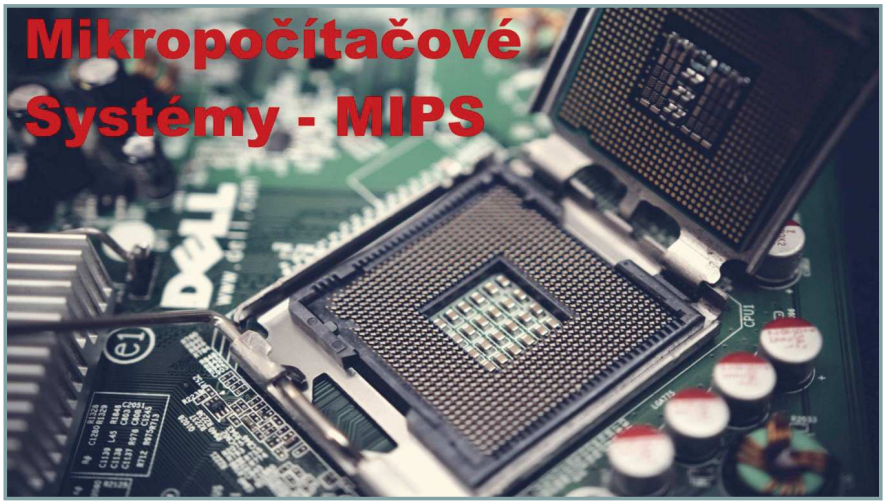



```c
#define LED1   PC3
#define SW1   PC2
#define LED2   PD7
#define SW2   PD6

int main(void)
{

    DDRC |= (1 << LED1);
    DDRC &= ~(1 << SW1);
    PORTC |= (1 << SW1);

    DDRD |= (1 << LED2);
    DDRD &= ~(1 << SW2);
    PORTD |= (1 << SW2);

    while(1)
    {
       if( !(PINC &(1<<SW1)))
       {
           PORTC |= (1 << LED1);
           PORTD &= ~(1 << LED2);
       }

       if(!(PIND &(1<<SW2)))
       {
           PORTD |= (1 << LED2);
           PORTD &= ~(1 << LED1);
        }
    }

    return 0;
}
```

```c
while(1)
   {

    if ( !(PINC & (1<<SW1)) )  // Ak je stlacene tlacitko (t.j. na vstupe log.0)...
      {
      PORTC |=  (1<<LED1);     // ... tak rozsviet LED, t.j. set PB5 na log. 1

       PORTD &= ~(1<<LED2);    // zhasni LED, t.j. clear PB5 na log. 0
      }
     if ( !(PIND & (1<<SW2)) ) // Ak je stlacene tlacitko (t.j. na vstupe log.0)...
      {
      PORTD |=  (1<<LED2);     // ... tak rozsviet LED, t.j. set PB5 na log. 1

       PORTC &= ~(1<<LED1);    // zhasni LED, t.j. clear PB5 na log. 0
      }
   }

return 0;
}
```

```c
int main(void)
{
  /*  SETUP  */

  /* Konfiguracia I/O: portB.5 je vystupny (LED1) a portB.4 je vstup (SW1) *
   * naviac je PortB.4 so zapnutym pull-up rezistorom cez reg. PORTB       */

   DDRB = 0b00100000;       // PORTB: LED1 na PB5  je output, SW1 (PB4) input
  PORTB = 0b00010000;       // LED Active low, LED off, pull-up ON

  /* Ak nechceme menit cely register, ale len jeden bit, potom pouzijeme *
   * niektoru z nasledovnych konstrukcii, vsetky su ekvivalentne         */

  DDRB |= (1 << LED1);

  /*   LOOP   */

  while(1)
    {
                  // 0b00010000
    if ( !(PINB & (1<<SW1)) )  // Ak je stlacene tlacitko (t.j. na vstupe log.0)...
      {
      PORTB |=  (1<<LED1);     // ... tak rozsviet LED, t.j. set PB5 na log. 1
      }                        // predosly riadok prekladac vezme ako bitovu operaciu
    else                       // ... v opacnom pripade
      {
       PORTB &= ~(1<<LED1);    // zhasni LED, t.j. clear PB5 na log. 0
      }
    }  /* end of while */
}
```

Toto je pripomienka k cvičeniam, že programy treba komentovať, ale nie bezhlavo skopírovať komentáre zo vzorového príkladu. Komentáre nemajú inými slovami opísať čo je na danom riadku, ale majú vysvetliť PREČO tam daný príkaz je a akú má funkciu v programe.

Mikropočítačové Systémy - MIPS

Prednáška 2
LCD displej

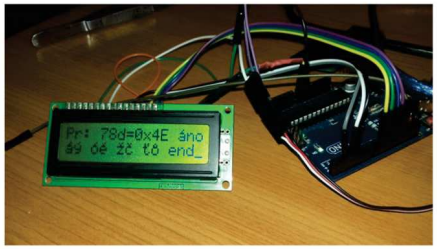
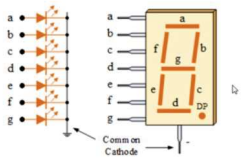
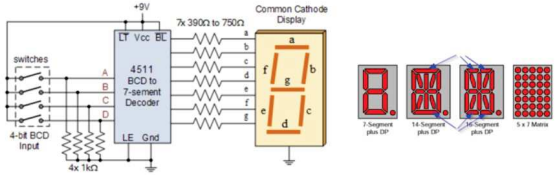


Image:
https://www.flickr.com/photos/4lkna/24804252238

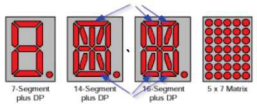
Image credit: David Falkner / flickr

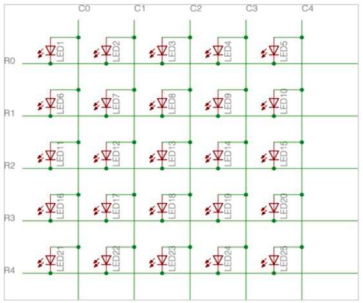
7-segmentový LED





Maticový displej



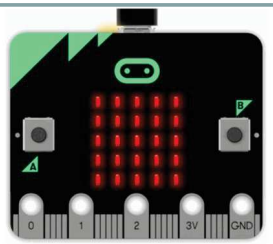


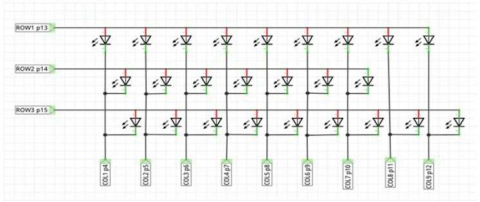
Komentár podobný nasledujúcemu.

5

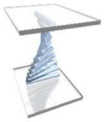
6

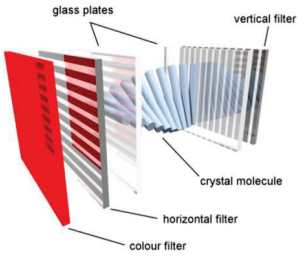
Maticový displej





7-segmentový LCD

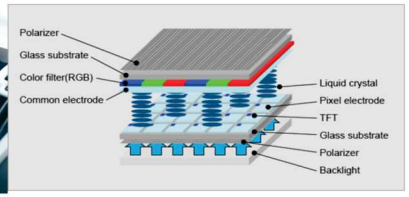




7

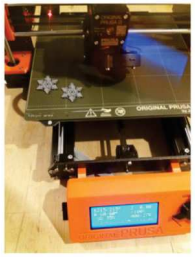
8

An example of a liquid crystal molecule



Structure of a TFT LCD

Aplikácie

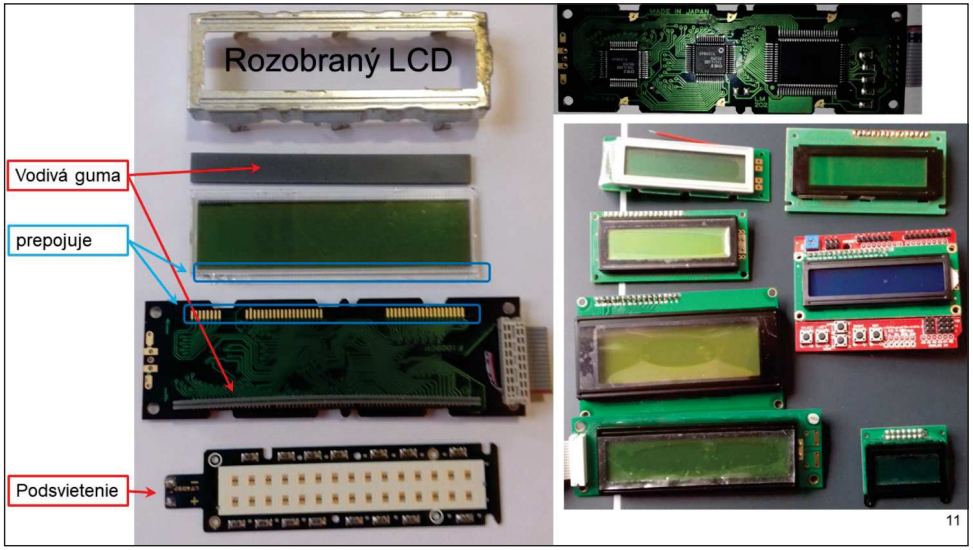


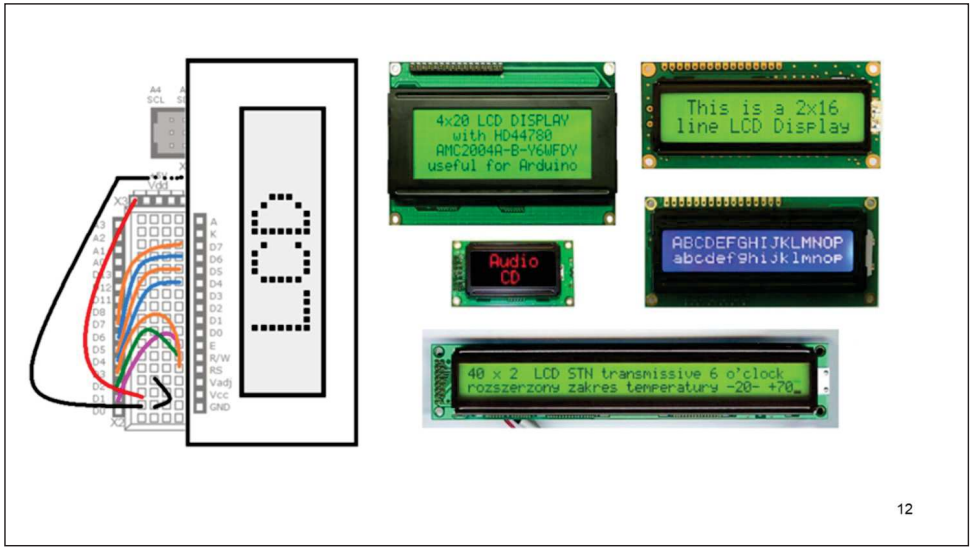
LCD display - zobrazovač je výstupne zariadenie, pomocou ktorého kc

