Dílenská praxe

A4	7. Alfanumerický displej		
Petřík Vít		1/14	Známka:
19. 2. 2020	Datum odevzdání:	25. 3. 2020	Odevzdáno:

Zadání:

Zpracujte program v programovacím jazyce C ovládající alfanumerický displej tak, aby obsahoval nejméně tyto funkce:

1) volbu druhu displeje (7segmentový/14segmentový)
2) zobrazení vhodně zvolené množiny znaků pro každý typ displeje
3) vhodně zvolená datová a programová struktura

Postup:

- Návrh zapojení
- Nadefinování datové struktury.
- Zapsání kombinací pro displeje.
- Zápis hlavní řídící logiky.
- Otestování.

Propojení PC a displeje:

PORT 1 - output	
Pin	Význam
0	Tx
1	Clk
2	-
3	-
4	-
5	-
6	-
7	-

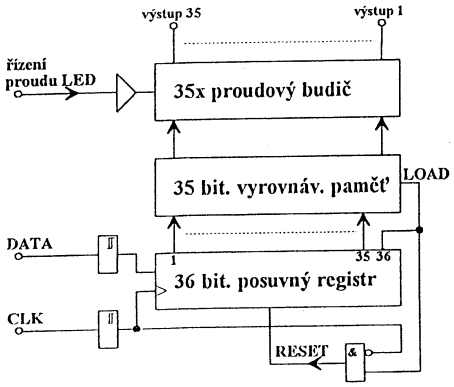
Schéma zapojení:

LABORATOŘE MIKROPROCESOROVÉ TECHNIKY

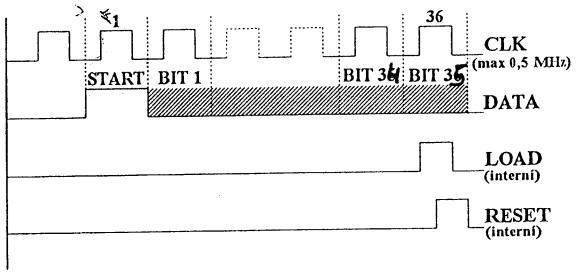
název úlohy: **Sériový displej LED s IO M5451**

Integrovaný obvod M5451 je určen k připojení až 35 segmentů LED displaye k mikroprocesoru (v multiplexním režimu až 64 segmentů). Informace o zobrazovaných datech se do obvodu přenáší sériově. K přenosu jsou potřeba jen 2 signálové vodiče: DATA a CLOCK (takt). Přenesená informace je platná na výstupu obvodu po přenosu posledního bitu. Přenos se zahajuje jedním startovacím bitem. Obvod umožňuje řídit jas celého displaye odporem připojeným mezi vývodem 19 a $+U_{CC}$. Výstupy jsou typu proudového zdroje a není tedy třeba použít omezovací odpory.

Blokové schéma obvodu:

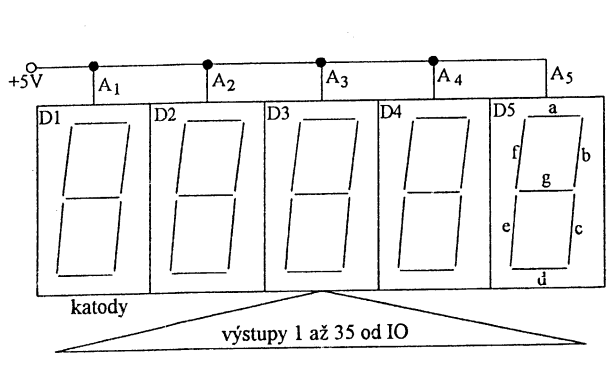


Časování řídících signálů:

