Vážení zákazníci,

dovolujeme si Vás upozornit, že na tuto ukázku knihy se vztahují autorská práva, tzv. copyright.

To znamená, že ukázka má sloužit výhradně pro osobní potřebu potenciálního kupujícího (aby čtenář viděl, jakým způsobem je titul zpracován a mohl se také podle tohoto, jako jednoho z parametrů, rozhodnout, zda titul koupí či ne).

Z toho vyplývá, že není dovoleno tuto ukázku jakýmkoliv způsobem dále šířit, veřejně či neveřejně např. umisťováním na datová média, na jiné internetové stránky (ani prostřednictvím odkazů) apod.

redakce nakladatelství BEN – technická literatura
redakce@ben.cz

chceme zpracovat. Druhý operand je maska, která určuje místa provedení negace.

Bitové posuvy **<<** a **>>** provádí posuv dvojkové hodnoty o stanovený počet bitů doprava nebo doleva. První operand je posouvaná hodnota, druhý určuje počet posunutí v bitech.

2.4 Cykly

Cykly slouží pro opakované provádění příkazů. Používání cyklů je při programování mikrokontrolérů nezbytné, protože samotný hlavní program, používá nekonečnou smyčku řešenou pomocí cyklu. Hlavní program provede úvodní inicializaci, která spočívá například v nastavení činnosti periferií. Poté se rozbíhá nekonečná smyčka, která reaguje na vstupní hodnoty a podle toho řídí běh programu.

2.4.1 Cyklus while – cyklus s podmínkou na začátku

Nejjednodušším příkladem cyklu je příkaz **while**. Je to cyklus, který provádí uvedený příkaz tak dlouho, dokud je podmínka pravdivá. Platnost podmínky se testuje vždy před vykonáním příkazu. Formát:

```
while(<logický výraz>)
  <příkaz>;
```

Příklad 1:

Výchozí hodnota proměnné **x** je 0. Cyklus bude postupně zvyšovat proměnnou **x** tak, že se bude měnit jako 1, 2, 3, 4 a 5. Jakmile je podmínka **x<5** neplatná (to se stane v okamžiku, kdy x=5) je cyklus ukončen a program pokračuje dalšími příkazy.

```
uint8_t x=0;

while(x<5)
  x++;
```

Příklad 2:

Cyklus **while** umožňuje opakovaně vykonat pouze jeden příkaz. Zapíšeme-li za **while** více příkazů, bude se opakovaně provádět pouze první příkaz. Další příkazy se provedou pouze jednou a to až po dokončení cyklu.

Pokud je nutné provádět v cyklu více příkazů, musíme je zapsat do bloku. *Blok* je definován pomocí složených závorek. Blok se z hlediska cyklu chápe jako jeden (složený) příkaz.

Níže zapsaný příklad funguje podobně jako Příklad 1. Hodnota proměnné **x** je ale odesílána na port B. Jelikož operace **x++** předchází zápisu na port, jsou na port zapsány hodnoty 1, 2, 3, 4 a 5.

```
uint8_t x=0;

while(x<5)
{
  x++;
  PORTB=x;
}
```

Pokud pořadí dílčích operací v bloku vzájemně zaměníme, budou na port odesílány hodnoty 0, 1, 2, 3 a 4:

```
uint8_t x=0;

while(x<5)
{
  PORTB=x;
  x++;
}
```

Příklad 3:

Nekonečný cyklus vytvoříme tak, že do závorek příkazu **while** zapíšeme logický výraz, který představuje stále platnou podmínku. Nejjednodušší je tedy do závorek uvést nenulovou hodnotu (například 1).

Jak bylo naznačeno výše, nekonečný cyklus je používán pro zápis hlavního programu. Mikrokontrolér je stále taktován hodinovým kmitočtem, musí tedy stále provádět nějaké příkazy. Pokud by hlavní program skončil, bylo by chování mikrokontroléru nedefinované.

```
while(1)  //nekonečná smyčka
{
  //příkazy
}
```

2.4.2 Cyklus do..while – cyklus s podmínkou na konci

Cyklus **do**..**while** funguje podobně jako příkaz **while**. Rozdíl je však v tom, že podmínka je testována až po provedení příkazu, který je v cyklu zapsán. Formát:

```
do
  <příkaz>;
while(<logický výraz>)
```

Příklad:

```
uint8_t x=0;
do
  x++;
while(x<5)
```

Výchozí hodnota proměnné **x** je 0 a postupně se mění na hodnoty: 1, 2, 3, 4, 5, 6.

