Mikropočítačové Systémy

MIPS

Distribuované vnorené počítačové systémy

Distributed Embedded Computer System

(Microcontrollers)

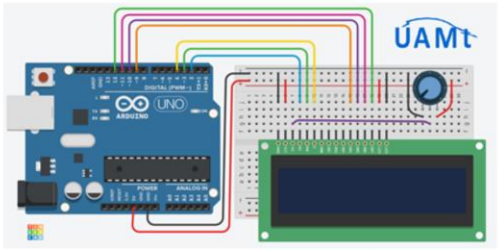
Prednáška 5.

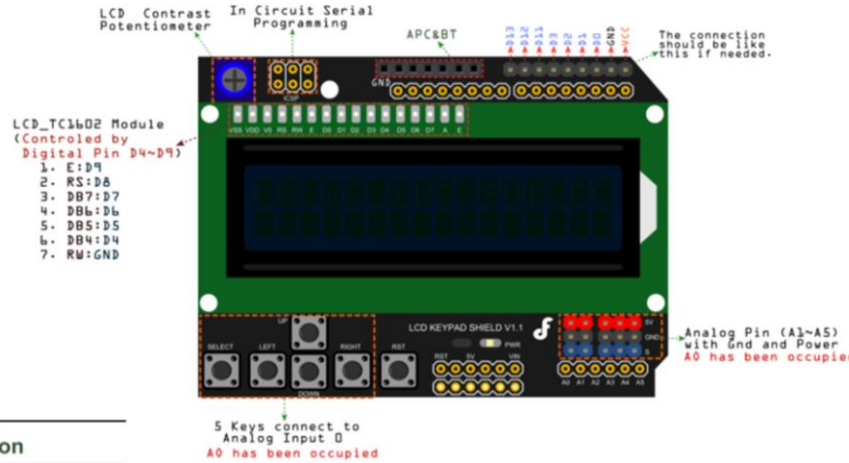
Modifikácia podkladov pre cv. Displej.

Prerušenia.

Aj napriek tomu, že program funguje môže mať nedostatky.
Napísať testovací program je zložitejšie ako napísať testovaný program.

Modifikácia podkladov cvičenia „LCD“.





Pin	Function	Instruction
Digital 4(D4)	D4~D7 are used as DB4~DB7	Four high order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the LCD.
Digital 5(D5)		
Digital 6(D6)		
Digital 7(D7)		
Digital 8(D8)	RS	Choose Data or Signal Display
Digital 9(D9)	Enable	Starts data read/write
Digital 10(D10)	LCD Backlight Control	
Analog 0(A0)	Button select	Select, up, right, down and left

RW = GND

2

Na cvičení som zaregistroval, že máme dva typy displajov.

Upravil som súbory: **lcd_ch.c** a **lcd_ch.h**, tak aby sa dali použiť pre oba displeje. Upravil som aj funkciu test **busy_flag().** Teraz možno pre displej (vľavo hore), kontrolovať aj stav pinu BF. Súčasne sa vyčíta aj stav AC.

Ak ho na začiatku modifikácie CGRAM displeja odpamätáme, môžeme ho po zápise špecialneho znaku obnoviť.

Toto v pôvodnom ani v modifikovanom programe nebolo. Po zápise špecialnych znakov som len nastavil AC na 1. riadok a 1. pozícia. Potom som

sa rozhodol, že doplním aj túto funkciu. T.j. ak sa bude počas výpisu modifikovať CGRAM, treba správne obnoviť obsah AC. Po určitej dobe som vymyslel testovací program, ktorý mal overiť funkčnosť. Všetko bolo takmer v **poriadku**. Len obsah neukazoval na novú hodnotu ale na starú.

Sú dve riešenia.

- Odpamätanú hodnou inkrementovať a je to.
- Správnejšie je nájsť chybu.
-

```c
int main(void)
{  /* inicializacia portov a displeja */
    lcd_init();
    // vytvorenie specialnych znakov, ulozenie do IRAM, potom  do CGRAM
    def_Clear_spec_znaky();
    def_spec_znaky();// SC a s + mekcen
    /* po ukonceni inicializacie je displej zapnuty a ak posleme nejake data, tak
       sa zacnu zobrazovat od prvej pozicie (riadok 0, stlpec 0)
       rovnako aj definovanie znakov vrati AC na prvu poziciu (riadok 0, stlpec 0)    */
    sprintf(riadok,"Aho"); // zapíšeme niekolko znakov do 1 riadku
    lcd_puts(riadok);
    def_spec_znaky_AC(); // tu zapisem c + mekcen a odpamatam AC ...
    lcd_data('j');      // znak j zadany ako znakova konstanta
    lcd_puts(" T = 35 \x04");
    lcd_command(0xc0);
    lcd_puts("Cv\7n. ");
    _delay_ms(3000);
    lcd_command(0xc0);
    sprintf(riadok,"Prev. %d = 0x%2X    ",value,value); // Zobrazíme ju (value) 2x: raz desiatkovo a potom hexa
    lcd_puts(riadok);
    _delay_ms(3000);
    lcd_command(0x0D);
    lcd_command(0xc0);
    lcd_puts("Prdn\10k.");

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

A	h	o

AC = 3

A	h	j		T		=		3	5	℃

AC = ? 2 ?

$\langle 0:7 \rangle \equiv \ldots$
$\langle 4 \rangle \equiv ℃$
$\langle 0 \rangle \equiv š$
$\langle 7 \rangle \equiv č$