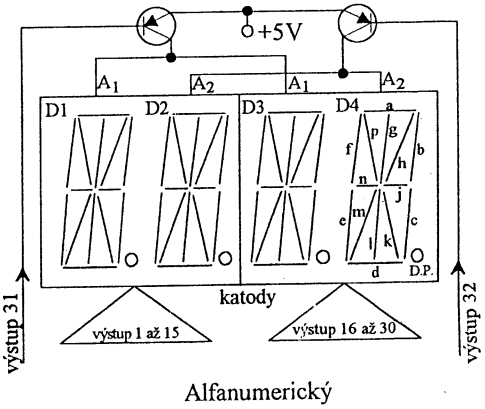


Sérioparalelní převodník je realizován posuvným registrem s délkou 36 bitů, který se ovládá vnějšími signály DATA a CLK a vnitřním RESET pro nulování všech bitů. RESET je generován také vždy po zapnutí napájení. Logické hodnoty ze vstupu DATA se zapíší a údaje v registru se posouvají s každou náběžnou hranou signálu CLK. Objeví-li se na posledním bitu posuvného registru log. 1, vygeneruje se signál LOAD, který zkopíruje 35 předcházejících bitů do vyrovnávacího registru a signál RESET. Výstupy vyrovnávacího registru již přímo ovládají jednotlivé budiče LED. Pro změnu údaje zobrazeného na display je nutno zapsat do vstupu DATA posuvného registru 36 binárních hodnot: první bit na sériové lince musí být vždy úrovně „H" (start bit), další bity pak mají hodnotu „H" - má-li příslušný segment v pořadí svítit, nebo „L" - má-li být segment zhasnut. Data se zapíší a registr posune vždy při změně hodinového signálu CLK z log. hodnoty „L" na „H". V případě použití více než 35 segmentů displaye je zvoleno zobrazení v multiplexním režimu a data se musí obměňovat alespoň 50krát za sekundu, aby se zobrazení jevilo jako stálé a neblikalo (v našem konkrétním zapojení u alfanumerického displaye).

Rozmístění pozic a označení segmentů displaye:

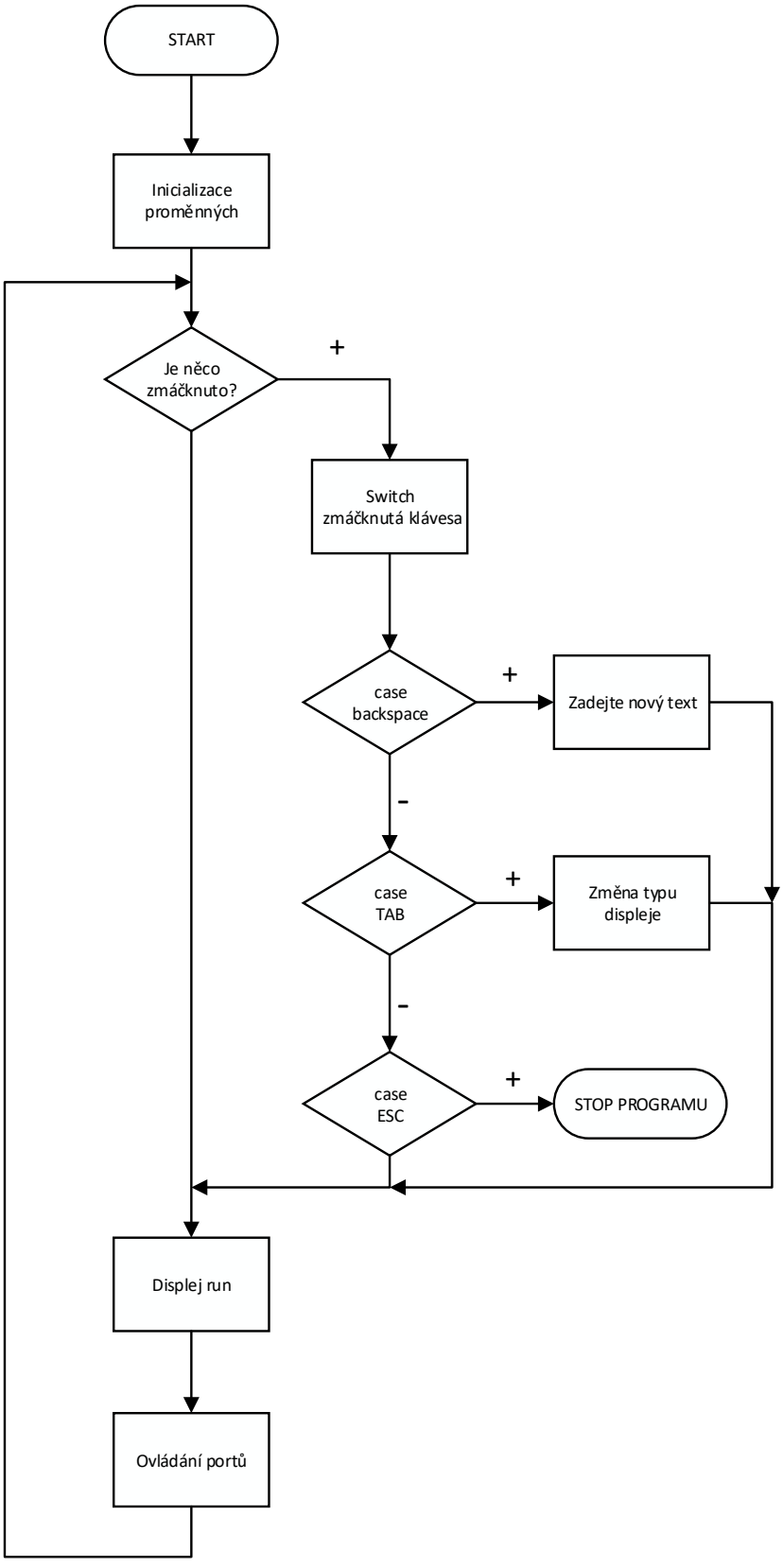


Numerický



Alfanumerický

Vývojový diagram:



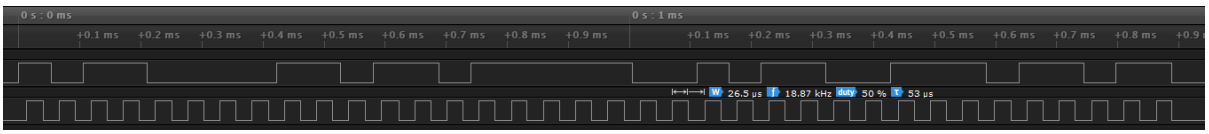
Výpis programu:

Příloha 1 – C++

Závěr:

Program funguje tak jak má. Kombinace pro displeje jsem vzal z veřejně dosutpného repozitáře, který se mi osvědčil z úlohy sériové linky. Problémem byl 14 segmentový displej, který má segmenty uspořádány jinak než repozitář. Tento problém jsem vyřešil programovým přeskládáním sledu bitů – nemusel jsem tedy nic otrocky přepisovat.

Při realizaci samotného přenosu jsem se soustředil na to, aby měl signál pokud možno všechny náležitosti. V podstatě tak, aby signál vypadal tak jako v datasheetu. To se mi povedlo viz screenshot z logického analyzátoru zobrazující přenos jednoho datového rámce.



Přílohy:

- Příloha 1 – 9 stran

Příloha 1

```
#include <conio.h>
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <iostream.h>

//porty
#define P1 0x300
#define P2 0x301
#define P3 0x300
#define P4 0x301

//specialni klavesy
#define TAB 9
#define BACKSPACE 8
#define ESC 27

//konstanty pro vyber displeje
#define NUMERIC_DISP 0
#define ALPHANUMERIC_DISP 1

char PORT[4];

typedef struct
{
    char port;
    char bit;
} pin;

//trida citace, je vyuzit pro generovani taktu
class Counter
{
private:
    int count, resolution, prescaler, prescalerCount;

public:
    //nastaveni citace
    //resolution, prescaler
    void setup(int res, int pre)
    {
        resolution = res;
        prescaler = pre;
        count = 0;
        prescalerCount = 0;
    }

    //vrati hodnotu citace
    int getCount()
    {
        return count;
    }

    //vyresetovani stavi citace
    void reset()
    {
        count = 0;
        prescalerCount = 0;
    }

    //cyklus citace
```

```
    void run()
    {
        if (prescalerCount >= prescaler)
        {
            count++;
            prescalerCount = 0;
        }
        if (count + 1 > resolution)
        {
            count = 0;
        }
        prescalerCount++;
    }
};

//trida bit-bangovaneho serivoeho prenosu, jenom Tx
class SerialTransmit
{
private:
    pin Tx, Clk;
    Counter square;
    char *buffer;
    int bufferSize;
    int index;
    char completed;

public:
    //konstruktor tridy
    SerialTransmit()
    {
        square.setup(4, 0); //nastaveni citace
        index = 0;
        bufferSize = 0;
        completed = 1;
    }

    //nastaveni vystupnich pinu
    void setup(pin Tx_pin, pin Clk_pin)
    {
        Tx = Tx_pin;
        Clk = Clk_pin;
    }

    //zapis bufferu
    char write(char *buff, int size)
    {
        if (!transmitCompleted())
            return 0;
        buffer = buff;
        bufferSize = size;
        index = 0;
        completed = 0;
        return 1;
    }

    //je prenos hotov?
    char transmitCompleted()
    {
        return completed;
    }

    //metoda, ktera je volana pri kazdem prubehu programu
```

```
    void run()
    {
        //pokud neni prenos hotov
        if (!completed)
        {
            //clock
            PORT[Clk.port] ^= (-(square.getCount() / 2) ^ PORT[Clk.port]) & (1UL << Clk.bit);
            //pokud je hodinovy signál v 1. ctvrtine nahodime nove dato
            if (square.getCount() == 1)
            {
                PORT[Tx.port] ^= (-(!!(buffer[index] & (1 << 0))) ^ PORT[Tx.port]) & (1UL << Tx.bit);
                index++;
                bufferSize--;
            }
            square.run();
            //pokud je prenos hotov
            if (!bufferSize && (square.getCount() == 0))
            {
                completed = 1;
                square.reset();
            }
        }
        //pokud je prenos hotov zajistime, ze na pinech bude 0
        else
        {
            PORT[Clk.port] ^= (-0 ^ PORT[Clk.port]) & (1UL << Clk.bit);
            PORT[Tx.port] ^= (-0 ^ PORT[Tx.port]) & (1UL << Tx.bit);
        }
    }
};

//trida displeje/posuvneho registru
class M5451
{
private:
    static const unsigned char SevenSegmentASCII[96];
    static const unsigned int FourteenSegmentASCII[96];

    SerialTransmit serial;
    char *string;

    char displayType;
    char newString;
    char anodeSwitch;

    char outputBuffer[36];

    //prida do bufferu bitovou kombinaci pro 7 segmentovy displej
    void addToBuffer7seg(char byte, char startBit, char numOfBits)
    {
        for (char i = 0; i < numOfBits; i++)
        {
            outputBuffer[startBit + i] = !!(byte & (1 << i));
        }
    }

    //prida do bufferu bitovou kombinaci pro 14 segmentovy displej
    void addToBuffer14seg(int byte, char startBit, char numOfBits)
    {
        //kombinace, ktere jsem vzal z githubove repozitare byly docela mimo :)
        //abych kombinace nemusel otrocky prepisovat, napsal jsem si takovou tu konstrukci
        //ktera kombinace opravi a na displeji jsou tedy zobrazeny spravne znaky
```

```
        //index pole bufferu, na ktere se ma ulozit dany bit se vezme z tohoto pole
