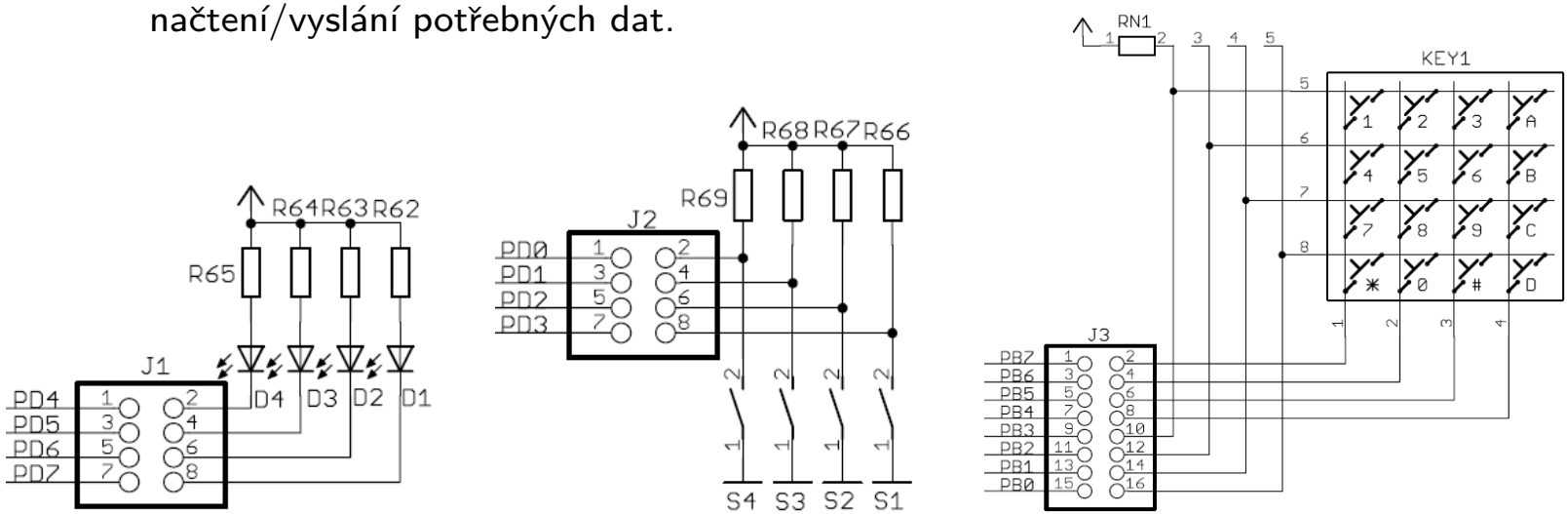
```
1  #include <avr\io.h>       // hlavičkový soubor mikrokontroléru
2      ...
3      // definování výstupních pinů PB4 a PB3
4      DDRB = 0b00011000 ;
5
6      // jiný způsob definování výstupních pinů PB4 a PB3
7      DDRB |= (1<<DDB4) | (1<<DDB3) ;
8      ...
9      // výstupní piny PB7 a PB0 nastaveny na vysokou úroveň
10     PORTB |= (1<<PB7) | (1<<PB0) ;
11
12     // načtení vstupních hodnot z portu B do proměnné temp
13     temp = PINB ;
```

Otázka

Jak v jazyce C zajistit opakované kopírování vstupních dat z portu A na výstupní port D?

Připojení externích periférií

- Připojení externích periférií (tlačítko, LED, klávesnice, ...) je obecně možné realizovat třemi způsoby:
  - **(1)** přímé připojení – jeden I/O pin je využit k připojení jednoho jednoduchého zařízení (tlačítko, LED, ...),
  - **(2)** maticové uspořádání – několik jednoduchých zařízení je uspořádáno do matice, což snižuje počet potřebných I/O pinů (maticová klávesnice),
  - **(3)** pomocí pomocných obvodů/řadičů: externí zařízení pracuje nezávisle na řídicím mikrokontroléru. V případě události generuje požadavek na přerušení, které zajistí načtení/vyslání potřebných dat.



