Cvičení z předmětu:

KAE/MPP

Katedra aplikované elektroniky a telekomunikací

přednášky:	prof. Ing. Jiří Pinker, CSc.	pinker@kae.zcu.cz	EK517
cvičení:	Ing. Petr Weissar, Ph.D.	weissar@kae.zcu.cz	EK515
	Ing. Kamil Kosturik, Ph.D.	kosturik@kae.zcu.cz	EK515
	Ing. Petr Krist, Ph.D.	krist@kae.zcu.cz	EK507

Plán cvičení

1. Úvod a seznámení – lab., IDE Keil uVision, první program
2. HW – porty, časovač, přerušení
3. HW – displej LED
4. HW – sériový port – úvod
5. HW – sériový port – přerušení, kruhový buffer
6. HW – LCD displej
7. Samostatná práce – zužitkování probraných znalostí
8. Sam. práce – dokončení. Grafický LCD
9. HW – PWM generované programově
10. Watchdog a práce s ním
11. Samostatná práce 2
12. Samostatná práce 2
13. Samostatná práce + Zápočet

Sem. práce - podrobnosti

- Semestrální práce se vypracovává ve skupinách (=dvojice)
- Typicky využívá HW vývojové desky
- Po domluvě jiný HW/mikroprocesor
  - AVR, Freescale, ARM, …
  - možno využít stávající projekt, bakalářku, …
- Programováno min. ze 2/3 v C/C++

Bezpečnost v laboratoři

- Protokol s datem zkoušky "padesátky"
- Pracuje se s bezpečným napětím, deska napájena z USB
- Změny HW konfigurace nechat schválit cvičícím
- Změny zásadně provádět při odpojeném napájení !!!

Podmínky zápočtu

- semestrální práce – obhájená, funkční, nutná aktivní znalost !!!
- praktická účast na cvičeních, znalost probírané problematiky

Doporučená literatura

- libovolná učebnice programovacího jazyka C
- Dokumentace k jednočipovým mikropočítačům x51
- Dokumentace vývojové desky a mikropočítače Atmel 89S8253
- Dokumentace periferních obvodů, příp. vlastního HW
- Knihy
  - Jednočipové mikropočítače řady 8051 – Roman Skalický (BEN)
  - C pro mikrokontroléry – Burkhard Mann (BEN)
  - Mikroprocesory a mikropočítače : obecné principy konstrukce současných mikroprocesorů a mikropočítačů – prof. Pinker (BEN 2004)

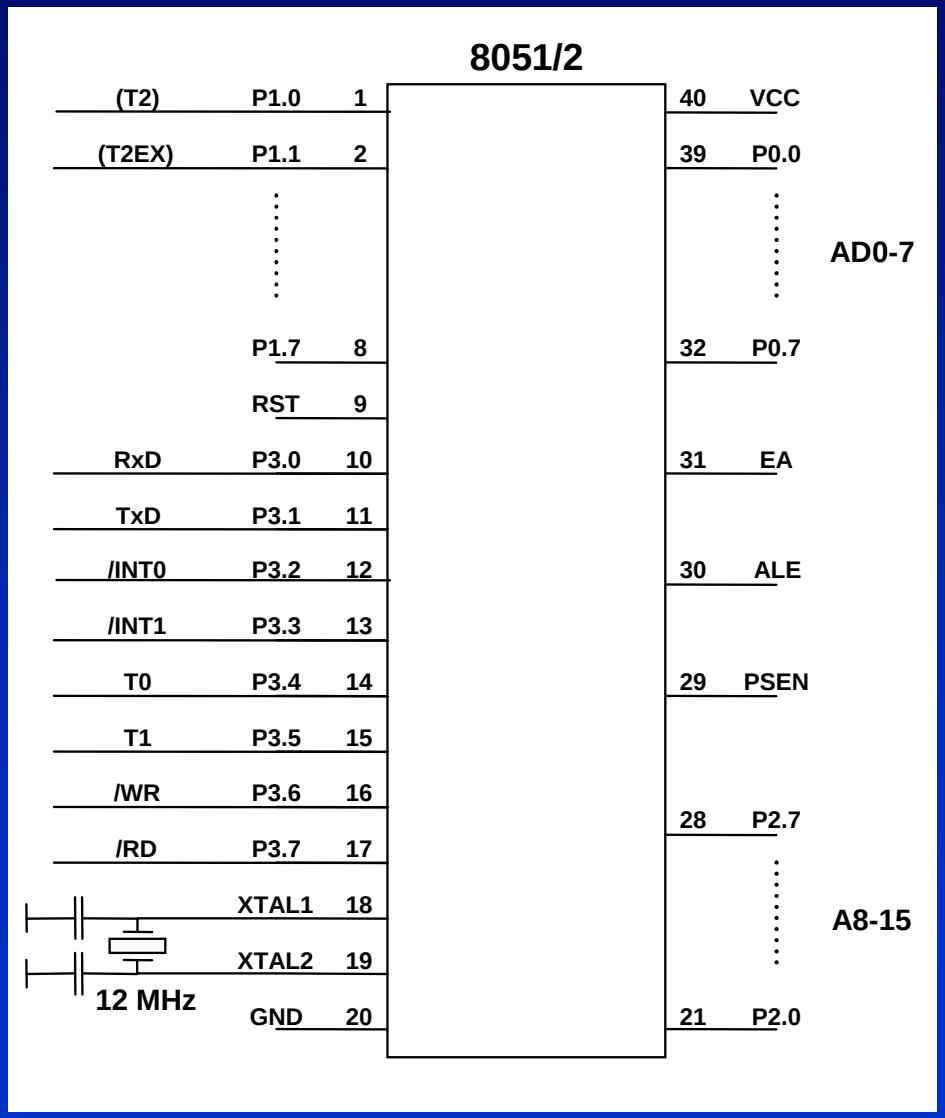
Další provozní informace

- Konzultace nejlépe v rámci cvičení
- Možno využívat laboratoř mimo své cvičení, respektovat rozvrh
- Ve volných hodinách mají v laboratoři přednost diplomanti

Pracovní soubory

- Interní síť se souborovým serverem mimo AFS prostor s mapovanými disky
  - disky X:, Z: - servisní a SW/systémový - ReadOnly
  - disk T: - dočasný datový, společný, maže se o půlnoci
  - disk H: - osobní data každé skupiny
- Přilogování jménem počítače – el510.. (uživatel PC bez hesla)
  - automaticky se spustí "logon" dávka s prohlížečem
  - přihlášení pomocí Orion jména/hesla člena skupiny
  - vyberte si předmět, kterému se hodláte věnovat
  - po ukončení prohlížeče se příslušně namapuje disk H:
- Po skončení práce se "odlogujte" ve WinXP
- Automaticky se odesílá obsah adresáře pojmenovaný "mailem"

HW – základní Intel 8051/2



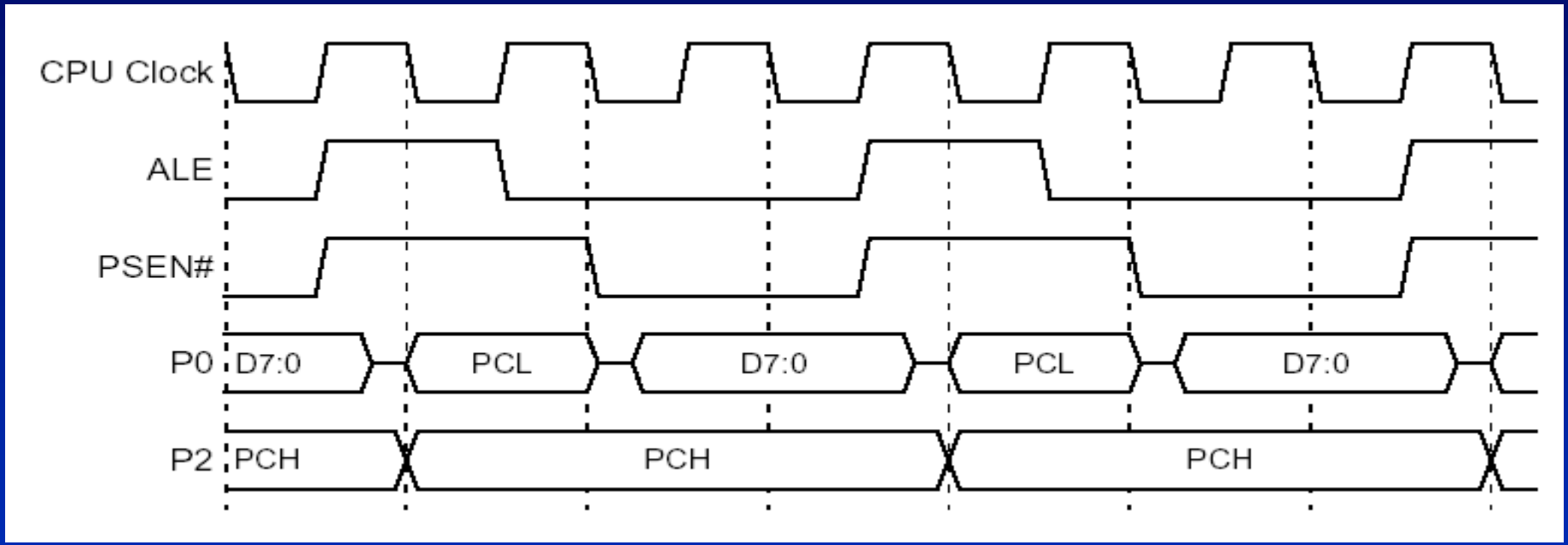
základní signály

- **XTAL1, 2** – zde 12MHz, pro komunikační aplikace se používá např. 11,0592MHz
- **RST** – ReSeT – typicky aktivní v 1
- **PSEN** – Program Store ENable – čtení z vnější paměti programu
- **ALE** – Address Latch Enable
- **EA** – External Access – povolení přístupu k vnější paměti programu (1=int, 0=ext)

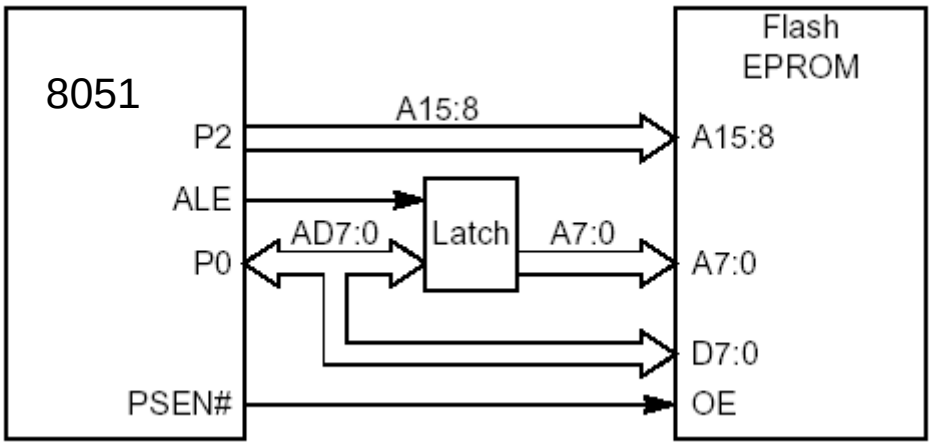
porty/brány

- **P0** – univerzální I/O nebo spodní část adresové sběrnice v časovém multiplexu s datovou sběrnicí
- **P1** – univerzální + alternativně vstupy/výstupy speciálních periférií
- **P2** – univerzální nebo horní část adresové sběrnice
- **P3** – univerzální, sdílení funkcí pro sériový kanál (**RxD**, **TxD**), externí přerušení (**INT0**, **INT1**), vstupy čítačů (**T0**, **T1**) a řízení vnější paměti dat (**RD**, **WR**)

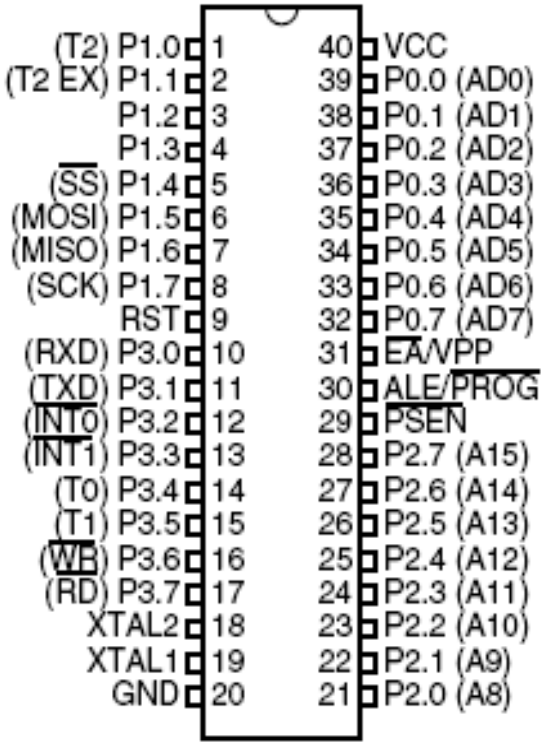
HW – x51 a A+D multiplex

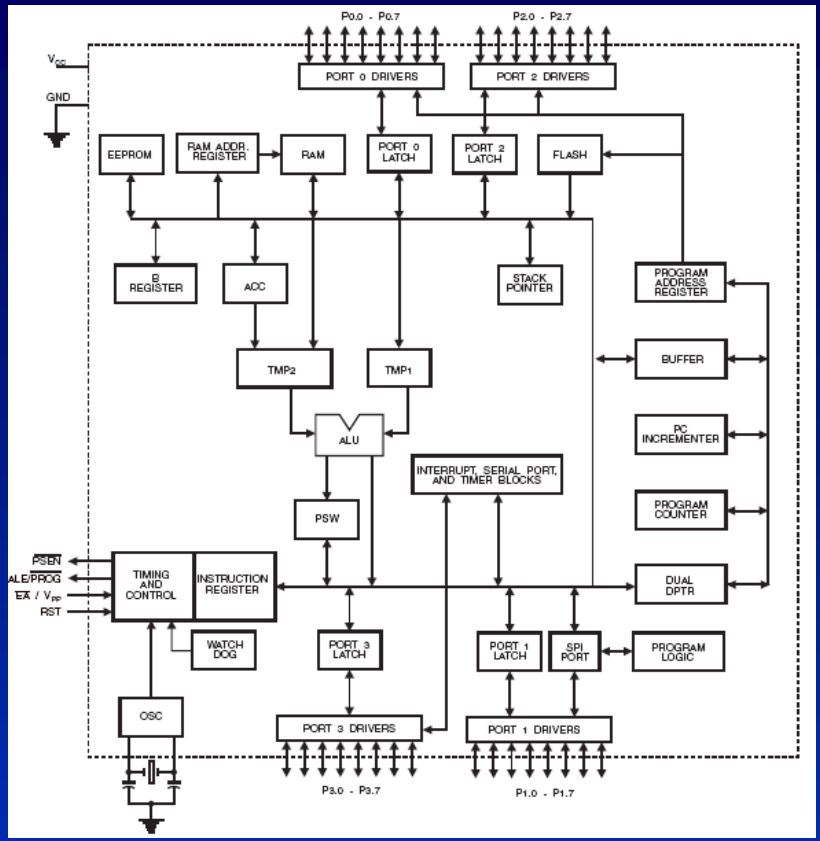


P0 – adresa A0:7 a D0:7
- sestupná hrana ALE zapisuje do vnějšího registru spodních 8 bitů adresové sběrnice
- v dalším kroku je P0 vyhrazena pro datovou sběrnici
- obdobně pro externí data, místo PSEN použit /RD nebo /WR



Atmel 89S8253



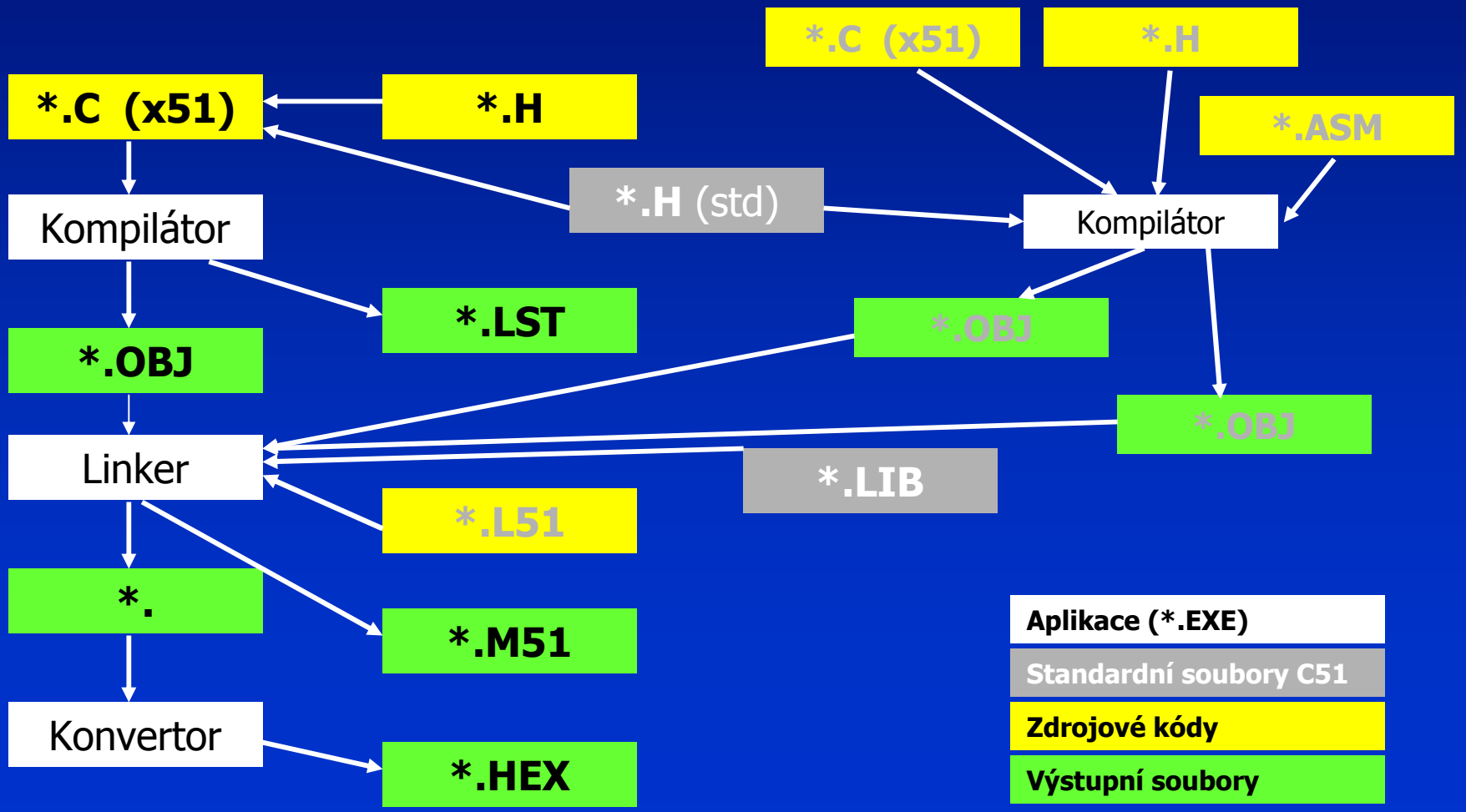


- 12kB FLASH paměť programu
- Programování ISP (v zapojení)
- 2kB EEPROM
- 256B interní RAM (=8052)
- Vylepšený sériový port
- SPI (sériové rozhraní MOSI/MISO/SCK)
- Watchdog, vylepšený RESET s Brown-out
- Dvojitý DPTR (výhoda při generování kódu v C)

C51 – práce se soubory

Použité IDE Keil uVision většinu podrobností skrývá

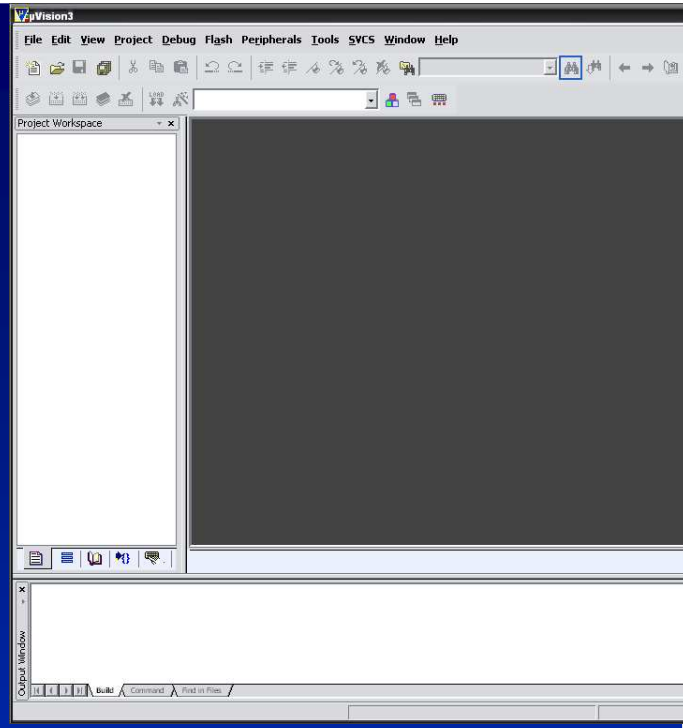
- soubory přesto vznikají "automaticky"
- příslušné parametry je možno nastavit dialogy v IDE nebo přepsáním souborů s konfigurací

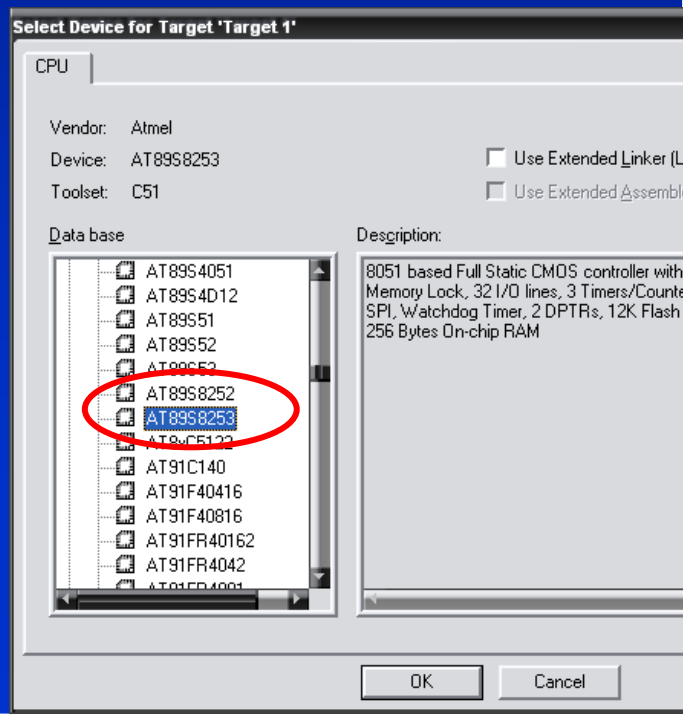


Keil uVision IDE

- Spustit ikonou na ploše 
- Zdrojové soubory představují projekt
  - určuje vlastnosti cílového procesoru
  - možno složit více modulů ("zdrojáků")
  - řeší kompilaci závislostí apod.
  - více projektů může sdílet jeden soubor
- Nový projekt - Project/New
  - Device zvolit **Atmel 89S8253**
  - "Standard Startup Code" - Ano
  - File/New ... - vytvoří nový soubor
  - Save as ... - vybrat adresář projektu
    - přípona souboru typicky .C
  - pozor, ještě není součástí projektu !!
  - vlastnost projektu (**Source Group 1**)
  - dále Add Files to Group - vyberu a klik Add
  - buď přidám další nebo "Close"

- Vše ukládat na H: !





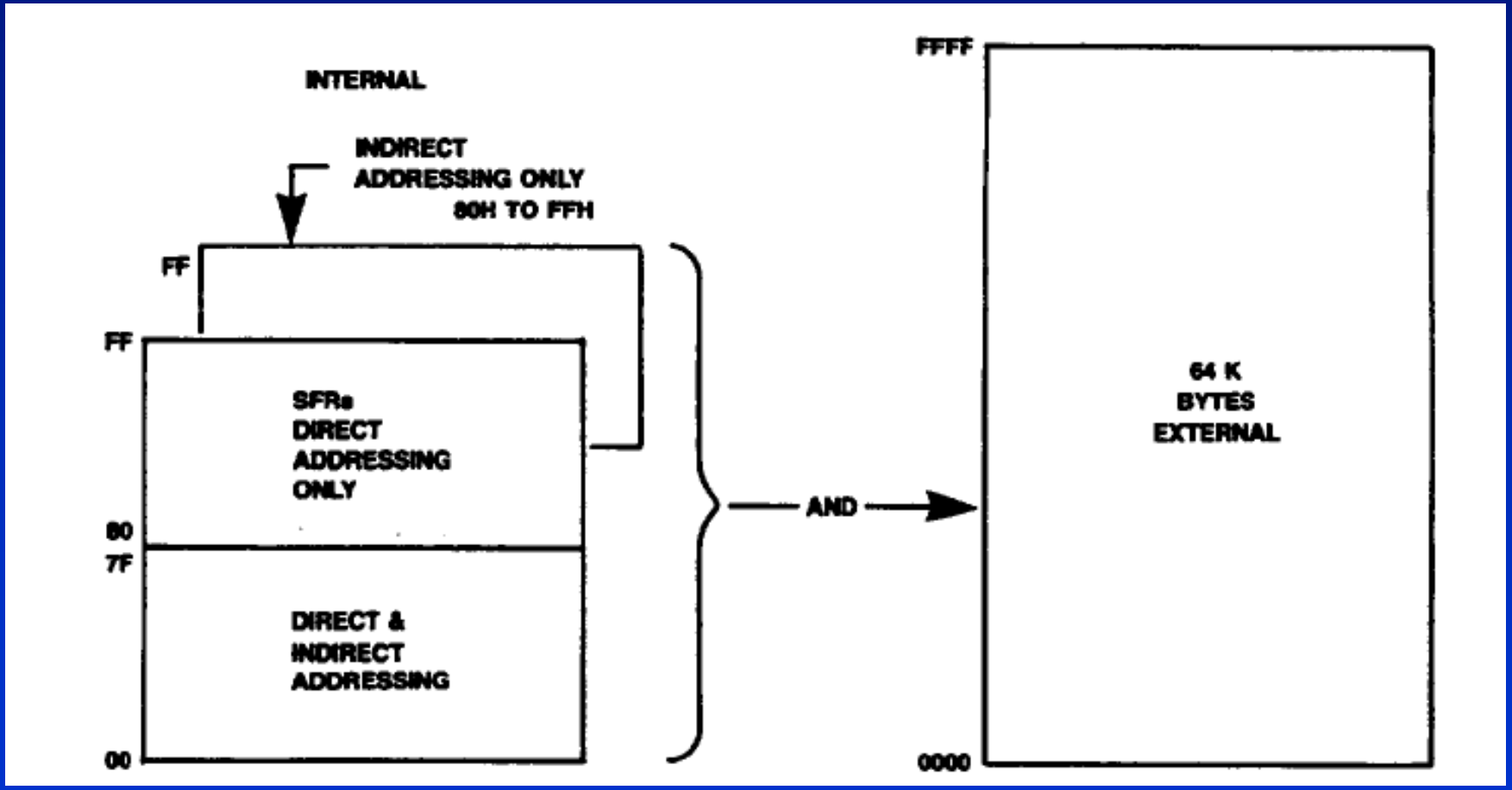
Triviální program - první aplikace

```
#include <reg51.h>

int main(void)
{
  P1 = 0x0F;     // na branu P1 vlozit hodnotu bitove 0000 1111
  while (1)      // nekonecna smycka
    ;            // prazdny obsah
}
```