```c
#ifndef _Shield_LCD
int8_t lcd_read_AC(void){ //  rd_BF
    char pom_AC ;
    while((pom_AC = busy_flag( )) & 0x80);
    return pom_AC;                  // display not busy
}
#endif
```

BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0

3

Keďže je CGRAM CMOS typu, je vhodné pre účely testovania doplniť program o jej vynulovanie **def_Clear_spec_znaky()**.

HD44780U

Table 6 Instructions (cont) HD44780.pdf (robotika.sk)

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
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*
	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 µs × 270/250 = 40 µs

Note: — indicates no effect.

* After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, t_{ADD} is the time elapsed after the busy flag turns off until the address counter is updated.

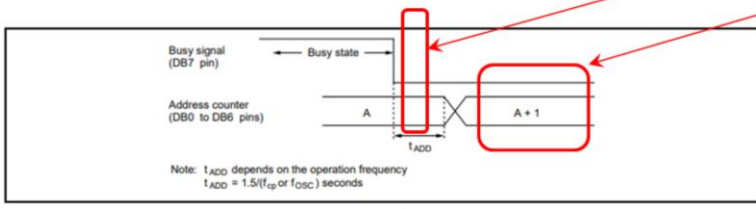


Figure 10 Address Counter Update

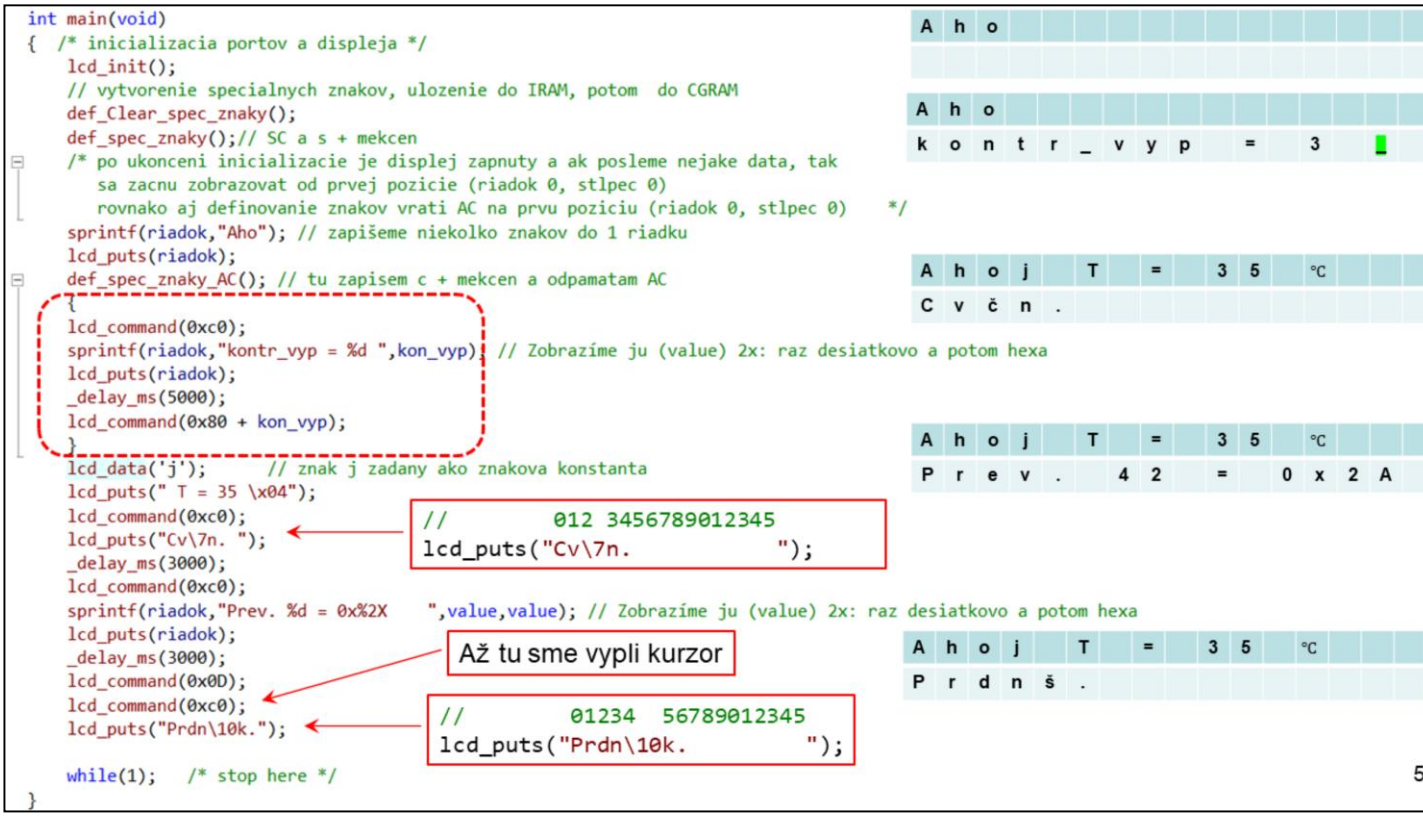
```
#ifndef _Shield_LCD
int8_t lcd_read_AC(void){ //  rd_BF
    char pom_AC ;
    while((pom_AC = busy_flag( )) & 0x80);
    return pom_AC;                    // display not busy
}
#endif
```

BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0

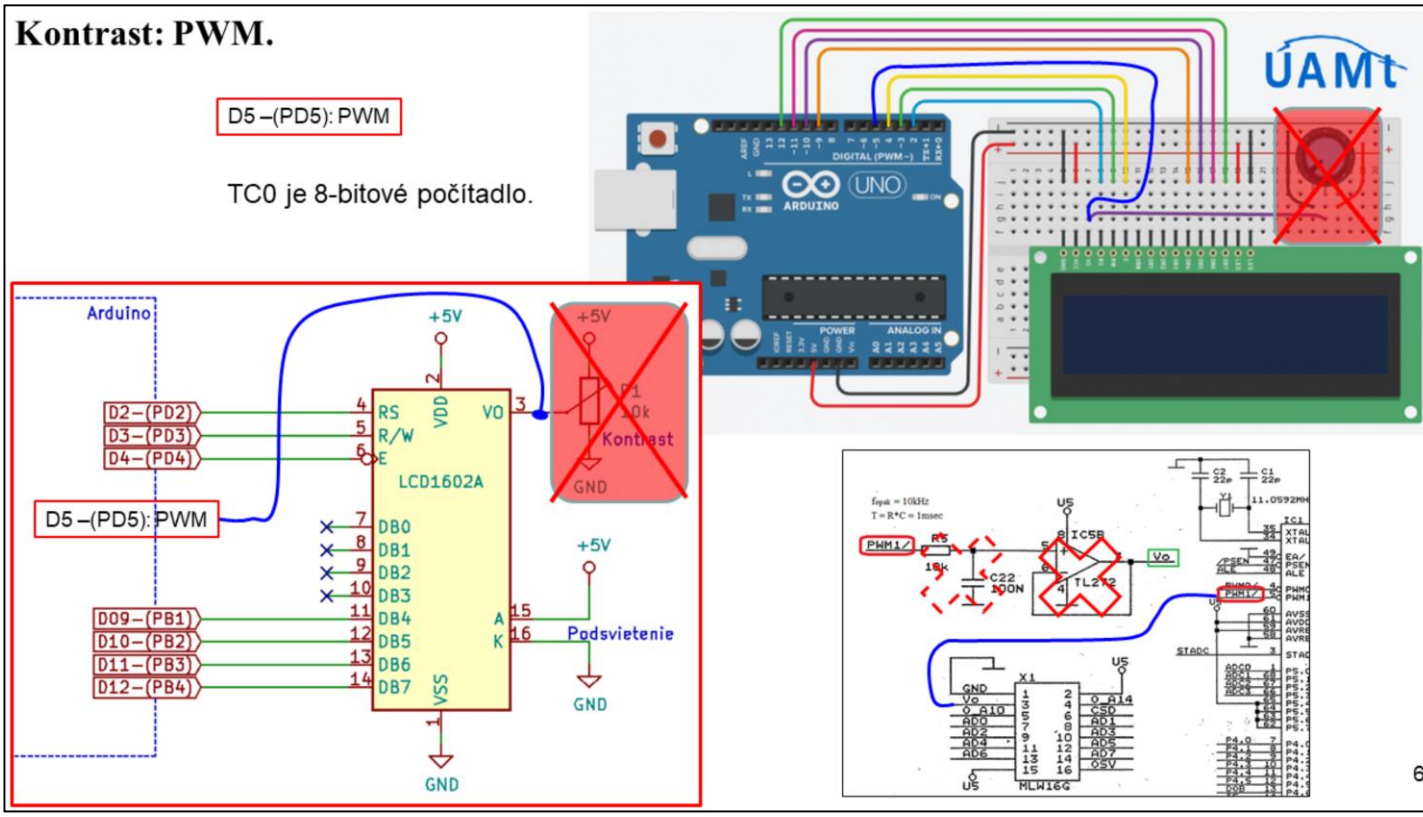
```
#ifndef _Shield_LCD
int8_t lcd_read_AC(void){ //  rd_BF
    char pom_AC ;

    while((pom_AC = busy_flag( )) & 0x80);
    // kedze po BF = 0 este cca 4us sa nezmenil obsah AC
    // treba vycitat este raz
    pom_AC = busy_flag( );
    return pom_AC;                    // display not busy