Jednoduché ošetření zákmitů mechanického tlačítka

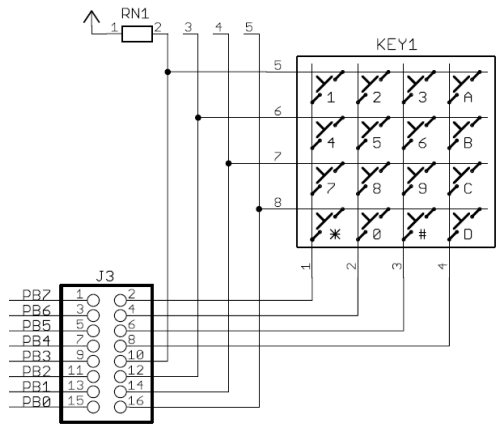
- Při sepnutí mechanického tlačítka dochází k zákmitům (nechtěné impulsy způsobující nekorektní výkon programu). Tyto zákmity je nutné vždy ošetřit!
- Možný způsob ošetření:
  - při stisku i uvolnění tlačítka je volána funkce realizující zpoždění,
  - po stisku se testuje, zda již bylo tlačítko uvolněno, teprve pak se pokračuje ve výkonu programu,
  - v rámci první zpožďovací smyčky (tj. při sepnutí) je možné realizovat "pípnutí reproduktoru", apod. (Pípnutí je reprezentováno vysíláním obdélníkového signálu o určité frekvenci),
  - při využití externího zdroje přerušení není nutné testovat stav tlačítka – před koncem obsluhy vynulovat příslušný příznakový bit (viz reg. GIFR, počítačová cvičení).

Příklad

Nakreslete vývojový diagram jednoduchého ošetření tlačítka.

Použití maticové klávesnice

- Maticová klávesnice 4×4 využívá jeden I/O port:
  - 4 piny jsou definovány jako vstupní (např. PB3–PB0),
  - 4 piny jsou výstupní (např. PB7–PB4).
- Zjištění stisknuté klávesy se provádí tzv. skenováním klávesnice, kdy jsou ve čtyřech krocích vyslány hodnoty na výstupní piny:
  (1) PB7:4 = 0b1110,
  (2) PB7:4 = 0b1101,
  (3) PB7:4 = 0b1011,
  (4) PB7:4 = 0b0111.
- Po každém kroku se skenují vstupní piny. Pokud je některý roven 0, bylo stisknuto příslušné tlačítko.



Obrázek: Připojení maticové klávesnice k I/O portu mikrokontroléru.