LCD zobrazovače sa rozdeľujú na:

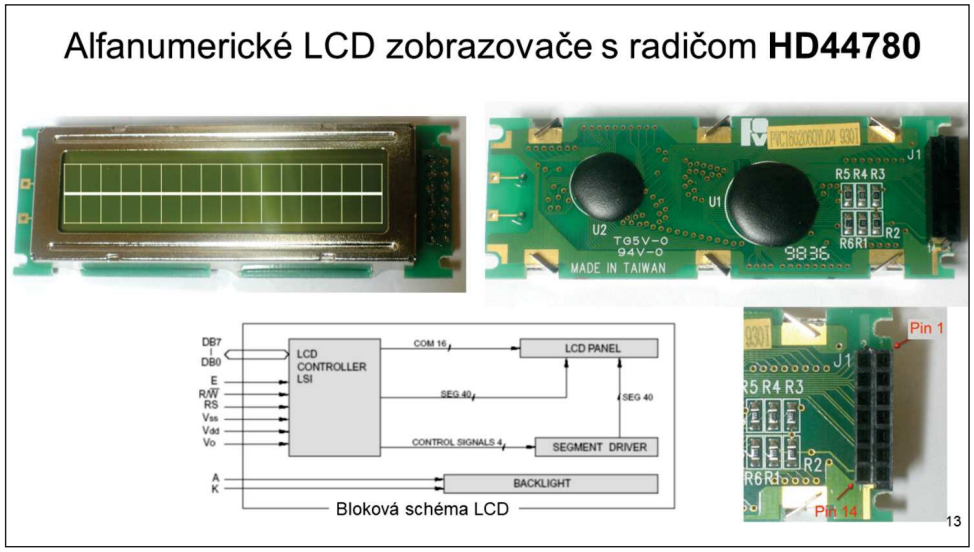
- **znakové** (majú pevné miesto na displeji),
- **alfanumerické**, zobrazovač je rozdelený na riadky a stĺpce v ktorých
- **grafické,** zobrazujú sa body na definovaných súradniciach displeja.

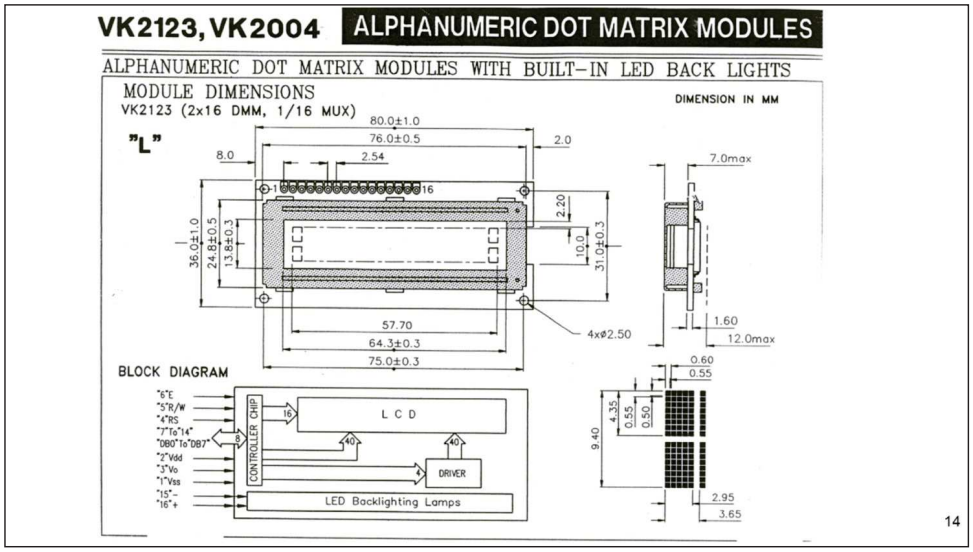
An example of a liquid crystal molecule
https://www.j-display.com/english/technology/lcdbasic.html





LCD zobrazovač vyzerá ako lego skladačka, len na prvý dojem. Rozobrať ho nie je problém, ale zložiť ho, problém je. Obrázok napravo je len časť z toho čo som kedy použil.





My budeme používať alfanumerický display 2x16 znakov.

To znamená 16 znakov v riadku a máme k dispozícii dva riadky.

Na riadenie LCD zobrazovačov sa používa radič HD44780, ktorý je štandardom v tejto oblasti. 44780 komunikuje s MMP pomocou 3 riadiacich signálov a 4, resp. 8 dátových signálov.

S nepatrnými úpravami ho možno pripojiť k systémovej zbernici: DB,CB a AB mnohých mikropočítačov.

Aj keď „44780“ sa objavil v čase „8080“, „8051“ nedá sa priamo pripojiť na zbernice DB, CB a AB.

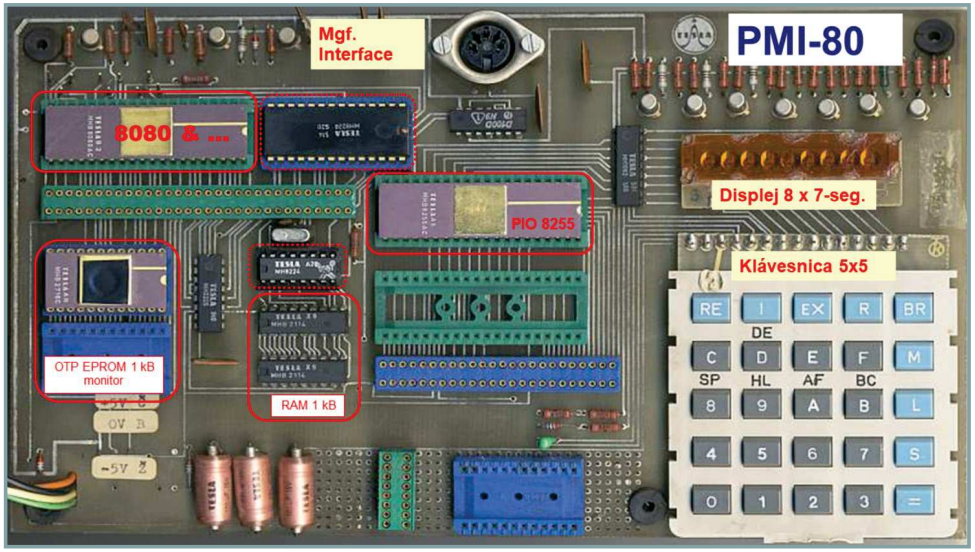
Pre CB procesora 8051 je typické

/RD – aktívny do nuly

/WR – aktívny do nuly

Ale LCD zobrazovač má spoločný signál RD/WR.

Takto vyzerala titulná strana KL display-a, s ktorým sme začínali cca pred 30-mi rokmi.



