Programovací techniky pro práci v reálném čase

- reálný čas = systém musí poskytnout odezvu na „událost“ v dostatečně krátkém čase (menším než je „perioda výskytu událostí“)
- Jak vhodně uspořádat kód programu při obsluze více „typů událostí“ (např. na jeden uP připojeno: tlačítka, sér. kanál, A/D převodník, LCD displej)?
  - polling
  - přerušení (+polling)
  - Operační systém reálného času – RTOS (Real Time Operating System) – příští semestr

Polling

- periodické dotazování na stav „události“

```
while (1) {
      if (Event_1 == 1) {
            // Obsluha udalosti 1
      }
      if (Event_2 == 1) {
            // Obsluha udalosti 2
      }
      if (Event_3 == 1) {
            // Obsluha udalosti 3
      }
}
```

- výhody: jednoduché
- nevýhody:
  - pomalé reakce (v případě dlouhého kódu např. u události 1 se nemusí stihnout reagovat na událost 3)
- čekání na „událost“

```
// kod před udalosti
while (Event_1 == 0);
// Obsluha udalosti 1
```

Příznaky

- speciální technika pro ulehčení práce – některé úlohy jinak neřešitelné
- příznak (flag) = proměnná (bit):
  - 1: nastala událost
  - 0: událost nenastala
- polling s příznaky
  - 2 události (Events), každá má svůj příznak `fEvent`

```
fEvent_1 = 0;
fEvent_2 = 0;
while (1) {
      // Nastaveni priznaku
      if (Event_1 == 1) fEvent_1 = 1;
      if (Event_2 == 1) fEvent_2 = 1;
      // Obsluha udalosti
      if (fEvent_1 == 1) {
            // obsluha udalosti 1
```

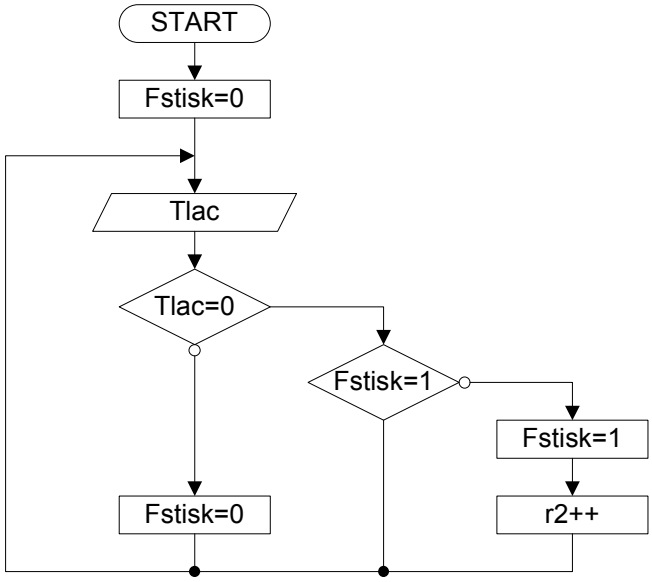
```
                fEvent_1 = 0;     // udalost obslouzena
            }
            if (fEvent_2 == 1) {
                // obsluha udalosti 2
                fEvent_2 = 0;     // udalost obslouzena
            }
        }
```

(možné i jiné varianty)

- Výhody: detekce události není závislá na délce kódu
- Nevýhoda – při déletrvajícím výskytu události může dojít k její vícenásobné obsluze (doba stisku tlačítka vs. rychlost CPU)
- řešení – další příznak `fDone` = informace o dokončené obsluze události

```
fDone_1 = 0;
fDone_2 = 0;
fEvent_1 = 0;
fEvent_2 = 0;
while (1) {
    // detekce udalosti 1
    if (Event_1 == 1) fEvent_1 = 1;
    else {
        fEvent_1 = 0;
        fDone_1 = 0;
    }
    // detekce udalosti 2
    if (Event_2 == 1) fEvent_2 = 1;
    else {
        fEvent_2 = 0;
        fDone_2 = 0;
    }
    // Obsluha udalosti 1
    if (fEvent_1 == 1) {
        // uz byla obslouzena?
        if (fDone_1 == 0) {
            // ne, obslouzim ji
            // KOD: obsluha udalosti 1
            fDone_1 = 1; // udalost byla obslouzena
        }
    }
    // Obsluha udalosti 2
    if (fEvent_2 == 1) {
        // uz byla obslouzena?
        if (fDone_2 == 0) {
            // ne, obslouzim ji
            // KOD: obsluha udalosti 2
            fDone_2 = 1; // udalost byla obslouzena
        }
    }
}
```