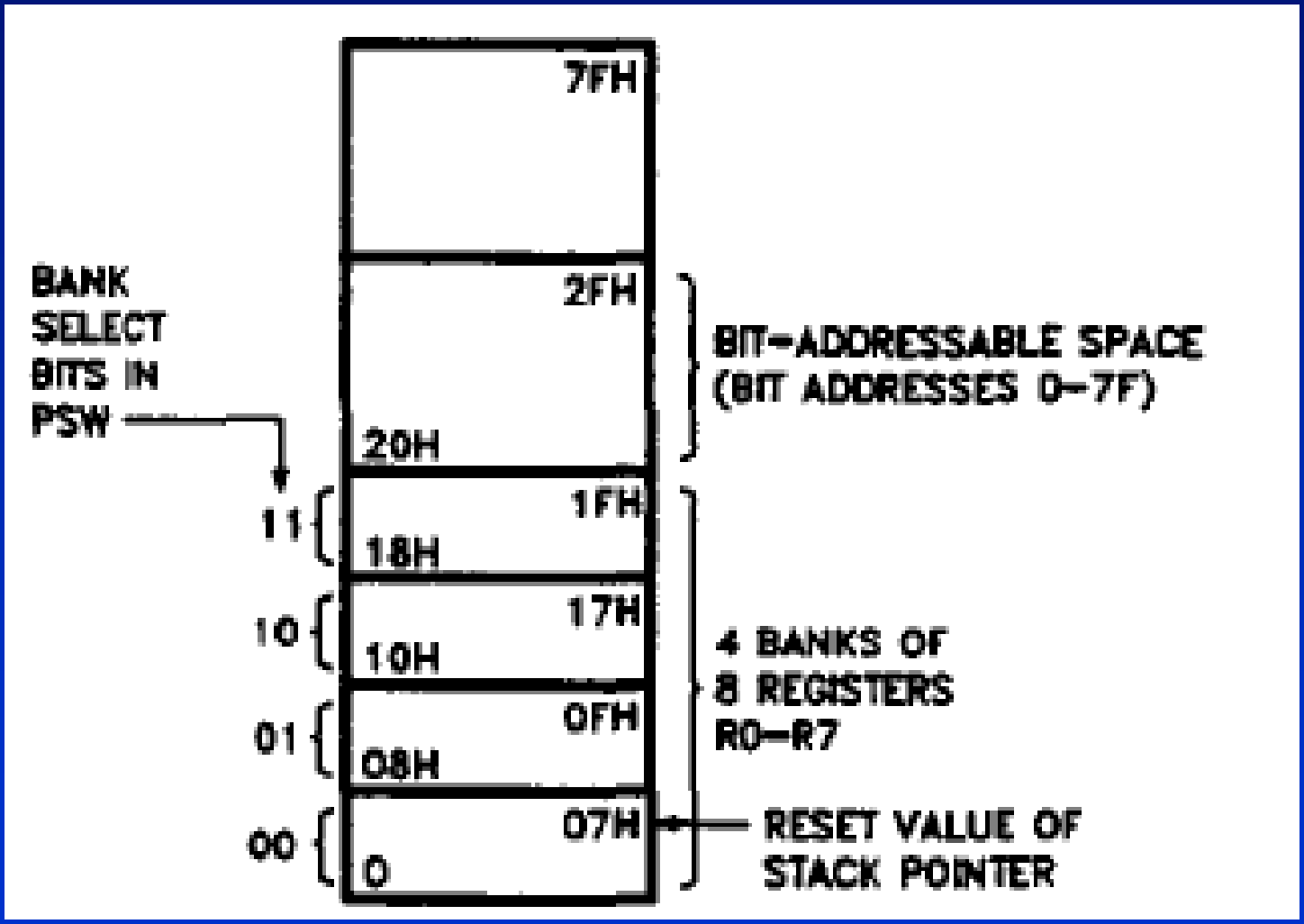
- Hlavičkový soubor REG51.H obsahuje definování adres řídících registrů periférií
  - SFR - Special Function Registers
- P1 - registr odpovídající bráně P1 (piny P1.0-7)
- Většina registrů je hodnota typu unsigned char
- Některé registry jsou bitově přístupné, pak jsou pojmenované i tyto bity

- Pro speciální registry konkrétního typu existují upravené hlavičky - v našem případě Atmel/REG8253.H

Datové adresové prostory 8051



Datová paměť 0x00 - 0x7F



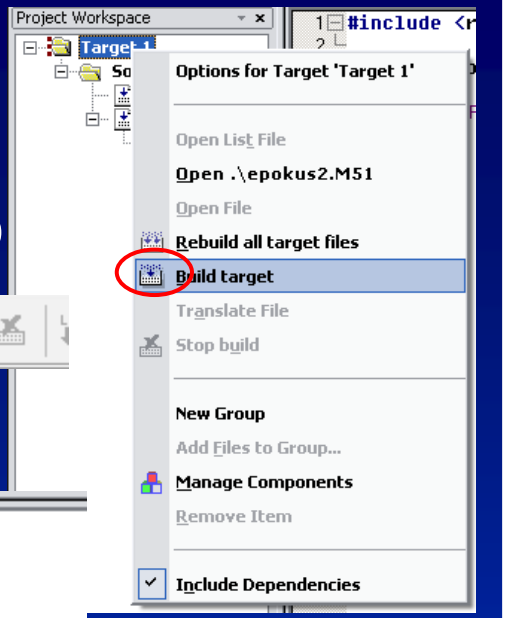
SFR registry AT89S8253 **0x80 - 0xFF**

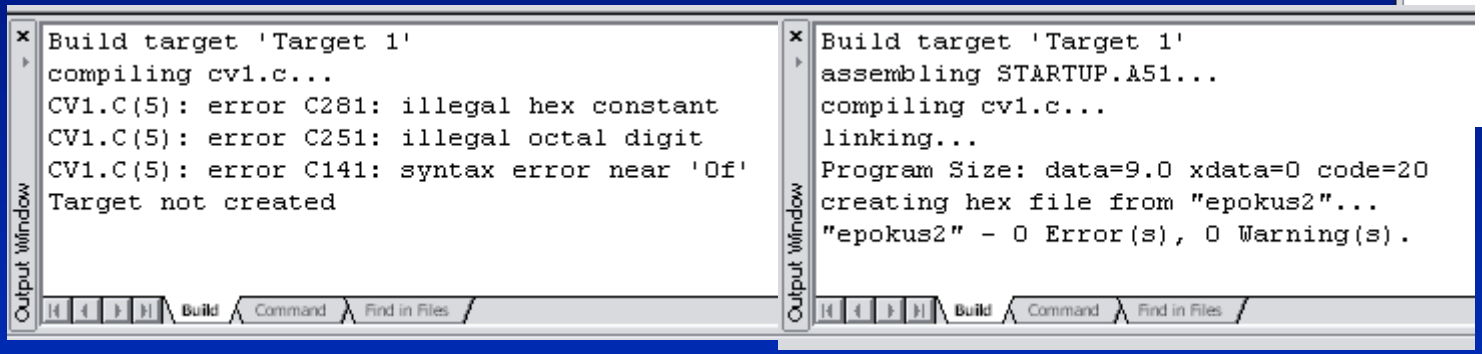
0F8H									0FFH
0F0H	B 00000000								0F7H
0E8H									0EFH
0E0H	ACC 00000000								0E7H
0D8H									0DFH
0D0H	PSW 00000000					SPCR 00000100			0D7H
0C8H	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			0CFH
0C0H									0C7H
0B8H	IP XX000000	SADEN 00000000							0BFH
0B0H	P3 11111111							IPH XX000000	0B7H
0A8H	IE 0X000000	SADDR 00000000	SPSR 000XXX00						0AFH
0A0H	P2 11111111						WDTRST (Write Only)	WDTCON 0000 0000	0A7H
98H	SCON 00000000	SBUF XXXXXXXX							9FH
90H	P1 11111111						EECON XX000011		97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XXXXXXX0	CLKREG XXXXXXX0	8FH
80H	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000	SPDR ########	PCON 00XX0000	87H

Note: # means: 0 after cold reset and unchanged after warm reset.

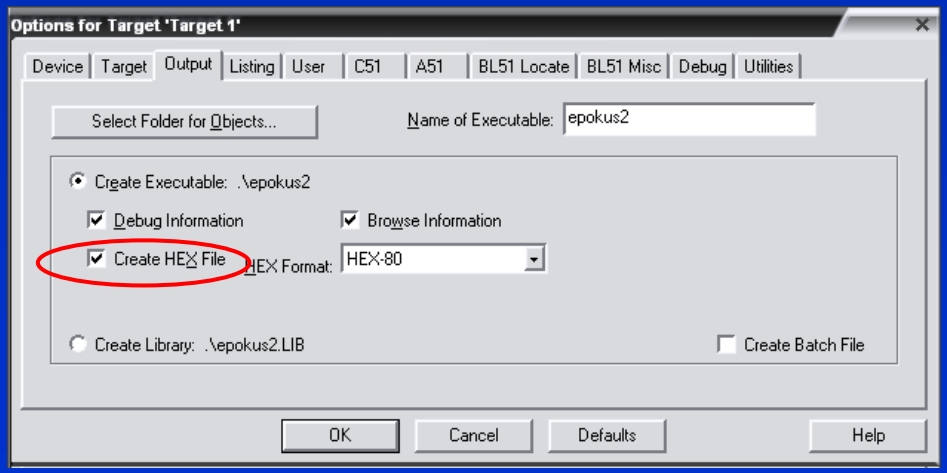
Přeložení programu

- Spuštění překladu - vlastnost projektu nebo menu Project/Build
- Výsledek v okně Build
- V případě chyb kliknutím přejde na kód

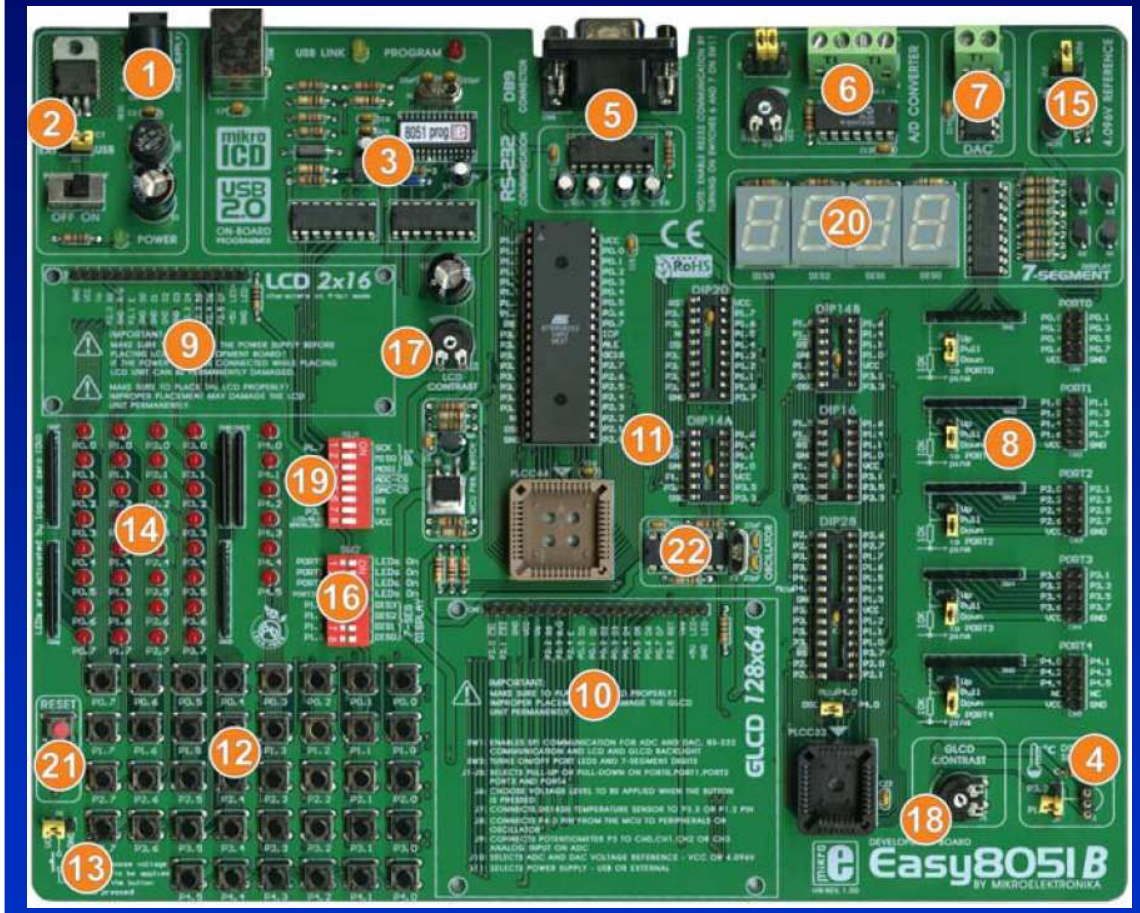




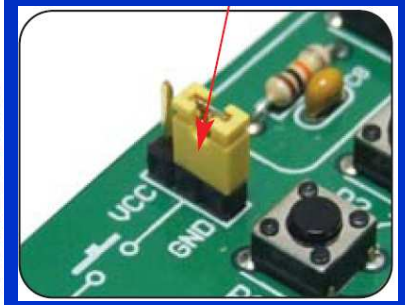
- V adresáři se vytváří/aktualizují soubory (.lst, .obj apod.)
- Kde je výstup **.HEX** ??
- Options for Target
- Záložka Output
- Create HEX File



Vývojová deska Easy8051B



1. napájení
2. výběr napájení ext/usb
3. USB programování
4. teplotní čidlo
5. sériový port RS232
6. A/D převodník
7. D/A převodník
8. porty procesoru P0 - P4
9. LCD displej 2x16 znaků - znakový
10. LCD displej 128x64 bodů - grafický
11. jednočipový mikropočítač AT89S8253
12. blok spínačů
13. přepínání Hi/Lo významu tlačítek
14. LED na portech
15. napěťová refence pro A/D a D/A
16. přepínání využití LED na portech a LED displeje
17. kontrast textového LCD
18. kontrast grafického LCD
19. přepínače periférií na desce
20. multiplexovaný LED displej 7-segment
21. obvod generování RESETu
22. generátor hodin pro procesor (8MHz)



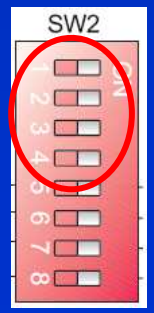
Počáteční nastavení

Napájení z USB (obr. 1)

Funkce tlačítka - stiskem se připojí GND (obr. 2)

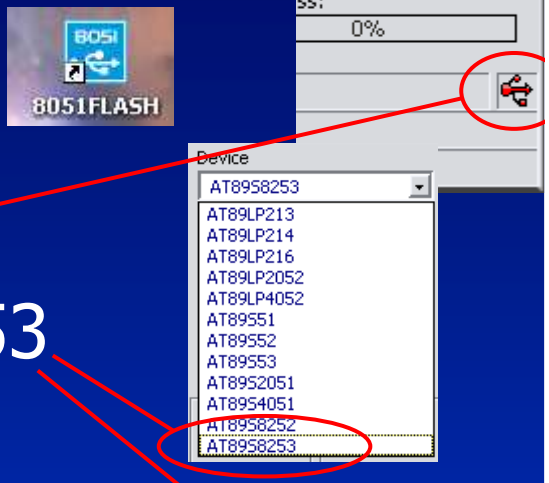
Zapnuty LED na všech portech (horní 4 přepínače)

Povoleny LED segmentovky (dolní 4 přepínače)

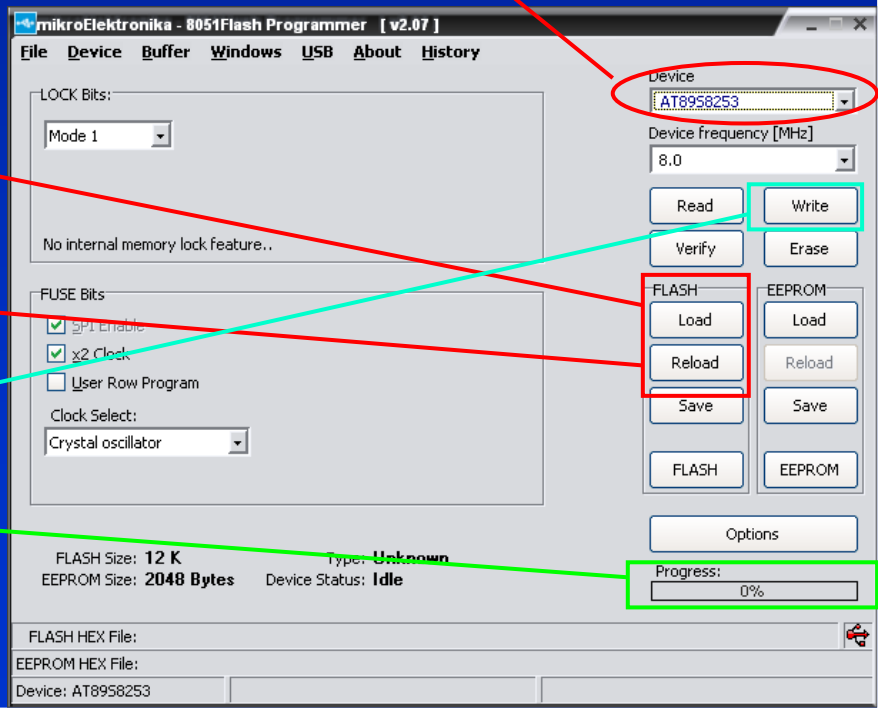


Flash loader, přenos do procesoru

- Spustit 8051FLASH - ikona na ploše
- Po zapnutí desky se objeví vpravo dole ikonka
- Kontrola/výběr "Device" - AT89S8253

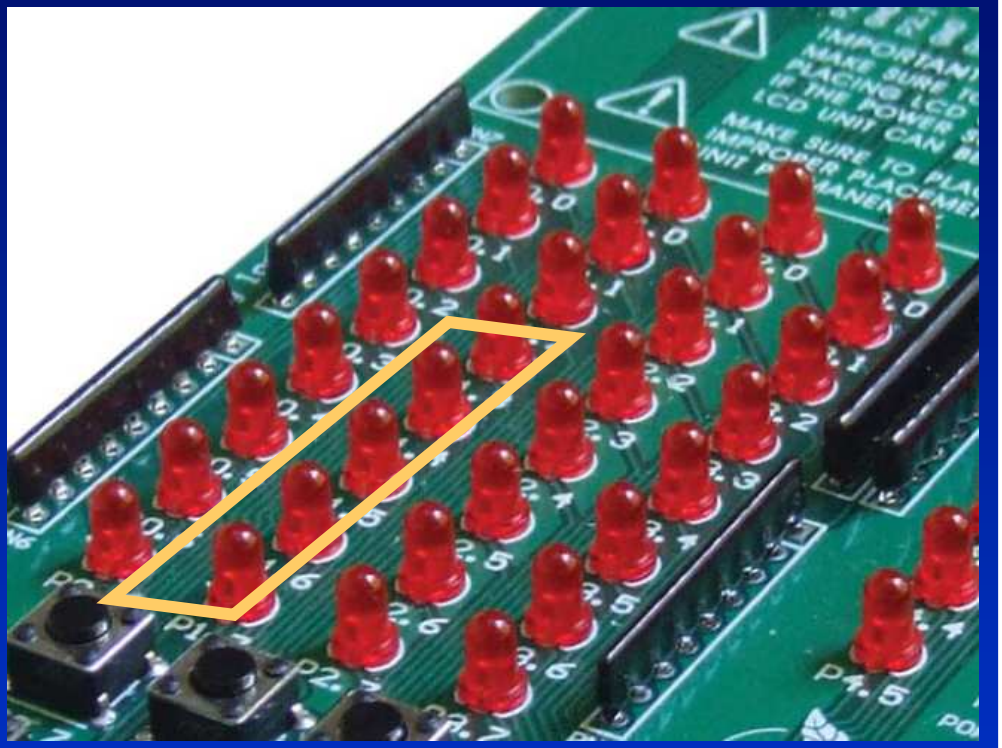


- Operace se soubory - Load v bloku Flash
- Příp. Reload posledního souboru (.HEX)
- Programování - Write
- Ukáže se "Progress"
- Program se ihned spustí



Výsledek první aplikace + úkol

- Výsledek programu
  - svítí P1.4 až P1.7
  - LED svítí při log 0



- **Rozmyslet si na příští cvičení**:
  - program, který rozsvítí všechny LED na P1
  - postupně je zhasne

Plán cvičení

1. Úvod a seznámení – lab., IDE Keil uVision, první program
2. **HW – porty, časovač, přerušení**
3. HW – displej LED
4. HW – sériový port – úvod
5. HW – sériový port – přerušení, kruhový buffer
6. HW – LCD displej
7. Samostatná práce – zužitkování probraných znalostí
8. Sam. práce – dokončení. Grafický LCD
9. HW – PWM generované programově
10. Watchdog a práce s ním
11. Samostatná práce 2
12. Samostatná práce 2
13. Samostatná práce + Zápočet

Řešení domácího úkolu

```
#include <reg51.h>

int main(void)
{
  int i, j;

  P1 = 0x00;     // na branu P1 vlozit hodnotu bitove 0000 0000
  for (i = 0; i < 8; i++)    // 8x - tedy pro kazdy bit 8-bitoveho slova
  {
    for (j = 0; j < 30000; j++)    // "hodne" opakovani
      ;                            // prazdny cyklus

    P1 <<= 1;    // posun hodnoty vlevo (smerem k MSB)
    P1 |= 0x01; // bitovy OR s hodnotou 0000 0001 - nejnizsi bit na 1
  }

  while (1)      // nekonecna smycka
    ;            // prazdny obsah
}
```

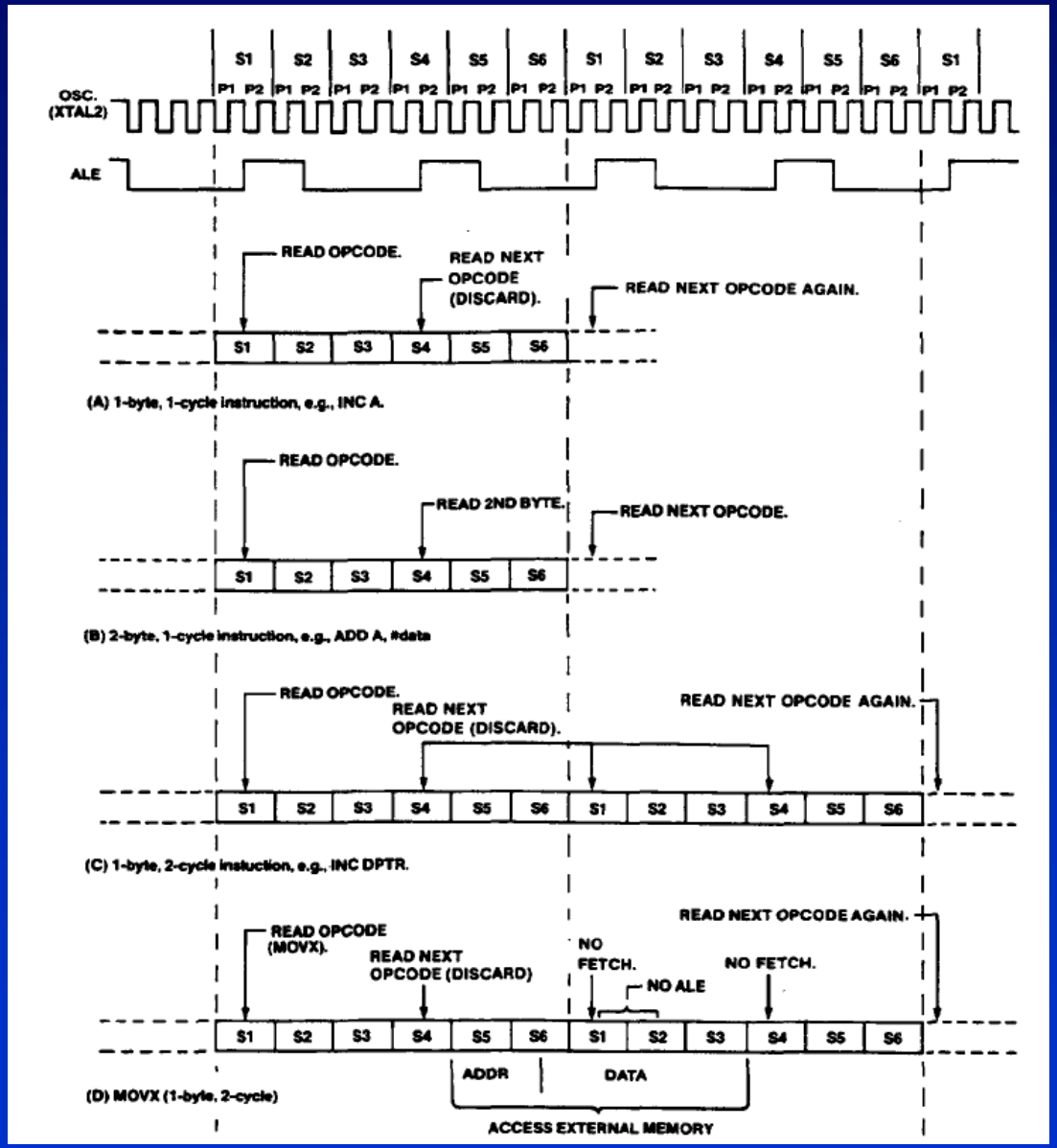
Časování akcí cyklem

- **For** cyklus v příkladu složí k pozdržení vykonávání programu na určitou dobu
  - dána rychlostí procesoru (takt hodin)
  - záleží na překladači, jak se přeloží
  - po dobu čekání prakticky nelze dělat nic jiného
  - nutno buď zkusmo ověřit (osciloskop) nebo kontrola v asembleru
  - **POZOR** - některé optimalizace "vyřadí" prázdné cykly a nahradí je přiřazením mezní hodnoty (zde tedy **for** nahrazen příkazem **j = 30000;**)
- Vynucené vykonávání prázdného těla cyklu
  - vypnutí optimalizace (liší se i u jednotlivých verzí)
  - vykonávání "prázdné operace" - **NOP** (No OPeration), kterou má každý procesor
  - v C51 použita pseudo-funkce **_nop_()**, která se překládá přímo instrukcí **NOP**
  - společně s dalšími (rotace apod.) najdeme v hlavičkovém souboru "intrins.h"

```
...
#include <intrins.h>
...
    for (j = 0; j < 30000; j++)
       _nop_(); // pseudo-funkce = instrukci NOP
...
```

```
Výstup v .LST souboru:
...
13  2             _nop_();
...
; SOURCE LINE # 13
0008 00                   NOP
...
```

