Protokol – „SADA DUM“

Číslo sady DUM:	**VY_32_INOVACE_EL_12**
Název sady DUM:	**Programování mikroprocesorů Atmel AVR**
Název a adresa školy:	**Střední průmyslová škola, Hronov, Hostovského 910, 549 31 Hronov**
Registrační číslo projektu:	**CZ.1.07/1.5.00/34.0596**
Číslo a název šablony:	**III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT**
Obor vzdělávání:	**Elektrotechnika**
Tématická oblast ŠVP:	**Počítačové řídicí systémy**
Předmět a ročník:	**Počítačové řídicí systémy**
Autor:	**Ing. Jiří Dítě**

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2013-11-13]. Dostupné z: www.atmel.com
- The Learning Pit Dot Com [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.thelearningpit.com

Datum vytvoření: **25.3.2014**

Anotace	Využití ve výuce
Po skončení výuky podle této sady budou žáci seznámeni se základy hardwaru a programováním mikroprocesorů ATmega32 v assembleru.	Tato sada bude využívána průběžně během školního roku v předmětu Počítačové řídicí systémy ve 3. ročníku průmyslové školy v oboru elektrotechnika, v části předmětu programování mikrokontrolérů.

Vytvořeno v rámci projektu OP VK zavedení nové oblasti podpory 1.5 s názvem Zlepšení podmínek pro vzdělávání na středních školách.

Digitální učební materiál

Programování ATmega32 – Základní syntaxe programu

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_01

Anotace: Základní syntaxe programu, direktivy assembleru, instrukce, návěští, komentář

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2013-11-13]. Dostupné z: www.atmel.com

Programování ATmega32 – Základní syntaxe programu

1. Základní pojmy a syntaxe programu

- ***Assembler*** – jazyk symbolických adres
  - ***Instrukce*** - symbolický zápis strojových instrukcí mikroprocesoru (viz. Instrukční sada)
  - ***Direktivy assembleru*** – instrukce pro překladač. Slouží pro vyhrazení paměťového prostoru, uložení konstant do paměti programu, výběr paměťového prostoru, vložení zdrojového textu z externího souboru nebo určení konce zdrojového textu.
    - **.EQU** - definice symbolu: slouží pro pojmenování literálu nebo nebo zavedení nového jména pro stávající symbol. (např.: .EQU konstanta=$19)
    - **.SET** - nastavení hodnoty symbolu: podobné jako EQU, symbol se oproti EQU může modifikovat (měnit)
    - **.CSEG** - výběr programového segmentu (oblasti kde bude psán program)
    - **.DSEG** - výběr datového segmentu (oblasti dat)
    - **.ESEG** - výběr segmentu EEPROM
    - .BYTE - vyhrazení paměťového prostoru v bajtech (v datové paměti): udává, kolik bajtu se vyhradí pro danou proměnnou (např.: PromA: .BYTE 4)
    - **.DB** - uložení konstanty do paměti programu nebo EEPROM (rozměr bajt): (např.: Stavy: .DB 12, 15,-18, $25)
    - **.DW** - uložení konstanty do paměti programu nebo EEPROM (rozměr word)
    - **.ORG** - nastavení počáteční adresy segmentu: (výchozí hodnoty CSEG (0), ESEG (0), DSEG ($60)
    - **.DEF** - definice symbolického jména registru: umožňuje uživatelské pojmenování registrů (např.: .DEF rychlost=R16)
    - **.INCLUDE** - vložení obsahu externího souboru: vloží do zdrojového souboru obsah jiného souboru

- ***Adresování***
  - ***Přímé adresování:*** Součástí instrukce je přímo adresa, která odkazuje na zdroj nebo cíl, kde je uložen operand. Mikroporcesory AVR mohou adresovat registry (R0-R31), vstupně-výstupní registry, paměť dat a paměť programu.
  - **Nepřímé adresování:** V zápisu instrukce je uveden jeden z adresovacích registrů X, Y, nebo Z ve kterém je uložena adresa, se kterou instrukce pracuje. Nepřímé adresování je vhodné pro zápis cyklů, při kterém dochází k opakovanému zápisu (čtení) na některou adresu.

- ***Symboly*** - Jsou to alfanumerické znaky vyjadřující číselné nebo znakové konstanty. Mohou to být malá i velká písmena, číslice a speciální znaky. Symbol nesmí začínat číslicí. Překladač nerozlišuje velká a malá písmena. Některé symboly nelze použít, protože jsou vyhrazena (například názvy instrukcí, registrů apod.)

- ***Návěští*** **-** Speciální symbol, který je zakončen dvojtečkou. Používá se k označení daného místa v programu, nebo pro pojmenování proměnné.

- ***ASCII literály*** **-** Rozlišujeme na znakové (uzavírají se do apostrofů, např.: 'A', nebo'5',…) a na řetězcové (uzavírají se do uvozovek, např.: "ahoj", nebo "1254")

- ***Komentář*** **-** Jedná se o poznámku v programu, která nemá žádný vliv na běh programu. Začíná vždy středníkem a končí koncem řádku.

- ***Čísla a operátory*** **-** Překladač podporuje zápis čísel ve dvojkové, osmičkové, desítkové a šestnáctkové soustavě:

  - ***Čísla***

Číselná soustava	Zápis	Číslice	Příklad zápisu
dvojková (binární)	předchází 0b	0, 1	0b11111010
osmičková (oktalová)	předchází 0	0 až 7	0372
desítková (decimální)	bez vodící 0	0 až 9	250
šestnáctková (hexadecimální)	předchází $ nebo 0x	0 až 9, A až F	$fa, 0xfa

  - ***Operátory assembleru***

Symbol	Název	Popis	Priorita
!	logická negace	vrací 1, pokud je výraz 0; vrací 0 pokud je výraz různý od 0	14
~	bitová negace	vrací invertované bity výrazu	14
–	záporné znaménko	otočí znaménko čísla (pro 1 vrátí –1, pro –25 vrátí 25)	14
*	násobení	vynásobí dva operandy	13
/	dělení	podělí celočíselně dva operandy	13
+	sčítání	sečte dva operandy	12
–	odčítání	odečte dva operandy	12
<<	posuv vlevo	posune levý operand o tolik bitů vlevo, kolik je určeno pravým operandem	11
>>	posuv vpravo	posune levý operand o tolik bitů vpravo, kolik je určeno pravým operandem	11
<	menší než	je-li levý operand menší než pravý, vrátí 1 (jinak 0)	10
<=	menší nebo rovno	je-li levý operand menší nebo roven pravému, vrátí 1 (jinak 0)	10
>	větší než	je-li levý operand větší než pravý, vrátí 1 (jinak 0)	10
>=	větší nebo rovno	je-li levý operand větší nebo roven pravému, vrátí 1 (jinak 0)	10
==	rovno	jsou-li oba operandy shodné, vrátí 1 (jinak 0)	9
!=	nerovno	jsou-li oba operandy různé, vrátí 1 (jinak 0)	9
&	bitový součin	vrátí bitový součin obou operandů	8
|	bitový součet	vrátí bitový součet obou operandů	7
^	výlučný bitový součet	vrátí výlučný bitový součet obou operandů	6
&&	logický součin	vrátí logický součin obou operandů (jsou-li oba operandy různé od 0, vrátí 1)	5
||	logický součet	vrátí logický součet obou operandů (jsou-li oba operandy rovny 0, vrátí 0)	4

- ***Speciální názvy:***

  RAMEND (poslední adresa v datové paměti - $085FH)
  FLASHEND (poslední adresa v programové paměti - $3FFFH)
  E2END (poslední adresa v paměti EEPROM - $03FFH)
  LOW (vrátí nižší bajt výrazu)
  HIGH (vrátí vyšší bajt výrazu)

2. Příklad programu

```
; >>>> Program – blikani LED na portu C.2

.include "m32def.inc"                          ; prirazeni jmen registru

; >>>> DATA ... oblast pameti dat
.dseg
.org    0x60                                   ;interni RAM ... pocatecni adresa

; >>>> PROGRAM ... oblast pameti programu
.cseg
.org    0x0000                                 ; zacatek pameti FLASH

Ini:            ldi     R16,LOW(RAMEND)        ; definice zasobniku
                out     SPL,R16
                ldi     R16,HIGH(RAMEND)
                out     SPH,R16

                sbi     DDRC,2                 ; pocatecni nastaveni ... smer vystup PC.2

Start:          cbi     PORTC,2                ; rozsvit LED (posle log.0 na port PC.2)
                rcall   DEL400                 ; cekej 0,4s

                sbi     PORTC,2                ; zhasni LED posle log.1 na port PC.2)
                rcall   DEL400                 ; cekej 0,4s

                rjmp    Start                  ; skok zpet na start

.include "delay.inc"                           ; zpozdovaci rutina
```

3. Úkol

a) Spusťte AVR studio a opište program výše. Program přeložte a nahrajte do mikrokontroléru. Proveďte verifikaci. Zkontrolujte funkčnost programu.

b) Prohlédněte si složku vámi vytvořeného projektu, najděte soubor s příponou *.hex a prohlédněte si ho. Pokuste se dekódovat jednotlivé instrukce a jejich operandy.

Digitální učební materiál

Programování ATmega32 – Operace s daty (přesuny dat)

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_02

Anotace: Přesun dat mezi registry, pamětí programu a pamětí dat

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2013-11-13]. Dostupné z: www.atmel.com

Programování ATmega32 – Operace s daty (přesuny dat)

1. Přesuny dat

Pod pojmem přesun dat rozumíme základní operace s daty, jejich načtení, uložení, kopírování atd.:

a) uložení konstanty do registru
b) přesun dat mezi registry
c) přesun dat mezi pamětí dat a registry
d) přesun dat mezi pamětí programu a registry

---

a) Uložení konstanty do registru – instrukce **LDI**

```
LDI     R16,0x25        ; do registru R16 uloží konstantu (25)H
```

Pozn.: Instrukce LDI pracuje pouze s registry R16 – R31 !

---

b) Přesun dat mezi registry – instrukce **MOV, IN, OUT**

```
MOV     R4,R16          ; obsah registru R16 se zkopíruje do registru R4
```

tedy: R4 … cílový registr
R16 … zdrojový registr

R4: _____ ← R16: _____

```
IN      R16,PINA        ; načte hodnotu I/O registru PINA do registru R16

OUT     PORTB,R18       ; hodnotu z registru R18 uloží do I/O registru PORTB
```

---

c) Přesun mezi pamětí dat a registry – instrukce **LD, ST**

Pro přesun používáme nepřímé adresování pomocí některého z registrů X, Y, nebo Z.

```
LD      R8,X            ; hodnota uložená v datové paměti na adrese, které
                        obsahuje registr X se zkopíruje do registru R8

LD      R5,X+           ; hodnota uložená v datové paměti na adrese, kterou
                        obsahuje registr X se zkopíruje do registru R5 a následně
                        se adresa v registru X zvýší o 1

LD      R17,-X          ; nejprve se adresa uložená v registru X sníží o 1 a poté
                        se hodnota uložená v datové paměti na adrese, kterou
                        obsahuje registr X zkopíruje do registru R17

ST      Y,R9            ; hodnota uložená v registru R9 se zkopíruje do datové
                        paměti na adresu uloženou v registru Y
```

d) Přesun mezi pamětí programu a registry – instrukce **LPM**

Pro přesun dat (konstant) z paměti programu (pouze čtení z paměti FLASH)

LPM R16,Z	; přesune hodnotu z paměti programu z adresy uložené v registru Z do registru R16
LPM R16,Z+	; přesune hodnotu z paměti programu z adresy uložené v registru Z do registru R16, následně zvýší adresu v registru Z o 1
LPM	; přesune hodnotu z paměti programu z adresy uložené v registru Z do registru R0

---

2. Úkoly

a) Vytvořte program, který vloží do registru R5 konstantu $(6C)_H$.

b) Vytvořte program, který vynuluje blok 10-ti bajtů v paměti dat počínaje adresou $(100)_H$. Nakreslete vývojový diagram.

c) Vytvořte program, který přesune blok 20-ti bajtů z paměti programu počínaje adresou $(2A0)_H$ do datové paměti od adresy $(60)_H$. Nakreslete vývojový diagram a řešte pomocí cyklu.

3. Příklad možného řešení

```
.include        "m32def.inc"                ; prirazeni jmen registru

.cseg
.org            0x0000

; >>> a) do R5 konstantu 6Ch

                ldi     R16,0x6C            ; nutno vlozit přes registr R16 – R31
                mov     R5,R16


; >>> b) vynulovani oblasti dat od adresy 100H

                ldi     XL,0x00             ; dolni bajt adresy
                ldi     XH,0x01             ; horni bajt adresy

                ldi     R17,0x00            ; vynulujeme pom registr
                ldi     R18,10              ; pocitadlo – 10 opakovani

cykl1:          st      X+,R17              ; hodnotu z R17 (0) vlozime na adresu v registru X
                                                    ; a inkrementujeme X
                dec     R18                 ; snizime pocitadlo a dokud není 0, tak skaceme
                brne    cykl1


; >>> c) presun oblasti dat

                ldi     XL,0x60             ; dolni bajt adresy v pameti dat
                ldi     XH,0x00             ; horni bajt adresy v pameti dat

                ldi     ZL,0x40             ; dolni bajt adresy v pameti programu
                ldi     ZH,0x05             ; horni bajt adresy v pameti programu

                ldi     R18,20              ; pocitadlo na 20

cykl2:          lpm     R17,Z+              ; cteni z pameti programu
                st      X+,R17              ; ulozeni do pameti dat

                dec     R18                 ; snizime pocitadlo a dokud není 0, tak skaceme
                brne    cykl2


konec:          rjmp    konec               ; konec
```

Digitální učební materiál

Programování ATmega32 – Logické operace, maskování

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_03

Anotace: Logické operace (AND, OR, XOR, NOT), maskování

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2013-11-13]. Dostupné z: www.atmel.com

Programování ATmega32 – Logické operace, maskování

1. Logické operace

Jsou to:
a) logický součin
b) logický součet
c) exkluzivní součet
d) logická negace (jednotkový doplněk)
e) nastavení a nulování bitů podle masky
f) rychlé nulování nebo nastavení všech bitů v registru

---

a) Logický součin – instrukce **AND, ANDI**

	10111010_B
AND	00001111_B
	00001010_B

```
AND   R5,R17          ; provede log. součin dat v registrech R5 a R17
                      ; výsledek uloží do registru R5

ANDI  R18,0b01100011  ; provede log. součin dat v registru R18
                      ; s konstantou, výsledek bude v registru R18
```

Pozn.: Instrukce ANDI pracuje pouze s registry R16 – R31 !

---

b) Logický součet – instrukce **OR, ORI**

	11100001_B
OR	00100100_B
	11100101_B

```
OR    R15,R7          ; provede log. součet dat v registrech R15 a R7
                      ; výsledek uloží do registru R15

ORI   R18,0b11101011  ; provede log. součet dat v registru R18
                      ; s konstantou, výsledek bude v registru R18
```

Pozn.: Instrukce ORI pracuje pouze s registry R16 – R31 !

---

c) Exkluzivní součet – instrukce **EOR**

	11101101_B
XOR	01010101_B
	10111000_B

```
EOR   R4,R14      ; provede exkluzivní součet dat v registrech R4 a R14
                  ; výsledek uloží do registru R4
```

---

d) Logická negace – instrukce **COM**

```
COM   R16         ; zneguje všechny bity v registru R16
```

Pozn.: Logickou negaci také někdy nazýváme, jako tzv. jednotkový doplněk

---

e) Nastavení a nulování bitů podle masky – instrukce **SBR, CBR**

	11110000_B		11110000_B
SBR	01010101_B	CBR	01010101_B
	11110101_B		00000101_B

```
SBR   R16,0b01100011   ; nastaví některé bity v registru R16, podle masky
                       ; kde jsou v masce jedničky, tak ty bity se nastaví

CBR   R16,0b01100011   ; nuluje některé bity v registru R16, podle masky
                       ; kde jsou v masce jedničky, tak ty bity se vynulují
```

Pozn.: Instrukce CBR a SBR pracuje pouze s registry R16 – R31 !

---

f) Rychlé nulování nebo nastavení všech bitů v registru – instrukce **CLR, SER**

```
CLR   R18         ; vynuluje všechny bity v registru R18

SER   R18         ; nastaví všechny bity v registru R18
```

Pozn.: Všechny výše zmíněné instrukce mimo instrukce SER také mění příznaky Z, N a V, instrukce COM navíc mění i příznak C !!!

C … příznak přetečení
V … příznak přeplnění
N … příznak záporného výsledku
Z … příznak nulového výsledku

2. Maskování

Při maskování se snažíme nastavit, vynulovat, nebo znegovat některé bity v registru podle tzv. masky. Maska je konstanta, která nám určuje, se kterými bity se maskování provádí. S výhodou k tomu používáme instrukce ANDI, ORI, EOR, SBR nebo CBR.

ANDI – vynuluje ty bity, které mají v masce na patřičných bitech 0, ostatní nemění
ORI – nastaví ty bity, které mají v masce na patřičných bitech 1, ostatní nemění
EOR - neguje ty bity, které mají v masce na patřičných bitech 1, ostatní nemění
SBR – nastaví ty bity, které mají v masce na patřičných bitech 1, ostatní nemění
CBR – nuluje ty bity, které mají v masce na patřičných bitech 1, ostatní nemění

3. Úkoly

a) Vytvořte program, který vloží do registru R15 konstantu $(6C)_H$, do registru R16 konstantu $(87)_H$ a proveďte s těmito registry operace logický součet, logický součin a výběrový součet. Výsledky postupně uložte do registrů R20, R21 a R22.

b) Vytvořte program, který zneguje horní nibl, vynuluje 1. a nastaví 3. bit registru R5.

4. Příklad možného řešení

```
.include        "m32def.inc"                ; prirazeni jmen registru

.cseg
.org            0x0000

; >>> a) Vytvořte program, který vloží do registru R15 konstantu (6C)H,
; do registru R16 konstantu (87)H a proveďte s těmito daty operace
; logický součet, logický součin a výběrový součet. Výsledky postupně uložte
; do registrů R20, R21 a R22.

                ldi     R16,0x6C            ; do R15 <- 6Ch
                mov     R15,R16

                ldi     R16,0x87            ; do R16 <- 87h

                mov     R20,R15             ; log. soucet
                or      R20,R16

                mov     R21,R15             ; log. soucin
                and     R21,R16

                mov     R22,R15             ; vyberovy soucet
                eor     R22,R16


; >>> b) Vytvořte program,
; který zneguje horní nibl, vynuluje 1. a nastaví 3. bit registru R5


                ldi     R16,0b11110000      ; maska pro negaci horniho niblu
                eor     R5,R16

                ldi     R16,0b11111101      ; maska pro nulovani 1. bitu
                and     R5,R16

                ldi     R16,0b00001000      ; maska pro nastaveni 3. bitu
                or      R5,R16


konec:          rjmp    konec               ; konec
```

Digitální učební materiál

Programování ATmega32 – Aritmetické operace

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_04

Anotace: Aritmetické operace (součet, rozdíl, násobení, inkrementace, dekrementace,…)

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2013-11-13]. Dostupné z: www.atmel.com

Programování ATmega32 – Aritmetické operace

1. Aritmetické operace

Jsou to:

a) aritmetický součet
b) aritmetický rozdíl
c) inkrementace a dekrementace
d) aritmetická negace – dvojkový doplněk
e) násobení

---

a) Aritmetický součet – instrukce **ADD, ADC, ADIW**

Př.1		10111010_B	**Př.2**		10010011_B
	+	00001111_B		+	11011000_B
	C=0	11001001_B		C=1	01101011_B
Př.3		10111010_B	**Př.4**		10010011_B
	+	00001111_B		+	11011000_B
	+C	0_B		+C	1_B
	C=0	11001001_B		C=1	01101100_B

```
ADD   R5,R17     ; provede arit. součet dat v registrech R5 a R17
                 ; výsledek uloží do registru R5 –> viz. Př.1, Př.2

ADC   R18,R17    ; provede arit. součet dat v registrech R18 a R17
                 ; včetně přenosu C
                 ; výsledek uloží do registru R18 –> viz. Př.3, Př.4

ADIW  R24,45     ; provede arit. součet dat v páru registrů R25:R24
                 ; s konstantou v rozsahu max. (0 – 63) - 45
                 ; první operand může být pouze R24, R26, R28 a R30
```

---

b) Aritmetický rozdíl – instrukce **SUB, SUBI, SBC, SBIW**

Př.1		10111010_B	**Př.2**		10010011_B
	-	00001111_B		-	11011000_B
	C=0	10101011_B		C=1	10111011_B
Př.3		10111010_B	**Př.4**		10010011_B
	-	00001111_B		-	11011000_B
	-C	0_B		-C	1_B
	C=0	10101011_B		C=1	01101010_B

```
SUB   R5,R17     ; provede rozdíl dat v registrech R5 a R17
                 ; výsledek uloží do registru R5 –> viz. Př.1, Př.2
```

```
SBC     R18,R17     ; provede rozdíl dat v registrech R18 a R17
                    ; včetně výpůjčky C
                    ; výsledek uloží do registru R18 –> viz. Př.3, Př.4

SBIW    R24,45      ; provede rozdíl dat v páru registrů R25:R24
                    ; s konstantou v rozsahu max. (0 – 63) - 45
                    ; první operand může být pouze R24, R26, R28 a R30

SUBI    R18,15      ; odečte konstantu 15 od registru R18
                    ; pracuje pouze s registry R16 – R31 !
```

---

c) Inkrementace a dekrementace – instrukce **INC, DEC**

```
INC     R4          ; zvýší hodnotu dat v registru R4 o 1

DEC     R15         ; sníží hodnotu dat v registru R15 o 1
```

---

d) Aritmetická negace – dvojkový doplněk – instrukce **NEG**

```
NEG     R16         ; vypočítá dvojkový doplněk k číslu uloženého v R16
```

Pozn. Dvojkový doplněk se vypočítá R16 = 0 – R16, tedy jedná se o vyjádření záporného čísla

---

e) Násobení – instrukce **MUL, MULS, MULSU**

```
MUL     R18,R17     ; celočíselné násobení R18 * R17 bez znaménka
                    ; Výsledek bude uložen v registrech R1:R0
                    ; 8 bitů * 8 bitů = 16 bitů

MULS    R24,R26     ; celočíselné násobení R24 * R26 se znaménkem
                    ; Výsledek bude uložen v registrech R1:R0
                    ; 8 bitů * 8 bitů = 16 bitů

MULSU   R18,R15     ; celočíselné násobení R18 * R15 kombinované
                    ; 1. operand (R18) … se znaménkem
                    ; 2. operand (R15) … bez znaménka
                    ; Výsledek bude uložen v registrech R1:R0
                    ; 8 bitů * 8 bitů = 16 bitů
```

Pozn.: Všechny výše zmíněné instrukce mění většinu příznaků!!!

---

2. Úkoly

a) Vytvořte program, který vypočítá součet 5-ti čísel uložených v registrech R2 – R6. Výsledek bude uložen v registru R7. Přenos do vyššího řádu neuvažujte. Výsledek v rozsahu (0-255).

b) Vytvořte program, který vypočítá aritmetický průměr 10-ti hodnot uložených v datové paměti od adresy 70H. Výsledek poté uloží na adresu 60H v datové paměti.

c) Vynásobte dvě 8-mi bitová čísla bez znaménka uložená v registrech R5 a R6. Výsledek uložte do registrového páru R8:R7 (Vyšší bajt na vyšší adrese).