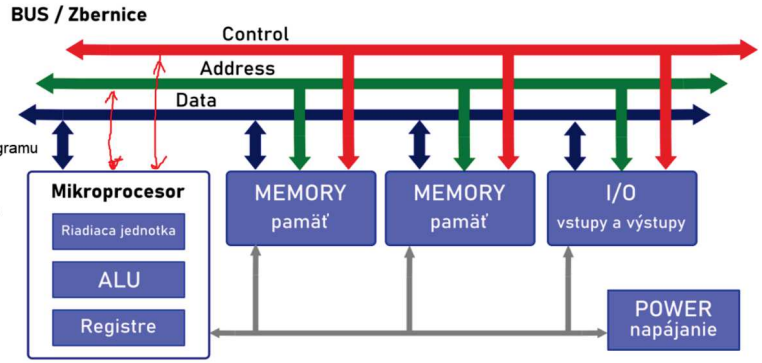
Štruktúra počítača

Procesor
riadi, realizuje výpočty

Pamäť
pamäť dát & pamäť programu

Input/Output
interface na okolitý svet

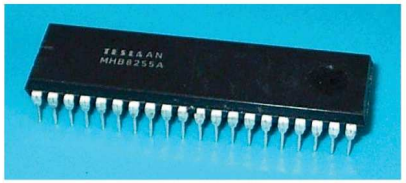
Napájanie

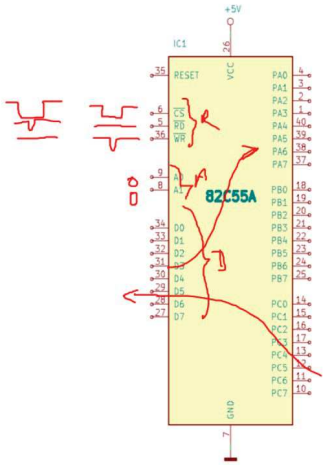


15

16

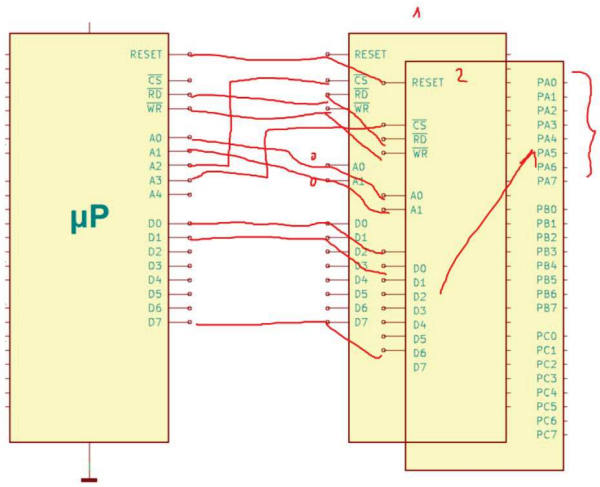
i8255



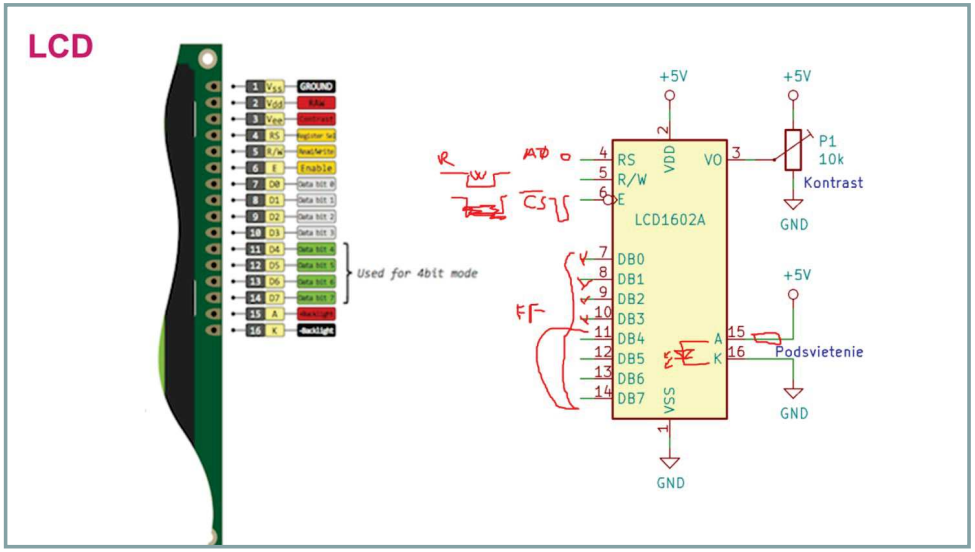


A_1	A_0	Port selected
0	0	port A
0	1	port B
1	0	port C
1	1	control register

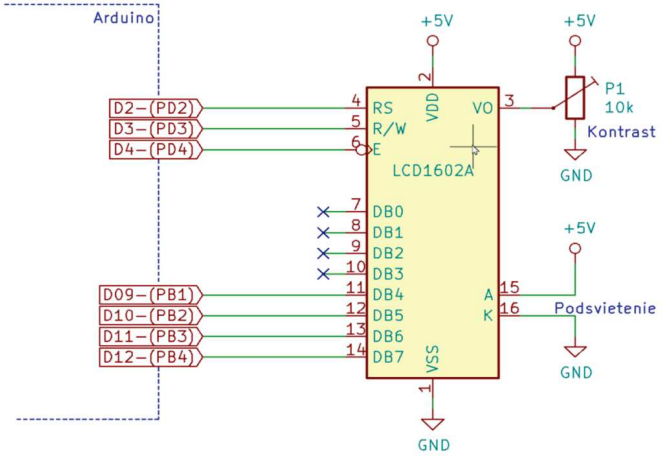
2x i8255



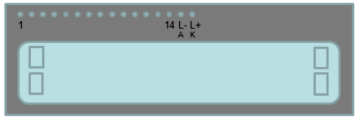
LCD


1 Vss GROUND
2 Vdd
3 Vee Contrast
4 RS Register Sel
5 R/W Read/Write
6 E Enable
7 D0 Data bit 0
8 D1 Data bit 1
9 D2 Data bit 2
10 D3 Data bit 3
11 D4 Data bit 4
12 D5 Data bit 5
13 D6 Data bit 6
14 D7 Data bit 7
15 A Backlight
16 K Backlight
Used for 4bit mode
+5V
+5V
P1
10k
Kontrast
GND
RS
R/W
E
VDD
VO
LCD1602A
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
VSS
A
K
Podsvietenie
+5V
GND
GND

LCD


Arduino
D2–(PD2)
D3–(PD3)
D4–(PD4)
D09–(PB1)
D10–(PB2)
D11–(PB3)
D12–(PB4)
+5V
+5V
P1
10k
Kontrast
GND
RS
R/W
E
VDD
VO
LCD1602A
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
VSS
A
K
Podsvietenie
+5V
GND
GND

Niekoľko možností umiestnenia konektora a priradenie signálov k pinom:



L+ ; L- ; A ; K

L- ; L+ ; K ; A

Väčšina výrobcov dodržuje priradenie signálov k pinom 1 až 14. Nejednoznačné je to pre piny, ktoré sú vyhradené pre podsvietenie.

Len jeden výrobca sa rozhodol **prehodiť funkciu pinov 1 a 2.**

Číslo pinu	Symbol
1	Vss
2	Vcc
3	Vee
4	RS
5	R/W
6	E
7	DB0
8	DB1
9	DB2
10	DB3
11	DB4
12	DB5
13	DB6
14	DB7

21

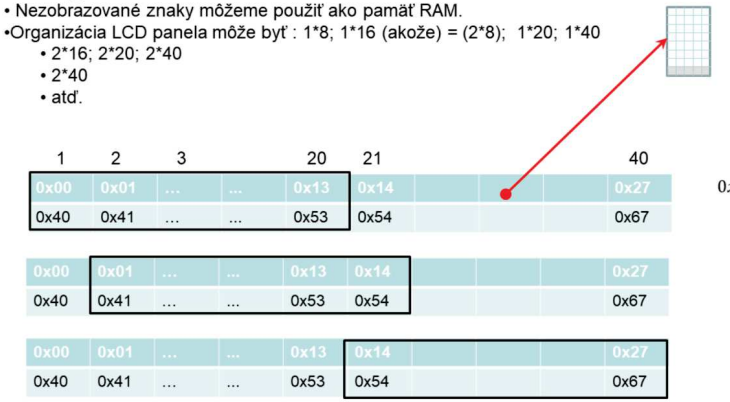
!!!! Pozor. !!!! Číslovanie pinov display-a je niekedy uvedené na hornej, niekedy na dolnej strane. Konektor by sme mali osadiť zo spodnej strany. Hlavne sa to týka dvojradových konektorov. Niekedy z popisu a celkovej dispozície display-a nie je jasné čo je „horný ľavý“ znak display-a.

Vývody, piny modulu display-a sú očíslované a majú pevnú funkciu (vývody 15, 16 sú iba pri niektorých typov display-ov a sú použité na podsvietenie display-a). Poznamenajme, že ak kúpime dva navonok rovnaké display-e, ale od rôznych výrobcov, nemusí tomu istému pinu odpovedať ten istý signál. ((Niekedy je Pin15 = LED+, a niekedy je Pin16 = LED+). Problémy môže spôsobovať aj označenie pinu pre pripojenie „kontrastu“ displeja.

DDRAM (Display Data RAM).

Kapacita DDRAM je 80 znakov (5*7 bodov) :

- Kapacita pamäte je 80 B. Jeden byte je priradený k jednému znaku. Teoreticky máme 256 možností toho čo na danej pozícii zobrazíme.
- Nezobrazované znaky môžeme použiť ako pamäť RAM.
- Organizácia LCD panela môže byť : 1*8; 1*16 (akože) = (2*8); 1*20; 1*40
  - 2*16; 2*20; 2*40
  - 2*40
  - atď.



$0x27 \cong 39\ dekadicky$

22

Na zobrazovaciu časť display-a sa môžeme pozerať ako na okno nad DDRAM. To čo je v okne, je vidieť. Ak chceme toho vidieť viac musíme okno pohnúť doprava, resp. doľava.

Jedno-riadkové vs. Viac-riadkové displeje

N = 0, → Jednoriadkový display

Displej 1x8

1	2	3	4	5	6	7	8	Dek.
0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	Hex.

N = 1, → "Dvojriadkový" display. V opačnom prípade sa znaky na pozíciách 08H – 39H nezobrazia!

Displej 1x16

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Dek.
0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x40	0x41	0x42	0x43	0x44	0x45	0x46	0x47	Hex.

INSTRUCTION	CODE										DESCRIPTION	Time (250 kHz)
E=?	RS	R/W	db 7	db 6	db 5	db 4	db 3	db 2	db 1	db 0		
Set the DD RAM address	0	0	1	MSB			ADD			LSB	DDRAM data is sent and received after this setting.	40 [us]

„1." riadok	0	0	0	0	0	0	0	Adresy: 0x00 až 0x3F
	0	1	1	1	1	1	1	
„2." riadok	1	0	0	0	0	0	0	Adresy: 0x40 až 0x7F
	1	1	1	1	1	1	1	

23

Jedno- a viac-riadkové displeje

Displej 2x16

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Dek.
0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F	Hex.
0x40	0x41	0x42	0x43	0x44	0x45	0x46	0x47	0x48	0x49	0x4A	0x4B	0x4C	0x4D	0x4E	0x4F	Hex.

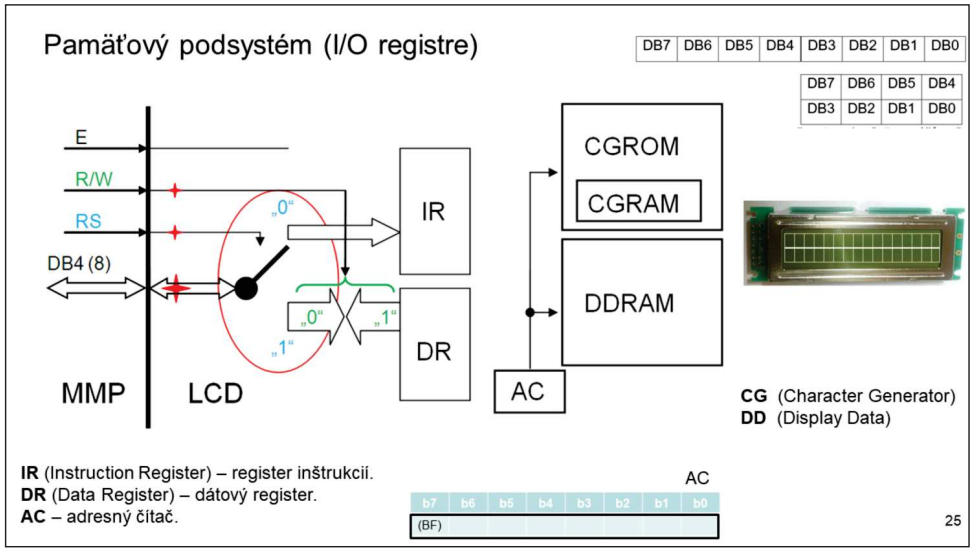
N = 1, → "Dvojriadkový" display
!!!! Pozor druhý riadok nezačína na „konci" prvého !!!

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Dek.
0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F	0x10	Hex.
0x41	0x42	0x43	0x44	0x45	0x46	0x47	0x48	0x49	0x4A	0x4B	0x4C	0x4D	0x4E	0x4F	0x50	Hex.