- příklad: inkrementace r2 na stisk tlačítka (reaguje na stisk, nikoli na držení tl.)




```
// tlacitko pripojene přes ext. Pull-up na PB3 (stisk = 0), pri
stisku inc r2
.include "m128def.inc"

.def  fStisk      = r16;      // bude v nem nula nebo 0xFF
.def  incReg      = r2;
.cseg
      .org  0
      rjmp  init
init: clr   fStisk
      clr   incReg
start:
      sbis  PINB, PB3   // tl. neni stisknute, preskocim...
      rjmp  tlac        // odskok na obsluhu
      clr   fStisk
      rjmp  tlacEnd
tlac: tst   fStisk
      breq  obsluz      // fStisk = 0 -> stisk jeste nebyl obslouzen
                        // (skocim na obsluhu)
      rjmp  tlacEnd
obsluz:
      inc   incReg            // incReg++
      ser   fStisk            // fStisk = 0xFF
tlacEnd:// dalsi akce (napr. dalsi tlacitka)
      rjmp  start
```

Přerušení

- opět několik možností řešení

```
// hlavni program
void main (void)
{
      while (1) {
      }
}
// ISR pro preruseni od Udalosti 1
void INT_Event_1(void)
{
      // Obsluha udalosti 1
      reti
}
```

```
// ISR pro preruseni od Udalosti 1
void INT_Event_2(void)
{
      // Obsluha udalosti 1
      reti
}
```

- výhoda: jednouché
- nevýhoda: omezená priorita (dána přerušením), dlouhý kód ISR blokuje ostatní přerušení

Přerušení + Polling

- pro většinu jednodušších úloh nejlepší varianta

```
// hlavni program
void main (void)
{
      while (1) {
            if (fEvent_1 == 1) {
                  // Obsluha udalosti 1
                  fEvent_1= 0;
            }
            if (fEvent_2 == 1) {
                  // Obsluha udalosti 2
                  fEvent_2 = 0;
            }
      }
}
// ISR pro preruseni od Udalosti 1
void INT_Event_1(void)
{
      fEvent_1 = 1;
      reti
}
// ISR pro preruseni od Udalosti 2
void INT_Event_2(void)
{
      fEvent_2 = 1;
      reti
}
```

- lze libovolně kombinovat s předcházejícími technikami
- Události s vysokou prioritou – kód přímo do ISR

Sériový přenos dat

- rozdíly oproti paralelnímu přenosu:
  - počet vodičů
  - rychlost
- simplexní, duplexní, poloduplexní přenos
- synchronní vs. asynchronní

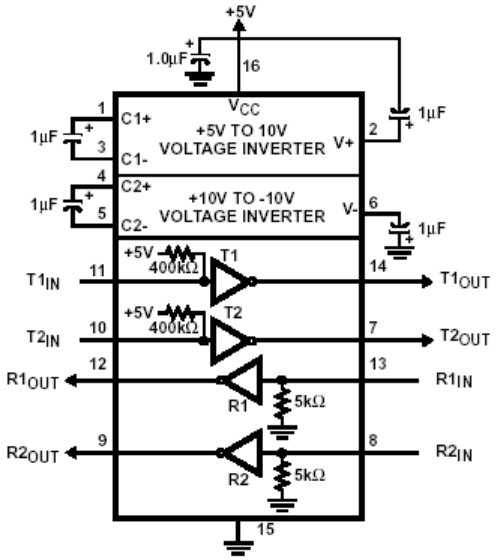
RS232

- standard ANSI
- 3 vodiče v základní konfiguraci – RXD, TXD, GND
  - další vodiče – řízení toku dat (handshaking)

