Středoškolská technika 2015

Setkání a prezentace prací středoškolských studentů na ČVUT

MODEL STARTOVACÍCH HODIN

Tomáš Pícha

SPŠ A VOŠ PÍSEK
Karla Čapka 402, 397 11 Písek

Anotace:

Práce je o realizaci modelu startovacích hodin. Jedná se o konstrukci a oživení desky plošného spoje s pěti 7segmentovými displeji. Displeje načítají od 000:00 ms do 999:99 ms. Díky využití mikrořadiče je činnost časomíry velmi přesná. K obsluze displejů je využíváno pěti posuvných registrů. Na registry jsou sériově přiváděny data z mikrořadiče a registr je poté paralelně vyšle na displeje. Načítání časomíry je ovládáno pomocí tlačítek. Obvod je napájen napětím +5 V. Zapojení může sloužit k měření nebo stopování času.

Klíčová slova:

Mikrořadič; Posuvný registr; Segment; Displej; Časomíra; Deska plošného spoje (DPS); Sériový přenos; Paralelní přenos

Annotation:

Implementation of the model countdown. This is a structure and a recovery in the printed circuit board with five 7-segment displays. Displays load from 000:00 ms to 999:99 ms. Thanks to the use of a microcontroller timekeeping activities are very accurate. Five shift registers are normally used to control the displays. Data are serially fed on the registers from the microcontroller and then register sends this data in parallel on display. Loading of the timer is controlled by buttons. The circuit is powered by +5 V. Basically this device can be used for measuring or time tracking .

Keywords:

Microcontroller; Second shift register; Segments; Display; Timing; The printed circuit board (PCB); Serial transfer; Parallel transfer

Obsah

1 Úvod

Důvodem vytvoření mé maturitní práce na toto téma, byl požadavek od obce, k vytvoření časomíry s obsluhou (terče, spínač) pro požární sport. Pořízení takovéto časomíry od specializovaných obchodů je cenově náročné. Pokud budu konstruovat časomíru osobně, její cena bude mnohem přijatelnější. Z důvodu velikosti, kterou by časomíra a terče musely mít, jsem se rozhodl, že vyrobím malý přenosný model, pouze pro prezentaci funkce obvodu.

Ve skutečné podobě by časomíra měla mít pět 7segmentových displejů, načítajících po sepnutí startovacího tlačítka od 000:00 ms do 999:99 ms neboť po dvou minutách je vyhlášena diskvalifikace týmu, proto tedy další displeje nejsou potřeba. Časomíra by měla být spouštěna startovacím tlačítkem a po sestřelení obou terčů se sepnou kontakty, které zastaví časomíru se zobrazeným výsledným časem. Pokud časomíru nikdo nezastaví, tedy terče nebudou sestřeleny, časomíra bude načítat až do 999:99 ms a poté se vynuluje. Obvod časomíry bude napájen napětím 12 V stejnosměrného napětí. Samotný displej časomíry bude nejspíše tvořen segmenty poskládanými z LED diod, nebo diodovými pásky, tak aby byl vidět na vzdálenost alespoň 60 metrů.

Samotný vytvořený model již nevyužiji, pro mě je důležitý především program vytvořený pro obvod a proto jsem se rozhodl, že model přenechám po dokončení práce škole.

Model pro školu je vytvořen tak, aby znovu načítal od nuly po přeplnění. V podmínkách hasičské soutěže je to, že dojde k přeplnění nepravděpodobné, neboť průměrné časy týmů se pohybují okolo 30 sekund. Ovšem kdyby došlo k přeplnění, není vhodné obvod znovu spouštět. Proto můj osobní obvod bude po přeplnění opět čekat na stisknutí startovacího tlačítka. Je přidáno resetovací tlačítko, které umožňuje obvod vynulovat i bez čekání na přeplnění časomíry.

Pro vytvoření obvodu jsem si vybral řešení s využitím mikrořadiče AT89S51 řídící chod displejů. Toto řešení jsem zvolil proto, neboť ze školy vím, jak programovat mikropočítač řady 8051.

Program pro mikropočítač je napsán v jazyku symbolických adres. Obvod je realizován na desce plošného spoje, proto jsem vytvořil schéma a návrh plošného spoje v programu Eagle.

2 Mikrořadič

Jádrem celého obvodu je mikrořadič AT89S51. Pomocí tohoto mikropočítače je řízena veškerá činnost časomíry. AT89S51 jsem vybral proto neboť je velmi podobný mikrořadičům z řady 8051, se kterými mám zkušenosti ze školy.

Mikrořadič nebo mikrokontrolér je název pro jednočipový (monolitický) mikropočítač. To znamená, že všechny své části má spojeny na jednom čipu. Základní části mikrořadiče jsou mikroprocesor, paměť a V/V obvody (Vstupní/výstupní obvody).

2.1 Mikroprocesor

Mikroprocesor (CPU - central processing unit) – Mikroprocesor řídí činnost celého mikrořadiče.

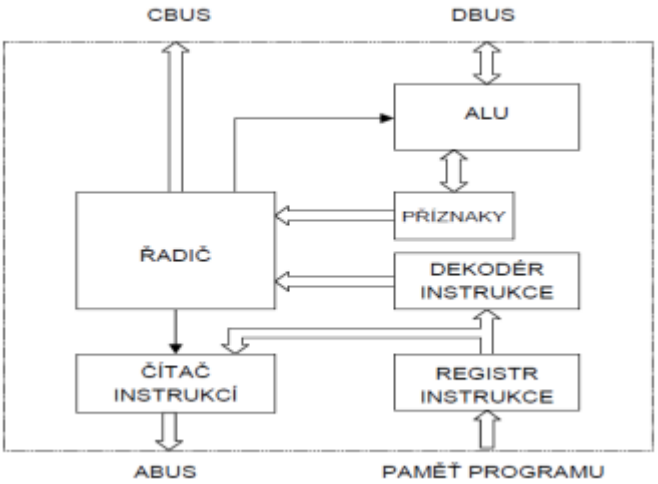
Mikroprocesor zpracovává instrukce programu. Pracuje tak, že načte instrukci z paměti programu (ROM), provede instrukci a načte další. Mikroprocesor je synchronní obvod a pro svou činnost využívá hodinového taktování hodinovými impulsy.

Základní části mikroprocesoru:

- aritmeticko-logická jednotka (ALU)

- registry (univerzální, speciální)

- řídicí obvody (řadič)



Obr. 1 Struktura mikroprocesoru [2]

Činnost mikroprocesoru spočívá v postupném čtení jednotlivých instrukcí z paměti, jejich dekódování a provádění jimi určené činnosti.

Každá instrukce se skládá z binárního kódu. Každá instrukce má svůj **operační znak**, což je specifická kombinace logických hodnot. K operačnímu znaku instrukce se mohou přidat data a adresa pokud to instrukce potřebuje pro svou činnost. Binární kód, ve kterém je instrukce uložena nazýváme **strojovým kódem**.

ALU

ALU provádí tyto operace:

- aritmetické (součet a rozdíl, příp. též součin a podíl)
- logické (součin, součet, negaci)
- posunové (posuny a rotace)

Registry:

Univerzální registry

Mají univerzální funkci. Pokud máme mikrořadič s harvardskou koncepcí je umístění registrů přímo v paměti RWM. Díky tomuto uložení je k nim snadný přístup. Můžeme do nich tedy uložit právě potřebná data a snadno s nimi v programu manipulovat.

Speciální registry

Speciální registry mají vyhrazenu zvláštní funkci, některé z nich mohou však sloužit i jako univerzální. Jedná se o různé řídící registry jako například registr TMOD, který ovládá činnost časovače. Speciální registry jako zásobník a čítač instrukcí jsou dále popsány, neboť bez nich by nebyl možný chod programu.

Mezi speciální registry patří:

Zásobník (zásobníková paměť)

Zásobník má strukturu typu LIFO (Last In – First Out), to znamená, že to co uložíme jako první, bude vybíráno jako poslední. Zásobník se využívá k automatickému ukládání návratových adres při volání podprogramů, proto aby čítač instrukcí věděl na jakou adresu se vracet při ukončení podprogramu. (Podprogram je část programu, která stojí mimo hlavní program a v průběhu programu se do něj dá vstupovat a po jeho provedení, se z něj opět vrátit).

Čítač instrukcí PC (program counter)

Čítač instrukcí PC je registr, v kterém se nachází adresa instrukce, která se bude právě vykonávat. PC zvýší svůj obsah vždy o jedna po načtení každé instrukce, aby odkazoval na následující instrukci. V době kdy se provádí načtená instrukce je, již v PC uložena adresa další instrukce.

Skoky v programu:

Cykly se v programu provádějí pomocí skoků. Skoků se dá docílit tím, že přepíšeme obsah PC tak, aby odkazoval na adresu instrukce, na kterou má program skákat. Instrukcí skoku je hned několik, například: SJMP, DJNZ, CJNE a další. Skoky také umožňují větvení programu.

Řadič

Řa*dič* řídí čítač instrukcí.

Řadič může být buď typu RISC (mikroprogramový automat), nebo CISC (pevně zapojená logika s menším počtem instrukcí)

2.2 Paměť

Paměť (operační, hlavní) slouží k uložení programu a dat.

Paměť ROM (read only memory) je paměť určená pouze ke čtení, informace je v ní uložená pevně, uchová se i po vypnutí napájení. Do této paměti se ukládá vytvořený program.