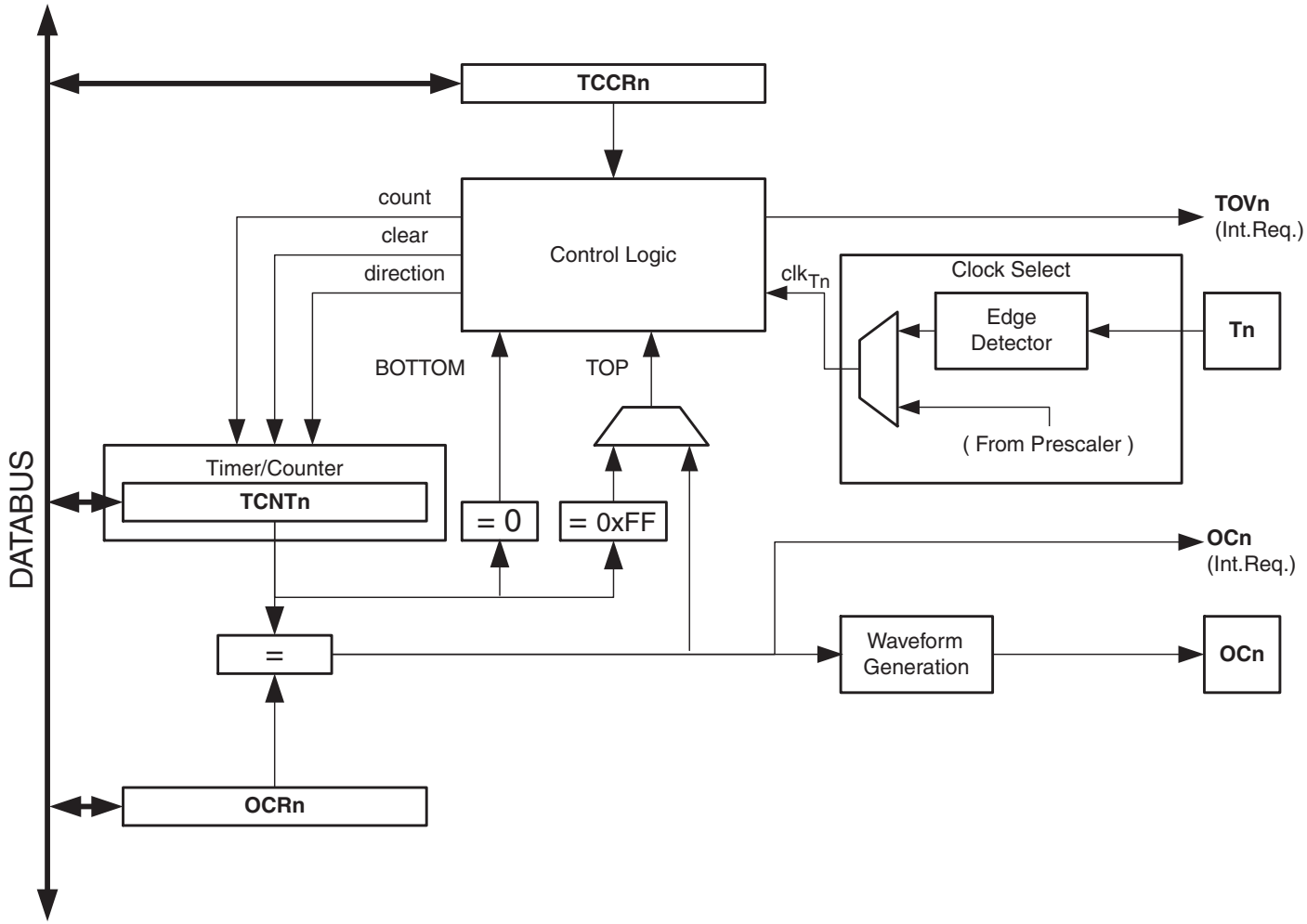
Příklad

Nakreslete vývojový diagram skenování maticové klávesnice.

Funkce interního časovače/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,
  - ...
- Obě funkce využívají registr(y), jehož obsah se inkrementuje pomocí hodinového, příp. externího signálu.
- Lze tak využít 8, 16 bitové časovače/čítače.

Blokové schéma 8bitového časovače/čítače 0



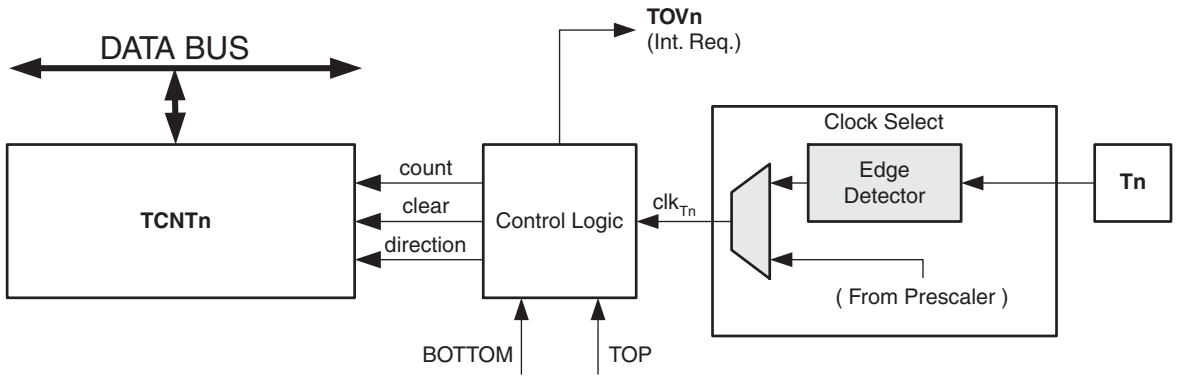
Obrázek: Blokové schéma časovače/čítače 0 (označení "n" udává číslo čítače)