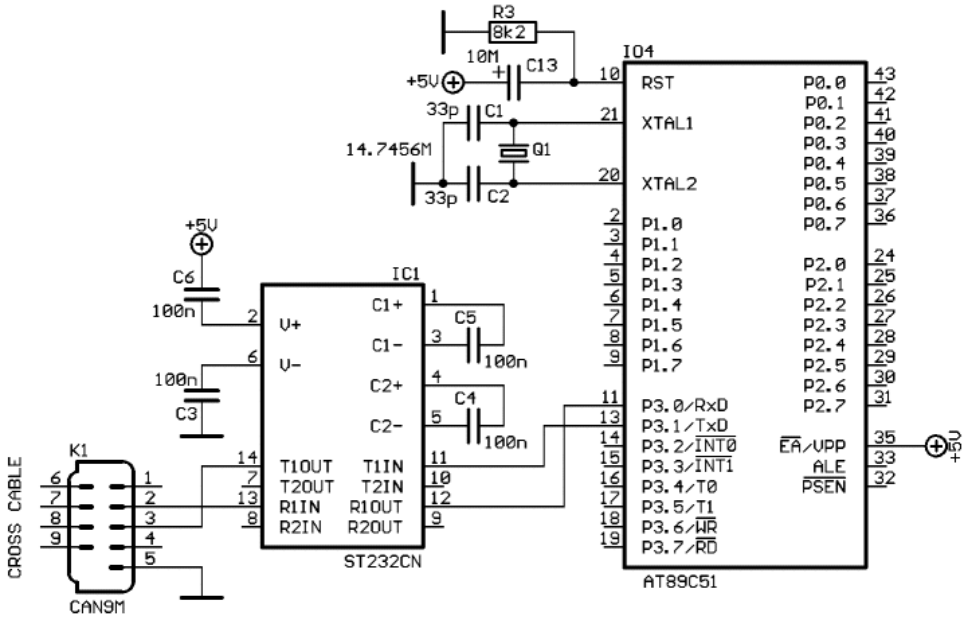
  - RTS – Request To Send (žádost o vysílání)
  - CTS – Clear To Send (připraven k vysílání)
  - DSR – Data Set Ready (připraven k příjmu)
- max. délka vedení cca 15m
- reprezentace logických úrovní
  - negativní logika
  - Log1 → -3 až -15V, Log0 → +3 až +15V (typicky +/- 12V)
  - při komunikaci s MCU (log úrovně TTL) → nutné přizpůsobení úrovní
  - nejlépe pomocí MAX232

MCU a MAX 232

- hlavní problém konverze → nutnost vysokých a záporných napětí → řešení = nábojová pumpa
- vnitřní zapojení obvodu



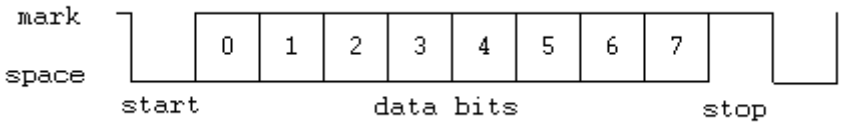
- připojení MAX232 k MCU



  - C3-C6 → obvykle 1M
- dva typy propojení:
  - DTE-DTE (DCE-DCE) → křížený kabel (konektory CAN 9M a 9F, propojeny piny 2-3,3-2…)
  - DTE-DCE → nekřížený („1:1“) kabel (2-2,3-3…)

UART

- RS232 = specifikace fyzické vrstvy (přenosového média)
- UART = specifikace formátu přenášených dat
- UART na RS232 = sériový port PC (Canon 9M)
- základní formát dat (v kladné logice!!!)
  - asynchronní přenos