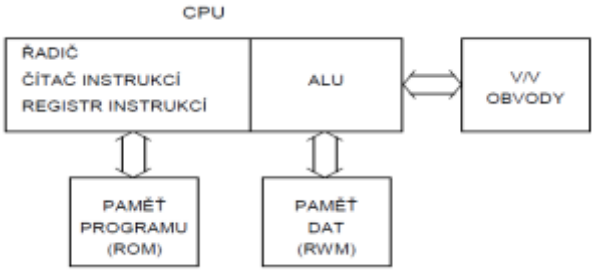
Paměť RWM (read write memory) je paměť, do které může mikroprocesor informaci též zapisovat, informace se však ztrácí po vypnutí napájení. Tato paměť obsahuje univerzální i speciální registry.

U mikrořadičů je nejobvyklejší architektura paměti Harvardská. Používá se ještě architektura Von Neumanova, ovšem ta není tak vhodná pro mikrořadiče.

Harvardská koncepce počítače:

- oddělená paměť pro program a pro data, pro adresování každé paměti se používá jiná množina adres. Obě množiny začínají adresou nula, maximální adresy jsou obecně různé, paměťové prostory jsou různě rozsáhlé.

- pro přenos dat a instrukcí slouží samostatné sběrnice (datová a instrukční). Takové uspořádání vede k rychlejší práci počítače, protože mohou být současně přenášena data i instrukce mezi pamětí a mikroprocesorem. [1]



Obr. 2 Struktura mikrořadiče [2]

2.3 Vstupně/výstupní obvody

V/V obvody umožňují komunikaci mikropočítače s okolím. Obsahují určitý počet přípojných míst, která se nazývají ***brány (porty).***

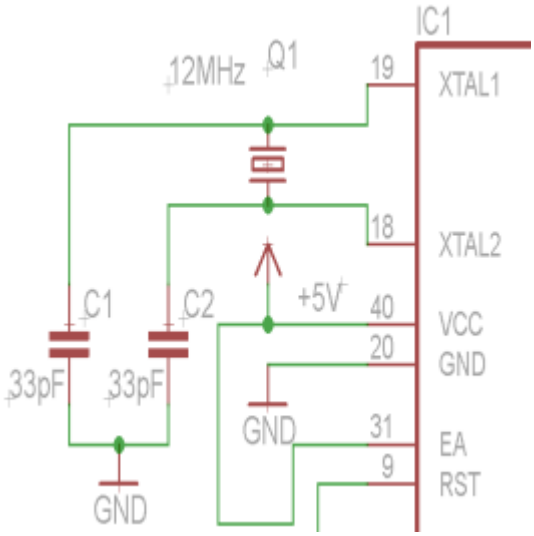
AT89S51 má celkem čtyři osmibitové brány P0 – P3, všechny jsou bitově adresovatelné. Po spuštění mikrořadiče jsou piny bran nastaveny na log. 1. Brány tvoří 32 z celkových 40 vývodů mikrořadiče. Pro možnost připojení periferní časomíry, jsou v obvodu na výstupech mikrořadiče přidělány svorkovnice, na které se dá periferní časomíra připojit.

Všechny použité V/V porty jsou popsány v následující tabulce:

Tab. 1 Vstupy a výstupy:

Číslo portu	Popis	Pojmenování	
14 (P3.4)	RESET	TL_RESET	Tlačítko RESET, které vyvolá programový reset. (nikoliv hardwarový)
13(P3.3)	START	TL_START	Tlačítko START, které odstartuje načítání časomíry
12(P3.2)	STOP	TL_STOP	Tlačítko STOP zastaví časomíru.
15(P3.5)	SWITCH1	TL_SWITCH1	Přepínač1
16(P3.6)	SWITCH2	TL_SWITCH2	Přepínač2
26(P2.5)	CLK	OUT_CLK	Výstup signálu CLK potřebného pro posuvné registry.
25(P2.4)	RST	OUT_RST	Ovládání resetu posuvných registrů.
39(P0.0)	DATA	OUT_DATA	Výstup dat, která přichází na posuvné registry.
27(P2.6)	Dvojtečka	OUT_DVOJT	Výstup pro dvojtečku.
21(P2.0)	LED1	OUT_LED1	Zelená led dioda.
22(P2.1)	LED2	OUT_LED2	Žlutá led dioda.
23(P2.2)	LED3	OUT_LED3	Červená led dioda.
6(P1.5)	MOSI		Vstup signálu MOSI - sériový zápis do pamětí
7(P1.6)	MISO		Vstup signálu MISO – sériové čtení z pamětí
8(P1.7)	SCK		Vstup SCK – taktovací signál při programování mikrořadiče
9	Reset		Při poklesu napájecího napětí na tomto portu se generuje reset.
18	Xtal1		Vstup oscilátoru.
19	Xtal2		Vstup oscilátoru.
20	GND		Uzemnění mikrořadiče.
31	EA		připojíme na napájecí napětí, abychom docílili log. 1 na tomto vstupu. Mikrořadič při logické 0 na EA čte pouze z vnitřní paměti dat.
40	VCC		Napájení mikrořadiče.

2.4 Časování CPU



Ke generování impulsů se používá krystalický oscilátor s frekvencí 11, 059 MHz připojený na vstupy Xtal1 a Xtal2. Impulsy řídí veškerou činnost mikrořadiče.

Strojový cyklu se skládá z jednotlivých taktů generovaných z impulsů oscilátoru. Instrukční cyklus je doba potřebná pro vykonání instrukce a skládá se ze strojových cyklů. Instrukční cyklus mohou mít jednotlivé instrukce různě dlouhý. Pro vykonání instrukce (jednobajtové až tříbajtové) jsou většinou potřebné jeden až dva strojové cykly.

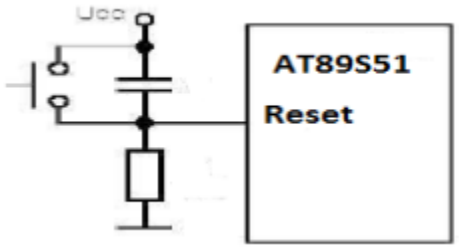
Obr. 3 Krystalický oscilátor připojený k mikrořadiči

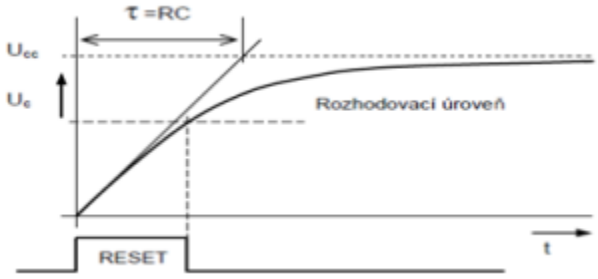
2.5 Reset CPU

Reset mikroprocesoru:

Resetem uvedeme mikrořadič do výchozího stavu:

čítač instrukcí se nastaví na počáteční adresu, zakáže se přerušení, ostatní registry jsou resetem ovlivněny různě podle typu mikrořadiče, výstupní signály se uvedou do definovaného stavu, brány se nastaví na log. 1 [2]





Obr. 4 a) RC obvod [3] b) Průběh napětí na kondenzátoru [2]

Generování signálu RESET:

Ke generování signálu RESET je připojen RC článek. Pokud je kondenzátor nabit na dostatečnou úroveň, RESET není aktivní. Pokud dojde k poklesu pod rozhodovací úroveň, generuje se hardwarový RESET. Doba opětovného nabití závisí na konstantě T = R*C.

Reset se generuje vždy po zapnutí napájení, aby se mikrořadič nastavil do výchozích hodnot. Reset je nutné generovat také vždy při poklesu napájecího napětí, neboť mikrořadiči by se mohlo stát, že začne provádět chybné instrukce z důvodu malého napájecího napětí.

Reset můžeme vyvolat i ručně, k tomuto účelu je v obvodu tlačítko, které zkratuje kondenzátor a tím přivede nulové napětí na vstup reset. V obvodu je i tlačítko (port P3.4), které vykonává programový reset. Při jeho stisknutí program skočí do jiného cyklu, ale nedochází k hardwarovému resetu celého mikrořadiče.

2.6 Čítače/časovače

Mikrořadič AT89S51 obsahuje standardně dva šestnáctibitové čítače čítající od předvolby směrem nahoru. Při přeplnění generují žádost o přerušení. Čítače jsou programově dostupné.

časovač - čítá vnitřní signál o frekvenci *fosc*/12, tj. inkrementuje během každého strojového cyklu. Doba čítání od vložené předvolby *P* do přeplnění je:

$$T_č = \frac{12}{f_{osc}} \cdot (2^n - P)$$

kde *n* je počet bitů čítače (podle režimu 16, 13 nebo 8).[4]

čítač - čítá vnější impulsy

Čítače/časovače jsou programovatelné, mohou pracovat v jednom ze čtyř pracovních režimů.

Mají ***dva řídicí registry, TMOD a TCON.***

***TCON** - registr řízení čítače/časovače (adresa 88H, po resetu vynulován):

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Obr. 5 Rozložení registru TCON [4]

TR0, 1 - spuštění (=1) / zastavení (=0) čítače 0,1. TF0, 1 - příznak přeplnění čítače 0,1 (po překročení maximální hodnoty čítače se příznak nastaví na 1 a současně se generuje ***žádost o přerušení***)

TMOD - registr nastavení režimů čítače/časovače (adresa 89H, po resetu vynulován):

7	6	5	4	3	2	1	0
GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
čítač 1				čítač 0			

Obr. 6 Rozložení registru TMOD [4]

GATE - umožňuje blokovat čítač vnějším signálem na vstupech INT0 , INT1

C/T - vybírá činnost čítače nebo časovače:

C/T = 0 - časovač

C/T = 1 - čítač

M1, M0 - nastavení režimu

V své práci využívám režim 1 jako šestnácti bitový časovač.