3. Příklad možného řešení

```
.include "m32def.inc"                   ; prirazeni jmen registru

.cseg
.org            0x0000

; >>> a) Vytvořte program, který vypočítá součet 5-ti čísel uložených
; v registrech R2 - R6. Výsledek bude uložen v registru R7.

                clr     R7              ; vynuluje R7

                add     R7,R2           ; k registru R7 postupne pricte R2, R3, ...R6
                add     R7,R3
                add     R7,R4
                add     R7,R5
                add     R7,R6



; >>> b) Vytvořte program, který vypočítá aritmetický průměr 10-ti hodnot
; uložených v datové paměti od adresy 70H. Výsledek poté uloží na adresu 60H
; v datové paměti.

                ldi     XL,LOW(0x70)    ; adresa zacatku tabulky
                ldi     XH,HIGH(0x70)

                ldi     R16,10          ; pocitadlo 10 bajtů

                ld      R17,X+          ; nacte prvni hodnotu z tabulky do R17

skok1:          ld      R18,X+          ; nacte dalsi hodnotu z tabulky
                add     R17,R18         ; secte dve hodnoty
                lsr     R17             ; vydeli dvema

                dec     R16             ; sniz pocitadlo cyklu
                brne    skok1           ; skoc pokud nenula

                ldi     XL,LOW(0x60)    ; adresa pro ulozeni vysledku
                ldi     XH,HIGH(0x60)

                st      X,R17


; >>> c) Vynásobte dvě 8-mi bitová čísla bez znaménka uložená v registrech R5 a R6.
; Výsledek uložte do registrového páru R8:R7 (Vyšší bajt na vyšší adrese).

                mul     R5,R6           ; vynasobeni
                mov     R8,R1           ; presun vysledku z R1:R0 do R8:R7
                mov     R7,R0

konec:          rjmp    konec
```

Digitální učební materiál

Programování ATmega32 – Časové zpoždění

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_05

Anotace: Časové zpoždění realizované pomocí smyčky (jednoduché, vnořené)

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2013-11-13]. Dostupné z: www.atmel.com

Programování ATmega32 – Časové zpoždění

1. Časové zpoždění

Činnost mikroprocesoru je taktována pomocí hodinového kmitočtu, který je vytvářen zpravidla krystalovým oscilátorem připojeným z vnějšku mikroprocesoru. Náš školní mikroprocesor pracuje na kmitočtu 16MHz. Doba jednoho strojového cyklu je dána převrácenou hodnotou kmitočtu, tedy **1/16MHz = 62,5ns**. Doba vykonání jedné instrukce je zpravidla 1, nebo 2 strojové cykly.

V některých aplikacích potřebujeme činnost mikroprocesoru přibrzdit, protože rychlost vykonávání instrukcí je příliš rychlá. Jedná se například o sluchové vjemy (generování tónu určitého kmitočtu), vizuální vjemy (blikání, animace, …), časování vnějších periférií, vytváření časových průběhů na výstupech mikroprocesoru a mnoho dalších.

Pro tuto činnost využíváme časové zpoždění. To lze realizovat dvěma základními způsoby:

a) Časovou smyčkou – kdy procesor vykonává instrukce, které ho na nějakou dobu zaměstnají.

b) S využitím časovačů – obvodů, které jsou integrovány na čipu společně s mikroprocesorem.

V tomto materiálu se budeme zabývat pouze realizací pomocí časové smyčky, využití časovačů probereme později.

2. Realizace zpoždění časovou smyčkou

a) jednoduchá smyčka

Základní konstrukce programu vychází ze sekvence několika instrukcí, které se opakují, například:

```
Delay1:   ldi    R16,K1     ; doba trvani 1 cyklus, pocet opakovani instrukce 1x
skok1:    nop                ; doba trvani 1 cyklus, pocet opakovani instrukce K1x
          dec    R16         ; doba trvani 1 cyklus, pocet opakovani instrukce K1x
          brne   skok1       ; doba trvani 2 cykly, pocet opakovani instrukce (K1-1)x
                             ; doba trvani 1 cykl, počet opakovani instrukce 1x
```

Bude-li například konstanta **K1 = 20**, můžeme vypočítat dobu zpoždění:

počet cyklů = 1 cykl x 1 + 1cykl x 20 + 1 cykl x 20 + 2 cykly x 19 + 1 cykl x 1 = **80 cyklů**

T = 62,5ns x počet cyklů = 62,5ns * 80 = 5000ns = 5µs

Při maximální hodnotě v registru 0 (256) bude zpoždění 1024 cyklů, tedy 64µs.

Pro některé aplikace je toto zpoždění stále malé a je proto nutné jednoduchou smyčku několikrát zopakovat. Tím vzniká tzv. vnořená smyčka.

b) vnořená smyčka

Typický příklad s použitím vnořené smyčky:

```
Delay2:   ldi    R17,K2     ; trvani 1 cyklus, pocet opakovani instrukce 1x
skok3:    ldi    R16,K1     ; trvani 1 cyklus, pocet opakovani instrukce K2x
skok2:    nop               ; trvani 1 cyklus, pocet opakovani instrukce K1xK2x
          dec    R16        ; trvani 1 cyklus, pocet opakovani instrukce K1xK2x
          brne   skok2      ; trvani 2 cykly, pocet opakovani instrukce (K1-1)xK2x
                            ; trvani 1 cyklus, pocet opakovani instrukce 1xK2x
          dec    R17        ; trvani 1 cyklus, pocet opakovani instrukce K2x
          brne   skok3      ; trvani 2 cykly, pocet opakovani instrukce (K2-1)x
                            ; trvani 1 cyklus, pocet opakovani instrukce 1x
```

Budou-li například konstanty **K1 = 20 a K2 = 10**, můžeme vypočítat dobu zpoždění:

počet cyklů = 1 cykl x 1 + 1cykl x 10 + 1 cykl x 20 x 10 + 1 cykl x 20 x 10 + 2 cykly x 19 x 10 + 1 cykl x 10 + 1cykl x 10 + 2 cykly x 9 + 1 cykl x 1 = **830 cyklů**

T = 62,5ns x počet cyklů = 62,5ns * 830 = 51875ns = 51,875 µs

Při maximální hodnotě v registrech 0 (256) bude zpoždění 262912 cyklů, tedy 16,432ms.

Pozn.: Pro větší zpoždění se vnoří více smyček do sebe.

3. Příklad zpožďovací rutiny pro časy 1ms, 100ms, 200ms, 400ms, 800ms a 1s

```
;*********************************************
;* zpozdovaci rutina                         *
;* zpozdeni 1ms, 100ms, 200ms, 400ms, 800ms, 1s  *
;* podporgram vyuziva:    registry R23, R24, R25  *
;*********************************************

DEL1S:      rcall   DEL200      ; zpozdeni 1s
DEL800:     rcall   DEL400      ; zpozdeni 800ms
DEL400:     rcall   DEL200      ; zpozdeni 400ms
DEL200:     rcall   DEL100      ; zpozdeni 200ms

DEL100:     ldi     R23,100     ; zpozdeni 100ms
delc:       rcall   DEL1ms
            dec     R23
            brne    delc
            ret

DEL1ms:     ldi     R24,21      ; zpozdeni 1ms
delb:       ldi     R25,253
dela:       dec     R25
            brne    dela
            dec     R24
            brne    delb
            ret
```

4. Úkoly

a) Vytvořte program, který realizuje časové zpoždění 25 μs.

b) Vytvořte program, který realizuje časové zpoždění 5 ms. Využijte konstrukci programu s využitím vnořené smyčky.

5. Příklad možného řešení

```
.include "m32def.inc"

.cseg
.org            0


; a) zpoždění 25 us
; jednoduchá smyčka

.equ            K1=100                  ; konstanta poctu opakovani

Delay1:         ldi     R16,K1          ; doba trvani 1 cyklus, pocet opakovani instrukce 1x
skok1:          nop                     ; doba trvani 1 cyklus, pocet opakovani instrukce K1x
                dec     R16             ; doba trvani 1 cyklus, pocet opakovani instrukce K1x
                brne    skok1           ; doba trvani 2 cykly, pocet opakovani instrukce K1x



; b) zpoždění 5 ms
; vnořená symčka

.equ            K2=141                  ; konstanty poctu opakovani
.equ            K3=141

Delay2:         ldi     R17,K2          ; doba trvani 1 cyklus, pocet opakovani instrukce 1x
skok3:          ldi     R16,K3          ; doba trvani 1 cyklus, pocet opakovani instrukce K2x
skok2:          nop                     ; doba trvani 1 cyklus, pocet opakovani instrukce K3xK2x
                dec     R16             ; doba trvani 1 cyklus, pocet opakovani instrukce K3xK2x
                brne    skok2           ; doba trvani 2 cykly, pocet opakovani instrukce (K3-1)xK2x
                                        ; doba trvani 1 cyklus, pocet opakovani instrukce 1xK2x
                dec     R17             ; doba trvani 1 cyklus, pocet opakovani instrukce K2x
                brne    skok3           ; doba trvani 2 cykly, pocet opakovani instrukce (K2-1)x
                                        ; doba trvani 1 cyklus, pocet opakovani instrukce 1x


konec:          rjmp    konec
```

Digitální učební materiál

Programování ATmega32 – Obsluha portů ATmega32

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_06

Anotace: Základní principy pro obsluhu portů, registry pro vstup a výstup

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2013-11-13]. Dostupné z: www.atmel.com

Programování ATmega32 – Obsluha portů ATmega32

1. Porty mikrokontroléru ATmega32

Jedná se o skupinu vývodů, které zajišťují styk mikrokontroléru s okolím. Mikrokontrolér ATmega32 obsahuje čtyři 8-mi bitové vstupně výstupní porty označené A, B, C a D. Porty mohou sloužit jako univerzální vstupně výstupní porty, nebo mohou mít alternativní funkci (ADC, TWI, USART, Časovače/čítače,…)

- ***Porty jako obecné vstupy a výstupy***

Všechny porty mikrokontroléru jsou obousměrné, tedy mohou v daném okamžiku sloužit buď jako vstupy, nebo jako výstupy. Dále je možné u každého portu individuálně konfigurovat tzv. PULL-UP rezistory, které vnitřně připojují daný vstup k logické jedničce (+Ucc). Všechny porty mají vnitřní budiče, které mohou přímo budit LED.

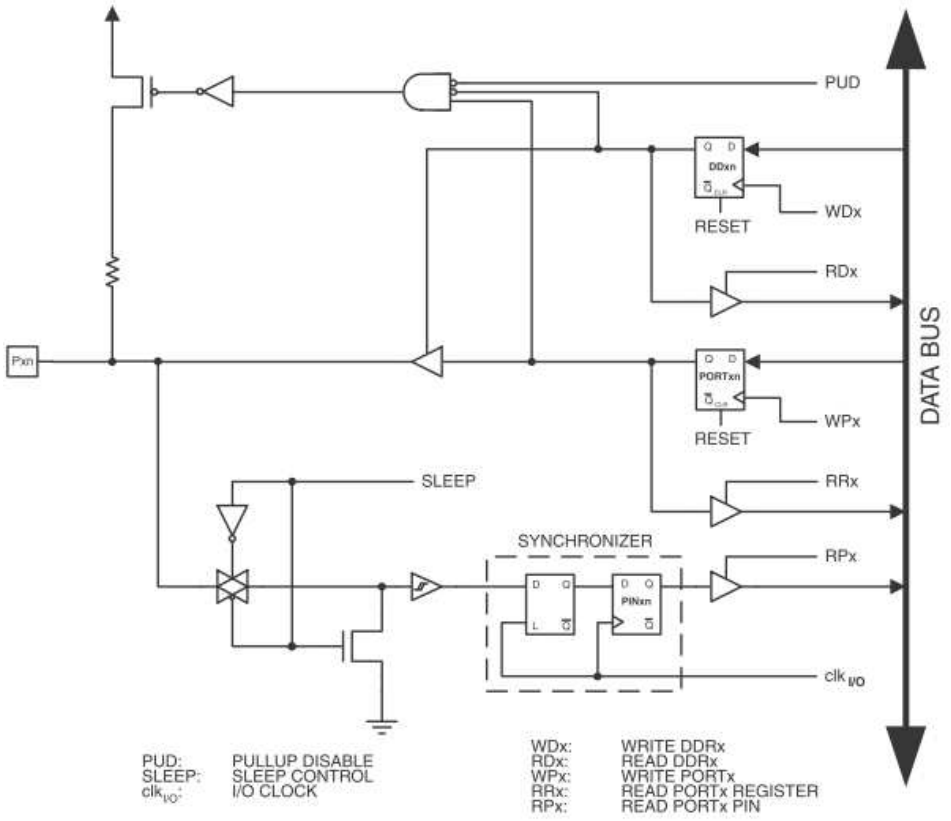
- ***Každý port je ovládán třemi registry:***
  - **registr DDRx** (Data Direction Registr) - určuje směr toku dat
    - log.1 - výstup z mikrokontroléru
    - log.0 - vstup do mikrokontroléru
  - **registr PORTx** - datový registr portu, obsahuje hodnotu zapsanou do vyrovnávacího registru
    - ***pokud je daný bit konfigurován jako výstup***, tak hodnota v registru PORTx odpovídá log. hodnotě na výstupu
    - ***pokud je daný bit konfigurován jako vstup***, tak hodnota v registru PORTx určuje, zda je připojen PULL-UP rezistor (log.1 - Pull-Up připojen, log.0 - Pull-Up odpojen)
  - **registr PINx** - (Pins Input) - registr pouze pro čtení, ve kterém je uložena hodnota odpovídající logické úrovni na jednotlivých vstupech

- Nesmíme zapomenout ještě na bit **PUD (Pull-Up Disable)**, což je bit obsažený v registru SFIOR, který slouží ke globálnímu odpojení všech PULL-UP rezistorů.

- **Po resetu jsou registry DDRx a PORTx vynulovány**, tedy jsou nastaveny jako vstupy bez PULL-UP rezistorů.

- **Porty lze řídit buď „bajtově“, nebo „bitově“.** Při „bajtové“ obsluze používáme zpravidla instrukce IN, OUT, MOV, LDI, a principy maskování. Při „bitové“ obsluze pak s výhodou používáme instrukce SBI, CBI, SBIS a SBIC.

DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)



2. Porty jako výstup

Při použití portu jako výstup využíváme pouze registry **PORTx a DDRx**. V registru DDRx nastavíme směr požadovaného vývodu portu na výstup zápisem log.1 a poté stav vývodu nastavujeme, nebo nulujeme zapsáním log.1, nebo log.0 do příslušného bitu registru PORTx.

Př.1: Nastavte všechny bity portu B jako výstupní, liché bity vynulujte a sudé bity nastavte.

a) Bajtová obsluha

```
ldi     R16,0xFF            ; nastav všechny bity pomocného registru
out     DDRB,R16            ; nastav PORTB - vše jako výstup
ldi     R16,0b01010101
out     PORTB,R16           ; nastaveni sudých a nulování lichých bitů PORTB
```

b) Bitová obsluha

```
sbi     DDRB,0      ; vývod B.0 jako výstup
sbi     DDRB,1      ; vývod B.1 jako výstup
sbi     DDRB,2      ; vývod B.2 jako výstup
sbi     DDRB,3      ; vývod B.3 jako výstup
sbi     DDRB,4      ; vývod B.4 jako výstup
sbi     DDRB,5      ; vývod B.5 jako výstup
sbi     DDRB,6      ; vývod B.6 jako výstup
sbi     DDRB,7      ; vývod B.7 jako výstup

sbi     PORTB,0     ; nastav vývod B.0 do log.1
sbi     PORTB,2     ; nastav vývod B.2 do log.1
```

```
        sbi     PORTB,4         ; nastav vývod B.4 do log.1
        sbi     PORTB,6         ; nastav vývod B.6 do log.1
        cbi     PORTB,1         ; nuluj vývod B.1 do log.0
        cbi     PORTB,3         ; nuluj vývod B.3 do log.0
        cbi     PORTB,5         ; nuluj vývod B.5 do log.0
        cbi     PORTB,7         ; nuluj vývod B.7 do log.0
```

Pozn.: Je vidět, že v tomto případě je bitová obsluha zdlouhavá a je vhodnější použít bajtovou. Nebude tomu však vždy. Pokud budeme pracovat pouze s jednotlivými vývody, tak bude bitová obsluha zcela jistě jednodušší, neboť při bajtové bychom musely provádět maskování, což by zpracování prodloužilo.

3. Porty jako vstup

Při použití portu jako vstup využíváme všechny registry **PORTx, DDRx a PINx**. V registru DDRx nastavíme směr požadovaného vývodu portu na vstup zápisem log.0. Zápisem do registru PORTx můžeme u některých vstupů zapnout tzv. PULL-UP rezistor, který zajistí definování stavu log.1 na vstupu (ošetření vstupu) připojením rezistoru PULL-UP k napájecímu napětí (provádíme zápisem log.1 do patřičného bitu registru PORTX). Hodnotu vstupu poté čteme pomocí instrukce PINx.

Př.2: Napište program, který bude hodnoty na vstupech portu A ukládat do registru R5. Na spodních čtveřici pinů aktivujte PULL-UP rezistory.

```
        ldi     R16,0x00            ; nuluj všechny bity pomocného registru
        out     DDRA,R16            ; nastav směr portu A - vše jako vstup
        ldi     R16,0b00001111
        out     PORTA,R16           ; aktivace PULL-UP rezistorů na spodních 4 bitech

cykl:   in      R5,PINA             ; načtení hodnot vstupů z portu A do R5
        …
        rjmp    cykl                ; cyklické čtení
```

4. Úkol

a) Vytvořte program, který logické hodnoty na vstupech portu A.0 – A.3 zneguje a odešle na výstupy portu C.0 – C.3. K indikaci logických úrovní na výstupu použijte LED diody zapojené anodou ke kladnému napájecímu napětí. Na vstupech budou připojena tlačítka k zemi.

b) Upravte předchozí program tak, aby se hodnoty ze vstupů A.0 a A.1 přímo kopírovaly na výstupy C.4 a C.5 a hodnoty ze vstupů A.2 a A.3 se negovaly a ukládaly na výstupy C.6 a C.7.

5. Možné řešení úkolu

- **Řešení úkol a)**

```
.include        "m32def.inc"

.cseg
.org            0

                ldi     R16,0x0f
                out     DDRC,R16            ; nastveni dolnich 4 bitu portu C jako vystup

                ldi     R16,0x00
                out     DDRA,R16            ; nasteni portu A jako vstup

                ldi     R16,0x0f
                out     PORTA,R16           ; aktivace PULL-UP rezistoru

skok:           in      R16,PINA            ; nacteni vstupu do R16
                com     R16                 ; zneguje cely registr R16

                andi    R16,0b00001111      ; ochrana - vynuluj vrchni 4 bity (nazijima me)

                out     PORTC,R16           ; odeslani na vystup

                rjmp    skok
```

- **Řešení úkol b)**

```
.include        "m32def.inc"

.cseg
.org            0

                ldi     R16,0xf0
                out     DDRC,R16            ; nastveni hornich 4 bitu portu C jako vystup

                ldi     R16,0x00
                out     DDRA,R16            ; nasteni portu A jako vstup

                ldi     R16,0x0f
                out     PORTA,R16           ; aktivace PULL-UP rezistoru

skok:           in      R16,PINA            ; nacteni vstupu do R16
                ldi     R17,0b00000011
                eor     R16,R17             ; negace 0. a 1. bitu

                andi    R16,0b00001111      ; ochrana - vynuluj vrchni 4 bity (nazijima me)

                swap    R16                 ; vymena niblů v R16

                out     PORTC,R16           ; odeslani na vystup

                rjmp    skok
```

Digitální učební materiál

Programování ATmega32 – Test1 – přesuny dat

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_07

Anotace: Test pro ověření znalostí – základní přesuny dat, adresování

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2013-11-13]. Dostupné z: www.atmel.com

Programování ATmega32 – Test1 přesuny dat

1. Úvodní pokyny pro vypracování

Pro mikrokontrolér ATmega32 vytvořte program podle následujícího zadání. Program vytvořte jako jeden celek do jednoho souboru a jednoho projektu. Jednotlivé body zadání vyznačte v programu pomocí komentáře (např.: ; a)).

Program přeložte a odsimulujte. Případné chyby opravte. Hotový program (respektive soubor se zdrojovým kódem ve formátu: T1_prijmeni.asm) nahrajte na disk Q:\upload.

2. Zadání

a) Do registru R4 uložte hodnotu (25)H, do registru R5 hodnotu (01100011)B a do registru R6 hodnotu (80)D.

b) Zkopírujte hodnotu z registru R4 do registru R24 a hodnotu z registru R5 do registru R25.

c) Znegujte 2., a 6. bit v registru R4.

d) Nastavte všechny sudé bity v registru R5.

e) Vynulujte 4 spodní bity v registru R6.

f) Do registrů R16 a R17 přesuňte data z oblasti dat z adres (75)H a (85)H.

g) Hodnotu z registru R5 uložte do paměti dat na adresu (90)H.

3. Bodové hodnocení

Za každý bod zadání jsou přiděleny maximálně 2 body. Tedy celkem 2*7 = 14 bodů. Hodnocení dle následující tabulky:

14 – 13 bodů	1
12 – 11 bodů	2
10 – 9 bodů	3
8 – 7 bodů	4
6 – 0 bodů	5

4. Možné řešení úkolu

```
.include        "m32def.inc"

.cseg
.org            0

;a)     Do registru R4 uložte hodnotu (25)H, do registru R5 hodnotu (01100011)B a do registru R6 hodnotu (80)D.

                ldi     R16,0x25
                mov     R4,R16
                ldi     R16,0b01100011
                mov     R5,R16
                ldi     R16,80
                mov     R6,R16

;b)     Zkopírujte hodnotu z registru R4 do registru R24 a hodnotu z registru R5 do registru R25.

                mov     R24,R4
                mov     R25,R5

;c) Znegujte 2., a 6. bit v registru R4

                ldi     R16,0b01000100
                eor     R4,R16

;d)     Nastavte všechny sudé bity v registru R5.

                ldi     R16,0b01010101
                or      R5,R16

;e)     Vynulujte 4 spodní bity v registru R6.

                ldi     R16,0b11110000
                and     R6,R16

;f)     Do registrů R16 a R17 přesuňte data z oblasti dat z adres (75)H a (85)H.

                ldi     XL,LOW(0x75)
                ldi     XH,HIGH(0x75)
                ld      R16,X

                ldi     XL,LOW(0x85)
                ldi     XH,HIGH(0x85)
                ld      R17,X

;g) Hodnotu z registru R5 uložte do paměti dat na adresu (90)H

                ldi     XL,LOW(0x90)
                ldi     XH,HIGH(0x90)
                st      X,R5

konec:          rjmp    konec
```

Digitální učební materiál

Programování ATmega32 – Světelné efekty - sekvence

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_08

Anotace: Základní principy pro obsluhu výstupů, tvorba sekvencí, využití tabulky

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2013-11-13]. Dostupné z: www.atmel.com

Programování ATmega32 – Světelné efekty, sekvence

1. Světelné efekty, sekvence

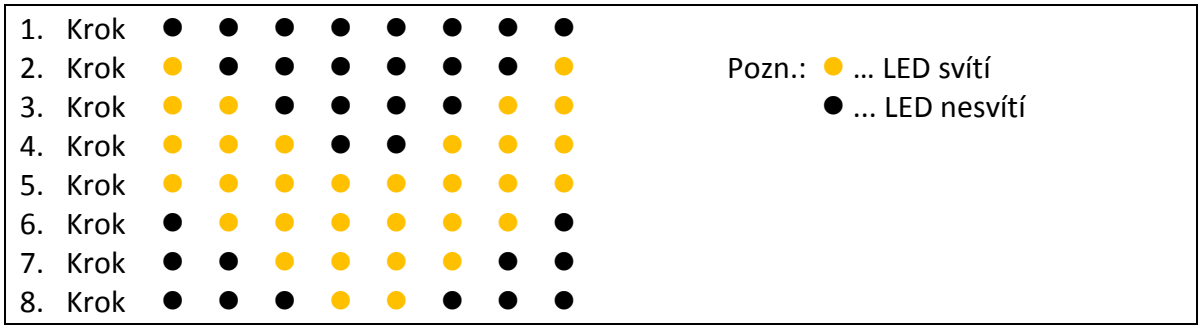
Světelné efekty jsou jedním z oblíbených úkolů, které se pomocí mikroprocesorů programují. Jedná se o určité sekvence logických jedniček a nul, které se posílají na výstupy a ovládají rozsvícení a zhasnutí jednotlivých světel připojených k výstupům.

My si vyzkoušíme takový program sestavit. Nejprve pomocí klasického lineárního programování a následně s využitím tabulek, uložených v paměti programu mikrokontroléru.

