UVOD V PROGRAMIRANJE MIKROKRMILNIKOV

Janez Pogorelc

UM-FERI, 2005

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

http://www.feri.uni-mb.si

Inštitut za Robotiko

http://www.ro.feri.uni-mb.si

http://www.hipulab.uni-mb.si/

CIP - Kataložni zapis o publikaciji
Univerzitetna knjižnica Maribor

004.42(075.8)

POGORELC, Janez
Uvod v programiranje mikrokrmilnikov : [zbrano gradivo za predavanja] / Janez Pogorelc. - 1. izd. - [Maribor] : Fakulteta za elektrotehniko, računalništvo in informatiko : Inštitut za robotiko, 2005

ISBN 86-435-0694-X

COBISS.SI-ID 54573825

ISBN 86-435-0694-X
9 788643 506943

Naslov: **UVOD V PROGRAMIRANJE MIKROKRMILNIKOV, Zbrano gradivo za predavanja**
Prva izdaja, marec 2005
Avtor: višji predavatelj mag. Janez Pogorelc, univ. dipl. inž.
Strokovni recenzent: doc. dr. Martin Terbuc, univ. dipl. inž., Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko
Vrsta učnega gradiva: zbrana gradivo za predavanja
Uredil in oblikoval: Janez Pogorelc
Slike: Janez Pogorelc
Oblikovanje naslovnice za CD zgoščenko: Janez Pogorelc
Naklada: 11 izvodov CD zgoščenk
Obseg: 57 strani
Izdala: Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za robotiko
Tisk: Janez Pogorelc
ISBN 86-435-0694-X

Kazalo

Seznam slik

Seznam tabel

Seznam programov

Namen in cilji učnega gradiva

Učno gradivo obravnava osnove uporabe in programiranja 8-bitnih PIC mikrokrmilnikov srednje kategorije. Namenjeno je predvsem študentom 2. letnika visokošolskega strokovnega programa Elektrotehnika, smer avtomatika pri predmetih Mikroelektronika in Mikroračunalniški sistemi.

Podan je podroben opis mikrokrmilniškega učnega kompleta **MPU-PIC16F876**, ki je bil razvit na Inštitutu za robotiko UM-FERI za podporo pedagoškemu procesu. Sledi opis zgradbe in delovanja mikrokrmilnika Microchip PIC 16F876 in uporabe programskih orodij za programiranje in preizkušanje programov v zbirnem in C-jeziku za mikrokrmilnike družine Microchip PIC 16F87x.

Za **programiranje v zbirnem jeziku** so podrobnejše opisani ukazi, načini naslavljanja, model pomnilnika in programski model vhodno-izhodnih vmesnikov (digitalni – logični vhodi/izhodi, analogni vhodi, PWM izhodi, časovniki, prekinitve, PWM izhodi). Predstavljen je postopek kodiranja elementarnih in naprednejših programov za dostop do vhodno/izhodnih vmesnikov ter obdelavo podatkov tako z logičnimi operacijami kot tudi računskimi operacijami v celoštevilčni aritmetiki.

Dodani so zgledi tipičnih programov za delo z vhodno/izhodnimi vmesniki, na osnovi katerih lahko kodiramo lastne programe in jih preizkušamo s pomočjo osebnega računalnika na mikrokrmilniškem učnem kompletu **MPU-PIC16F876**.

Za **programiranje v C-jeziku** je podan opis ustreznih programskih orodij in principov kodiranja algoritmov za dostop do vhodno/izhodnih vmesnikov. Vendar je za razumevanje programov v C-jeziku **potrebno poznavanje ANSI C programskega jezika**, medtem ko je **za osnovno delo v zbirnem jeziku priročnik namenjen tudi začetnikom**. Za nevešče uporabnike programskega jezika C, je priporočljiv predhodni obisk tečaja iz osnov programiranja v ANSI C jeziku, ali da se naučijo osnov iz ustreznih priročnikov.