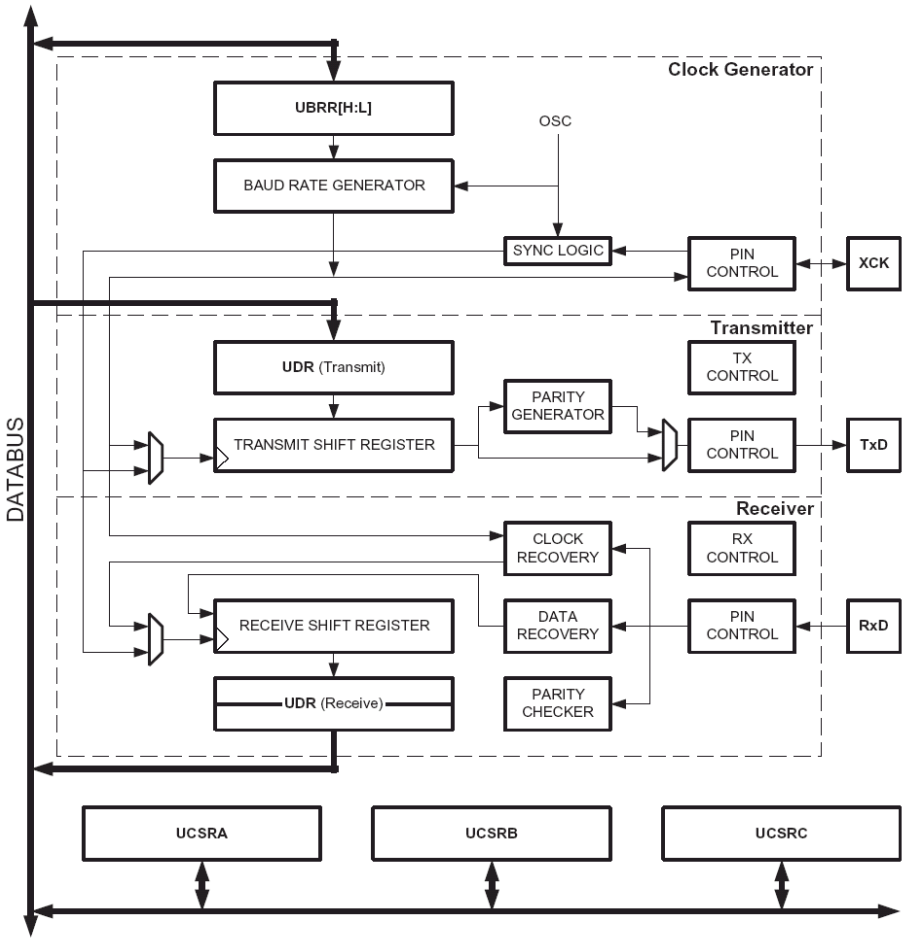
  - klid na lince = Log1
  - start bit (log0) – zahajuje přenos dat
  - 5-9 datových bitů
  - (paritní bit) – v obr. není
  - stop bit (log1)
    - ukončuje komunikaci (odděluje dva datové rámce)
    - délka 1;1,5;2 bity
- přenosová rychlost (Bd) = $1/T_b$

Sériový kanál AVR

- Je obsažen téměř ve všech typech AVR.
- Vlastnosti:
  - Plně duplexní,
  - Asynchronní nebo synchronní komunikace,
  - 5 až 9 datových bitů, 1 nebo 2 stop bity, parita žádná lichá nebo sudá
  - Detekce falešného start bitu, chybného znaku a přetečení přijímacího bufferu
  - Vlastní generátor přenosové rychlosti.
  - 3 samostatné vektory přerušení:
    - - vysílání dokončeno
    - - vysílací vyrovnávací registr prázdný
    - - příjem dokončen
- Registry
  - Řídící:

- Nastavení + detekce stavů UCSRA, UCSRB, UCSRC
- Nastavení rychlosti přenosu: UBRR(H:L)

o Datový registr: UDR
- registr pro vysílání i příjem, skutečnost = 2 fyzicky oddělená místa
- zápis do UDR = odvysílání dat
- AVR přijme data → umístí je do UDR → přečtením je vyzvedneme



- Piny :TxD (vysílaní), RxD (příjem) a XCK (hodiny – jen v případě synchronního přenosu)
- Vysílač i přijímač → 1B buffer (vždy UDR + posuvného vysílacího/přijímací registr) → lze pracovat bez přestávky mezi rx/tx rámců

Nastavení přenosové rychlosti

- Přes UBR
- Baud Rate Generátor → obsahuje čítač, dekrementován každou T_{clk} z hodnoty v UBRR → přednastavení UBRR = rychlost rx/tx v Bd (bit^{-1})

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps).