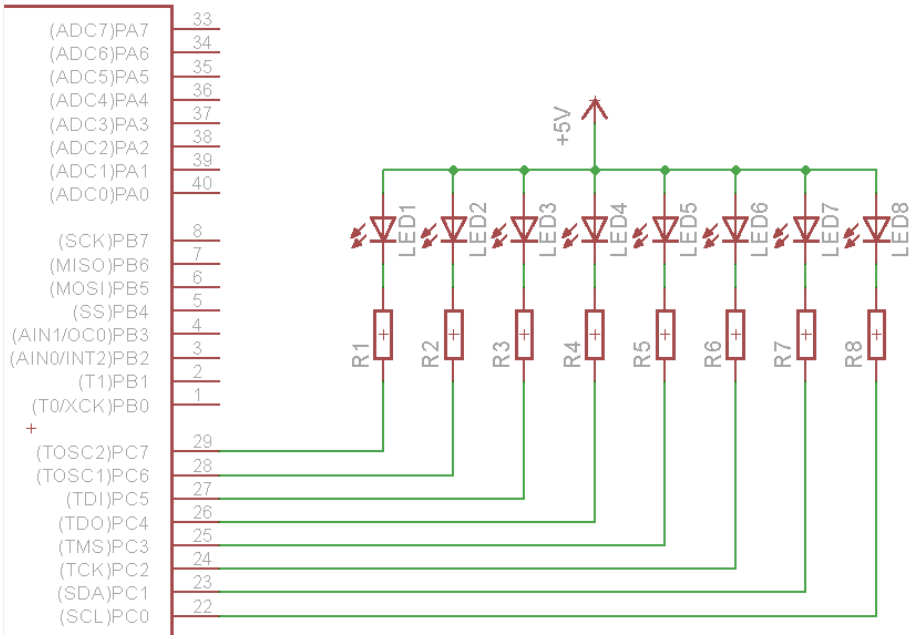
2. Jednoduchý světelný efekt

Vytvořte program, který vytvoří na portu C světelný efekt naznačený na obrázku níže. Efekt se skládá z 8-mi kroků, které se cyklicky opakují. Doba setrvání v jednom kroku je 400ms.

a) Pomocí lineárního programování
b) Pomocí tabulky sekvencí uložené v paměti programu



LED diody jsou připojeny k portu C dle následujícího schématu.



a) možné řešení s využitím klasického programování

```
.include        "m32def.inc"

.cseg
.org            0


ini:            ldi     R16,high(ramend)        ; inicializace zasobniku
                out     SPH,R16
                ldi     R16,low(ramend)
                out     SPL,R16

                ldi     R16,0xff                ; inicializace portu C
                out     DDRC,R16                ; vse jako vystupy


start:          ldi     R16,0b11111111
                out     PORTC,R16               ; vse zhasnout

                rcall   DEL400                  ; zpozdeni 400ms

                ldi     R16,0b01111110
                out     PORTC,R16               ; rozsvit LED 1,8

                rcall   DEL400                  ; zpozdeni 400ms

                ldi     R16,0b00111100
                out     PORTC,R16               ; rozsvit LED 1,2,7,8

                rcall   DEL400                  ; zpozdeni 400ms

                ldi     R16,0b00011000
                out     PORTC,R16               ; rozsvit LED 1,2,3,6,7,8

                rcall   DEL400                  ; zpozdeni 400ms

                ldi     R16,0b00000000
                out     PORTC,R16               ; rozsvit všechny LED

                rcall   DEL400                  ; zpozdeni 400ms


                ldi     R16,0b10000001
                out     PORTC,R16               ; zhasni LED 1,8

                rcall   DEL400                  ; zpozdeni 400ms

                ldi     R16,0b11000011
                out     PORTC,R16               ; zhasni LED 1,2,7,8

                rcall   DEL400                  ; zpozdeni 400ms

                ldi     R16,0b11100111
                out     PORTC,R16               ; zahsni LED 1,2,3,6,7,8

                rcall   DEL400                  ; zpozdeni 400ms

                rjmp    start                   ; opakuj


.include        "h:\prs\delay.inc"
```

b) možné řešení s využitím tabulky v paměti programu

```
.include        "m32def.inc"

.cseg
.org            0


ini:            ldi     R16,high(ramend)        ; inicializace zasobniku
                out     SPH,R16
                ldi     R16,low(ramend)
                out     SPL,R16

                ldi     R16,0xff                ; inicializace portu C
                out     DDRC,R16                ; vse jako vystupy


start:          ldi     R18,8                   ; pocet bajtu (sekvenci) v tabulce

                ldi     ZL,low(2*Tab)           ; nacti adresu tabulky do registru Z
                ldi     ZH,high(2*Tab)

skok:           lpm     R16,Z+                  ; do registru R16 nacti hodnotu z tabulky
                out     PORTC,R16               ; posli na vystupy portu C

                rcall   DEL400                  ; zpozdeni 400ms

                dec     R18                     ; sniz pocitadlo o 1
                brne    skok                    ; skok pokud neni 0

                rjmp    start                   ; opakuj


Tab:            .db     0b11111111,0b01111110
                .db     0b00111100,0b00011000
                .db     0b00000000,0b10000001
                .db     0b11000011,0b11100111


.include        "h:\prs\delay.inc"
```

3. Úkol

Výše uvedené příklady ověřte při praktickém cvičení a pokuste se modifikovat program tak, abyste vytvořili ještě minimálně tři různé druhy efektů s LED diodami podle vlastního výběru.

Digitální učební materiál

Programování ATmega32 – 7mi segmentový LED displej

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_09

Anotace: Druhy a obsluha 7mi segmentového displeje, postupné zobrazování čísel

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.atmel.com
- The Learning Pit Dot Com [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.thelearningpit.com

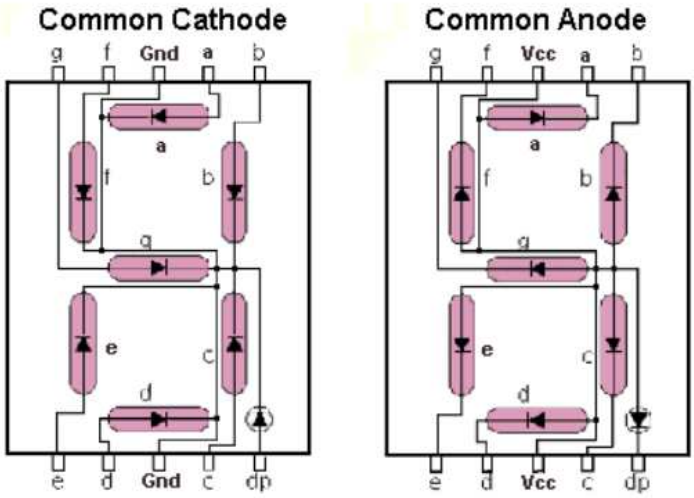
Programování ATmega32 – 7mi segmentový LED displej

1. Sedmisegmentový LED displej - druhy

Sedmi segmentové displeje můžeme rozdělit podle způsobu řízení na:

a) displej se společnou anodou Ccommon Anode): Jednotlivé segmenty mají jednu společnou elektrodu a tou je anoda, která je připojena ke kladnému pólu napájecího zdroje. Katody jednotlivých segmentů jsou vyvedeny jednotlivě a po přivedení záporného potenciálu se daný segment rozsvítí. Log. 0 odpovídá rozsvícenému segmentu, log. 1 zhasnutému.

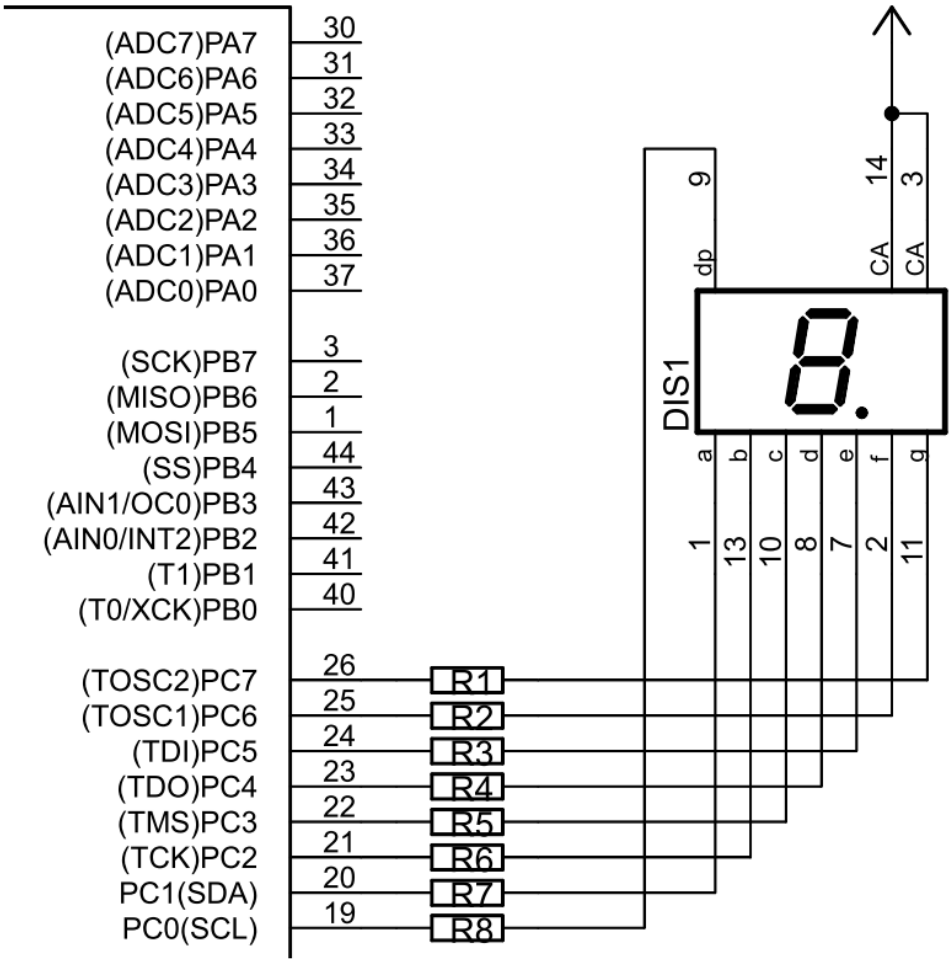
b) displej se společnou katodou (Common Cathode): Společnou elektrodou všech segmentů je tentokrát katoda, která je připojena k zápornému potenciálu napájecího zdroje. Segmenty se rozsvěcí přivedením kladného napětí na anody jednotlivých segmentů. Rosvícení segmentů je realizováno přivedením log. 1 a zhasnutí přivedením log. 0.



2. Zobrazované znaky

Nejčastěji vypadají zobrazované znaky dle následujícího obrázku. Výjimečně k nim můžeme přidat i některé znaky z abecedy, ale jsme značně omezeni rozložením segmentů. Jako znakové se častěji využívají šestnáctisegmentové displeje, nebo displeje maticové.

3. Připojení sedmisegmentovky k mikrokontroléru ATmega32



4. Úkol

Vytvořte program pro mikrokontrolér ATmega32, který po spuštění bude na 7mi segmentovém zobrazovači, který je připojen k mikrokontroléru k portu C, zobrazovat cyklicky čísla od 0 do 9 v intervalech 1s (bude zobrazovat jednotky sekund). Pro generování zpoždění využijte knihovnu delay.inc.

5. Možné řešení úkolu

```
.include        "m32def.inc"
.cseg
.org            0x0000

ini:            ldi     R16,LOW(RAMEND)         ; definice zasobniku
                out     SPL,R16
                ldi     R16,HIGH(RAMEND)
                out     SPH,R16

                ldi     R16,0xff                ; pocatecni nastaveni ... smer vystup cely port C
                out     DDRC,R16
                out     PORTC,R16               ; vse zhasnuto


start:          ldi     ZL,low(2*Tab)           ; nacti adresu tabulky do registru Z
                ldi     ZH,high(2*Tab)

                ldi     R18,10                  ; pocitadlo zobrazovanych cifer

skok:           lpm     R17,Z+                  ; do registru R17 nacti hodnotu z tabulky a zvys adresu o 1
                out     PORTC,R17               ; posli na vystupy na 7seg

                rcall   DEL1s                   ; cekej 1s

                dec     R18                     ; sniz pocitadlo o 1
                brne    skok                    ; pokud neni pocitadlo 0, tak skok

                rjmp    start                   ; znovu start


Tab:    .db     0x81,0xf3,0x49,0x61,0x33,0x25,0x05,0xf1,0x01,0x21


.include        "n:\prs\delay.inc"
```

Digitální učební materiál

Programování ATmega32 – Dekodér BCD -> 7segmentovka

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_10

Anotace: Softwarové řešení dekodéru BCD čísla na 7mi segmentový zobrazovač

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.atmel.com
- The Learning Pit Dot Com [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.thelearningpit.com

Programování ATmega32 – Dekodér BCD -> 7segmentovka

1. Úvod

Tyto dekodéry slouží pro dekódování informace přiváděné v BCD kódu na 7 výstupů, které ovládají jednotlivé segmenty na 7mi segmentové zobrazovači. Vyrábějí se buď jako hardwarové (například obvody 7446, 7447, …), nebo se dají vytvořit pomocí programu v mikrokontroléru (softwarově).

Příklad pravdivostní tabulky dekodéru pro displej se společnou katodou:

Zobrazované číslo	Vstupy				Výstupy						
	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

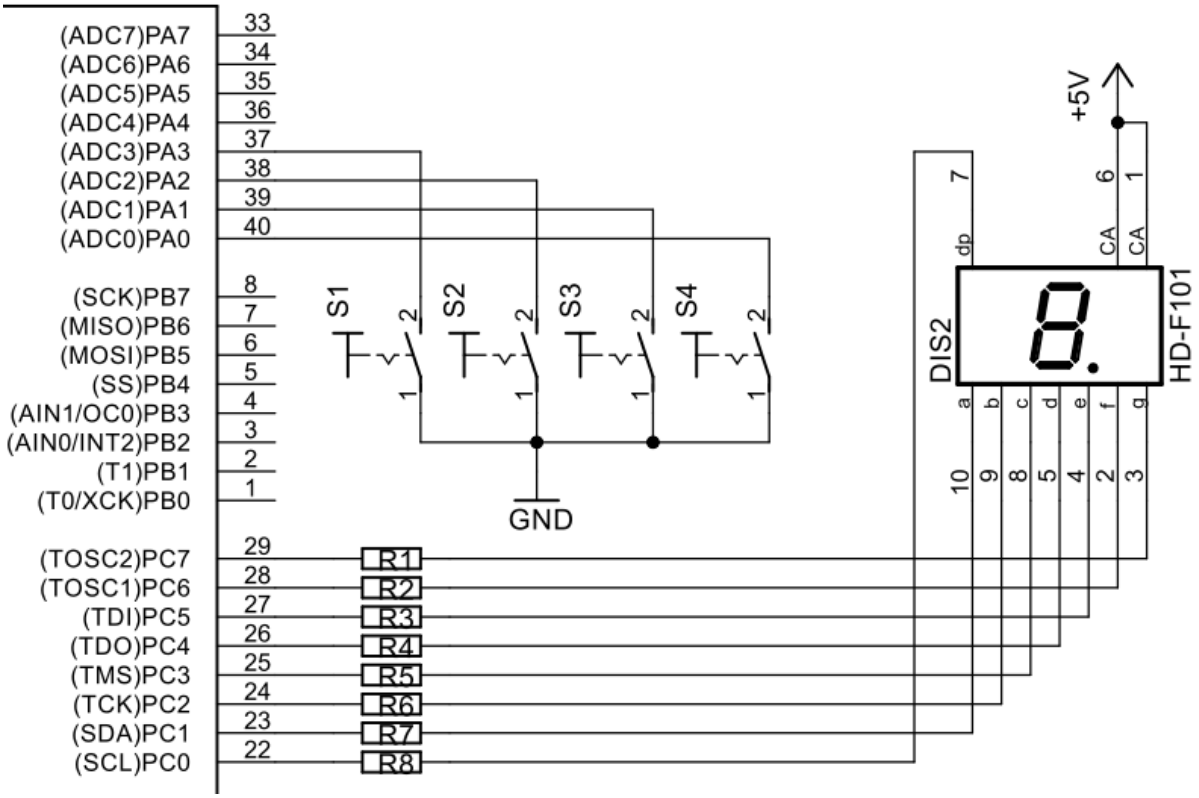
Příklad pravdivostní tabulky dekodéru pro displej se společnou anodou:

Zobrazované číslo	Vstupy				Výstupy						
	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0

2. Úkol

Realizujte dekodér BCD na sedmi segmentový zobrazovač se společnou anodou, který bude zobrazovat čísla 0 – 9. Ošetřete stavy pro kombinace na vstupech, ve kterých není funkce dekodéru definována (zhasnutím displeje, definováním speciálních kombinací a podobně). 7mi segmentový zobrazovač je připojen k mikrokontroléru prostřednictvím portu C. Vstupní BCD informace je přiváděna na spodní čtyři bity portu A (podle schéma připojení níže.

3. Připojení sedmisegmentovky a vstupů k mikrokontroléru ATmega32



4. Možné řešení úkolu

```
.include        "m32def.inc"
.cseg
.org            0x0000

ini:            ldi     R16,LOW(RAMEND)         ; definice zasobniku
                out     SPL,R16
                ldi     R16,HIGH(RAMEND)
                out     SPH,R16

                ldi     R16,0xff                ; pocatecni nastaveni ... smer vystup cely port C

                out     DDRC,R16
                out     PORTC,R16               ; vse zhasnuto

                ldi     R16,0x0f                ; PULL-UP PA0-3
                out     PORTA,R16


start:          in      R16,PINA                ; nacti z portu A vstupy
                andi    R16,0x0f                ; vymaskuj pouze spodni 4 bity

                ldi     ZL,low(2*Tab)           ; nacti adresu tabulky do registru Z
                ldi     ZH,high(2*Tab)

                add     ZL,R16                  ; pricti k ZL R16
                clr     R18
                adc     ZH,R18                  ; pricti pripadny prenos

                lpm     R17,Z                   ; do registru R17 nacti hodnotu z tabulky
                out     PORTC,R17               ; posli na vystupy na 7seg


                rjmp    start                   ; znovu start


Tab:    .db     0x81,0xf3,0x49,0x61,0x33,0x25,0x05,0xf1,0x01,0x21,0xfe,0xfe,0xfe,0xfe,0xfe,0xfe
```

Digitální učební materiál

Programování ATmega32 – RS klopný obvod

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_11

Anotace: RS klopný obvod, tlačítkové zapnutí a vypnutí výstupu

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

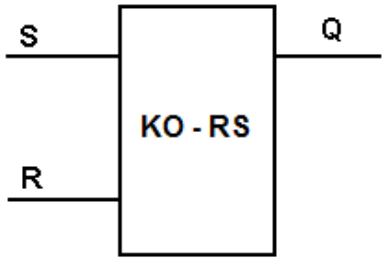
Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.atmel.com
- The Learning Pit Dot Com [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.thelearningpit.com

Programování ATmega32 – RS klopný obvod

1. Co je to RS klopný obvod?

Jedná se o sekvenční obvod, který má dva vstupy R…(Reset - nulování) a S… (Set - nastavení) a jeden výstup označený zpravidla jako Q. Výstup se může nalézat v jednom ze stabilních stavů (log. 0, nebo log. 1).



Tyto obvody nalezneme v nejrůznějších aplikacích (např.: ovládání osvětlení typu ZAPNUTO, VYPNUTO, tlačítkové ovládání stolní vrtačky a jiných pohonů typu START, STOP a podobně).

Chování obvodu je popsáno v následující tabulce:

Vstup R	Vstup S	Výstup Qn+1	Popis funkce
0	0	Qn	Není aktivní ani jeden vstup, výstup se nemění a setrvává v předchozím stavu
0	1	1	Při aktivaci vstupu S (Set) se výstup nastaví (log. 1)
1	0	0	Při aktivaci vstupu R (Reset) se výstup vynuluje (log. 0)
1	1	X	Neurčitý stav – tento stav nesmí nastat, pokud není ošetřen (viz. dále)

Definováním neurčitého stavu, kdy jsou oba vstupy aktivní (odpovídá např. stisknutí obou tlačítek současně), můžeme vytvořit RS klopný obvod s přednostním nulováním (pokud přiřadíme výstupu místo neurčitého stavu stav log. 0), nebo RS klopný obvod s přednostním nastavením (pokud přiřadíme výstupu místo neurčitého stavu stav log. 1).

My budeme řešit tento RS klopný obvod softwarově a s výhodou k tomu můžeme požít buď instrukce pro porovnání společně s instrukcemi podmíněných skoků (cp, cpi, brne, breq, …), což je zpravidla univerzální řešení, nebo lze využít instrukce přeskoků, které nám mnohdy program zjednoduší (sbic, sbis, sbrc, sbrs). V další kapitole si ukážeme obě varianty řešení.

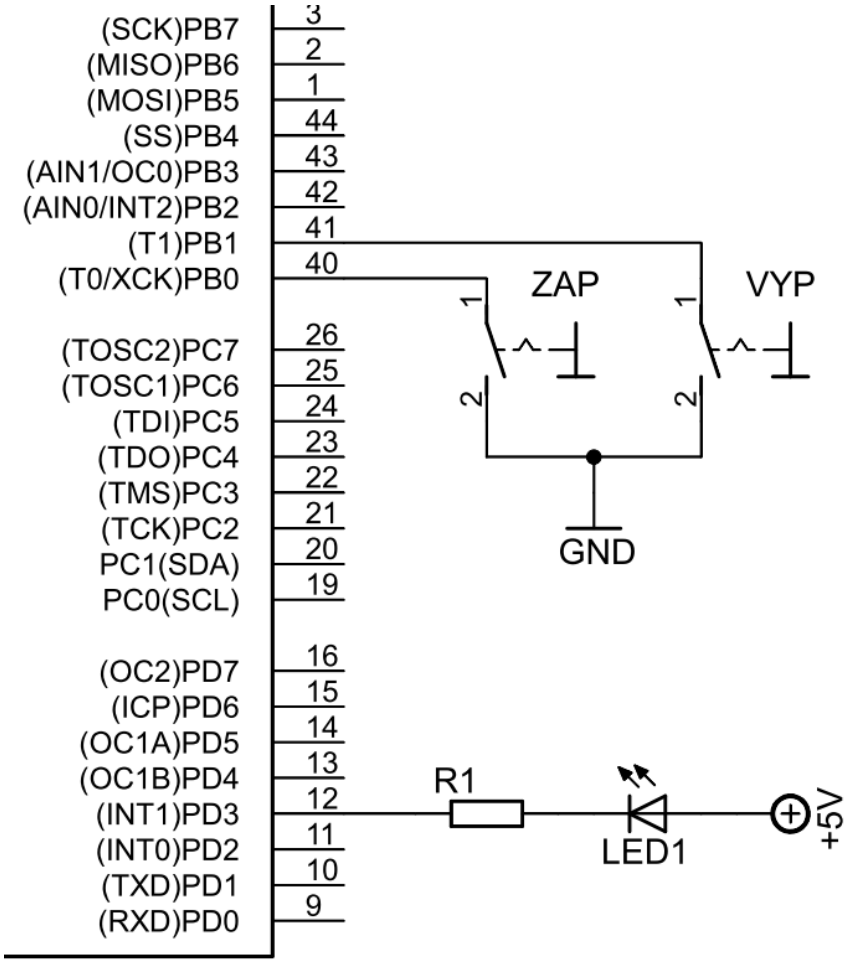
2. Příklad softwarové realizace RS klopného obvodu

Vytvořte program, který realizuje RS klopný obvod, který je ovládán pomocí dvojice tlačítek připojených na port B (PB.0 … ZAP, PB.1 … VYP). Výstup je na portu D na pinu PD.3 a je k němu připojena signalizační LED dioda. Viz schéma zapojení.

Program vytvořte:

a) s využitím instrukcí pro porovnání a podmíněných skoků
b) s využitím instrukcí přeskoků

3. Schéma zapojení



4. Možné řešení

a) S využitím instrukcí pro porovnání a podmíněných skoků

```
.include        "m32def.inc"

.cseg
.org            0


Ini:            ldi     R16,0b00000000      ; port B jako vstupy
                out     DDRB,R16

                ldi     R16,0b00000011      ; PULL-up na PB.0 a PB.1
                out     PORTB,R16

                ldi     R16,0b00001000      ; port D.3 jako vystup
                out     DDRD,R16


Start:          in      R16,PINB            ; nacte port a vymaskuje 2 tlactka
                andi    R16,0b00000011

                cpi     R16,0b01            ; otestuje vstupy na vypnuti
                breq    LEDoff

                cpi     R16,0b10            ; otestuje vstupy na zapnuti
                breq    LEDon

                rjmp    Start


LEDon:          cbi     PORTD,3             ; rozsviti LED
                rjmp    Start

LEDoff:         sbi     PORTD,3             ; zhasne LED
                rjmp    Start
```

b) S využitím instrukcí přeskoků

```
.include        "m32def.inc"

.cseg
.org            0


Ini:            ldi     R16,0b00000000      ; port B jako vstupy
                out     DDRB,R16

                sbi     PORTB,0             ; PULL-up na PB.0 a PB.1
                sbi     PORTB,1

                sbi     DDRD,3              ; port D.3 jako vystup

Start:          sbis    PINB,1              ; otestuje vstupy na vypnuti (pokud ne, preskoci nasl. instr.)
                rjmp    LEDoff

                sbis    PINB,0              ; otestuje vstupy na zapnuti (pokud ne, preskoci nasl. instr.)
                rjmp    LEDon

                rjmp    Start


LEDon:          cbi     PORTD,3             ; rozsviti LED
                rjmp    Start

LEDoff:         sbi     PORTD,3             ; zhasne LED
                rjmp    Start
```

5. Úkol

Modifikujte předchozí program tak, aby po stisku tlačítka ZAP začala LED na výstupu blikat s periodou 800ms a po stisknutí tlačítka VYP se zhasnula. Připojení tlačítek a LED je stejné. Můžete použít libovolnou konstrukci programu a samozřejmě podprogram se zpožďovací rutinou delay.inc.