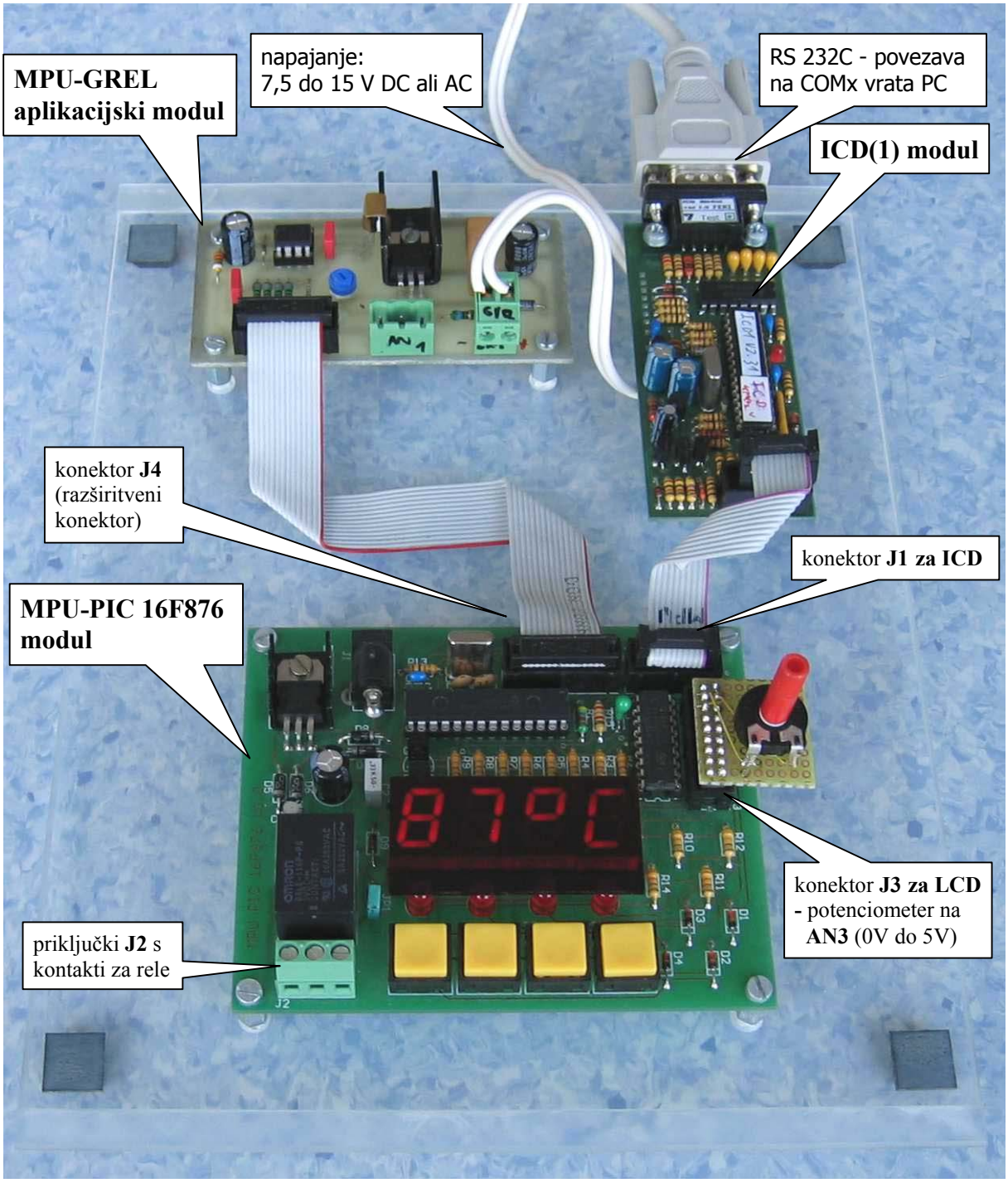
Zbrano gradivo je primerno tudi za tiste, ki uporabljajo druge tipe Microchip PIC mikrokrmilnikov srednje kategorije (s 14-bitno programsko besedo), npr.: 16F84, 16F877A in drugi.

V učnem gradivu je večje število izpisov popolnih (delujočih) programov tako v zbirnem kot tudi v C jeziku. Še več zgledov kot tudi opisov študentskih projektov, ki temeljijo na uporabi PIC mikrokrmilnikov, je na voljo uporabnikom na spletnih straneh predmetov Mikroelektronika in Mikroračunalniški sistemi v okviru spletišča Inštituta za robotiko (http://www.ro.feri.uni-mb.si/izobrazevanje/).

Pričujoče učno gradivo je bilo oblikovano »po merilih za e-učna gradiva« iz **Poslovnika založniške dejavnosti v e-izobraževanju na Univerzi v Mariboru** (aktivno kazalo, nadpovezave na poglavja, slike, tabele in HTML reference).

1. Predstavitev MPU-PIC16F876 učnega kompleta

Spoznali boste učni komplet MPU-PIC16F876 in osnovne lastnosti mikrokrmilnika PIC 16F876. Naučili se boste identificiranja vhodnih oziroma izhodnih priključkov mikrokrmilnika.