}
#endif
```

4

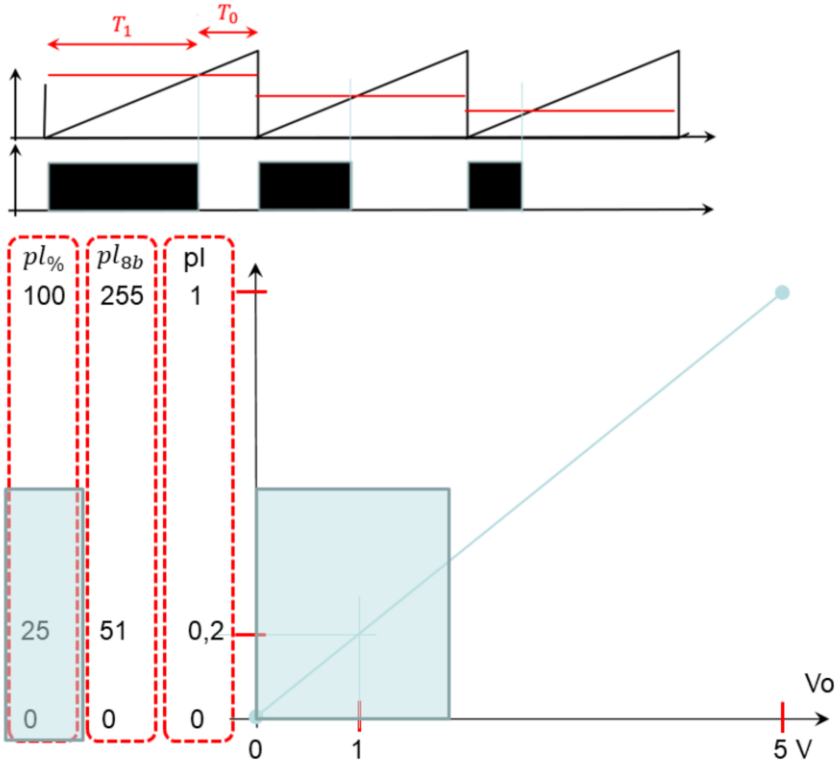
Z KL je zrejme, že veľmi rýchle vyčítavame. Stačí upraviť program a všetko je OK. Ešte si pripomeňme, že CHÝBA **TIMEOUT**.



Nesmieme zabudnúť na dve veci. Nulovanie predchádzajúcich výpisov a na skutočnosť, že keď niečo na displeji nevidíme, že to vskutočnosti nie je. Mám na mysli KURZOR.



D5- (PD5) možno zapojiť ako PWM výstup TC0. Meraním na Vo som zistil, že Vo = 0,3V je dostatočný kontrast. Za Vo > 1,0V sa už znaky nedajú rozoznať. To platí pre DC. Pre PWM budeme musieť overiť.



$Vo \in (0V, 5{,}0V)$

$Vo_{optimálne} = 0{,}3V$

$Vo_{min} = 0{,}0V$

$Vo_{max} = 1{,}0V$

$$pl = \frac{T_1}{T_1 + T_0}, < (0, 1>)$$

$$pl = \frac{Vo}{5V}$$

$$pl_{8b} = pl * 255$$

7

Displej „svieti“, ak je kontrast správne nastavený. T.j., ak je Vo blízke nule. Ak chceme pozitívnu logiku, potom by malo platiť.

Čím menšie plnenie, tým väčší kontrast.

Vytvorenie V I R U Su

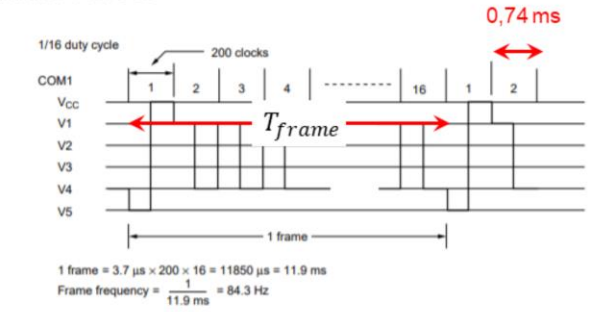
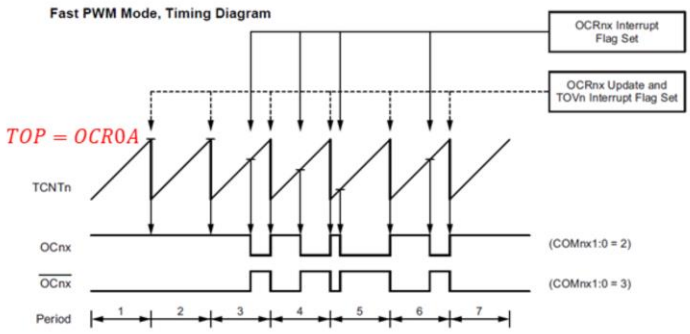
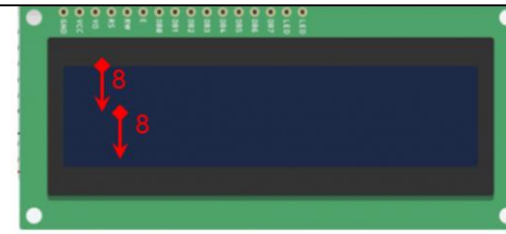


Figure 24 Frame Frequency





Meraním sme zistili:

$T_{frame} = 12{,}16\ ms \Rightarrow f_{frame} = 82{,}23Hz.$

$T_{LCD\ OSC} = 3{,}914\ \mu s \Rightarrow f_{LCD\ osc} = 255{,}5kHz.$

Z uvedeného je zrejmé, že ak má displej a ľudské oko pracovať ako filter, musí byť $f_{PWM} \gg f_{frame}$

t.j. aspoň $f_{PWM} = 10 * f_{frame} \cong 1kHz$

??? Správna úvaha ????

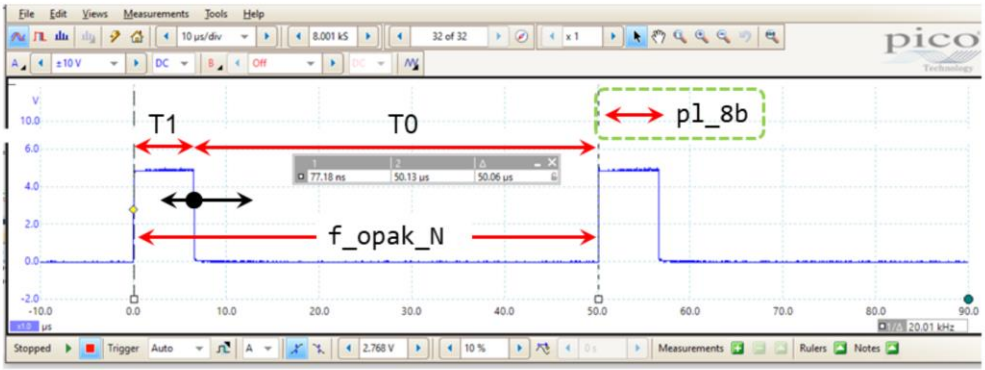
Z dôvodov „iných“ nastavíme PWM naTC0 takto: Mod = 7 (Fast PWM), kde je TOP hodnota daná registrom OCR0A. Zvolíme „non-inverting mode compare ...“

Mód v ktorom je výstup nulovaný pri zhode, t.j. TCNT0 = OCR0B.

$$f_{PWM} = \frac{f_{clk}}{N * (\langle OCR0A \rangle)} = \frac{16000000\ Hz}{8 * 100}$$

8

Frekvncia oscilátora je v pásme +- 30 percent. N = 8, je vlastne delička 8-mi.



$$f_{PWM} = \frac{f_{clk}}{N * (\langle OCR0A \rangle)} = \frac{16000000\ Hz}{8 * 100} = 20\text{kHz}$$

TOP = OCR0A =100
N (Delička) = 8

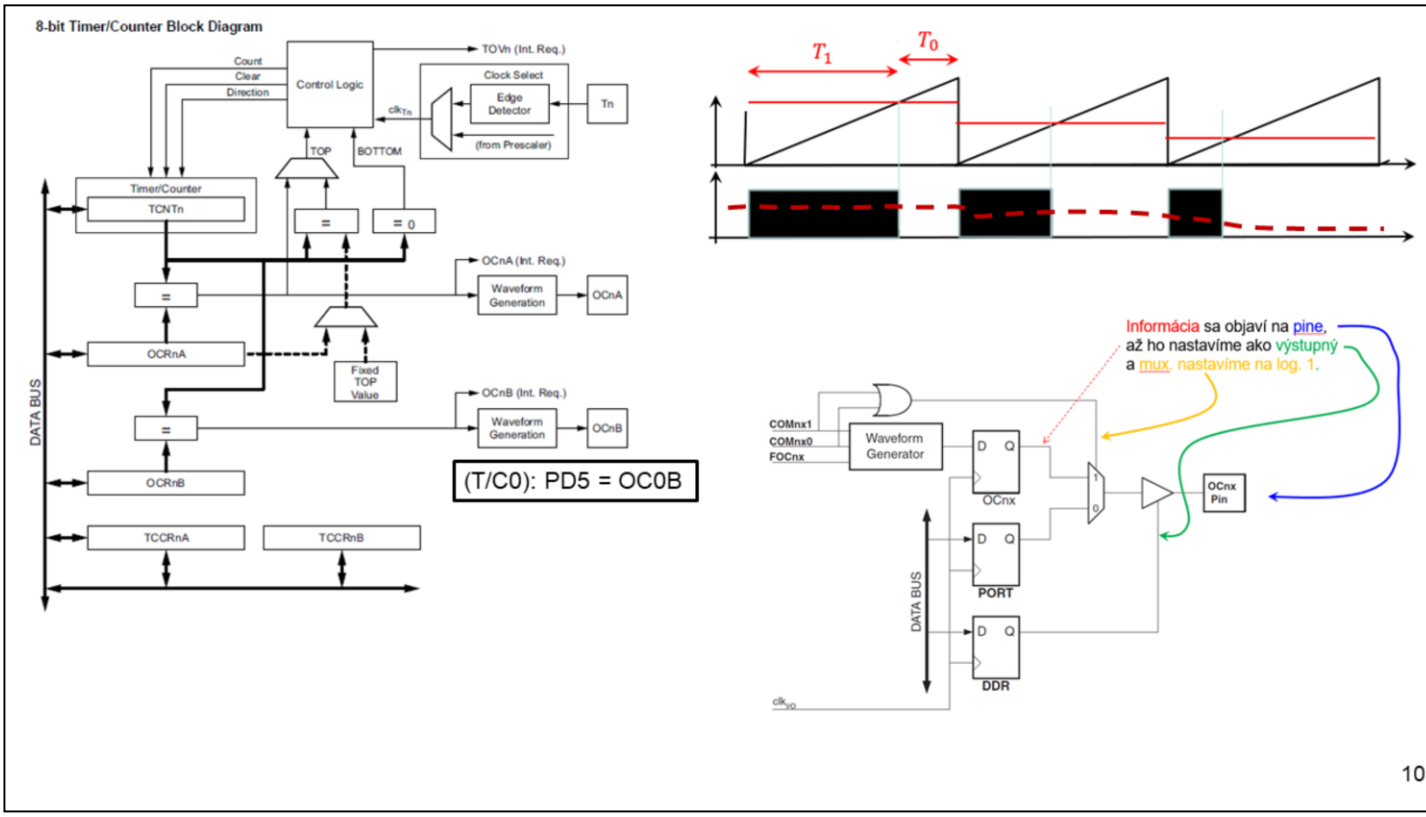
```
// uvodne nastavenie
OCR0B = pl_8b;
OCR0A = f_opak_N;

    while(1){;
          lcd_command(0xC0 + 0);// 0b1100 0000
          sprintf(Riadok1,"pl_8b =%3d", pl_8b);
          zob_text(Riadok1);
```

N_ones*100msTOP = 500ms