Blokové schéma 8bitového časovače/čítače 0

- Č/č 0 je 8bitový registr schopen inkrementovat (dekrementovat) svou hodnotu. K č/č 0 lze přistupovat pomocí registrů:
  - TCNT0 (Timer/Counter0) – datová hodnota časovače/čítače 0,
  - OCR0 (Output Compare Register) – hodnota pro neustálé porovnávání s datovým registrem,
  - TCCR0 (Timer/Counter Control Register) – řídicí registr. Např. nastavení hodnoty předděličky, povolení přerušení, atd.
- Možnost generování přetečení:
  - při přetečení č/č (dosažení maximální hodnoty a znovu návrat k hodnotě 0),
  - při rovnosti hodnoty datového registru a komparačního.

Časovač/čítač 0, funkce časování (čítání)

- Vnitřní signály:
  - count: inkrementace nebo dekrementace obsahu registru TCNT0,
  - direction: výběr mezi inkrementací a dekrementací,
  - clear: nulování obsahu datového reg. TCNT0,
  - clk_{T0} - hodinový signál č/č 0,
  - BOTTOM: signalizace dosažení minimální hodnoty 0x00,
  - TOP: signalizace dosažení nejvyšší hodnoty čítání 0xFF nebo hodnoty v komparačním registru.



Obrázek: Funkce čítání čítače/časovače.

Časovač/čítač 0, funkce časování (čítání)

- Č/č 0 může být řízen vnitřním zdrojem hodinového signálu (lze zpomalit pomocí předděličky), nebo externím signálem přes pin T0.

Tabulka: Nastavení předděličky 8bitového časovače/čítače 0 v kontrolním registru TCCR0

CS02:0	Popis funkce
0b000	Hodinový signál odpojen, tj. časovač/čítač 0 nepracuje.
0b001	Bez předděličky – f_{CPU}.
0b010	Předdělička 8 – $f_{CPU}/8$.
0b011	Předdělička 64 – $f_{CPU}/64$.
0b100	Předdělička 256 – $f_{CPU}/256$.
0b101	Předdělička 1 024 – $f_{CPU}/1\,024$.
0b110	Externí zdroj řídicího signálu, reakce na sestupnou hranu.
0b111	Externí zdroj řídicího signálu, reakce na náběžnou hranu.

Otázka

Co je to přetečení (anglicky: overflow)?

Příklad

Za jak dlouho dojde k přetečení 8bitového č/č 0, je-li CS02:CS00=010 a $f_{clk} = 1\,MHz$?

Časovač/čítač 0, funkce časování (čítání)