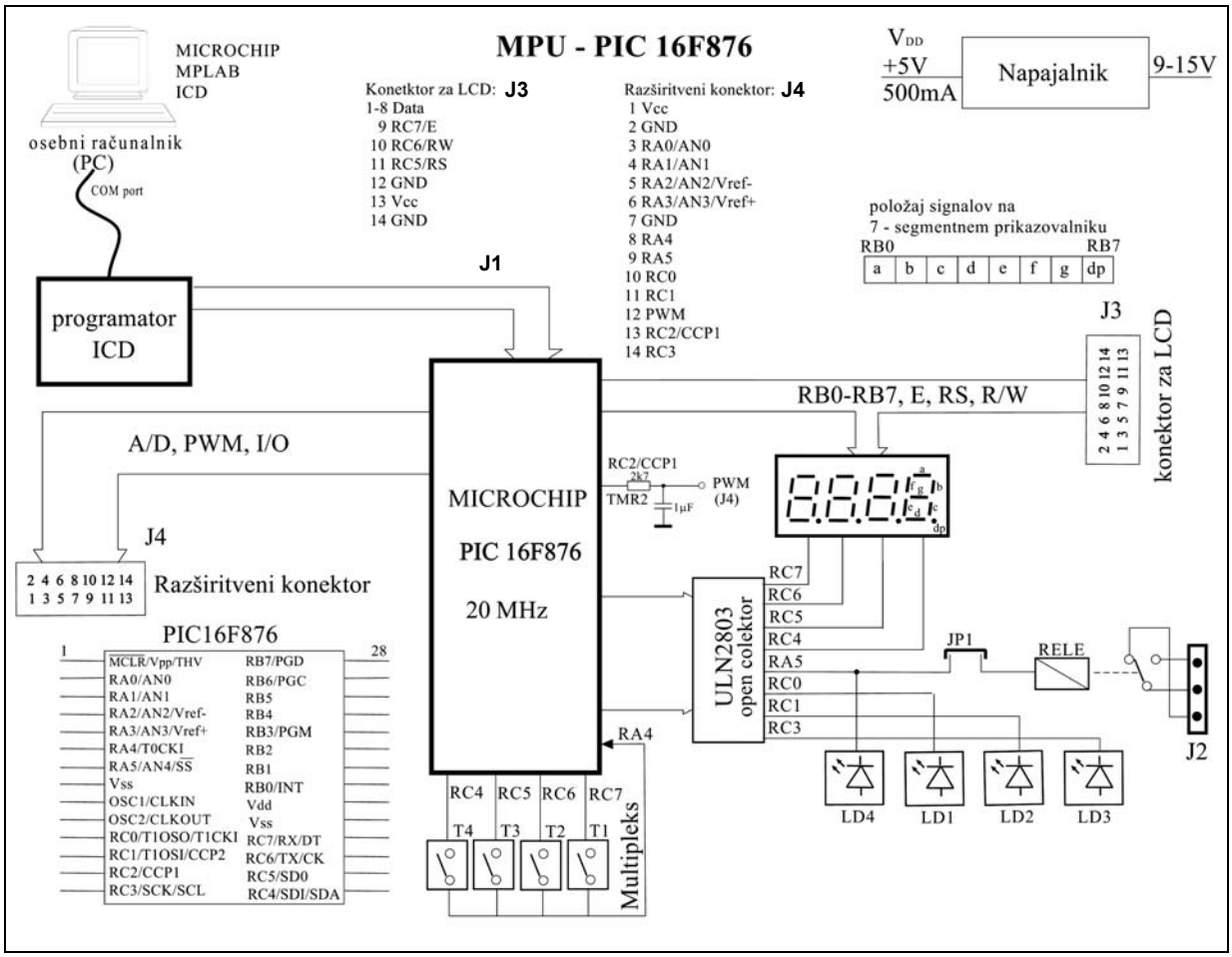
Slika 1-1: Fotografija MPU PIC 16F876 kompleta

Mikrokrmilniški učni komplet **MPU-PIC 16F876** je sestavljen iz treh modulov (Slika 1-1):

- **MPU-PIC16F876** modul
- **ICD**(1) modul
- **MPU-GREL** aplikacijski modul

1.1. Modul MPU-PIC16F876

Zgradbo modula in povezavo vhodno/izhodnih naprav na priključke vrat prikazuje blokovna shema (Slika 1-2):



Slika 1-2: Blokovna shema MPU PIC 16F876 modula

Modul **MPU-PIC16F876** je bil razvit na UM-FERI, Inštitutu za robotiko za potrebe učenja osnov programiranja in uporabe PIC mikrokrmilnikov [4]. Zgrajen je na osnovi mikrokrmilnika **Microchip PIC16F876.** Na kartici so na voljo tudi:

- 4 enote LED indikatorjev;
- 4 enote - 7-segmentni LED prikazovalnik;
- 4 tipke;
- rele z delovnim in mirovnim kontaktom;
- 14-polni razširitveni konektor (priključki za analogne vhode, PWM izhod, časovniki, ...) (J4);
- 10-polni konektor za priključitev ICD(1) modula (J1);

- konektor za povezavo z osebnim računalnikom (RS232C oz. COMx vrata);
- 14-polni konektor za priključitev LCD prikazovalnika (J3).

