Laboratorní úloha na předmět *Periferní zařízení*

Úloha E: Sériová komunikace – RS232

Datum měření: *4.dubna 2006*

Michal Augustýn
Josef Bouška
Martin Prchlík
Michal Trs

1. Zadání

Sestavte program pro komunikaci po sériové lince a to přímým programováním registrů asynchronního obvodu UART 8250 dle následujících pokynů:

- napište program pro vysílač znaků po sériové lince,
- napište program pro přijímač znaků přicházejících po sériové lince,
- programy odlaďte a předveďte funkčnost na propojené dvojici počítačů PC; přijímané a vysílané znaky zobrazujte v oddělených částech obrazovky monitoru, prozkoumejte chování obvodu 8250 při různých rychlostech.

Poznámka: Programy nejprve napište bez použití přerušení, až poté s využitím služeb přerušovacího systému.

2. Teorie

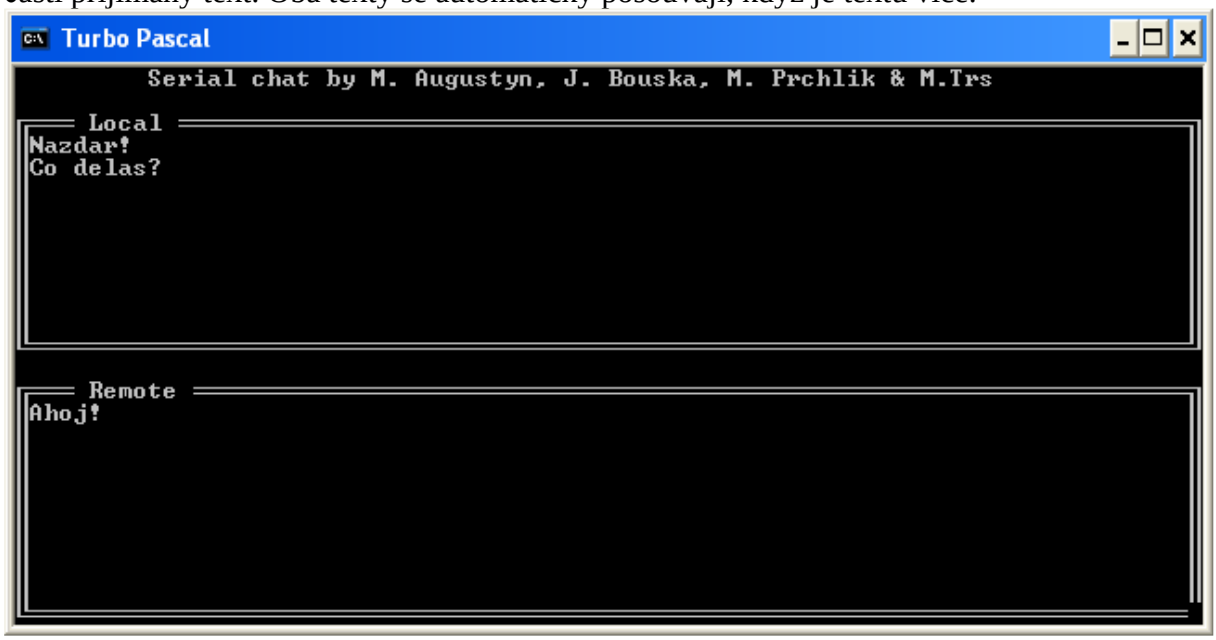
Rozhraní RS-232 (podle evropské normy V.24) je určeno pro sériový přenos dat mezi dvěma jím vybavenými zařízeními přes nějaké přenosové médium, nejčastěji telefonní síť. Pro kratší vzdálenosti je možno toto vynechat a zařízení propojit přímo, nehledě na možnost propojení prakticky s jakýmkoliv zařízením s příslušným obslužným programem. V našem měření použijeme upravený kabel (zvaný nulový modem) na přímé propojení dvou osobních počítačů, vybavených tímto rozhraním.
Standardizovaná specifikace definuje všechny vlastnosti rozhraní tak, aby byla zaručena propojitelnost zařízení různých výrobců. Máme tu především vyjmenování všech signálů, kterými rozhraní ovládáme, popis jejich významu, výkres propojovacích konektorů a způsob přenosu informace po vodičích.
Původní podoba rozhraní využívá ke své práci 25 signálů. Mnoho z nich se nejčastěji v praktických aplikacích nevyskytuje, setkáme se proto s krátkou (4 signály), střední (9) a velkou variantou (všech 25 signálů). My, protože máme k dispozici zástrčku s 25 kolíky, použijeme poslední z nich.
Propojovací kabel má 25-pinový konektor Canon s kolíčky na obou koncích, na počítačích jsou podobné konektory, jenom s dutinkami.
Zařízení komunikují v arytmickém režimu, každý balíček dat (jeho velikost můžeme určit, pro naše účely vybereme 8 bitů) je uvozen synchronizační značkou (start bit) a podobnou i ukončen (stop bit, jehož velikost si můžeme vybrat). Přenos může být navíc zabezpečen paritou. Řízení toku dat po kanále je hardwarové.
Čip, který ovládá činnost rozhraní, je UART 16550, je však zpětně kompatibilní s verzí 8250 pro PC XT a to jak programově, tak vývodově. Programováním čipu nastavíme konkrétní atributy, se kterými bude pak přenos dat probíhat.