6. Možné řešení úkolu

```
.include        "m32def.inc"

.cseg
.org            0


Ini:            ldi     R16,high(ramend)        ; inicializace zásobníku
                out     SPH,R16
                ldi     R16,low(ramend)
                out     SPL,R16

                ldi     R16,0b00000000          ; port B jako vstupy
                out     DDRB,R16

                sbi     PORTB,0                 ; PULL-up na PB.0 a PB.1
                sbi     PORTB,1

                sbi     DDRD,3                  ; port D.3 jako vystup

Start:          sbi     PORTD,3                 ; zhasni LED
                sbic    PINB,0                  ; testuje zapnuti
                rjmp    Start

Blikej:         cbi     PORTD,3                 ; rozsviti LED
                rcall   DEL400

                sbis    PINB,1                  ; testuje vypnuti
                rjmp    Start

                sbi     PORTD,3                 ; zhasni LED
                rcall   DEL400

                sbis    PINB,1                  ; testuje vypnuti
                rjmp    Start

                rjmp    Blikej


.include        "h:\prs\delay.inc"
```

Digitální učební materiál

Programování ATmega32 – Ošetření zákmitů tlačítka

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_12

Anotace: Způsoby ošetření zákmitů tlačítka, jednoduchý čítač impůlsů

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

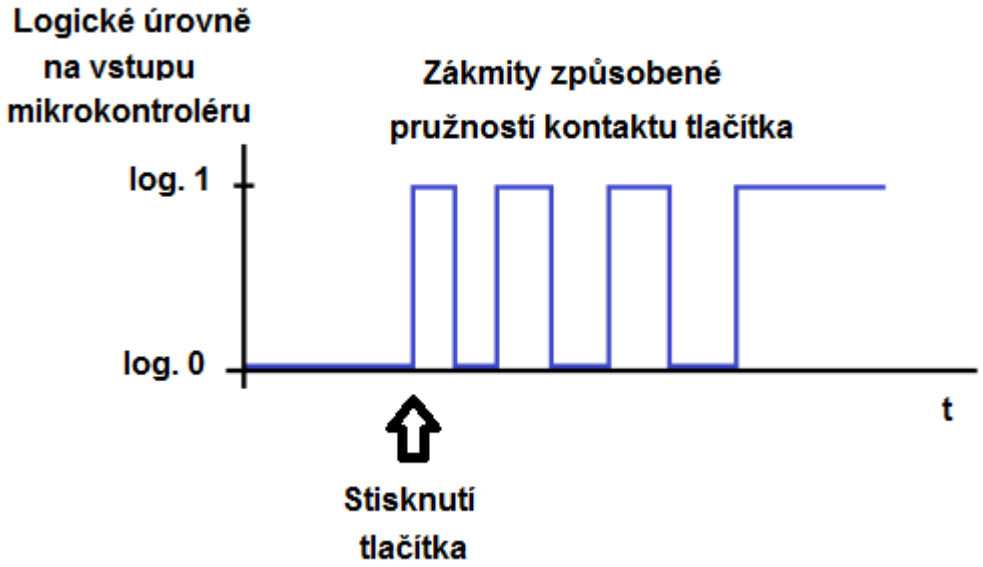
Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.atmel.com
- The Learning Pit Dot Com [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.thelearningpit.com

Programování ATmega32 – Ošetření zákmitů tlačítka

1. Co jsou to zákmity tlačítka?

Zákmity tlačítka rozumíme rychlé změny mezi log. 0 a log. 1 při stisknutí nebo uvolnění tlačítka připojeného ke vstupu mikrokontroléru. Díky pružnosti kontaktů dojde několika falešným impulzům způsobených odskoky od kontaktu. I přesto, že je doba, po kterou vznikají impulsy velice krátká (řádově jednotky až desítky milisekund) a okem nezaznamenatelná, mikrokontrolér je schopný tyto impulzy bez problému zachytit. Tyto falešné impulzy však můžou výrazně ovlivnit chování obvodu.



Představme si, že máme tlačítko, pomocí něhož chceme každým stiskem zvýšit hodnotu na displeji o 1. I přesto, že stiskneme tlačítko pouze jednou krátce, mikrokontrolér zaznamená všechny zákmity, jako potenciální stisky tlačítka a tedy zvýší hodnotu na displeji nejen o jedna, ale o více podle počtu falešných zákmitů.

To by se nám například při nastavování digitálních hodin moc nelíbilo. Je tedy zapotřebí zajistit filtrování těchto zákmitů.

2. Způsoby ošetření zákmitů tlačítka

Ošetření zákmitů je možné řešit buď hardwarově, nebo softwarově. My se zaměříme na softwarové řešení.

Při softwarovém řešení se používají dva základní principy:

a) Opakované čtení vstupu se zpožděním

Tato metoda je založena na jednoduchém principu. Zaznamená-li mikrokontrolér stisknutí tlačítka, počká po dobu odeznění zákmitů a čte vstup znovu. Pokud je vyhodnocen opět stisk tlačítka, provede se patřičná akce, pokud ne, nedělá se nic. Doba čekání je stanovena zkusmo dle typu tlačítka, obvykle 1 – 10 ms.

b) Několikanásobné čtení vstupu

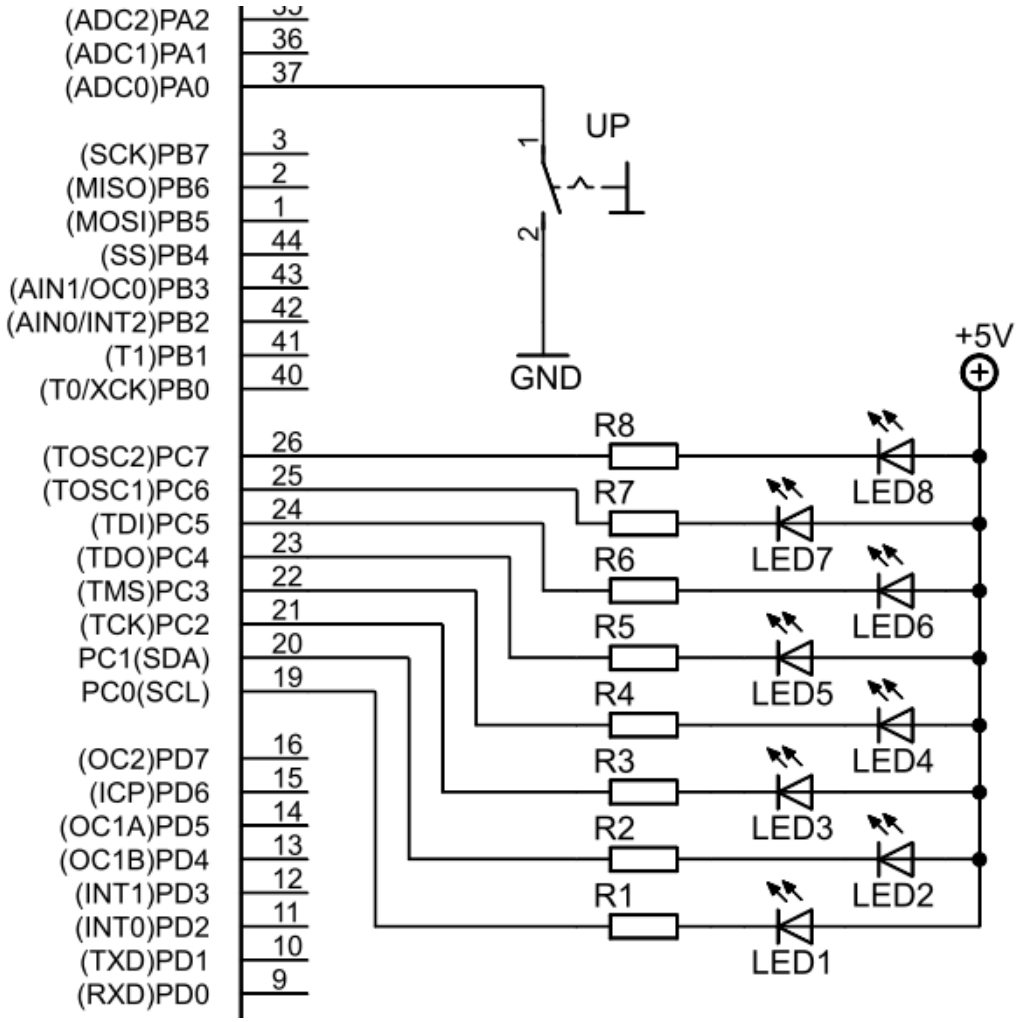
Při tomto způsobu ošetření zákmitů je vstup s tlačítkem čten opakovaně v kratších intervalech. Korektní stisk tlačítka je vyhodnocen, pokud je minimálně 10 – 20 po sobě přečtených hodnot stejných (např. log. 1). Pokud ne, je zřejmé, že došlo k zákmitům a čtení se musí opakovat.

Tato metoda je složitější na programování, ale zato je spolehlivější a rychlejší. S výhodou lze pro vyhodnocování použít časovač, který periodicky filtruje zákmity na vstupech a uživatel již pracuje s hodnotami filtrovanými.

Pozn.: Ne vždy je nutné ošetření zákmitu provádět. V některých aplikacích se například zákmity vůbec neprojeví (např.: RS klopný obvod, …).

3. Vzorový příklad

Vytvořte program, který po stisku tlačítka připojeného na port PA.0 inkrementuje registr R5. Obsah registru zobrazujte pomocí 8-mi LED diod připojených k portu C. Ošetřete zákmity tlačítka způsobem opožděného čten vstupu.

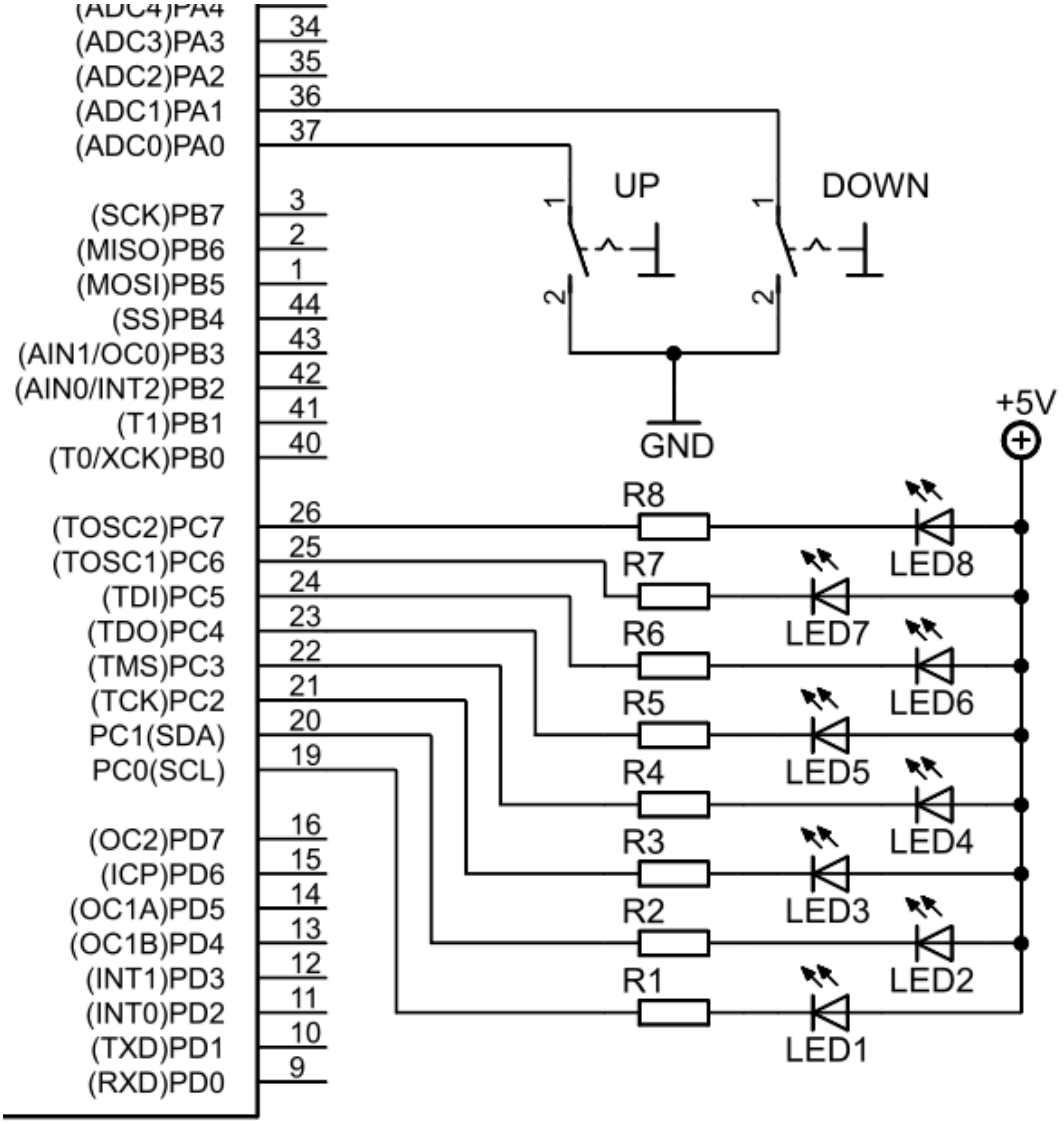


4. Možné řešení

```
.include  "m32def.inc"                  ; prirazeni jmen registru

.cseg
.org      $0000                         ; zacatek pameti FLASH


main:
ini:      ldi     R16,LOW(RAMEND)       ; definice zasobniku
          out     SPL,R16
          ldi     R16,HIGH(RAMEND)
          out     SPH,R16


inic:     ldi     R16,0b11111111        ; pocatecni nastaveni ... PC -> vystup vse v 1
          out     DDRC,R16
          out     PORTC,R16
          ldi     R16,0b00000000        ; pocatecni nastaveni ... PA.0 -> vstup
          out     DDRA,R16
          ldi     R16,0b00000001        ; pul-up zapnuty
          out     PORTA,R16

          clr     R5                    ; vynuluj reg5


start:
          sbic    PINA,0                ; jestlize je PA.0 log.0 (stisknuto TL), tak nasledujici instrukci preskoc
          rjmp    start                 ; jinak skoc zpet na start

          rcall   DEL1ms                ; zpozdi cteni tlacitka kvuli zakmitum 5ms
          rcall   DEL1ms
          rcall   DEL1ms
          rcall   DEL1ms
          rcall   DEL1ms

          sbic    PINA,0                ; znovu precti, pokud opravdu stisknuto, tak preskoc
          rjmp    start                 ; jinak skoc zpet na start

zvys:     inc     R5                    ; zvys R5 o 1
          mov     R16,R5                ; uloz R5 do pomocneho reg R16
          com     R16                   ; zneguj obsah, aby LED svitily spravne
          out     PORTC,R16             ; presun R16 na port C

cykl:     sbis    PINA,0                ; cekej na uvolneni tlacitka
          rjmp    cykl                  ; pokud drzim, tak cykl

          rcall   DEL1ms                ; zpozdi cteni tlacitka kvuli zakmitum 5ms
          rcall   DEL1ms
          rcall   DEL1ms
          rcall   DEL1ms
          rcall   DEL1ms

          sbis    PINA,0                ; opravdu bylo tlacitko uvolneno?
          rjmp    cykl                  ; pokud ne, tak cykl
          rjmp    start                 ; jinak zpet na start


.include  "h:\prs\delay.inc"
```

5. Úkol

Vytvořte program, který při stisku tlačítka Tlup připojeného k portu PA.0 zvýší hodnotu registru R5 o 1 a pomocí tlačítka TLdown připojeného k portu PA.1 sníží hodnotu registru R5 o 1. Hodnotu z registru R5 zobrazujte na portu C pomocí 8-mi připojených LED diod podobně jako v předchozím příkladu.



6. Možné řešení úkolu

```
.include  "m32def.inc"                    ; prirazeni jmen registru

.cseg
.org      $0000                           ; zacatek pameti FLASH


main:
ini:      ldi       R16,LOW(RAMEND)       ; definice zasobniku
          out       SPL,R16
          ldi       R16,HIGH(RAMEND)
          out       SPH,R16


inic:     ldi       R16,0b11111111        ; pocatecni nastaveni ... PC -> vystup
          out       DDRC,R16
          out       PORTC,R17             ; vsechny LED zhasnuty
          ldi       R16,0b00000000        ; pocatecni nastaveni ... PA.0, PA.1 -> vstup
          out       DDRA,R16
          ldi       R16,0b00000011        ; pul-up zapnuty
          out       PORTA,R16

          clr       R5                    ; nastavi R5 na 0
          mov       R16,R5
          com       R16
          out       PORTC,R16
;-----------------------------------------------------

start:    sbis      PINA,0                ; jestlize je PA.0 log.0 (stisknuto TL)
          rjmp      prictix               ; tak skoc na pricti
          sbis      PINA,1                ; jestlize je PA.1 log.0 (stisknuto TL)
          rjmp      odectix               ; tak skoc na odecti

          rjmp      start                 ; pokud neni stisknuto nic, tak skoc na start

prictix:  rcall     DEL1ms                ; zpozdeni pro opetovne cteni tlacitek
          rcall     DEL1ms
          rcall     DEL1ms
          rcall     DEL1ms
          rcall     DEL1ms

          sbis      PINA,0                ; kdyz je opravdu stisknuto, tak pricti, jinak zpet
          rjmp      pricti
          rjmp      start

odectix:  rcall     DEL1ms                ; zpozdeni pro opetovne cteni tlacitek
          rcall     DEL1ms
          rcall     DEL1ms
          rcall     DEL1ms
          rcall     DEL1ms

          sbis      PINA,1                ; kdyz je opravdu stisknuto, tak odecti, jinak zpet
          rjmp      odecti
          rjmp      start

;-----------------------------------------------------
pricti:   inc       R5                    ; zvys R5 o 1
          mov       R16,R5
          com       R16                   ; zneguj R16
          out       PORTC,R16             ; posli na Port C
```

```
            rjmp    uvolTL                  ; skok na uvolneni tlacitka
;-----------------------------------------------------

odecti:     dec     R5                      ; sniz R5 o 1
            mov     R16,R5
            com     R16                     ; zneguj R16
            out     PORTC,R16               ; posli na Port C

            rjmp    uvolTL                  ; skok na uvolneni tlacitka
;-----------------------------------------------------

uvolTL:     in      R16,PINA                ; cekej na uvolneni tlacitek
            andi    R16,0b00000011          ; maskovani
            cpi     R16,0b00000011          ; testuje uvolneni obou tlacitek soucasne
            brne    uvolTL

            rcall   DEL1ms                  ; zpozdeni pro opetovne cteni tlacitek
            rcall   DEL1ms
            rcall   DEL1ms
            rcall   DEL1ms
            rcall   DEL1ms

            in      R16,PINA                ; znovu cte uvolneni tlacitek
            andi    R16,0b00000011          ; maskovani
            cpi     R16,0b00000011          ; testuje uvolneni obou tlacitek soucasne
            brne    uvolTL

            rjmp    start                                   ; zpet na start

;-----------------------------------------------------

.include            "h:\prs\delay.inc"
```

Digitální učební materiál

Programování ATmega32 – Praktické cvičení - porty

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_13

Anotace: Samostatná práce, souhrnné opakování znalostí z obsluhy vstupů a výstupů

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

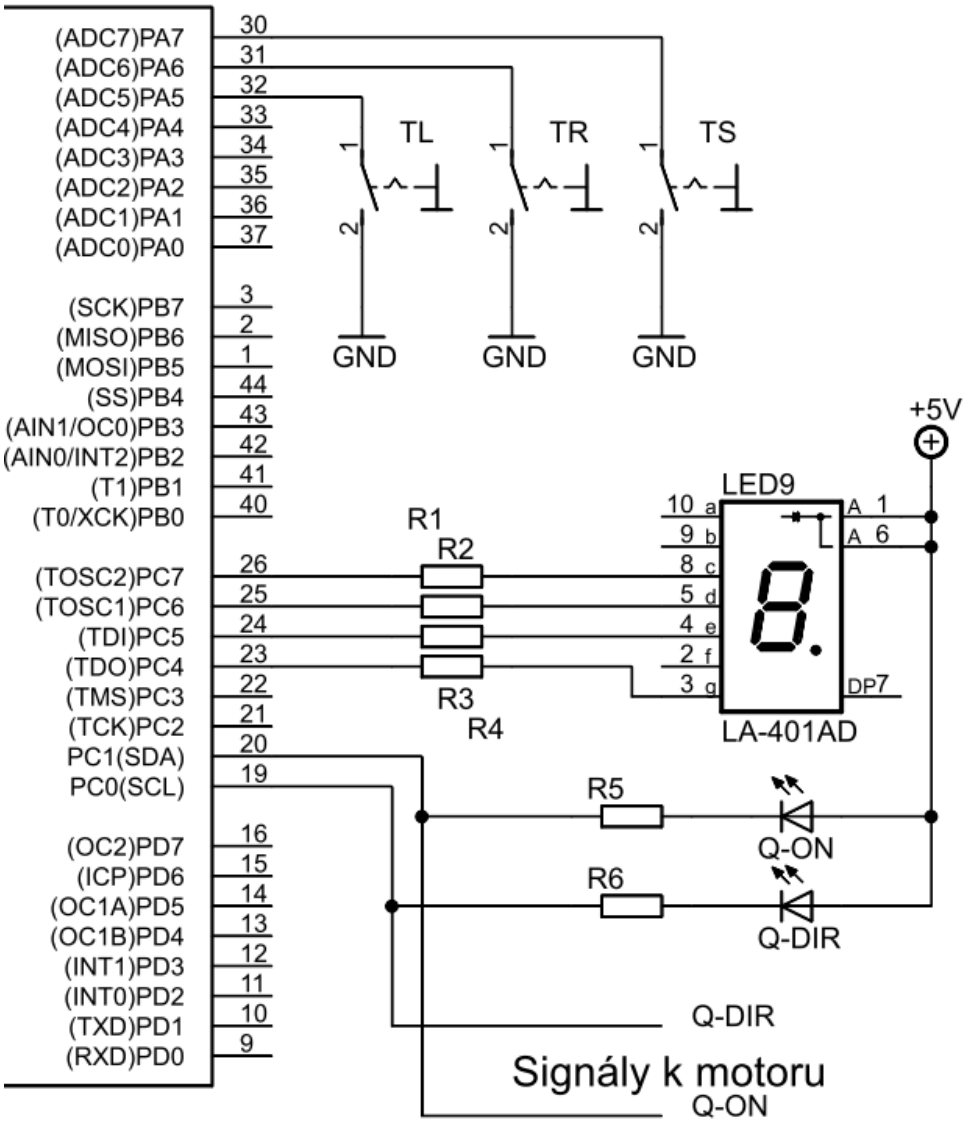
- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.atmel.com
- The Learning Pit Dot Com [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.thelearningpit.com

Programování ATmega32 – Praktické cvičení - porty

1. Zadání

Vytvořte program pro řízení dopravníku s obousměrným provozem. Pohyb dopravníku zajišťuje stejnosměrný elektromotor, který je spouštěn z mikrokontroléru dvěma signály Q_{ON} (log. 0 … motor stojí, log. 1 … motor běží) a Q_{DIR} (log. 0 … otáčení vlevo, log. 1 … otáčení vpravo). K mikrokontroléru jsou připojena tři tlačítka, kterými je ovládáno zařízení (TL … tlačítko pro zapnutí směru vlevo, TR … tlačítko pro zapnutí směru otáčení vpravo a TS … tlačítko pro zastavení motoru). Zapojení výstupů a tlačítek je znázorněno ve schématu zapojení níže. Dále je k mikroprocesoru připojen sedmisegmentový displej, na kterém je zobrazován pohyb motoru pomocí čtyř segmentů (c, d, e, g).