```
          while(N_ones--) _delay_ms(100);

          N_ones = 5;
          pl_8b++;
          if (pl_8b > 50)pl_8b = 5;
          OCR0B = pl_8b;
}
```

9

V skutočnosti dosadzujeme do OCR0A nie sto ale 99. „Je tam plus prechod na nula.“ pl_8b a f_opak_N sú pomerné čísla vystupujúce vo vzťahoch. Interval zmien, viď. program.



Nezabudni nastaviť PD5 ako výstupný.

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

```
void ini_TC0(void){
set_bit(DDRD,PIND5); //OC0B
//  Nastavenie TC0
// 7 6         5 4         3      2     1 0
// COM0A[1:0]  COM0B[1:0]               WGM0[1:0]
TCCR0A = 0b00100011;// OC0B  PWM mod = 7

// 7 6 5 4  3      2   1   0
//          WGM02  CS0[2:0]
TCCR0B = 0b00001010;// fosc/8

OCR0B = pl_8b;
OCR0A = f_opak_N;
}
```

Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on[1)2)]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

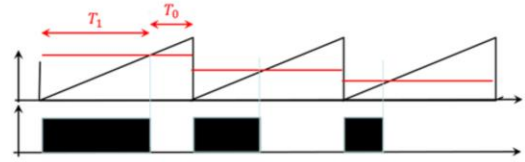
Notes: 1. MAX = 0xFF
2. BOTTOM = 0x00

Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Compare Output Mode, Fast PWM Mode[1)]

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Reserved
1	0	Clear OC0B on compare match, set OC0B at BOTTOM, (non-inverting mode)
1	1	Set OC0B on compare match, clear OC0B at BOTTOM, (inverting mode).

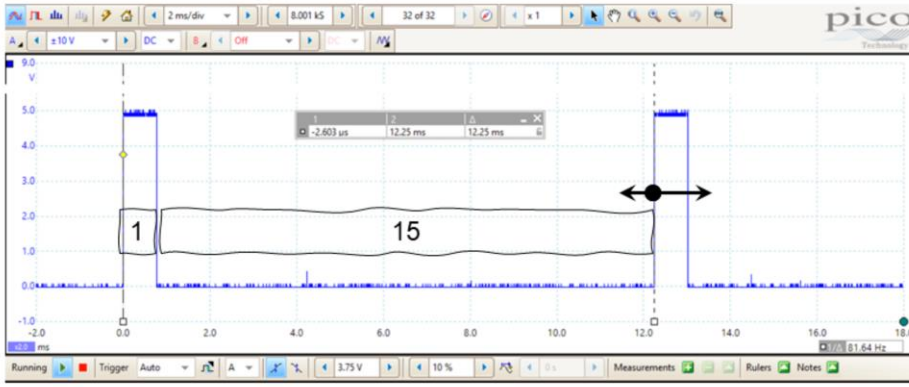


11

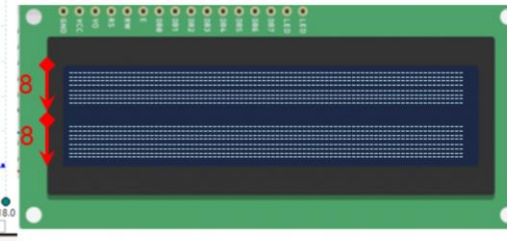
Tomuto nastaveniu: fPWM =20kHz. Stačí len pridať inkrementovanie /dekrementovanie premennej `pl_8b` v intervale 5 až 50 pomocou tlačítok a bude vedieť nastavovať kontrast.

VIRUS:

Snažíme sa nastaviť $f_{PWM} = f_{frame}$



Teraz zachováme plnenie na Hodnote 1/16 a budeme meniť `f_opak_N` tak, aby sme dostali niečo okolo $f_{PWM} = 81{,}64Hz$.



Teoretická hodnota:

$$f_{PWM} = \frac{f_{clk}}{N * (\langle OCR0A \rangle)} = \frac{16000000\ Hz}{1024 * 191} = 81{,}81Hz$$

Meraním sme zistili:

$T_{frame} = 12{,}16\ ms \Rightarrow f_{frame} = 82{,}23Hz.$

$T_{PWM} = 12{,}25\ ms \Rightarrow f_{PWM} = 81{,}64Hz.$

12

Zmeníme nastavenie TC0.

Plnenie je 1/16. T.j. jeden riadok nesvieti (5V) a 15 riadkov svieti (0V).

Keďže $f_{PWM} = 0{,}9928 * f_{frame}$ nesvietiaci riadok sa pomaly pohybuje po displeji. Ak sa dotkneme radiča displeja, zohrejeme ho. Jeho frekvencia sa zmení. Podľa toho ako sa zmení pomer f_{PWM}/f_{frame} sa bude nesvietiaci riadok pohybovať nahor/nadol buď rýchlejšie alebo pomalšie.

`Vyhovuje f_opak_N = 190.`

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

```
// uvodne nastavenie
OCR0B = pl_8b; //  f_opak_N/16, resp. deleno 8
OCR0A = f_opak_N;