3. Implementace

Program s využitím přerušení i bez využití přerušení jsme implementovali v programovacím jazyce Borland Pascal 7.0, v němž byly dodány základní kostry programů, ze kterých se mělo při implementaci vycházet.

3.1. Vykreslování

V obou programech je využita jednotka *Drawing*, která zajišťuje vykreslování odesílaného a přijímaného textu. Pro vykreslování jsou použity jen procedury a funkce dostupné ve standardních jednotkách Pascalu. Jednotka je napsána tak, že ihned po inicializaci jednotky (tedy po spuštění programu) dojde k výmazu obrazovky, vykreslení rámečků a statických textů.
Jednotka pracuje tak, že se do horní poloviny obrazovky vypisuje odesílaný text, do dolní části přijímaný text. Oba texty se automaticky posouvají, když je textu více.



Drawing.pas

```
unit Drawing;

interface

  procedure LocalWrite(c: char);
  procedure RemoteWrite(c: char);


implementation
  uses crt;

  const
    width = 78;
    height = 10;
    bX = 2;
    lbY = 4;
    rbY = 16;

  var lX, lY, rX, rY : integer;


  procedure WriteBorder;
```

```
  var i : integer;
  begin
    Window(1, 1, 80, 25);
    GotoXY(10, 1);
    Write('Serial chat by M. Augustyn, J. Bouska, M. Prchlik & M.Trs');

    {rovne cary}
    for i := bX to bX + width - 1 do
      begin
        GotoXY(i, lbY - 1);           Write(chr(205));
        GotoXY(i, lbY + height - 1);  Write(chr(205));
        GotoXY(i, rbY - 1);           Write(chr(205));
        GotoXY(i, rbY + height - 1);  Write(chr(205));
      end;

    {svisle cary}
    for i := lbY to lbY + height - 2 do
      begin
        GotoXY(bX - 1, i);          Write(chr(186));
        GotoXY(bX + width, i);      Write(chr(186));
      end;
    for i := rbY to rbY + height - 2 do
      begin
        GotoXY(bX - 1, i);          Write(chr(186));
        GotoXY(bX + width, i);      Write(chr(186));
      end;

    {rohy}
    GotoXY(bX - 1, lbY - 1);        Write(chr(201));
    GotoXY(bX + width , lbY - 1);   Write(chr(187));
    GotoXY(bX - 1, lbY + height - 1);        Write(chr(200));
    GotoXY(bX + width , lbY + height - 1);   Write(chr(188));

    GotoXY(bX - 1, rbY - 1);        Write(chr(201));
    GotoXY(bX + width , rbY - 1);   Write(chr(187));
    GotoXY(bX - 1, rbY + height - 1);        Write(chr(200));
    {GotoXY(bX + width , rbY + height - 1);   Write(chr(188));}

    GotoXY(bX + 3, lbY - 1);    Write(' Local ');
    GotoXY(bX + 3, rbY - 1);    Write(' Remote ');

  end;


procedure WriteUni(c: char; bY: integer; var x, y: integer);
  begin
    Window(bX, bY, bX + width - 1, bY + height - 2);
    case c of
      #13: begin
            x := 1;
            Inc(y);
            if y = height then
              begin
               Dec(y);
               GotoXY(1, 1);
               DelLine;
              end;
          end;
      #10: ;
      #27: ;
    else
      begin
        GotoXY(x, y);
        Write(c);
        Inc(x);
        if x = width then
          begin
            x := 1;
            Inc(y);
            if y = height then
              begin
               Dec(y);
               GotoXY(1, 1);
               DelLine;
              end;
          end;
      end;
```

```
      end;
      Window(bX, lbY, bX + width - 1, lbY + height - 2);
      GotoXY(lX, lY);
    end;


  procedure LocalWrite(c: char);
    begin
      WriteUni(c, lbY, lX, lY);
    end;


  procedure RemoteWrite(c: char);
    begin
      WriteUni(c, rbY, rX, rY);
    end;



  begin

    lX := 1;
    lY := 1;
    rX := 1;
    rY := 1;

    ClrScr;
    WriteBorder;
    Window(bX, lbY, bX + width - 1, lbY + height - 2);
    GotoXY(lX, lY);

end.
```