0x00	0x01	…	…	0x0F					0x27
0x40	0x41	…	…	0x4F					0x67

24

Z príkazu „Set the DD RAM adress" je zrejmé, že dokáže adresovať až 2 krát 64 adries, ale implementovaných má len 2 krát 40.



Červené hviezdičky na obrázku predstavujú pullup „odpory“, viď. KL obvodu HD44780.

Display komunikuje s okolím cez 3 piny (W, R/W a RS) riadiacej zbernice a 8 /(4) piny dátovej zbernice.

DBUS (DataBus – má vnútorné pullup-y) je tvorená 4, resp. 8 vodičmi. Závisí to od užívateľom zvoleného módu. Pri *osembitovom* móde sa využívajú všetky dátove piny: DB0, DB1, ..., DB7. Pri *štvorbitovom* móde sa využívajú len piny DB4, DB5, DB6 a DB7. Najskôr sa prenesú dátové bity DB7 až DB4 a potom DB3 až DB0. . T.j. najskôr High nibble potom Low nibble.

Display, ako periféria MMP môže byť mapovaná do adresného priestoru pamäte RAM procesora, resp. MMP alebo môže byť pripojená na I/O PORT-y. Komunikácia display-a s MMP sa realizuje cez dva osem bitové registre.

IR (Instruction Register) – register inštrukcií.

DR (Data Register) – dátový register.

Na adresovanie je použitý jeden adresovací pin RS (Register Select). Pin E (Enable) je vo funkcii čip selektu.

AC (Adress Counter) – adresné počítadlo, je vnútorný register radiča. Adresujeme pomocou neho CGROM, CGRAM a DDRAM. Bit B7 adresného čítača (AC) je tzv. **BF** - (Busy Flag).

Vnútorne je pamäť displeja členená na tri časti:

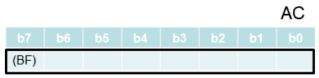
DDRAM (Display Data RAM). Kapacita tejto pamäte je 80B. Ak je display 1x16, potom môžeme zobraziť 16 znakov a zvyšok môžeme použiť ako pamäť RAM pre všeobecné použitie. Organizácia LCD panela môže byť aj 2x40, aj napr.. 4x20.

CGROM (Character Generator ROM). Ak si položíme otázku čo sa zobrazí na paneli, ak na príslušnú pozíciu pošleme ASCII kód znaku? Odpoveď je takáto: To, čo sa nachádza v pamäti CGROM na odpovedajúcom mieste. Je viac menej „záhadou“ čo sa nachádza v horne polovici ASCII tabuľky.

CGRAM (Character Generator RAM). Už z názvu je jasné, že časť pamäte znakov (vzorov) si môžeme naprogramovať sami. ASCII kódy 0x00 až 0x0F sú vyhradené pre vlastné naprogramovanie. Problémom, je že nemáme k dispozícii 16 kódov, teda znakov, ale len 8 znakov. Znaky s adresami 0x00 až 0x07 sa zrkadlia do časti pamäte 0x08 až 0x0F. Keďže radič display-a nemá RESET, po pripojení napájania je obsah tejto pamäte náhodný.

Riadiacu zbernicu - CB tvoria:

- EN (Enable). Výberový signál vo funkcii CS.
  - Signál je aktívny do „1“
  - Dobežná hrana vykoná príkaz definovaný pomocou: DB, RS, a RW
  - ale nemôžeme tvrdiť, že stačí vygenerovať, dobežnú hranu.
  - Ak bude High úroveň kratšia ako PW_{EH}, viď. KL, dobežná hrana nespôsobí nič.
- RS (Register Select).
  - RS = „0“ ⇒ dáta sa spracujú ako inštrukcia LCD (napr.: zmaž LCD)
  - RS = „1“ ⇒ dáta sa zobrazia na LCD ako znak
- RW(Read/Write ≡ „1“/„0“).
  - RW = „0“ informácia na DB sa zapíše do LCD
  - RW = „1“
    - Test stavu LCD (Get LCD Status)
    - „čítanie“ z LCD



26

EN (Enable – nemá vnútorný pullup) Komunikácia s LCD displejom sa zahajuje „nábežnou hranou“ tohto signálu. Podľa časového priebehu signálov by mali byť riadiace signály (RS a R/W) nastavené pred nábežnou hranou EN. Potom sa data vyčítajú z dátovej zbernice, resp. zapíšu, ak treba, na dátovú zbernicu. „Dobežnou hranou“ tohto signálu sa príkaz spracuje. Počas spracovávania je radič displeja „BUSY“. T.j. nie je schopný spracovať ďalší príkaz.

Display nedokáže spracovať dva príkazy naraz. Oneskorenie závisí od typu inštrukcie a od použitého oscilátora v LCD (250 kHz, 270 kHz). Tento nedostatok sa dá eliminovať dvomi spôsobmi:

- po každom príkaze sa počká definovaný čas.
- oveľa efektívnejšou je metóda pri ktorej sa pred každou inštrukciou – príkazom otestuje stav radiča displeja.

BF - (Busy Flag) Tento bit, príznak nás informuje o stave radiča displeja. Ak pošleme príkaz alebo dáta do LCD, ich spracovanie trvá určitú dobu. Počas vykonávania operácie, sa tento príznak nastaví (BF =“1“). Pullup ho ťahá do jednotky. Potom ako sa príkaz vykoná, tento bit sa vynuluje (BF =“0“). Obsah tohto bitu vieme prečítať tak, že spracujeme príkazom “Get_LCD_Status”. Vykoná sa: RS = „0“ R/W = „1“. V okamžiku odoslania tohto príkazu (zacvičíme signálom E) obvod 44780 nastaví DB7 do “1”, ak je ešte stále spracovávaný predchádzajúci príkaz. Z uvedeného je zrejmé: Pred každou činnosťou s LCD, treba zistiť, či je BF = „0“. Je zrejmé, že ak v cykle testujeme DB7 a LCD nezmení stav tohoto bitu, program “zamrzne”. Z tohto dôvodu sa programovo povolí len určitý počet testov. Niečo ako timeout.

RS (Register Select – má vnútorný pullup)

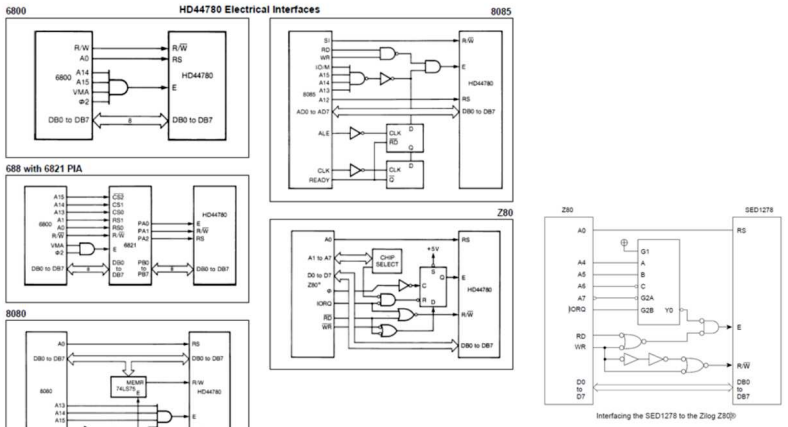
- RS=“0”. Dáta sa spracujú ako príkaz alebo špeciálna inštrukcia, napr. zmazanie display-a.
- RS=“1”. Dáta sa zobrazia na LCD ako znak.

R/W (Read/Write – má vnútorný pullup) Nepísané pravidlo hovorí, že slovu Read odpovedá „log.1“ (je pred znakom negácie) a slovu Write odpovedá „log.0“ (negácia „log. 1“). Ak

- R/W= “0” - informácia z DBUS sa zapíše do display-a.
- R/W= “1”, -buď sa testuje stav displeja alebo sa z neho číta.

26

Možnosti pripojenia LCD k systémovej zbernici MMP: R/W ↔ /R, /W



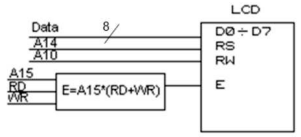
27

Niekoľko spôsobov pripojenia LCD k mikropočítačom.

Je zrejmé, že priame pripojenie radiča 44780 k systémovej zbernici je problematické.

Radiče LCD SED1278 HD 44780 and HD 66780 sú ekvivalentné. Len v KL sú „“nepatrné rozdiely.

Číslo pinu	Symbol	I/O	Funkcia
1	V_{SS}	-	0V napájanie
2	V_{DD}	-	+5V napájanie
3	V_{EE}	-	Kontrast
4	RS	I	0 = vstup je inštrukcia 1 = vstup sú dáta
5	R/W	I	0 = zápis dát do LCD 1 = čítanie dát z LCD
6	E	I	Aktivácia displeja
7	DB0	I/O	Dáta, bit 0
8	DB1	I/O	Dáta, bit 1
9	DB2	I/O	Dáta, bit 2
10	DB3	I/O	Dáta, bit 3
11	DB4	I/O	Dáta, bit 4
12	DB5	I/O	Dáta, bit 5
13	DB6	I/O	Dáta, bit 6
14	DB7	I/O	Dáta, bit 7



Čas trvania pulzu Enable: min. 450ns

Nech: Procesor rady 80C51 (80C552 má frekvenciu oscilátora $f_{osc} = 12MHz$.

Čas trvania signálu RD/WR je podľa KL

$t_{RD\,WR\,min.} = 6 * t_{OSC} - 100ns = 400\,ns$

Vyhovuje $f_{osc} = 11{,}0592MHz$.

a10 = 1/0 : R/W ,
a14 = 1/0 : data/inštrukcia ,
a15 = 1: (CS “displej”)

a15	a14	a13	a12	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2	a1	a0
1	RS	x	x	x	R/W	x	x	x	x	x	x	x	x	x	x

28

Pripojenie display-a k MMP (80C51) je možné pomocou zberníc (AB, DB, CB) resp. pomocou I/O portov. Tu je pripojenie realizované pomocou systémovej zbernice. Podľa KL procesorov tejto rady trvá $t_{CLK}=t_{OSC}=1/(12MHz)$. Trvanie impulzu RD, resp. WR je 400ns a pre $f_{OSC}=12MHz$.