2.4.3 Cyklus for – cyklus s předem daným počtem opakování

Pro přehlednější zápis cyklu je možné použít příkaz **for**. Výhodou tohoto příkazu je, že zápis je strukturován do tří logických částí:

1. Inicializace řídicí proměnné cyklu,
2. Podmínka provádění cyklu,
3. Předpis pro změnu řídicí proměnné.

Jednotlivé části se zapisují do společných závorek a vzájemně se oddělují středníky. Opět je možné opakovaně provádět pouze jeden příkaz, pro seskupení více příznaků použijeme blok.

Formát:

```
for(<inicializace>;<podmínka>;<změna_řídicí_proměnné>)
  <příkaz>;
```

Příklad:

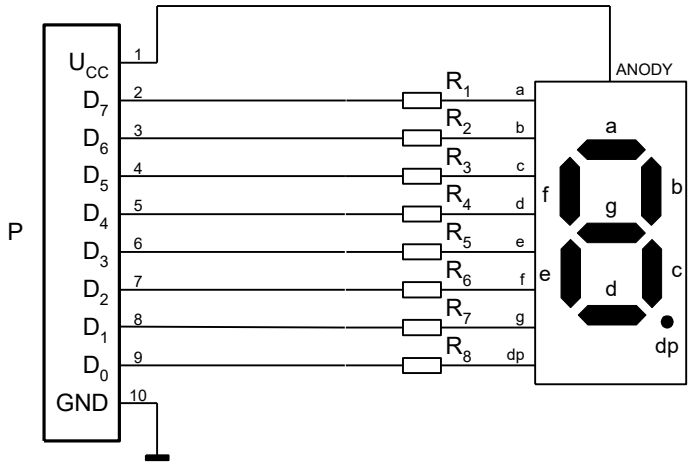
Výchozí hodnotu proměnné **x** nemusíme nastavovat před spuštěním cyklu, inicializaci provedeme až v rámci cyklu. Nastavili jsme výchozí hodnotu **x=0**, po každém průchodu se **x** zvýší o 1 a podmínka omezí provádění tak, že **x** bude postupně nabývat hodnot: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 a 10. Poslední průchod cyklem bude s hodnotu **x=9**, následně se však hodnota **x** zvýší ještě jednou a po ukončení cyklu bude **x=10**. Takže na port B se odešlou hodnoty 0 až 9, ale po ukončení cyklu bude **x=10**.

```
uint8_t x=0;

for(x=0;x<=9;x++)
  PORTB=x;
```

2.5 Přípravek M7SEG – 7segmentovka

Přípravek **M7SEG** použijeme pro prezentování výše uvedených programových konstrukcí v několika příkladech.



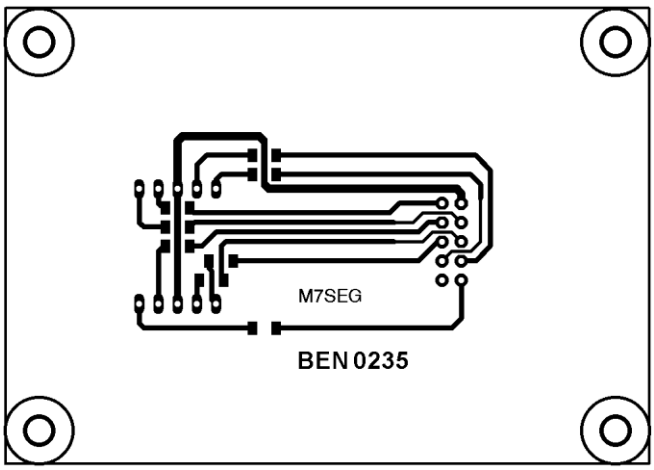
*Obr. 2.11 Schéma zapojení přípravku **M7SEG***

Schéma zapojení přípravku **M7SEG** je uvedeno na obr. 2.11, jedná se o přípravek vybavený 7segmentovkou se společnou anodou. Katody jednotlivých segmentů jsou připojeny přes omezovací rezistory R_1 až R_8, nyní není použit budič sběrnice typu **74245** jako u dříve popsaného přípravku **M8LED**. Segmenty svítí (opět) při log. 0.