    while(1){;
     lcd_command(0xC0 + 0);// 0b1100 0000
     sprintf(Riadok1,"f_opak_N =%3d", f_opak_N);
     zob_text(Riadok1);
     while(N_ones--) _delay_ms(100);
     N_ones = 20;
     f_opak_N++;
     if (f_opak_N > 200)f_opak_N = 170;
     OCR0B = f_opak_N/16;
     OCR0A = f_opak_N;
}
```

Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on[1)(2)]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes: 1. MAX = 0xFF
2. BOTTOM = 0x00

Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Compare Output Mode, Fast PWM Mode[1)]

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Reserved
1	0	Clear OC0B on compare match, set OC0B at BOTTOM, (non-inverting mode)
1	1	Set OC0B on compare match, clear OC0B at BOTTOM, (inverting mode).

13

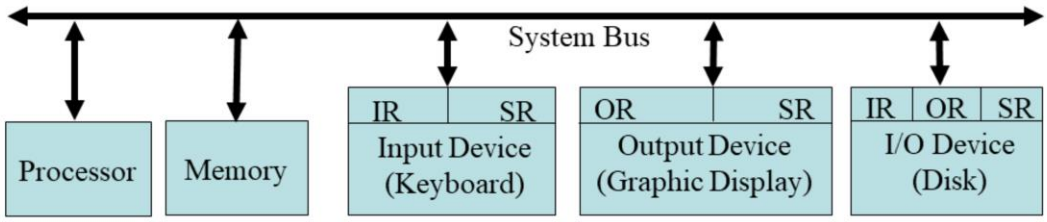
Každé 2 sekundy mením f_opak_N aby sme videli ako sa mení pohyb nerozsvecujúceho sa riadku.

Nesmieme zabudnúť, že aj teraz je pin PWM nastavený ako OUTPUT.

To čo vidíme na displeji je vlastne stroboskopický efekt. Teoreticky by som mal vedieť zo zadaných údajov, rýchlosti pohybujúceho nesvietiaceho sa riadku a jednej frekvencie, napr. f_PWM vypočítať tú druhu, teda f_frame. Je to pravda?

Interrupts - Prerušenia

Na počiatku bolo všetko jednoduché. Prvé počítače nemali prerušovací podsystém. Postupnosť vykonávania inštrukcií sa riadila pomocou obsahu adresného čítača. Neskôr sa zistilo, že niektoré periférne obvody sú vzhľadom na CPU a pamäť pomalé. Procesor sa musel o ich pozornosť usilovať. Neustále kontroloval stavový register takéhoto pomalého zariadenia, aby zistil, či je alebo nie je pripravené na spoluprácu s procesorom. Týmto sa rýchlosť procesora znížila. Takáto metóda zisťovania stavu periférie sa nazýva: *Polled Method* - opytovacia, testovacia, … .



Tri základné spôsoby komunikácie s I/O podsystémom:
Polled Method – opytovacia metóda
Interrupt Method – prerušenia
DMA – priamy prenos medzi pamäťou a perifériami

14

Prenosová rýchlosť I/O zariadení:

•Klávesnica: „cca“ 10 znakov (B)/sek. Charakteristika prenosu: Komunikácia procesor – klávesnica. Väčšinou sa náhodné prenášajú byty, resp. niekoľko bytov. CPU musí neustále *testovať „zatlačenie*“ klávesnice. Kód zatlačeného znaku sa uloží do *Input register* a radič klávesnice nastaví príznak pripravenosti znaku. Procesor znak prečíta a zruší príznak => znak sa prečíta len raz. Procesor trávi veľa času testovaním: => *Interrupt*.

Opačný proces možno badať smerom k displayu, resp. k tlačiarni.

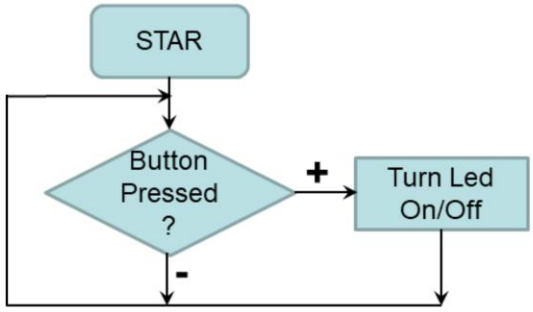
•Laserová tlačiareň: cca 100 000 znakov/sek,

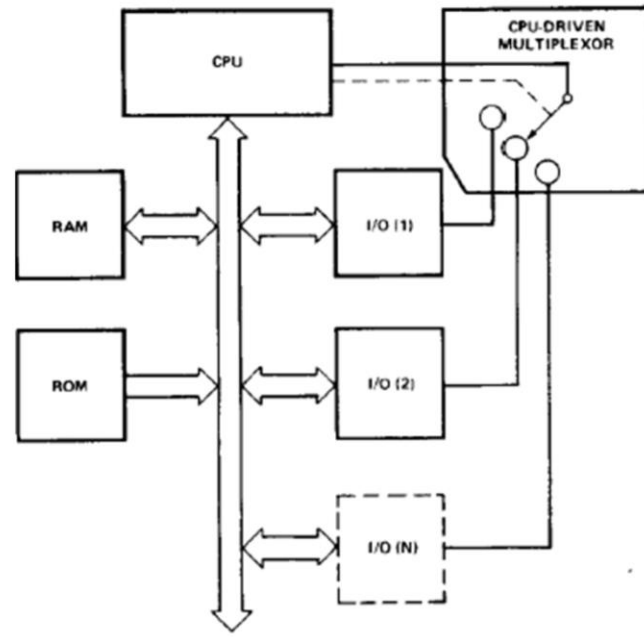
•Grafický display: cca 30 000 000 znakov/sek,

•Pevný disk: cca milióny B/sek. Charakteristika prenosu: Prenos veľkého množstva údajov organizuje procesor po bytoch, resp. slovách. => Strata času. Výhodnejšie je prenos organizovať po blokoch (zabezpečuje hardware) : => Rýchlejší prenos. Procesor „oddychuje“, resp. robí inú prácu. => *DMA*.

Obsluha periférií:
Polled Method – opytovacia metóda

Spôsob komunikácie – protokol je riadený CPU pomocou programu.
Nech má I/O(2) „pripravený“ znak pre CPU.
CPU adresuje I/O(2) a prečíta znak do ACU. Atď.
Ak nie je rýchlosť CPU a I/O(2) rovnaká, CPU preberie znak niekoľko krát.
Riešenie je v použití: *Status registra (bitu)*.



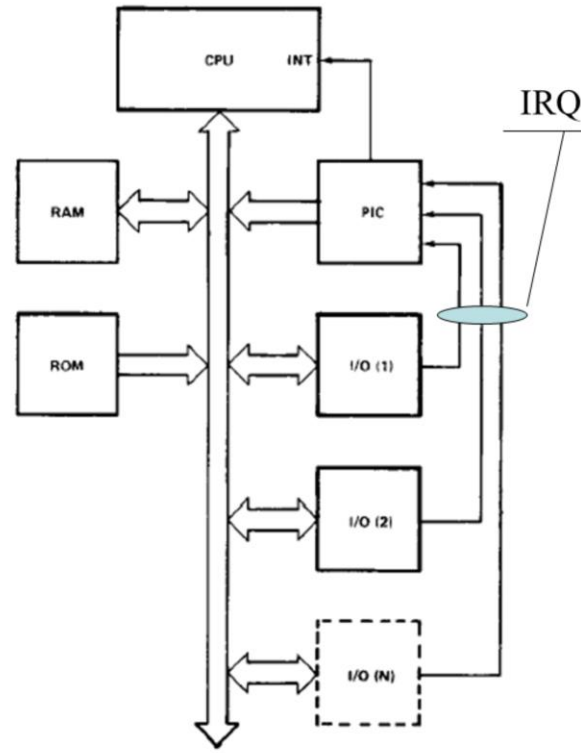


15

Tvorcovia hardwaru prišli na to, že testovanie príznaku sa ľahko realizuje obvodovo. Do procesora dorobili vstup, asynchrónne pracujúci s názvom INTerrupt, ktorý procesor „testuje na pozadí“ – počas SC. Cez tento vstup sa hardwarovo procesoru oznamuje, že mimo procesora sa niečo deje, o čom by mal vedieť. Mal by zareagovať. Takýto spôsob obsluhy periférneho obvodu, takéto prerušenie vykonávania programu je nazvané hardwarové prerušenie a počítačová prax to jednoducho nazvala prerušenie – INTERRUPT. Iné ako hardwarové nebolo. V tejto etape vývoja počítačov je prerušenie každému jasné: Požiadavka a vyhodnotenie prerušenia vzniká na hardwarovej úrovni (na základe žiadosti o prerušenie IRQ). CPU vie vykonávať len a len program, z tohto dôvodu obsluha prerušenia je len vykonanie programu, ktorý sa len nepatrne líši od bežného podprogramu.

Obsluha periférií:
Interrupt– prerušenie

Niekedy je potrebné prerušiť beh programu, napr.: pokles napájania. Prerušením behu programu môžeme tiež informovať CPU o ukončení udalosti: napr. odvysielanie znaku cez USART. Moderné CPU majú inštrukcie, ktoré sa správajú podobne ako hardwarové prerušenie. T.j. Ak je prerušená činnosť CPU, riadenie sa dočasne odovzdá inému „programu“. Po skončení sa vráti k pôvodnej činnosti. Viacnásobné prerušenia: Priorita.



16