3.2.Komunikace bez využití přerušení

Tento program nám fungoval od začátku naprosto bez problémů.
Pouze stačí provést správně inicializaci podle daných materiálů a následně běží smyčka, která testuje stisknutou klávesu. Při stisku klávesy dojde k jejímu výpisu na obrazovku a zároveň odeslání. V hlavní smyčce programu se dále testuje dostupnost nových dat na portu. Pokud jsou připravena data k příjmu, dojde k výpisu přijatého znaku do příslušné části obrazovky.
Program se spouští s parametrem, který udává bázi portu (tedy např. pro *COM1* je to hodnota *3f8*). K ukončení programu dojde po stisku klávesy *Escape*.

Normal.pas

```
Program Normal;

uses DOS,CRT, Drawing;

const
{------------------- COM ports in PC -------------------}

 COM1= $3f8;
 COM2= $2f8;

{------------------- 8250 registers --------------------}

 RBR= 0;          { received data - read only -  if DLAB=0 }
 THR= 0;          { transmit data - write only - if DLAB=0 }

 DLL= 0;          { baud generator (LSB) - if DLAB=1 }
 DLM= 1;          { baud generator (MSB) - if DLAB=1 }

 IER= 1;          { interrupt enable - r/w }
 IIR= 2;          { interrupt identification - read only }
 LCR= 3;          { line control register - r/w }
 MCR= 4;          { modem control register - r/w }
 LSR= 5;          { line status - read only }
```

```
 MSR= 6;          { modem status register - read only }

{------------------- 8250 register bits ----------------}

 IER_RX= 1;       { intr on received data - priority 2 }
 IER_TX= 2;       { intr on data transmitted - priority 3 }
 IER_ERR= 4;      { intr on receive error (see LSR) - priority 1 }
 IER_MS= 8;       { intr on modem sts change (see MSR) - priority 4 }

 IIR_PEND= 1;     { active intr request }
 IIR_MASK= 6;     { source id mask }
 IIR_MS= 0;       { modem sts change }
 IIR_TX= 2;       { data transmitted }
 IIR_RX= 4;       { data received }
 IIR_ERR= 6;      { receive error }

 LCR_5BIT= 0;     { byte length codes }
 LCR_6BIT= 1;
 LCR_7BIT= 2;
 LCR_8BIT= 3;

 LCR_2STOP= 4;     { two stop bits }
 LCR_PTYEN= 8;     { parity enabled }
 LCR_EVPTY= $10;   { even parity }
 LCR_FIXPTY= $20;  { fixed parity }
 LCR_INTREN= $40;  { interrupt enabled }
 LCR_DLAB= $80;   { baud gen. access enabled }

 MCR_DTR= 1;       { DTR modem signal }
 MCR_RTS= 2;       { RTS modem signal }
 MCR_OUT1= 4;      { user signal }
 MCR_OUT2= 8;      { user signal - wired to 3 state IRQ driver }
 MCR_LOOP= $10;   { test loop }

 LSR_RX= 1;        { data received  }
 LSR_OR= 2;        { overrun - read once }
 LSR_PR= 4;        { parity error - read once }
 LSR_FR= 8;        { frame error - read once }
 LSR_BT= $10;     { break error - read once }
 LSR_BF= $20;     { tx buffer empty }
 LSR_TX= $40;     { serializer empty }

 MSR_CSR_CH= 1;    { CSR has changed - read once }
 MSR_DSR_CH= 2;    { DSR has changed - read once }
 MSR_RI_END= 4;    { RI ended - read once }
 MSR_RLSD_CH= 8;   { RLSD changed - read once }
 MSR_CSR= $10;
 MSR_DSR= $20;
 MSR_RI= $40;
 MSR_RLSD= $80;

 DLM_300= 1;       { 300 Baud }
 DLM_HIGH= 0;      { more than 300 Baud }

 DLL_300= $80;    { 300 Baud }
 DLL_600= $c0;    { 600 Baud }
 DLL_1200= $60;   { 1200 Baud }
 DLL_2400= $30;   { 2400 Baud }
 DLL_4800= $18;   { 4800 Baud }
 DLL_9600= $0c;   { 9600 Baud }