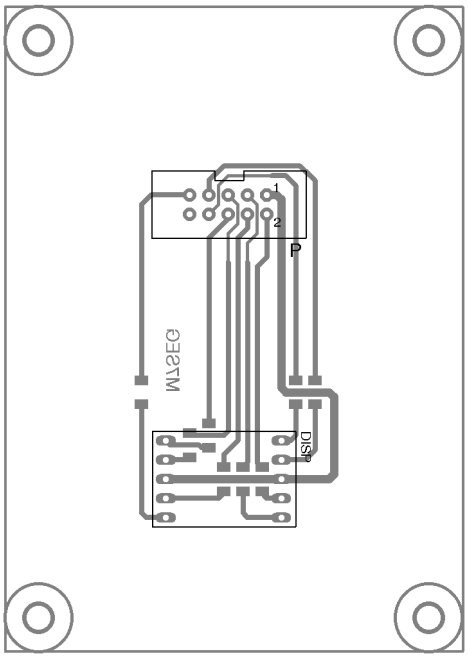
Rozpis součástek:

Označení	dle TME	stručný popis	počet
DISP	SA56-11EWA	7segmentový LED displej, velikost 14 mm, společná anoda	1 ks
R_1 až R_8	RC1206JR-07680R	SMD rezistor 680 Ω (velikost 1206)	8 ks
P	ZL231-10PG	kolíková lišta 2×5 rozteč 2,54 mm	1 ks

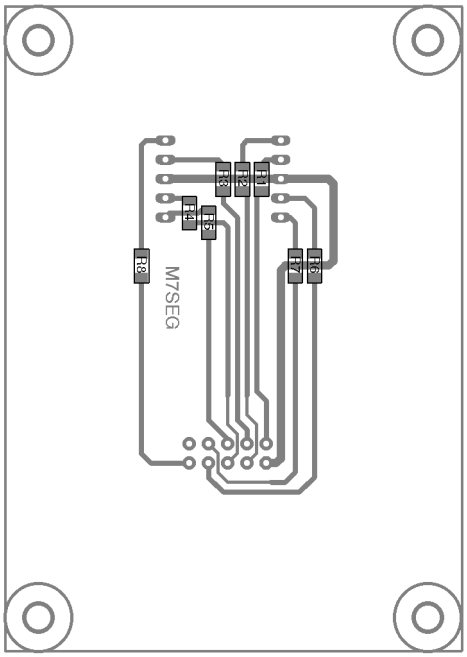
Jako dno lze použít jeden díl krabičky **KM-22** (TME).



*Obr. 2.12 Výkres desky plošných spojů přípravku **M7SEG***



Obr. 2.13 Osazovací plánek přípravku (strana běžných součástek)



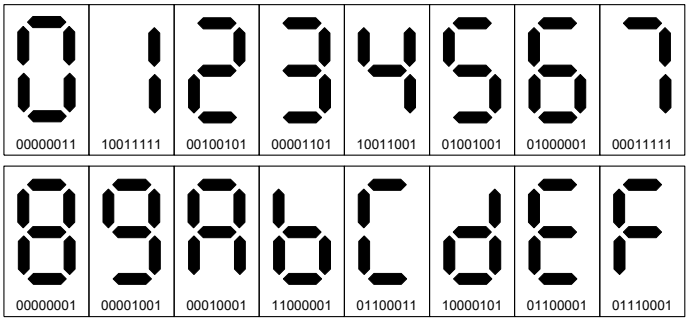
Obr. 2.14 Osazovací plánek přípravku (strana SMD)

2.6 PROG_02 – základní ovládání 7segmentovky

Přípravek **M7SEG** neobsahuje dekodér, který by hodnotu zobrazované číslice převedl na odpovídající kombinaci segmentů.