Jakmile hodnota v registru čítače přechází ze samých jedniček (tj. z FFFFH) na samé nuly, nastaví se příznak TFx v registru TCON a je generována žádost o přerušení.[4]

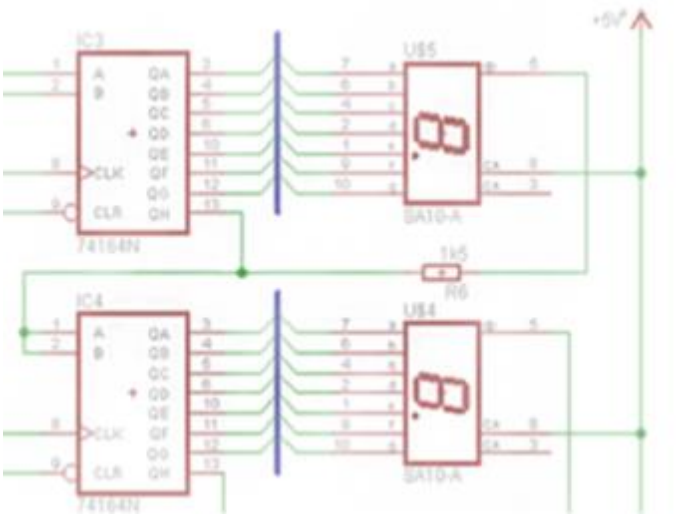
Časovač kontroluje, jestli již uběhlo 10ms a pak pokračuje v programu. Tím docílíme velmi přesného fungování časomíry (dochází ke zpoždění asi 10us při obsluze časovače), než kdybychom zpoždění řešili pomocí programového cyklení. Pokud chceme vložit předvolbu, uložíme nejprve spodní bajt předvolby do TL0 a poté horní bajt předvolby do TH0. Předvolbu je vhodné zavádět na začátku programu jako konstantu. Časovač načítá, dokud nedojde k přeplnění. Přeplnění indikujeme log. 1 v bitu TF0. Poté bit vynulujeme, znovu uložíme předvolbu a pokračujeme v programu.

3 Periférie

3.1 Posuvné registry

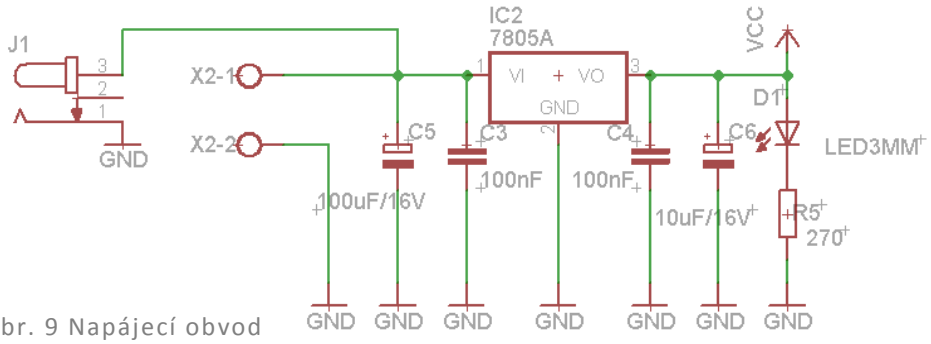
Abychom mohli ovládat displeje paralelně přímo z mikrořadiče, potřebovali bychom mnohem více portů, než máme k dispozici (5x7=35 vývodů jen pro displeje a je třeba počítat i s ovládacími tlačítky). Pro ovládání displejů jsem tedy zapojil do obvodu pět posuvných registrů typu 74164N, které stačí ovládat dvěma vodiči z mikrořadiče. Prvním vodičem vedeme data, které chceme zobrazit a druhým vysíláme taktovací signál CLK. Dále můžeme použít vstup pro reset, který je aktivní log. 0, ovšem v programu se neuplatní. Na svém výstupu má registr osm vývodů, kterými lze ovládat displej. Z osmého vývodu lze posílat data na další registr a tým vytvořit kaskádu registrů pro ovládání více displejů.

Činnost posuvného registru spočívá v tom, že na vstup A a B (na prvním a druhém portu posuvného registru) přivedeme sériově data, která chceme zobrazit na výstupu. Data se zapisují do posuvného registru sestupnou hranou taktovacího signálu na vstupu CLK. Data se přesouvají z prvního registru až do posledního a jsou neustále posílána na výstupy registrů. Ve chvíli kdy přestaneme vysílat signál CLK, registry zobrazují aktuální stav, dokud nedojde opět k vysílání CLK nebo resetu registrů. Možné zapojení registrů je na obrázku číslo 8.



Obr. 8 Zapojení posuvných registrů

3.2 Napájecí obvod



Obr. 9 Napájecí obvod

K napájení mikrořadiče je potřeba stabilizovaných +5 V. Napájecí obvod, který je na desce plošného spoje využívá stabilizátor napětí 7805A, který má na svém výstupu stabilizované napětí +5 V. Pro indikaci toho, že obvod je pod napětím, je připojena LED dioda D1. K připojení vnějšího zdroje napětí může být použita jak svorka pro dva vodiče, tak napájecí konektor jack.

3.3 Displeje

K zobrazení čísel slouží 7segmentové displeje o velikosti 34 mm. Displeje mají společnou anodu a na každý segment připadají dvě led diody, proto není potřeba zapojovat ochranné rezistory. Displeje

jsou připojeny k posuvným registrům, tak aby konstrukce desky plošného spoje byla co nejjednodušší. Zapojení displejů je na obr. 8.

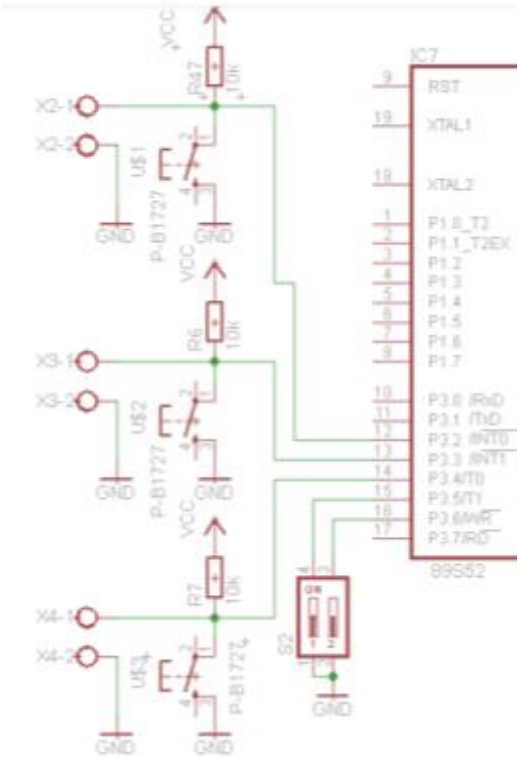
3.4 Ovládání

K ovládání mikrořadiče jsou využity tři tlačítka RESET, START a STOP, které po sepnutí spojí port mikrořadiče se zemí a mikrořadič na to patřičně reaguje. K dispozici jsou ještě dva přepínače, které původně měly stejnou funkci jako STOP tlačítko, ale pro školní potřeby nejsou využívány.

V obvodu jsou k dispozici ještě tři LED diody (červená, žlutá, zelená), které slouží jako indikační diody.

Červená dioda se aktivuje po přeplnění časomíry, aby bylo znatelné, že časomíra načítá opět od nuly. Zelená dioda se zapne po stisknutí STOP tlačítka, tedy zastavení časomíry. Je možné připojit i extérní tlačítka na svorky, které jsou k dispozici na desce plošného spoje.

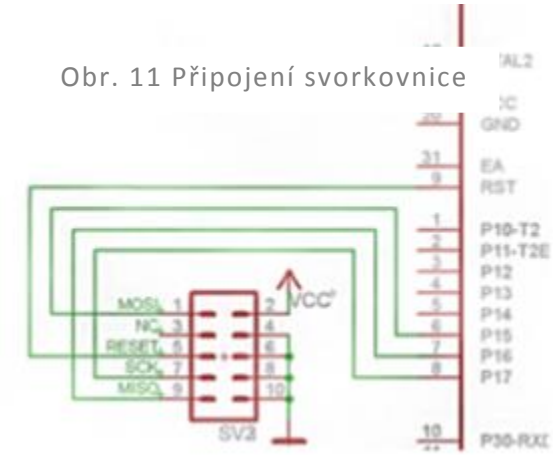
Rezistory připojené vůči napájecímu napětí slouží jako ochranné.



3.5 Programování mikrořadiče

K programování vnitřní paměti Flash se využívá šesti vodičů. RESET procesoru, vodiče pro sériový zápis (MOSI) a čtení (MISO) obsahu interních pamětí FLASH + jeden vodič pro synchronizaci přenosu dat (SCK) a vodiče pro napájecí napětí. Pro toto připojení je v obvodu k dispozici svorkovnice.

Obr. 11 Připojení svorkovnice



4 Program

K naprogramování mikrořadiče jsem zvolil Jazyk symbolických adres (assembly language). Tento jazyk jsem si vybral na základě svých zkušeností ze školy. Díky přímým instrukcím a prací přímo s paměťovými místy mikrořadiče, je vhodný pro programování takto jednoduchým činností. Souvislý zdrojový kód je uveden v přílohách.