2. Schéma zapojení



3. Hodnocení úkolu

Minimální požadavky pro získání daného počtu bodů

a) Ovládání výstupů Q_{ON} a Q_{DIR} pomocí tlačítek TL, TR a TS.	**30 bodů**
b) Vizualizace běhu motoru pomocí sedmisegmentového zobrazovače.	**40 bodů**
c) Celková funkčnost	**20 bodů**
d) Přehlednost programu, komentáře	**10 bodů**

Známka odpovídající bodové úspěšnosti

100 – 90	bodů	…	**Výborný**
89 – 70	bodů	…	**Chvalitebný**
69 - 50	bodů	…	**Dobrý**
49 - 30	bodů	…	**Dostatečný**
29 - 0	bodů	…	**Nedostatečný**

Doba vypracování je 2 x 45 minut.

4. Možné řešení

```
.include   "m32def.inc"

.cseg
.org                0

;-------------------------------------------------------------------------
Ini:                ; inicializace zásobníku
                    ldi         R16,high(ramend)
                    out         SPH,R16
                    ldi         R16,low(ramend)
                    out         SPL,R16

                    ; inicializace vstupu a vystupu
                                                        ; portA
                    ldi         R16,0x00
                    out         DDRA,R16
                    ldi         R16,0b11100000
                    out         PORTA,R16
                                                        ; portC
                    ldi         R16,0b11110011
                    out         DDRC,R16
                    ldi         R16,0xff
                    out         PORTC,R16

                    ; inicializace 7seg. dipsleje
                    clr         R20                     ; pocitadlo
                    rcall       Zobraz                  ; zobrazeni symbolu na 7seg. displ.

;-------------------------------------------------------------------------
Start:              ; hlavni program
                    ; testovani vstupu

                    cbi         PORTC,0                 ; Qon = 0 ... zastaveno
                    cbi         PORTC,1                 ; Qdir = 0 ... otaceni vlevo

                    sbis        PINA,6                  ; vpravo
                    rjmp        Tvpravo
                    sbis        PINA,5                  ; vlevo
                    rjmp        Tvlevo

                    rjmp        Start                   ; cyklus

;-------------------------------------------------------------------------
Tvpravo:            ; otacenivpravo

                    sbis        PINA,7                  ; kontrola stop
                    rjmp        Start

                    inc         R20                     ; pocitadlo 0 - 3
                    andi        R20,0x03                ; maska modulo4

                    rcall       Zobraz                  ; zobrazeni symbolu na 7seg. displ.

                    sbi         PORTC,0                 ; Qon = 1 ... zapnuto
                    sbi         PORTC,1                 ; Qdir = 1          ... otaceni vpravo

                    rcall       DEL100                  ; zpozdeni 100 ms

                    rjmp        Tvpravo                 ; cyklus
```

```
Tvlevo:         ; otaceni vlevo

                sbis    PINA,7          ; kontrola stop
                rjmp    Start

                dec     R20             ; pocitadlo 0 - 3
                andi    R20,0x03        ; maska modulo4

                rcall   Zobraz          ; zobrazeni symbolu na 7seg. displ.

                sbi     PORTC,0         ; Qon = 1 ... zapnuto
                cbi     PORTC,1         ; Qdir = 0          ... otaceni vlevo

                rcall   DEL100          ; zpozdeni 100 ms

                rjmp    Tvlevo          ; cyklus

;-------------------------------------------------------------------------
Zobraz:         ; zobrazeni na 7seg – otaceni

                ldi     ZH,high(2*Tab)  ; nacte pocatecni adresu tabulky se smboly
                ldi     ZL,low(2*Tab)

                clr     R21             ; pricteni poradi polozky v tabulce
                add     ZL,R20
                adc     ZH,R21

                lpm     R16,Z           ; nacetni symbolu z Tab

                in      R17,PORTC       ; maskovani a ochrana na portu C
                andi    R17,0b00001111
                or      R17,R16
                out     PORTC,R17

                ret                     ; návrat z podprogramu

;-------------------------------------------------------------------------
                ; tabulka symbolu na displeji
Tab:            .db     0b01110000,0b10110000,0b11010000,0b11100000

;-------------------------------------------------------------------------
                ; zpozdovaci podprogram
.include   "h:\prs\delay.inc"
```

Digitální učební materiál

Programování ATmega32 – Inteligentní LCD displej

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_14

Anotace: Obsluha inteligentního LCD displeje s řadičem HD44780 – základní informace

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.atmel.com
- Ovládání znakových LCD s řadičem HD44780 [online]. 2014 [cit. 2014-02-17]. Dostupné z: http://elektronika.kvalitne.cz/ATMEL/necoteorie/LCDmatice.html

Programování ATmega32 – Inteligentní LCD displej

1. Inteligentní LCD displej s řadičem HD44780

Pro zobrazování informací, je možné použít nejrůznější typy displejů. Jedním z velice často používaných je znakový LCD displej s integrovaným řadičem HD44780. Displeje s tímto řadičem se vyrábějí v různých velikostech od jednořádkových s 8mi znaky na řádek až po čtyřřádkové s 40-ti znaky na řádek a také v různých barevných provedeních, s a bez podsvícení a s různým umístěním připojovacího konektoru.

Velkou výhodou těchto displejů je jejich snadná implementace do zařízení díky jednoduchému rozhraní a jejich cenová dostupnost.

2. Standardní zapojení vývodů

Vývod	Název	Funkce
1	Vss	GND
2	Vcc	napájení +5V
3	Vee, V0	nastavení kontrastu
4	RS	volba mezi 0 - instrukce, 1 - data
5	R/W	volba mezi 0 - zápis, 1 - čtení
6	E	hodinový vstup
7	DB0	data 0
8	DB1	data 1
9	DB2	data 2
10	DB3	data 3
11	DB4	data 4
12	DB5	data 5
13	DB6	data 6
14	DB7	data 7
15	LED+	anoda podsvětlení
16	LED-	katoda podsvětlení

Komunikace s displejem probíhá buď po plných 8mi datových vývodů, nebo je možné komunikovat pomocí 4 datových vývodů (DB4 – DB7). Nižší datové bity jsou poté uzemněny. Tím ušetříme několik vstupů mikrokontroléru. Dalším vývodem, který není nutné při komunikaci využívat je vstup R/W. Ve většině aplikací vystačíme pouze se zápisem dat do displeje, a tudíž tento vývod můžeme také uzemnit. Celkový počet komunikačních vodičů tak bude 6 (4x datové, E, RS).

Aby displej pracoval, je nutné ho napájet. Pro napájení slouží vývody Vss -> GND, Vcc -> +5V a Vo -> vstup pro řízení kontrastu displeje (napětí pro tento vstup je většinou odvozeno od napájecího pomocí trimru zapojeného jako napěťový dělič (viz schéma zapojení).

Podsvícení u displejů nemusí být vždy vyvedeno a zapojeno. Vyrábí se displeje jak s podsvícením, tak bez něho. Pokud displej podsvícení nemá, je vidět pouze za přítomnosti okolního světla.

3. Řízení displeje – instrukce

Význam instrukce	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Délka
smaže disp. a nastaví kurzor na začátek	0	0	0	0	0	0	0	0	0	1	1,64 ms
nastaví kurzor na začátek	0	0	0	0	0	0	0	0	1	x	1,64 ms
smer posuvu kurzoru I/D (0=vlevo, 1=vpravo), posuv textu S (0=ne, 1=ano)	0	0	0	0	0	0	0	1	I/D	S	40 us
D - zapne displej, C - zapne kurzor, B - zapne blikání kurzoru	0	0	0	0	0	0	1	D	C	B	40 us
1x posune (S/C=0 kurzor, S/C=1 text) smerem (R/L=0 vlevo, R/L=1 vpravo)	0	0	0	0	0	1	S/C	R/L	x	x	40 us
inicializace: DL=0 4-bit, DL=1 8-bit mód N=0 jednořádkový, N=1 dvouřádkový disp. F=0 font 5x8, F=1 font 5x10	0	0	0	0	1	DL	N	F	x	x	40 us
přepnutí na zápis do CGRAM	0	0	0	1	adresa v CGRAM						40 us
přepnutí na zápis do DDRAM	0	0	1	adresa v DDRAM							40 us
čtení příznaku BF (BF=0 příjem povolen, BF=1 řadič zaneprázdněn), čtení adresy v DDRAM	0	1	BF	adresa v DDRAM							0 us
zápis dat do CGRAM nebo DDRAM	1	0	data								40 us
čtení dat z CGRAM nebo DDRAM	1	1	data								40 us

Po připojení napájecího napětí je nutné provést inicializaci displeje, při které se mimo jiné nastaví, zda bude komunikace probíhat 4 bitově, nebo 8mi bitově. Doporučuje se použít sekvenci příkazů danou výrobcem displeje. Další instrukce jsou patrné z výše uvedené tabulky.

Za zmínku stojí, že displej má dvě paměti DDRAM a CGRAM. Do paměti DDRAM jsou ukládány znaky, které chceme na displeji zobrazit. Jedná se o standardní znaky odpovídající svým kódem ASCII tabulce, tedy první část. Pokud bychom potřebovali zobrazovat speciální znaky (například češtinu), je displej vybaven pamětí CGRAM, do které je možné

předdefinovat a uložit až 8 uživatelských znaků. Znaky se poté zobrazují v DDRAM pomocí kódů 0 – 7.

U dvou a více řádkových displejů, jsou počáteční adresy začátků řádků různé. Proto je nutné věnovat pozornost katalogovým listům jednotlivých výrobců.

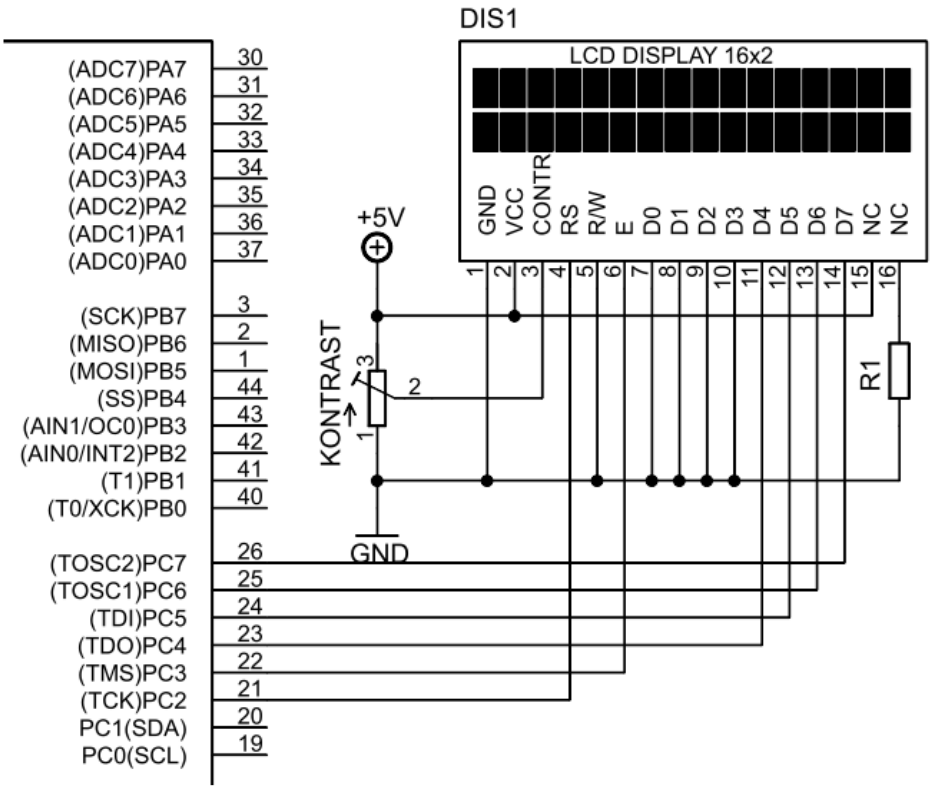
Typické rozložení adres v DDRAM pro displej 2x16 znaků:

1. řádek	00h – 0fh
2. řádek	40h – 4fh

Typické rozložení adres v DDRAM pro displej 4x20 znaků:

1. řádek	00h – 13h
2. řádek	40h – 53h
3. řádek	14h – 27h
4. řádek	54h – 67h

4. Schéma připojení displeje k ATmega32



5. Úkol

Vyhledejte na internetu, nebo v dostupné literatuře přesný postup při inicializaci LCD displeje 2x16 znaků se čtyřbitovou komunikací a sestavte program pro inicializaci displeje.

6. Možné řešení úkolu – inicializace LCD displeje s 4bitovou komunikací

```
LCDini:
                ; nastaveni komunikace
                ldi     R16,0b00110000
                rcall   LCDout                  ; odesli do LCD
                rcall   DEL5ms                  ; zpozdeni 5ms

                ldi     R16,0b00110000
                rcall   LCDout                  ; odesli do LCD
                rcall   DEL1ms                  ; zpozdeni 1ms

                ldi     R16,0b00110000
                rcall   LCDout                  ; odesli do LCD
                rcall   DEL1ms                  ; zpozdeni 1ms

                ldi     R16,0b00100000
                rcall   LCDout                  ; odesli do LCD
                rcall   DEL1ms                  ; zpozdeni 1ms

                ;nastaveni komunikace DL=0, N=1
                 ldi    R16,0b00101000
                rcall   LCDset                  ; odesli do LCD

                ; smaz displej
                ldi     R16,0b00000001
                rcall   LCDset                  ; odesli do LCD
                rcall   DEL1ms
                rcall   DEL1ms

                ; vypni displej
                ldi     R16,0b00001000
                rcall   LCDset                  ; odesli do LCD

                ; zapni, smaz displej
                ldi     R16,0b00001100
                rcall   LCDset                  ; odesli do LCD

                ; vstupni rezim standart
                ldi     R16,0b00000110
                rcall   LCDset                  ; odesli do LCD

                ret
```

Digitální učební materiál

Programování ATmega32 – Obsluha LCD displeje

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_15

Anotace: Obsluha inteligentního LCD displeje s řadičem HD44780 – knihovna pro obsluhu

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

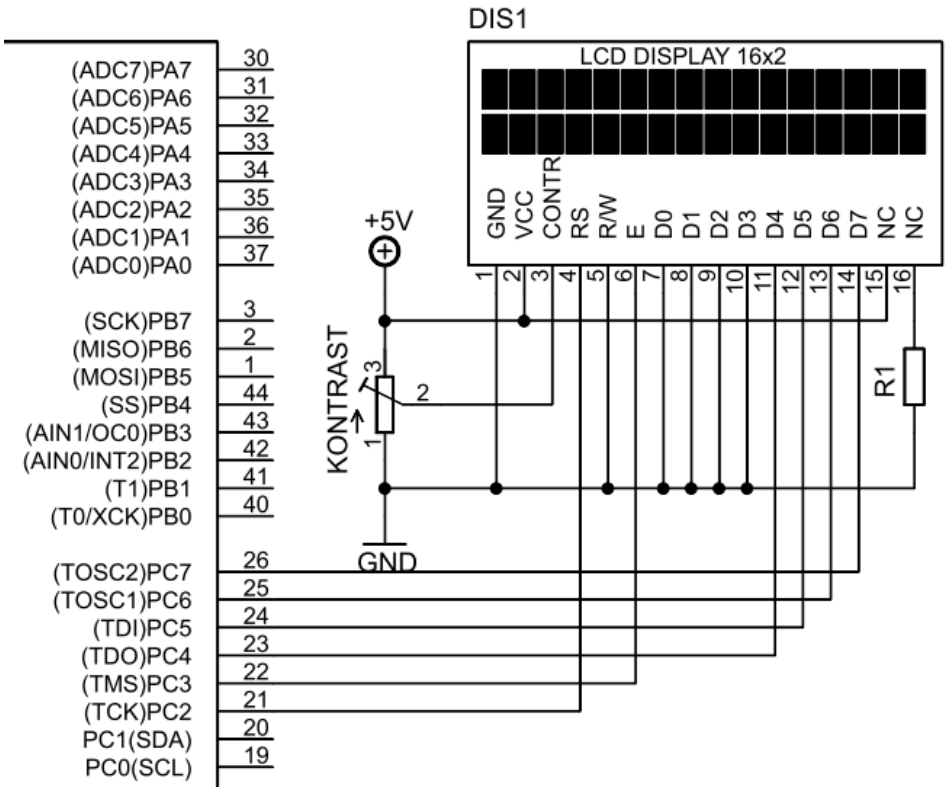
- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.atmel.com
- Ovládání znakových LCD s řadičem HD44780 [online]. 2014 [cit. 2014-02-17]. Dostupné z: http://elektronika.kvalitne.cz/ATMEL/necoteorie/LCDmatice.html

Programování ATmega32 – obsluha LCD displeje

1. Knihovna pro LCD displej

Komunikace s LCD displejem není nijak složitá, ale je vhodné si vytvořit knihovnu pro obsluhu základních funkcí, jako zápis znaku na displej, smazání displeje, inicializace displeje a další. Knihovna, kterou budeme používat, obsahuje soubor nejdůležitějších podprogramů pro komunikaci a je uložena v souboru s názvem lcd.inc.

Je připravena pro komunikaci s LCD displejem prostřednictvím 4 datových vodičů a 2 řídicích (E, RS). Připojení displeje k mikrokontroléru je naznačeno na obrázku níže.



2. Podprogramy obsažené v knihovně a jejich volání

a) inicializace displeje - LCDini

Nejdůležitějším podprogramem je inicializace displeje, která nastaví 4 bitovou komunikaci pro dvouřádkový displej s 16-ti znaky na řádek, vypnutým kurzorem a standardním způsobem výpisu.

Způsob volání:	pouhé volání podprogramu bez parametrů

Př.: rcall LCDini ; inicializace displeje

b) zapsání dat (znaku) na displej – LCDdata

Tento podporgram vyšle data (znak), který je reprezentován ASCII kódem a je uložen v registru R16 do displeje a zobrazí ho na aktuální pozici kurzoru.

Způsob volání:	volání podprogramu s parametrem -> v R16 je ASCII kód zobrazovaného znaku

```
Př.:    ldi     R16,‘A‘
        rcall   LCDdata          ; zobrazí znak A
```

c) zapsání příkazu řadiči – LCDset

Podprogram vyšle příkaz pro displej (řadič), který je uložen v registru R16. Může to být například zapnutí a vypnutí kurzoru, způsob vypisování textu, …

Způsob volání:	volání podprogramu s parametrem -> v R16 kód příkazu

```
Př.:    ldi     R16,0x0f    ; zapnutí displeje a kurzoru včetně blikání
        rcall   LCDset      ; pošle příkaz do řadiče displeje
```

d) smazání celého displeje – LCDclr

Podprogram pro smazání celého displeje a nastavení kurzoru na první pozici prvního řádku.

Způsob volání:	volání podprogramu bez parametruu

```
Př.:    rcall   LCDclr      ; smaže displej
```

e) nastavení kurzoru na první pozici 1. řádku, 2. řádku – LCDrow1, LCDrow2

Zavoláním podprogramu se nastaví kurzor na první pozici prvního, nebo druhého řádku na displeji.

Způsob volání:	volání podprogramu bez parametru

```
Př.:    rcall   LCDrow1     ; nastavení kurzoru na 1. řádek, 1. pozici
        …
        rcall   LCDrow2     ; nastavení kurzoru na 2. řádek, 1. pozici
```

f) nastavení kurzoru na zadanou pozici 1. řádku, 2. řádku – LCDrow1p, LCDrow2p

Zavoláním podprogramu se nastaví kurzor na první pozici prvního, nebo druhého řádku na displeji.

Způsob volání:	volání podprogramu s parametrem -> v R16 pozice na displeji od leva

```
Př.:    ldi     R16,5
        rcall   LCDrow1p      ; nastavení kurzoru na 1. řádek, 5. pozici
        …
        ldi     R16,12
        rcall   LCDrow2p      ; nastavení kurzoru na 2. řádek, 12. pozici
```

g) vypsání BCD čísla na displeji – LCDbcd

Vypíše na displeji BCD číslo uložené v registru R16.

Způsob volání:	volání podprogramu s parametrem -> v R16 BCD číslo

```
Př.:    ldi     R16,0x3B
        rcall   LCDbcd        ; vypíše na displeji číslo 3B
```

h) vypsání řetězce znaků na displeji – LCDstring

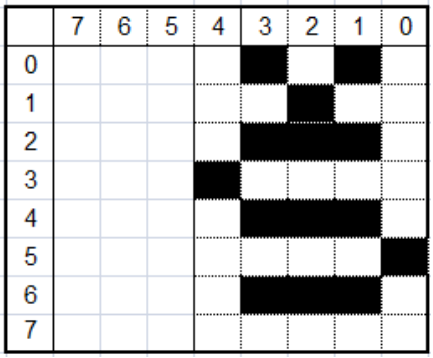
Vypíše na displeji řetězec znaků uložený v paměti programu. Počáteční adresa textu se nastaví do registru Z, do registru R17 se vloží počet znaků v řetězci a zavolá se daný podprogram.

Způsob volání:	volání podprogramu s parametry -> v R17 počet znaků v řetězci v ZH:ZL počáteční adresa řetězce

```
Př.:    ldi     ZH,high(Text*2)
        ldi     ZL,low(Text*2)
        ldi     R17,10
        rcall   LCDstring     ; vypíše na displeji text SPS Hronov
        …

Text:   .db     " SPS Hronov"
```

i) definice uživatelského znaku – LCDznak

Nadefinuje v paměti CGRAM uživatelský znak. Znak se skládá z 8mi bajtů, kde každý bajt reprezentuje zobrazení části znaku od shora dolů (viz. Obrázek níže). V displeji je možné nadefinovat celkem 8 uživatelských znaků uložených v DDRAM na adresách 0 – 7.