 DLL_19200= $06;  { 19200 Baud }


{---------------------------------------------------------------------}
type word=0..65535;
     parstr=string[255];


{---------------------------------------------------------------------}
var
  com: word;
  loop: boolean;
  c: char;
  argv1, argv2: parstr;

{---------------------------------------------------------------------}
procedure init_com(portaddr:word; loop: boolean);
```

```
begin
  port [portaddr+LCR]:= LCR_DLAB;   { path to baud gen on }
  port [portaddr+DLM]:= DLM_HIGH;   { more than 300 Baud  }
  port [portaddr+DLL]:= DLL_1200;   { 1200 Baud }
  port [portaddr+LCR]:= LCR_8BIT;   { baud gen off, 8 bit, no par }
  port [portaddr+IER]:= 0;          { no interrupt }
  if loop then
     port [portaddr+MCR]:= MCR_DTR or MCR_LOOP
  else
     port [portaddr+MCR]:= MCR_DTR;
end;

{---------------------------------------------------------------------}
function rx_rdy (portaddr: word): boolean;
begin
  rx_rdy:=((port[portaddr+LSR] and LSR_RX) <> 0);
end;

{---------------------------------------------------------------------}
function tx_rdy(portaddr: word): boolean;
begin
  tx_rdy:=((port [portaddr+LSR] and LSR_BF) <> 0);
end;

{---------------------------------------------------------------------}
function rx_get(portaddr: word): char;
begin
  rx_get:= chr(port [portaddr+RBR]);
end;

{---------------------------------------------------------------------}
procedure tx_put(portaddr: word; c : char);
begin
  port [portaddr+THR]:= ord (c);
end;

{---------------------------------------------------------------------}
function hx (t: parstr): word;
var ix: integer;
    w:  word;
    ok: boolean;

begin w:=0; ok:=true;
      for ix:=1 to length(t) do if ok then
      begin if t[ix] in ['0'..'9'] then      w:=16*w+ord(t[ix])-ord('0')
            else if t[ix] in ['A'..'F'] then w:=16*w+ord(t[ix])-ord('A')+10
            else if t[ix] in ['a'..'f'] then w:=16*w+ord(t[ix])-ord('a')+10
            else ok:=false;
            if w>$3FF then ok:=false;
      end;
      if ok then hx:=w else hx:=0;
end;

{---------------------------------------------------------------------}
begin
  com:=COM1;
  loop:=false;
  c:='a';

  argv1:=ParamStr(1);
  argv2:=ParamStr(2);

  case ParamCount of
  0: begin writeln ('Usage: serial <port hex addr> [l]');
           writeln ('COM1 is at 3F8, COM2 is at 2F8.');
           halt;
     end;

  1: begin com:=hx(argv1);
           if com=0 then
           begin writeln ('Port address must be hex number in the range 10..3FF');
                 halt;
           end;
     end;


  2: begin com:=hx(argv1);
```

```
           if com=0 then
           begin writeln ('Port address must be hex number in the range 10..3FF');
                 halt;
           end;
           loop:=true;
     end;

  end;

  init_com(com,loop);

  while true do
    begin
      if (rx_rdy(com)) then RemoteWrite(rx_get(com));
      if keypressed then
        begin
          c := readkey;
          if Ord(c) = 27 then break;
          if ord(c) = 0 then
            begin
              c := readkey;
              if Ord(c) = 27 then break;
            end;
          if tx_rdy(com) then
            begin
              tx_put(com, c);
              LocalWrite(c);
            end;
        end;
    end;

end.
```