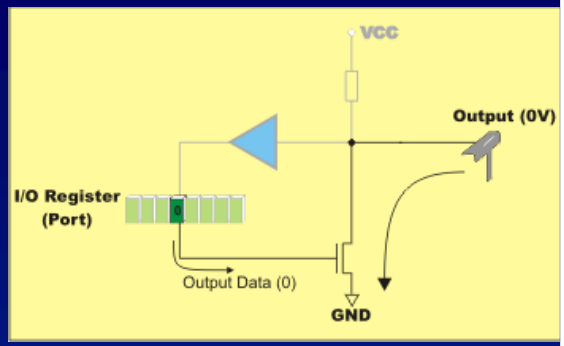
Časování instrukcí

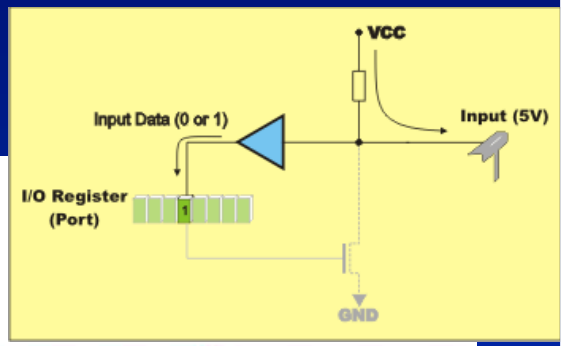


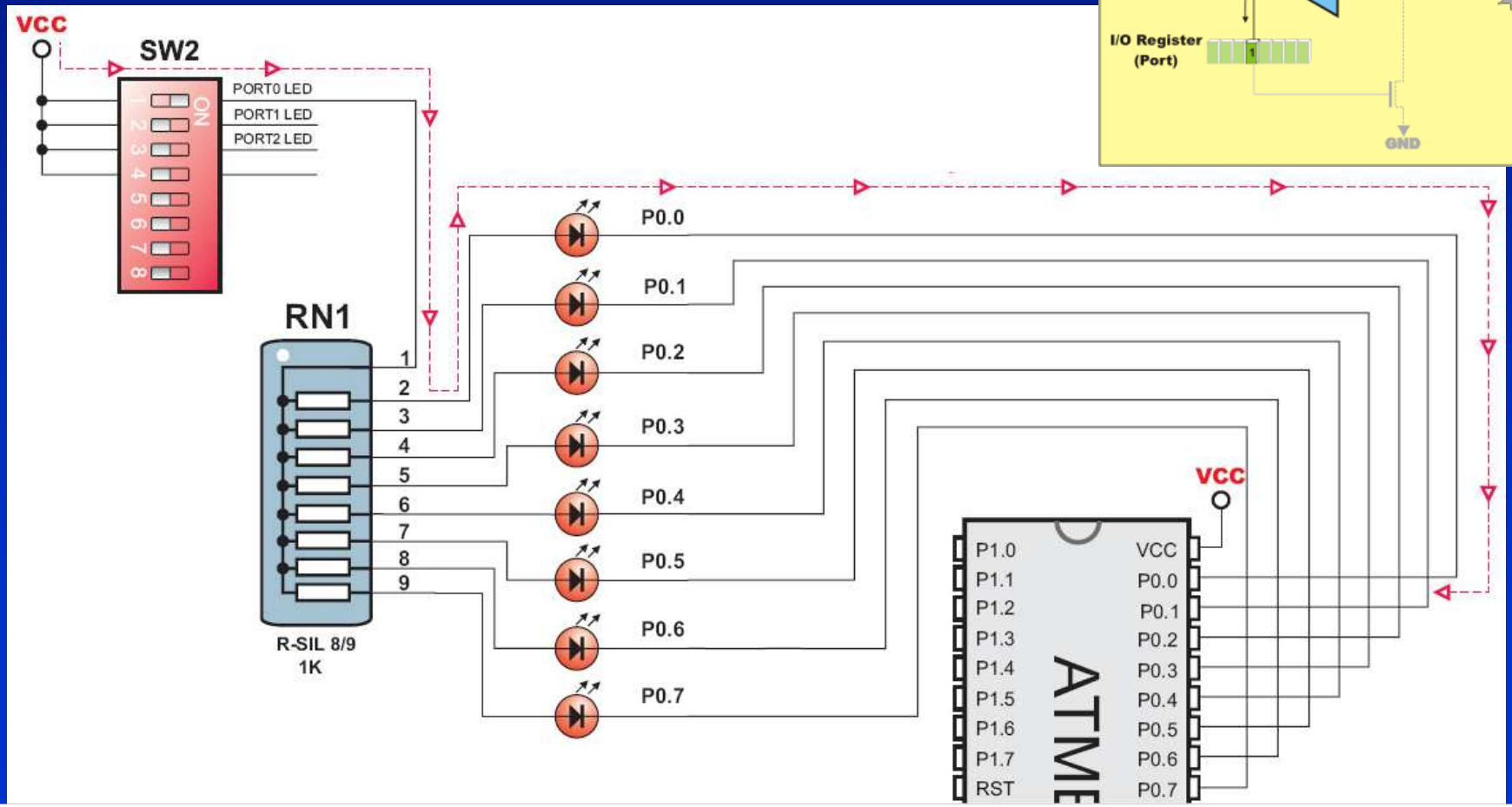
Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS (Continued)				
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12
LOGICAL OPERATIONS				
ANL	A,Rn	AND Register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12

Rozsvícení LED

- "Svítí" se stavem log. 0
  - proud do GND je výrazně větší, než proud z VCC přes interní pull-up
  - pozor na max. proud/obvod vývodem GND (datasheet)
  - běžné řešení buzení LED v číslicové elektronice

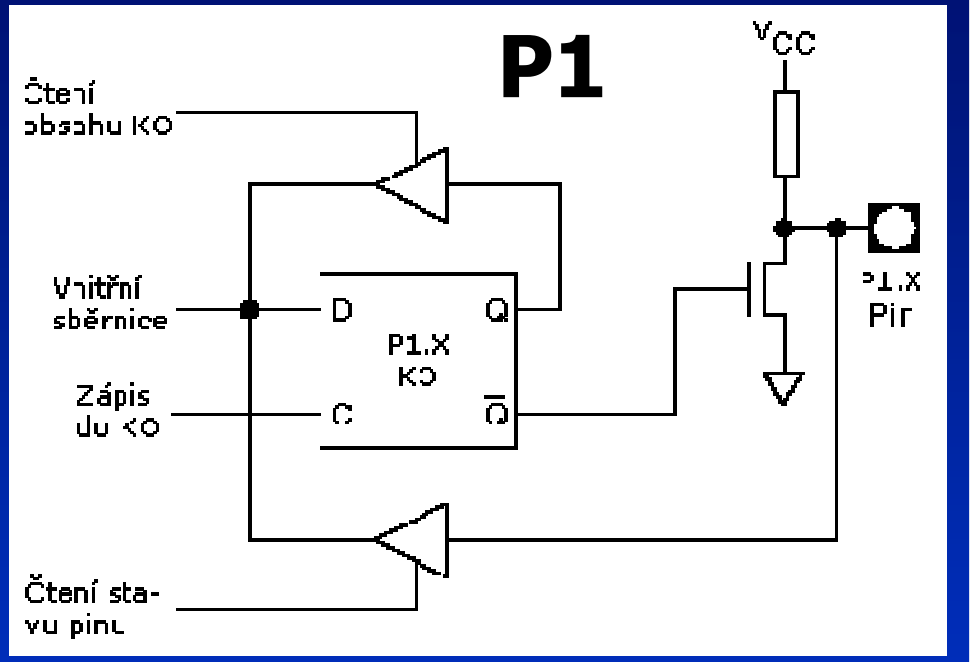


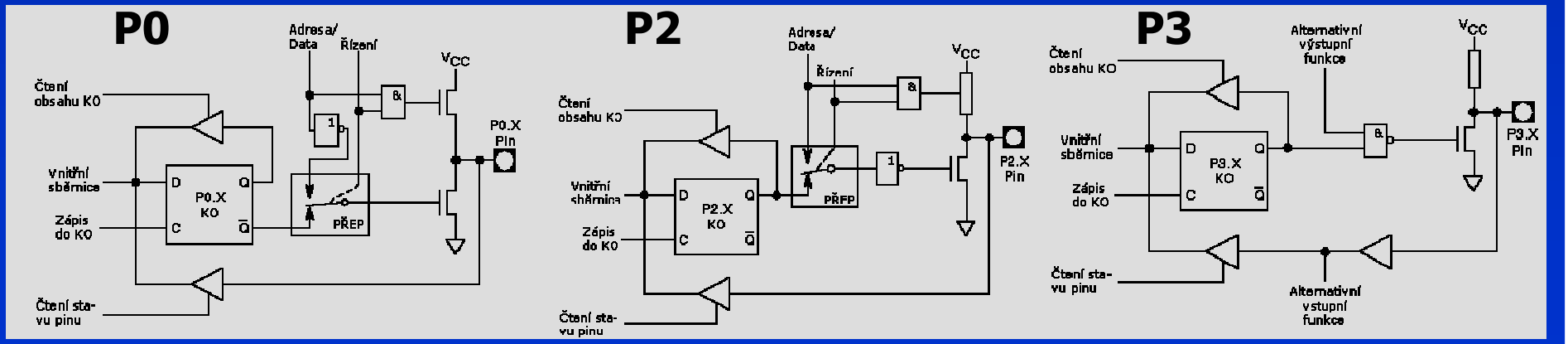




IO brány procesoru 8051

- P1 - P3
  - pseudo-obousměrné
- P0
  - klasicky obousměrná
  - vyžaduje externí pull-up
- P3 alternativní funkce
  - u nových i P1





HW časovač

```
#include <reg51.h>

int main (void)
{
  P1 = 0x0f;  // 0000 1111

  TMOD = 0x01;
  TR0 = 1;

  while (1)
  {
    while (!TF0)
      ;

    TF0 = 0;

    TH0 = (65536-50000) / 256;
    TL0 = (65536-50000) % 256;

    P1 = ~P1;
  }
}
```

Čítače/časovače

- čítače v 8051 - **CT0** (volný) a **CT1** (pro sériový kanál)
- 4 režimy: 0 = kompatibilita s 8048
  1 = 16-bitový čítač
  2 = 8-bitový čítač s auto-reload
  3 = speciální spojení **CT0** a **CT1**
- čítání probíhá v registrech **THx** a **TLx** (x = 0 nebo 1)
- čítače čítají vzhůru
- při přetečení se nastaví příznak **TFx** a vyvolá přerušení, pokud je povoleno

while (!TF0);

- čekání, dokud čítač nenapočítá maximum = nepřeteče

TF0=0;

- při interruptu se příznak přetečení nuluje automaticky, zde je nutno jej "shodit" programově

TH0 = ...; TL0 = ...;

- znovu nastavení čítače, jinak by počítal od 0
- hodnota znamená: *napočti 50000x do přetečení*

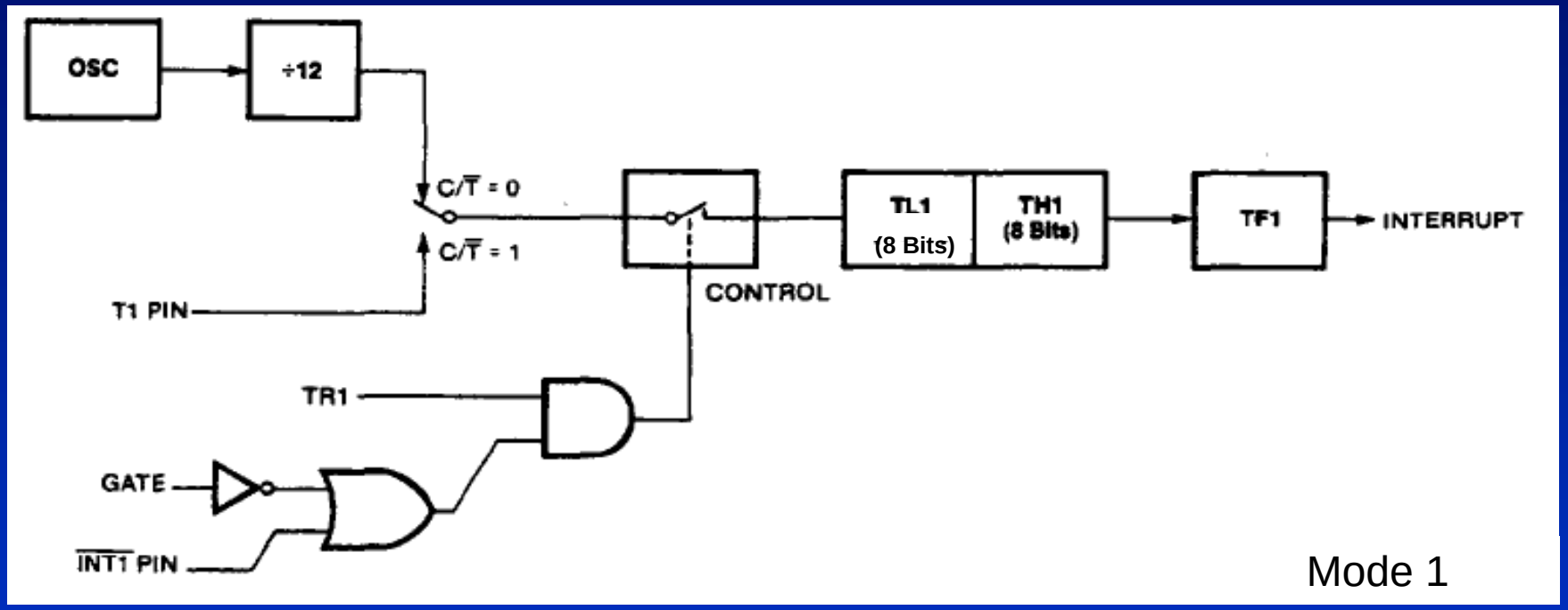
P1 = ~P1;

- bitová negace hodnoty

/256 a %256

- registry neleží vedle sebe, je nutno naplnit zvlášť horní a dolní polovinu 16-bitového slova
- C51 to efektivně přeloží jako "vezmi horní" a "dolní" bajt 16-bitové hodnoty

HW timer/counter 8051 - Mode 1



Mode 1

Registry řízení čítačů/časovačů

TMOD – řízení čítačů

- horní 4 bity řídí **CT1**, dolní pak **CT0**

D7	D6	D5	D4	D3	D2	D1	D0
GATE1	**C/T1**	**M11**	**M10**	**GATE0**	**C/T0**	**M01**	**M00**

- **GATEx** – povolení "hradlování" čítání vstupem (pinem) **INTx**
- **C/Tx** – 0 = čítač vnitřních hodin/12, 1 = čítač sestupnou hranou pinu **Tx**
- **M1x**, **M0x** – režim čítače vyjádřen bitově (0-3, takže 00 - 11)
- ! registr není bitově přístupný, nutno zachovávat stav druhého čítače, např.:

```
TMOD = (TMOD & 0xF0) | 0x06;  // mod 2 pro citac CT0 s externim vstupem
```

TCON – stavy čítačů + přerušení

- horní 4 bity sledují stavy čítačů, dolní pak aktivní přerušení, resp. nastavují jeho vlastnosti
- registr je bitově přístupný

D7	D6	D5	D4	D3	D2	D1	D0
TF1	**TR1**	**TF0**	**TR0**	**IE1**	**IT1**	**IE0**	**IT0**

- **TFx** – příznak přetečení čítače
- **TRx** – povolení běhu čítače
- **IEx** – nastaven při detekci vnějšího přerušení
- **ITx** – vnější přerušení spouštěno sestupnou hranou (=1) nebo hladinou L (=0)

Využití přerušení u časovače

Přerušení

- není efektivní čekat na událost, je lepší si od ní nechat vyvolat svoji funkci

```
#include <reg51.h>

void timer0(void) interrupt 1
{
  TH0 = (65536-50000) / 256;
  TL0 = (65536-50000) % 256;

  P1 <<= 1; P1 |= 0x01;

  if (!~P1)      // P1 == 0xff
    P1 = 0xfe;   // 1111 1110
}

int main (void)
{
  TMOD = 0x01;
  TR0 = 1;

  ET0 = 1;
  EA = 1;

  while (1)
    ;
}
```

Registr řízení přerušení

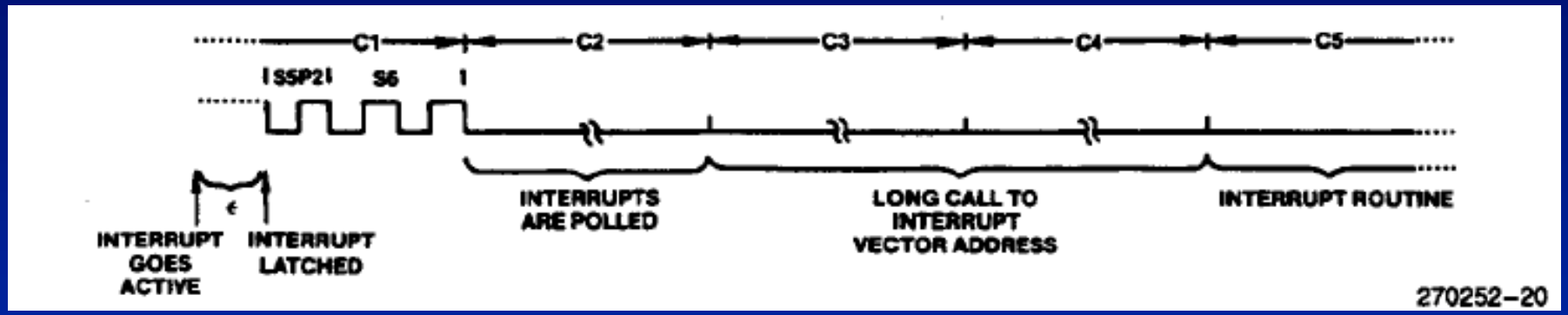
- registr **IE** - jednotlivé bity povolují přerušení
- globálně všechna přerušení povoluje ještě bit **EA**
- **ETx** přerušení od čítače/časovače
- **EXx** přerušení od vnějších vstupů
- **ES** přerušení od sériového kanálu
- nejvyšší prioritu má **EX0**, nejnižší **ES**
- případně možno změnit v registru **IP**

D7	D6	D5	D4	D3	D2	D1	D0
EA			**ES**	**ET1**	**EX1**	**ET0**	**EX0**

Obsluha přerušení

- funkce typu void bez parametrů s klíčovým slovem **interrupt** v hlavičce
- číslo odpovídá bitu v registru **IE**
- překladač jej vnitřně přepočte na skutečné adresy přerušovacích vektorů
- neexistuje kontrola na úrovni překladače, zda je korektně obslouženo přerušení, které se povoluje !!

Přerušení mikrokontroleru 8051



Interrupt Source	Vector Address
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI & TI	0023H
TF2 & EXF2	002BH

C51 - Indexy přerušovacích vektorů

Interrupt Number	Interrupt Description	Address
0	EXTERNAL INT 0	0003h
1	TIMER/COUNTER 0	000Bh
2	EXTERNAL INT 1	0013h
3	TIMER/COUNTER 1	001Bh
4	SERIAL PORT	0023h

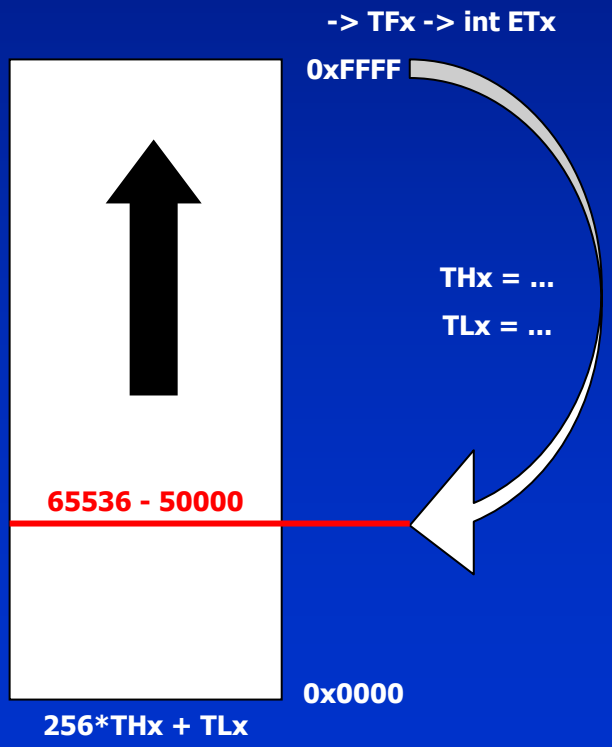
Interrupt Number	Address
0	0003h
1	000Bh
2	0013h
3	001Bh
4	0023h
5	002Bh
6	0033h
7	003Bh
8	0043h
9	004Bh
10	0053h
11	005Bh
12	0063h
13	006Bh
14	0073h
15	007Bh

Interrupt Number	Address
16	0083h
17	008Bh
18	0093h
19	009Bh
20	00A3h
21	00ABh
22	00B3h
23	00BBh
24	00C3h
25	00CBh
26	00D3h
27	00DBh
28	00E3h
29	00EBh
30	00F3h
31	00FBh

Režimy čítačů/časovačů

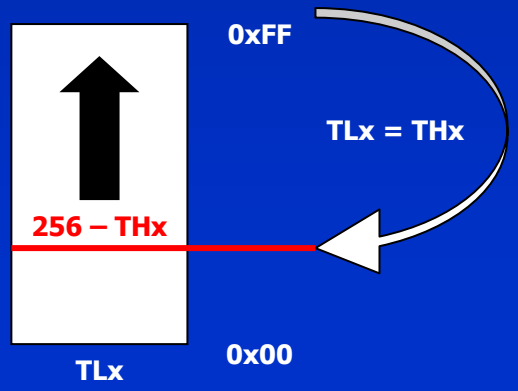
16-bitový čítač – režim 1

- počítá vzhůru v **THx** a **TLx**, po přetečení nastaví **TFx**, případně vyvolá přerušení
- nové nastavení hodnoty (reload) nutno provést manuálně
- určitá prodleva mezi přetečením a nastavením, u přerušení záleží na právě vykonávané instrukci a na nutnosti např. ukládat registry procesoru do zásobníku
- případná korekce náročně vyčíslitelná, závislá navíc na konkrétním sestavení programu

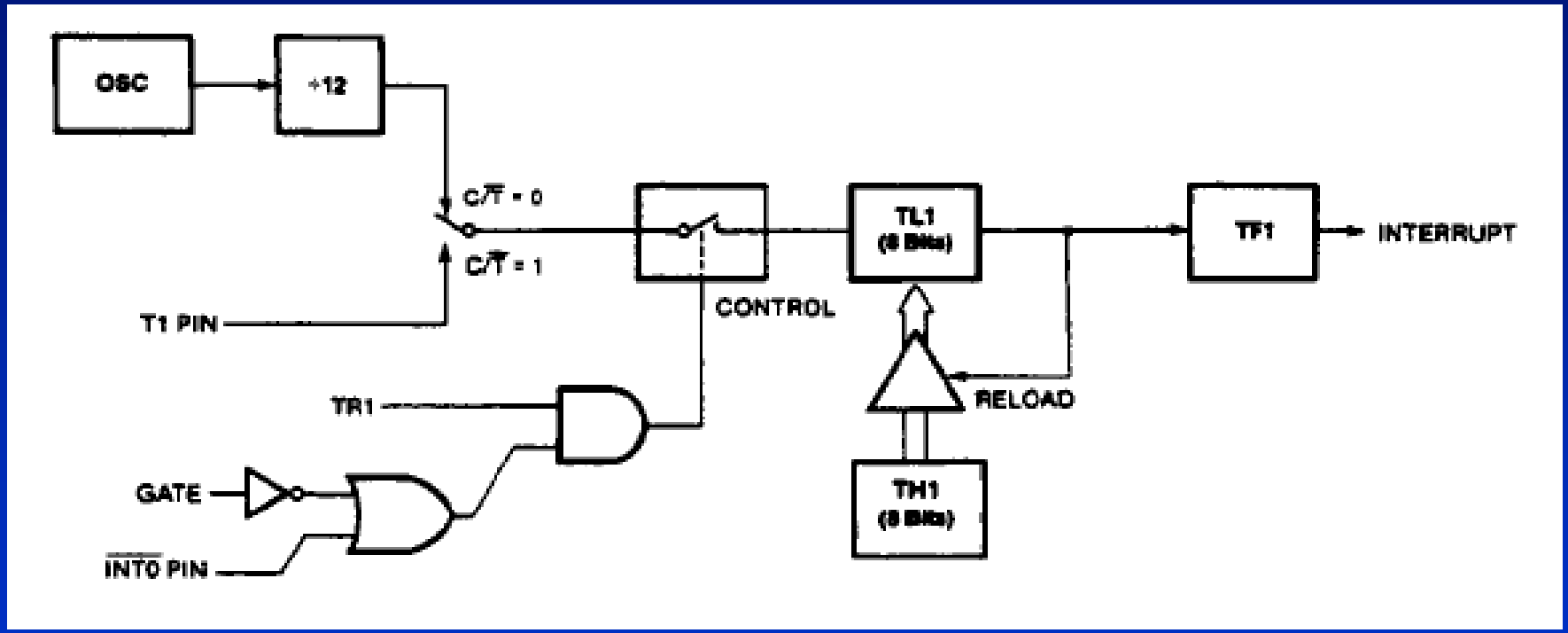


8-bitový čítač – režim 2

- počítá vzhůru jen v **TLx**
- reload automatický obsahem **THx**
- žádná SW režie a tedy žádná prodleva při reloadu
- vhodné pro přesné měření času
  - pro 8MHz je krok čítače 1.5us
  - výhodný 12MHz krystal, pak je jeden krok čítače 1us



HW timer/counter 8051 - Mode 2



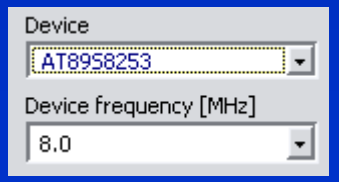
Časovač v 8-bitovém režimu

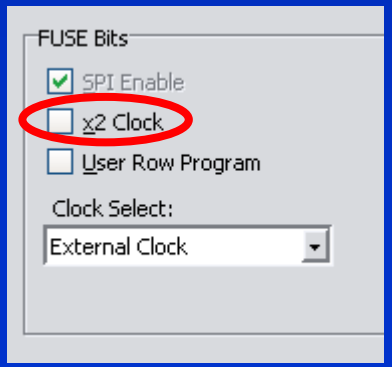
```
#include <reg51.h>

void timer0(void) interrupt 1
{
  static int i = 0;

  i++;
  if (i < 1000)
    return;
  i = 0;

  P1 <<= 1; P1 |= 0x01;
  if (!~P1)      // P1 == 0xff
    P1 = 0xfe;   // 1111 1110
}

int main(void)
{
  TMOD = 0x02;    // CT0 jako 8-bitovy citac
  TH0 = 256 - 100;  // reload hodnota 100us
  TR0 = 1;

  ET0 = 1; // povol preruseni od CT0
  EA = 1;  // povol preruseni globalne
  while (1)
    ;
}
```

static int i = ...;
- není-li řečeno jinak, proměnné jsou **automatické** = vznikají na počátku bloku a zanikají na jeho konci
- proměnná **statická** vzniká na počátku programu a její "životnost" je po celou dobu běhu
- samozřejmě platí "lokálnost" = proměnná je "viditelná" pouze v rámci svého bloku

- zde v proměnné **i** napočítáme 999 průchodů přerušením a až ten 1000. se provede akce
- nezapomenout vynulovat **i** každý 1000. průchod !