Způsob volání:	volání podprogramu s parametry -> v R16 adresa znaku v DDRAM v ZH:ZL adresa tabulky ve FLASH

```
Př.:    ldi     ZH,high(Znak*2)     ; adresa ve FLASH
        ldi     ZL,low(Znak*2)
        ldi     R16,0               ; adresa znaku v DDRAM
        rcall   LCDznak             ; nadefinuje znak š
        …


        ldi     ZH,high(Text*2)     ; ukázka zobrazení s definovaným znakem
        ldi     ZL,low(Text*2)
        ldi     R17,10
        rcall   LCDstring           ; vypíše na displeji text Spš Hronov
        …

Text:   .db     “Sp“,0,“ Hronov“
Znak:   .db     10,4,14,16,14,1,14,0        ; nadefinování š
```

3. Výpis knihovny lcd.inc

```
;***********************************************************
;* rutiny pro rizeni LCD dipleje - vyuziva registry R16 a R17
;* pripojeni:        RS - PC.2
;*                   E  - PC.3
;*                   D4 - PC.4
;*                   D5 - PC.5
;*                   D6 - PC.6
;*                   D7 - PC.7
;* LCDini - Inicializace LCD displeje
;* LCDout - posle do displeje horni nibl R16
;* LCDset - zapis prikazu do displeje (prikaz v R16)
;* LCDdata - zapis dat do displeje  (data v R16)
;* LCDclr - smaze cely display
;* LCDrow1 - nastav kurzor na zacatek 1. radku
;* LCDrow2 - nastav kurzor na zacatek 2. radku
;* LCDrow1p - nastav kurzor na pozici na 1. radku (pozice v R16)
;* LCDrow1p - nastav kurzor na pozici na 2. radku (pozice v R16)
;* LCDbcd - vypise BCD cislo ulozene v R16 na LCD
;* LCDznak - nadefinuje znak v CGRAM (v Z zacatek tabulky znaku, v R16 adresa 0-7)
;* LCDstring - vypis retezce znaku na displej (zacatek v Z a v R17 pocet znaku)
;* NTA - prevod BCD na ASCI
;***********************************************************

.equ            RS=2
.equ            E=3
.equ            D4=4
.equ            D5=5
.equ            D6=6
.equ            D7=7
;---------------------------------------------------------------------
LCDini:         ldi     R16,0xfc                ; vystupy
                out     DDRC,R16

                cbi     PORTC,RS                ; vynuluj RS
                cbi     PORTC,E                 ; vynuluj E
                rcall   DEL100                  ; cekej 100 ms

                ; nastaveni komunikace
                ldi     R16,0b00110000
                rcall   LCDout                  ; odesli do LCD
                rcall   DEL5ms                  ; zpozdeni 5ms

                ldi     R16,0b00110000
                rcall   LCDout                  ; odesli do LCD
                rcall   DEL1ms                  ; zpozdeni 1ms

                ldi     R16,0b00110000
                rcall   LCDout                  ; odesli do LCD
                rcall   DEL1ms                  ; zpozdeni 1ms

                ldi     R16,0b00100000
                rcall   LCDout                  ; odesli do LCD
                rcall   DEL1ms                  ; zpozdeni 1ms

                ;nastaveni komunikace DL=0, N=1
                 ldi    R16,0b00101000
                rcall   LCDset                  ; odesli do LCD

                ; smaz displej
                ldi     R16,0b00000001
```

```
                rcall   LCDset                  ; odesli do LCD
                rcall   DEL1ms
                rcall   DEL1ms

                ; vypni displej
                ldi     R16,0b00001000
                rcall   LCDset                  ; odesli do LCD

                ; zapni, smaz displej
                ldi     R16,0b00001100
                rcall   LCDset                  ; odesli do LCD

                ; vstupni rezim standart
                ldi     R16,0b00000110
                rcall   LCDset                  ; odesli do LCD

                ret
;-------------------------------------------------------------------
LCDout:         ; vstup R16 -> horni nibl
                cbi     PORTC,D7                ; vynulovani bitu D7-D4
                cbi     PORTC,D6
                cbi     PORTC,D5
                cbi     PORTC,D4

                sbrc    R16,7                   ; nasatvi Data D7-D4 podle R16(High)
                sbi     PORTC,D7
                sbrc    R16,6
                sbi     PORTC,D6
                sbrc    R16,5
                sbi     PORTC,D5
                sbrc    R16,4
                sbi     PORTC,D4

                sbi     PORTC,E                 ; impulz E
                nop
                nop
                nop
                cbi     PORTC,E

                ret
;-------------------------------------------------------------------
LCDdata:         ; posli data na LCD
                sbi     PORTC,RS                ; nastav RS
                rjmp    lcd1

LCDset:         ; posli prikaz do LCD
                cbi     PORTC,RS                 ; vynuluj RS

lcd1:           push    R16                      ; uloz R16
                rcall   LCDout
                pop     R16                      ; obnov R16
                swap    R16                      ; prohod nibly
                rcall    LCDout

                rcall   DEL40us                 ; zpozdeni 40us

                ret
;-------------------------------------------------------------------
LCDclr:         ldi     R16,0b00000001          ; smaz cely displej
                rcall   LCDset
                rcall   DEL2ms                  ; zpozdeni 2ms

                ret
```

```
;-------------------------------------------------------------------
LCDrow1:        clr     R16
LCDrow1p:       ldi     R17,0x80
                add      R16,R17                ; secti bazovou adresu s pozici
                rjmp     LCDset                 ; skoc na zapis prikazu

LCDrow2:        clr     R16
LCDrow2p:       ldi     R17,0xc0
                add     R16,R17                 ; secti bazovou adresu s pozici
                rjmp     LCDset                 ; skoc na zapis prikazu

;-------------------------------------------------------------------
LCDbcd:         push    R16                    ; vypise bcd cislo z R16 na LCD
                swap    R16
                andi    R16,0x0f
                rcall   NTA
                rcall   LCDdata
                POP     R16
                andi    R16,0x0f
                rcall   NTA
                rcall   LCDdata

                ret

;-------------------------------------------------------------------
LCDznak:        ldi     R17,8                  ; nastavi uzivatelsky znak v CGRAM
                mul     R16,R17                ; vysledek R1:R0

                ldi     R16,0x40               ; nastaveni adresy v pameti
                add     R16,R0                 ; generatoru znaku

                rcall   LCDset

                ldi     R17,8                  ; pocitadlo

lcdzn1:         lpm     R16,Z+                 ; nacte z tabulky cast znaku

                rcall   LCDdata
                dec     R17
                brne    lcdzn1

                ret

;-------------------------------------------------------------------
NTA:            ldi     R17,246                ; prevede cislo v BCD na ASCI pro
                add     R16,R17                ; zobrazeni na displej
                brcs    nta1
                ldi     R17,58
                add      R16,R17
                ret
nta1:           ldi     R17,65
                add      R16,R17
                ret
;-------------------------------------------------------------------
LCDstring:      lpm     R16,Z+                 ; zobrazi text z adresy v Z a pocet znaku v R17
                rcall   LCDdata
                dec     R17
                brne    LCDstring

                ret
;-------------------------------------------------------------------
```

Digitální učební materiál

Programování ATmega32 – Analogové vstupy

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_16

Anotace: Analogové vstupy, A/D převodník, obsluha analogových vstupů ATmega32

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2013 [cit. 2014-01-02]. Dostupné z: www.atmel.com

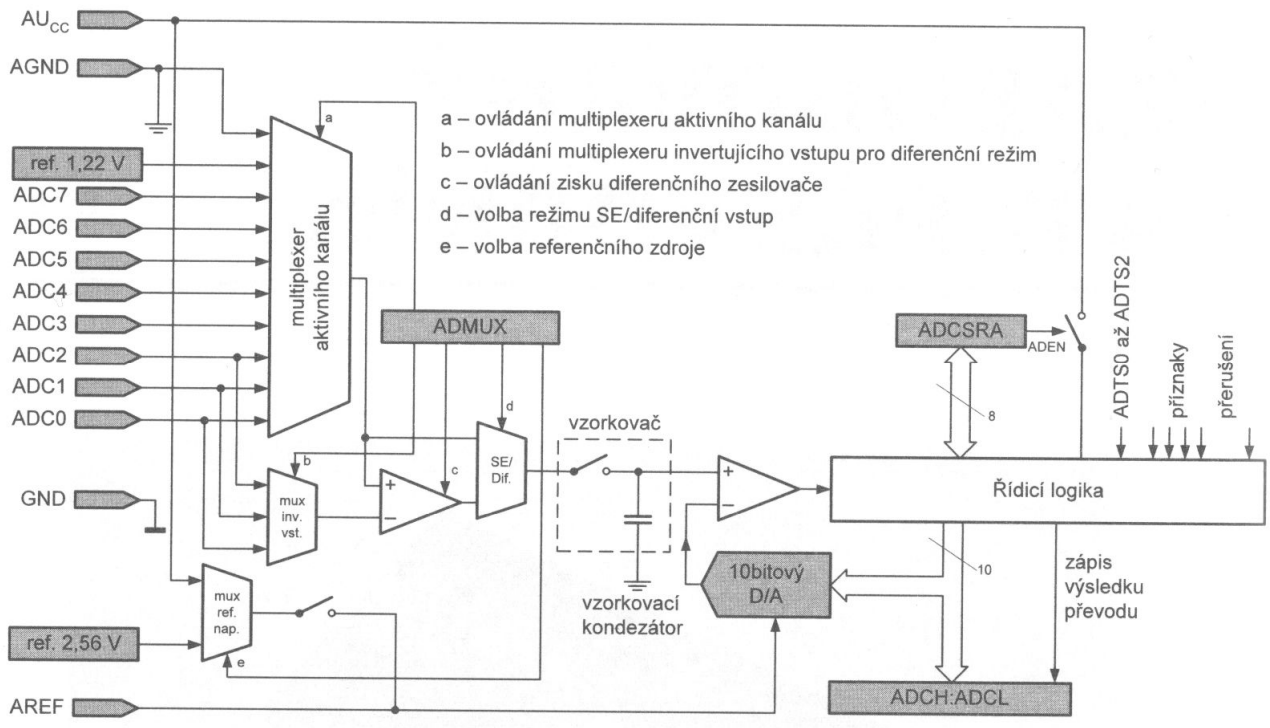
Programování ATmega32 – analogové vstupy

1. Analogové vstupy – základní parametry

- 10-ti bitové rozlišení
- nelinearita 0,5 LSB
- absolutní chyba 2 LSB
- doba převodu 13 až 260 us
- rychlost převodu až 15 ksps (15000 vzorků za sekundu)
- 8 vstupních kanálů SE (multiplexovaných)
- 7 diferenčních kanálů
- 2 diferenční kanály s volitelným ziskem (1x, 10x, 200x)
- integrovaná reference 2,56V
- režimy - jednoduchý převod, automatické spouštění, volný běh
- přerušení po dokončení převodu
- možnost spouštět A/D převod pomocí dalších přerušení
- potlačení šumu v režimu IDLE

Mikrokontrolér ATmega32 obsahuje A/D převodník s postupnou aproximací. Výsledek převodu je ve formátu 10-ti bitového čísla, které je uloženo v registrech ***ADCH:ADCL*** a je možné ho zarovnat doleva nebo doprava podle nastavení bitu ***ADLAR*** v reistru ***ADMUX*** (viz dále).

2. Blokové schéma



3. Řízení A/D převodu

Výsledek převodu:

Při čtení výsledku z registrů ***ADCH:ADCL*** je nutné číst nejprve obsah registru ***ADCL***, tím se zablokuje případná změna registrů a poté přečíst obsah registru ***ADCH***. Pokud vystačíme s rozlišením pouze 8 bitů, je možné číst samotný registr ***ADCH*** při zarovnání výsledku vlevo.

Volba vstupních kanálů:

Vstupní analogové kanály se volí pomocí bitů **MUX0 - MUX4** v registru **ADMUX**. Všechny vstupy lze konfigurovat jako ***SE*** (měří napětí proti zemi). Některé lze konfigurovat jako ***diferenční***. V tom případě se pak do převodníku přivádí rozdílové napětí dvou vstupů.

Činnost A/D převodníku

Činnost převodníku se povoluje bitem ***ADEN*** v registru ***ADCSRA***

ADEN = log.0 ... činnost A/D zakázána

ADEN = log.1 ... činnost A/D povolena

Spouštění převodu

A/D převod může být spuštěn několika způsoby:

1. ***Jednoduchý převod*** - každý převod je odstartován ***zápisem log. 1*** do bitu ***ADSC*** registru ***ADCSRA***. Po skončení převodu se bit automaticky vynuluje.

2. ***Automatické spouštění přerušením*** - převod může být spuštěn různými zdroji přerušení. Povoluje se nastavením bitu ***ADATE*** v registru ***ADCSRA***, zdroj spuštění se volí pomocí bitů ***ADTS0 - ADTS2*** v registru ***SFIOR***.

3. ***Volný běh*** - volí se podobně jako automatický režim spouštění volbou spouštění příznakem ***ADIF***, který indikuje konec převodu. Následující převod začne okamžitě po dokončení předchozího. První převod se musí odstartovat ručně zápisem ***log. 1*** do bitu ***ADSC*** registru ***ADCSRA***.

Referenční napětí AREF lze zvolit:

- **vnější** referenční napětí přivedené na pin **AREF**
- napětí na vývodu **AUcc**
- vnitřní referenční napětí **2,56V**

Pozn1.: Ve všech případech je nutné připojit na vývod ***AREF*** blokovací kondenzátor.
Pozn2.: Volba referenčního napětí se provádí pomocí bitů **REFS0** a **REFS1** v registru **ADMUX**.

Předdělička a časování převodu

Převodník s postupnou aproximací potřebuje pro dosažení dané přesnosti vstupní hodinový kmitočet v rozsahu ***50-200kHz***. Z tohoto důvodu je na vstupu připojena předdělička, která dělí vstupní hodinový kmitočet na kmitočet vhodný pro A/D převodník. Dělící poměr se volí pomocí bitů **ADPS0 - ADPS2** v registru **ADCSRA**. Normální převod trvá 13 hodinových cyklů, rozšířený pak 25 hodinových cyklů (při prvním zapnutí A/D převodníku ADEN=1). Vzorkování zabere 1,5 hodinového cyklu pro normální převod a 13,5 hodinového cyklu pro rozšířený převod.

4. Řídicí registry

A/D převodník je ovládán čtveřicí registrů ***ADMUX, ADCSRA, ADCH:ADCL a SFIOR***.

Registr – ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	**REFS0**	**ADLAR**	**MUX4**	**MUX3**	**MUX2**	**MUX1**	**MUX0**	**ADMUX**
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	

REFS1, REFS0 výběr referenčního zdroje

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

ADLAR zarovnání výsledku (log. 0 ... vpravo, log. 1 ... vlevo)

MUX4 - MUX0 výběr kanálu

MUX4..0	Single Ended Input
00000	ADC0
00001	ADC1
00010	ADC2
00011	ADC3
00100	ADC4
00101	ADC5
00110	ADC6
00111	ADC7

Registr – ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

ADEN zapnutí A/D převodu (nastavením bitu)
ADSC start převodu (nastavením bitu, po skončení převodu se automaticky nuluje)
ADATE automatické spouštění (nastavením se povoluje)
ADIF příznak přerušení A/D převodníku (bit se nastaví při dokončení převodu, nuluje se HW při vstupu do obslužné rutiny, nebo SW zápisem log.1)
ADIE povolení přerušení (log.1 ... povoleno)
ADPS2 - ADPS0 nastavení předděličky

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Registr – SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADTS2 - ADTS0 volba zdroje pro automatické spouštění

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

Registrový pár – ADCH:ADCL

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	**ADC9**	**ADC8**	**ADCH**
	ADC7	**ADC6**	**ADC5**	**ADC4**	**ADC3**	**ADC2**	**ADC1**	**ADC0**	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	**ADC8**	**ADC7**	**ADC6**	**ADC5**	**ADC4**	**ADC3**	**ADC2**	**ADCH**
	ADC1	**ADC0**	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Digitální učební materiál

Programování ATmega32 – Analogové vstupy, příklady

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_17

Anotace: Příklady obsluhy analogových vstupů, barograf

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

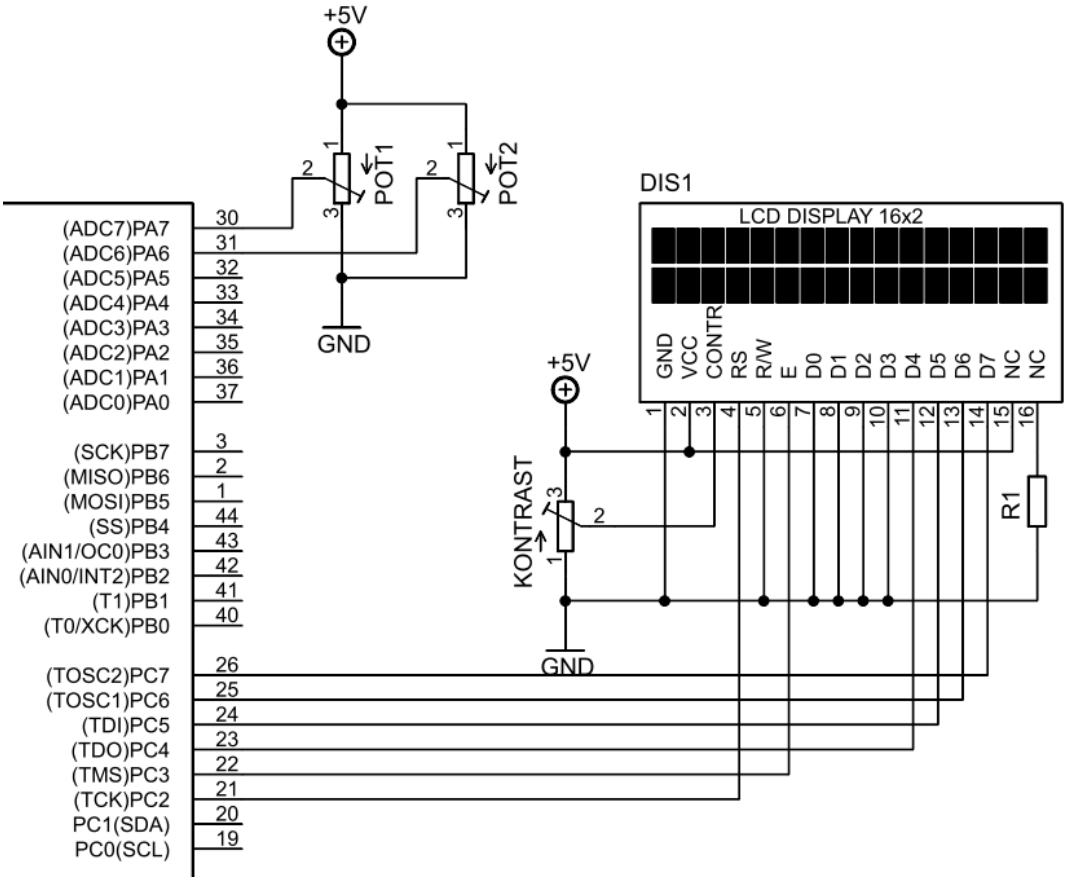
- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2014 [cit. 2014-03-01]. Dostupné z: www.atmel.com

Programování ATmega32 – analogové vstupy, příklady

1. Příklad 1 – Převod analogové hodnoty na 8mi bitové číslo

Sestavte program, který provede převod dvou analogových napětí v rozsahu 0 - 5 V přivedených na vstupy AD6 a AD7 mikrokontroléru ATmega32 na číslo v 8mi bitovém formátu. Výsledné hodnoty budou zobrazeny pomocí LCD displeje v hexadecimálním tvaru.

Schéma zapojení:



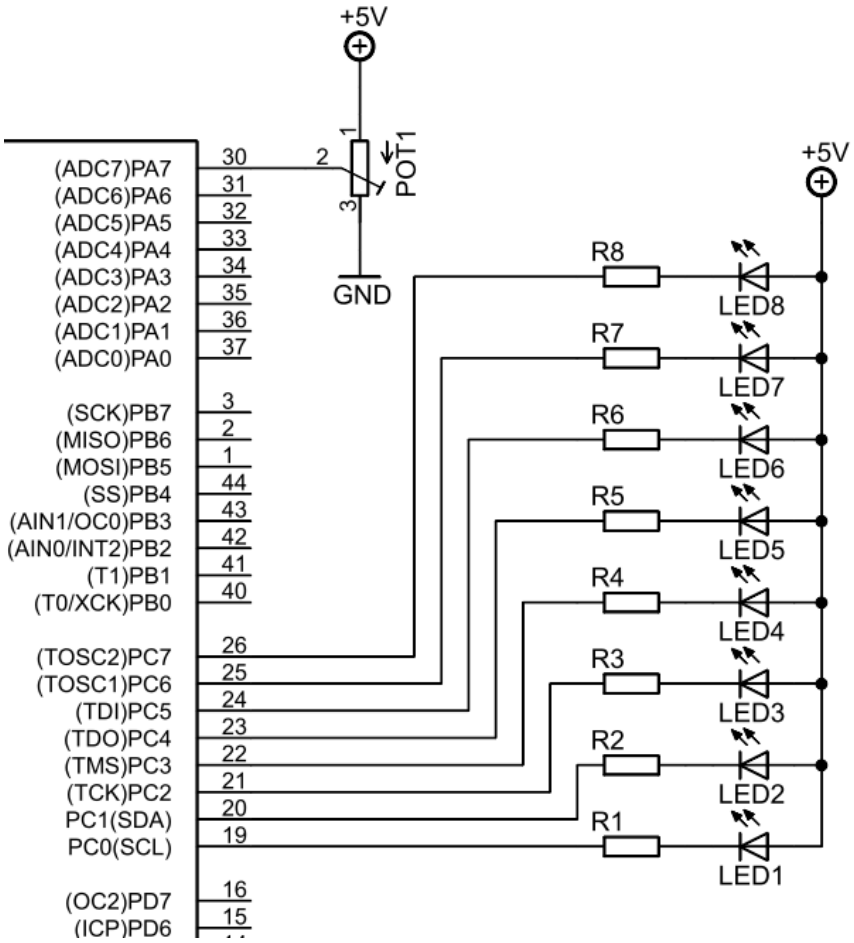
Možné řešení

```
.include  "m32def.inc"

.cseg
.org      $0000                              ; zacatek pameti FLASH

Ini:      ldi     R16,LOW(RAMEND)            ; definice zasobniku
          out     SPL,R16
          ldi     R16,HIGH(RAMEND)
          out     SPH,R16

          rcall   LCDini                     ; inicializace LCD

          ldi     R16,0b10000111             ; nastavime preddelicku 125kHz, jednorazovy prevod
          out     ADCSRA,R16

;-------------------------------------------------------------------------------------------------------------------------------

Start:

; 1. mereni

          rcall   LCDrow1                    ; nastav kurzora na 1. pozici 1. řádku

          ldi     R16,0b00100111             ; nastavi prevod 7 kanal, zarovnani vlevo
          out     ADMUX,R16

          sbi     ADCSRA,ADSC                ; start prevodu
cekej1:   sbic    ADCSRA,ADSC
          rjmp    cekej1                     ; cekej na konec prevodu

          in      R16,ADCH                   ; nacteni zmerene hodnoty do R16
          rcall   LCDbcd                     ; zobrazeni na LCD

; 2. mereni

          rcall   LCDrow2                    ; nastav kurzora na 1. pozici 2. řádku

          ldi     R16,0b00100110             ; nastavi prevod 6 kanal, zarovnani vlevo
          out     ADMUX,R16

          sbi     ADCSRA,ADSC                ; start prevodu
cekej2:   sbic    ADCSRA,ADSC
          rjmp    cekej2                     ; cekej na konec prevodu

          in      R16,ADCH                   ; nacteni zmerene hodnoty do R16
          rcall   LCDbcd                     ; zobrazeni na LCD

          rjmp    start



.include  "h:\prs\delay.inc"
.include  "h:\prs\lcd.inc"
```

2. Příklad 2 – Měření napětí s převodem na sloupec LED – BARGRAF

Sestavte program, který bude měřit napětí v rozsahu 0 - 5 V na vstupu AD7 a zobrazovat jej pomocí 8mi LED diod připojených na port C mikrokontroléru ATmega32. Zobrazení velikosti napětí bude simulovat narůstající sloupec z rozsvícených LED diod (BARGRAF). Při nulovém napětí na vstupu nesvítí žádná LED. Při maximálním napětí na vstupu (5V) svítí všechny LED. Rozdělení komparačních úrovní volte lineárně.

Schéma zapojení:



Možné řešení

```
.include  "m32def.inc"
.cseg
.org      $0000                                 ; zacatek pameti FLASH

Ini:      ldi       R16,LOW(RAMEND)             ; definice zasobniku
          out       SPL,R16
          ldi       R16,HIGH(RAMEND)
          out       SPH,R16

          ldi       R16,0b11111111              ; pocatecni nastaveni ... PC -> vystup
          out       DDRC,R16

          ldi       R16,0b00100111              ; nastavi prevod 7 kanal, zarovnani vlevo
          out       ADMUX,R16
          ldi       R16,0b10000111              ; nastavime preddelicku 125kHz, jednorazovy prevod
          out       ADCSRA,R16
;-------------------------------------------------------------------------------------------------------------------------------
start:    sbi       ADCSRA,ADSC                 ; start prevodu
cekej:    sbic      ADCSRA,ADSC
          rjmp      cekej                       ; cekej na konec prevodu

          in        R19,ADCH                    ; do R19 vložíme převedené číslo

          ldi       R18,0b00000000              ; pokud je hodnota menší než 10h, tak nesvítí nic

          cpi       R19,0x10
          brlo      pryc
          ldi       R18,0b00000001              ; pokud je hodnota mezi 10h a 2fh, tak svítí jedna LED

          cpi       R19,0x30
          brlo      pryc
          ldi       R18,0b00000011              ; pokud je hodnota mezi 30h a 4fh, tak svítí dvě LED

          cpi       R19,0x50
          brlo      pryc
          ldi       R18,0b00000111              ; pokud je hodnota mezi 50h a 6fh, tak svítí tři LED

          cpi       R19,0x70
          brlo      pryc
          ldi       R18,0b00001111              ; pokud je hodnota mezi 70h a 8fh, tak svítí čtyři LED

          cpi       R19,0x90
          brlo      pryc
          ldi       R18,0b00011111              ; pokud je hodnota mezi 90h a afh, tak svítí pět LED

          cpi       R19,0xb0
          brlo      pryc
          ldi       R18,0b00111111              ; pokud je hodnota mezi b0h a cfh, tak svítí šest LED

          cpi       R19,0xd0
          brlo      pryc
          ldi       R18,0b01111111              ; pokud je hodnota mezi d0h a efh, tak svítí sedm LED

          cpi       R19,0xf0
          brlo      pryc
          ldi       R18,0b11111111              ; pokud je hodnota mezi f0h a ffh, tak svítí jedna LED

pryc:     com       R18                         ; negace - LED se rozsviceji log.0
          out       PORTC,R18                   ; zapsani na PORT
          rjmp      start
```

Digitální učební materiál

Programování ATmega32 – Čítač/časovač

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_18

Anotace: Čítač, časovač, druhy čítačů/časovačů, způsoby řízení

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2014 [cit. 2014-03-01]. Dostupné z: www.atmel.com

Programování ATmega32 – Čítač/časovač

1. Čítač/časovač - úvod

Čítače/časovače jsou nedílnou součástí každého mikrokontroléru. Různé typy mikrokontrolérů obsahují minimálně jeden čítač/časovač, výkonnější mikrokontroléry i několik. Mikrokontrolér ATmega32 obsahuje celkem 3 čítače /časovače označené 0, 1 a 2. Čítač/časovač 0 a 2 jsou 8mi bitové, čítač/časovač 1 je 16ti bitový.