1.1.1. Mikrokrmilnik PIC 16F876

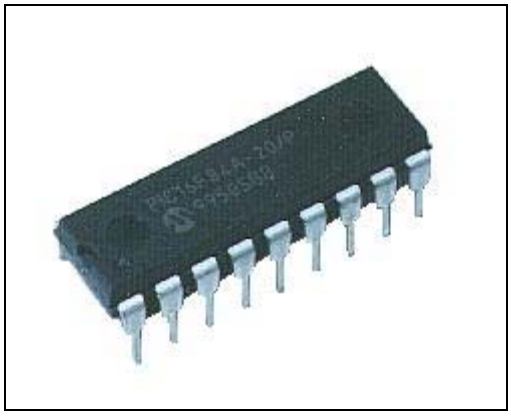
Mikrokrmilnik PIC (angl.: Peripheral Interface Controller) je integrirano vezje (IC), ki je namenjeno krmiljenju in zaznavanju zunanjih (perifernih) naprav v skladu s programom, ki ga izvaja vgrajena procesna enota (CPE). V primerjavi z živim bitjem predstavlja CPE možgane, PIC pa je ekvivalent živčnemu sistemu [16].

V splošnem je **mikrokrmilnik integrirano vezje, ki v enem čipu združuje procesno enoto, pomnilnik (RAM, ROM) in vhodno/izhodne (periferne) vmesnike** [1]. Ameriški proizvajalec **MICROCHIP** [7] je specializiran za izdelavo 8-bitnih eno-čipnih mikrokrmilnikov in spada med največje tovrstne proizvajalce v svetu.

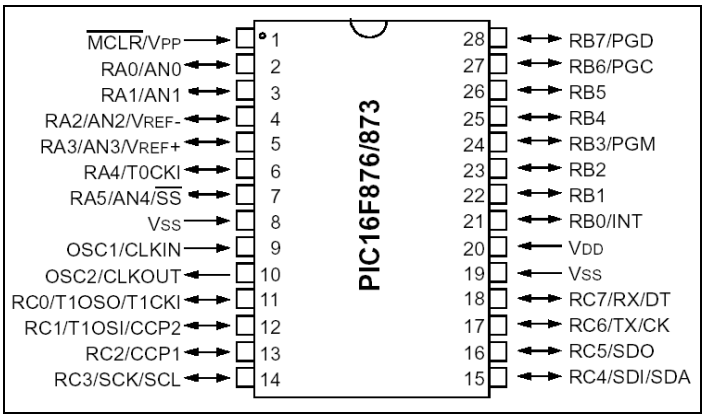
V nadaljevanju bodo obravnavani PIC mikrokrmilniki **»srednje kategorije«** (angl.: **Mid Range**) z vsebovanim programskim pomnilnikom tipa FLASH ROM (možno ga je reprogramirati več kot 1000-krat). Značilnost te kategorije je, da imajo čipi oznako **16Fxxx** in da vsi vsebujejo enako **procesno enoto s 14-bitnimi ukazi**.

Obravnavani **MPU sistem** vsebuje mikrokrmilnik **16F876** (Slika 1-4) v 28-polnem ohišju [8] (tudi 16F873 je v enakem ohišju, vendar vsebuje manj pomnilnika). Med bližnje sorodnike spadajo tudi mikrokrmilniki **16F877/874**, ki so nameščeni v ohišju s 40 priključki, zato imajo dodatnih 12 vhodno/izhodnih priključkov (s tem tudi več perifernih vmesnikov). Ker imajo vsi navedeni mikrokrmilniki enako procesno enoto in v osnovi enake vhodno/izhodne vmesnike, jih pogosto označujemo kar **16F87x(A).** Pripona **A** predstavlja tehnološko posodobljeno različico integriranega vezja z enakimi funkcijami.

Popularni so tudi mikrokrmilniki z oznako **PIC16F84(A)** [2], ki imajo enako procesno enoto, vendar manj perifernih vmesnikov, zato so vgrajeni v 18-polno ohišje, uporabni pa so za manj zahtevne aplikacije (Slika 1-3).