Možnost zrychlení vykonávání
- interní zdvojení hodinového taktu





Příklad stavového automatu

Zdrojový text dostupný na **z:\temp\stav51.c**

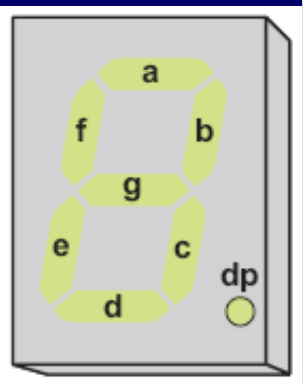
```
#include <reg51.h>

int stav = 3;
void timer0(void) interrupt 1
{
  static int counter = 0;

  counter++;
  if (counter < 500)
    return;
  counter = 0;

  switch(stav)
  {
    case 0:
      if (P0 == 0xff)
      {
        P1 = 0xff; P0 = 0xfe;
      }
      else
      {
        P0 <<= 1; P0 |= 1;
      }

      if (P0 == 0x7f)
        stav = 1;
      break;
    case 1:
      P0 = 0xff; P1 = 0x7f;
      stav = 2;
      break;
    case 2:
      P1 = 0xff; P2 = 0x7f;
      stav = 3;
      break;
```

```
    case 3:
      if (P3 == 0xff)
      {
        P2 = 0xff; P3 = 0x7f;
      }
      else
      {
        P3 >>= 1; P3 |= 0x80;
      }

      if (P3 == 0xfe)
        stav = 4;
      break;
    case 4:
      P3 = 0xff; P2 = 0xfe;
      stav = 5;
      break;
    case 5:
      P2 = 0xff; P1 = 0xfe;
      stav = 0;
      break;
  }
}

int main(void)
{
  P0 = 0xff; P1 = 0xff;
  P2 = 0xff; P3 = 0xff;
  stav = 3;

  TMOD = 0x02;
  TH0 = 256 - 100;
  TR0 = 1;

  ET0 = 1;
  EA = 1;
  while(1)
    ;
}
```

Plán cvičení

1. Úvod a seznámení – lab., IDE Keil uVision, první program
2. HW – porty, časovač, přerušení
3. **HW – displej LED**
4. HW – sériový port – úvod
5. HW – sériový port – přerušení, kruhový buffer
6. HW – LCD displej
7. Samostatná práce – zužitkování probraných znalostí
8. Sam. práce – dokončení. Grafický LCD
9. HW – PWM generované programově
10. Watchdog a práce s ním
11. Samostatná práce 2
12. Samostatná práce 2
13. Samostatná práce + Zápočet

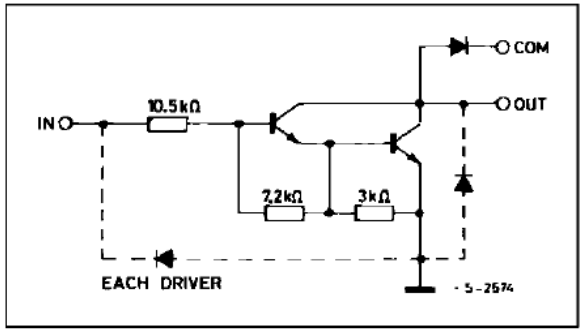
7-segmentový LED displej

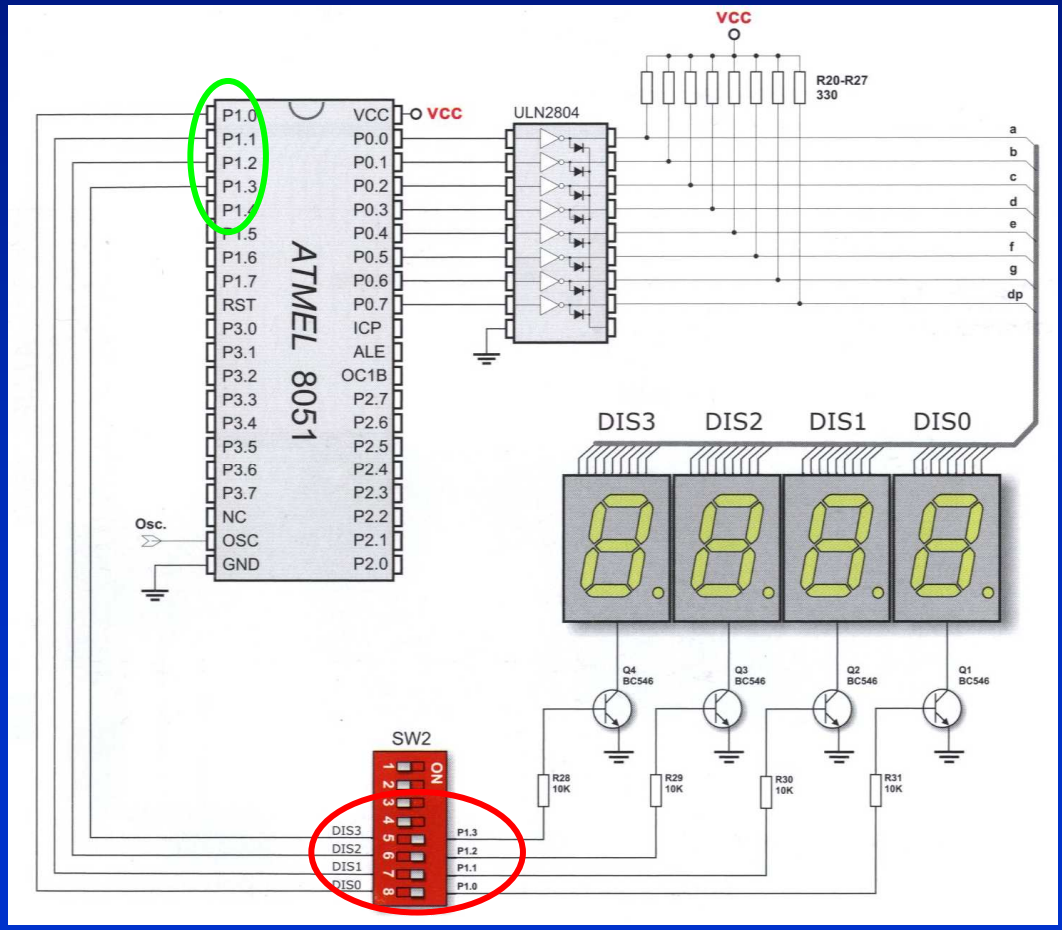


Zapojen v režimu multiplexu

- každá číslice aktivována **log.1** (**!!**) na společně katodě
  - spodní 4-bity brány **P1 (P1.0 - 4)** - nastavit **SW2** !
- zobrazené číslo určeno bitovým vzorem brány **P0**
  - viz. segmenty (opět svítí log.0)
- segmenty spínány přes budič
- 8xDarlingtony - **ULN2804**
- brána P0 využita i grafickým LCD
  - **nutno vyndat/odpojit**

For ULN2804A (each driver for 6-15 V CMOS/PMOS





Princip dynamického zobrazování

- Cyklicky se rozsvěcí jednotlivé pozice displeje (7-segmentovky)
  - každá svítí krátký čas – zde střída 1:4
  - rychlost přepínání tak vysoká, aby oko nepozorovalo blikání
  - optimálně využít přerušení od časovače
  - nutno v programu držet „paměť zobrazení"
  - cyklicky se vybírá obsah paměti a aktivuje výstupy segmentů pro příslušnou segmentovku
  - nezapomenout „minulou" zhasnout a novou aktivovat až po ustálení výstupu „segmentů" (pro x51 s instrukčním cyklem stovek ns zanedbatelné)
- Výhody principu
  - menší spotřeba (jen jedna segmentovka)
  - potřeba méně vývodů (zde 4+7)
  - efektivně použitelné do 8 pozic (střída 1:8, potřeba vyšší proud)

7-segment LED - kód programu

```
#include <reg51.h>

code unsigned char gen[10] = {
  0xc0,     // 0 - 1100 0000
  0xf9,     // 1 - 1111 1001
  0xa4,     // 2 - 1010 0100
  0xb0,     // 3 - 1011 0000
  0x99,     // 4 - 1001 1001
  0x92,     // 5 - 1001 0010
  0x82,     // 6 - 1000 0010
  0xf8,     // 7 - 1111 1000
  0x80,     // 8 - 1000 0000
  0x90      // 9 - 1001 0000
};

char buf[4] = {3, 2, 1, 0};  // VRAM

void timer0(void) interrupt 1
{
  static unsigned char u = 0;// znaky
  static int i = 0;          // cas

  P1 &= 0xF0;    // zhasnout vsechny
  P0 = gen[buf[u]];  // bitovy vzor
  P1 |= 1 << u;  // zapni jeden digit
  u++;           // pocitadlo znaku
  if (u > 3)     // vice nez mame ?
    u = 0;       // opet od zacatku

  i++;           // pocet intu
  if (i < 2000)
    return;

  i = 0;    // teprve kazdy n-ty
            // pocitej v leve cislici:
  buf[3] = (buf[3] + 1) % 10;
}

int main(void)
{
  TMOD = 0x02;       // CT0 v rezimu 2
  TH0 = 256 - 200;   // 200us
  TR0 = 1;           // citac bezi

  ET0 = 1;           // povol preruseni
  EA = 1;

  while(1)
    ;
}
```

Úpravy programu pro displej LED

- V každém zobrazovacím přerušení se 2x použije přístup do tabulky - **buf[gen[u]]**
  - málo efektivní
  - lépe v **buf** (=VRAM) ukládat přímo kódy znaků získané v okamžiku „požadavku zobrazení“
  - přístup do **gen** je tedy „málo častý“ = dle potřeby
- Rozšířit generátor na znaky A-F a „pomlčku“
  - pozor na nastavenou velikost pole **gen**

```
...
code unsigned char gen[]
= {
  ...
  0x88, // A - 1000 1000
  0x83, // b - 1000 0011
  0xC6, // C - 1010 0110
  0xA1, // d - 1010 0001
  0x86, // E - 1000 0110
  0x8E, // F - 1000 1110
  0xBF  // - - 1011 1111
};

... interrupt 1
...
 P0 = buf[u];
...
```

Úpravy programu pro displej LED - 2

- Doplněna funkce pro výpis hexadecimální hodnoty:

```
void write_buf_hex(unsigned char val)
{
  buf[0] = gen[val & 0x0f];  // stejne jako val%16
  buf[1] = gen[val >> 4];    // stejne jako val/16
}
```

- Doplněna funkce inicializace počátečního obsahu VRAM

```
void init_buf(void)
{
  buf[0] = buf[1] = buf[2] = buf[3] = gen[0];
}
```

Úpravy programu pro displej LED - 3

- Synchronizace akce v **main** podle interruptu
  - globální proměnná **tick**
  - **tick** nastavena v „akci" interruptu
  - **tick** testovaná v **main** a podle ní inkrementováno počítadlo a vypsáno na displej

```
...
bit tick = 0;     // globalni - priznak tiknuti
void timer0(void) interrupt 1
{
...
  i = 0;        // teprve kazdy n-ty
  tick = 1;     // uplynul cas - 2000x interrupt
}

int main(void)
{
  unsigned char x = 0;     // lokalni pocitani

  init_buf();              // inicializace VRAM
...
  while(1)        // hlavni smycka
  {
    if (tick)     // uplynul cas ?
    {
      tick = 0;   // zrusit priznak

      x++;        // pocitadlo
      write_buf_hex(x);  // zobrazit
    }
  }
}
```

Úpravy programu pro displej LED - 4

- Doplněna funkce pro výpis dekadické hodnoty:

```
void write_buf_dec(unsigned char val)
{
  if (val < 100)
  {
    buf[2] = gen[val % 10];
    buf[3] = gen[val / 10];
  }
  else
  {
    buf[2] = gen[16];
    buf[3] = gen[16];
  }
}
```

- Možné optimalizace
  - vylepšení výpisu „neplatné“ hodnoty
  - druhý parametr určující pozici, na kterou vypsat
  - přidat diagnostiku návratovou hodnotou – „nevypsatelná data“
  - stejně vylepšit i „výpis hex“

Námět na samostatnou práci

- Jednoduché hodiny
  - tikají po 1 sec
  - 2 proměnné v **main** - **vteřiny** a **minuty**
  - vyřešit přetékání po 59 (!)
  - zobrazovat pomocí **write_buf_dec** na příslušné pozici LED displeje

Plán cvičení

1. Úvod a seznámení – lab., IDE Keil uVision, první program
2. HW – porty, časovač, přerušení
3. HW – displej LED
4. **HW – sériový port – úvod**
5. HW – sériový port – přerušení, kruhový buffer
6. HW – LCD displej
7. Samostatná práce – zužitkování probraných znalostí
8. Sam. práce – dokončení. Grafický LCD
9. HW – PWM generované programově
10. Watchdog a práce s ním
11. Samostatná práce 2
12. Samostatná práce 2
13. Semestrální práce 2 + Zápočet

```
...
int main(void)
{
  unsigned char sec = 0, min = 0;
  ...
  while(1)
  {
    if (tick)
    {
      tick = 0;

      sec++;
      if (sec >= 60)
      {
        sec = 0;
        min++;

        if (min >= 60)
          min = 0;
      }

      buf[0] = gen[sec % 10];
      buf[1] = gen[sec / 10];

      buf[2] = gen[min % 10];
      buf[3] = gen[min / 10];
    }
  }
}
```

Domácí úkol - hodiny

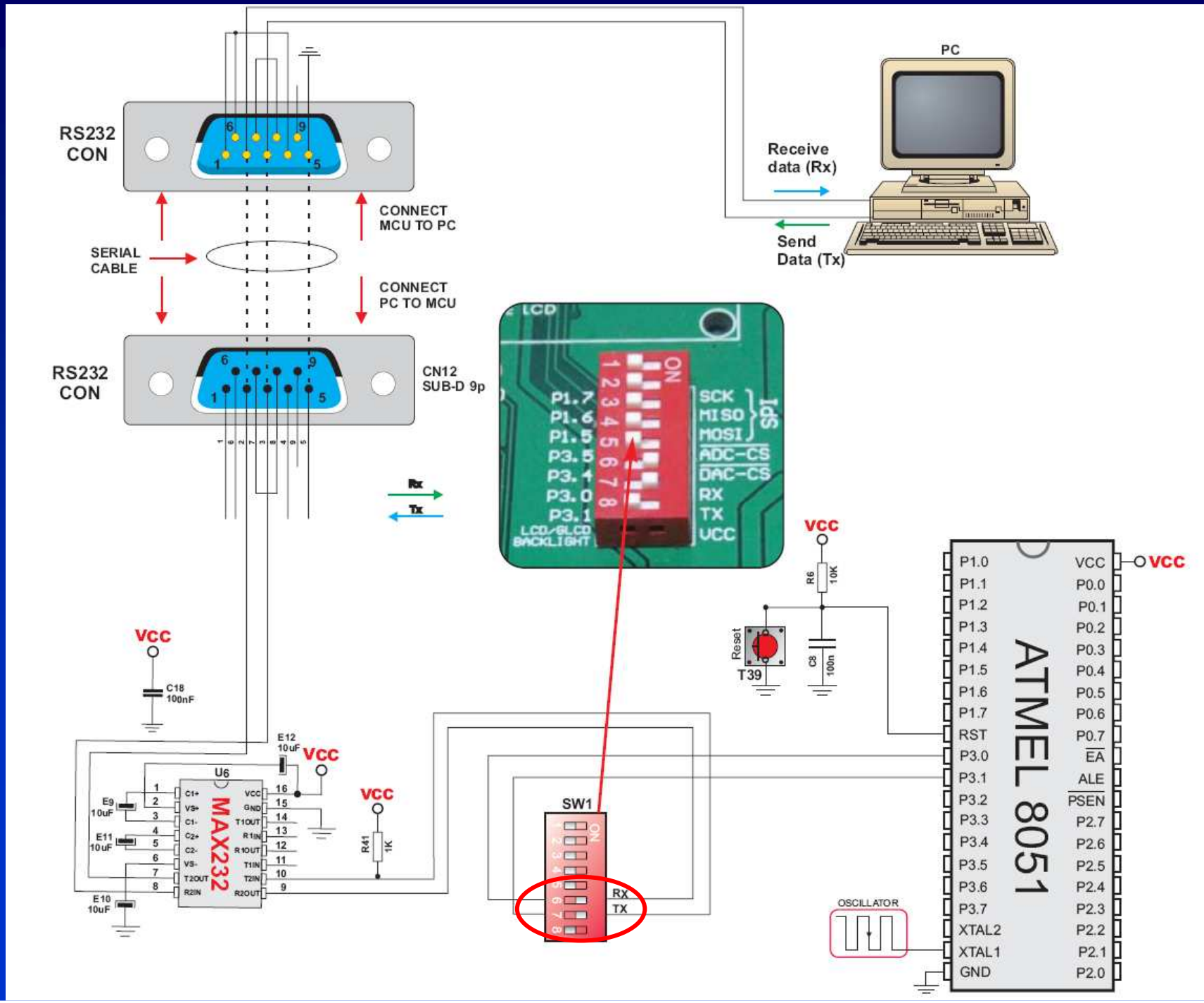
- Časování
  - počet intů – 5000 (x200µs)
  - pro otestování možno zrychlit snížením
- Kód zjednodušen
  - funkčnost **write_buf_dec** vložena přímo do **main**
- Možná optimalizace
  - měnit **buf** pro minuty jen při změně minut

Sériový port – základ komunikace

```
#include <reg51.h>
#include <stdio.h>

int main(void)
{
  unsigned char u;

  TMOD = 0x20;              // CT1 rezim 2 = 8-bitovy citac
  TH1 = 0xF3;               // pro rychlost 2400/4800 Bd pri 12MHz krystalu
  TR1 = 1;                  // start citace
  SCON = 0x52;              // standardni nastaveni serioveho kanalu
  PCON |= 0x80;             // SMOD=1 => zdvojeni seriove rychlosti (na 4800)

  puts("\nStart\n");        // uvodni "textik"
  while (1)
  {
    u = getchar();          // precti znak ze ser. linky
    P1 = u;                 // zobraz binarne na brane P1 (8x LED)
    putchar(u + 1);         // posli zpet po ser. lince do terminalu
  }
}
```

#include <stdio.h>

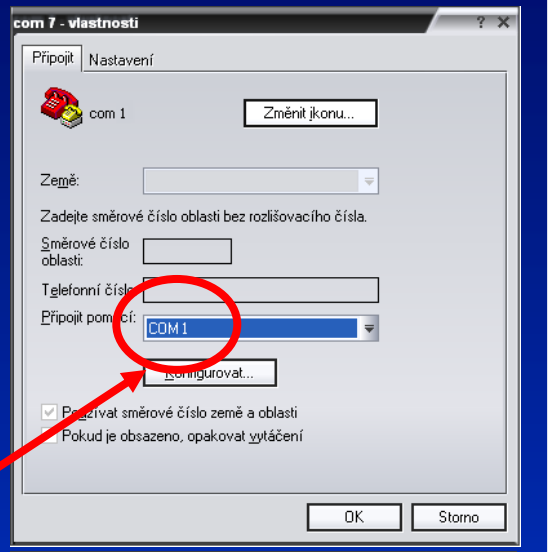
- standardní vstup/výstup probíhá prostřednictvím sériové linky a připojeného terminálu
- použitelné běžné funkce **putchar**, **getchar**, **puts**, **printf**, ...
- neřeší nastavení portu a přenosové rychlosti !

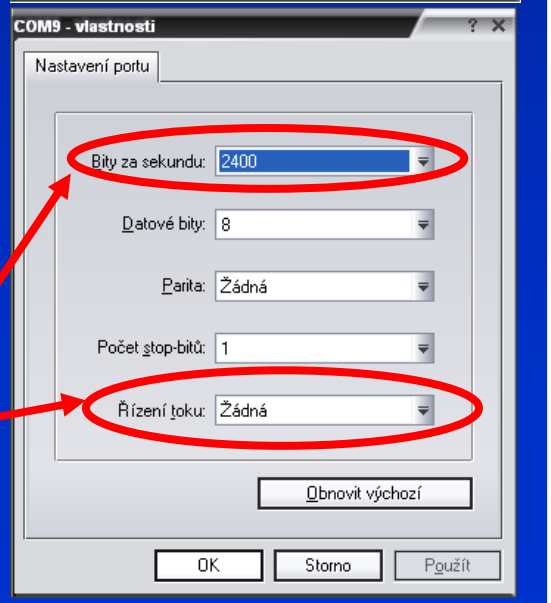

RS232
CON
SERIAL
CABLE
CONNECT
MCU TO PC
CONNECT
PC TO MCU
RS232
CON
CN12
SUB-D 9p
PC
Receive
data (Rx)
Send
Data (Tx)
Rx
Tx
LCD
ON
P1.7
P1.6
P1.5
P3.5
P3.4
P3.0
P3.1
SCK
MISO
MOSI
SPI
ADC-CS
DAC-CS
RX
TX
VCC
LCD/GLCD
BACKLIGHT
VCC
R6
10K
Reset
T39
C8
100n
VCC
C18
100nF
E12
10uF
VCC
U6
MAX232
E9
10uF
E11
10uF
E10
10uF
VCC
R41
1K
SW1
RX
TX
ATMEL 8051
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
RST
P3.0
P3.1
P3.2
P3.3
P3.4
P3.5
P3.6
P3.7
XTAL2
XTAL1
GND
VCC
P0.0
P0.1
P0.2
P0.3
P0.4
P0.5
P0.6
P0.7
EA
ALE
PSEN
P2.7
P2.6
P2.5
P2.4
P2.3
P2.2
P2.1
P2.0
VCC
OSCILLATOR

Hyperterminal - SW na PC

- Menu Start / Příslušenství / Komunikace / **Hyperterminál**
- Ikonka **COM 1** na ploše (někde je i více variant)
- Pozor, název portu nesmí být jménem souboru, zvolit např. "COM 1" (s mezerou)

- Připojení k portu
- Odpojení od portu
- **Nastavení komunikace** - **COM1** (fyzické zařízení)
- **Vlastnosti komunikace:**
  - **4800b/s**
  - 8 datových bitů
  - bez parity
  - 1 stop-bit
  - řízení toku **ŽÁDNÁ** (!) (defaultně "handshaking")





Nastavení seriového portu x51

SCON – řízení sériového kanálu

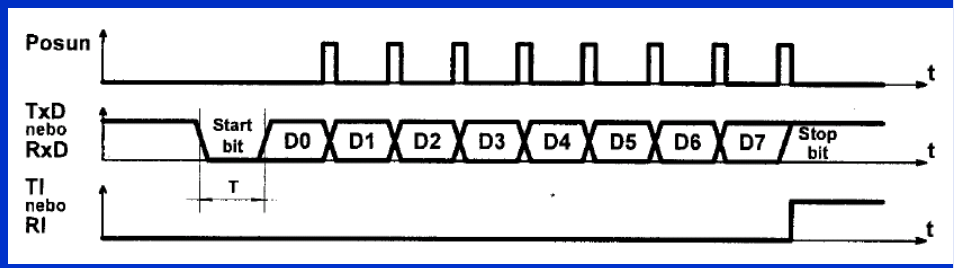
D7	D6	D5	D4	D3	D2	D1	D0
SM0	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**

- **SM0-1** – režim sériového kanálu
  - mód 0 – 00 – 8-bitový synchronní přenos na **RxD**, hodiny na **TxD** ($f_{osc}/12$)
  - mód 1 – 01 – 8-bitový asynchronní přenos na **RxD**, **TxD**, 1 start-bit (0), 1 stop-bit (1)
    - bitová rychlost dána přetečením čítače **CT1** dělená 32 nebo 16 podle **SMOD**
  - mód 2 – 10 – 9-bitový UART na **RxD**, **TxD**, rychlost $f_{osc}/32$ nebo $f_{osc}/64$ podle **SMOD**
  - mód 3 – 11 – 9-bitový UART na **RxD**, **TxD**, rychlost přetečením **CT1**

$$\text{rychlost} = \frac{2^{SMOD}}{32} \cdot \frac{f_{OSC}}{12 \cdot (256 - TH1)} \qquad TH1 = 256 - \frac{2^{SMOD}}{32} \cdot \frac{f_{OSC}}{12 \cdot \text{rychlost}}$$

- **SM2** – povolení víceprocesorové komunikace v 9-bitových režimech
- **REN** – povolení příjmu
- **TB8**, **RB8** – 9-bit při vysílání/příjmu
- **TI** – příznak prázdného vysílacího posuvného registru (při stop-bitu)
- **RI** – příznak přijatého znaku (bajtu)

- případné přerušení společné pro **RI**/**TI**
- nutno programově zjistit, čím bylo vyvoláno
- (**!!**) funkce z std. knihovny „neumí" využívat přerušení – nelze využívat společně



Seriový port - pokračování

PCON – řízení napájení

- nejvyšší bit slouží k zdvojení rychlosti sériové komunikace (**SMOD** = 1)
- režimy snížené spotřeby pouze pro CMOS obvody
- vylepšené klony přidávají další režimy/bity

D7	D6	D5	D4	D3	D2	D1	D0
SMOD	**-**	**-**	**-**	**-**	**-**	**PD**	**IDL**

- **IDL** – režim Idle – zastavení procesorového jádra, periférie běží, probuzení přerušením
- **PD** – režim Power-Down – zastaví se vše včetně oscilátoru, zachován pouze obsah RAM a SFR, probuzení pouze RESETem

---

Vizuální kontrola komunikace

- **RxD** na portu **P3.0**, **TxD** na portu **P3.1**
- kontrola v poli LED (musí být povolené port P3 na LED-ky)
- po RESETu blikne **TxD** (= úvodní zpráva), při příjmu znaku RxD i TxD (znak + odpověď)

---

u = getchar();

- zobrazují se 2 znaky na terminálu – funkce **getchar()** provádí automaticky "echo"
- náhrada speciální funkcí C51 – **u = _getkey();** - ta již "ne-echuje„ - ověřit

Sériový port – posílání znaků

```
#include <reg51.h>
#include <stdio.h>

void init_serial()  // 4800 pri 12 MHz
{
  TMOD = (TMOD & 0x0F) | 0x20;  // POUZE CT1 !!
  TH1 = 0xF3;         // rychlost 2400/4800
  SCON = 0x52;        // rezim serioveho kanalu
  PCON |= 0x80;       // SMOD=1 => 4800b/s
  TR1 = 1;            // start citace
}

int main(void)
{
  unsigned char u;

  init_serial();
  puts("\nStart\n");

  while (1)
  {
    putchar(u);   // posli znak
    u++;
  }
}
```

- Výstup po znacích
  - všechny ASCII znaky (8-bitové)
  - pípnutí je znak BELL (kód 7)
  - přepis od začátku okna – zřejmě znak FF (FormFeed) = nová stránka (kód 12)

Sériový port - printf

```
...
  while (1)
  {
//    putchar(u);
    printf("%c", u);    // posli znak
    u++;
  }
}
```

```
Build target 'Target 1'
linking...
Program Size: data=10.0 xdata=0 code=170
creating hex file from "cviceni04"...
"cviceni04" - 0 Error(s), 0 Warning(s).
```

```
Build target 'Target 1'
compiling cv04b.c...
linking...
Program Size: data=31.1 xdata=0 code=1157
creating hex file from "cviceni04"...
"cviceni04" - 0 Error(s), 0 Warning(s).
```

- Použití univerzální funkce **printf**
  - větší obsazená datová i programová paměť (ve srovnání s **putchar**)
  - pomalejší (musí min. zpracovat formátovací řetězec)
  - typicky se používá max. jen v době ladění
  - lépe nahradit specializovanými funkcemi pro konkrétní datové typy

```
...
  printf("%c %d\n", u, u);
...
```

- Defaultní parametr pro **%d** je 16-bitový u **printf**
  - nutno přetypovat, jinak se bere „nižší" bajt z okolní paměti a předávaný jako „vyšší" bajt
  - např. **(unsigned int)u**

```
J 18944
K 19200
L 19456
M 19712
N 19968
O 20224
P 20480
Q 20736
R 20992
S 21248
```

```
] 93
^ 94
_ 95
` 96
a 97
b 98
c 99
d 100
e 101
f 102
g 103
```

Seriový port + LED displej

```
int main(void)
{
  unsigned char u; unsigned char cnt = 0;

  init_buf();
  init_serial();
  puts("\nStart\n");

  TMOD = (TMOD & 0xf0) | 0x02;  // CT0 pouze
  TH0 = 256 - 200; TR0 = 1;  // init + run
  ET0 = 1; EA = 1;  // interrupty

  while (1)
  {
    u = _getkey();
    write_buf_hex(u);
    putchar((u >= 0x20) ? u : '?');

    cnt++;
    if (cnt >= 10)
      cnt = 0;

    buf[3] = gen[cnt];
  }
}
```

- Z minulého cvičení vzít kód LED displej
  - funkce **init_buf()**
  - funkce **write_buf_hex()**
  - funkce **timer0** – obsluha přerušení
  - pole **gen** – podoba znaků
  - pole **buf** – VRAM
- Zobrazuje se na displeji
  - HEX kód příchozího znaku – pravé 2 pozice
  - počítadlo znaků – levá pozice

Plán cvičení

1. Úvod a seznámení – lab., IDE Keil uVision, první program
2. HW – porty, časovač, přerušení
3. HW – displej LED
4. HW – sériový port – úvod, spojení s displejem
5. **HW – sériový port – přerušení + kruhový buffer**
6. HW – LCD displej
7. Samostatná práce – zužitkování probraných znalostí
8. Sam. práce – dokončení. Grafický LCD
9. HW – PWM generované programově
10. Watchdog a práce s ním
11. Samostatná práce 2
12. Samostatná práce 2
13. Samostatná práce + Zápočet

```
typedef unsigned char byte;
typedef unsigned int word;
...
bit _ready_send = 1; // odeslano
bit _ready_recv = 0; // prijato
byte _recv_data = 0; // co prislo

void serial(void) interrupt 4
{
  if (TI)  // byl od vysilani ?
  {
    TI = 0;             // shod priznak
    _ready_send = 1;    // možno dalsi
  }

  if (RI)  // byl od prijmu ?
  {
    RI = 0;             // shod priznak
    _ready_recv = 1;    // neco prislo
    _recv_data = SBUF; // RCV registr
  }
}

byte my_getchar()
{
  while(!_ready_recv)  // cekej prijem
    ;
  _ready_recv = 0;     // shod priznak

  return _recv_data;   // vrat prijate
}
```