Co je to čítač?

Čítač je obvod, který počítá impulzy přicházející z vnějšku do mikrokontroléru. Může sloužit například pro měření kmitočtu vnějšího signálu.

Co je to časovač?

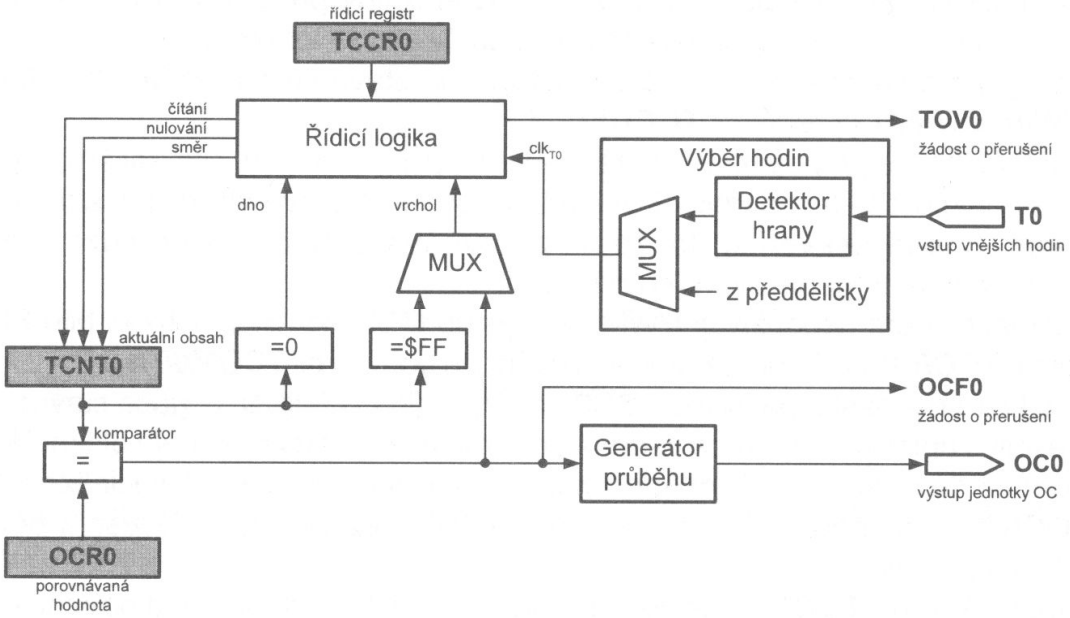
Časovač počítá impulzy, které jsou odvozeny od vnitřního hodinového signálu. Používá se zpravidla pro generování zpoždění, nebo výstupního signálu.

2. Čítač/časovač 0

a) Základní vlastnosti

- 8-mi bitový jednokanálový čítač/časovač pro všeobecné použití
- 10-ti bitová předdělička pro hodinový signál
- možnost generování přerušení při přetečení čítače/časovače a nebo při shodě čítače/časovače s komparačním registrem
- vynulování čítače/časovače při shodě obsahu s komparačním registrem
- Output Compare (výstupní komparátor) – porovnává aktuální stav čítače/časovače se speciálním registrem a v případě, že se obě hodnoty rovnají, tak se provede změna výstupu OC (vynuluje, nastaví, neguje).
- PWM výstup – speciální režim obvodu Output Compare, který na výstupu generuje signál, který je modulován pulzně šířkovou modulací (PWM).

b) Blokové schéma



TCNT0 – registr čítače/časovače (8 bitů), obsahuje aktuální hodnotu čítače/časovače
OCR0 – registr obsahuje hodnotu pro porovnání
TCCR0 – řídicí registr (nastavují se v něm módy (režimy) čítače/časovače
TOV0 – žádost o přerušení při přetečení čítače/časovače
OCF0 – žádost o přerušení při rovnosti obsahu registrů **TCNT0** a **OCR0**
T0 – vstup vnějšího hodinového signálu
OC0 – výstup jednotky Output Compare

Pozn.: Signály **TOV0** a **OCF0** korespondují s maskami přerušení v registru **TIMSK** a s příznaky přerušení v registru **TIFR**.

c) Pracovní režimy

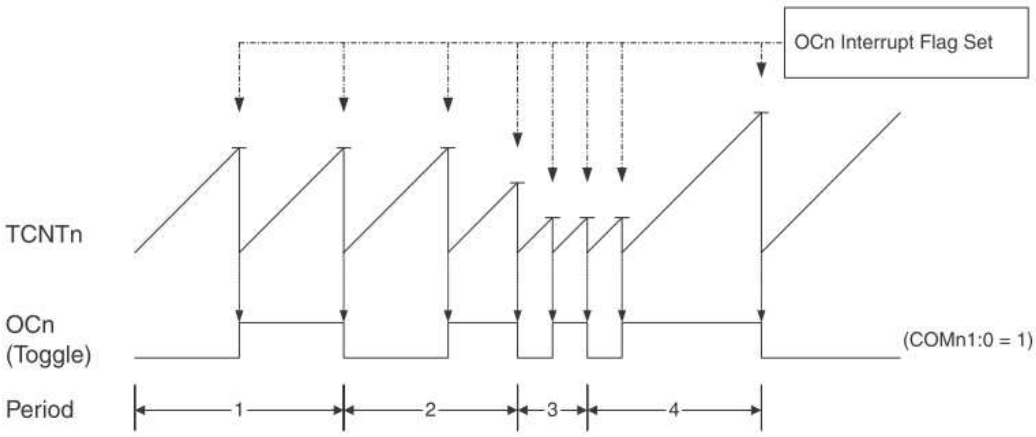
Pracovní režimy se nastavují pomocí patřičných bitů v registru **TCCR0**. Čítač/časovač 0 má **celkem 3 režimy**:

I. Normální režim

S každým příchodem impulzu se obsah registru **TCNT0** zvýší o 1. Při dosažení svého maxima (FFH) čítač přeteče a počítá opět od nuly. V tomto okamžiku je možné vyvolat přerušení signálem **TOV0**.

II. CTC - Clear Timer on Compare Match

V tomto režimu se čítač nuluje, jestliže se obsah registru **TCNT0 = OCR0**. Vhodným naplněním registru **OCR0** je tedy možné zkrátit rozsah čítače. Tento registr pracuje ve funkci modulo. Přerušení je vyvoláno při nulování čítače. Tímto způsobem je možné generovat na výstupu **OC0** signál jehož kmitočet je závislý na obsahu registru **OCR0**.

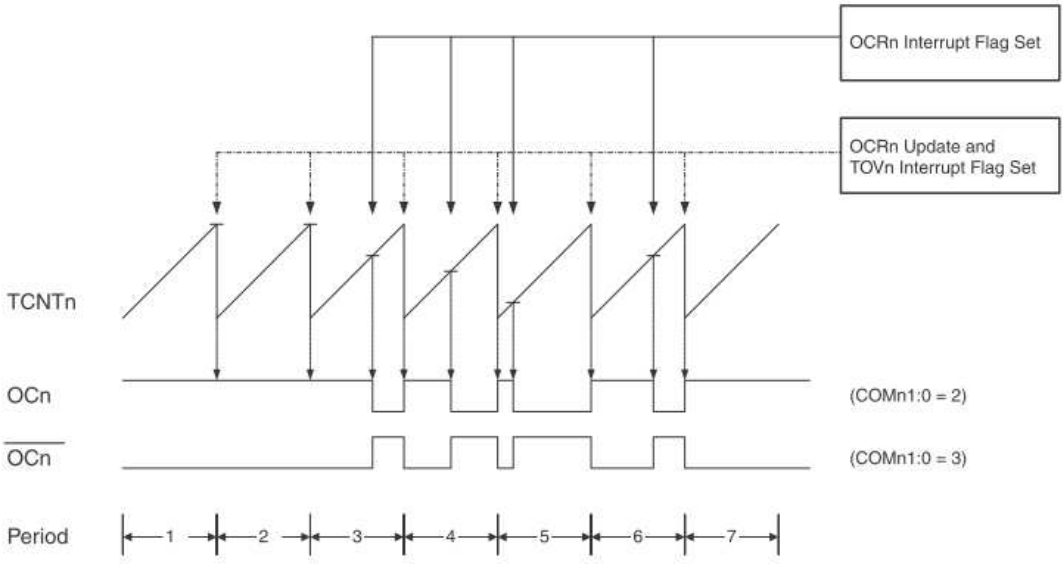


III. PWM režim

Existují dva PWM režimy (rychlý a fázově korigovaný).

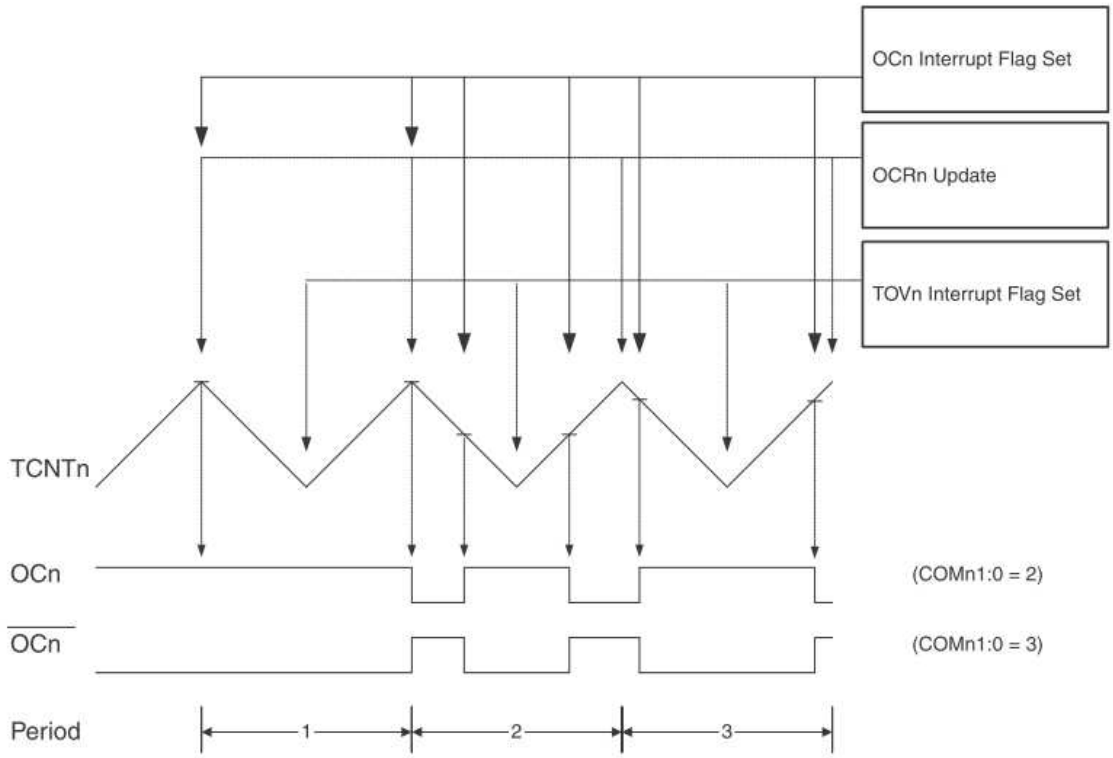
- ***rychlý PWM režim***

Tento režim poskytuje vysokorychlostní generaci PWM signálu na výstupu. Jedná se o tzv. jednofázové řízení. To znamená, že čítač počítá od 0 do maxima (FFH) a poté přeteče do nuly. Pracovní kmitočet je 2x větší než u fázově korigovaného PWM.
Tento režim se používá pro regulaci výkonu, nebo D/A převod.



- ***fázově korigovaný režim PWM***

Režim poskytuje PWM průběh s velkým rozlišením. Používá se tzv. dvojfázová realizace. Čítač čítá opakovaně z nuly do maxima a zpět z maxima do nuly. Toto řízení má sice menší pracovní kmitočet, ale v některých aplikacích má přednost právě díky většímu rozlišení (8 bitů).



d) Registry čítače/časovače 0

Registr TCCR0 - řídicí registr čítače/časovače 0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

FOC0 - vynucení OC výstupu (při nastaveném bitu dojde při shodě TCNT0 = OCR0, ale nesmí být nastaven při PWM režimech)

WGM01, WGM00 - režim generovaného průběhu

WGM01	WGM00	Režim	Vrchol	Aktualizace OCR0	Nastavení TOV0
0	0	normální	$FF	okamžitě	na vrcholu
0	1	fázově korigovaný PWM	$FF	na vrcholu	na dnu
1	0	CTC	OCR0	okamžitě	na vrcholu
1	1	rychlý PWM	$FF	na vrcholu	na vrcholu

COM01, COM00 - režim OC vývodu

- režimy bez PWM

COM01	COM00	Popis
0	0	čítač/časovač 0 odpojen od vývodu OC0
0	1	negace stavu vývodu OC0 při shodě (toggle)
1	0	vynulování vývodu OC0 při shodě (log. 0)
1	1	nastavení vývodu OC0 při shodě (log. 1)

- rychlý režim PWM

COM01	COM00	Popis
0	0	čítač/časovač 0 odpojen od vývodu OC0
0	1	vyhrazeno
1	0	OC0 = 0 po shodě, OC0 = 1 při $FF (neinvertující režim)
1	1	OC0 = 1 po shodě, OC0 = 0 při $FF (invertující režim)

- fázově korigovaný režim PWM

COM01	COM00	Popis
0	0	čítač/časovač 0 odpojen od vývodu OC0
0	1	vyhrazeno
1	0	OC0 = 0 po shodě při čítání nahoru, OC0 = 1 po shodě při čítání dolů (neinvertující režim)
1	1	OC0 = 1 po shodě při čítání nahoru, OC0 = 0 po shodě při čítání dolů (invertující režim)

CS02, CS01, CS00 - výběr hodin

CS02	CS01	CS00	Popis
0	0	0	stop, čítač/časovač 0 je zastaven
0	0	1	$f_{clk\_I/O}$ (hodiny mikrokontroléru)
0	1	0	$f_{clk\_IO}/8$
0	1	1	$f_{clk\_IO}/64$
1	0	0	$f_{clk\_IO}/256$
1	0	1	$f_{clk\_IO}/1024$
1	1	0	sestupná hrana T0
1	1	1	náběžná hrana T0

Registr TIMSK - masky přerušení čítačů/časovačů

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
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	

OCIE0 - povolení přerušení při shodě **TCNT0 = OCR0**
TOIE0 - povolení přerušení při přetečení čítače/časovače

Registr TIFR - příznaky přerušení čítačů/časovačů

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
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	

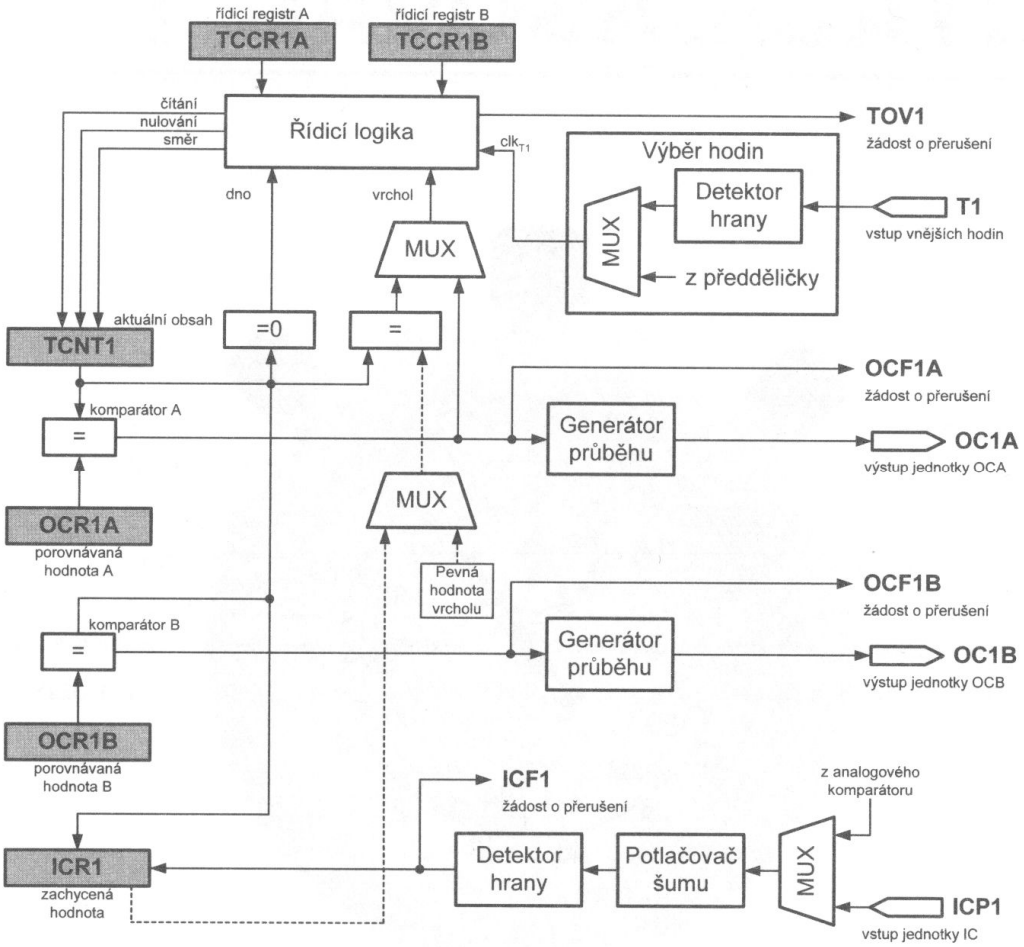
OCF0 - příznak přerušení při shodě **TCNT0 = OCR0** (Nuluje se HW při vstupu do přerušovací rutiny, lze nulovat i SW zápisem log. 1)
TOV0 - příznak přerušení při přetečení čítače/časovače (Nuluje se HW při vstupu do obslužné rutiny, lze nulovat i SW zápisem log. 1)

3. Čítač/časovač 1

a) Základní vlastnosti

- 16-ti bitový dvoukanálový čítač/časovač pro všeobecné použití
- dvě nezávislé jednotky Output Compare
- 10-ti bitová předdělička pro hodinový signál
- CTC režim
- zdokonalené funkce - PWM
- 4 nezávislé zdroje přerušení (TOV1, OCF1A, OCF1B, ICF1)
- Input Capture (záchytný registr) - jedná se o obvod, který umožňuje zachytit stav čítače/časovače při příchodu impulzu na vstup mikrokontroléru ICP

b) Blokové schéma



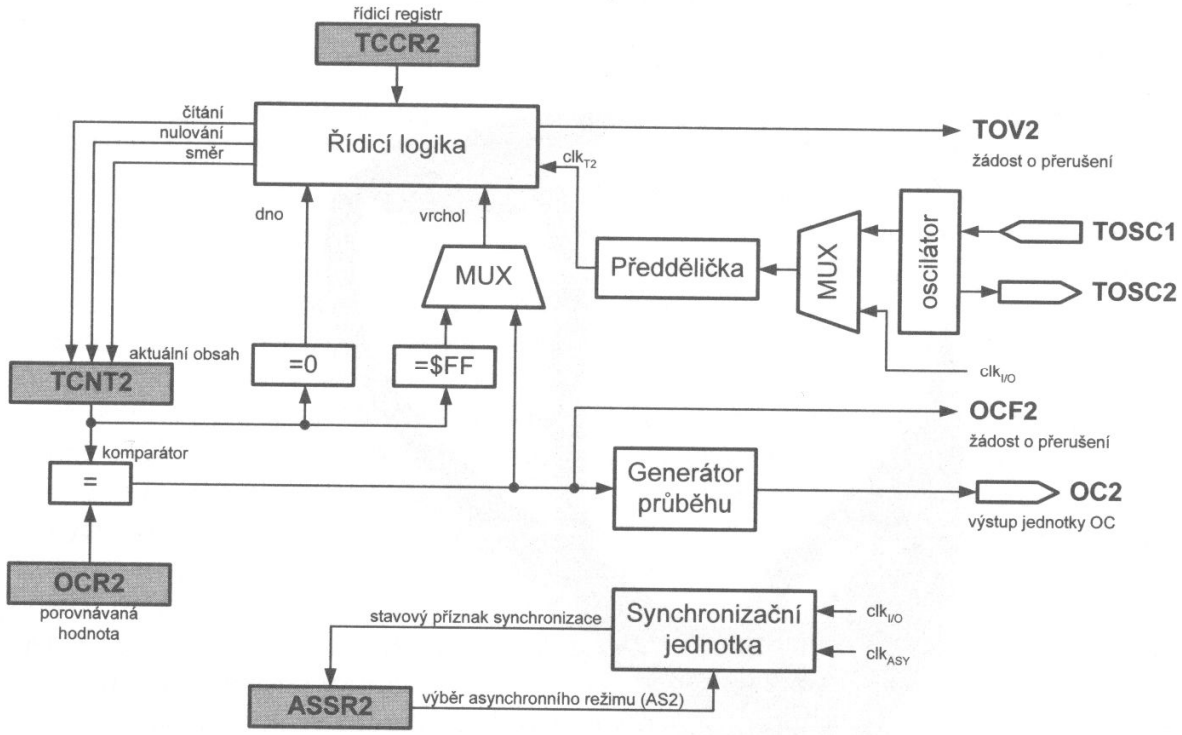
TCNT1 - registr čítače/časovače (16 bitů), obsahuje aktuální hodnotu čítače/časovače
OCR1A, OCR1B - registry pro porovnání
TCCR1A, TCCR1B - řídicí registry čítače/časovače
TOV1 - žádost o přerušení od přetečení čítače/časovače
OCF1A, OCF1B - žádosti o přerušení při rovnosti obsahů reg. **TCNT1** a **OCR1A**, nebo **OCR1B**
T1 - vstup vnějšího hodinového signálu
OC1A, OC1B - výstupy jednotek Output Compare
ICP1 - vstup jednotky Input Capture

4. Čítač/časovač 2

a) Základní vlastnosti

- 8-mi bitový jednokanálový čítač/časovač pro všeobecné použití
- 10-ti bitová předdělička pro hodinový signál
- jako zdroj hodinového kmitočtu lze použít hod. signál mikrokontroléru, nebo hodinkový krystal 32kHz (asynchronní signál)
- může pracovat jako generátor kmitočtu, nebo jako čítač vnějších událostí

b) Blokové schéma



TCNT2 - registr čítače/časovače (8 bitů), obsahuje aktuální hodnotu čítače/časovače
OCR2 - registr pro porovnání
TCCR2 - řídicí registr čítače/časovače
TOV2 - žádost o přerušení od přetečení čítače/časovače
OCF2 – žádost o přerušení při rovnosti obsahu registrů **TCNT2 a OCR2**
OC2 - výstup jednotky Output Compare
TOSC1, TOSC2 - vývody pro připojení hodinového krystalu 32kHz
ASSR2 - řídicí registr pro asynchronní operace

Pozn.: Bližší popis čítačů v katalogovém listu výrobce.