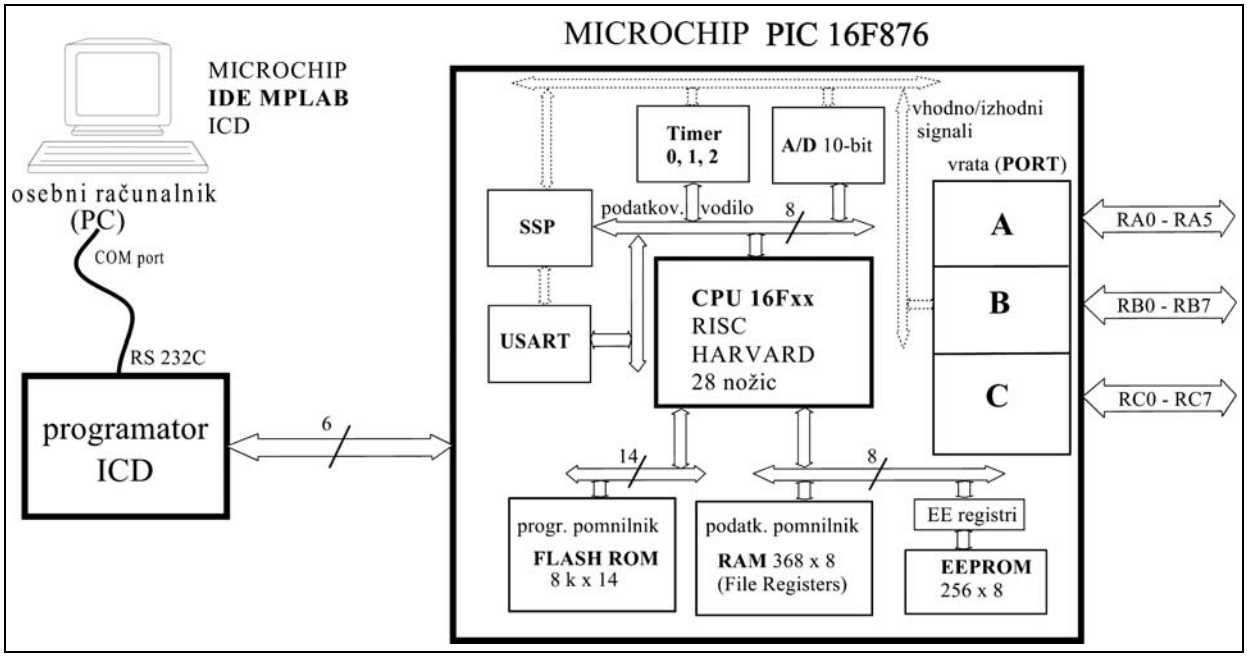
Slika 1-3: PIC 16F84A v »DIL« ohišju



Slika 1-4: Razporeditev priključkov PIC 16F876

Mikrokrmilnik **PIC16F876** ima speljane priključke (Slika 1-4) vseh vsebovanih vhodno/izhodnih vmesnikov na **vrata** (angl.: **PORT**) **A**, **B** in **C** (Slika 1-5). Vsaka

vrata vsebujejo tipično 8 linij (kar ustreza osmim bitom v registrih), pri čemer se v imenu oznake priključkov pojavlja številčna oznaka med 0 in 7, ki predstavlja pozicijo bita v registru. Ker je število vsebovanih perifernih vmesnikov veliko večje od razpoložljivega števila nožic, imajo mnogi priključki dvojni ali celo trojni namen, katerega izberemo v ustreznem konfiguracijskem registru.



Slika 1-5: Zgradba mikrokrmilnika PIC 16F876

Pomembnejše lastnosti mikrokrmilnika PIC 16F876:

- mikrokrmilnik s HARVARD arhitekturo in 8-bitno procesno enoto z RISC naborom ukazov;
- **35 ukazov** fiksne dolžine (14 bitov);
- **8Kx14-bitnih besed programskega pomnilnika (FLASH);**
- **368 zlogov podatkovnega pomnilnika (datotečni registri - f);**
- **256 zlogov električno zbrisljivega pomnilnika (EEPROM);**
- **22 vhodno/izhodnih priključkov** (tokovna obremenljivost do 25 mA);
- 3 časovniki (2x8-bit in 1x16-bit);
- 2 PWM vmesnika (10 bitov);
- 5 analognih vhodov (10-bitnih) za analogno/digitalno pretvorbo;
- sinhroni in asinhroni serijski vmesnik (SSP, I2C, RS232, USART/SCI, ...);
- 13 prekinitvenih izvorov;
- **maksimalno 20 MHz** urin takt (**večina ukazov** se izvede v **0,2 μs**);
- **nizka poraba** (20 μA pri napajalni napetosti 3 V in urinem taktu 32 kHz).