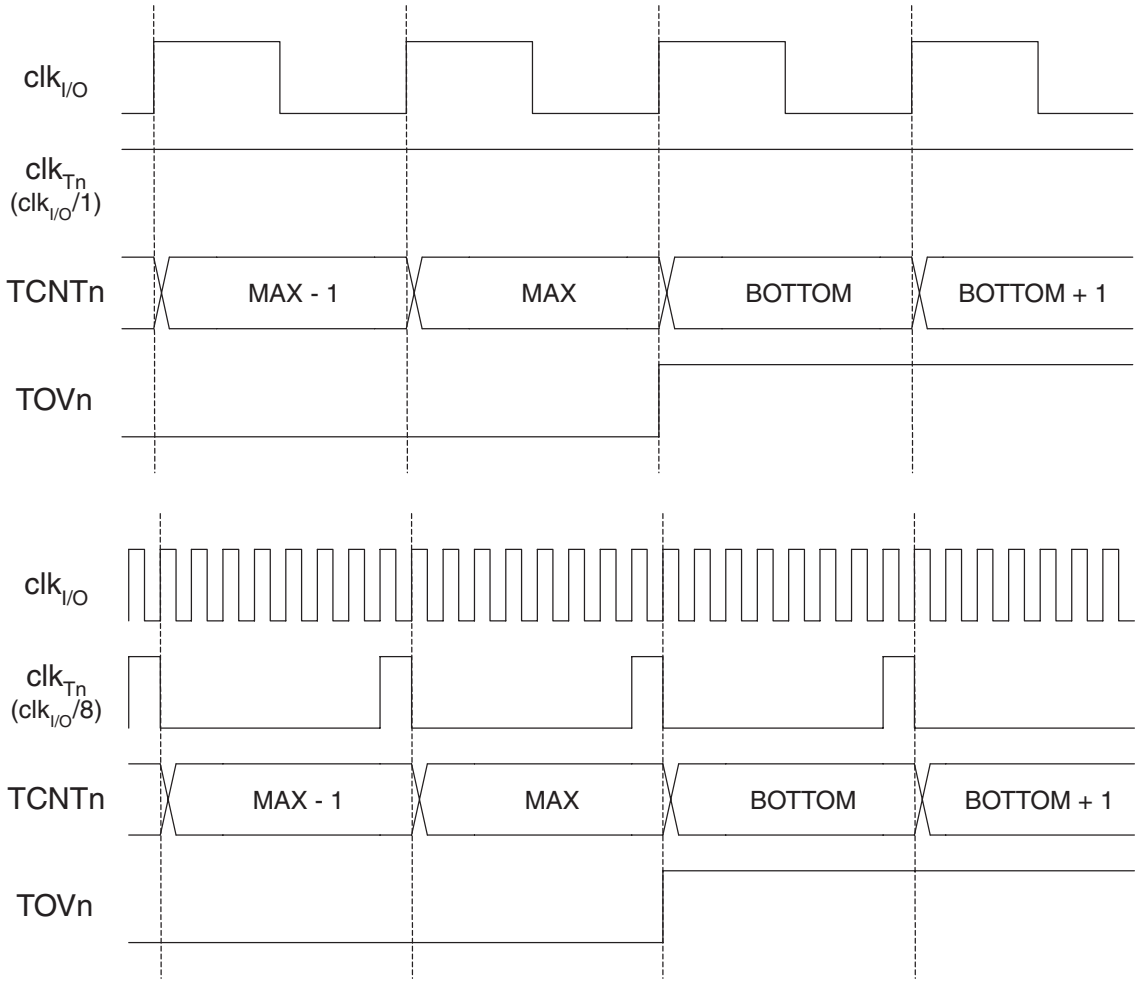
- Doba přetečení "n" bitového časovače t_{OVF}, kde f_{CPU} je frekvence hodinového signálu jádra mikrokontroléru a N je hodnota předděličky:

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

- Doba přetečení "n" bitového časovače s nenulovou počáteční hodnotou *init*:

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

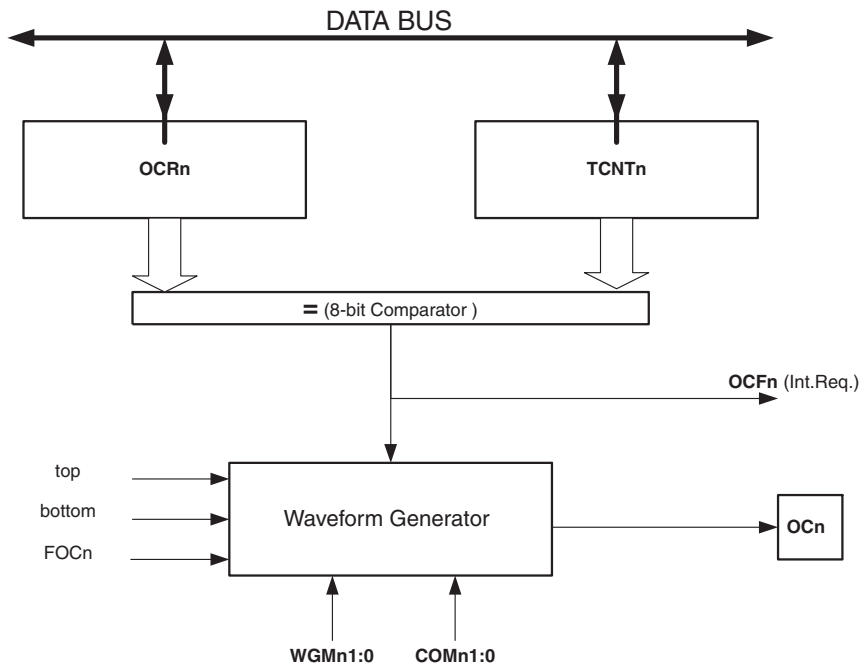
Význam předděličky hodinového signálu



Obrázek: Rozdíl mezi rychlostí čítání bez předděličky a s předděličnou $N = 8$

Časovač/čítač 0, funkce porovnávání (komparace)

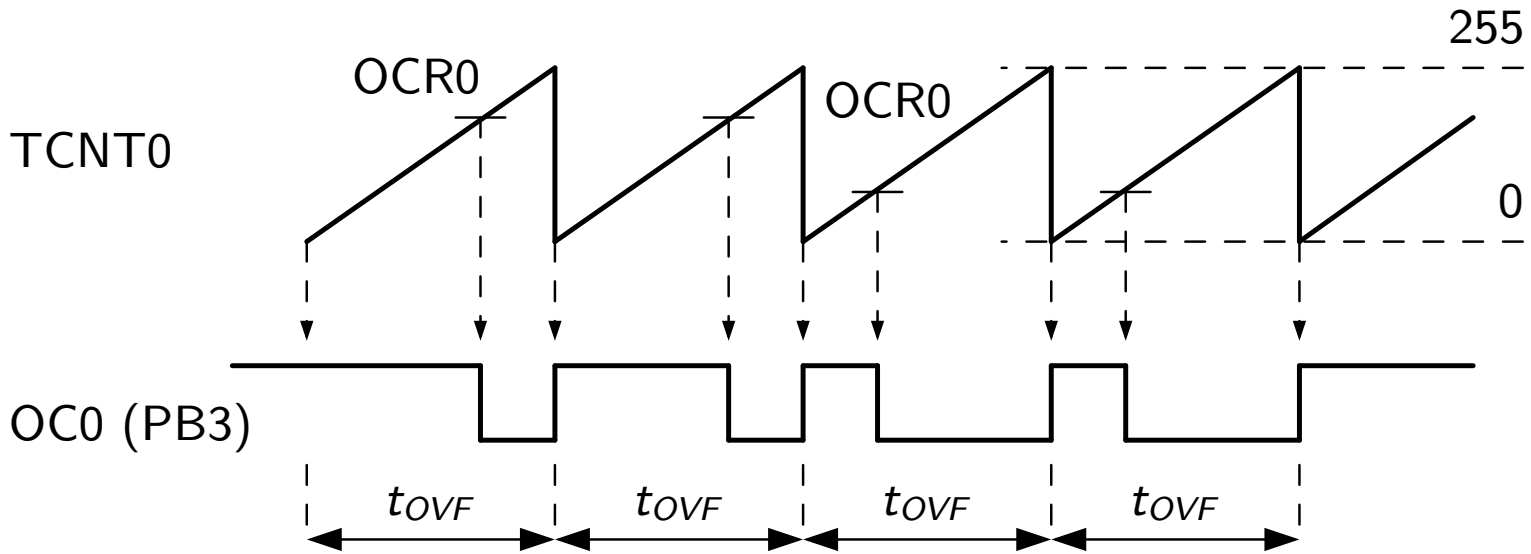
- Obsah registru OCR0 je při každé změně datového registru porovnáván právě s TCNT0.
- Jestliže se obsahy obou registrů rovnají, lze generovat přerušení (pokud je řádně povoleno).
- Komparace lze snadno využít ke generování signálů, např. pulsně šířkovou modulaci (PWM – Pulse Width Modulation).