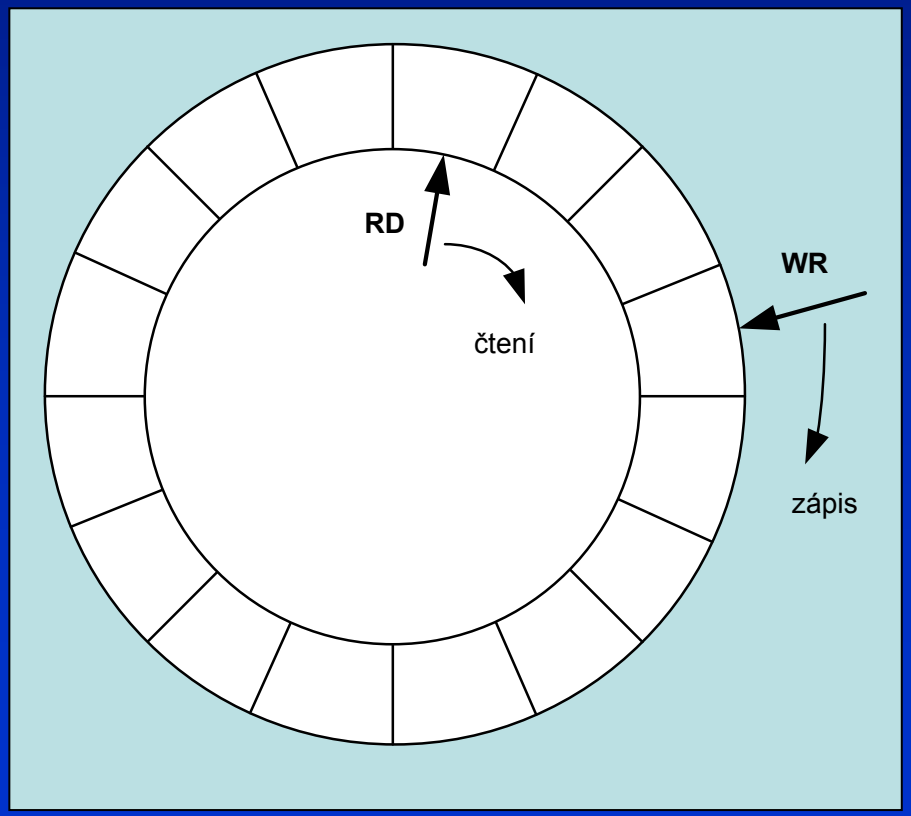
```
void my_putchar(byte u)
{
  while(!_ready_send) // cekej na
    ;                 // odvysilani
  _ready_send = 0;    // shod priznak
  SBUF = u;           // SEND register
}

void my_puts(char *cp) // retezec
{
  for(; *cp; cp++)    // znak po znaku
    my_putchar(*cp);  // posilej
}

int main(void)
{
  byte b;
  init_serial();      // inicializace

  ES = 1; EA = 1;     // povol INT-y

  my_puts("\nAhoj\n");

  while(1)            // echo s vypisem
  {
    b = my_getchar();
    P1 = b;           // na branu P1
    my_putchar(b);
  }
}
```

Sériový port a přerušení - poznámky

- Definovány nové „unsigned" datové typy pomocí **typedef**
  - úspora psaní kódu
  - často používané, doporučuji, budeme používat
- „Globální" proměnné určené pro provoz, neužívat v programu
  - funkci API (rozhraní mezi programem a komunikační částí) je pomocí funkcí typu getchar/putchar
  - funkce API testují vnitřní příznaky a příp. čekají na dokončení operace = zatím blokující mechanismus obdobný std. knihovně
- Funkce obsluhy přerušení společná pro příjem a vysílání
  - nutno rozlišit podmínkou podle **TI** nebo **RI**
  - nezapomenout vynulovat příznaky
  - **TI** se nastavuje před koncem odesílání znaku (začátek stop-bitu)
  - **RI** se nastavuje při ukončení příjmu (po stop-bitu)
- Registr **SBUF** – na jedné adrese (jméno registru)
  - přijímací registr – čtení – obsahuje platná data při nastaveném **RI**
  - odesílací registr – zápis – po vložení hodnoty začne data odesílat
- Funkce **main** téměř stejná jako pro „obyčejnou" komunikaci
  - nezapomenout povolit interrupty

Kruhový buffer - princip

Kruhový buffer
- používá se při požadavku zpracovávat přicházející data i přes případné vytížení aplikace
- plnění a čtení probíhá typicky v různých threadech
- data se ukládají do bufferu a mohou být pak dávkově zpracována, až „je na to čas"



buffer – pole hodnot s ukazateli
- **WR ukazovátko** – pozice pro zápis nové hodnoty
- každým zápisem se inkrementuje pokud není plný buffer
- **RD ukazovátko** – pozice pro čtení hodnoty
- každým čtením se inkrementuje, pokud není prázdný buffer

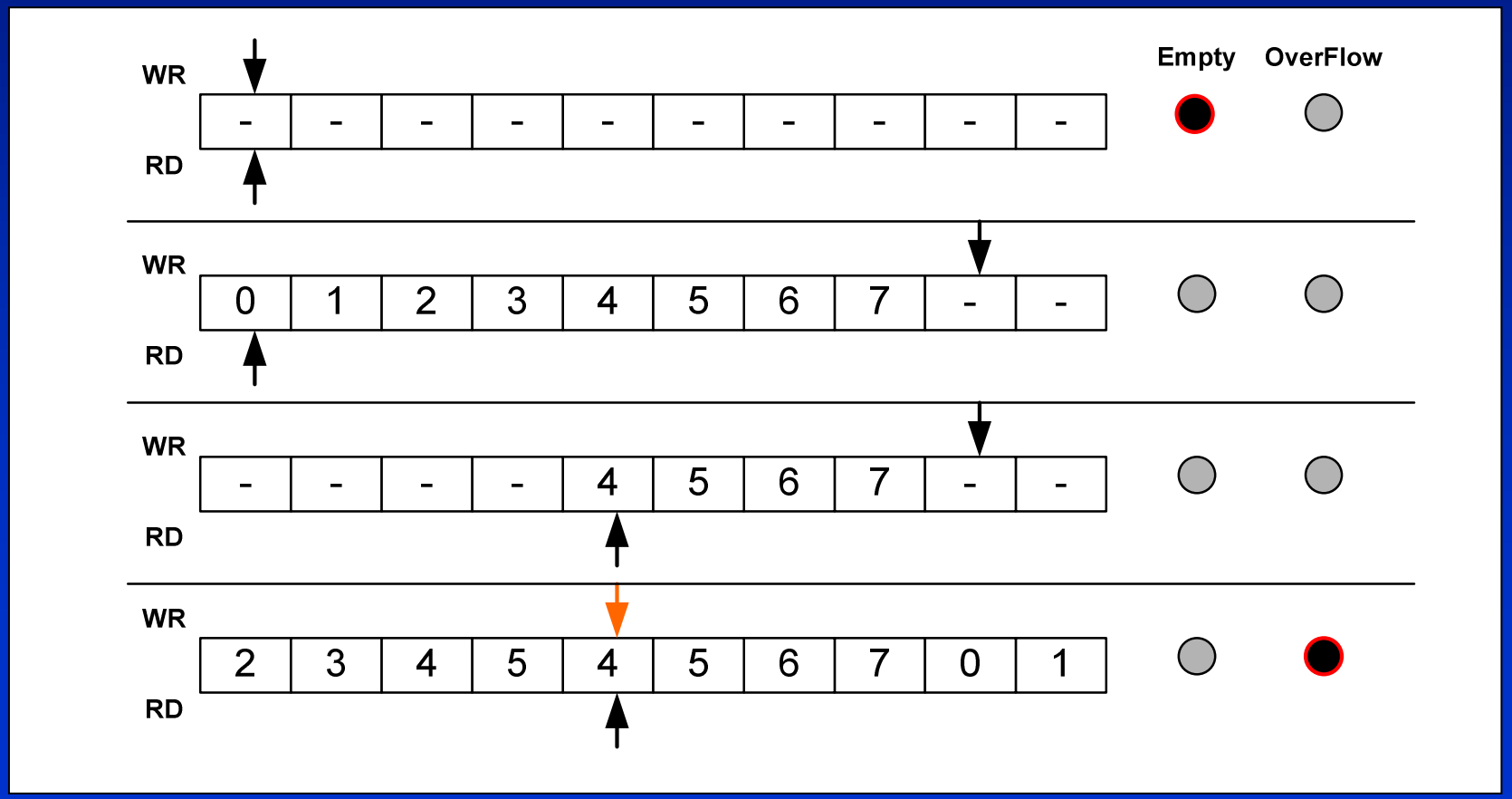
pomocný příznak empty
- = buffer je prázdný
- typicky se používá k testu, zda se v bufferu již něco objevilo a je potřeba to zpracovat

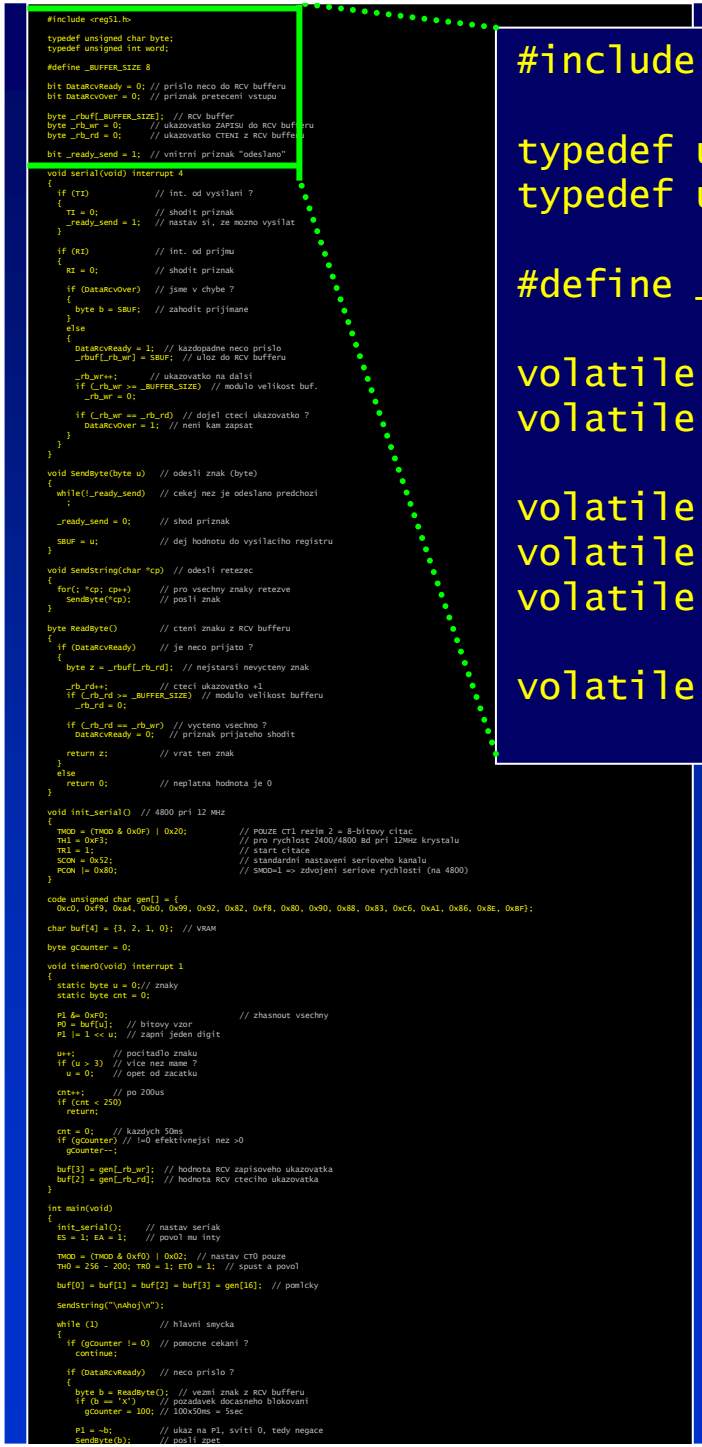
pomocný příznak overflow
- = došlo k přetečení bufferu
- typicky potřeba kontrolovat, jestli nedošlo ke ztrátě dat

Kruhový buffer - provoz

Příklad operací

- začíná se prázdným bufferem délky 10 hodnot
- naplněno 8 hodnot (pro jednoduchost čísla 0 – 7)
- odebráno 4 hodnoty
- pokus o přidání 8 nových hodnot – musí se objevit příznak **OverFlow**





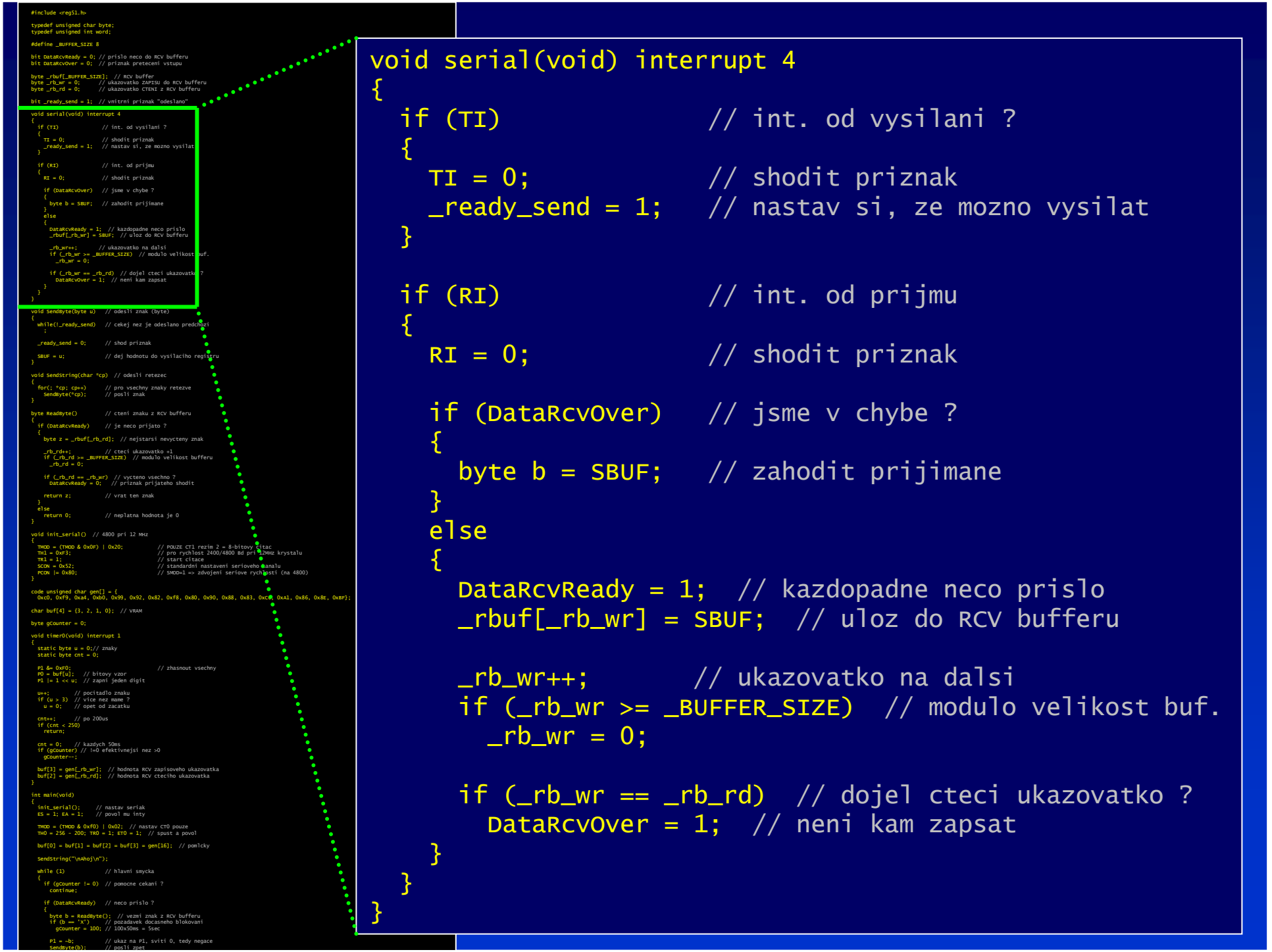
```
#include <reg51.h>

typedef unsigned char byte;
typedef unsigned int word;

#define _BUFFER_SIZE 8

volatile bit DataRcvReady = 0; // prislo neco do RCV bufferu
volatile bit DataRcvOver = 0;  // priznak preteceni vstupu

volatile byte _rbuf[_BUFFER_SIZE];  // RCV buffer
volatile byte _rb_wr = 0; // ukazovatko ZAPISU do RCV bufferu
volatile byte _rb_rd = 0; // ukazovatko CTENI z RCV bufferu

volatile bit _ready_send = 1;  // vnitrni priznak "odeslano"
```

Privátní proměnné algoritmu

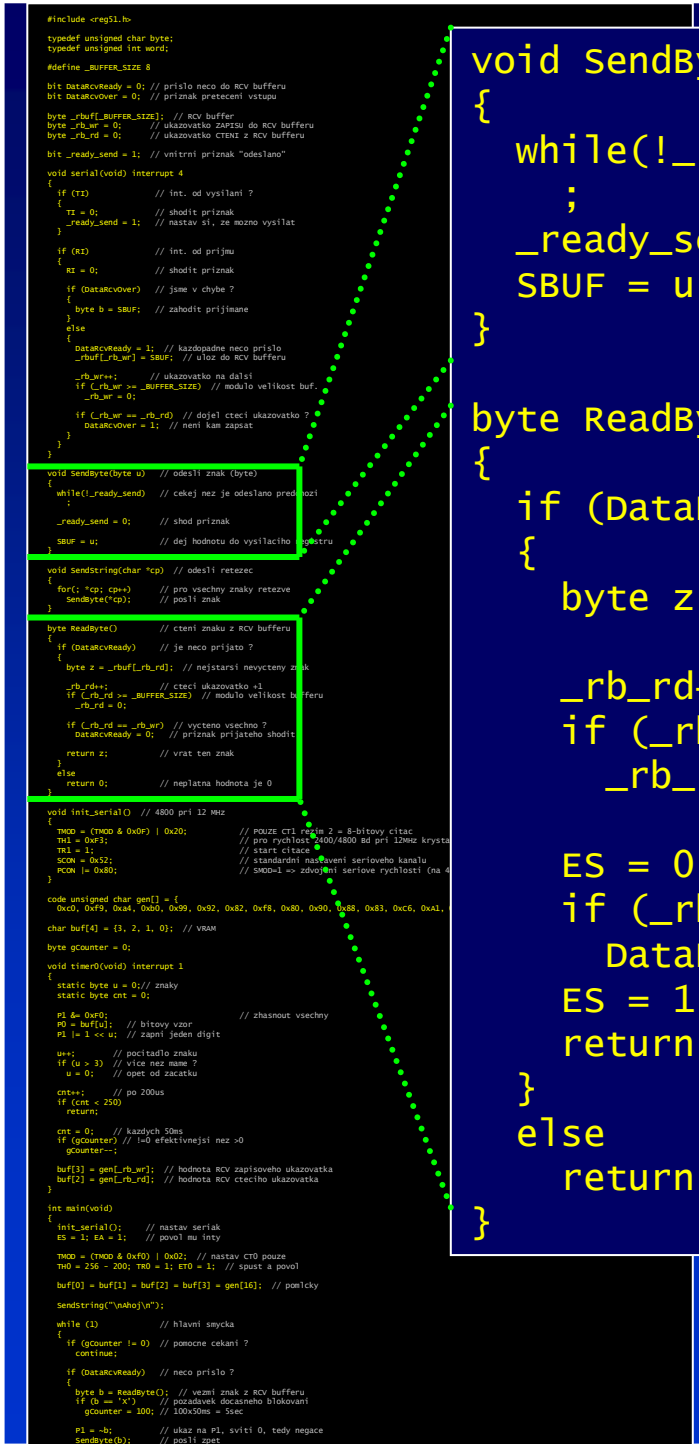
- „skryté“ za podtržítkem
- buffer pro data, ukazovátko pozice pro zápis i čtení
- odesílání zatím bez bufferu = potřebuje příznak „možno odesílat“
- velikost bufferu nadefinována symbolicky
- **volatile** = žádné optimalizace, nutné pro proměnné, ke kterým se přistupuje z přerušení i normálního kódu

Veřejné proměnné algoritmu

- typicky příznaky (stačí bitová proměnná)
- příznak alespoň jednoho přijatého znaku v bufferu
- příznak přeplnění bufferu



```
#include <reg51.h>

typedef unsigned char byte;
typedef unsigned int word;

#define _BUFFER_SIZE 8

bit DataRcvReady = 0; // prislo neco do RCV bufferu
bit DataRcvOver = 0;  // priznak preteceni vstupu

byte _rbuf[_BUFFER_SIZE];  // RCV buffer
byte _rb_wr = 0;      // ukazovatko ZAPISU do RCV bufferu
byte _rb_rd = 0;      // ukazovatko CTENI z RCV bufferu

bit _ready_send = 1;  // vnitrni priznak "odeslano"

void serial(void) interrupt 4
{
  if (TI)             // int. od vysilani ?
  {
    TI = 0;           // shodit priznak
    _ready_send = 1;  // nastav si, ze mozno vysilat
  }

  if (RI)             // int. od prijmu
  {
    RI = 0;           // shodit priznak

    if (DataRcvOver)  // jsme v chybe ?
    {
      byte b = SBUF;  // zahodit prijimane
    }
    else
    {
      DataRcvReady = 1;  // kazdopadne neco prislo
      _rbuf[_rb_wr] = SBUF;  // uloz do RCV bufferu

      _rb_wr++;       // ukazovatko na dalsi
      if (_rb_wr >= _BUFFER_SIZE)  // modulo velikost buf.
        _rb_wr = 0;

      if (_rb_wr == _rb_rd)  // dojel cteci ukazovatko ?
        DataRcvOver = 1;  // neni kam zapsat
    }
  }
}

void SendByte(byte u)  // odesli znak (byte)
{
  while(!_ready_send)  // cekej nez je odeslano predchozi
    ;

  _ready_send = 0;     // shod priznak

  SBUF = u;            // dej hodnotu do vysilaciho registru
}

void SendString(char *cp)  // odesli retezec
{
  for(; *cp; cp++)     // pro vsechny znaky retezve
    SendByte(*cp);     // posli znak
}

byte ReadByte()        // cteni znaku z RCV bufferu
{
  if (DataRcvReady)    // je neco prijato ?
  {
    byte z = _rbuf[_rb_rd];  // nejstarsi nevycteny znak

    _rb_rd++;          // cteci ukazovatko +1
    if (_rb_rd >= _BUFFER_SIZE)  // modulo velikost bufferu
      _rb_rd = 0;

    if (_rb_rd == _rb_wr)  // vycteno vsechno ?
      DataRcvReady = 0;  // priznak prijateho shodit

    return z;          // vrat ten znak
  }
  else
    return 0;          // neplatna hodnota je 0
}

void init_serial()  // 4800 pri 12 MHz
{
  TMOD = (TMOD & 0x0F) | 0x20;          // POUZE CT1 rezim 2 = 8-bitovy citac
  TH1 = 0xF3;                           // pro rychlost 2400/4800 Bd pri 12MHz krystalu
  TR1 = 1;                              // start citace
  SCON = 0x52;                          // standardni nastaveni serioveho kanalu
  PCON |= 0x80;                         // SMOD=1 => zdvojeni seriove rychlosti (na 4800)
}

code unsigned char gen[] = {
  0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90, 0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E, 0xBF};

char buf[4] = {3, 2, 1, 0};  // VRAM

byte gCounter = 0;

void timer0(void) interrupt 1
{
  static byte u = 0;// znaky
  static byte cnt = 0;

  P1 &= 0xF0;                   // zhasnout vsechny
  P0 = buf[u];   // bitovy vzor
  P1 |= 1 << u;  // zapni jeden digit

  u++;           // pocitadlo znaku
  if (u > 3)  // vice nez mame ?
    u = 0;   // opet od zacatku

  cnt++;      // po 200us
  if (cnt < 250)
    return;

  cnt = 0;    // kazdych 50ms
  if (gCounter) // !=0 efektivnejsi nez >0
    gCounter--;

  buf[3] = gen[_rb_wr];  // hodnota RCV zapisoveho ukazovatka
  buf[2] = gen[_rb_rd];  // hodnota RCV cteciho ukazovatka
}

int main(void)
{
  init_serial();     // nastav seriak
  ES = 1; EA = 1;    // povol mu inty

  TMOD = (TMOD & 0xf0) | 0x02;  // nastav CT0 pouze
  TH0 = 256 - 200; TR0 = 1; ET0 = 1;  // spust a povol

  buf[0] = buf[1] = buf[2] = buf[3] = gen[16];  // pomlcky

  SendString("\nAhoj\n");

  while (1)             // hlavni smycka
  {
    if (gCounter != 0)  // pomocne cekani ?
      continue;

    if (DataRcvReady)   // neco prislo ?
    {
      byte b = ReadByte();  // vezmi znak z RCV bufferu
      if (b == 'X')         // pozadavek docasneho blokovani
        gCounter = 100; // 100x50ms = 5sec

      P1 = ~b;          // ukaz na P1, sviti 0, tedy negace
      SendByte(b);      // posli zpet
```



```
void SendByte(byte u)    // odesli znak (byte)
{
  while(!_ready_send)    // cekej nez je odeslano predchozi
    ;
  _ready_send = 0;       // shod priznak
  SBUF = u;              // dej hodnotu do vysilaciho registru
}

byte ReadByte()          // cteni znaku z RCV bufferu
{
  if (DataRcvReady)      // je neco prijato ?
  {
    byte z = _rbuf[_rb_rd];  // nejstarsi nevycteny znak

    _rb_rd++;            // cteci ukazovatko +1
    if (_rb_rd >= _BUFFER_SIZE)  // modulo velikost bufferu
      _rb_rd = 0;

    ES = 0;              // zakazat inty seriaku
    if (_rb_rd == _rb_wr)  // vycteno vsechno ?
      DataRcvReady = 0;    // priznak prijateho shodit
    ES = 1;              // povolit inty zpet
    return z;            // vrat ten znak
  }
  else
    return 0;            // neplatna hodnota je 0
}
```

ReadByte()

- vyčte jeden byte z bufferu (pokud tam je)
- posune čtecí ukazovátko +1 (modulo velikost bufferu)

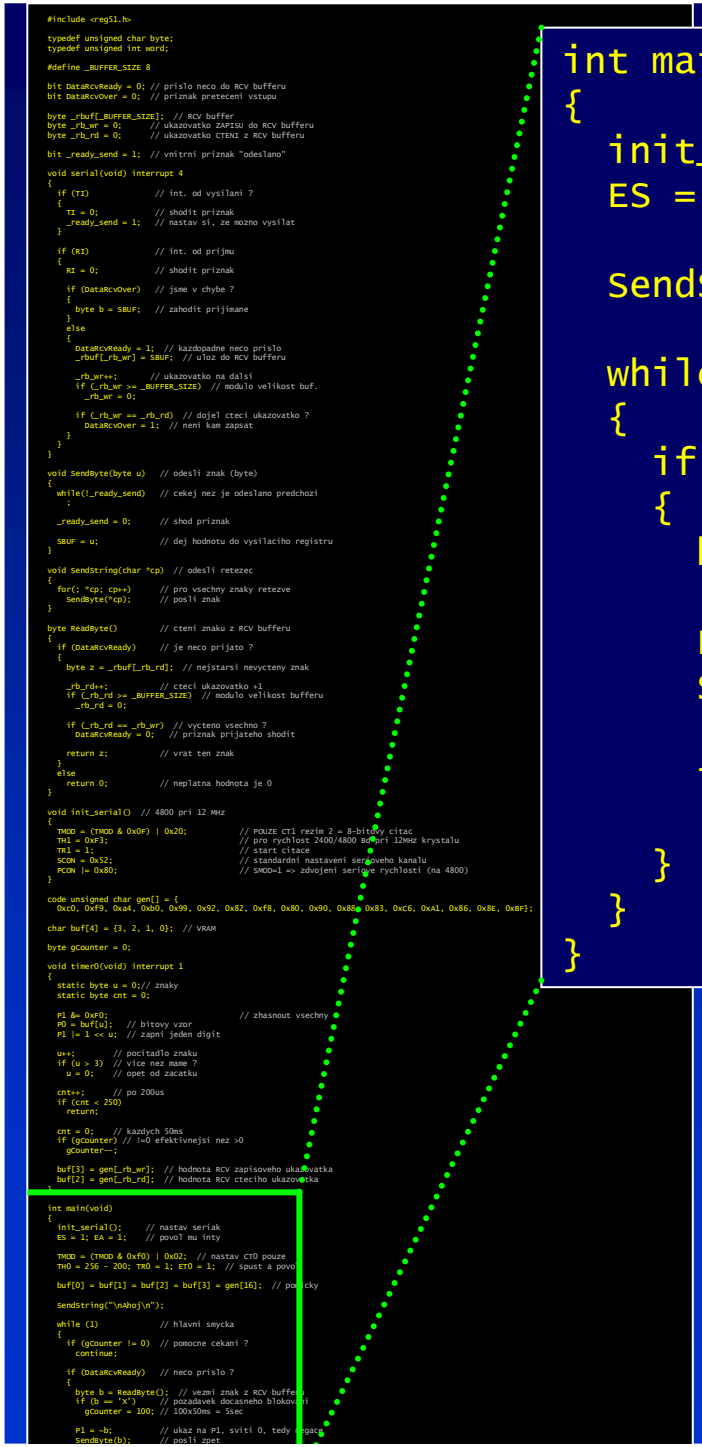
```
void SendString(char *cp)  // odesli retezec
{
  for(; *cp; cp++)         // pro vsechny znaky retezve
    SendByte(*cp);         // posli znak
}

void init_serial()  // 4800 pri 12 MHz
{
  TMOD = (TMOD & 0x0F) | 0x20;     // POUZE CT1 rezim 2 = 8-bitovy citac
  TH1 = 0xF3;                      // pro rychlost 2400/4800 Bd pri 12MHz krystalu
  TR1 = 1;                         // start citace
  SCON = 0x52;                     // standardni nastaveni serioveho kanalu
  PCON |= 0x80;                    // SMOD=1 => zdvojeni seriove rychlosti (na 4800)
}
```