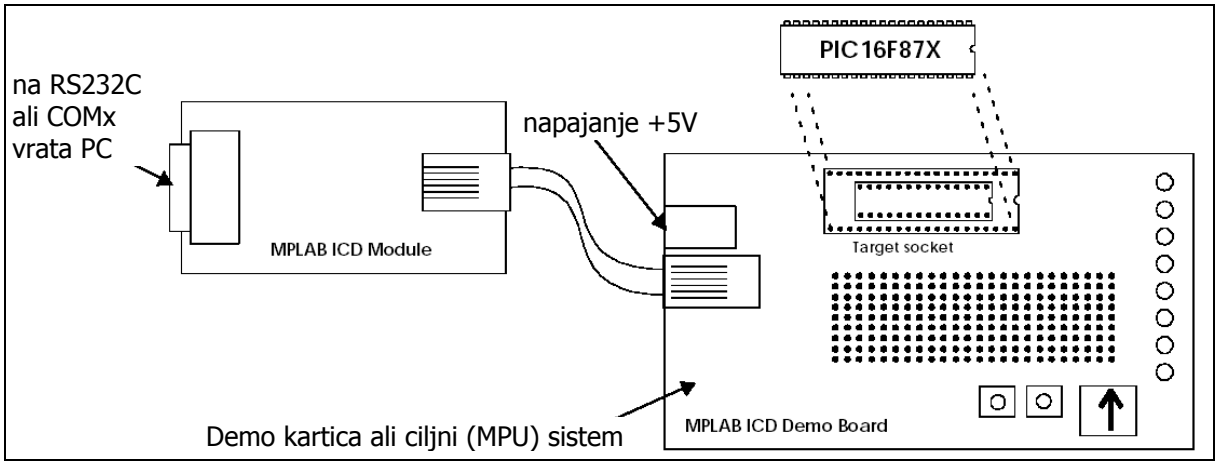
Mikrokrmilnike PIC16F87x programiramo z uporabo integriranega programskega okolja ***Microchip IDE MPLAB*** [10] (priporoča se V5.70 za uporabo skupaj z **ICD1**) v zbirnem jeziku (Microchip Assembler) in/ali v C-jeziku (potrebna je namestitev ene od različic prevajalnika).

1.2. ICD modul

Za programiranje in razhroščanje programov (angl.: Debugging, kontrolirano izvajanje programa) v ciljnem sistemu MPU-PIC16F876 je predvidena uporaba **ICD(1) modula** [9].

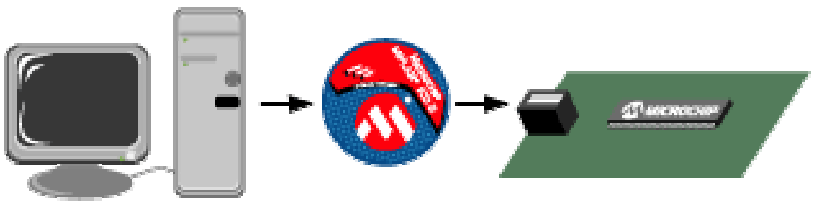
Proizvajalec **Microchip** [7] je hkrati s predstavitvijo mikrokrmilnikov 16F87x dal na tržišče razmeroma dostopen in zmogljiv razvojno-testni komplet za programiranje in testno izvajanje programov (razhroščanje) v ciljnem sistemu **ICD** (angl.: **I**n **C**ircuit **D**ebugger). Prednost uporabe **ICD** modula napram cenejšim različicam programatorjev je v tem, da je možno serijsko programiranje kar v vezju (**ICSP**, angl.: **I**n **C**ircuit **S**erial **P**rogramming), kar pomeni, da ni potrebno fizično poseganje v vezje.

ICD modul (Slika 1-6) je na eni strani povezan z osebnim računalnikom na serijska vrata (COMx port oz. RS232C), na drugi strani pa na ciljni sistem z vzporednim ali telefonskim kablom preko 10-polnega ali 6-polnega (telefonskega) konektorja. **ICD** modul deluje pod kontrolo programskega okolja **MPLAB** (V5.70) in omogoča razen programiranja tudi testno izvajanje programa (izvajanje po korakih, izvajanje do prekinitvene točke, sledenje spremenljivk in pomnilniških lokacij, ...).



Slika 1-6: Shematski prikaz povezave ICD(1) modula

Kasneje je **Microchip** dal na tržišče prenovljeno različico modula pod imenom **ICD2** (Slika 1-7), ki uporablja popularnejšo povezavo z osebnim računalnikom (**USB**) in podpira širši nabor mikrokrmilnikov s FLASH ROM pomnilnikom (tudi 18Fxxx), razen tega pa tudi omogoča uporabo naprednejše različice **MPLAB** integriranega programskega okolja (V6.xx).