BAUD Baud rate (in bits per second, bps)

f_{OSC} System Oscillator clock frequency

UBRR Contents of the UBRRH and UBRRL Registers, (0 - 4095)

- Bit U2X zdvojnásobuje rychlost přenosu

- nelze použít libovolné f_{clk} (nelze odvodit normované Bd) → tabulka UBRR vs. f_{clk} viz. datasheet, doporučená max. chyba v Baud Rate by měla být < 1,5%
  - obvyklé jsou „nepěkné“ f_{clk}, např. 1,8432MHz 11,0592 MHz 14,7456 MHz
  - Příklady nastavení:

Baud Rate (bps)	f_{osc} = 1.0000 MHz				f_{osc} = 1.8432 MHz				f_{osc} = 2.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%

Nastavení vlastností

- Registr UCSRC

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

  - **Důležitá poznámka** – Registr UCSRC sdílí stejné paměťové místo jako UBBRH. Kam se bude **zapisovat** → řídí bit URSEL:
    - URSEL = 1 → do UCSRC
    - URSEL = 0 → do UBRRH.
    - Postup při **čtení** registrů (málokdy) → viz. datasheet
  - UMSEL

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

- UPM1:0 → parita

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- USBS → stop bit

USBS	Stop Bit(s)
0	1-bit
1	2-bit

- UCSZ1:0 + UCSZ2 (z registru UCSRB viz. níže) → počet datových bitů

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

- Registr UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

  - Přijat znak → RXC (RX complete) nastaven do 1 (může vyvolat přerušení → RXCIE)
  - Vysílací registr prázdný → UDRE (UDR empty) do 1 (může vyvolat přerušení → UDRIE)
    - 0 = UDR obsahuje dosud neodvysílaný byte
    - 1 = UDR prázdný (**trvale!!!**)
  - Vysílání ukončeno → TXC do 1 (může vyvolat přerušení → TXCIE)
  - FE, DOR, PE → nastaveny do 1 při chybách příjmu (viz. datasheet)
- Registr UCSRB

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- UCSZ2 → viz. tabulka nastavení počtu datových bitů výše
- RXEN (RX enable)→ povolení přijímače
- TXEN → povolení vysílače
- RXCIE (RX interrupt enable) → povolení přerušení od přijímače („dokončen příjem znaku“)
- TXCIE → povolení přerušení od vysílače („znak odvysílán“)
- UDRIE → povolení přerušení od vysílače („UDR prázdný“)

- Příklad nastavení (ATmega32, 18.432 MHz, 2400Bd, 8b, no parita, 1 stop bit, bez přerušení)
  - ASM:

```
// 2400 B na 18.432MHz
clr   r16
out   UCSRC, r16        // URSEL je do 0
ldi   r16, LOW(479)
ldi   r17, HIGH(479)
out   UBRRL, r16
out   UBRRH, r17
// URSEL = 1, TX a RX on
ldi   r16, (1<<RXEN) | (1<<TXEN)
out   UCSRB
ldi   r16, (1<<URSEL) | (1<<UCSZ1) | (1<<UCSZ0)
// 8b (UCSZ2 je implicitne v 0), noparity, 1sb
out   UCSRC, r16
```

Vysílání

- Musí být povoleno (TXEN = 1); zahájeno zápisem UDR
- Vysílání posloupnosti bytů → nutno čekat na prázdný UDR:
  - Polling
  - Přerušení
- Ad polling:
  - ASM:

```
      out   UDR, r0           // 1.Byte (data v r0)
waitTX:
      sbis  UCSRA, UDRE       // dokud nenastaven UDRE cekej
      rjmp  waitTX
      out   UDR, r0           // 2.Byte (v r0)
```