4.1 Začátek programu a pojmenování

Na začátku programu musíme uložit do čítače instrukcí počáteční adresu, tedy nulu, aby začal program načítat od začátku. Používáme k tomu instrukci ORG

```
        ORG     00H
```

Poté přejdeme k pojmenování paměťových míst, adres a konstant, které budeme v programu používat. Tento krok není nezbytně nutný pro chod programu, ale velice usnadňuje práci programátorovi, neboť vždy stačí provést změnu zde a nemusíme přepisovat všechny instrukce, kterých se změna týká.

```
TL_RESET        BIT P3.4
TL_START        BIT P3.3
TL_STOP         BIT P3.2
TL_SWITCH1      BIT P3.5
TL_SWITCH2      BIT P3.6
OUT_CLK         BIT P2.5
OUT_DATA        BIT P0.0
OUT_RST         BIT P2.4
OUT_DVOJT       BIT P2.6
OUT_LED1        BIT P2.0
OUT_LED2        BIT P2.1
OUT_LED3        BIT P2.2
PV              EQU 56330
```

4.2 Počáteční nastavení stavu registrů a čekání na start

Přejdeme k tělu samotného programu. Návěští START nám označuje začátek programu a umožňuje nám se v průběhu programu na něj vracet. Instrukcí MOV TMOD,#00000001B nastavíme vnitřní čítač mikrořadiče do režimu 1 tedy jako šestnácti bitový časovač. Bit TF0 nás informuje, jestli nedošlo k přeplnění čítače a není nutní provést obsluhu přerušení. Bit TF0 vynulujeme, abychom mohli spustit čítání od začátku. Instrukce **MOV** TL0**,#LOW** PV a **MOV** TH0**,#HIGH** PV zapíší předvolbu, kterou jsme si na začátku pojmenovali jako PV, nejprve do dolního bajtu poté do horního bajtu. Časovač spustíme nastavením bitu TR0. Příkaz **ACALL** NULUJ skáče na obsluhu podprogramu pro vyčištění paměti. **MOV** R0**,**#20D vloží do registru R0 adresu paměťového místa, kam se budou ukládat hodnoty zobrazovaná prvním displejem.

```
START:      MOV     TMOD,#00000001B
            CLR     TF0
            MOV     TL0,#LOW  PV
            MOV     TH0,#HIGH PV
            SETB    TR0
            ACALL   NULUJ
            MOV     R0,#20D
```

V této části program čeká na stisknutí START tlačítka. **ACALL** LED volá obsluhu podprogramu pro dvojtečky. **ACALL** NASTAV ukládá stavy jednotlivých segmentů do paměti. **ACALL** CLK vysílá stavy segmentů sériově na posuvné registry, kde dojde k jejich zobrazení na displejích, momentálně tedy 000:00. **ACALL** ZPOZD vyvolává zpoždění 10 ms, aby časomíra správně zobrazovala. Zpoždění v této části programu není tak důležité, ovšem je spojeno s ovládáním stavu dvojtečky. **JNB** TL_START,NACTI kontroluje stav portu, na kterém je připojeno START tlačítko. Jestli není stisknuto a pokud je skočí na další část programu, jinak instrukce **SJMP** CEKEJ vrací zpět na návěští CEKEJ.

```
CEKEJ:      ACALL   LED
            ACALL   NASTAV
            ACALL   CLK
            ACALL   ZPOZD
            JNB     TL_START,NACTI
            SJMP    CEKEJ
```

4.3 Načítání

Pokud bylo stisknuto START tlačítko, dojde k obsluze části programu, kde už časomíra načítá. Nejprve testujeme, jestli není stisknuto tlačítko RESET nebo STOP. Poté proběhne opět obsluha displejů voláním podprogramů NASTAV, CLK, ZPOZD. Instrukcí INC @R0 zvětšíme hodnotu, která je uložena na adrese na kterou odkazuje R0. Tedy na začátku jsme do R0 uložili adresu 20D a nyní zvýšíme hodnotu na adrese 20D z 0 na 1. **CJNE** @R0,#0AH,NACTI je testování a skok při nerovnosti, takže otestuje, jestli není překročeno maximální číslo 9, pokud ano, pokračuje v programu, jinak skáče na návěští NACTI, aby se načítalo dále. Pokud dojde k překročení, musí se paměťové místo vynulovat a přesouváme se na další displej, tím že inkrementujeme R0, z adresy 20D na 21D a skočíme na obsluhu druhého displeje.

```
NACTI:      JNB     TL_STOP,KONEC
            JNB     TL_RESET,START
            ACALL   NASTAV
            ACALL   CLK
            ACALL   ZPOZD
            INC     @R0
            CJNE    @R0,#0AH,NACTI
            MOV     @R0,#00H
            INC     R0
            SJMP    DIS2
```

4.4 Ukončení načítání

Tato část programu se vykonává, pokud dojde ke stisknutí STOP tlačítka. Je velmi podobná počátečnímu stavu programu, kdy se čeká na tlačítko START, ovšem zde se nezobrazují na displejích nuly, ale výsledný čas který zobrazovala časomíra ve chvíli zmáčknutí STOP tlačítka. Zastavení časomíry nám indikuje zelená LED dioda. Pokud dojde ke stisknutí RESET tlačítka, časomíra se vrací na návěští START, kde se vynuluje a opět čeká na START tlačítko.

```
KONEC:    CLR     OUT_LED1
          ACALL   CLK
          ACALL   ZPOZD
          ACALL   NASTAV
          JNB     TL_RESET,START
          SJMP    KONEC
```

4.5 Obsluha dalších displejů

Obsluha druhého a následujících displejů je totožná s obsluhou toho prvního. Pokud dojde k přeplnění i pátého displeje program skočí na návěští START a dojde k vynulování a časomíra začne načítat od začátku. Celá část je v přílohách.

```
DIS2:     INC     @R0
          CJNE    @R0,#0AH,NACTI
          MOV     @R0,#00H
          INC     R0
```

4.6 Uložení segmentů do paměti

Přesouváme se k podprogramu ukládání stavů segmentů do paměti. Na začátku opět uložíme do R0 adresu 20D , neboť se v průběhu programu různě mění.

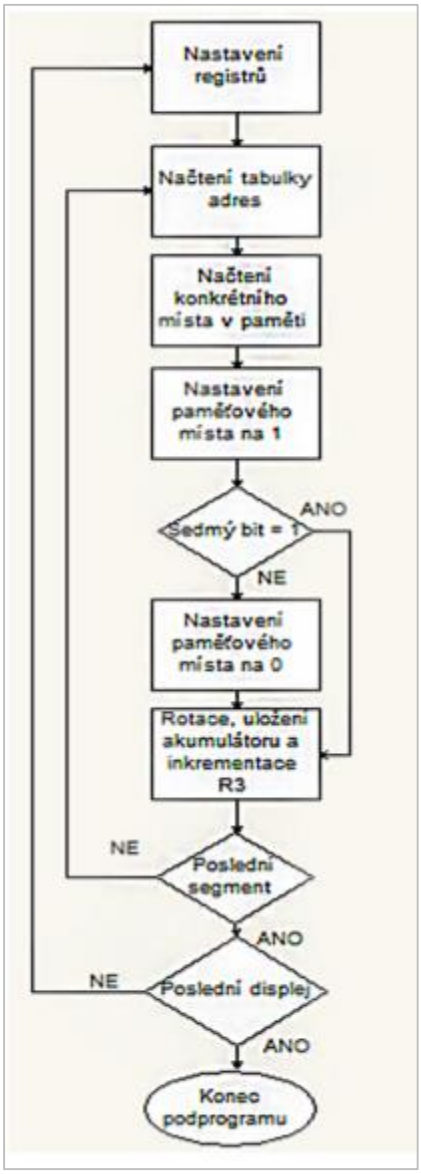
```
NASTAV:   MOV     R0,#20D
```

Zkontrolujeme podprogram pro stav dvojtečky. Dále provádíme načítání dat z tabulky. V tabulce TAB jsou uloženy binární kombinace, které se vysílají na posuvné registry. Uložíme do akumulátoru hodnotu, která má být zobrazena. Například pokud jsme na prvním displeji a chceme zobrazit číslo 1, v registru R0 musí být uložena adresa 20D a v ní hodnota 01. Instrukcí **MOV** DPTR**,**#TAB vložíme do šestnácti bitového registru DBTR počáteční adresu tabulky. Pomocí **MOVC** A**,**@A+DPTR přičteme k počáteční adrese tabulky hodnotu zobrazované číslice a data, která jsou na tomto místě uložíme zpět do akumulátoru. Tím se do akumulátoru uloží právě bitová kombinace pro daný displej. Tuto kombinace si uložíme do pomocného registru B, abychom si jí později nepřepsali. Registr R3 nám slouží jako ukazatel na segment v displeji. Před obsluhou segmentů jej vynulujeme.

```
NEXTDIS:  ACALL   LED
          MOV     A,@R0
          MOV     DPTR,#TAB
          MOVC    A,@A+DPTR
          MOV     B,A
```

```
        MOV     R3,#00H
```

Obsluha segmentů spočívá v uložení stavů jednotlivých segmentů do paměti. Registr R4 slouží jako ukazatel na místo v tabulce adres TABADR, která obsahuje počáteční adresy, od kterých se segmenty jednotlivých displejů mají ukládat. Provedeme stejnou operace načítání z tabulky jako při načítání binární kombinace pro displej, ale nyní se do akumulátoru uloží právě počáteční adresa pro segmenty. K této adrese se pomocí **ADD** A,R3 přičte obsah R3, aby akumulátor odkazoval na paměťové místo, kam má být uložen stav segmentu. Obsah akumulátoru přesuneme do registru R1, který slouží k testování toho, jestli má svítit nebo ne. Paměťové místo nastavíme na logická 1, tedy 0FFH, přesuneme do akumulátoru bitovou kombinace při displej z registru B a testujeme, zda na sedmém bitu je log. 1, pokud ano, necháme paměťové místo nastaveno na 0FFH a pokračujeme na další segment. Pokud ne, uložíme do paměťového místa log. 0 a teprve poté pokračujeme na další segment.