Obrázek: Funkce porovnávání časovače/čítače.

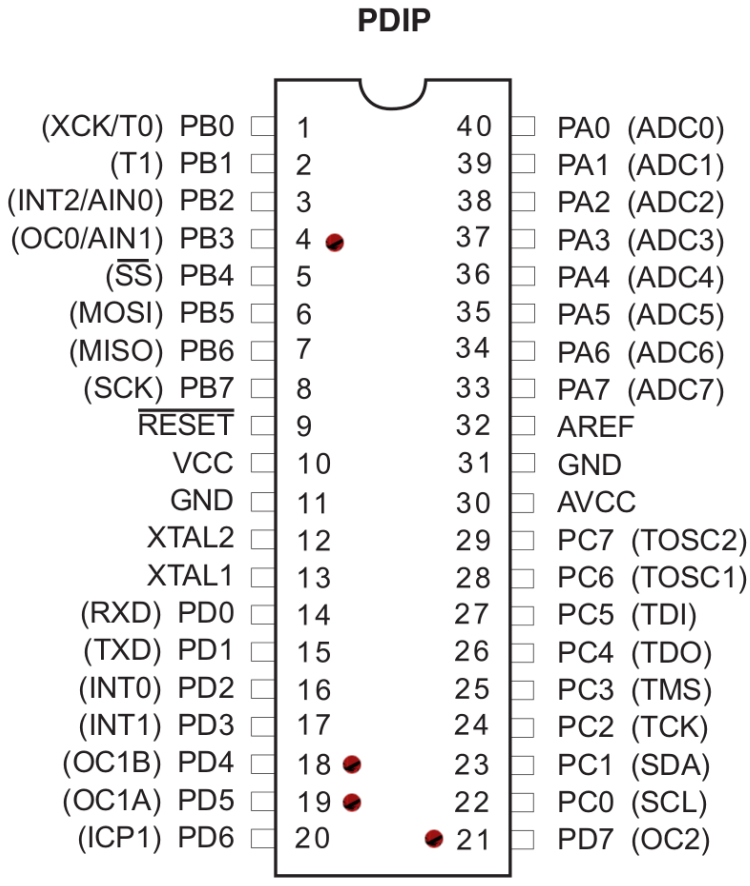
Časovač/čítač 0, funkce generováni PWM

- Informace je PWM signálem přenášena proměnnou střídou obdélníkového signálu.
- Princip činnosti:
  - přetečení časovače: vysoká úroveň generovaného signálu (začátek periody),
  - při rovnosti datového a komparačního registru: nízká úroveň (lze spouštět přerušení).



Obrázek: Princip generování PWM signálu pomocí změny hodnoty komparačního registru 8bitového časovače.

Výstupy generovaného signálu



Obrázek: Generování PWM signálu u mikrokontroléru ATmega16 – vliv předděličky.

Obsah přednášky

Obsluha přerušení v JSA

- Při použití přerušení (i podprogramů) je nutné definovat zásobník, tj. naplnit registrový pár SPH:SPL. Jedná se o I/O registry (viz přednáška o Instrukční sadě), proto se použije kombinace instrukcí `LDI` a `OUT`.
- Běžná praxe u číslicových systémů: hlavní část programu se zacyklí pomocí nekonečné smyčky. Z ní se skáče na obslužné programu pouze při příchodu požadavku na přerušení.
- Vektory přerušení použité v ukázce – platí pro ATmega16:
  - 0x0000 – reset,
  - 0x0002 – externí požadavek na přerušení INT0,
  - 0x001C – byl dokončen A/D převod.