  - Lze čekat i na TXC (neobvyklé, např. při komunikaci po RS485)
- Ad přerušení
  - Příznaky TXC a UDRE mohou generovat přerušení (povolení TXCIE a UDRIE v UCSRB).
  - UDR se nemaže automaticky po vstupu do přerušovací rutiny, ale až po zápisu nové hodnoty do UDR. Žádný zápis → přerušení vyvoláváno dokud:
    - Přerušení zakázáno
    - Nový zápis do UDR
    - obvykle → v handleru od UDRE zákaz UDRIE

- TXC se maže automaticky po vstupu do handlenu
- Příklad viz níže:

Příjem

- Musí být nejprve povolen (RXEN = 1).
- Princip:
  - celý rámec přijat do přijímacího posuvného registru (tzn. je detekován stop bit)
  - jeho přesun obsahu do UDR.
  - Nastavení příznaku kompletního příjmu RXC (popř. přerušení).
- Opět → polling na RXC do 1 (nevhodné – kdy bude příjem?) nebo přerušení.
- Ad přerušení
  - v přijímacím bufferu nepřečtená data → nastaven RXC.
  - RXC nulován automaticky přečtením UDR. Data nepřečtena = neustálé generování přerušení
- Příklad: příjem + vyslání 1B

```
.include "m32def.inc"

        .cseg
        .org  0
        rjmp  init
        .org  URXCaddr
        rjmp  RX_int
        .org  UDREaddr
        rjmp  TX_int

init:   // 38400 Bd na 16 MHz
        ldi   r16, 0
        ldi   r17, 25
        out   UBRRL, r16
        out   UBRRH, r17
        ldi   r16, (1<<RXCIE) | (1<<RXEN) | (1<<TXEN)   // ne UDRIE!!!
        out   UCSRB, r16
        ldi   r16, (1<<URSEL) | (1<<UCSZ1) | (1<<UCSZ0)
        out   UCSRC, r16
        sei
        // zapis dat = povoleni preruseni od UDR
        // byte pro odvysilani predpokladam v r0
        in    r16, UCSRB
        ori   r16, (1<<UDRIE)   // UDRIE do 1
        out   UCSRB, r16
end:    rjmp  end

RX_int:
        in    r1, UDR           // prijaty byte ulozen do r1
        reti

TX_int:
        push  r16
        out   UDR, r0
        // zakaz preruseni od UDR
        in    r16, UCSRB
        andi  r16, ~(1<<UDRIE)  // UDRIE do 0
        out   UCSRB, r16
```

```
        pop   r16
        reti
```

- Odvysílání pole: zákaz UDRIE až po posledním prvku pole

```
.include "m32def.inc"

        .dseg
data: .byte 10
pocet:.byte 1

        .org  0
        rjmp  init
        .org  URXCaddr
        rjmp  RX_int
        .org  UDREaddr
        rjmp  TX_int

init: // nastaveni stejne
        // odvysilani pole „data“, pocet prvku v „pocet“
        lds   r3, pocet   // pocitadlo odvys.bytu v handleru TX_int
        ldi   XL, LOW(data)
        ldi   XH, HIGH(data)
        sei
              // povolim UDRIE = zahajim vysilani
        in    r16, UCSRB
        ori   r16, (1<<UDRIE)   // UDRIE do 1
        out   UCSRB, r16
end:    rjmp  end

RX_int:
        in    r1, UDR           // prijaty byte ulozen do r1
        reti

TX_int:
        push  r16
        ld    r16, X+
        out   UDR, r16
        dec   r3    // pocet--
        brne  endTX // if (pocet != 0) skoci
        // zakaz preruseni od UDR
        in    r16, UCSRB
        andi  r16, ~(1<<UDRIE)  // UDRIE do 0
        out   UCSRB, r16
endTX:
        pop   r16
        reti
```

Příznaky chyb při příjmu

- Lze využít kontrolních mechanismů a zjišťovat zda nenastala při příjmu chyba:
  - Framing Error (FE, chyba rámce – špatný stop bit)
  - Data OverRun (DOR, přetečení přijímacího bufferu)
  - Parity Error (PE, chybná parita v přijatých datech)
- příznaky FE, DOR a PE v registru UCSRA – chyba = příznak do 1, negenerují žádná přerušení !
- použití → před čtením dat kontrola příznaků