Odeslání textu

- další z „veřejných“ funkcí (SendByte, ReadByte)
- v principu stále stejná, jen upravené názvosloví

Inicializace seriového portu

- již známý obsah
- pozor – neobsahuje povolení přerušení, to je až v **main**



```
#include <reg51.h>

typedef unsigned char byte;
typedef unsigned int word;

#define _BUFFER_SIZE 8

bit DataRcvReady = 0; // prislo neco do RCV bufferu
bit DataRcvOver = 0;  // priznak preteceni vstupu

byte _rbuf[_BUFFER_SIZE];  // RCV buffer
byte _rb_wr = 0;      // ukazovatko ZAPISU do RCV bufferu
byte _rb_rd = 0;      // ukazovatko CTENI z RCV bufferu

bit _ready_send = 1;  // vnitrni priznak "odeslano"

void serial(void) interrupt 4
{
  if (TI)             // int. od vysilani ?
  {
    TI = 0;           // shodit priznak
    _ready_send = 1;  // nastav si, ze mozno vysilat
  }

  if (RI)             // int. od prijmu
  {
    RI = 0;           // shodit priznak

    if (DataRcvOver)  // jsme v chybe ?
    {
      byte b = SBUF;  // zahodit prijimane
    }
    else
    {
      DataRcvReady = 1;  // kazdopadne neco prislo
      _rbuf[_rb_wr] = SBUF;  // uloz do RCV bufferu

      _rb_wr++;       // ukazovatko na dalsi
      if (_rb_wr >= _BUFFER_SIZE)  // modulo velikost buf.
        _rb_wr = 0;

      if (_rb_wr == _rb_rd)  // dojel cteci ukazovatko ?
        DataRcvOver = 1;  // neni kam zapsat
    }
  }
}

void SendByte(byte u)   // odesli znak (byte)
{
  while(!_ready_send)   // cekej nez je odeslano predchozi
    ;

  _ready_send = 0;      // shod priznak

  SBUF = u;             // dej hodnotu do vysilaciho registru
}

void SendString(char *cp)  // odesli retezec
{
  for(; *cp; cp++)      // pro vsechny znaky retezve
    SendByte(*cp);      // posli znak
}

byte ReadByte()         // cteni znaku z RCV bufferu
{
  if (DataRcvReady)     // je neco prijato ?
  {
    byte z = _rbuf[_rb_rd];  // nejstarsi nevycteny znak

    _rb_rd++;           // cteci ukazovatko +1
    if (_rb_rd >= _BUFFER_SIZE)  // modulo velikost bufferu
      _rb_rd = 0;

    if (_rb_rd == _rb_wr)  // vycteno vsechno ?
      DataRcvReady = 0;   // priznak prijateho shodit

    return z;           // vrat ten znak
  }
  else
    return 0;           // neplatna hodnota je 0
}

void init_serial()  // 4800 pri 12 MHz
{
  TMOD = (TMOD & 0x0F) | 0x20;          // POUZE CT1 rezim 2 = 8-bitovy citac
  TH1 = 0xF3;                           // pro rychlost 2400/4800 Bd pri 12MHz krystalu
  TR1 = 1;                              // start citace
  SCON = 0x52;                          // standardni nastaveni seriaveho kanalu
  PCON |= 0x80;                         // SMOD=1 => zdvojeni seriove rychlosti (na 4800)
}

code unsigned char gen[] = {
  0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90, 0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E, 0xBF};

char buf[4] = {3, 2, 1, 0};  // VRAM

byte gCounter = 0;

void timer0(void) interrupt 1
{
  static byte u = 0;// znaky
  static byte cnt = 0;

  P1 &= 0xF0;                   // zhasnout vsechny
  P0 = buf[u];   // bitovy vzor
  P1 |= 1 << u;  // zapni jeden digit

  u++;           // pocitadlo znaku
  if (u > 3)  // vice nez mame ?
    u = 0;    // opet od zacatku

  cnt++;      // po 200us
  if (cnt < 250)
    return;

  cnt = 0;    // kazdych 50ms
  if (gCounter) // !=0 efektivnejsi nez >0
    gCounter--;

  buf[3] = gen[_rb_wr];  // hodnota RCV zapisoveho ukazovatka
  buf[2] = gen[_rb_rd];  // hodnota RCV cteciho ukazovatka
}
```

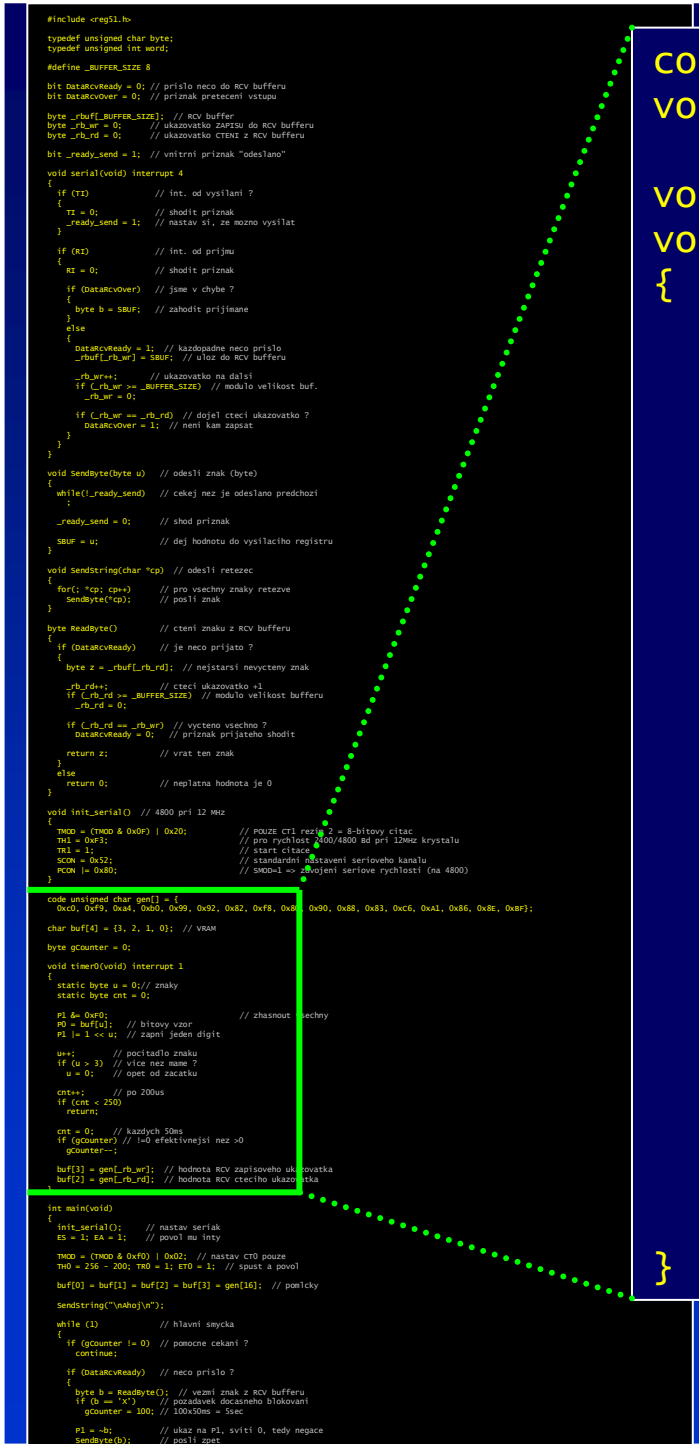
```
int main(void)
{
  init_serial();     // nastav seriak
  ES = 1; EA = 1;    // povol mu inty

  SendString("\nAhoj\n");

  while (1)                 // hlavni smycka
  {
    if (DataRcvReady)       // neco prislo ?
    {
      byte b = ReadByte();  // vezmi znak z RCV bufferu

      P1 = ~b;              // ukaz na P1, sviti 0, tedy negace
      SendByte(b);          // posli zpet

      if (DataRcvOver)      // preplneno
        SendString("\r\nOver\r\n");
    }
  }
}
```

Inicializační sekvence

- seriový port + interrupty od něj
- úvodní text

Hlavní smyčka

- pokud je něco v přijímacím bufferu
- přečti, zobraz na P1 (v „přímé“ podobě), pošli zpět
- v případě přeplnění vypiš – zatím nemůže reálně nastat



```
#include <reg51.h>

typedef unsigned char byte;
typedef unsigned int word;

#define _BUFFER_SIZE 8

bit DataRcvReady = 0; // prislo neco do RCV bufferu
bit DataRcvOver = 0;  // priznak preteceni vstupu

byte _rbuf[_BUFFER_SIZE];  // RCV buffer
byte _rb_wr = 0;      // ukazovatko ZAPISU do RCV bufferu
byte _rb_rd = 0;      // ukazovatko CTENI z RCV bufferu

bit _ready_send = 1;  // vnitrni priznak "odeslano"

void serial(void) interrupt 4
{
  if (TI)             // int. od vysilani ?
  {
    TI = 0;           // shodit priznak
    _ready_send = 1;  // nastav si, ze mozno vysilat
  }

  if (RI)             // int. od prijmu
  {
    RI = 0;           // shodit priznak

    if (DataRcvOver)  // jsme v chybe ?
    {
      byte b = SBUF;  // zahodit prijimane
    }
    else
    {
      DataRcvReady = 1;  // kazdopadne neco prislo
      _rbuf[_rb_wr] = SBUF;  // uloz do RCV bufferu

      _rb_wr++;       // ukazovatko na dalsi
      if (_rb_wr >= _BUFFER_SIZE)  // modulo velikost buf.
        _rb_wr = 0;

      if (_rb_wr == _rb_rd)  // dojel cteci ukazovatko ?
        DataRcvOver = 1;  // neni kam zapsat
    }
  }
}

void SendByte(byte u)   // odesli znak (byte)
{
  while(!_ready_send)   // cekej nez je odeslano predchozi
    ;

  _ready_send = 0;      // shod priznak

  SBUF = u;             // dej hodnotu do vysilaciho registru
}

void SendString(char *cp)  // odesli retezec
{
  for(; *cp; cp++)      // pro vsechny znaky retezve
    SendByte(*cp);      // posli znak
}

byte ReadByte()         // cteni znaku z RCV bufferu
{
  if (DataRcvReady)     // je neco prijato ?
  {
    byte z = _rbuf[_rb_rd];  // nejstarsi nevycteny znak

    _rb_rd++;           // cteci ukazovatko +1
    if (_rb_rd >= _BUFFER_SIZE)  // modulo velikost bufferu
      _rb_rd = 0;

    if (_rb_rd == _rb_wr)  // vycteno vsechno ?
      DataRcvReady = 0;   // priznak prijateho shodit

    return z;           // vrat ten znak
  }
  else
    return 0;           // neplatna hodnota je 0
}

void init_serial()  // 4800 pri 12 MHz
{
  TMOD = (TMOD & 0x0F) | 0x20;          // POUZE CT1 rezim 2 = 8-bitovy citac
  TH1 = 0xF3;                           // pro rychlost 2400/4800 Bd pri 12MHz krystalu
  TR1 = 1;                              // start citace
  SCON = 0x52;                          // standardni nastaveni serioveho kanalu
  PCON |= 0x80;                         // SMOD=1 => zdvojeni seriove rychlosti (na 4800)
}

code unsigned char gen[] = {
  0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90, 0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E, 0xBF};

char buf[4] = {3, 2, 1, 0};  // VRAM

byte gCounter = 0;

void timer0(void) interrupt 1
{
  static byte u = 0;// znaky
  static byte cnt = 0;

  P1 &= 0xF0;                   // zhasnout vsechny
  P0 = buf[u];   // bitovy vzor
  P1 |= 1 << u;  // zapni jeden digit

  u++;           // pocitadlo znaku
  if (u > 3)  // vice nez mame ?
    u = 0;    // opet od zacatku

  cnt++;     // po 200us
  if (cnt < 250)
    return;
  cnt = 0;    // kazdych 50ms

  if (gCounter) // !=0 efektivnejsi nez >0
    gCounter--;

  buf[3] = gen[_rb_wr];  // hodnota RCV zapisoveho uk.
  buf[2] = gen[_rb_rd];  // hodnota RCV cteciho uk.
}

int main(void)
{
  init_serial();    // nastav seriak
  ES = 1; EA = 1;   // povol mu inty

  TMOD = (TMOD & 0xf0) | 0x02;  // nastav CT0 pouze
  TH0 = 256 - 200; TR0 = 1; ET0 = 1;  // spust a povol

  buf[0] = buf[1] = buf[2] = buf[3] = gen[16];  // pomlcky

  SendString("\nAhoj\n");

  while (1)             // hlavni smycka
  {
    if (gCounter != 0)  // pomocne cekani ?
      continue;

    if (DataRcvReady)   // neco prislo ?
    {
      byte b = ReadByte();  // vezmi znak z RCV bufferu
      if (b == 'X')         // pozadavek docasneho blokovani
        gCounter = 100; // 100x50ms = 5sec

      P1 = ~b;          // ukaz na P1, sviti 0, tedy negace
      SendByte(b);      // posli zpet
```

Vypsání hodnoty ukazovátek bufferu na LED displeji

- opakuje se v interruptu, který zároveň obnovuje displej

```
int main(void)
{
  init_serial();      // nastav seriak
  ES = 1; EA = 1;     // povol mu inty

  TMOD = (TMOD & 0xf0) | 0x02;  // nastav CT0 pouze
  TH0 = 256 - 200; TR0 = 1; ET0 = 1;  // spust a povol

  buf[0] = buf[1] = buf[2] = buf[3] = gen[16];  // pomlcky

  SendString("\nAhoj\n");

  while (1)               // hlavni smycka
  {
    if (gCounter != 0)  // pomocne cekani ?
      continue;

    if (DataRcvReady)   // neco prislo ?
    {
      byte b = ReadByte();  // vezmi znak z RCV bufferu
      if (b == 'X')     // pozadavek docasneho blokovani
        gCounter = 100; // 100x50ms = 5sec

      P1 = ~b;          // ukaz na P1, sviti 0, tedy negace
      SendByte(b);      // posli zpet

      if (DataRcvOver)  // preplneno
        SendString("\r\nOver\r\n");
    }
  }
}
```

Sledování ukazovátek

- povolit přerušení od časovače - 200µs
- vyčištění LED displeje
- zbytek v interruptu

Zablokování příjmu

- po příchodu znaku ‘X’ se na 5 sec zastaví příjem znaků
- časuje se globální dekrementující proměnnou, která DEC každých 50ms (timer0)
- nastavit 100x, nepřijímáme, dokud se nevynuluje

Vizuálně funkce

- čtecí ukazovátko se nemění, jen zapisovací
- po uplynutí času jsou znaky v bufferu zpracovány a odeslány

Námět na zamyšlení

- Odesílací buffer
  - upravit veřejnou funkci **SendByte()** a interruptovou **serial()**
  - přidat příznak přetečení **DataSendOver**
  - přidat privátní pomocné proměnné
    - **byte _wbuf[_BUFFER_SIZE]** – vysílací buffer
    - **byte _wb_wr** – ukazovátko zápisu do SEND bufferu
    - **byte _wb_rd** – ukazovátko čtení ze SEND bufferu
  - není potřebný příznak **_ready_send**
- Ostatní kód se výhodně nemění

- Možno přidat výpis na LED displej
  - **buf[1] = gen[_wb_wr]; // hodnota SEND zapisoveho ukazovatka**
  - **buf[0] = gen[_wb_rd]; // hodnota SEND cteciho ukazovatka**

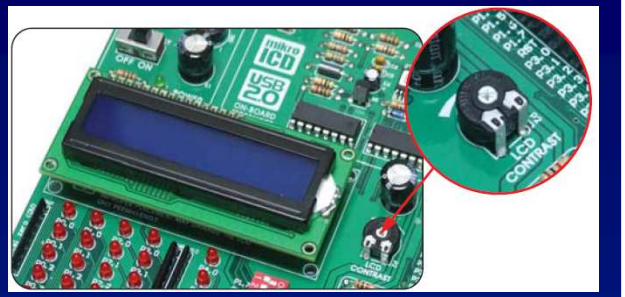
Plán cvičení

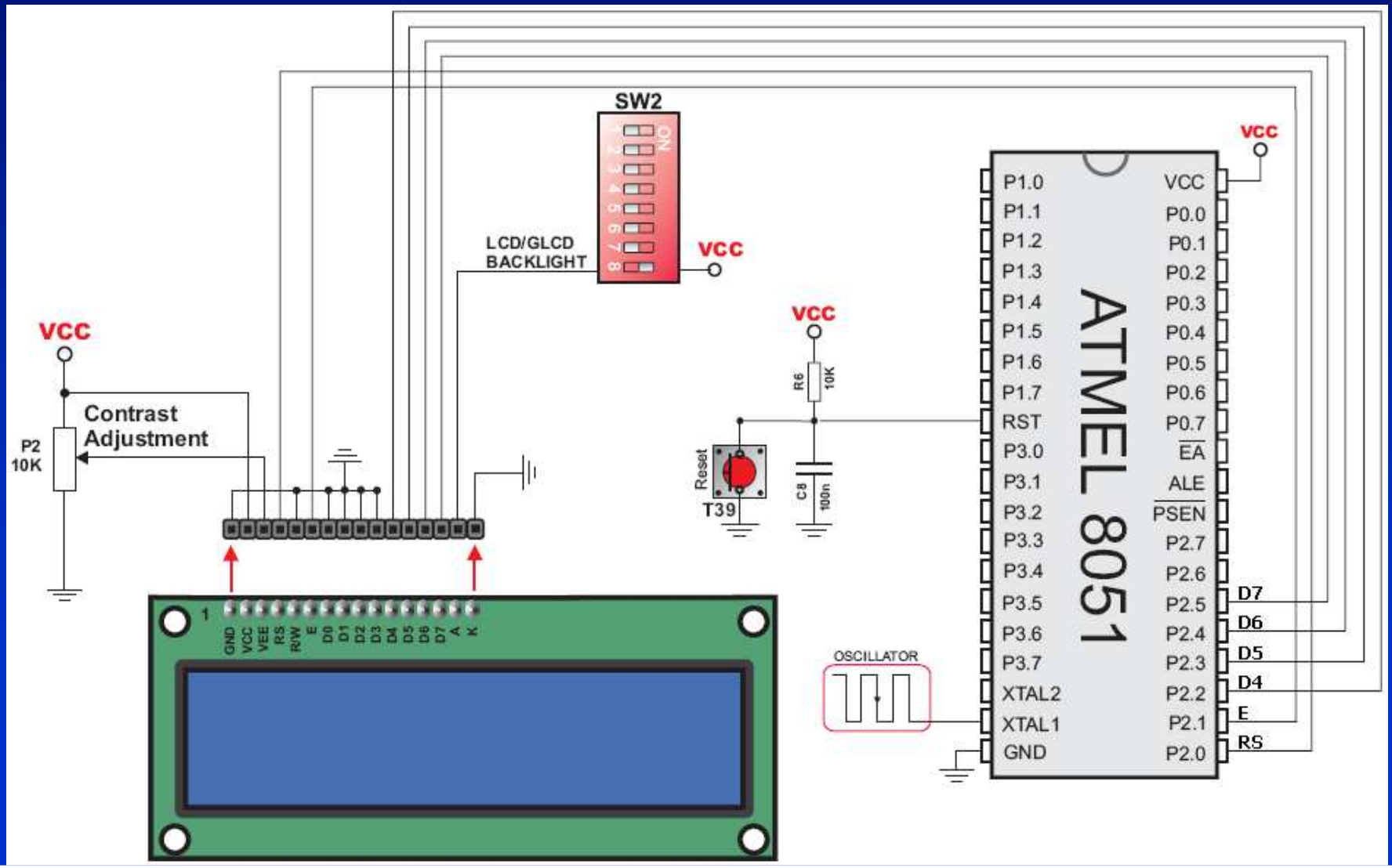
1. Úvod a seznámení – lab., IDE Keil uVision, první program
2. HW – porty, časovač, přerušení
3. HW – displej LED
4. HW – sériový port – úvod, spojení s displejem
5. HW – sériový port – přerušení + kruhový buffer
6. **HW – LCD displej**
7. Samostatná práce – zužitkování probraných znalostí
8. Sam. práce – dokončení. Grafický LCD
9. HW – PWM generované programově
10. Watchdog a práce s ním
11. Samostatná práce 2
12. Samostatná práce 2
13. Samostatná práce + Zápočet

LCD displej 2x16 znaků

- 2x16 textový displej
- Standardní řadič založený na obvodu **HD44780**
  - "inteligentní řadič" - činnost řízena "povely"
  - na displeji vlastní RAM pro zobrazenou informaci
  - možnost 4-/8-bitové komunikace
  - dokumentace na "vnitřním webu" - **http://klab**
- Použita 4-bitová komunikace
  - bity **D4-D7** na bráně **P2.2-5**
  - **P2.1** - **RS** (výběr "řízení" = log. 0, "data" = log. 1)
  - **P2.0** - **E** (= Enable) - puls do "1" zapisuje do LCD
  - signál **R/W** displeje připojen na log. 0
    - trvale stav "write"
    - nelze číst příznak **BUSY** (= splnění příkazu)
    - nutno čekat na dokončení programovým čekáním

Zapojení LCD displeje





Příkazy k ovládání LCD displeje

Instruction	Code RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{ep} or f_{osc} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*

Základ časování

- f_{osc} = 250kHz, takže časy 1.60ms a 40µs
- typicky se doporučuje časy 2x až 5x delší – výrobci běžně nedodržují původní specifikaci

Formát příkazů

- RS = 0, R/W = 0
- příkaz rozpoznán podle MSB
- typicky stačí inicializační sekvence typu Set_interface – Clear – Mode_Set
- adresa v DD RAM určuje pozici, kam se budou zapisovat data

Formát dat

- RS = 1, R/W = 0
- po posledním povelu „nastavení CGRAM“ se zapisuje do generátoru znaků, po nastavení DDRAM do paměti zobrazení

Čtení stavu nebo dat

- RS = 0 nebo 1, R/W = 1
- na desce není zapojeno = **NELZE**

LCD displej – další informace

- Doplnění významu bitů příkazů
- Proces 4-bitové komunikace
- Časování zápisového cyklu - především délka E pulsu

I/D = 1:	Increment	
I/D = 0:	Decrement	
S = 1:	Accompanies display shift	
S/C = 1:	Display shift	
S/C = 0:	Cursor move	
R/L = 1:	Shift to the right	
R/L = 0:	Shift to the left	
DL = 1:	8 bits, DL = 0: 4 bits	
N = 1:	2 lines, N = 0: 1 line	
F = 1:	5 × 10 dots, F = 0: 5 × 8 dots	
BF = 1:	Internally operating	
BF = 0:	Instructions acceptable	

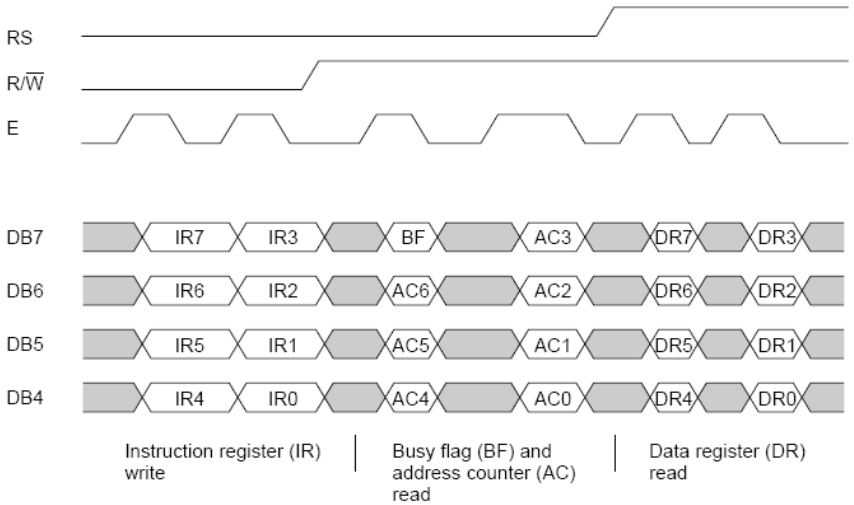
DDRAM: Display data RAM
CGRAM: Character generator RAM
ACG: CGRAM address
ADD: DDRAM address (corresponds to cursor address)
AC: Address counter used for both DD and CGRAM addresses