T.j. nevyhovuje. Ak však chceme nastaviť presne prenosové rýchlosti USART-u, treba zvoliť $f_{OSC}=11{,}0952MHz$, čomu odpovedá trvanie impulzu RD, resp. WR minimálne 443 ns, čo je na hranici dovoleného minimálneho času. Poznamenajme, že už tento procesor dokázal pracovať až na frekvencii oscilátora $f_{OSC}=16MHz$.

Tento jednoduchý príklad ukazuje, že už v čase vzniku radiča 44780 bol tento „pomalou“ perifériou.

Prepojenie: MMP ⇔ Display. Priradenie pinov. Orientácia.

1.

b7	b6	b5	b4	b3	b2	b1	b0	PORT B
			D7	D6	D5	D4		Display
			D3	D2	D1	D0		DATA

write: `DDRB |= (1<<1)|(1<<2)|(1<<3)|(1<<4);  // Piny 1,2,3,4, PORTB ako output (Data pre display)`

read: `DDRB &= ~((1<<1)|(1<<2)|(1<<3)|(1<<4)); // Piny 1,2,3,4, PORTB ako input (Data pre MMP)`

2.

```
#define PORTB_DATA_H(x) PORTB &=0b11100001; PORTB |= (x & 0xF0 )>>3;
#define PORTB_DATA_L(x) PORTB &=0b11100001; PORTB |= (x & 0x0F )<<1;
```

D7	D6	D5	D4	D3	D2	D1	D0	⟨x⟩

≙ *Display DATA*

3.

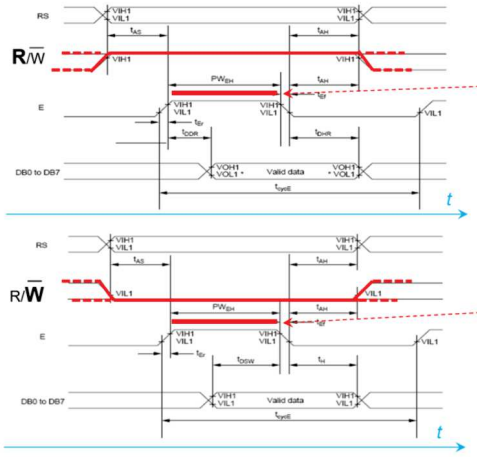
b7	b6	b5	b4	b3	b2	b1	b0	PORT D
			E	R/W	RS			

```
DDRD |= (1<<En_pin);    // Pin D4 (Enable)  PORTD  output
DDRD |= (1<<RW_pin);    // Pin D3 (RW)      PORTD  output
DDRD |= (1<<RS_pin);    // Pin D2 (RS)      PORTD  output
```

29

1. Podľa odporučenia máme display pripojiť 4-bitovo k pinom PORTB. Pre zápis dát do display-a piny 1, 2, 3, 4 nastavíme ako output a pre čítanie (BF) ich nastavíme ako input.
2. Pred zápisom High, resp. Low nibble data na piny PORTB treba ich správne vyposúvať a vymaskovať. Použijeme dve makrá. Jedno pre High a druhé pre Low niblle. Ak budeme dáta z display-a čítať (BF) budeme musieť urobiť opačný postup.
3. Bity 4, 3, 2 portu D sú vždy výstupné.

Časovanie kontrolera HD44780U



Read Operation

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	1000	—	—	ns
Enable pulse width (high level)	PW_{EH}	450	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	25	
Address set-up time (RS, R/W to E)	t_{AS}	60	—	—	
Address hold time	t_{AH}	20	—	—	
Data delay time	t_{DDR}	—	—	360	
Data hold time	t_{DHR}	5	—	—	

Write Operation

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	1000	—	—	ns
Enable pulse width (high level)	PW_{EH}	450	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	25	
Address set-up time (RS, R/W to E)	t_{AS}	60	—	—	
Address hold time	t_{AH}	20	—	—	
Data set-up time	t_{DSW}	195	—	—	
Data hold time	t_H	10	—	—	

30

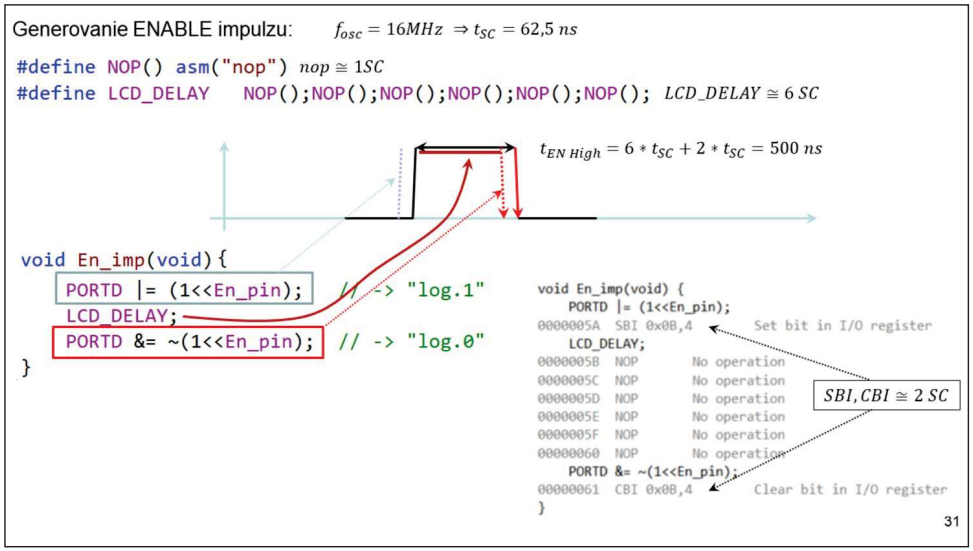
K časovaniu a k celkovému času trvania inštrukcií display-a treba uviesť, že sa vyrábajú s dvoma typmi RC oscilátora 250kHz a 270kHz. Pričom presnosť je +- 30%.

Časové priebehy – operácia Write. Ak komunikujeme s LCD display-om cez I/O Port(-y), dáta musia biť pripravené pred dobežnou hranou signálu E.

Časové priebehy – operácia Read. Ak komunikujeme s LCD display-om cez I/O Port(-y), dáta musia biť prečítané okamžite po dobežnej hrane signálu E.

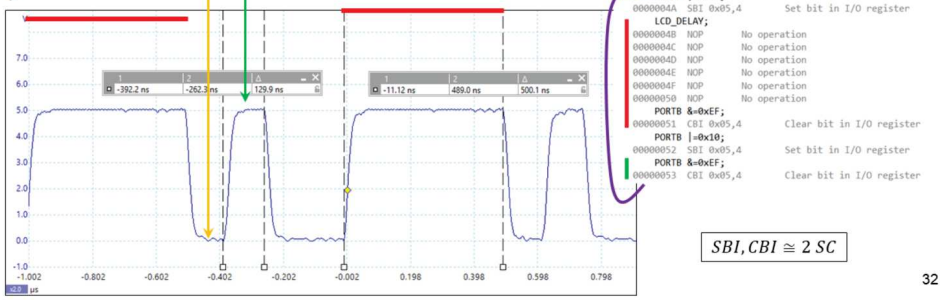
Dá sa povedať, čo KL to iné údaje o minimálnej hodnote času trvania Enable impulzu. Ten čas sa pohybuje v intervale 150ns až 500ns. Je vhodné vybrať ten najhorší prípad.

Na cvičeniach budeme používať AVR ATMEGA 328, ktorý dokáže pre 16MHz vygenerovať na I/O pinoch najkratší impulz 2SC (cca 130ns). Takto krátky impulz by mal byť už obvodmi LCD odfiltrovaný. Ale aj napriek tomu display pracoval správne.



Podľa KL-ov má Enable impulz trvať od 150 ns do 500 ns. Keďže nevieme, ktorý display (radič) bude použitý, treba generovať impulz odpovedajúci najhoršiemu možnému stavu. Nakoniec sme odskúšali aj najkratší možný impulz – cca 130ns. Viď. obr. z osciloskopu. Aj keď hovoríme o riadení display-a, máme na mysli SC AVR mikrokontrolera.

```
#define NOP() asm("nop")
#define LCD_DELAY   NOP();NOP();NOP();NOP();NOP();NOP();
#include <avr/io.h>
void main(void)
{   DDRB = 0x10;
    while(1){
        PORTB |=0x10;
        LCD_DELAY;
        PORTB &=0xEF;
        PORTB |=0x10;
        PORTB &=0xEF;
    }
}
```




```
DDRB = 0x10;
00000048  LDI R24,0x10      Load immediate
00000049  OUT 0x04,R24      Out to I/O location
    PORTB |=0x10;
0000004A  SBI 0x05,4        Set bit in I/O register
    LCD_DELAY;
0000004B  NOP               No operation
0000004C  NOP               No operation
0000004D  NOP               No operation
0000004E  NOP               No operation
0000004F  NOP               No operation
00000050  NOP               No operation
    PORTB &=0xEF;
00000051  CBI 0x05,4        Clear bit in I/O register
    PORTB |=0x10;
00000052  SBI 0x05,4        Set bit in I/O register
    PORTB &=0xEF;
00000053  CBI 0x05,4        Clear bit in I/O register
```

$$SBI, CBI \cong 2\ SC$$

INSTRUCTION / E=?	RS	R/W	db7	db6	db5	db4	db3	db2	db1	db0	DESCRIPTION	Time (250 kHz)	
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter. Sets I/D=1.	1.64 [ms]	410 cloks
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.	1.64 [ms]	
Entry mode set	0	0	0	0	0	0	0	1	I/D	S=0	I/D = 0/1 (Decrement/ Increment) These operations are performed during data R/W of DDRAM/CGRAM	40 [us]	10 cloks
Display ON/OFF control	0	0	0	0	0	0	1	D	C	B	D= 0/1 (OFF/ON Display) C= 0/1 (OFF/ON Cursor) B= 0/1 (OFF/ON Blink Cursor)	40 [us]	
Cursor or Display shift	0	0	0	0	0	1	S/C	R/L	*	*	S/C=0/1 Cursor/Display shift R/L=0/1 Left /Right	40 [us]	
Function set	0	0	0	0	1	DL	N	F	*	*	DL= 0/1 (4/8 bits) N= 0/1 (1/ 2(1) lines) F= 0 (5*7 dots)	40 [us]	
Set the CGRAM address	0	0	0	1	MSB		ACG			LSB	CGRAM data is sent and received after this setting.	40 [us]	
Set the DDRAM address	0	0	1	MSB			ADD			LSB	DDRAM data is sent and received after this setting.	40 [us]	
Read busy flag & address	0	1	BF	MSB			AC			LSB	Reads Busy flag & AC	40 [us]	?!? 0 [us] ?!?
Write data to CG or DD RAM	1	0	MSB							LSB	Writes data into DDRAM or CGRAM.	40 [us]	
Read data from CG or DD RAM	1	1	MSB							LSB	Reads data from DDRAM or CGRAM.	40 [us]	