Obr. 12 Diagram nastavení paměťových míst

```
NEXTSEG:
        MOV     A,R4
        MOV     DPTR,#TABADR
        MOVC    A,@A+DPTR
        ADD     A,R3
        MOV     R1,A
        MOV     @R1,#0FFH
        MOV     A,B
        JB      ACC.7, POSUN
        MOV     @R1,#00H
        MOV     R1,#00H
```

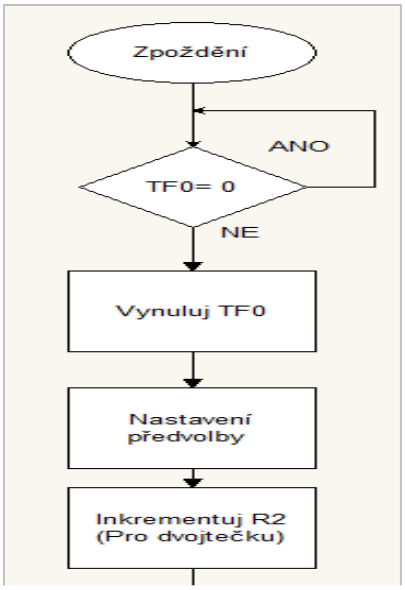
Před tím, než budeme testovat další segment, provedeme rotaci akumulátoru do leva. Tím zajistíme, že při dalším testování bude na sedmém bitu akumulátoru hodnota dalšího segmentu. Hodnotu opět uložíme do registru B a inkrementujeme R3, aby se stav dalšího segmentu uložil na další paměťové místo. Pokud jsme otestovali všech sedm segmentů v displeji, přesuneme se na další displej tím, že inkrementujeme R0. Musíme inkrementovat i R4, aby odkazoval na počáteční adresu, od které se mají ukládat stavy segmentů dalšího displeje. Pokud již R0 přesáhl adresu 24D, která je adresou pátého displeje, tedy došlo k obsloužení všech pěti displejů, vynulujeme R4 a R0 nastavíme opět na

první displej. Tento krok není tak nutný, protože na začátku nastavujeme registry tak, aby odkazovaly tam, kam mají. Toto je spíše taková pojistka. Instrukce return ukončuje podprogram.

```
POSUN:
        RL      A
        MOV     B,A
        INC     R3
        CJNE    R3,#08H,NEXTSEG
        INC     R0
        INC     R4
        CJNE    R0,#25D,NEXTDIS
        MOV     R4,#00H
        MOV     R0,#20D
        RET
```

4.7 Zpoždění

Podprogram zpoždění generuje zpoždění 10 ms, protože displej s nejnižší váhou zobrazuje právě desítky milisekund. Zpoždění se docílí pomocí časovače. Na začátku programu jsme spustili časovač. Příkaz **JNB** TF0,$ testuje, jestli již došlo k přeplnění, které se indikuje jedničkou v TF0, pokud k přeplnění nedošlo tak díky tomu že skáče na $ skáče sám na sebe a čeká, dokud k přeplnění nedojde. Po přeplnění, tedy po té co uběhne 10 ms, vynuluje příznak přeplnění a uloží znovu předvolbu do časovače, aby znovu načítal 10 ms. Ještě musíme inkrementovat registr R2. R2 slouží k ovládání stavu dvojtečky.



Obr. 13 Diagram zpoždění

```
ZPOZD:
        JNB     TF0,$
        CLR     TF0
        MOV     TL0,#LOW  PV
        MOV     TH0,#HIGH PV
        INC     R2
        RET
```

4.8 Blikání dvojtečky

Podprogram pro dvojtečku kontroluje, kolikrát inkrementoval registr R2. Pokud inkrementoval 100krát, tedy uběhla jedna sekunda, dvojtečka zhasne, pokud inkrementoval 200krát, uběhly dvě sekundy a dvojtečka se rozsvítí a vynuluje se registr R2, aby začínal od začátku.

```
LED:    CJNE    R2,#6AH,LEDOFF
        MOV     R2,#00H
        SETB    OUT_DVOJT
LEDOFF: CJNE    R2,#35H,LEDON
        CLR     OUT_DVOJT
```

```
LEDON:          RET
```

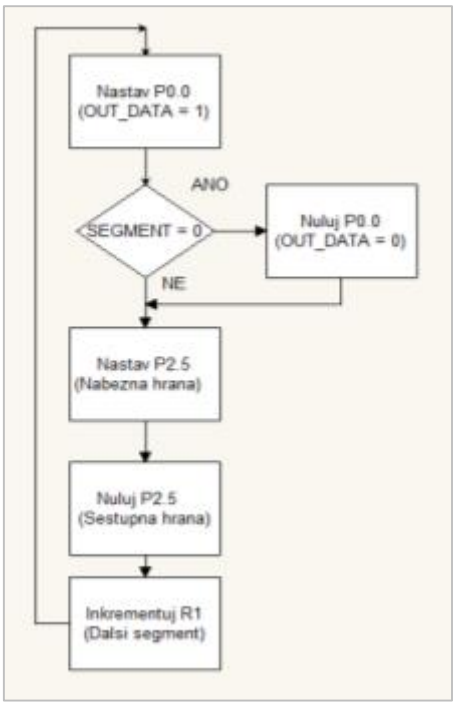
4.9 Vysílání dat

Vysílání dat probíhá pomocí podprogramu CLK. Nejprve do registru R1 zapíšeme adresu 30D, od této adresy jsou uloženy v paměti stavy jednotlivých segmentů. Poté nastavíme port pro výstup dat na log. 1 a otestujeme, jestli se v paměti na tomto místě nachází nula a pokud ne přejdeme k vysílání CLK, pokud ano, změníme stav portu na nulu a poté přejdeme k vysílání CLK.

```
CLK:      MOV     R1,#30D
NEXTCLK:  SETB    OUT_DATA
          CJNE    @R1,#00H,VYSLI
          CLR     OUT_DATA
```

Protože posuvné registry reagují na sestupnou hranu signálu CLK vytváříme na výstupním portu CLK signálu střídavě log. 1 a log. 0. To jak dlouho bude stav logické úrovně trvat, určíme pomocí instrukce NOP. NOP je prázdná instrukce, která nic nevykonává, pouze zdržuje program. Reakce posuvného registru je ovšem dostatečně rychlá proto není potřeba používat velký počet instrukcí NOP. Po vyslání jednoho paměťového místa inkrementujeme R1, abychom se přesunuli na další paměťové místo. Otestujeme, jestli již nejsou vyslány všechny segmenty všech displejů. Poslední segment je uložen na adrese 69D. Po ukončení přenosu vynulujeme R1 a vyskočíme z podprogramu.

```
VYSLI:    SETB    OUT_CLK
          NOP
          NOP
          NOP
          NOP
          CLR     OUT_CLK
          NOP
          NOP
          NOP
          NOP
          NOP
          INC     R1
          CJNE    R1,#70D,NEXTCLK
          MOV     R1,#00H
          RET
```



4.10 Nulování paměti

Nulování probíhá tak, že do R0 uložíme adresu 20D, kde jsou hodnoty pro první displej, a v každém cyklu paměťové místo vynulujeme a přesuneme se na další. Končíme na adrese 69D, kde je uložen stav posledního segmentu pátého displeje. Poté vynulujeme všechny registry.

```
NULUJ:  MOV   R0,#20D
 NUL:   MOV   @R0,#00H
        INC   R0
        CJNE  R0,#70D,NUL
        MOV   R2,#00H
        MOV   R3,#00H
        MOV   R4,#00H
        MOV   R5,#00H
        MOV   R6,#00H
        MOV   R7,#00H
        RET
```

4.11 Tabulky

V tabulce čísel TAB jsou uloženy binární hodnoty všech zobrazovaných číslic tak, aby se vždy rozsvítily správné segmenty. Segmenty se rozsvěcují, pokud je na ně přivedena log. 0, neboť mají společnou anodu. Binární kombinace je poskládaná tak, aby konstrukce spojů posuvných registrů a displejů byla co nejjednodušší.

```
TAB:    DB 10001000b
        DB 11101011b
        DB 01001100b
        DB 01001001b
        DB 00101011b
        DB 00011001b
        DB 00011000b
        DB 11001011b
        DB 00001000b
        DB 00001011b
```

Tabulka adres TABADR určuje počáteční paměťová místa, od kterých se ukládají stavy segmentů jednotlivých displejů. Načítání z této tabulky zjednodušuje program, protože se vždy uloží všech sedm segmentů a pak se jednoduše odkážeme na další místo v paměti a ukládáme znovu.

```
TABADR:
DB      30D
DB      38D
DB      46D
DB      54D
DB      62D
```

END nám ukončuje program. Říká překladači, že nemá dál překládat.

```
END
```

5 Závěr

Výsledkem mé práce je funkční model časomíry. Použité součástky jsou vybírány tak, aby byly dostačující pro činnost a zároveň cenově přijatelné. Obvod je realizován na desce plošného spoje.

Vytvořený model se chová podle reálných požadavků. Obvod se po naprogramování mikrořadiče pracuje zcela správně. Dvě LED diody o průměru 5 mm znázorňují dvojtečku oddělující řády sekund a milisekund. Obvod má jednoduché ovládání. Startovací tlačítko spouští běh časomíry. Namísto terčů je možné využít STOP tlačítko, které po stisknutí způsobí, že se časomíra zastaví na aktuálním čase a proto, aby se znovu rozběhla, je nutné jí resetovat tlačítkem pro reset. Resetovací tlačítko nám také umožňuje vynulovat časomíru, aniž bychom čekali na její přeplnění. Pro školní potřeby se načítání časomíry po přeplnění opět spustí, bez nutnosti stisknout startovací tlačítko. K dispozici jsou tři indikační LED diody. Červená dioda se aktivuje po přeplnění časomíry, aby bylo znatelné, že časomíra načítá opět od nuly. Zelená dioda se zapne po stisknutí STOP tlačítka, tedy zastavení časomíry. Je možné připojit i extérní tlačítka na svorky, které jsou k dispozici na desce plošného spoje. Přepínače, které jsou na desce, neplní momentálně žádnou funkci.