Obsluha přerušení v JSA

```
1   .include <m16def.inc>          ; def. soubor mikrokontroléru ATmega16
2   .cseg                          ; paměťový segment Flash
3   .org 0x0000                    ; ulož od adresy 0x0000
4       RJMP reset                 ; skoč na návěští reset
5   .org 0x0002                    ; vektor přerušení INT0
6       RJMP tlacitko              ; skoč na návěští tlacitko
7   .org 0x001C                    ; vektor přerušení A/D převodníku
8       RJMP ad_prevod             ; skoč na návěští ad_prevod
9
10  .org 0x002A                    ; ulož od adresy 0x002A
11  reset:
12      LDI  R16, high(RAMEND)     ; definice zásobníku, tj. naplnění registrů SPH:L
13      ...                        ; dokončit definování zásobníku
14      ...                        ; naplnění I/O registrů pro INT a A/D
15      SEI                        ; globální povolení všech přerušení
16
17  loop:                          ; nekonečná smyčka
18      RJMP loop                  ; skok na návěští loop
19
20  tlacitko:
21      ...                        ; obsluha přerušení INT0
22      RETI                       ; ukončení obsluhy přerušení
23
24  ad_prevod:
25      ...                        ; obsluha přerušení A/D převodníku
26      RETI                       ; ukončení obsluhy přerušení
```

Obsluha přerušení v jazyce C

- 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:
  - INT0_vect – externí přerušení,
  - ADC_vect – přerušení od A/D převodníku,
  - ...
- (Všechny informace a ukázky využívají překladače GCC s knihovnou avr-libc.)

Parametry makra obsluhy přerušení `ISR()` pro ATmega16

Č.	Adresa	Parametr v C (GCC)	Popis přerušení
1	0x0000		Externí reset, připojení napájení.
2	0x0002	INT0_vect	Externí požadavek na přerušení 0.
3	0x0004	INT1_vect	Externí požadavek na přerušení 1.
4	0x0006	TIMER2_COMP_vect	Časovač/čítač 2 – shoda komparace.
5	0x0008	TIMER2_OVF_vect	Časovač/čítač 2 – přetečení.
6	0x000A	TIMER1_CAPT_vect	Časovač/čítač 1 – zachycení.
7	0x000C	TIMER1_COMPA_vect	Časovač/čítač 1 – shoda s komparátorem A.
8	0x000E	TIMER1_COMPB_vect	Časovač/čítač 1 – shoda s komparátorem B.
9	0x0010	TIMER1_OVF_vect	Časovač/čítač 1 – přetečení.
10	0x0012	TIMER0_OVF_vect	Časovač/čítač 0 – přetečení.
11	0x0014	SPI_STC_vect	Dokončení sériového přenosu SPI.
12	0x0016	USART_RXC_vect	USART – kompletní příjem dat.
13	0x0018	USART_UDRE_vect	USART – prázdný datový registr.
14	0x001A	USART_TXC_vect	USART – kompletní vyslání dat.
15	0x001C	ADC_vect	ADC – dokončení A/D převodu.
16	0x001E	EE_RDY_vect	EEPROM – komunikace připravena.
17	0x0020	ANA_COMP_vect	Změna výstupu analogového komparátoru.
18	0x0022	TWI_vect	Událost na sériové sběrnici I2C.
19	0x0024	INT2_vect	Externí požadavek na přerušení 2.
20	0x0026	TIMER0_COMP_vect	Časovač/čítač 0 – shoda komparace.
21	0x0028	SPM_RDY_vect	Uložení do programové paměti připraveno.

Obsluha přerušení v jazyce C

```
1  #include <avr\io.h>          // hlavičkový soubor pro MCU
2  #include <avr\interrupt.h>   // hlavičkový soubor pro přerušení
3  ...                          // případné další knihovny
4
5  ISR( INT0_vect ){            // obsluha externího přerušení INT0
6      ...                      // kód obsluhy přerušení
7  }
8
9  ISR( ADC_vect ){             // obsluha přerušení A/D převodníku
10     ...                      // kód obsluhy přerušení
11 }
12
13 int main( void ){
14     ...                      // kód hlavní funkce
15     sei() ;                  // globální povolení všech přerušení
16     while( 1 ) ;             // nekonečná smyčka
17     return( 1 ) ;            // hlavní funkce vrací hodnotu 1
18 }
```