Inštrukčná sada radiča displeja: Inštrukcie sa ľahko dekódujú. Stačí zistiť na ktorom mieste je po nulách jednotka. Veľmi dôležitý je údaj o čase trvania inštrukcie, aj keď je to len orientačný údaj. Už sa začína v KL objavovať aj bezrozmerný čas SC zariadenia.

```c
//zapise data do Data Register
void lcd_data(unsigned char data){
    //while(busy_flag());
    while(rd_BF());
    PORTD |= (1<<RS_pin);       //    (RS = High)
    PORTD &= ~(1<<RW_pin);      //    (RW = Low, write)
    wr_data (data);
}

// zapise command do Instruction Register
void lcd_command(unsigned char command){
    //while(busy_flag());
    while(rd_BF());
    PORTD &= ~(1<<RS_pin);      //    (RS = Low)
    PORTD &= ~(1<<RW_pin);      //    (RW = Low, write)
    wr_data (command);
}

void wr_data (unsigned char data) {
    PORTB_DATA_H(data);         // data High nibble
    En_imp();

    PORTB_DATA_L(data);         // data Low nibble
    En_imp();
}
```

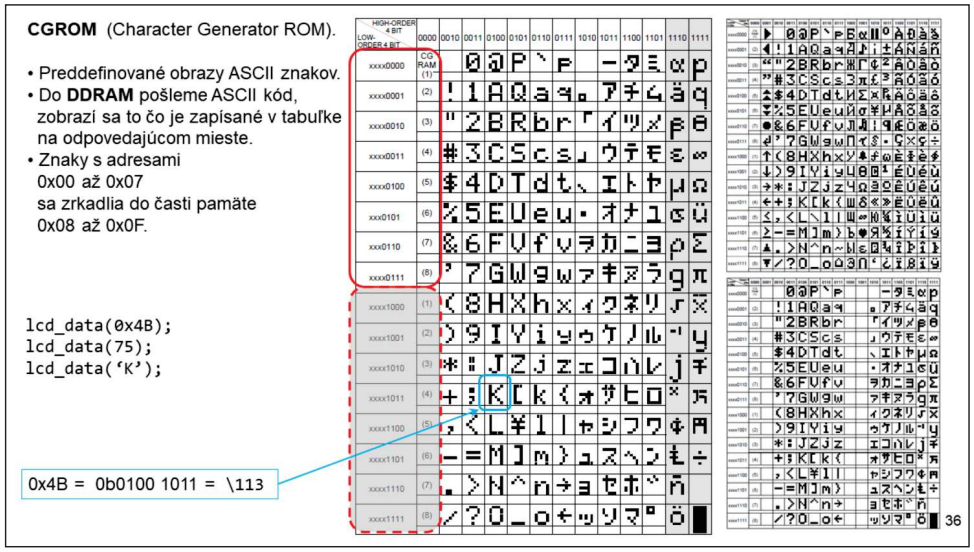
34

Podprogramy tu uvedené zapíšu dáta do IR, resp. DR registra display-a.

```
lcd_command(0x01);

lcd_command(0x02);

lcd_command(0b0000 1DCB);

lcd_command(0x80+0x40+3);

lcd_data(0x??);
```

INSTRUCTION	CODE										DESCRIPTION
	RS	R/W	db7	db6	db5	db4	db3	db2	db1	db0	
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter. Sets I/D=1.
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.
Entry mode set	0	0	0	0	0	0	0	1	I/D	S=0	I/D = 0/1 (Decrement/ Increment) These operations are performed during data R/W of DDRAM/CGRAM
Display ON/OFF control	0	0	0	0	0	0	1	D	C	B	D= 0/1 (OFF/ON Display) C= 0/1 (OFF/ON Cursor) B= 0/1 (OFF/ON Blink Cursor)
Cursor or Display shift	0	0	0	0	0	1	S/C	R/L	*	*	S/C=0/1 Cursor/Display shift R/L=0/1 Left /Right
Function set	0	0	0	0	1	DL	N	F	*	*	DL= 0/1 (4/8 bits) N= 0/1 (1/ 2(1) lines) F= 0 (5*7 dots)
Set the CGRAM address	0	0	0	1	MSB		ACG			LSB	CGRAM data is sent and received after this setting.
Set the DDRAM address	0	0	1	MSB			ADD			LSB	DDRAM data is sent and received after this setting.
Read busy flag & address	0	1	BF	MSB			AC			LSB	Reads Busy flag & AC
Write data to CG or DD RAM	1	0	MSB							LSB	Writes data into DDRAM or CGRAM.
Read data from CG or DD RAM	1	1	MSB							LSB	Reads data from DDRAM or CGRAM.

Inštrukčná sada radiča displeja: Inštrukcie sa ľahko dekódujú. Stačí zistiť na ktorom mieste je po nulách jednotka. Veľmi dôležitý je údaj o čase trvania inštrukcie, aj keď je to len orientačný údaj. Už sa začína v KL objavovať aj bezrozmerný čas SC zariadenia.



Obsah pamäte CGROM, kódová mapa znakov vo formáte 5x8 (5x7 +" "). Miesto medzi úvodzovkami je „miesto" na kurzor.

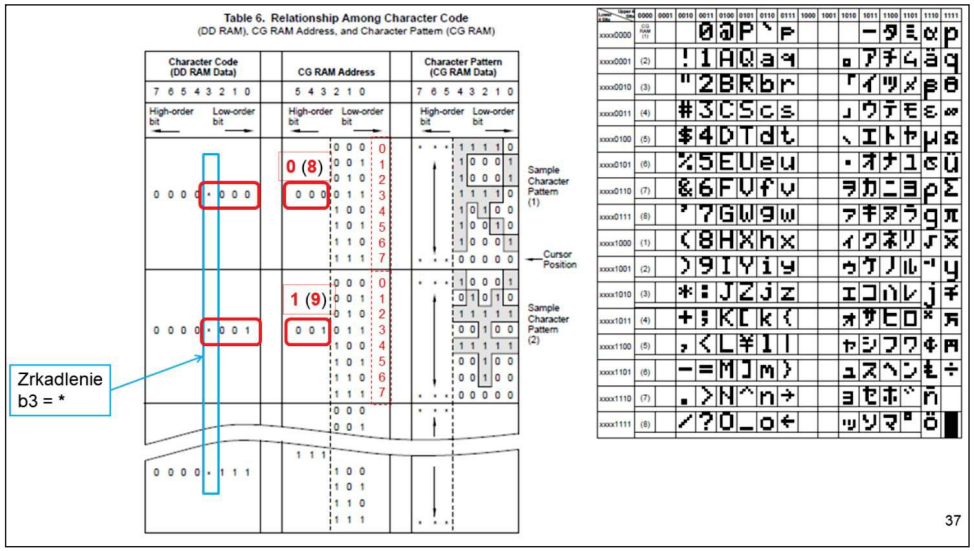
Väčšina displejov používa znakovú sadu danú ASCII tabuľkou (spodných 128 kódov). Na objednávku sa dajú získať aj znakové sady s ruštinou, ...

Prvých osem znakov v znakovej sade sa dá nastaviť, čo umožní na displeji tvoriť rôzne efekty, grafické symboly, slovenčinu alebo animácie. Pre slovenčinu však väčšinou osem znakov nepostačuje, čo si vynucuje meniť dynamicky generátor znakov podľa konkrétnej potreby **grafických** znakov.

Užívateľom definované znaky môžu byť na adresách 00H až 07H a zrkadlia sa na adresách 08H až 0FH.

Jeden zo spôsobov ako zaradiť znak do reťazca ASCII kódov zobrazovaného na LCD, je zapísať jeho poradové číslo v znakovej sade. T.j. ak chceme zobraziť text „**ABab**" ,môžeme prvý znak - **A**, zapísať oktálne \101. Zobrazené znaky budú v oboch prípadoch totožné. Ak by sme ale na pozíciu nula znakovej sady uložili napríklad obraz znaku **č**, a zapísali by sme zobrazovaný text "\000Bab", na display-i by sa nezobrazilo nič. Prekladač chápe číslo nula ako koniec reťazca.

Ak chceme zobraziť aj znak uložený v CGRAM na adrese NULA, pomôžeme si „zrkadlením". Adresa NULA je totožná s adresou OSEM.



Keďže b3 “chýba“, adresy 0 (0b0000) až 7 (0b0111) sa zrkadlia do priestoru adries 8 (0b1000) až 15 (0b1111).

Preddefinované znaky ASCII

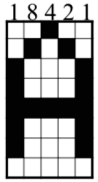
Znak (písmeno) *odpovedá* číselnému kódu.

Otom ako znak „vyzerá“ nám hovorí ***Font***.

3 2 1 0 3 2 1 0

1 8 4 2 1

Nazačiatku to vyzeralo asi takto:
Pr.:Font = 5*8(7+1)



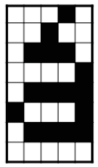
0x04
0x0A
0x11
0x11
0x1F
0x11
0x11
0x00 “kurzor”

Znak: **A**

ASCII kód: 0x41

Znak: **á**

ASCII kód: 0x??