Díky využití vnitřního časovače mikrořadiče časomíra bez problémů načítá s velkou přesností od 000:00 ms do 999:99 ms s reakcí na ovládací tlačítka. Projevuje se pouze malé zpoždění při obsluze časovače, ale v časových intervalech, ve kterých časomíra načítá, je toto zpoždění zanedbatelné.

Model může být využit nejen k načítání času, ale lze jej kdykoliv přeprogramován k jinému využití díky svorkovnici, která je umístěné na desce. Využití je jen v mezích hardwarového vybavení obvodu, pokud nepoužijeme svorkovnici k připojení extérního obvodu a rozšíříme tak možnosti stávající verze. Díky posuvným registrům je možné ovládat všech 5 7segmentových displejů pouze za pomocí tří pinů mikrořadiče a to pinu pro data, CLK a dvojtečku. Pin pro reset posuvných registrů nepoužívám, i když je zapojen, proto ho neuvádím jako nutnost pro řízení posuvných registrů. Toto představuje výhodu pro připojení extérní časomíry, neboť stačí udělat připojení pouze pomocí tří řídících pinů, napájení a uzemnění.

Mikrořadič pro svou činnost potřebuje stabilizované napětí +5 V, pro tyto účely jsem vyrobil stabilizační obvod. Díky stabilizátoru bude obvod pod stálým požadovaným napětím +5 V, po přivedení napájecího napětí. K napájení můžeme použít konektor jack, nebo přímo pomocí vodičů. Zelená LED dioda umístěná v napájecím obvodu signalizuje, že obvod je pod napětím.

Vzhledem k tomu, že se časomíra zastaví v okamžiku sepnutí terčů, odpadá chyba lidského faktoru. Toto je velká výhoda pro přesné určení výsledných časů při hasičské soutěži. Zapojení je spolehlivé a díky relativně nízké výrobní ceně (v řádech stokorun, viz kalkulace v přílohách) a jednoduché

konstrukci, je přijatelným řešením na rozdíl od drahých a složitých zobrazovačů, u kterých jejich pořízení dosahuje přibližně dvou tisíc korun.

Poznatky získané při tvoření práce využiji ke konstrukci časomíry, která bude sloužit při hasičských soutěžích v naší obci. V obvodu vynechám 7segmentové displeje a nahradím je připojením externí velkou časomírou. Velká časomíra bude napájena napětím +12 V, proto musím vyrobit rozhraní, které bude například pomocí tranzistorů spínat jednotlivé segmenty v reakci na data vysílaná z mikrořadiče.

Použitá literatura

Čerpal jsem z materiálů dostupných na školním disku n/Sindelar/SPS/Mit/Teorie konkrétně z těchto souborů:

- 2.1_Zákl_pojmy
- 3.1_Jednočip_mikropoč
- 3.2_Řada_8051
- 3.3_Periférie_na_čipu

[1] použito z: n/Sindelar/SPS/Mit/Teorie soubor 2.1_Zákl_pojmy

[2] použito z: n/Sindelar/SPS/Mit/Teorie soubor 3.1_Jednočip_mikropoč

[3] použito z: n/Sindelar/SPS/Mit/Teorie soubor 3.2_Řada_8051

[4] použito z: n/Sindelar/SPS/Mit/Teorie soubor 3.3_Periférie_na_čipu

Dále jsem čerpal z:

ANTOŠOVÁ, Marcela a Vratislav DAVÍDEK. *Číslicová technika: [učebnice]*. 4., aktualiz. vyd. České Budějovice: Kopp, 2009, 305 s. ISBN 978-80-7232-394-4.