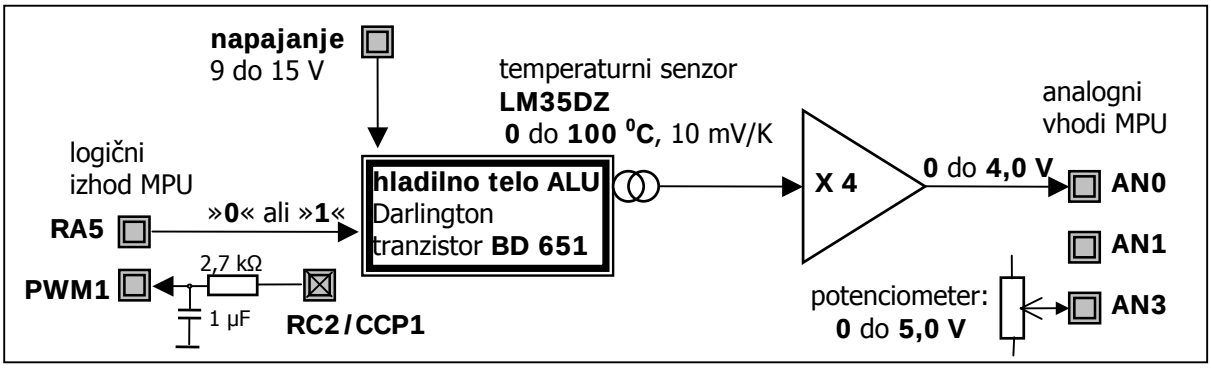


Slika 1-7: Simbolični prikaz povezave ICD2 modula

Opozorilo: Če uporabljamo **ICD modul tudi za razhroščanje** (angl.: Debug mode, kontrolirano izvajanje programa), se moramo zavedati, da so zaradi komunikacije s ciljnim mikrokrmilnikom **zasedena nekatera sredstva**: linije **RB3**, **RB6**, **RB7**, del programskega pomnilnika (na koncu ROM-a) in del podatkovnega pomnilnika.

1.3. MPU-GREL aplikacijski modul

MPU-GREL je kartica (Slika 1-8), ki vsebuje polprevodniški temperaturni senzor (signal je preko ojačevalnika spojen na analogni vhod mikrokrmilnika **AN0**).



Slika 1-8: Blokovna shema aplikacijskega modula MPU-GREL

Senzor **LM35DZ** je pritrjen na aluminijasto hladilno telo, ki pripada močnostnemu tranzistorju, katerega je možno segrevati z vklopom oz. izklopom preko digitalnega (preklopnega, logičnega) izhoda mikrokrmilnika **RA5**.

Na kartici (podrobna shema: http://www.ro.feri.uni-mb.si/predmeti/mikro_el/PIC_grelec.pdf) je na voljo še nekaj priključkov PIC16F876, ki so namenjeni testiranju vmesnikov in preizkušanju programov:

- **AN1** – prost priključek za analogni vhod številka 1;
- **AN3** – priključek za analogni vhod številka 3, oziroma pozitivno referenčno vrednost V_{ref+} (na sponko je priključen potenciometer za ročno nastavitev napetosti, ki je fizično nameščen na MPU modulu, konektor J3);
- **PWM1** – filtriran pulzno-širinski izhod (priključek mikrokrmilnika **RC2/CCP1**), ki je speljan preko RC člena s časovno konstanto približno 2,7 ms.

Za razumevanje tematike je potrebno tudi predznanje iz osnov digitalne elektronike in gradnikov mikroprocesorskih sistemov, kar je na voljo med drugim v [3]. Dobri opisi zgradbe in uporabe sistemov s PIC mikrokrmilniki so tudi v [2] in [6].

Vprašanja za utrjevanje:

1. *Kateri moduli tvorijo MPU-PIC16F876 komplet ?*
2. *Kakšen tip procesne enote je vsebovan v mikrokrmilniku PIC16F876 ?*
3. *Kakšni pomnilniški elementi so vsebovani v mikrokrmilniku PIC16F876 ?*
4. *Koliko vhodno/izhodnih priključkov ima mikrokrmilnik PIC16F876 ?*
5. *Katere funkcije opravlja ICD modul ?*
6. *Katere vhodne in izhodne signale vsebuje modul MPU-GREL ?*