Túto idilku pokazil Intel, keď do svojich procesorov zaradil inštrukciu int (software interrupt). Aby sme v tom mali jasno, na úvod treba povedať, že prerušenia delíme do skupín:

Vonkajšie – hardwarové. Prerušenie vyvolané technickými prostriedkami. O prerušenie požiada periféria, keď potrebuje reakciu procesora. Niekedy sa tento typ prerušenia nazýva aj **asynchrónné.** Priamo nesúvisí s vykonávaným programom a môže nastať kedykoľvek. Procesor má zvyčajne dva prerušovacie vstupy pre externé prerušenia (CPU má rád poriadok, preto treba deje zosynchronizovať):

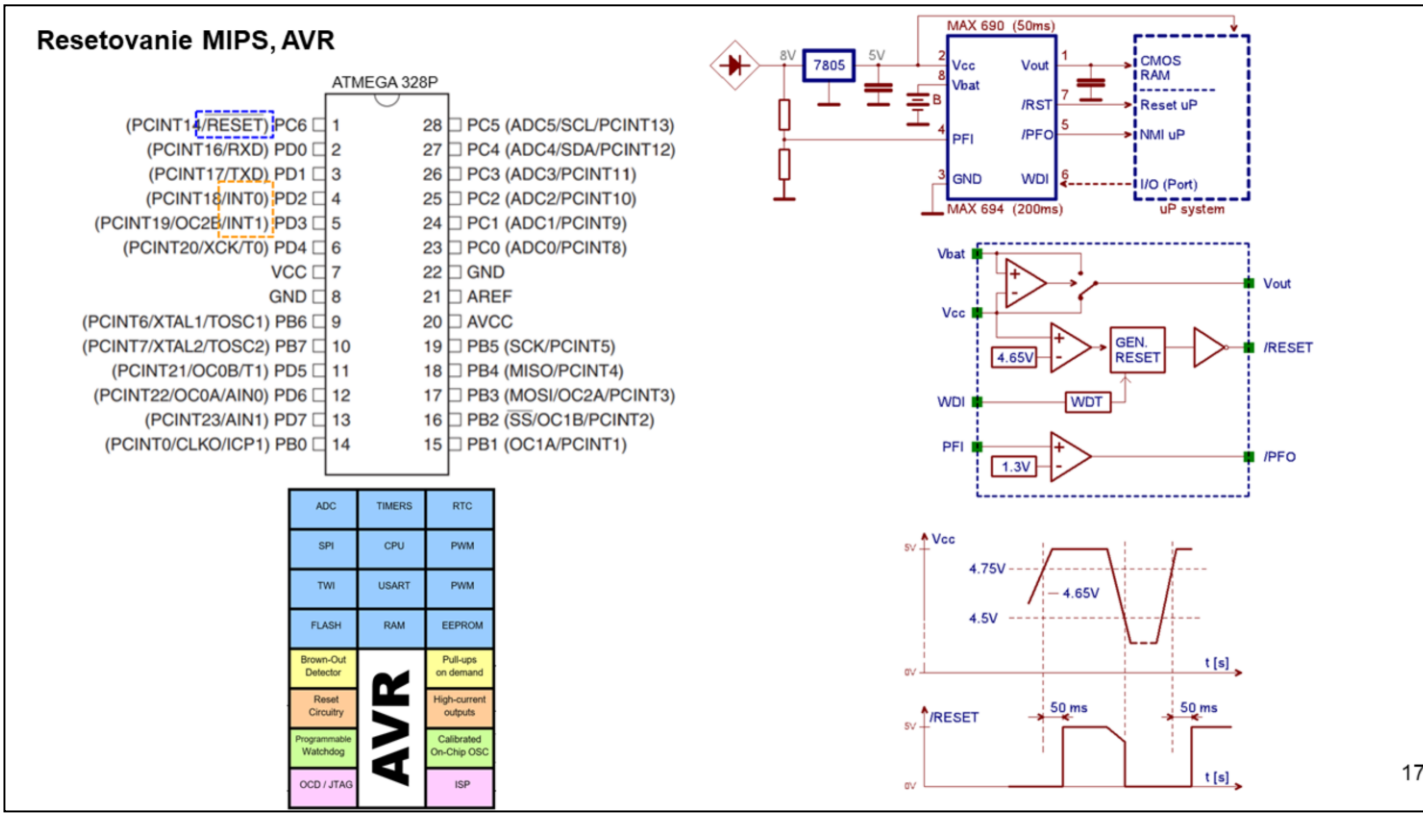
- **Vstup nemaskovateľného prerušenia** (signál procesora: Non Mascable Interrupt - NMI). NMI - sa používa hlavne na signalizovanie

“katastrofických “ udalostí ako je pokles napájacieho napätia alebo chyba parity operačnej pamäte alebo zbernice. Tento vývod procesora je teda určený pre prerušenia, ktoré nie je možné zakázať.

- **Vstup maskovateľného prerušenia** (signál procesora: Mascable Interrupt - INTR). Obsluhu tohto prerušenia možno programovo zakázať, inštrukciou CLI, vynulovaním bitu IF (Interrupt Flag) v príznakovom registri procesora. Pokiaľ je IF = 0 procesor prerušenie ignoruje. Tento príznak sa vzťahuje len na INTR. Nie na vnútorné prerušenia a NMI.

Vnútorné – softwarové. Priamo súvisí s vykonávaným programom a nemôžeme ho zakázať.

Niekedy sa tento typ prerušenia nazýva aj **synchrónnym.** Procesory AVR nemajú softvarové prerušenia. Môžu len niektoré prerušenia softwarovo vyvolať.



Resetovanie:

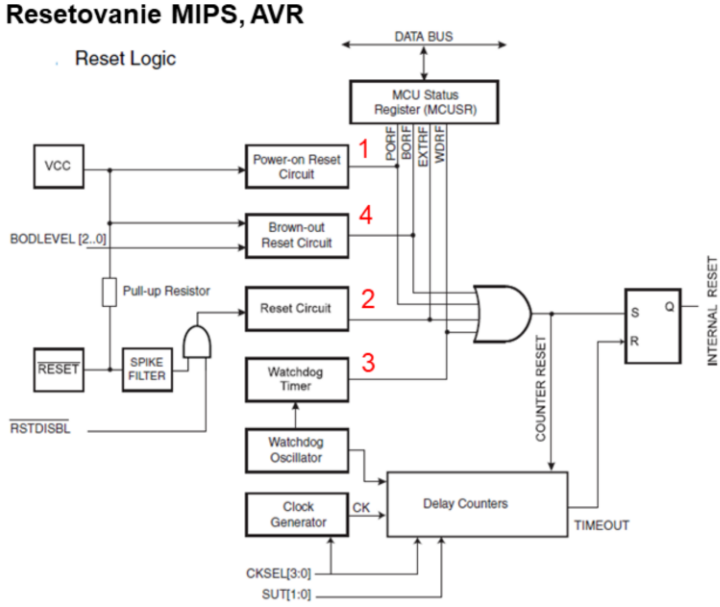
Počas Reset-u, všetky I/O Register-e sa nastavia na inicializačné hodnoty (okamžite, ako sa aktivuje signál Reset. Netreba k tomu žiaden hodinový signál) a program sa začne vykonávať od **Reset** vektora. Inštrukcia vektora **Reset** musí byť **JMP** – absolútny skok na inštrukcie obsluhy rutuny Reset.

Ak užívateľský program nepoužije prerušenia, môže byť uložený do pamäte za adresu vektora **Reset**. Atď.

Potom čo sa objaví **Reset**, aktivuje sa oneskorovací čítač. Počas tohto oneskorenia môže napájanie dosiahnuť stabilnú hodnotu, potrebnú k normálnej činnosti. Oneskorenie sa dá nastaviť pomocou **Fuses** - poistiek.

ATmega328 má 4 zdroje Reset-u:

Resetovanie MIPS, AVR



MCUSR – MCU Status Register

Bit	7	6	5	4	3	2	1	0
0x34 (0x54)	–	–	–	–	WDRF	BORF	EXTRF	PORF
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	See Bit Description			

Zdroje Reset-u

1. **Power-on Reset**. MCU je resetované, ak napájanie je pod prahovou hodnotou (**VPOT**).
2. **External Reset**. MCU je resetované, ak sa na pine **RESET** objaví nízka úroveň signálu na čas dlhší ako je minimálny.
3. **Watchdog Reset**. MCU ie resetované, ak Watchdog Timer napočíta nastavenú periódu a je jeho činnosť je povolená.
4. **Brown-out Reset**. MCU ie resetované, a napájacie napätie VCC poklesne pod hodnotu (**VBOT**) a Brown-out Detector je povolený.
5. JTAG AVR Reset.

18

Počas Reset-u, sa všetky I/O Register-e nastavia na inicializačné hodnoty (okamžite, ako sa aktivuje signál Reset. Netreba k tomu žiaden hodinový signál) a program sa začne vykonávať od **Reset** vektora. Inštrukcia vektora **Reset** musí byť **JMP** – absolútny skok na inštrukcie obsluhy rutuny Reset.

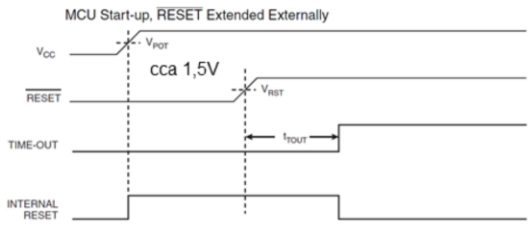
Ak užívateľský program nepoužije prerušenia, môže byť uložený do pamäte za adresu vektora **Reset**. Atď.

Potom čo sa objaví **Reset**, aktivuje sa oneskorovací čítač. Počas tohto oneskorenia môže napájanie dosiahnuť stabilnú hodnotu, potrebnú k normálnej činnosti. Oneskorenie sa dá nastaviť pomocou **Fuses** - poistiek.

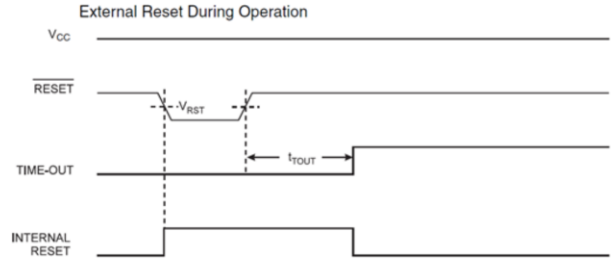
Príznaky RESET sa nulujú zápisom nuly alebo Power-on Reset-om.

Zdroje Reset-u

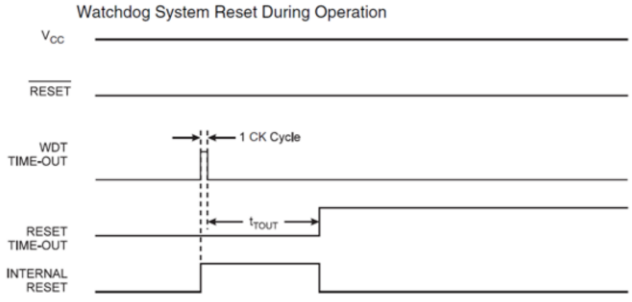
Power-on Reset. MCU je resetované, ak napájanie je pod prahovou hodnotou (**VPOT**).



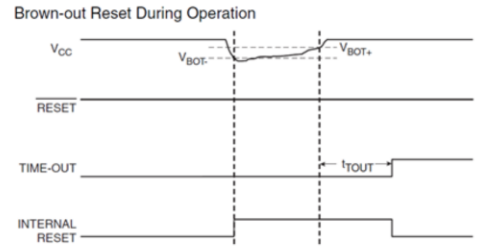
External Reset. MCU je resetované, ak sa na pine **RESET** objaví nízka úroveň signálu na čas dlhší ako je minimálny.



Watchdog Reset. MCU ie resetované, ak Watchdog Timer napočíta nastavenú periódu a je jeho činnosť je povolená.



Brown-out Reset. MCU ie resetované, a napájacie napätie VCC poklesne pod hodnotu (**VBOT**) a Brown-out Detector je povolený.



19

Power-on Reset. MCU je resetované, ak napájanie je pod prahovou hodnotou (**VPOT**).

External Reset musí mať minimálnu dĺžku trvania. Kratšie môžu byť odfiltrované.

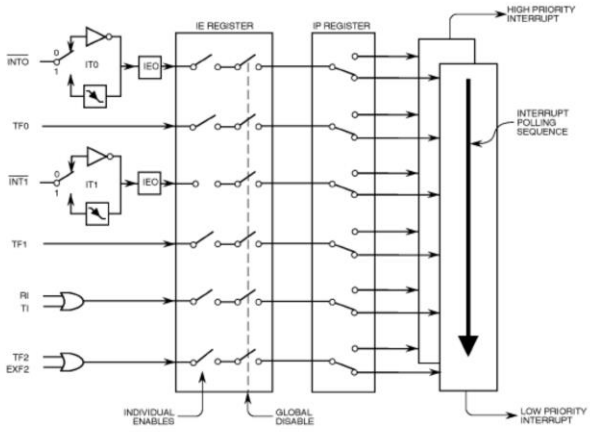
Watchdog Reset. MCU ie resetované, ak Watchdog Timer napočíta nastavenú periódu a je jeho činnosť je povolená.

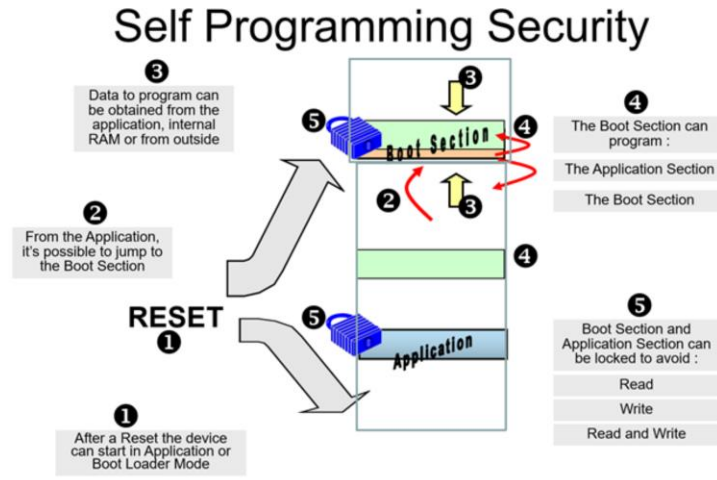
Úrovne registrované Brown-out Detector-om sa dajú zvoliť pomocou BODLEVEL Fuses. Prahová hodnota má hysteréziu:

VBOT+ = VBOT + VHYST/2 and VBOT- = VBOT - VHYST/2. hysterézia je cca 50mV. A typické hodnoty nastavitelné pomocou Fuses sú: 1,8V, 2,7V a 4.3V.

Interrupt - Prerušenia.

Takto bol zapojený přerušovací podsystém procesora 8051





20

Procesory AVR majú niekoľko prerušení. Každému prerušeniu odpovedá samostatný prerušovací vektor. Každému prerušovaciemu vektoru odpovedá program obsluhy prerušenia. Najnižšie pamäťové miesta v pamäti programu sú definované ako: *Reset* a *Interrupt vektory*. Každé prerušenie má priradený bit pre povolenie prerušenia. Prerušovací podsystém ma pridelený bit (**Global Interrupt Enable** – **GIE** v **Status Register**) pre povolenie, resp. zakázanie prerušení ako celku.

Každý zdroj prerušenia má pridelený samostatný bit, do ktorého musí byť zapísaná jednotka, ak chceme toto prerušenie povoliť.

Všetky zdroje prerušenia môžu byť „automaticky“ zakázané. Závisí to od celkového nastavenia AVR. Najnižšia adresa v pamäti programu odpovedá

RESETu a potom nasledujú postupne podľa priority (reset má najvyššiu prioritu. Niektoré zdroje ho zaraďujú do kategórie NMI. Nižšia adresa vyššia priorita.) prerušovacie vektory ostatných zdrojov prerušení. Z prerušení má najvyššiu prioritu:
INT0 – External Interrupt Request 0.
Prerušovacie vektory môžu byť presmerované do BOOTFLASH pamäte. Podobne to platí aj pre **RESET**.

Interrupt - Prerušenia.

Reset and Interrupt Vectors in ATmega328 and ATmega328P

VectorNo.	Program Address(2)	Source	Interrupt Definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

Reset and Interrupt Vectors Placement in ATmega328 and ATmega328P

BOOTRST	IVSEL	Reset Address	Interrupt Vectors Start Address
1	0	0x000	0x002
1	1	0x000	Boot Reset Address + 0x0002
0	0	Boot Reset Address	0x002
0	1	Boot Reset Address	Boot Reset Address + 0x0002

21

Prerušovací vektor 1 má vyššiu prioritu ako prerušovací vektor 2, atď.
Prerušovacie vektory môžu byť posunuté do „Boot Flash section“ nastavením bitu **IVSEL** v registri

MCU Control Register (MCUCR).

Pri obsluhe prerušenia a pri volaní podprogramu sa návratová adresa, obsah registra -*Program Counter* (**PC**) uloží do zásobníka – *Stack*. *Stack* je umiestnený v dátovej pamäti **SRAM**. Veľkosť *Stacku* je limitovaná veľkosť **SRAM** (mínus čosik). Každý užívateľský program musí inicializovať **SP** počas podprogramu obsluhy **Reset**u. *Stack Pointer* - **SP** je umiestnený v **I/O** priestore a možno ho čítať aj doňho zapisovať.

SREG – AVR Status Register

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – I: Global Interrupt Enable
Ak bit **GIE** vynulujeme sú zakázané všetky prerušenia. Ak je bit **GIE** nastavený môžeme individuálne povoliť, resp. zakázať jednotlivé zdroje prerušenia. **I** bit sa nuluje hardwarovo po vyvolaní prerušenia a je nastavený inštrukciou **RETI**, aby sa umožnila obsluha ďalších čakajúcich prerušení. Bit **I** môžeme nastavovať a mazať programovo pomocou inštrukcií **SEI** a **CLI**.

EICRA – External Interrupt Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 1,0 – INT1,0: External Interrupt Request 1,0 Enable
Lokálne povolenie/zakázanie externého prerušenia.

MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	BODS(1)	BODSE(1)	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 1 – IVSEL: Interrupt Vector Select
Ak je bit **IVSEL** nastavený na log. 0, prerušovacie vektory sú umiestnené na začiatok Flash pamäte. Ak je bit **IVSEL** nastavený na log. 1, prerušovacie vektory sú umiestnené na začiatok Boot Loader časti Flash pamäte.
Bit 0 – IVCE: Interrupt Vector Change Enable
Ak chceme zmeniť **IVSEL** bit, bit **IVCE** musí byť nastavený na log. 1. Bit **IVCE** sa nuluje Hardwarovo štyri SC po nastavení tohto bitu. Pri nastavení sa zakážu všetky prerušenia.

MCUSR – MCU Status Register

Bit	7	6	5	4	3	2	1	0	
0x34 (0x54)	–	–	–	–	WDRF	BORF	EXTRF	PORF	MCUSR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	See Bit Description				

Bit 3 – WDRF: Watchdog System Reset Flag
Tento bit sa nastaví ak je reset generovaný Watchdog-om. Vynuluje sa pri Power-on Reset alebo zápisom log. 0 do tohto bitu.
Bit 2 – BORF: Brown-out Reset Flag
Tento bit sa nastaví ak je reset generovaný Brown-out. Vynuluje sa pri Power-on Reset alebo zápisom log. 0 do tohto bitu.
Bit 1 – EXTRF: External Reset Flag
Tento bit sa nastaví ak je reset generovaný External Reset. Vynuluje sa pri Power-on Reset alebo zápisom log. 0 do tohto bitu.
Bit 0 – PORF: Power-on Reset Flag
Tento bit sa nastaví ak je reset generovaný Power-on. Tento bit sa vynuluje len zápisom log. 0 do tohto bitu.

22

I: Global Interrupt Enable

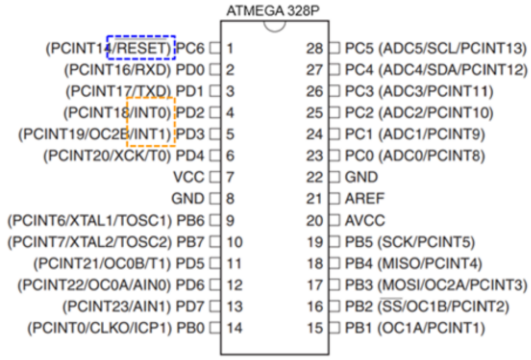