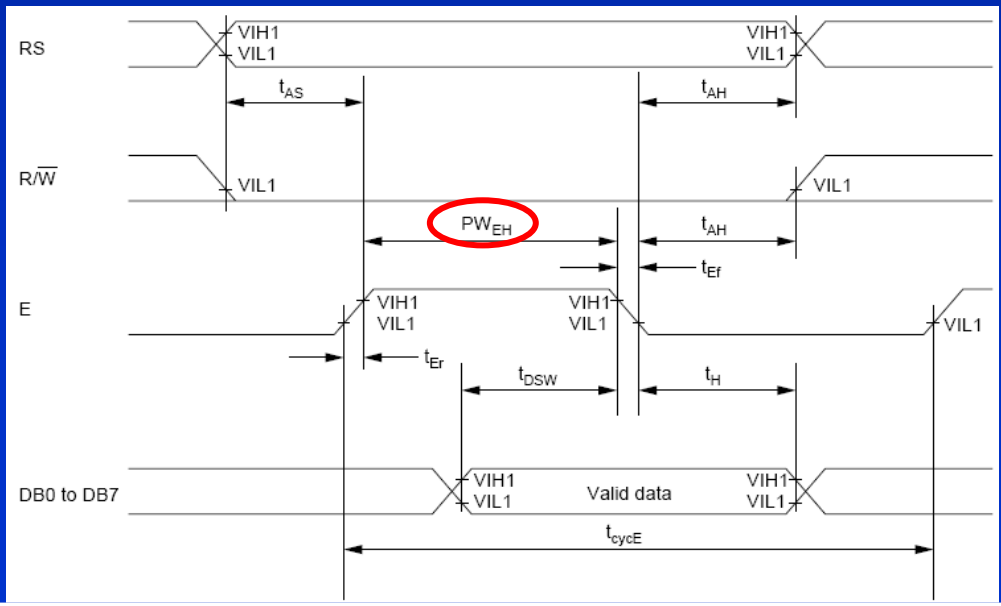
Execution time changes when frequency changes
Example:
When f_{cp} or f_{OSC} is 250 kHz,

$$37\ \mu s \times \frac{270}{250} = 40\ \mu s$$

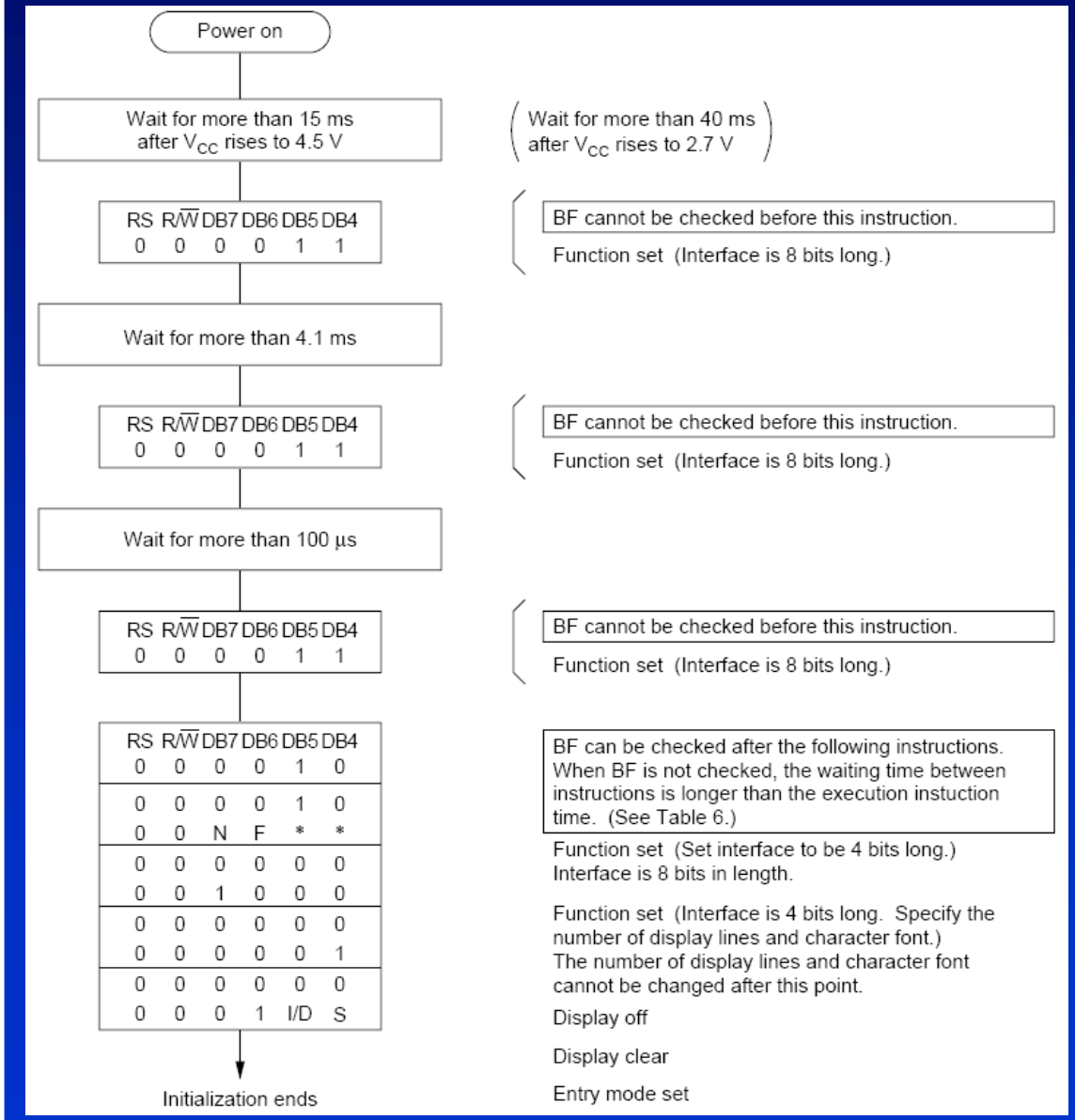
Write Operation

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	500	—	—	ns
Enable pulse width (high level)	PW_{EH}	230	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20	
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—	
Address hold time	t_{AH}	10	—	—	
Data set-up time	t_{DSW}	80	—	—	
Data hold time	t_H	10	—	—	





LCD – inicializační sekvence



Úvodní čekání
- min. 15 ms dle specifikace
- typicky se doporučuje časy min. 5x delší

Přechod na 4-bitovou komunikaci
- zápis první hodnoty jakoby přes 8-bitové rozhraní (bitově 0011)
- čekání min. 4.1ms (raději déle)
- zopakování akce zápis+čekání 4.1ms
- zopakování akce zápis+čekání 100µs

Inicializace ve 4-bitovém přenosu
- přenos běžných 8-bitových povelů (dle tabulky) po nibblech, vyšší napřed
- při každé 4-bitové hodnotě puls E
- po celém příkazu buď testovat příznak BUSY nebo čekat potřebný čas

Doporučená sekvence
- 0x2C – 00101100 – 4-bit rozhraní, 2 řádky
- 0x0F – 00001111 – Displej+Cursor ON
- 0x01 – 00000001 – Clear – 1.6ms čekat
- 0x06 – 00000110 – Entry mode
- 0x80 – 10000000 – pozice v DDRAM

```
#include <reg51.h>

typedef unsigned char byte;
typedef unsigned int word;

volatile byte bCekej100 = 0;

void timer0(void) interrupt 1
{
  if (bCekej100)
    bCekej100--;
}

int main(void)
{
  TMOD = 0x02;
  TH0 = 256 - 100;
  TR0 = 1;
  ET0 = 1;
  EA = 1;

  while(1)
  {
    bCekej100 = 200;
    while(bCekej100)
      ;

    P3++;
  }
}
```

Časování

- K časování použit CT0 a přerušení
  - vyvoláno každých 100 tiků
  - počítáme 100-vky v globální proměnné
  - čekání realizováno nastavením nenulové hodnoty a čekáním na vynulování
  - **POZOR** – typicky nutno vkládat hodnotu požadovaného času **+1**, protože zápis je asynchronní k vyvolání interruptu a ve skutečnosti bude čekání probíhat:

    $bc*100 > \text{čas} \geq (bc-1)*100 \ [\mu s]$
  - každých 20ms se inkrementuje P3 (200x100us) = ověření funkčnosti

- Pro zjednodušení deklarací opět vytvořeny nové typy **byte** a **word**

Funkce a makra pro displej

```
#define PULS_ENABLE {P2 |= 0x02; P2 &= 0xfd;}
#define SET_DATA(d) {P2 = (d & 0x0f) << 2;}

#define CEKEJ100(ticks100) {bCekej100 = (ticks100 + 1); while(bCekej100) ;}
```

- Makro PULS_ENABLE vytvoří na **P2.1** puls dlouhý 1 instrukční cyklus = doba 1µs pro 12MHz
- Makro SET_DATA nastaví na **P2.2-5** spodní 4 bity hodnoty = maskování + posun
- Makro CEKEJ100 nastaví požadavek čekání ve 100-kách tiků čítače (dáno nastavením čítače a přerušení) a počká na dopočítání požadované doby (pozor na +1 !!)

```
void writelcd(byte b, bit d)
{
  byte bb = b & 0xf0;

  bb >>= 4;
  SET_DATA(bb);
  if (d) P2 |= 0x01;
  PULS_ENABLE;
  CEKEJ100(1);

  bb = b & 0x0f;
  SET_DATA(bb);
  if (d) P2 |= 0x01;
  PULS_ENABLE;
  CEKEJ100(1);
}
```

Zápis dat do LCD připojeného 4-bitově

- Parametry funkce
  - zapisovaná hodnota (typu byte)
  - příznak zda je to příkaz (d = 0) nebo data (d = 1)
- Zápis horního nibble
  - maskovat horní 4 bity
  - posun dolů (4 bity vpravo)
  - nastavit data na **P2** (pomocí makra)
  - pokud jsou to data nastav **P2.0** na log. 1
  - vygeneruj puls na **E** (makrem na **P2.2**)
- Zápis spodního nibble
  - stejná sekvence včetně **E** pulsu

```
...
  CEKEJ100(200);    // po zapnuti

  SET_DATA(0x03);   // init 8-bit
  PULS_ENABLE;
  CEKEJ100(100);

  SET_DATA(0x03);   // init 8-bit
  PULS_ENABLE;
  CEKEJ100(100);

  SET_DATA(0x03);   // init
  PULS_ENABLE;
  CEKEJ100(100);

  SET_DATA(0x02);   // zapni 4-bit
  PULS_ENABLE;
  CEKEJ100(40);

  writelcd(0x2c, 0);  // 0010 1100
  writelcd(0x0f, 0);  // 0000 1100
  writelcd(0x01, 0);  // 0000 0001
  CEKEJ100(20);  // min 1.64ms
  writelcd(0x06, 0);  // 0000 0110

  writelcd(0x80, 0);  // 1000 0000

  writelcd('A', 1);   // zobrazit A
...
```

Inicializace

- Po zapnutí čekat cca 20ms
- 8-bitovou komunikací vykonat
  - 3x odeslat příkaz 0x03
  - vždy čekat 40us (zde lépe 10ms)
- Nastavit 4-bit komunikaci
  - odeslat 0x02 a 40us čekat
- Další akce již 4-bitovým postupem
  - 4-bit interface + 2 radky + font 5x7
  - disp. zap + kurzor zap. + kurzor blik
  - smaz displej a cekej 2ms
  - pohyb kurzoru + ne pohyb displeje

- Většina čekání z důvodu kompatibilit prodloužena z doporučených
- Dokumentace uvádí
  - po zapnutí 15ms
  - dlouhé akce 1.64ms
  - normální příkazy a data 40us

Výpis textu na LCD displej

```
...
  char *cptr, *text = "Ahoj KAE/MPP";
  int i;
...
  writelcd(0x80, 0);  // 1000 0000 - prikaz nastav kurzor na 0,0
  for (i = 0; i < 24; i++)
    writelcd('0' + i % 10, 1);

  writelcd(0xc0, 0);  // 1100 0000 - prikaz nastav kurzor na 0,1 (2. radek)
  for(cptr = text; *cptr; cptr++)
    writelcd(*cptr, 1);
...
```

- Předpokládá se automatický posun kurzoru
- Interní paměť displeje typicky 64 nebo 128 B
- Viditelných 16 znaků je pouze výřez řádku (viz. první **for**)
- Nastavení kurzoru (pozice výpisu) pro 2-řádkový displej
  - příkaz 1xxx xxxx = pozice 0 - 127
  - řádek 1 začíná na pozici 0
  - řádek 2 začíná na pozici 64 (0x40)

Náměty k zamyšlení

- Doplnit výpis znaků přicházejících po sériové lince na LCD displej
  - nastavit správně sériovou komunikaci
  - respektovat nutnost přechodu na nový řádek
  - P3.1 = TxD a P3.0 = RxD – nesmí se pracovat s P3 !!
- Vytvořit počítadlo zobrazující hodnotu na displeji
  - inkrement po vhodné době (např. 100 msec)
  - psát stále od stejné pozice
  - nutno vytvořit funkci pro převod číslo->text

Plán cvičení

1. Úvod a seznámení – lab., IDE Keil uVision, první program
2. HW – porty, časovač, přerušení
3. HW – displej LED
4. HW – sériový port – úvod, spojení s displejem
5. HW – sériový port – přerušení + kruhový buffer
6. HW – LCD displej
7. **Samostatná práce – zužitkování probraných znalostí**
8. Sam. práce – dokončení. Grafický LCD
9. HW – PWM generované programově
10. Watchdog a práce s ním
11. Samostatná práce 2
12. Samostatná práce 2
13. Samostatná práce + Zápočet

Zužitkování probraných znalostí

- Navrhněte a realizujte aplikaci "hodiny"
  - čas zobrazen na LCD displeji
  - nastavení aktuálního času pomocí sériové linky (terminálu na PC)
- Rady a nápady
  - navrhnout vhodnou vnitřní reprezentaci času
  - vyřešit korektní zobrazení času na LCD
  - vyřešit přesné časování (interrupt)
  - příjem znaků ze sériové linky
    - zřejmě stačí bez interruptu
    - příjem znaku se pozná podle příznaku RI
  - korektní zpracování číselné informace = nastavení nové hodnoty
- Možné problémy
  - vykonávání kódu v interruptech co nejkratší
  - žádné blokující akce (čekání "na něco")
  - funkce scanf je na C51 nepoužitelná

  - mají se hodiny během nastavování zastavit ?

Plán cvičení

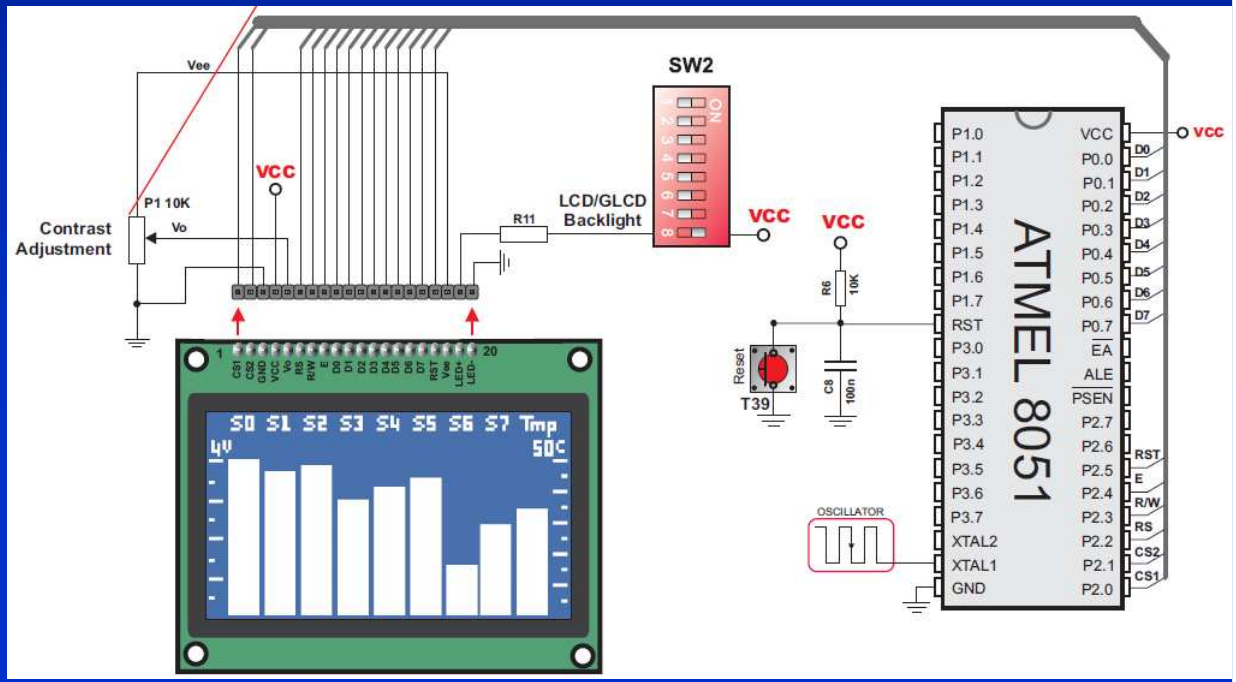
1. Úvod a seznámení – lab., IDE Keil uVision, první program
2. HW – porty, časovač, přerušení
3. HW – displej LED
4. HW – sériový port – úvod, spojení s displejem
5. HW – sériový port – přerušení + kruhový buffer
6. HW – LCD displej
7. Samostatná práce – zužitkování probraných znalostí
8. **Sam. práce – dokončení. Grafický LCD**
9. HW – PWM generované programově
10. Watchdog a práce s ním
11. Samostatná práce 2
12. Samostatná práce 2
13. Samostatná práce + Zápočet

Dokončení samostatné práce

- Na jaké problémy se narazilo ?

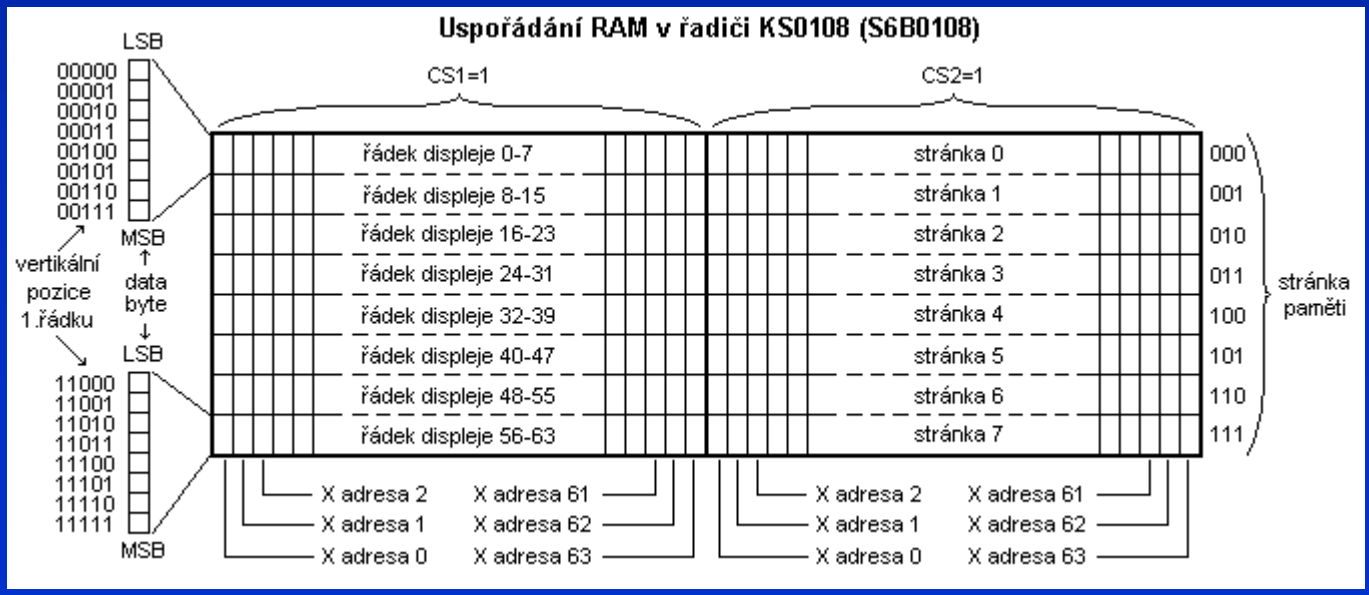
Grafický LCD

- Rozlišení 128x64, podsvícení, 2x řadič KS108 (poloviny displeje)
- Připojení
  - Pozor, nesmí být najednou textový a grafický LCD
  - Data – obousměrně brána P0
  - Řídící signály – bity P2
  - RST = reset
  - E = enable
  - RW = read/write
  - RS = reg. select
  - CSx = výběr řadiče



Organizace grafické paměti

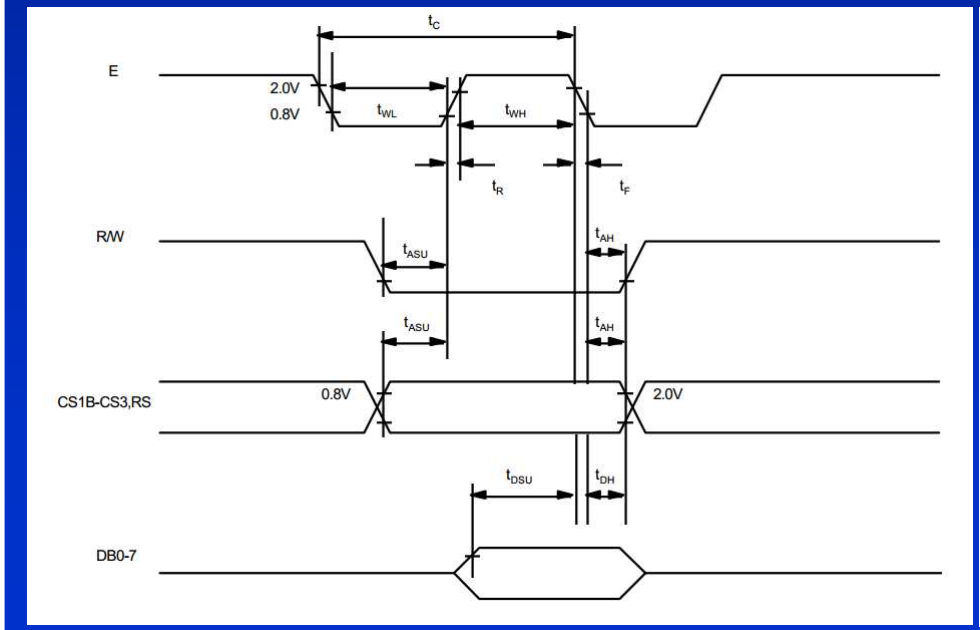
- Každý řadič ovládá polovinu displeje, chování identické
  - Rozděleno na stránky 64 sloupců x 8 pixelů
  - Každý sloupec odpovídá jednomu byte v paměti
  - LSB nahoře
- Nutno vybrat stránku, v rámci ní nastavit pozici sloupce a tam zapsat/číst byte
  - Pro další operaci se vnitřní ukazovátko adresy inkrementuje

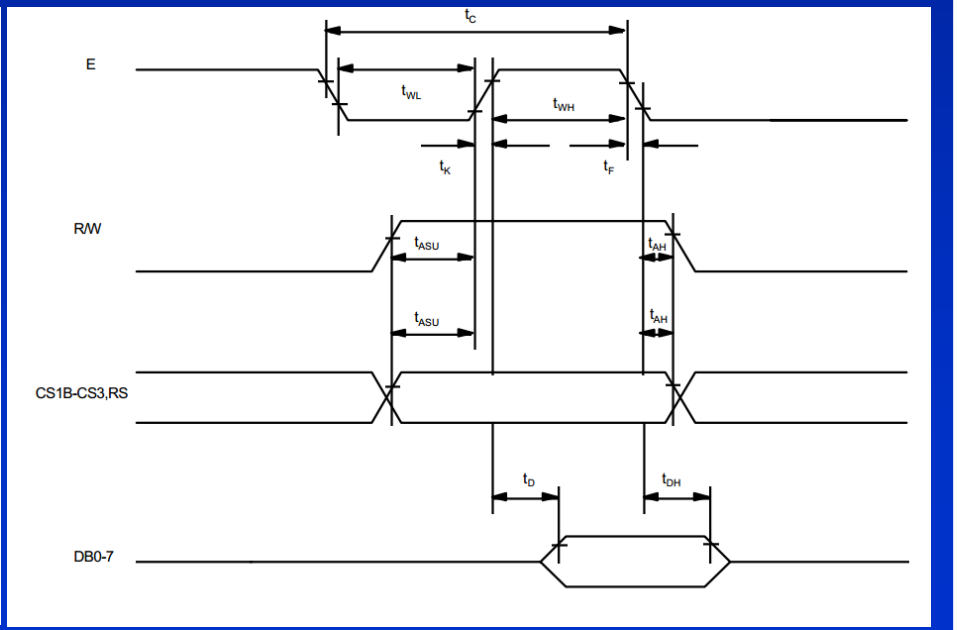


Časování komunikace s řadičem

(3) MPU Interface

Chatacteristic	Symbol	Min	Typ	Max	Unit
E Cycle	t_C	1000	-	-	ns
E High Level Width	t_{WH}	450	-	-	ns
E Low Level Width	t_{WL}	450	-	-	ns
E Rise Time	t_R	-	-	25	ns
E Fall Time	t_F	-	-	25	ns
Address Set-Up Time	t_{ASU}	140	-	-	ns
Address Hold Time	t_{AH}	10	-	-	ns
Data Set-Up Time	t_{SU}	200	-	-	ns
Data Delay Time	t_D	-	-	320	ns
Data Hold Time (Write)	t_{DHW}	10	-	-	ns
Data Hold Time (Read)	t_{DHR}	20	-	-	ns





Příkazy řadiče

RS	R/W	Function
L	L	Instruction
	H	Status read (busy check)
H	L	Data write (from input register to display data RAM)
	H	Data read (from display data RAM to output register)

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Function
Display ON/OFF	L	L	L	L	H	H	H	H	H	L/H	Controls the display on or off. Internal status and display RAM data is not affected. L:OFF, H:ON
Set Address	L	L	L	H	Y address (0~63)						Sets the Y address in the Y address counter.
Set Page (X address)	L	L	H	L	H	H	H	Page (0~7)			Sets the X address at the X address register.
Display Start Line	L	L	H	H	Display start line (0~63)						Indicates the display data RAM displayed at the top of the screen.
Status Read	L	H	BUSY	L	ON/OFF	RESET	L	L	L	L	Read status. BUSY L: Ready H: In operation ON/OFF L: Display ON H: Display OFF RESET L: Normal H: Reset
Write Display Data	H	L	Write Data								Writes data (DB0:7) into display data RAM. After writing intruction, Y address is increased by 1 automatically.
Read Display Data	H	H	Read Data								Reads data (DB0:7) from display data RAM to the data bus.