Přílohy

I. Obrázky reálné desky plošného spoje

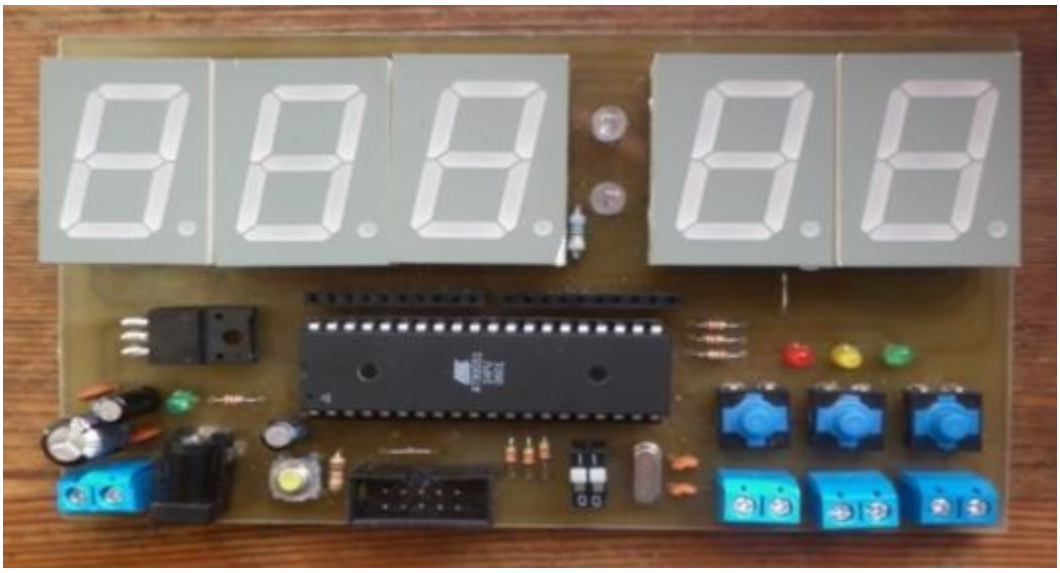
II. Schéma modelu

III. Schéma zapojení pro osobní využití

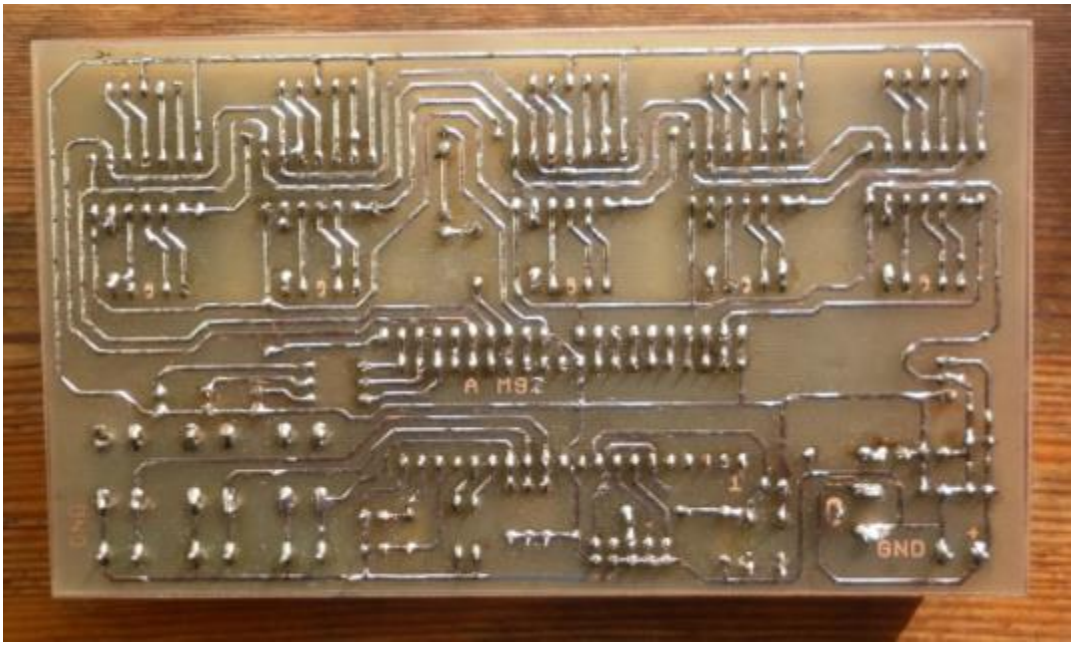
IV. Kalkulace

V. Zdrojový program

Obrázky reálné desky plošného spoje



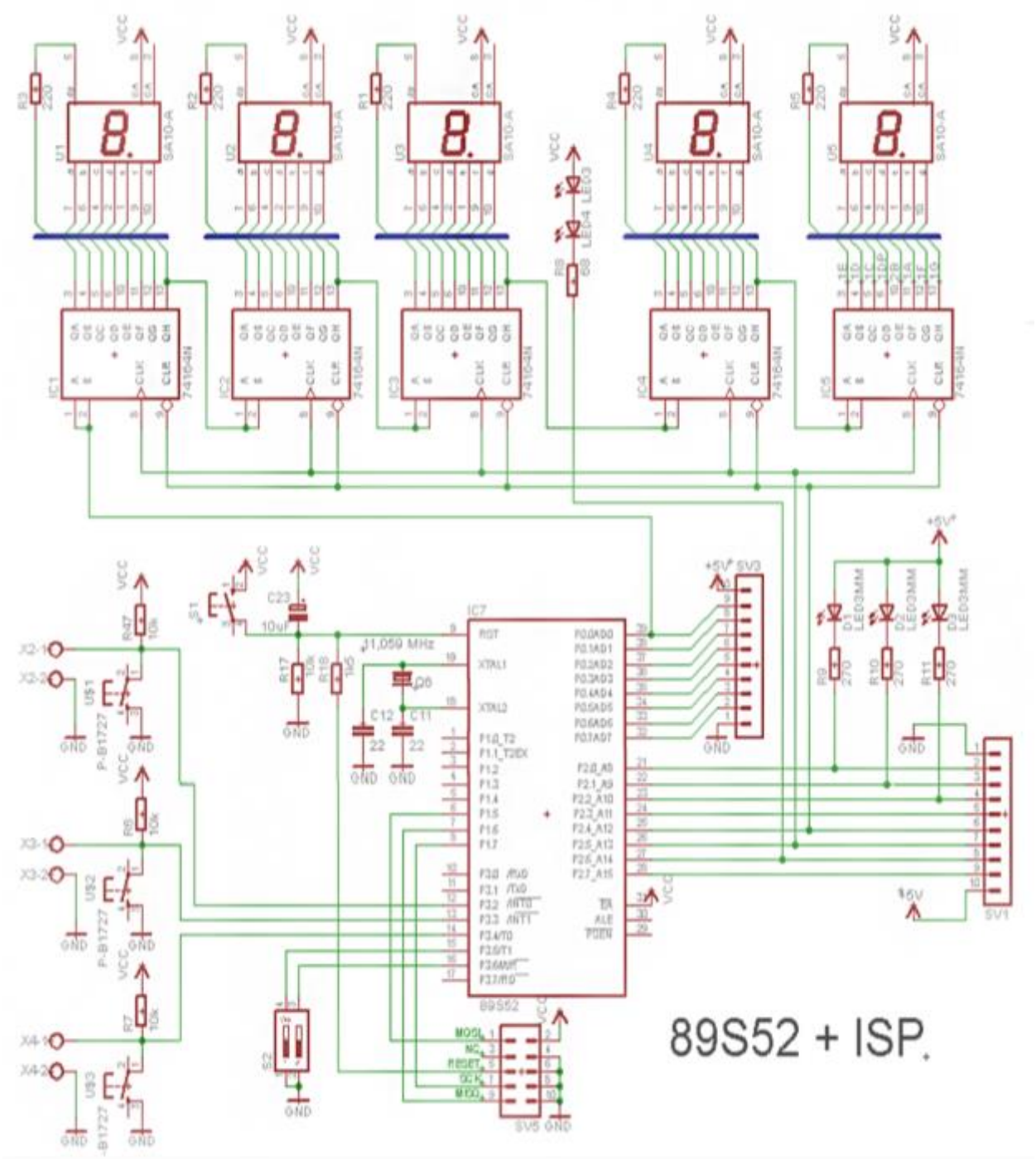
Obr. 15 a) Reálná deska plošného spoje, součástky



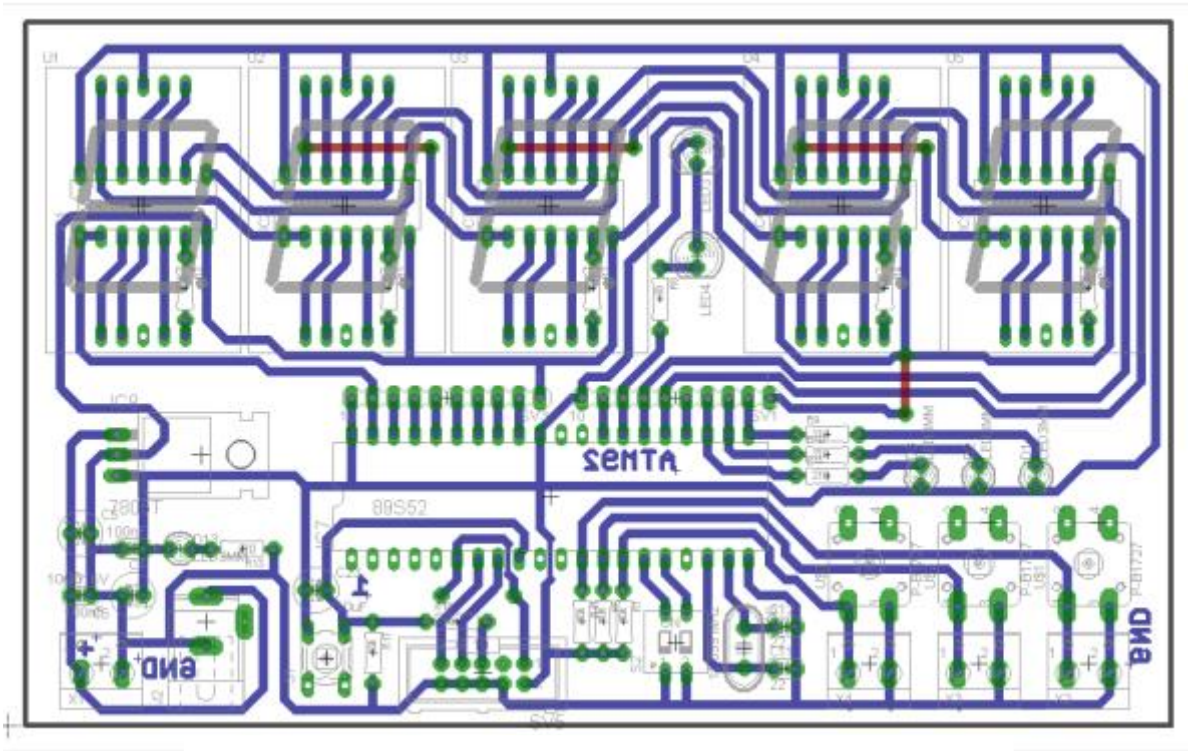
Obr. 15 b) Reálná deska plošného spoje

Schéma modelu

Zapojení musí být doplněno napájecím obvodem který je popsán v kapitole 3.2 Napájecí obvod



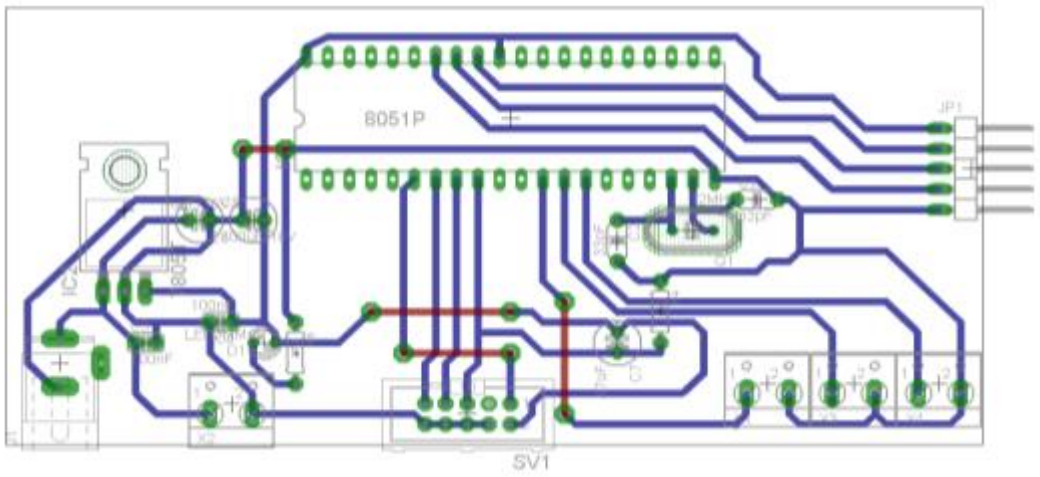
Obr. 16 a) Schéma modelu



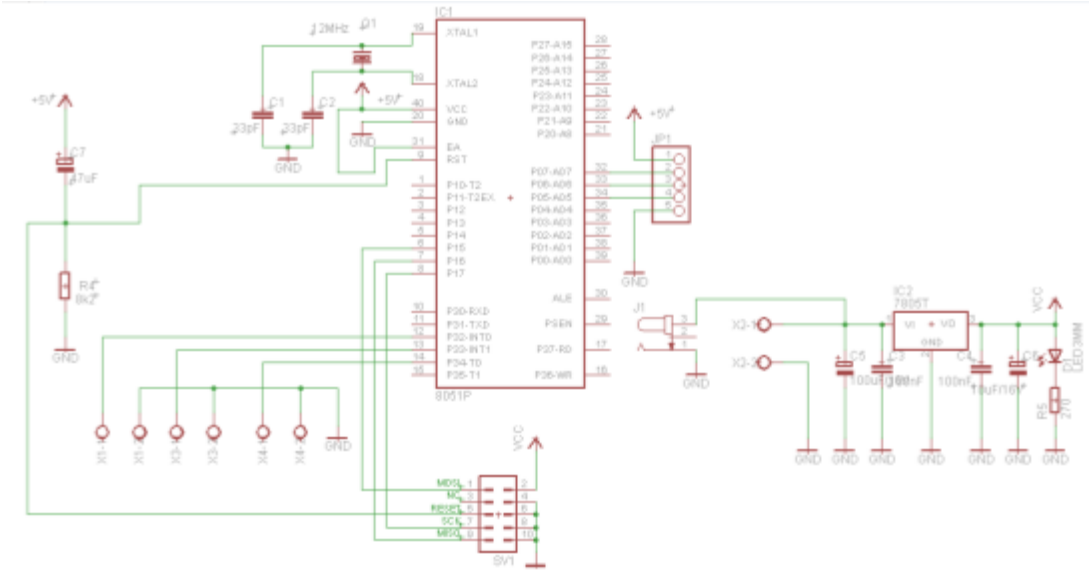
Obr. 16 b) Zapojení modelu, deska plošného spoje

Zapojení pro osobní využití

Přibližně takto bude vypadat zapojení, které budu konstruovat pro osobní využití. Segmentové zobrazovače a tlačítka budou připojena externě.



Obr. 17 a) Zapojení pro osobní užití, deska plošného spoje



Obr. 17 b) Zapojení pro osobní užití, schéma

Přibližná kalkulace

Jednotlivé rezistory a kondenzátory uváděné nejsou.