        static char bitPosition[15] = {0, 1, 2, 3, 4, 5, 12, 8, 13, 6, 7, 11, 10, 9, 14};
        for (char i = 0; i < numOfBits; i++)
        {
            outputBuffer[startBit + bitPosition[i]] = !!(byte & (1 << i));
        }
    }

public:
    //konstruktor tridy
    M5451(pin Tx_pin, pin Clk_pin, char disp, char *s)
    {
        onChange(); //vymazani bufferu
        serial.setup(Tx_pin, Clk_pin); //nastaveni seriovky
        displayType = disp;
        string = s;
        anodeSwitch = 0; //prohazovani anod
    }

    //nastavení pointeru na stringu
    void setString(char *s)
    {
        onChange();
        string = s;
    }

    //zmenu typu displeje
    void setDisplayType(char disp)
    {
        onChange();
        displayType = disp;
    }

    //typ displeje?
    char getDisplayType()
    {
        return displayType;
    }

    //vynulovani bufferu
    void onChange()
    {
        newString = 1;
        for (char i = 0; i < 36; i++)
        {
            outputBuffer[i] = 0;
        }
    }

    //prenos hotov?
    char transmitCompleted()
    {
        return serial.transmitCompleted();
    }

    void run()
    {
        //rozhodnuti podle typu displeje
        switch (displayType)
        {
        case NUMERIC_DISP:
            //pokud mame novy text a zaroven je dokoncen predchozi prenos
```

```
            if (newString && transmitCompleted())
            {
                char stringElement = 0;
                outputBuffer[0] = 1;    //start bit
                //procyklovani textem
                for (char i = 0; i < 5; i++)
                {
                    stringElement = string[i] >= ' ' ? SevenSegmentASCII[string[i] - ' '] : 0;
                    addToBuffer7seg(stringElement, (i * 7) + 1, 7); //zapsani do bufferu
                }
                serial.write(outputBuffer, sizeof(outputBuffer) / sizeof(*outputBuffer));
                //poslani bufferu na seriovku
                newString = 0;
            }
            break;
        case ALPHANUMERIC_DISP:
            if (transmitCompleted())
            {
                int stringElement = 0;
                outputBuffer[0] = 1;
                char help = 0;
                //jedna anoda
                if (!anodeSwitch)
                {
                    stringElement = string[0] >= ' ' ? FourteenSegmentASCII[string[0] - ' '] : 0;
                    addToBuffer14seg(stringElement, 1, 15); //zapis do bufferu

                    stringElement = string[2] >= ' ' ? FourteenSegmentASCII[string[2] - ' '] : 0;
                    addToBuffer14seg(stringElement, (1 * 15) + 1, 15);
                }
                //druha anoda
                else
                {
                    stringElement = string[1] >= ' ' ? FourteenSegmentASCII[string[1] - ' '] : 0;
                    addToBuffer14seg(stringElement, 1, 15);

                    stringElement = string[3] >= ' ' ? FourteenSegmentASCII[string[3] - ' '] : 0;
                    addToBuffer14seg(stringElement, (1 * 15) + 1, 15);
                }
                //zapis ktera anoda mi byt aktivni
                outputBuffer[31] = !anodeSwitch;
                outputBuffer[32] = anodeSwitch;

                //poslani na seriovku
                serial.write(outputBuffer, sizeof(outputBuffer) / sizeof(*outputBuffer));
                anodeSwitch = !anodeSwitch; //zmena anody
            }
            break;
        }
        serial.run();
    }
};

//definice kombinaci 7-segmentoveho displeje
const unsigned char M5451::SevenSegmentASCII[96] = {
    0x00, /* (space) */
    0x86, /* ! */
    0x22, /* " */
    0x7E, /* # */
    0x6D, /* $ */
    0xD2, /* % */
    0x46, /* & */
```

```
0x20, /* ' */
0x29, /* ( */
0x0B, /* ) */
0x21, /* * */
0x70, /* + */
0x10, /* , */
0x40, /* - */
0x80, /* . */
0x52, /* / */
0x3F, /* 0 */
0x06, /* 1 */
0x5B, /* 2 */
0x4F, /* 3 */
0x66, /* 4 */
0x6D, /* 5 */
0x7D, /* 6 */
0x07, /* 7 */
0x7F, /* 8 */
0x6F, /* 9 */
0x09, /* : */
0x0D, /* ; */
0x61, /* < */
0x48, /* = */
0x43, /* > */
0xD3, /* ? */
0x5F, /* @ */
0x77, /* A */
0x7C, /* B */
0x39, /* C */
0x5E, /* D */