Použitelné příkazy

nastavení adresy – X osa

nastavení stránky – Y osa

Status

příznak BUSY po akci

příznak RESET po startu

Čtení + zápis dat

Základní knihovna pro GLCD

- Nutné operace se soubory – **z:\temp** nebo Courseware
  - Připojit hlavičkový soubor "**glcd.h**"
  - Přidat do projektu "**glcd.c**"
  - V pracovním adresáři ještě "**fonts.h**" – používá se 5x8 font
- Hi-level funkce pro texty (řádky odpovídají "stránkám" v řadiči)
  - **bit putcharXR(char c, byte x, byte row);** - vypíše znak
  - **bit putsXR(char *cp, byte x, byte row);** - vypíše řetězec
- Lo-level funkce
  - **void init_glcd(void);**
  - **void write_cmnd(byte g_cmnd, bit controller);** - příkaz pro řadič
  - **void write_data(byte g_data, bit controller);** - data pro řadič
- Makra
  - **SET_G_PAGE(page, radic)** – nastavení aktivní stránky 0-7 v řadiči
  - **SET_G_YADR(addr, radic)** – nastavení "sloupce" 0-63 v řadiči
- Na počátku programu zavolat **init_glcd();**

```
int main(void)
{
  byte x;
  word w;

  init_glcd();

  SET_G_PAGE(0, 0);      // levy radic, prvni stranka
  SET_G_YADR(0, 0);      // od pocatku
  for(x = 0; x < 64; x++)
  {
    write_data(x, 0);    // bitovy vzor pocitadlo
    wait_busy(0);
  }

  SET_G_PAGE(4, 1);      // pravy radic, 4 stranka
  SET_G_YADR(0, 1);      // od zacatku
  for(x = 0; x < 64; x++)
  {
    write_data(x, 1);    // bitovy vzor pocitadlo
    wait_busy(1);
  }

  putsXR("KAE/MPP 8. cviceni", 10, 7);

  for (x = 0; x < 200; x++)
  {
    putcharXR('A', x, 4);  // pohyblive pismeno
    for(w = 0; w < 20000; w++)  // zpozdeni
      ;
  }

  while(1)
    ;
}
```

Knihovna pro GLCD - příklad

Inicializace
- displej se vymaže (vyplní vzorem 0x00)

Vyplnit vzorem "pocitadlo"
- na stranku v levem i pravem radici
- je vidět "nejrychlejší" bit LSB horní

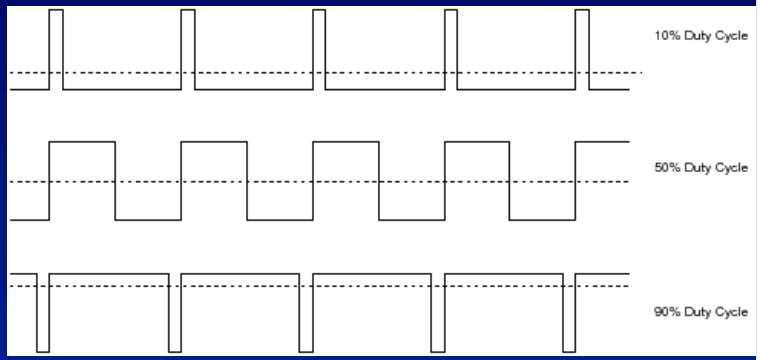
Vypsat text
- automaticky se přepíná řadič
- znaky za displejem se "zahodí"
- za každý znak se přidá prázdný "sloupeček"

Pohyblivé písmeno
- nechává stopu
- automaticky přepíná na hraně řadičů
- "za" displejem se "zahazuje"

Plán cvičení

1. Úvod a seznámení – lab., IDE Keil uVision, první program
2. HW – porty, časovač, přerušení
3. HW – displej LED
4. HW – sériový port – úvod, spojení s displejem
5. HW – sériový port – přerušení + kruhový buffer
6. HW – LCD displej
7. Samostatná práce – zužitkování probraných znalostí
8. Sam. práce – dokončení. Grafický LCD
9. **HW – PWM generované programově**
10. Watchdog a práce s ním
11. Samostatná práce 2
12. Samostatná práce 2
13. Samostatná práce + Zápočet

Generování PWM



- PWM – pulsně šířková modulace
  - digitální signál konstatní frekvence
  - proměnná střída (0-100%)
  - použitelné pro generování analogové hodnoty bez D/A předvodníku, "stačí" filtr
- Jednočipové mikropočítače mívají speciální blok pro PWM nebo vyhrazené režimy čítačů/časovačů
- Generování "ručně" pomocí 2 čítačů
  - jeden generuje základní frekvenci (např. 160us)
  - druhý měří délku pulsu
  - přesná šířka pulsu, náročnější optimalizace přerušení
- Generování pomocí jednoho čítače
  - měří se jen základní rozlišovací jednotka (např. 20us)
  - šířky pulsu jsou násobky základního kroku
  - perioda základního signálu je také násobkem "kroku" (8x20)

PWM pomocí 2 čítačů

```
#include <reg51.h>
typedef unsigned char byte;

volatile byte xsirka = 256 - 1;
void timer0(void) interrupt 1
{               // preruseni od CT0
  TL1 = xsirka; // doba pulsu
  TR1 = 1;      // zapni CT1
  P2 = 0x00;    // LED sviti - H-L
}

void timer1(void) interrupt 3
{               // preruseni od CT1
  TR1 = 0;      // stop citani
  P2 = 0xFF;    // LED zhasnout - L-H
}

int main(void)
{
  byte x, w; char c;

  TMOD = 0x22; // oba v 8-bit rezimu

  ET0 = 1; ET1 = 1; EA = 1;  // inty
...
```

```
...
  TH0 = TL0 = 256 - 160;  // 160us
  TR0 = 1;      // spust CT0
  TH1 = 0;      // max. hodnota
  TR1 = 0;      // CT1 stoji

  while(1)
  {
    w = 1;      // klidova hodnota
    x = P3;     // stav tlacitek P3
    for (c = 8; c > 0; c--)
    {       // najde nejvyssi aktivni
      if (!(x & 0x80))  // je to ?
      {
        w = 20 * c; // sirka pulsu
        break;      // a nalezeno
      }

      x <<= 1;      // zkus dalsi
    }

    xsirka = 256 - w;  // count UP
    P0 = ~w;  // vizualizace hodnoty
  }
}
```

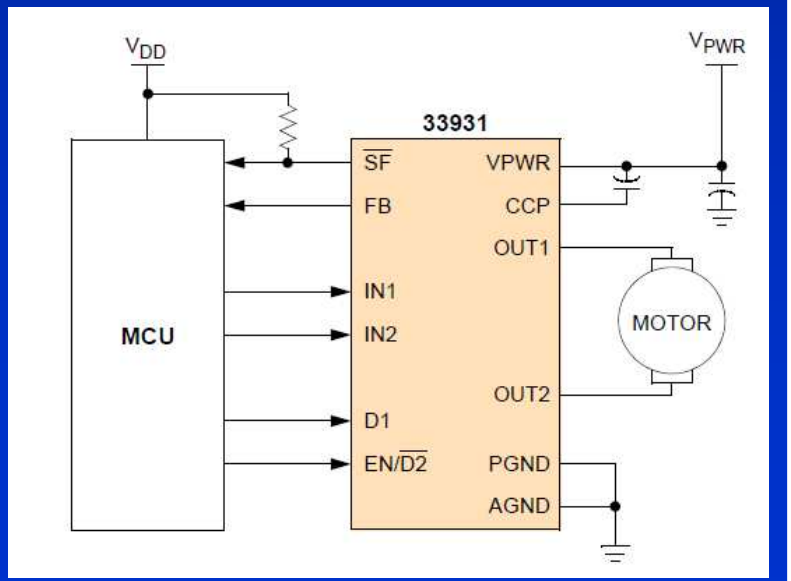
PWM pomocí 1 čítače

```
#include <reg51.h>
typedef unsigned char byte;

volatile byte psirka = 0;

void timer0(void) interrupt 1
{
  static byte cnt = 0; // perioda
  static byte w = 0;   // puls

  cnt--;           // doba periody
  if (cnt == 0)    // konec periody ?
  {
    P2 = 0x00;     // LED sviti - H-L
    w = psirka;    // odpocet sirky
    cnt = 8;       // taktu periody
  }

  if (w == 0)      // konec sirky ?
    P2 = 0xff;     // LED zhasni - L-H
  else
    w--;           // doba pulsu
}

...
```

```
int main(void)
{
  TMOD = 0x02;  // 8-bitový CT0

  TH0 = TL0 = 256 - 20;  // 20us
  TR0 = 1;      // zapnout
  ET0 = 1; EA = 1;  // interrupty

  while(1)
  {
    byte x; char c;

    x = P3;     // stav tlacitek
    for (c = 8; c > 0; c--)
    {
      if ((x & 0x80) == 0)
        break; // sepnuto ?

      x <<= 1; // hledej dalsi
    }

    psirka = c; // hodnota 0-8
    P0 = ~c; // vizualizace hodnoty
  }
}
```

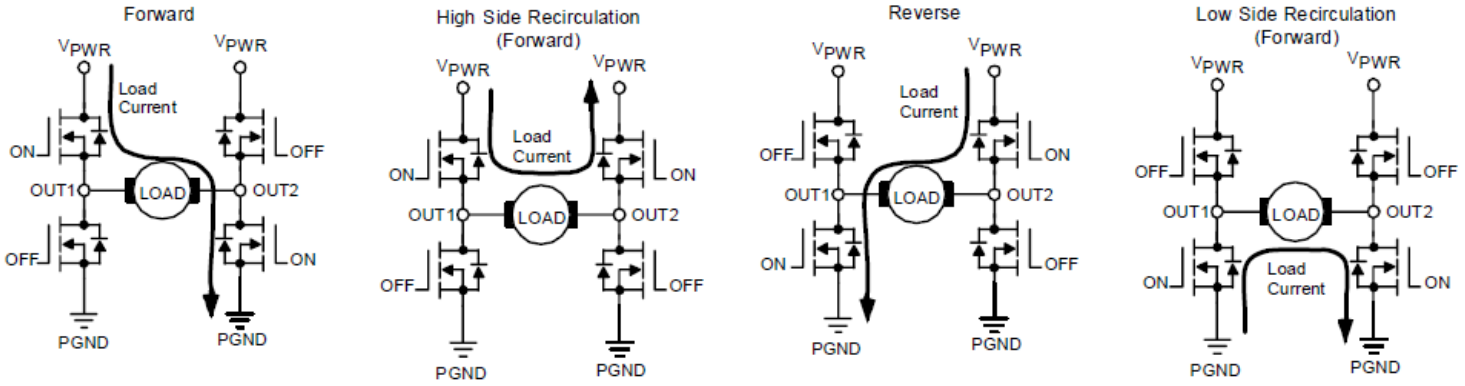
H-můstek pro buzení SS motoru

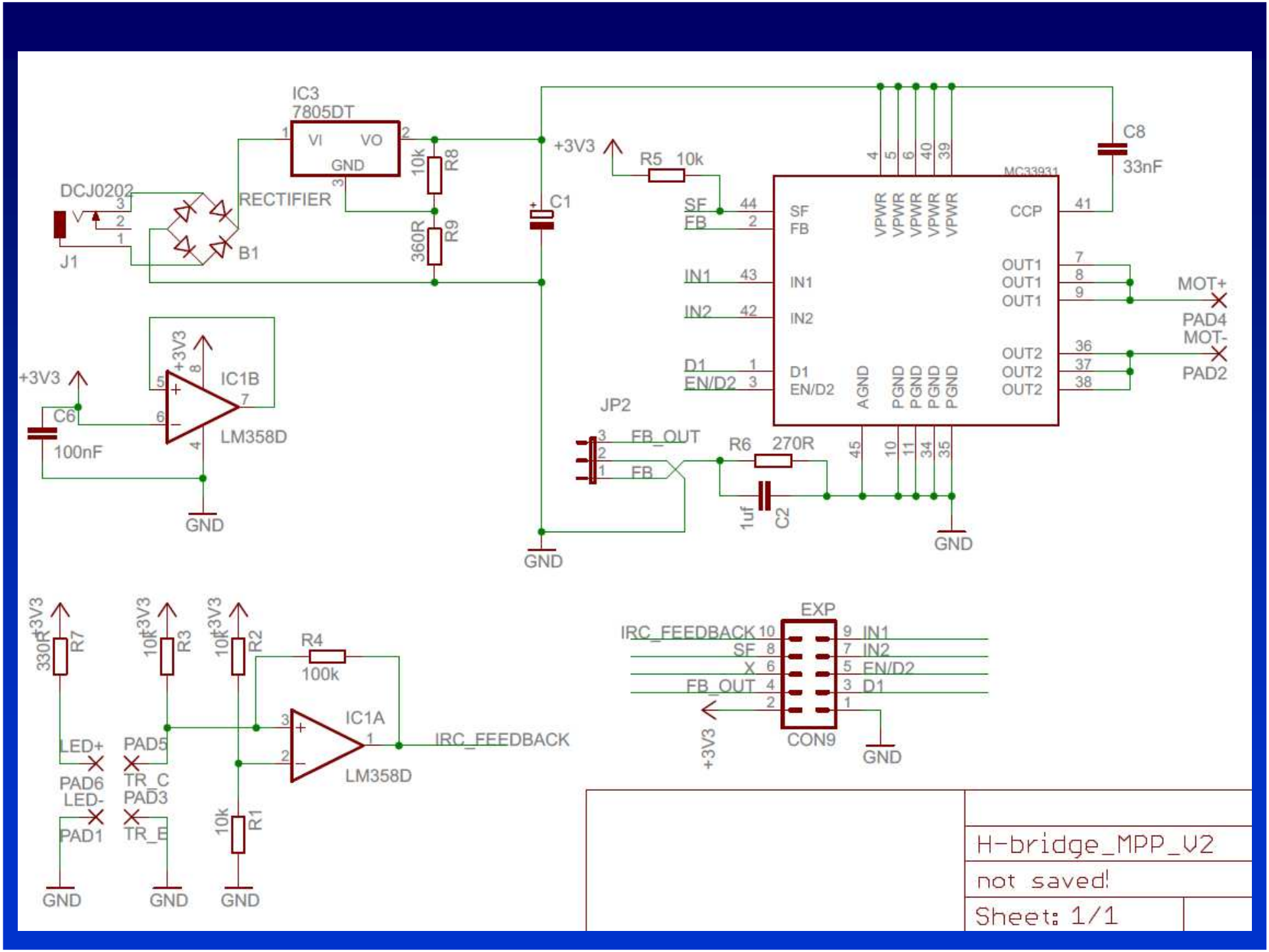
- Použito integrované řešení **MC33931**
- Vnitřně plný H-můstek s řízením
  - chování „anti-smoke" = nepustí proudy proti sobě
- Jednoduchý interface
  - 2 bity pro řízení směru
  - povolovací bity umožňující brzdění nebo „volné otáčení"
  - zpětné hlášení stavu
- Připojeno k portu P1
  - P1.1 = IN1
  - P1.3 = IN2
  - P1.5 = EN/D2
  - P1.7 = D1
  - P1.0 = vstup snímače otáček



H-můstek – signály

Device State	Input Conditions				Status	Outputs	
	EN/$\overline{D2}$	D1	IN1	IN2	$\overline{SF}$	OUT1	OUT2
Forward	H	L	H	L	H	H	L
Reverse	H	L	L	H	H	L	H
Free Wheeling Low	H	L	L	L	H	L	L
Free Wheeling High	H	L	H	H	H	H	H
Disable 1 (D1)	H	H	X	X	L	Z	Z
IN1 Disconnected	H	L	Z	X	H	H	X
IN2 Disconnected	H	L	X	Z	H	X	H
D1 Disconnected	H	Z	X	X	L	Z	Z
Under-voltage Lockout(29)	H	X	X	X	L	Z	Z
Over-temperature(30)	H	X	X	X	L	Z	Z
Short-circuit(30)	H	X	X	X	L	Z	Z
Sleep Mode EN/$\overline{D2}$	L	X	X	X	H	Z	Z
EN/$\overline{D2}$ Disconnected	Z	X	X	X	H	Z	Z




IC3
7805DT
VI
VO
GND
RECTIFIER
DCJ0202
J1
B1
10k
R8
360R
R9
C1
+3V3
R5 10k
C8
33nF
MC33931
SF
FB
IN1
IN2
D1
EN/D2
VPWR
CCP
OUT1
OUT2
AGND
PGND
MOT+
PAD4
MOT-
PAD2
JP2
FB_OUT
R6 270R
1uf
C2
IC1B
LM358D
C6
100nF
R4
100k
IC1A
IRC_FEEDBACK
330R
R7
R3
R2
R1
LED+
PAD6
LED-
PAD1
PAD5
TR_C
PAD3
TR_E
EXP
CON9
X
H-bridge_MPP_V2
not saved!
Sheet: 1/1

Úprava PWM pro motor

```
#define IN1ON { P1 |= 0x02; }
#define IN1OFF { P1 &= ~0x02; }

#define IN2ON { P1 |= 0x08; }
#define IN2OFF { P1 &= ~0x08; }

#define MENABLE { P1 &= ~0x80; }
#define MDISABLE { P1 |= 0x80; }

void timer0(void) interrupt 1
{
  static byte cnt = 0; // perioda
  static byte w = 0;   // puls

  cnt--;           // doba periody
  if (cnt == 0)    // konec periody ?
  {
    IN2OFF;
    w = psirka;    // odpocet sirky
    cnt = 8;       // taktu periody
  }

  if (w == 0)      // konec sirky ?
    IN2ON
  else
    w--;           // doba pulsu
}

int main(void)
{
...
  MENABLE;
  IN1ON;
...
```

Upravena varianta s jedním interruptem
- měníme jednu intrruptovou funkci
- zbývá CT1 pro sériový port

Makra
- nastavení bitů můstku do příslušných úrovní

Provoz
- povolení otáčení – MENABLE - odpovídá vstupu D1
- bit EN/D2 je v log. 1 – je to stav po RESETu obvodu
- točení motorem – změna jednoho signálu IN
--- jeden INx nastaven do stálé úrovně, druhý INy dělá PWM
--- podle INx se určije směr

Upraven kód interruptu
- pozor, u IN2ON středník NENÍ !!!

Upraven kód main
- nezapomenout na ENABLE
- při použití IN1OFF
-- motor by točil na druhou stranu
-- byl by opačný význam poměru střídy

Náměty k vylepšení/doplnění

- Ovládání pomocí sériového portu
  - máme volný CT1 = možno použít k nastavení rychlosti
  - styl řízení libolvolný - číslo nebo +- nebo jinak
- Jemné ovládání + a – změny
  - včetně vypisování aktuální hodnoty
  - seriová linka nebo pomocí tlačítek (vstupů Px.x)
- Zobrazení hodnoty na LCD – např. znakovém
  - ideálně ve formě bargrafu – řada znaků odpovídá hodnotě
- Regulace otáček pomocí čidla otáček
  - vypisovat aktuální rychlost
  - udržet otáčky i při změně zatížení

Plán cvičení

1. Úvod a seznámení – lab., IDE Keil uVision, první program
2. HW – porty, časovač, přerušení
3. HW – displej LED
4. HW – sériový port – úvod, spojení s displejem
5. HW – sériový port – přerušení + kruhový buffer
6. HW – LCD displej
7. Samostatná práce – zužitkování probraných znalostí
8. Sam. práce – dokončení. Grafický LCD
9. HW – PWM generované programově
10. **Watchdog a práce s ním**
11. Samostatná práce 2
12. Samostatná práce 2
13. Samostatná práce + Zápočet

Obvod/blok WatchDog

- Doslovný překlad – „hlídací pes“
- Slouží k „hlídání“ správné činnosti programu
  - pokud dojde k zacyklení v nějakém místě vinou SW nebo HW chyby, je po nějakém (nastaveném) čase vyvolán RESET
  - procesor se dostane do definované počátečního stavu
- WDT – watchdog timer
  - časovač, takt typicky odvozen předděličkou ze systémových hodin
  - po přetečení vyvolá RESET
  - možno vynulovat zápisem na pin (externí obvod) nebo do registru (interní část)
  - doba do RESETu dána předděličkou a „délkou“ čítače (počtem bitů)
  - vynulování se typicky provádí každým průchodem hlavní smyčkou programu
- V procesoru AT89S8253 je přímo vestavěn blok WDT

Práce s WDT v AT89S8253

D7	D6	D5	D4	D3	D2	D1	D0
SMOD1	**SMOD0**	**-**	**POF**	**GF1**	**GF0**	**PD**	**IDL**

- Registr **PCON** rozšířen
  - v bitu **POF** (Power On Flag) je při zapnutí napájení vždy nastavena log. 1
  - při RESETu se stav bitů nemění

D7	D6	D5	D4	D3	D2	D1	D0
PS2	**PS1**	**PS0**	**WDIDLE**	**DISRTO**	**HWDT**	**WSWRST**	**WDTEN**

- Registr **WDTCON**
  - **PS0-2** – dělička hodin
    - základ při 12MHz je 16ms * 2^{PS}
    - rozsah 16ms – 2048ms (=2sec)
  - **WDIDLE** = 1 – zakázání WDT čítání v IDLE módu
  - **DISRTO** = 0 – RST pin se chová jako výstupní a může resetovat připojené vnější obvody
  - **HWDT** = 0 – WDT zap/vyp bitem **WDTEN** (tzv. SW-driven-mode)
  - **HWDT** =1
    - WDT spuštěn zápisem hodnot 0x1E/0xE1 do registru **WDTRST** a vypnut pouze resetem
    - restart WDT se provádí stejným zápisem 0x1E/0xE1 do registru **WDTRST**
  - **WSWRST** = 1 – resetuje WDT, automaticky nastaven zpět na 0
  - **WDTEN** = 1 – povoluje běh WDT nebo lze číst status v HW-mode

Ověření COLD startu

```
#include <reg8253.h>
#include <stdio.h>
#include <intrins.h>

typedef unsigned int word;
typedef unsigned char byte;

int main(void)
{
  TMOD = 0x20;
  SCON = 0x52;
  TH1 = 0xf3;
  TR1 = 1;
  PCON |= 0x80;

  printf("\nStart APP, typ %s, PCON = %02X\n",
    (PCON & 0x10) ? "COLD" : "WARM", (word)PCON);

  PCON &= 0xef;  // shodit Power Off Flag

  while(1)
    ;
}
```

Include
- pozor, připojujeme **reg8253.h**, aby byly k dispozici i rozšiřující registry

Sériový port
- std. inicializace na 4800

Výpis stavu
- ternární operátor vybírá podle 4. bitu jeden z textů
- zkoumaný bit je POF
- dále se vypisuje obsah registru PCON
- **POZOR** – **printf** očekává 16-bitové parametry pro celočíselné hodnoty – proto přetypování !!

Vynulování POF
- příznak každopádně vynulován, příští restart bude detekován
- hodnota 0xEF je bitově 1110 1111

Úprava detekce a start WDT

```
...
int main(void)
{
  if (PCON & 0x10)  // COLD start ?
  {
    word w;
    for (w = 0; w < 50000; w++)
      _nop_();
  }
  ...
}
```

2xRESET ?
- při zapnutí se nastartuje aplikace v x51 a zároveň se inicializuje obvod USB-programmeru
- ten poté provede RESET obvodu x51
- to je již detekováno jako „WARM start“

Doplněno čekání
- v případě studeného startu (**POF** v log. 1)
- počet cyklů určen zkusmo
- časování zajištěno instrukcí **NOP**
- nezapomenout připojit **intrins.h**

```
...
#define WDT_TICK {WDTCON |= 0x02;}
...
int main(void)
{
  ...
  WDTCON = 0xeb; // 1110 1011
  while(1)
  {
    getchar();
    WDT_TICK;
  }
}
```

WDT_TICK
- nulování WDT se často definuje jako makro

Nastavení WDT registru
- předdělička na hodnotu **111**, tedy čas přetečení 2048ms
- SW-driven-mode

Hlavní smyčka
- **getchar()** je blokující operace, čeká na znak
- pokud přijde znak do cca 2sec, WDT se resetuje
- pokud nepřijde znak po dobu 2 sec, WDT provede RESET

Detekce chybné funkce

```
void funkce1(char z)
{
  if (z != 'x')  return;

  while(1) ;
}

void funkce2(char z)
{
  if (z != 'q')  return;

  while(1) ;
}

... // do mainu
  while(1)
  {
    if (RI)
    {
      char x = getchar();
      funkce1(x);
      funkce2(x);
    }
    WDT_TICK;
  }
}
```

Uměle vytvořené blokující funkce

- v případě, že předaný parametr není ‘x’ nebo ‘q’, funkce ihned končí
- pokud je, zůstává „viset“
- měl by se uplatnit WDT, pokud je spuštěn

Hlavní smyčka je neblokující

- pokud je detekován přijatý znak, je vyzvednut pomocí getchar()
- každý průchod hlavní smyčkou resetuje WDT
- k zastavení dojde pouze chybou (= zablokováním) ve volaných funkcích

Uložení aktuální operace

```
#define VERIFY_BYTE 0xa5
#define WRITE_OPERATION_CODE(x) {*((byte data *)0xfe) = x; *((byte data *)0xff) = VERIFY_BYTE;}
#define READ_OPERATION_CODE (*((byte data *)0xfe))
#define READ_OPERATION_VERIFY (*((byte data *)0xff))

#define FUNCTION_MAIN 0

#define FUNCTION_1 1
#define FUNCTION_2 2

void funkce1(char z)
{
  WRITE_OPERATION_CODE(FUNCTION_1);
  ...

void funkce2(char z)
{
  WRITE_OPERATION_CODE(FUNCTION_2);
  ...

int main(void)
{
  byte oper_status = READ_OPERATION_CODE;
  byte oper_verify = READ_OPERATION_VERIFY;
  ...
  printf("Oper. code %02X\n", (word)oper_status);
  printf("Oper. verify %02X - %s\n",
    (word)oper_verify, (oper_verify == VERIFY_BYTE) ? "OK" : "Fail");
  ...
  WRITE_OPERATION_CODE(FUNCTION_MAIN);
  ...
```

Předposlední byte vnitřní RAM obsahují kód operace
- RAM má 256 byte, takže adresa je **0xFE**
- zápis na pevné místo v paměti je pomocí přetypování
- vytvořeno makro
- nutno zavolat makro nejlépe na počátku každé funkce

Poslední byte je verifikační = obsahuje definovanou hodnotu
- po zapnutí má RAM náhodný obsah
- poslední byte má adresu **0xFF**
- zapisovací makro zapíše také zvolenou hodnotu

Další makra a použití
- makra definující prováděnou funkci a makro pro čtení kódu a stavu
- při startu vypíšu stav (= poslední operace) a verifikační byte

Automatické mazání datové paměti

- Při standardním vytvoření projektu v prostředí se přidá **STARTUP.A51**
  - slouží k inicializaci procesoru po startu aplikace
  - nastavuje symbol zásobníku tak, aby jej linker umístil za data
    - na x51 roste zásobník směrem k vyšším adresám
  - maže datovou oblast RAM
    - defaultně nastaveno na prvních 128 byte
    - pro paměť 256 B by se mělo mazat celých 256 B
  - je psán v asembleru
  - na konci skáče na adresu funkce main (C-čkovské)
- STARTUP-kód tedy potenciálně smaže naše „stavy" uložené pro detekci chyby pomocí WDT

- **Řešení ?**
  - upravit STARTUP tak, aby nemazal poslední 2 byte RAM
    - otevřete stávající **STARTUP.A51** a uložte jako např. **startwdt.a51**
    - v „*Source Group 1*" odeberte stávající STARTUP a přidejte ten nový
    - obsah nového následuje …

```
$NOMOD51
IDATALEN        EQU     0FEH

; Standard SFR Symbols
ACC     DATA    0E0H
SP      DATA    81H

                NAME    ?C_STARTUP

?C_C51STARTUP   SEGMENT   CODE
?STACK          SEGMENT   IDATA

                RSEG    ?STACK
                DS      1

                EXTRN CODE (?C_START)
                PUBLIC  ?C_STARTUP

                CSEG    AT      0
?C_STARTUP:     LJMP    STARTUP1

                RSEG    ?C_C51STARTUP
STARTUP1:
                MOV     R0,#IDATALEN - 1
                CLR     A
IDATALOOP:      MOV     @R0,A
                DJNZ    R0,IDATALOOP

                MOV     SP,#?STACK-1
                LJMP    ?C_START

                END
```

STARTWDT.A51

Základní ASM konstrukce

- **EQU** odpovídá **#define** – tedy definuje hodnoty
- **xSEG** a **SEGMENT** jsou informace pro linker, jak má tento kód spojit s ostatními částmi programu
- řádek začíná návěštím, hodnotou nebo odsazením
- vše za středníkem je komentář
- hexadecimální čísla končí **H**

Instrukce asembleru

- **LJMP** – skok na adresu/návěští
- **MOV** – přesun hodnot
-- parametr **Rx** – registr (x = 0 až 7)
-- #parametr – přímo hodnota, jinak adresa
-- @parametr – nepřímé adresování, použita hodnota v registru R0 nebo R1
- **DJNZ** – instrukce „dekrement and jump“
-- v podstatě do-while cyklus dekrementující

Mazací smyčka

- cyklus přes návěští **IDATALOOP**
- zapíšu 0 na adresu RAM počínaje **IDATALEN** – 1 až po adresu 0
- nutno tedy upravit hodnotu **IDATALEN**
-- zvoleno **0xFE** (tedy 254)
-- nebude se mazat **0xFE** a **0xFF** = požadovaný výsledek

WDT - shrnutí

- WatchDog Timer je běžně používaný prostředek k detekci nesprávného chodu aplikace z HW nebo SW příčin
- Realizace buď interní (většina moderních mikropočítačů) nebo externí (bývá spojen také s obvody RESETu)
- Doba dopočítání WDT a následného RESETu záleží na aplikaci, jak dlouho může být „zaseknutá"

- Nulování WDT se
  - zásadně nedává do přerušení – to nejspíše nastává, i když se program zastavil
  - typicky se dává do hlavní smyčky programu
    - pokud je dlouhá, možno i na více místech

- Detekce RESETu vlivem WDT
  - nejlépe někam do RAM uložit kód operace nebo funkce, která se prováděla
  - při startu zkontrolovat, zda je kód platný, příp. zda nešlo o studený start
  - zabezpečení hodnoty stavu
    - triviálně doplňujícím bajtem
    - lépe chránit více hodnot kontrolním součtem či CRC
  - POZOR, typicky nutno upravit STARTUP kód
    - běžně maže oblast datové paměti a inicializuje některé registry

Plán cvičení

1. Úvod a seznámení – lab., IDE Keil uVision, první program
2. HW – porty, časovač, přerušení
3. HW – displej LED
4. HW – sériový port – úvod, spojení s displejem
5. HW – sériový port – přerušení + kruhový buffer
6. HW – LCD displej
7. Samostatná práce – zužitkování probraných znalostí
8. Sam. práce – dokončení. Grafický LCD
9. HW – PWM generované programově
10. Watchdog a práce s ním
11. **Samostatná práce 2**
12. Samostatná práce 2
13. Samostatná práce + Zápočet

Samostatná práce 2

- Zadání
  - ovládejte otáčky SS motoru pomocí PWM
  - v interakci s uživatelem využijte sériový kanál a displej na desce
  - umožněte změnu směru otáčení a vhodnou volbu rychlosti
- Hodnocení
  - nutná **základní funkčnost** = známka **2**
  - **každý další realizovaný nápad zlepšuje hodnocení**
- Cílem je využít znalosti získané během předchozích cvičení