AT89S51-24PU DIL40 ATMEL	42,00,-	x1
74164 8BIT.POSUVNÝ REGISTR	9,00,-	x5
SVORKOVNICE 2PIN	3,00,-	x4
TLAČÍTKO	4,00,-	x4
Displej 7segmentový SA 10-21 HWA, 25 mm,	28,00,-	x5
KRYSTALICKÝ OSCILÁTOR	4,00,-	x1
LED 3 MM a 5MM	1,00,-	x5
STABILIZÁTOR	10,00,-	x1
CELKEM:	**274,-**	

Zdrojový kód

```
                ORG     00H

        TL_RESET BIT P3.4
        TL_START BIT P3.3
        TL_STOP BIT P3.2
        TL_SWITCH1 BIT P3.5
        TL_SWITCH2 BIT P3.6

        OUT_CLK BIT     P2.5
        OUT_DATA BIT    P0.0
        OUT_RST BIT     P2.4
        OUT_DVOJT BIT P2.6
        OUT_LED1        BIT P2.0
        OUT_LED2        BIT P2.1
        OUT_LED3        BIT P2.2
        PV                      EQU                     56330


 START:                                 ; začátek
                MOV     TMOD,#00000001B ;nastavení čítače 0 v režimu 1 jako
časovače
                                CLR     TF0
                                MOV     TL0,#LOW  PV
                                MOV     TH0,#HIGH PV
                        SETB    TR0
                ACALL   NULUJ           ; vynulování všech paměťových míst
                MOV     R0,#20D         ;vložím adresu prvního displeje, do
které budu ukládat hodnotu požadované zobrazované číslice, 0,1,2....
 CEKEJ:
                ACALL   LED ;podprogram pro testování stavu dvojtečky
                ACALL   NASTAV          ;uložení stavu segmentů do paměti
                ACALL   CLK                     ;vyslání dat
                ACALL   ZPOZD                   ;zpoždění (necelých 10ms)
                JNB     TL_START,NACTI
                SJMP    CEKEJ
 NACTI:         JNB     TL_STOP,KONEC   ;testuje jestli není jedno STOP
tlačítko stisknuto
                JNB     TL_RESET,START
                ACALL   NASTAV
                ACALL CLK       ;vyšle hodnoty
                ACALL   ZPOZD   ;zpoždění aby byla čísla na displejích
postřehnutelná
                INC     @R0
                CJNE    @R0,#0AH,NACTI  ;kontroluje zde nejsem mimo tabulku
                MOV     @R0,#00H
                INC     R0      ;posunu se na další display
                SJMP    DIS2
 KONEC:         CLR     OUT_LED1        ;rozsvícení zelené LED
                ACALL   CLK
                ACALL   ZPOZD
                ACALL   NASTAV
                JNB     TL_RESET,START
                SJMP KONEC
                ; stále vysílá hodnoty které byly načteny v době sepnutí
STOP tlačítka
```

```
 DIS2:                ;obsluha dalších displejů
                INC     @R0                                     ;inkrementuje
číslici pro druhý displej (paměť na adrese 21D inkrementuje o 1 a tím dojde
v příštím vyslání k vyslání dalšího čísla)
                CJNE    @R0,#0AH,NACTI          ;testuje      jestli      jsme
nepřesáhli číslo 9
                MOV     @R0,#00H
                INC     R0                                              ;pokud
ano vynuluje druhý displej a pokračuje v programu na další displej
                INC     @R0
                CJNE    @R0,#0AH,NACTI                          ;testuje
jestli jsme nepřesáhli číslo 9
                MOV     @R0,#00H
                INC     R0
                INC     @R0             ; přesune na další displej
                CJNE    @R0,#0AH,NACTI                          ;testuje
jestli jsme nepřesáhli číslo 9
                MOV     @R0,#00H
                INC     R0
                ; přesune se na pátý displej
                INC     @R0
                CJNE    @R0,#0AH,NACTI          ;testuje      jestli      jsme
nepřesáhli číslo 9
                ACALL   NULUJ                   ;Pokud  i  pátý  displej  se
přeplnil, program se vynuluje
                CLR     OUT_LED3   ;rozsvícení  červené  LED  pro  indikaci
přeplnění
                SJMP    NACTI
NASTAV:
;tento  podprogram  obsluhuje  nastavení  paměťových  míst  pro  jednotlivé
segmenty
MOV     R0,#20D ; vrátím se zpět na první displej

        NEXTDIS:                        ;zde probíhá obsluha paměti
                ACALL   LED             ;kontroluje stav dvojtečky
                 MOV    A,@R0
                MOV     DPTR,#TAB       ;vložím do DPTR první adresu tabulky
s čísly
                MOVC    A,@A+DPTR
                MOV     B,A             ; uložím  číslici  do  B  abych  si  ho
nepřepsal při dalším testování
                MOV     R3,#00H         ;vynuluji  registr  pro  odkazování  na
segmenty

        NEXTSEG:        ; zde nastavuji jednotlivé segmenty a ukládám jejich
hodnotu do paměti

                MOV     A,R4    ;v R4 je hodnota která  slouží k odkazovaní
na místo v adresová tabulce (adresová tabulka pomáhá při ukládání dat do
paměti)
                MOV     DPTR,#TABADR
                MOVC    A,@A+DPTR       ;určím první místo v paměti pro daný
display

                ADD     A,R3                    ; od  této  adresy  se  budu
ukládat stavy jednotlivých segmentů  (tedy počátek TABADR na který odkazuje
R4 + číslo v R3 které ve spojení s TABADR určuje adresu paměťového místa
pro daný segment)
                MOV     R1,A
```

```
                MOV     @R1,#0FFH;Nyní na paměťové místo které je uloženo v
R1 zapíšeme samé 1
                MOV     A,B
                JB      ACC.7, POSUN
                MOV     @R1,#00H        ;pokud není jedna, nastaví se
paměťové místo v R1 na nulu aby to odpovídalo stavu segmentu
                        MOV     R1,#00H         ;po skončení testování
vynuluji R1

 POSUN:
        ;posunu číslo v akumulátoru (číslo které znázorňuje číslici
vysílanou na displej), abych mohl otestovat její další segment jak proběhlo
v předchozích krocích
                RL      A               ;posune aktuální číslici o 1 do leva
                MOV     B,A             ;uložím rotovanou číslici do B aby se
v dalších krocích nepřepsala
                INC     R3      ;zvolím další segment
                CJNE    R3,#08H,NEXTSEG                         ;pokud se
nastavilo všech 7 segmentů na displej pokračujeme na další displej
                INC     R0                      ;přesune na další displej
                INC     R4
                ;přesune odkaz na další místo adresové tabulky        (každý
displej má 8 paměťových míst)

                CJNE    R0,#25D,NEXTDIS
                ;testuje jestli nejsme už na konci (pátý displej má adresu
24D, proto testuji jestli nebyla překročena)
                MOV     R4,#00H         ; vynulování R4 (odkazu na TABADR )
                MOV     R0,#20D ;pokud se ještě nastavily všechny 4 displeje
vrací se na začátek jejich obsluhy
                RET     ;návrat z podprogramu

 ZPOZD: ;Zpoždění  10ms

        JNB     TF0,$
        CLR     TF0
        MOV     TL0,#LOW  PV
        MOV     TH0,#HIGH PV
        INC     R2              ;inkrementuje   R2,   které   řídí   blikání
dvojtečky, pokud inkrementuje 10krát (10x100ms = 1s ), pokud inkrementuje
na  20(20x10ms = 2s) zhasne dvojtečku

        RET
 LED:
 ;pokud inkrementuje 10krát (10x100ms = 1s ), pokud inkrementuje na  20
(20x10ms = 2s) zhasne dvojtečku a vynuluje R2
                CJNE    R2,#6AH,LEDOFF
                MOV     R2,#00H
                SETB    OUT_DVOJT
 LEDOFF:        CJNE    R2,#35H,LEDON
                CLR     OUT_DVOJT
 LEDON:         RET
 CLK:
        MOV     R1,#30D                 ; od adresy 30D jsou uloženy stavy
jednotlivých segmentů
 NEXTCLK:
        SETB    OUT_DATA                        ;nastavit vývod P2.0 na 1
        CJNE    @R1,#00H,VYSLI          ;kontrola zdali je segment nulový
        CLR     OUT_DATA                        ;pokud je segment nulový
nastaví vývod na 0
 VYSLI:
```

```
        SETB    OUT_CLK                 ;náběžná hrana CLK
        NOP
        NOP
        NOP
        NOP
        CLR     OUT_CLK                 ;sestupná   hrana,   na   ní   reaguje
posuvný registr
        NOP
        NOP
        NOP
        NOP
        NOP
        INC     R1
        CJNE    R1,#70D,NEXTCLK
        MOV     R1,#00H
        RET
 NULUJ: MOV     R0,#20D                 ;vynuluje od adresy 20H
 NUL:   MOV     @R0,#00H        ;vynuluje aktuální paměťové místo
        INC     R0              ;další paměťové místo
        CJNE    R0,#70D,NUL
        MOV     R2,#00H
        MOV     R3,#00H
        MOV     R4,#00H
        MOV     R5,#00H
        MOV     R6,#00H
        MOV     R7,#00H

        RET
   TAB:                                         ;tabulka čísel
                            DB 10001000b    ;0
                            DB 11101011b
                            DB 01001100b
                            DB 01001001b
                            DB 00101011b
                            DB 00011001b
                            DB 00011000b
                            DB 11001011b
                            DB 00001000b
                            DB 00001011b ;9

        TABADR:
        ; adresová tabulka, na těchto paměťových místech začínají adresy
pro jednotlivé displeje
        DB              30D
        DB              38D
        DB              46D
        DB              54D
        DB              62D

        END
```