0x02
0x04
0x0E
0x01
0x0F
0x11
0x0F
0x00 “kurzor”

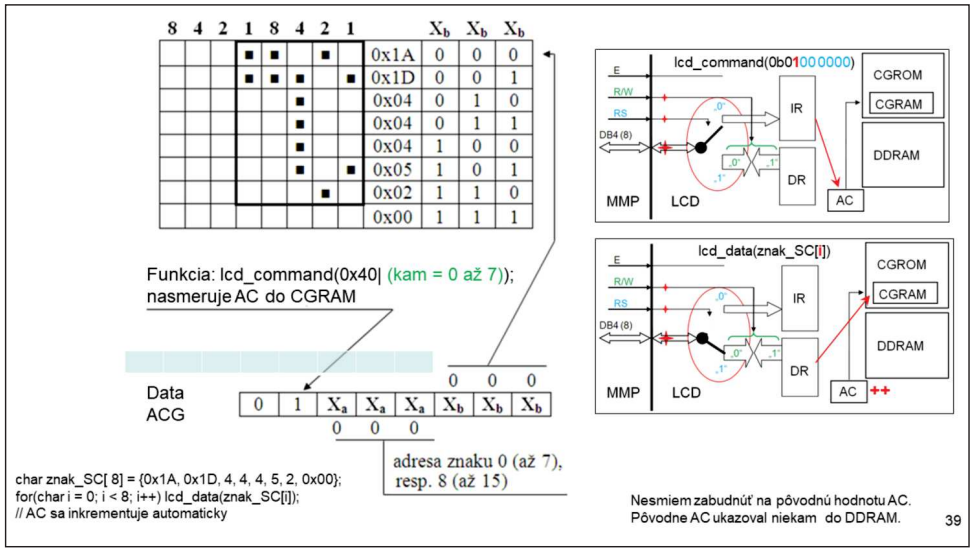
38

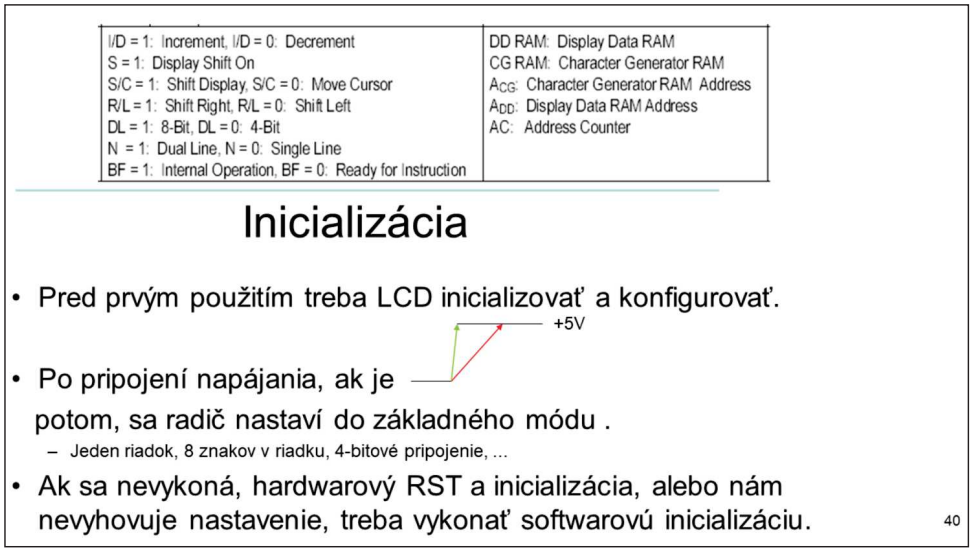
Vlastné znaky vieme vytvoriť v rastri: **5x7** alebo 5x10 bodov. Veľkosť znaku, ktorý vytvárame je 5x8 bodov, t.j. rozlíšenie 5 (šírka) x 7 (výška). 8-mi riadok „**ignorujeme“**. Zapisujú sa doňho nuly.

Bit nastavený do jednotky – pixel sa „zobrazí“. Bit nastavený do nuly – pixel sa „nezobrazí“.

Odpovedá to prázdnej pozícii – nesvieti. Na tomto mieste sa môže objaviť kurzor.

V móde 5x10 je to trochu iné. Nevidel som aplikáciu ktorá by zobrazovala znaky v móde 5x10.





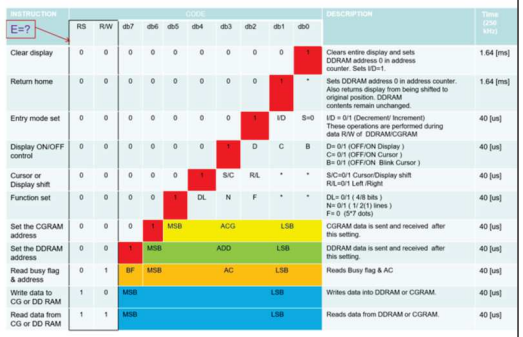
Pripomeňme: AC ukazuje aj do DDRAM aj do CGRAM.

Správny postup generovania vlastných znakov je nasledovný:

1. Odpamätáme obsah AC – ukazoval do DDRAM.
2. Nastavíme AC na prvú pozíciu znaku vytváraného v CGRAM.
3. Vytvoríme, zapíšeme znak.
4. Obnovíme obsah AC. Chceme pokračovať v písaní znakov do DDRAM.

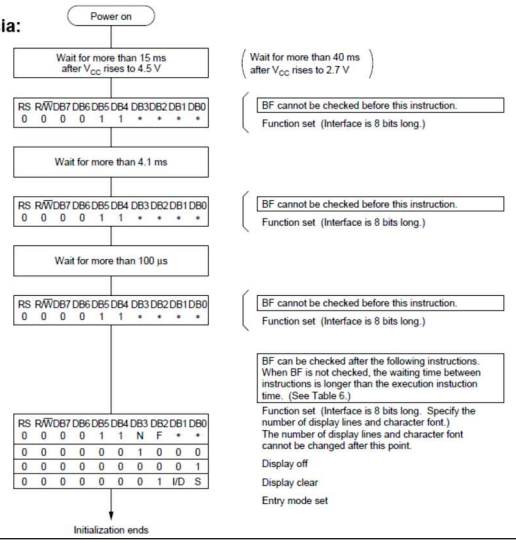
Reset vyvolaný po nábehu napájania:

1. Clear **Display (Kód inštr. = 0x01)**
   Kurzor: Ľavý horný roh. AC = 0x00. I/D = 1 (inkrement)
2. **Function Set (Kód inštr.= 0x30** (0x20|0x10))
   DL = 1 ... DB je 8-bitová.
   N = 0 ... Jednoriadkový display
   F = 0 .... Font 5x7
3. **Display ON/OFF Control (Kód inštr.= 0x08)**
   D = 0 ... Displej vypnutý.
   C = 0 ... Kurzor vypnutý.
   B = 0 .... Blikanie kurzora vypnuté.
4. **Entry Mode Set (Kód inštr.= 0x06** (0x04|0x2))
   I/D = 1 ... Inkrementovanie.
   S = 0 ... Posúvanie displeja vypnuté.



INSTRUCTION	CODE										DESCRIPTION	Time (250 kHz)
E=?	RS	R/W	db7	db6	db5	db4	db3	db2	db1	db0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter. Sets I/D=1.	1.64 [ms]
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.	1.64 [ms]
Entry mode set	0	0	0	0	0	0	0	1	I/D	S=0	I/D = 0/1 (Decrement/ Increment) These operations are performed during data R/W of DDRAM/CGRAM	40 [us]
Display ON/OFF control	0	0	0	0	0	0	1	D	C	B	D= 0/1 (OFF/ON Display) C= 0/1 (OFF/ON Cursor) B= 0/1 (OFF/ON Blink Cursor)	40 [us]
Cursor or Display shift	0	0	0	0	0	1	S/C	R/L	*	*	S/C=0/1 Cursor/Display shift R/L=0/1 Left /Right	40 [us]
Function set	0	0	0	0	1	DL	N	F	*	*	DL= 0/1 (4/8 bits) N= 0/1 (1/ 2(1) lines) F= 0 (5*7 dots)	40 [us]
Set the CGRAM address	0	0	0	1	MSB		ACG			LSB	CGRAM data is sent and received after this setting.	40 [us]
Set the DDRAM address	0	0	1	MSB			ADD			LSB	DDRAM data is sent and received after this setting.	40 [us]
Read busy flag & address	0	1	BF	MSB			AC			LSB	Reads Busy flag & AC	40 [us]
Write data to CG or DD RAM	1	0	MSB							LSB	Writes data into DDRAM or CGRAM.	40 [us]
Read data from CG or DD RAM	1	1	MSB							LSB	Reads data from DDRAM or CGRAM.	40 [us]

Softvérová inicializácia:
8 - bit Interface



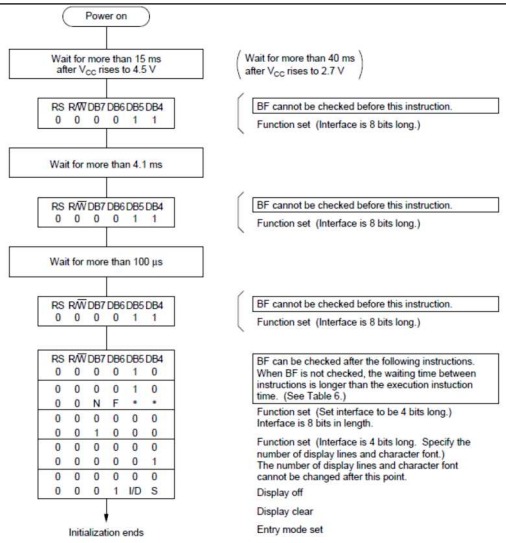
Ak typ display-a neodpovedá inicializácii po nábehu, treba vykonať softvérovú inicializáciu.

Dá sa povedať, že je to niečo ako alchýmia. Postupnosť príkazov a časy treba dodržať. Netreba sa pýtať prečo.

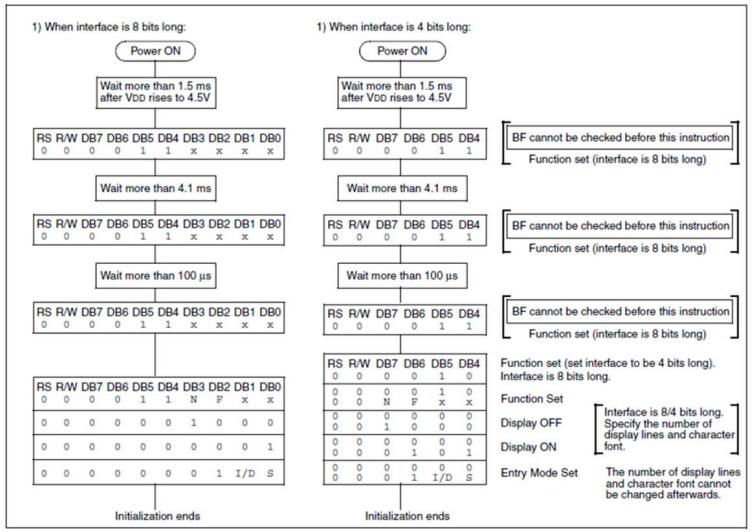
Ak nám aj napriek správne vykonanej inicializácie nebude display „svietiť", položme si otázku. Máme správne nastavený kontrast? Vlastne začali sme odpovedať na otázku s predchádzajúcej strany.