Dekódování tedy musíme provést programově. Pokud chceme zobrazit například číslici 0, je třeba rozsvítit segmenty **a**, **b**, **c**, **d**, **e**, **f** a segmenty **g** a **dp** nechat zhasnuté. Ze schématu dle obr. 2.11 vyplývá, že je třeba připojit dvojkovou kombinaci **00000011**.

Situace pro všechny šestnáctkové číslice je uvedena formou obr. 2.15 (kód pro zobrazení je uveden vždy pod číslicí).



Obr. 2.15 Číslice 0 až F a kódy pro jejich zobrazení

Účelem příkladu **PROG_02** je vyzkoušet funkci přípravku **M7LED** tak, že na něm opakovaně zobrazujeme číslice 0 až F.

Převod zobrazované číslice na odpovídající kombinaci řídicích signálu 7segmentovky (kód dle obr. 2.15) bude nejsnazší provést pomocí pole. Opět se bude jednat o pole 8bitových čísel bez znaménka, zvolíme tedy typ **uint8_t**. Deklaraci lze zapsat takto:

```
uint8_t tab7seg[16]=
  {0b00000011, 0b10011111, 0b00100101, 0b10011001, //atd.
```

Pro úsporu datové paměti (takové pole zabere 16 bajtů, kapacita datové paměti je jen 224 bajtů) je vhodné použít modifikátor **const**. Tento modifikátor určuje, že pole je určeno pouze pro čtení (to nám nevadí, protože jednotlivé kódy potřebuje z pole pouze „vytáhnout“, není třeba je přepisovat) a hlavně, že bude uloženo ne do datové paměti, ale do paměti programu! Programová paměť má podstatně větší kapacitu.

Jak převod číslice na kód funguje? Pokud zapíšeme **tab7seg[0]** je výsledkem hodnota 0b00000011, podobně pro **tab7seg[2]** dostáváme kód 0b00100101, atd.. Kód pro zobrazení číslice tedy „vytahujeme“ z pole **tab7seg** pomocí indexu, který odpovídá zobrazované číslici. Pole provede „transformaci“ číslice na kód pro její zobrazení.

Opakované zobrazování číslic 0 až F docílíme pomocí cyklu, v tomto případě je vhodné použít cyklus **for**. Pomocí řídicí proměnné **index** (mění se od 0 do 15) měníme číslici pro zobrazení, kterou polem **tab7seg** převedeme na odpovídající kód a odešleme na port B. Po zobrazení číslice je třeba vložit krátké zpoždění, opět využijeme funkci **__delay_ms**:

```
for(index=0;index<=15;index++) //meni index od 0 do 15
{
  PORTB=tab7seg[index];
  __delay_ms(500); //cekani
}
```

Celý program je vypsán níže. Přípravek **M7SEG** musí být připojen na port B.

PROG_02:

```
#pragma config FOSC = INTOSCIO  //vnitrni osc., RA6 a RA7 volne
#pragma config WDTE = OFF       //vypnuti WDT
#pragma config LVP = OFF        //vypnuti LVP, RB4 volny

#define _XTAL_FREQ 4000000UL    //taktovaci kmitocet 4 MHz

#include <xc.h>
#include <stdint.h>

void main(void)
{
  const uint8_t tab7seg[16]=
  {0b00000011,  //0
   0b10011111,  //1
   0b00100101,  //2
   0b00001101,  //3
   0b10011001,  //4
   0b01001001,  //5
```

```
   0b01000001,  //6
   0b00011111,  //7
   0b00000001,  //8
   0b00001001,  //9
   0b00010001,  //A
   0b11000001,  //b
   0b01100011,  //c
   0b10000101,  //d
   0b01100001,  //E
   0b01110001}; //F

  uint8_t index;

  TRISB=0;  //vsechny vyvody portu B jsou vystupy

  while(1)  //nekonecna smycka
  {
    for(index=0;index<=15;index++) //meni index od 0 do 15
    {
      PORTB=tab7seg[index];
      __delay_ms(500); //cekani
    }
  }

  return;
}
```