3.3.Komunikace s využitím přerušení

Tento program nám v laboratoři nefungoval (vyzkoušet doma nebylo možno). Nakonec se nám ho ale podařilo upravit do funkční verze. Problémů bylo několik. V daném mustru byla např. jako bázová adresa uvedena adresa *300h*, což jsme přehlídli. Bylo nutné tedy hodnotu přepsat na *3f8h*. Dále jsme zjistili, že záleží na tom, zda se nejprve provede inicializace portu nebo dojde k nastavení vektoru přerušení. Pro správnou funkci bylo třeba, aby došlo nejdříve k nastavení vektoru přerušení a až následně k inicializaci portu.
Tento program se od verze bez využití přerušení liší jen inicializací a tím, že se v hlavní smyčce programu neprovádí testování, zda jsou připravena data k příjmu. Místo toho tu je procedura označená direktivou interrupt, což je obsluha přerušení. V této obslužné proceduře dojde k výpisu přijatého znaku a k odhlášení přerušení (odesláním hodnoty *20h* na port *20h*).
K ukončení programu dojde po stisku klávesy *Escape*.

Inter.pas

```
Program RS232;

uses DOS,CRT, Drawing;

const BAS=$3f8;

var II: integer;
    OK: boolean;
    DATA: byte;
    PLAT: boolean;
    c: char;


procedure INTER; interrupt;
var b:byte;
```

```
begin
  RemoteWrite(chr(port[BAS]));
  b := port[bas+2];
  port[$20]:=$20;
end;


procedure Send(c:char);
begin
  while (port[bas+5] and $60) <> $60 do;
  port[bas]:=ord(c);
end;

var b, imask: byte;
    p: Pointer;

begin

  b := port[bas+5];
  b := port[bas+2];

  getintvec($0C, p);
  setintvec($0C, addr(inter));


  port[bas + 1] := 0;                    { deny innterupt on com }

  port[bas + 3]:= $80;                   { DLAB=1, for speed settings }
  port[bas + 0]:= $60;           { speed to 1200 }
  port[bas + 1]:= 0;
  port[bas + 3]:= $03;                   { 8b data, 1b stop, no parity }
 { port[bas + 2]:= $07;}                 { 1 byte length buffer and its cleaning }
  port[bas + 4]:= $0F;                   { OUT2 to 1, enable int }
  imask := port[$21];
  port[$21]:= imask and $EF;
  port[bas + 1]:= $01;                   { int on received character }


  while true do
    begin
     c := readkey;
     if Ord(c) = 27 then break;
     if ord(c) = 0 then
      begin
        c := readkey;
        if Ord(c) = 27 then break;
      end;
     Send(c);
     LocalWrite(c);
    end;

    port[$21] := imask;
    setintvec($0C, p);
  end.
```

4. Závěr

Povedlo se nám úspěšně implementovat a odladit programy pro komunikaci po sériové lince přímým programováním registrů asynchronního obvodu UART 8250.
Byly vytvořeny dvě verze programu – s využitím přerušení a bez využití přerušení. Oba tyto programy využívají jednotku *Drawing*, která slouží k vykreslování odeslaného a přijatého textu.