Ak bit **GIE** vynulujeme sú zakázané všetky prerušenia. Ak je bit **GIE** nastavený môžeme individuálne povoliť, resp. zakázať jednotlivé zdroje prerušenia. **I** bit sa nuluje hardwarovo po vyvolaní prerušenia a je nastavený inštrukciou **RETI**, aby sa umožnila obsluha ďalších čakajúcich prerušení. Bit **I** môžeme nastavovať a mazať programovo pomocou inštrukcií **SEI** a **CLI**.

Externé prerušenia sú vyvolané cez piny **INT0** a **INT1**. *Poznamenajme*, že ak sú tieto prerušenia povolené, vyvolajú sa aj keď odpovedajúce piny budú konfigurované ako výstupné. Táto vlastnosť sa dá využiť pri generovaní *softwarového prerušenia*.

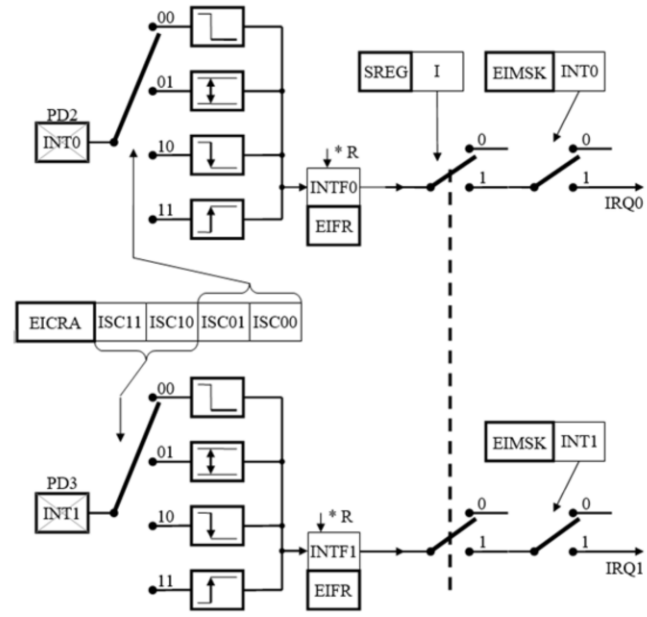
Pri vyvolaní obsluhy prerušenia sa bit *Global Interrupt Enable*, označený ako **I**-bit, vynuluje a všetky ďalšie prerušenia sa zakážu. Používateľ môže vo svojom programe tento bit nastaviť a tým povolí vnorené prerušenia. Všetky povolené prerušenia môžu potom prerušiť práve vykonávané prerušenie.

V podstate máme dva typy prerušenia:

1. je vyvolaný úrovňou signálu.
2. je vyvolaný zmenou signálu.



ISCx1	ISCx0	Description
0	0	The low lewel of INTx generates an interrupt request.
0	1	Any logical change on INTx generates an interrupt request.
1	0	The faling edge of INTx generates an interrupt request.
1	1	The risingedge of INTx generates an interrupt request.



INT0 pin **PD2**
INT1 pin **PD3**

23

V podstate máme dva typy prerušenia.

1. typ je spúšťaný – zapínaný nejakou udalosťou, ktorá nastaví príznak prerušenia - *Interrupt Flag*.
Tieto prerušenia sú obsluhované vykonananím odpovedajúceho podprogramu obsluhy prerušenia. V okamžiku vstupu do obsluhy prerušenia sa vynuluje požiadavka o obsluhu prerušenia.
Požiadavku o prerušenie - *Interrupt Flag*, môžeme zrušiť zapísaním log. 1 do tohto bitu. Ak vznikne požiadavka o prerušenie v čase, keď je odpovedajúci bit pre povolenie prerušenia vypnutý, bude toto prerušenie zapamätané až do okamžiku povolenia prerušenia. Ak takéto prerušenie chceme zrušiť, musíme ho vynulovať softwarovo.
Obdobne to platí aj pre prípad, keď bit **GIE** je vynulovaný. Keď ho zapneme, nastavené prerušenia sa vykonajú v poradí priority.

2. typ prerušenia bude nastavený tak dlho, ako bude prítomná požiadavka o prerušenie. Takéto prerušenie nemusí mať *Interrupt Flag* (záchytný register). Ak požiadavka o prerušenie zmizne skorej ako sa začne obsluhovať, prerušovací podsystém sa bude správať tak, ako keby táto požiadavka nikdy nenastala.
Pri návrate z prerušenia sa prerušovací podsystém správa tak, ako keby v nasledujúcom kroku neexistovala požiadavka na akékoľvek prerušenie. T.j. vykoná sa jedna inštrukcia „hlavného programu“ a potom sa bude pýtať: Je nejaká ďalšia požiadavka o prerušenie? Ak áno, obslúži prerušenie.
!!!Poznámka!!! *Status Register* nie je automaticky zálohovaný pri vstupe do

obsluhy prerušenia a obnovovaný pri návrate z prerušenia. Toto musí zabezpečiť software – teda programátor.
Ak použijeme inštrukciu **CLI** na vypnutie prerušenia, udeje sa to okamžite. Žiadne prerušenie sa nevykoná po inštrukcii **CLI**. Dokonca ani keď sa objaví počas vykonávania tejto inštrukcie.
Existuje niekoľko postupností inštrukcií pri vykonávaní ktorých musí byť prerušenie zakázané: Napr.: počas zápisu do **EEPROM**. Zápis do **EEPROM**, vnútornej, patri medzi tzv. atomické operácie.
Hodnota na pine **INTx** je vzorkovaná pred detekovaním zmeny. Ak hrana alebo prepnutie trvá dlhšie ako 1 **SC** bude generovať prerušenie. Kratšie pulzy nemusia byť zachytené. Ak navolíme prerušenie vyvolané úrovňou, táto musí trvať až do vstupu do obsluhy prerušenia.
Externé prerušenia sú spúšťané pinmi INT0 a INT1 alebo niektorým z pinov PCINT23..0. Upozorňujeme, že ak je povolené, prerušenia sa spustia, aj keď sú piny INT0 a INT1 alebo PCINT23..0 nakonfigurované ako výstupy.

Odozva na prerušenie, AVR

Odozva na požiadavku o prerušenie ktoréhokoľvek prerušenia trvá minimálne 4 **SC**. Po tomto čase sa začne vykonávať obsluha prerušenia.
Počas štyroch **SC** sa uloží obsah **PC** do *Stack*u.
Do **PC** sa zapíše adresa vektoru obsluhy prerušenia a vykoná sa inštrukcia **jump**. Toto trvá 3 **SC**.
Ak sa **interrupt** objaví počas viac cyklovej inštrukcie, táto sa najskôr dokončí a potom sa začne obsluhovať prerušenie.
Ak sa prerušenie objaví počas **SLEEP** módu, odozva sa predĺži o 4 **SC**. Počas tejto doby „*start-up*“ **MCU** zistí z čoho sa to vlastne prebúdza.
Návrat z podprogramu prerušenia trvá 4 **SC**. Počas tejto doby sa vyberú zo zásobníka dva byty – obnoví sa **PC**. **SP** sa inkrementuje o 2 a znovu sa nastaví bit **I** v registri **SREG**. Ak existuje počas inštrukcie **RETI** nejaká neobslúžená požiadavka o prerušenie, najskôr sa vykoná **RETI** potom jedna inštrukcia s hlavného programu a potom sa môže začať realizovať obsluha prerušenia.