0x79, /* E */
0x71, /* F */
0x3D, /* G */
0x76, /* H */
0x30, /* I */
0x1E, /* J */
0x75, /* K */
0x38, /* L */
0x15, /* M */
0x37, /* N */
0x3F, /* O */
0x73, /* P */
0x6B, /* Q */
0x33, /* R */
0x6D, /* S */
0x78, /* T */
0x3E, /* U */
0x3E, /* V */
0x2A, /* W */
0x76, /* X */
0x6E, /* Y */
0x5B, /* Z */
0x39, /* [ */
0x64, /* \ */
0x0F, /* ] */
0x23, /* ^ */
0x08, /* _ */
0x02, /* ` */
0x5F, /* a */
0x7C, /* b */
0x58, /* c */
0x5E, /* d */
```

```
    0x7B, /* e */
    0x71, /* f */
    0x6F, /* g */
    0x74, /* h */
    0x10, /* i */
    0x0C, /* j */
    0x75, /* k */
    0x30, /* l */
    0x14, /* m */
    0x54, /* n */
    0x5C, /* o */
    0x73, /* p */
    0x67, /* q */
    0x50, /* r */
    0x6D, /* s */
    0x78, /* t */
    0x1C, /* u */
    0x1C, /* v */
    0x14, /* w */
    0x76, /* x */
    0x6E, /* y */
    0x5B, /* z */
    0x46, /* { */
    0x30, /* | */
    0x70, /* } */
    0x01, /* ~ */
    0x00, /* (del) */
};
//definice kombinaci 14-segmentoveho displeje
const unsigned int M5451::FourteenSegmentASCII[96] = {
    0x0000, /* (space) */
    0x4006, /* ! */
    0x0202, /* " */
    0x12CE, /* # */
    0x12ED, /* $ */
    0x3FE4, /* % */
    0x2359, /* & */
    0x0200, /* ' */
    0x2400, /* ( */
    0x0900, /* ) */
    0x3FC0, /* * */
    0x12C0, /* + */
    0x0800, /* , */
    0x00C0, /* - */
    0x4000, /* . */
    0x0C00, /* / */
    0x0C3F, /* 0 */
    0x0406, /* 1 */
    0x00DB, /* 2 */
    0x008F, /* 3 */
    0x00E6, /* 4 */
    0x2069, /* 5 */
    0x00FD, /* 6 */
    0x0007, /* 7 */
    0x00FF, /* 8 */
    0x00EF, /* 9 */
    0x1200, /* : */
    0x0A00, /* ; */
    0x2440, /* < */
    0x00C8, /* = */
    0x0980, /* > */
    0x5083, /* ? */
```

```
0x02BB, /* @ */
0x00F7, /* A */
0x128F, /* B */
0x0039, /* C */
0x120F, /* D */
0x0079, /* E */
0x0071, /* F */
0x00BD, /* G */
0x00F6, /* H */
0x1209, /* I */
0x001E, /* J */
0x2470, /* K */
0x0038, /* L */
0x0536, /* M */
0x2136, /* N */
0x003F, /* O */
0x00F3, /* P */
0x203F, /* Q */
0x20F3, /* R */
0x00ED, /* S */
0x1201, /* T */
0x003E, /* U */
0x0C30, /* V */
0x2836, /* W */
0x2D00, /* X */
0x00EE, /* Y */
0x0C09, /* Z */
0x0039, /* [ */
0x2100, /* \ */
0x000F, /* ] */
0x2800, /* ^ */
0x0008, /* _ */
0x0100, /* ` */
0x1058, /* a */
0x2078, /* b */
0x00D8, /* c */
0x088E, /* d */
0x0858, /* e */
0x14C0, /* f */
0x048E, /* g */
0x1070, /* h */
0x1000, /* i */
0x0A10, /* j */
0x3600, /* k */
0x0030, /* l */
0x10D4, /* m */
0x1050, /* n */
0x00DC, /* o */
0x0170, /* p */
0x0486, /* q */
0x0050, /* r */
0x2088, /* s */
0x0078, /* t */
0x001C, /* u */
0x0810, /* v */
0x2814, /* w */
0x2D00, /* x */
0x028E, /* y */
0x0848, /* z */
0x0949, /* { */
0x1200, /* | */
0x2489, /* } */
```

```
    0x0CC0, /* ~ */
    0x0000, /* (del) */
};

int main(void)
{
    //inicializace promennych a tridy displeje
    system("cls");
    char inputStr[] = "12345";
    cout << "Zobrazeny text muzete zmenit pomoci klavesy backspace\n\r";
    cout << "Typ displeje pomoci klavesy TAB\n\r";
    pin Tx = {0, 0};
    pin Clk = {0, 1};
    M5451 display(Tx, Clk, NUMERIC_DISP, inputStr);
    char exitLoop = 0;
    while (1)
    {
        //vezme stisknutou klavesu (pokud nejake je)
        if (kbhit())
        {
            switch (getch())
            {
                //zapis noveho textu
            case BACKSPACE:
                cout << "\n\rZadejte text k zobrazeni: ";
                gets(inputStr);
                display.onChange();
                break;
                //zmena typu displeje
            case TAB:
                display.setDisplayType(!display.getDisplayType());
                break;
                //odchod z programu
            case ESC:
                exitLoop = 1;
                break;
            }
        }

        //metoda displeje pro chod
        display.run();

        //nahozeni bytu na port
        outportb(P1, PORT[0]);

        //pokud zmackneme ESC a je dokonceno vysilani opustime smycku
        if (exitLoop && display.transmitCompleted())
        {
            break;
        }
    }
    //konec programu a navrat do Borlanda
    while (!kbhit())
        ;
    return 0;
}
```