Digitální učební materiál

Programování ATmega32 – Čítač/časovač - programy

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_19

Anotace: Ukázkové programy s využitím čítačů/časovačů, jednoduchý čítač, PWM výstup, …

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2014 [cit. 2014-03-01]. Dostupné z: www.atmel.com

Programování ATmega32 – Čítač/časovač - programy

1. Jednoduchý čítač impulsů

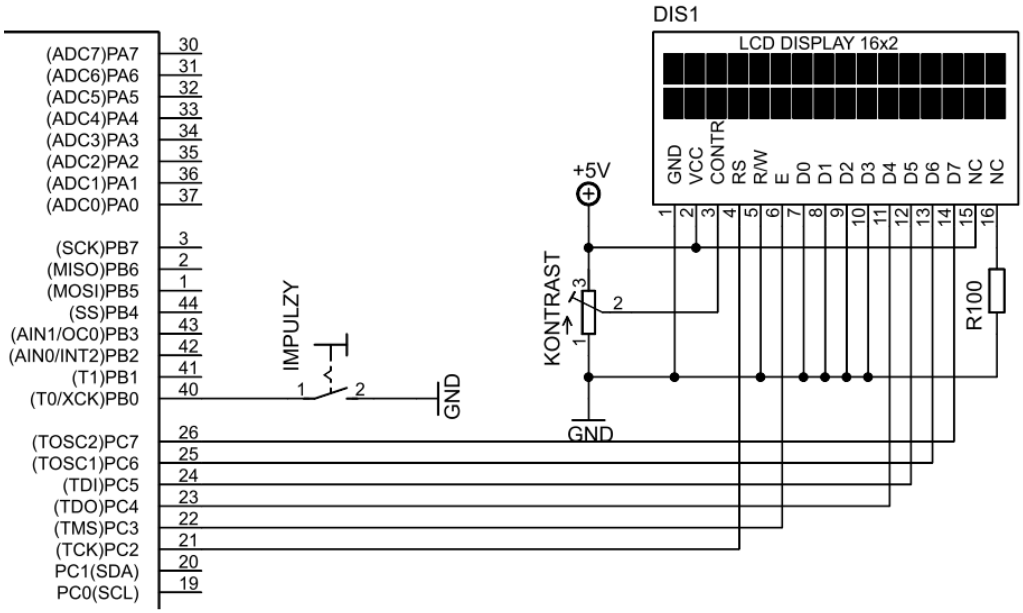
- ***Zadání:***

Vytvořte program, který bude počítat stisky tlačítka připojeného na vstup T0 čítače 0. Aktuální hodnotu čítače průběžně zobrazujte na LCD displeji.

a) Nastavte čítač pro čítání v normálním režimu

b) Nastavte čítač, který bude čítat modulo 10 (režim CTC)

- ***Schéma zapojení:***



- ***Možné řešení:***

a) Normální režim čítače

```
.include        "m32def.inc"
.cseg
.org            0

Ini:            ldi     R16,LOW(RAMEND)         ; inicializace zasobniku
                out     SPL,R16
                ldi     R16,HIGH(RAMEND)
                out     SPH,R16

                ldi     R16,0b00000110
                out     TCCR0,R16               ; inicializace citace, normalni rezim, vstup T0, sestupna hrana

                sbi     PORTB,0                 ; aktivace PULL-UP rezistoru na pinu T0

                rcall   LCDini                  ; inicializace displeje
                rcall   LCDclr                  ; smazani displeje
;-----------------------------------------------------------------------------------------------------------------------------------
Start:          rcall   LCDrow1                 ; nastav pozici kurzoru

                in      R16,TCNT0               ; nacti aktuální hodnotu citace do R16
                rcall   LCDbcd                  ; zobraz aktuální stav citace v BCD tvaru

                rjmp    Start                   ; opakuj
;-----------------------------------------------------------------------------------------------------------------------------------
.include        "delay.inc"
.include        "lcd.inc"
```

b) CTC režim čítače – modulo 10

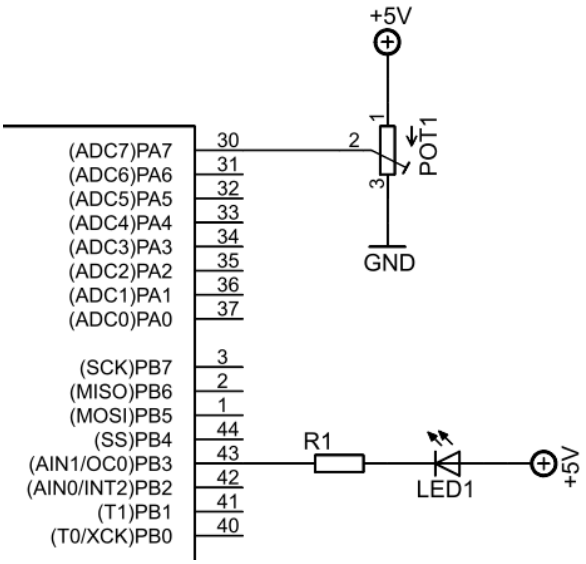
```
.include        "m32def.inc"
.cseg
.org            0

Ini:            ldi     R16,LOW(RAMEND)         ; inicializace zasobniku
                out     SPL,R16
                ldi     R16,HIGH(RAMEND)
                out     SPH,R16

                ldi     R16,0b00001110
                out     TCCR0,R16               ; inicializace citace, CTC rezim, vstup T0, sestupna hrana
                ldi     R16,9
                out     OCR0,R16                ; modulo 10 -> vrchol 9

                sbi     PORTB,0                 ; aktivace PULL-UP rezistoru na pinu T0

                rcall   LCDini                  ; inicializace displeje
                rcall   LCDclr                  ; smazani displeje
;-----------------------------------------------------------------------------------------------------------------------------------
Start:          rcall   LCDrow1                 ; nastav pozici kurzoru

                in      R16,TCNT0               ; nacti aktuální hodnotu citace do R16
                rcall   LCDbcd                  ; zobraz aktuální stav citace v BCD tvaru

                rjmp    Start                   ; opakuj
;-----------------------------------------------------------------------------------------------------------------------------------
.include        "h:\prs\delay.inc"
.include        "h:\prs\lcd.inc"
```

2. Řízení jasu LED diody

- ***Zadání:***

Vytvořte program, který pomocí potenciometru připojeného na vstup AD7 bude řídit jas LED diody pomocí PWM modulace. LED dioda je připojena k vývodu OC0 (PB.3). Při realizaci využijte časovač 0. Vyzkoušejte rozdíly v nastavení různých hodnot předděličky čítače.

- ***Schéma zapojení:***



- ***Možné řešení:***

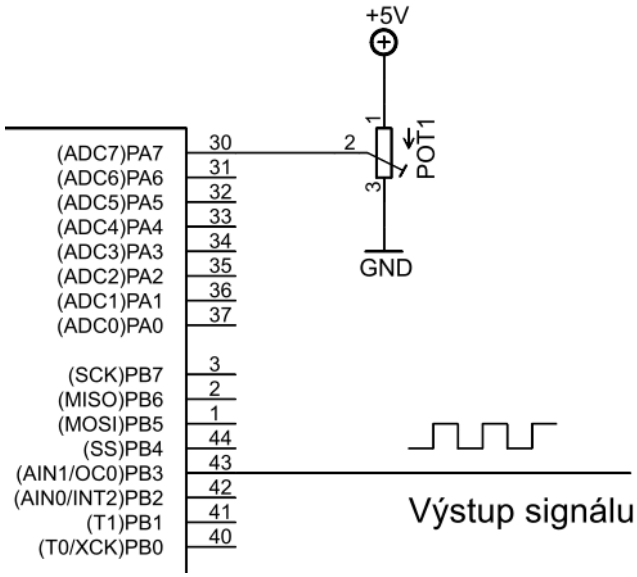
```
.include    "m32def.inc"

.cseg
.org        0                           ; zacatek pameti FLASH

main:
ini:        ldi     R16,LOW(RAMEND)     ; definice zasobniku
            out     SPL,R16
            ldi     R16,HIGH(RAMEND)
            out     SPH,R16

inic:       sbi     DDRB,3              ; port PB3 jako vystup

            ldi     R16,0b00100111      ; nastavi prevod 7 kanal, zarovnani vlevo
            out     ADMUX,R16
            ldi     R16,0b10000111      ; nastavime preddelicku 125kHz, jednorazovy prevod
            out     ADCSRA,R16

            ldi     R16,0b01101001      ; inicializace PWM na PB3, OC neinvertujici, bez preddelicky
            out     TCCR0,R16
;-----------------------------------------------------------------------------------------------------------------------------------------------

start:      sbi     ADCSRA,ADSC         ; start prevodu

cekej:      sbic    ADCSRA,ADSC
            rjmp    cekej               ; cekej na konec prevodu

            in      R19,ADCH            ; nacti prevedenou hodnotu do R19
            out     OCR0,R19            ; hodnotu z R19 zapis do OCR0

            rjmp    start               ; cyklus
```

3. Generátor obdélníkového průběhu s proměnným kmitočtem

- ***Zadání:***

Vytvořte program, který bude generovat obdélníkový průběh na výstupu OC0 mikrokontroléru ATmega32. Generovaný kmitočet bude nastavitelný v určitém rozsahu pomocí napětí z potenciometru POT1 na analogovém vstupu AD7. Při řešení využijte mód CTC čítače/časovače a zjistěte měřením rozsah generátoru pro různé nastavení předděličky. Naměřené hodnoty porovnejte s teoretickými předpoklady.

- ***Schéma zapojení:***



- ***Možné řešení:***

```
.include    "m32def.inc"

.cseg
.org        0                           ; zacatek pameti FLASH

Ini:        ldi     R16,LOW(RAMEND)     ; definice zasobniku
            out     SPL,R16
            ldi     R16,HIGH(RAMEND)
            out     SPH,R16

            sbi     DDRB,3              ; port PB3 jako vystup

            ldi     R16,0b00100111      ; nastavi prevod 7 kanal, zarovnani vlevo
            out     ADMUX,R16
            ldi     R16,0b10000111      ; nastavime preddelicku 125kHz, jednorazovy prevod
            out     ADCSRA,R16

            ldi     R16,0b00011101      ; inicializace PWM, CTC rezim, OC zmena pri preteceni,
            out     TCCR0,R16           ; preddelicka /1024

;-----------------------------------------------------------------------------------------------------------------------
Start:      sbi     ADCSRA,ADSC         ; start prevodu

cekej:      sbic    ADCSRA,ADSC
            rjmp    cekej               ; cekej na konec prevodu

            in      R19,ADCH
            out     OCR0,R19            ; hodnotu z AD prevodu posli do OCR0
            rjmp    Start
```

Digitální učební materiál

Programování ATmega32 – Systém přerušení

Šablona: III/2 - Inovace a zkvalitnění výuky prostřednictvím ICT

Sada: VY_32_INOVACE_EL_12 - Programování mikroprocesorů Atmel AVR

DUM: VY_32_INOVACE_EL_12_20

Anotace: Systém přerušení, zdroje, obsluha, vnější přerušení, reset

Autor: Ing. Jiří Dítě

Škola: Střední průmyslová škola, Hronov, Hostovského 910

Obor: Počítačové řídicí systémy

Předmět: Počítačové řídicí systémy

Ročník: 3.

Použitá literatura:

- MATOUŠEK DAVID. Práce s mikrokontroléry ATMEL AVR - ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006.
- Atmel Corporation - Microcontrollers, 32-bit, and touch solutions [online]. 2014 [cit. 2014-03-01]. Dostupné z: www.atmel.com

Programování ATmega32 – Systém přerušení

1. Přerušení - přerušovací systém

Systém přerušení, nebo též přerušovací systém je nedílnou součástí každého mikrokontroléru. Každý mikrokontrolér disponuje různě propracovaným přerušovacím systémem, který mimo jiné závisí také na periferiích umístěných na čipu.

Přerušení -> Jedná se o reakci programu na určitou událost.

Co se děje při příchodu žádosti o přerušení?

Pokud mikroprocesor vyhodnotí žádost o přerušení, provede následující sled operací:

- přeruší právě prováděný (běžící) program,
- začne se vykonávat tzv. rutina obsluhy přerušení (program obsluhy přerušení),
- po obsloužení přerušení se mikroprocesor vrátí zpět k původnímu (hlavnímu) programu do místa, kde byl přerušen a pokračuje v jeho vykonávání.

Pozn.: Přerušení může být vyvoláno z kteréhokoliv místa v programu. Je nutné myslet na to, že během přerušení nesmí být měněn obsah registrů, které používá program a samozřejmě obsah stavového registru SREG. Vede to zcela jistě k nekorektnímu běhu programu!!!

Zdroje přerušení

Mikrokontrolér ATmega32 disponuje celkem 21 zdroji přerušení. Tato přerušení mají vyhrazeny v paměti programu FALSH speciální adresy začínající na adrese 0000_H a končí na adrese 0028_H. Čím nižší adresa přerušení, tím vyšší priorita daného přerušení. Znamená to, že pokud se v danou dobu sejdou dvě (nebo i více) přerušení, obslouží se nejprve přerušení, které má nižší adresu a po skončení obsluhy se obslouží přerušení s vyšší adresou. Vždy je vykonáváno pouze jedno přerušení!

Druhy přerušení

- reset - přerušení s nejvyšší prioritou
- vnější přerušení (INT0, INT1, INT2)
- přerušení od čítačů/časovačů (OC2, OVF2, ICP1, OC1A, OC1B, OVF1, OVF0, OC0)
- přerušení od sběrnice SPI (SPI)
- přerušení od sběrnice USART (URXC, UDRE, UTXC)
- přerušení od sběrnice TWI (TWI)
- přerušení od A/D převodníku a analogového komparátoru (ADDC, ACI)
- přerušení od paměti EEPROM na čipu (ERDY)
- přerušení od jednotky pro zápis do programové paměti (SPMR)

V následující tabulce jsou uvedeny adresy jednotlivých přerušení včetně jejich stručného popisu:

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

Každý zdroj přerušení definuje prostor na jednu instrukci o délce 1Word (1Slovo). Jedná se o instrukci ***RJMP***, která skočí na adresu (návěští) v paměti programu, kde je umístěn obslužný program.

2. Obsluha přerušení

Mikrokontrolér disponuje několika registry pro řízení přerušení. Zde budou zmíněny registry tzv. globální.

Registr SREG

Jak již bylo dříve zmíněno, přerušení je **globálně** řízeno bitem **I** stavového registru ***SREG***.

- **I** = log. 1… povoleno přerušení
- **I** = log. 0… zakázáno přerušení

Pokud dojde k aktivaci přerušení, je bit **I** v **SREG** automaticky vynulován a po návratu z přerušení opět automaticky nastaven při vykonání instrukce pro návrat z přerušení **RETI**.

Základní vlastnosti přerušení

- každý zdroj přerušení má svůj tzv. **maskovací registr,** ve kterém lze nastavením patřičných bitů aktivovat přerušení od daného zdroje. Například pro vnější vstupy přerušení je to registr **GICR**, pro čítače/časovače je to registr **TIMSK** atd.
- každý zdroj přerušení má svůj tzv. **příznakový registr**, ve kterém se nastaví patřičný bit, pokud dojde k přerušení. Tento bit lze u některých registrů ručně nulovat zápisem log. 1 na patřičný bit. Například pro vnější vstupy přerušení je to registr **GIFR**, pro čítače/časovače je to registr **TIFR** atd.
- **pokud je globálně zakázáno přerušení, tak se nové žádosti zapamatují** formou příznaků v příznakových registrech a po povolení přerušení se postupně podle priority obslouží
- Vnější **úrovňově citlivé vstupy nepoužívají příznak**, a tedy pamatují se pouze po dobu, kdy je daný vstup aktivní.
- **stavový registr SREG není automaticky ukládán při vstupu do přerušení!!! Nutno** si jej hned na začátku každého přerušení uložit do zásobníku a před návratem z přerušení opět obnovit ze zásobníku.

Časová odezva přerušení

Časová odezva přerušení udává, jak rychle je přerušení mikrokontroléru obslouženo. **Minimální čas** pro obsloužení jednoho přerušení jsou **4 + 2 hodinové cykly**.

- Během 4 cyklů dojde k uložení obsahu PC (programového čítače) do zásobníku, obsah SP (ukazatele zásobníku) je snížen o 2 a zakáže se globální přerušení (vynulováním bitu I v SREG)
- Během dalších 2 cyklů je vykonána instrukce RJMP, která skočí na návěští, kde začíná rutina obsluhy přerušení

Následně se provede samotná rutina, jejíž délka je závislá na konkrétní sekvenci instrukcí
Návrat z přerušení obstarává instrukce **RETI**, která trvá další **4 hodinové cykly**.

- Dojde k vyzvednutí návratové adresy ze zásobníku a uložení do PC (programového čítače), SP (ukazatel zásobníku je zvýšen o 2 a povolí se globální přerušení (I=1).
- Běh programu se vrátí do místa, kde byl přerušen a pokračuje dále.

3. Reset mikrokontroléru

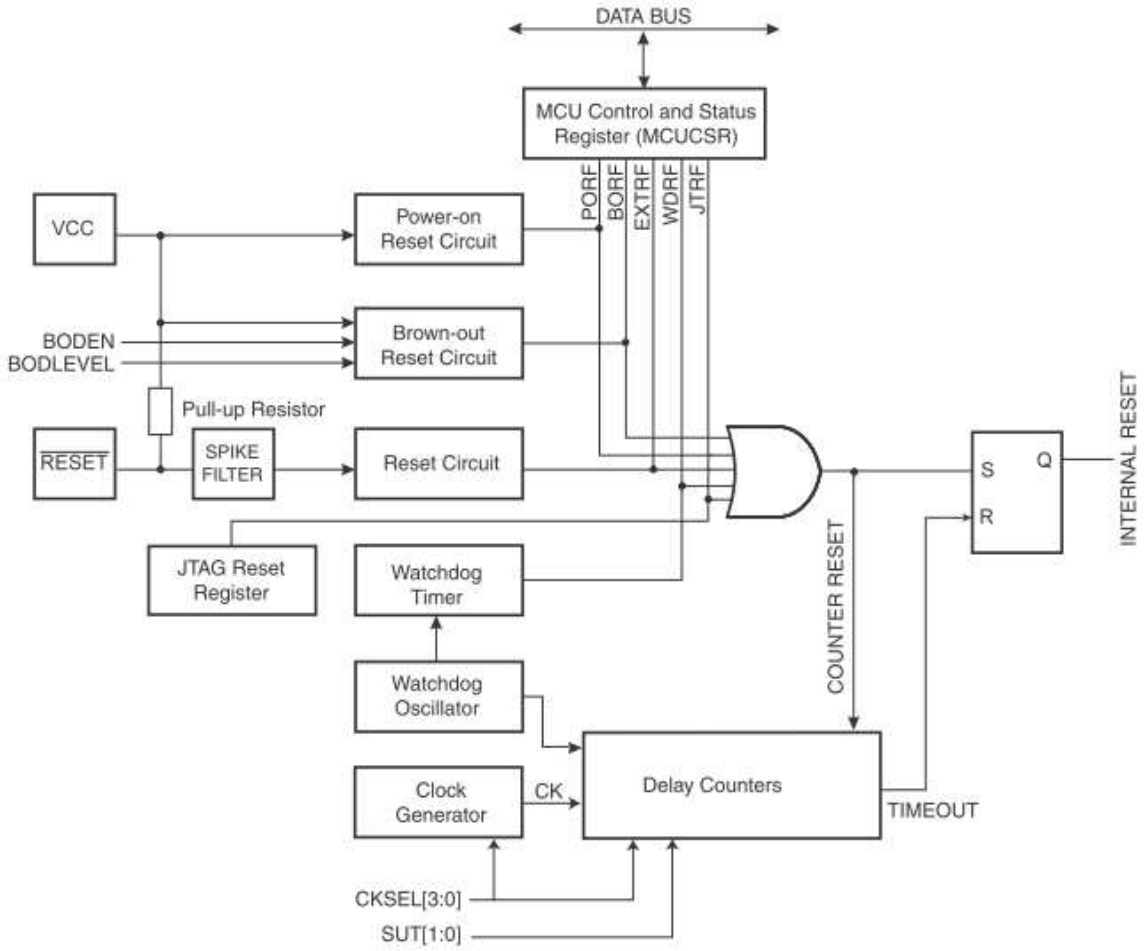
Jedním ze zcela specifických přerušení je RESET. Toto přerušení má nejvyšší prioritu a při jeho aktivaci dochází k tzv. znovuspuštění programu (od adresy 0000_H). Mimo to se obsahy některých registrů uvedou do výchozího stavu. Hovoříme také o tzv. inicializaci mikrokontroléru.

Zdroje resetu

Mikrokontrolér ***ATmega32*** obsahuje pět zdrojů resetu:

- ***POR (Power-On Reset)*** *- reset po připojení napájecího napětí (vyvolá se, pokud napětí překročí hodnotu asi 1,3V)*
- ***Vnější Reset*** *- k resetu dojde po přivedení úrovně log. 0 na vstup /RESET*
- ***Watchdog Reset*** *- vyvolá ho vypršení času časovače WDT (Watch-Dog-Timer)*
- ***Brown-out Reset*** *- reset vyvolaný poklesem napájecího napětí pod hodnotu 2,7V (BODLEVEL = 1), nebo 4,0V (BODLEVEL = 0)*
- ***JTAG AVR Reset*** *- reset vyvolaný od ladicího rozhraní JTAG*

Logika zpracovávající RESET



Po provedení resetu je možné v registru MCUCSR zjistit, jaký zdroj byl příčinou resetu. Registr MCUCSR obsahuje příznaky (indikátory) zdrojů přerušení.

- ***Registr MCUCSR***

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

PORF - příznak resetu po připojení napájecího napětí (lze ho nulovat zápisem log. 0)
EXTRF - příznak vnějšího resetu (nuluje se při POR nebo zápisem log. 0)
BORF - příznak Brown-Out resetu (nuluje se při POR nebo zápisem log. 0)
WDRF - příznak WDT resetu (nuluje se při POR nebo zápisem log. 0)
JTRF - příznak JTAG resetu (nuluje se při POR nebo zápisem log. 0)

4. Vnější přerušení - vstupy INT0, INT1, INT2
K vyvolání přerušení od těchto vstupů může dojít v těchto případech:

- přítomnost ***náběžné hrany*** signálu na vstupu
- přítomnost ***spádové hrany*** signálu na vstupu
- přítomnost ***náběžné i spádové*** hrany signálu na vstupu (pouze u vstupů INT0 a INT1)
- přítomnost ***úrovně log. 0*** na vstupu (pouze u vstupů INT0 a INT1)

Pozn.: Toto přerušení lze vyvolat i tehdy, jsou-li tyto vývody konfigurovány jako výstupy!

Registry pro řízení přerušení - GICR, GIFR, MCUCR, MCUSR

- **Registr GICR (General Interrupt Control Register)**

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Prvním registrem pro **řízení přerušení** je registr **GICR**. Obsahuje 3 důležité bity (INT0, INT1, INT2) pomocí nichž se **povoluje přerušení od daného vstupu**. (**log. 0 - zakazuje** přerušení, **log. 1 - povoluje** přerušení). Po resetu mikrokontroléru jsou přerušení zakázána.

- **Registr GIFR (General Interrupt Flag Register)**

Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	INTF2	–	–	–	–	–	GIFR
Read/Write	R/W	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Jedná se o **příznakový registr**. Pomocí příznaků INTF0, INTF1 a INTF2 se udržují požadavky na přerušení od příslušných vstupů v případě zakázaném přerušení (při vykonávání jiného přerušení nebo při globálním zakázání přerušení,…).
Pozor!!! Pokud je vstup konfigurován na reakci od úrovně log. 0, k zapamatování nedojde. Příznak bude nastaven pouze po dobu, kdy je vstup v log. 0!!!

- **Registr MCUCR (MCU Control Register)**

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
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	

Pomocí bitů ISC11, ISC10, ISC01 a ISC00 se konfiguruje citlivost daného vstupu na hranu nebo úroveň.

Citlivost vstupu INT0:
Pro nastavení slouží bity ISC01 a ISC00 (viz následující tabulka):

ISC01	ISC00	Popis
0	0	Vstup INT0 se aktivuje log.0 (úrovňově citlivý vstup)
0	1	Jakákoliv změna na vstupu generuje přerušení
1	0	Vstup INT0 se aktivuje sestupnou hranou (hranově citlivý vstup)
1	1	Vstup INT0 se aktivuje náběžnou hranou (hranově citlivý vstup)

Citlivost vstupu INT1:
Pro nastavení slouží bity ISC11 a ISC10 (viz následující tabulka):

ISC11	ISC10	Popis
0	0	Vstup INT1 se aktivuje log.0 (úrovňově citlivý vstup)
0	1	Jakákoliv změna na vstupu generuje přerušení
1	0	Vstup INT1 se aktivuje sestupnou hranou (hranově citlivý vstup)
1	1	Vstup INT1 se aktivuje náběžnou hranou (hranově citlivý vstup)

- **Registr MCUCSR (MCU Control and Status Register):**

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

V tomto registru je využit pouze jeden bit a to ISC2, který slouží pro konfiguraci citlivosti přerušení od vstupu INT2.
- ISC2 = log. 0 - vstup INT2 je citlivý na **sestupnou hranu** signálu
- ISC2 = log. 1 - vstup INT2 je citlivý na **náběžnou hranu** signálu