Niekedy to vyzerá tak, že KL tvorili ľudia, ... Viete okamžite zistiť, ktorý riadok komentára patrí ku ktorému inicializačnému príkazu.

Softvérová inicializácia:
4 - bit Interface



4-bitová inicializácia je trocha zložitejšia. Musíme byť dôsledný. Nezabudnime počet NIBBLOV je párny.



Zrozumiteľnejšie zapísaná inicializácia.

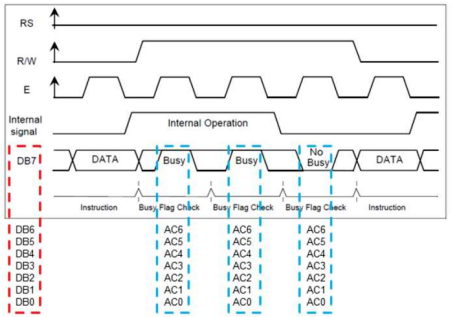
Správne vykonávanie operácií

Čas trvania inštrukcií t_v je premenlivý, viď. tab. inštrukcie display-a.
$T_v = f(f_{OSC\ disp}, \text{typ instr.})$; $f_{OSC\ disp} = 270, 250kHz \pm 30\%$

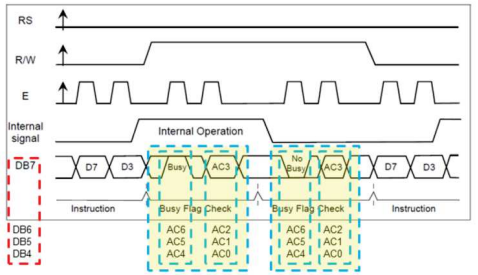
Počas spracovávania príkazu, LCD ďalší príkaz neakceptuje.
Máme dve možnosti:

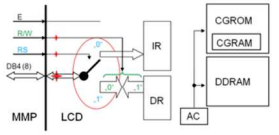
- „Počkáme“
- Testujeme stav signálu BUSY (bitová premenná)
  - Ak je LCD BUSY, potom je DB7 = „1“, ak je predchádzajúca operácia ukončená DB7 = „0“

Časovanie:8-b DB



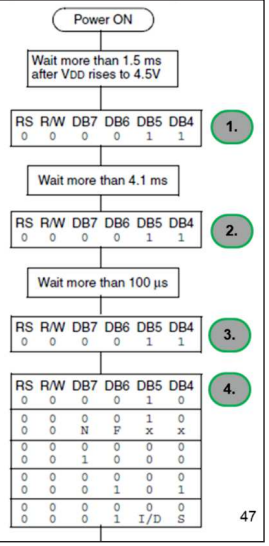
Časovanie:4-b DB





Prvá časť oboch obrázkov predstavuje zaslanie príkazu do display-a. Druhá je testovanie stavu busy. Aj keď by nám pri 4-bitovej komunikácii stačilo vyčítať prvé 4 bity, príkaz treba dokončiť. Ak čítame obsah AC, na pozícii b7 sa objaví stav BF.

```
char lcdInit4(void)
{   // 4 bitove pripojenie display-a
    _delay_ms(15);
    // 1. -------------------
    PORTD &= ~(1 << RS_pin);        // set RS  = to "log. 0"  Instruction Register (IR)
    PORTD &= ~(1 << RW_pin) ;       // set R/W = to "log. 0" - Write
    // RS R/W   DB7 DB6 DB5 DB4   DB3 DB2 DB1 DB0
    // 0  0     0   0   1   1     x   x   x   x     = 0x30
    PORTB_DATA_H(0x30); // 8-bitove pripojenie
    En_imp();
    // 2. -------------------
    // zopakujem  8-bitove pripojenie
    _delay_ms(5); En_imp();
    // 3. -------------------
    // zopakujem  8-bitove pripojenie
    _delay_ms(1); En_imp(); // -  stacilo by delay 0.1ms, momentalne najkratsie nastavitelne je 1ms
    // 4. -------------------
    // zmenim na 4-bitove pripojenie
    // RS R/W    DB7 DB6 DB5 DB4    DB3 DB2 DB1 DB0
    // 0  0      0   0   1   0      x   x   x   x    = 0x20
    PORTB_DATA_H(0x20); // 4-bitove pripojenie
    _delay_ms(1); En_imp(); // -  stacilo by delay 0.04ms
    // -------------------
    // LCD function set      mod: DL = 0 - 4-bitove data, N = 1 - 2 riadky, F = 0 - 5x7 dots
    // RS R/W    DB7 DB6 DB5 DB4    DB3 DB2 DB1 DB0
    // 0  0      0   0   1   DL     N   F   x   x     = 0x28
    _delay_ms(1); wr_IR(0x28);
    // RS R/W    DB7 DB6 DB5 DB4    DB3 DB2 DB1 DB0
    // 0  0      0   0   0   0      0   0   0   1     = 0x01 , Display clear
    _delay_ms(2); wr_IR(0x01);
    // RS R/W    DB7 DB6 DB5 DB4    DB3 DB2 DB1 DB0
    // 0  0      0   0   0   0      0   1   I/D S      = 0x06, I/D = 1 - inkrement
    _delay_ms(2); wr_IR(0x02);
    // RS R/W    DB7 DB6 DB5 DB4    DB3 DB2 DB1 DB0
    // 0  0      0   0   0   0      1   D   C   B      = 0x0E, D = C = 1 - Display a kurzor zapnute
    _delay_ms(1); wr_IR(0x0E);                              // B = 0 - blikanie vypnute
}
```



Posledné tri príkazy pre radič display-a sa úplne nezhodujú s doporučením. Je to chyba? Bude to fungovať? Odskúšajte!

```
int busy_flag(void){   // namiesto testu BF pockam dlhsie ako je trvanie najdlhsieho prikazu
    _delay_ms(2);
    return(0);
}
int rd_BF( void){      // test BF
    unsigned char pom;
    PORTD &= ~(1<<RS_pin);  //    (RS = Low)
    PORTD |= (1<<RW_pin);   //    (RW = High, read)
    // port B, datove bity budu teraz input
    DDRB &= ~((1<<1)|(1<<2)|(1<<3)|(1<<4)); //  piny 1,2,3,4 PORTB nastavime output, (Data pre disp)
    // Spravne by sa malo vycitavaj pred dobeznou hranou EN impulzu. Je tam urcity presah.
    En_imp();                       // Mozeme vycitat aj po dobeznej hrane.
    pom = (PINB & 0b00011110)<<3; // vycitam High nibble AC
    En_imp();
    pom |= (PINB & 0b00011110)>>1; // vycitam Low nibble AC
    // PORTB, datove bity zase output
    DDRB |= (1<<1)|(1<<2)|(1<<3)|(1<<4);    // piny 1,2,3,4 PORTB nastavime input (Data pre disp)
    if(pom & 0x80) return(1);   // display je busy
    else
    return(0);                  // display je not busy
}
```

```
void def_znak(unsigned char *ZnakArray,unsigned char kam) {
    lcd_command(0x40|(kam << 3)); //nastavenie adresy znaku v CGRAM
    for(unsigned char i = 0; i < 8; i++) lcd_data( *(ZnakArray + i));
}
```

Pri písaní programu sme si pomohli aj zabudovanými knižnicami.

```
#include <util/delay.h>
```

V tejto knižnici nájdeme aj funkciu:
_delay_ms(„počet milisekund“);

```c
int main(void) {
    ini_ports(); // ini LCD ports
    lcd_init();  // in LCD
    // definovanie 8-mich speciálnych znakov do CGRAM
    // pred definovaním spec. znakov by sme mali odpamätat obsah aktuálneho AC
    // teraz je na "vlavo hore"
    for (unsigned char i = 0; i < 8; i++) { // zapis specialnych znakov do CGRAM
        def_znak( Znak[i],i);// á, ý, ó, é, ž ?,?, ô
    }
    lcd_command(0x02); // AC ukazoval na "vlavo hore". Tak len zopakujeme.
    Vypis_na_display(); //
    while(1){
      //TODO:: Please write ...
    }
}
```

```c
char Riadok[20];
```

49

```c
void Vypis_na_display(void){
    // zapíšeme niekolko znakov do 1 riadku
    lcd_data(0x50); // P
    lcd_data('r');      // r
    lcd_data(':');      // :
    // namiesto prikazu goto riadok, pozicia
    // len nastavíme kurzor na 1. riadok 4. pozicia (5-ta)
    lcd_command(0x80 + 4); // 0b1000 0000 + 0b0000 0100
    // a pokracujeme vo výpise
    // "formátovaný výpis" pomocou printf
    // číslo CIS vypíšeme dekadicky a hexadecimálne
    char CIS = 78;  // hexadecimálne 0X4E a pridáme text áno
    // ak by sme zapísali adresu znaku á, teda \000 další text by sa neobjavil
    // nula je koniec retazca
    // sprintf je formatovany vypis do retazca
    sprintf(Riadok,"%dd=0x%X \010no ", CIS,CIS );
    zob_text(Riadok);
    // len nastavíme kurzor na 2. riadok 1. pozicia (resp. 0)
    lcd_command(0xC0 + 0);  // 0b1100 0000
    // a vypíšeme špeciálne znaky z CGRAM a pridame text end
    sprintf(Riadok,"\010\1 \2\3 \4\5 \6\7 e\156d" );
    zob_text(Riadok);
}
void zob_text(char *s){
    register unsigned char c;
    while((c = *s++))   lcd_data(c); // retazec konci "nulou"
}
```

50

Podprogramy tu uvedené zapíšu dáta do IR, resp. DR registra display-a.

```c
#include <avr/io.h>
#include "lcd_ch.h"

int main(void)
{
    lcd_init();                     // inicializacia portov a displeja
    lcd_data('A');                  // zobraz pismeno A na aktualnej pozicii
    lcd_data(104);                  // zobraz pismeno 'h' = 104dec = 0x68 ASCII

    lcd_command(0xc0);
    lcd_command(0x80 + 4);

              // 01234567890123456 +1 na koniec retazca
 char riadok[]= {"                "};
 int value = 42;                    // nejaka premenna

 sprintf(riadok,"Hodnota = %d ",value); // Zobrazíme ‘value’ desiatkovo %d
 lcd_puts(riadok);

  while(1); /* stop here */
}
```