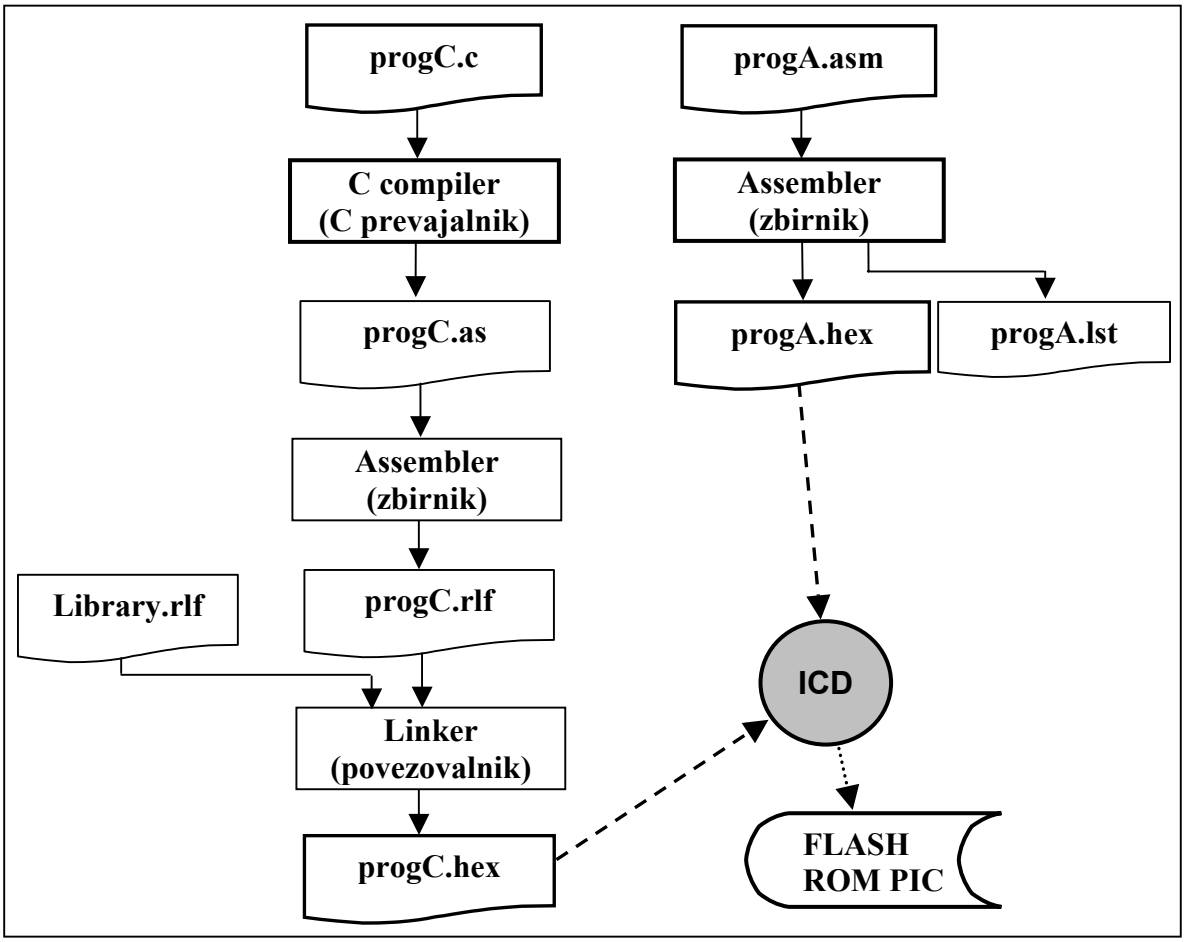
2. Programiranje PIC16F87x v zbirnem jeziku

Naučili se boste oblikovanja programa v zbirnem jeziku (splošni format) ter osnovne programske ukaze in njihov pomen. Ko boste obvladali nekatere podrobnosti, boste v nadaljevanju lahko uporabljali zgolj zgoščeno tabelo nabora vseh 35 ukazov.

Programiranje PIC16F876 lahko poteka v **zbirnem jeziku** (Assembler, integriran v **MPLAB**) [6] ali v **programskem jeziku C** [5][12] (potrebno ga je dodatno naložiti, npr. [14]).

Izvorni program [1] (angl.: Source code programm) napišemo s pomočjo osebnega računalnika v enem od tekstovnih urejevalnikov (Beležnica, Notepad, UltraEdit, ...) ali kar v urejevalniku v **MPLAB** programskem okolju. Izvorni program v zbirnem jeziku ima obvezno končnico **.asm**, izvorni program v C-jeziku pa **.c**.

Diagram poteka (Slika 2-1) prikazuje postopek pretvarjanja (prevajanja) izvornega programa v C-jeziku [14] ali zbirnem jeziku [9] v datoteko izvršljive oblike (.hex), ki jo lahko s posredovanjem **ICD** modula (ali simulatorja) naložimo v pomnilnik mikrokrmilnika v ciljnem sistemu (ali kar zaženemo z **MPLAB Simulator** orodjem).



Slika 2-1: Postopek tvorjenja izvršljivega programa

V kolikor **prevajalnik** ali **zbirnik** zazna **sintaktično napako**, se postopek prekine, kar pomeni, da mora uporabnik odpraviti napake ter **ponovno sprožiti proces**

prevajanja (angl.: Build all) **in nalaganja** (angl.: Download) ter **programiranja** (angl.: Program) **FLASH ROM pomnilnika** v ciljnem mikrokrmilniku. Če se pri izvajanju programa na ciljnem sistemu pokažejo **logične napake**, je seveda **potrebno odpraviti le-te in ponoviti postopek prevajanja.**

2.1. Program v zbirnem jeziku

V splošnem imajo programi, ki jih pišemo (kodiramo) v zbirnem jeziku, dokaj strogo sintakso. Izvorni program je sestavljen iz zaporedja vrstic, ki (ločeno) predstavljajo:

- **programsko izvedljivo vrstico;**
- vrstico z psevdoukazom ali zbirniško smernico;
- komentarsko ali prazno vrstico.

Format prvih dveh tipov vrstic je natanko določen (Program 2-1). Priporočljivo je, da stolpce (polja) ločujemo z znakom 'TAB', sicer je po pravilih za ločevanje polj dovolj en presledek. Vsaka vrstica je razdeljena na več polj [1]:

- **Polje1** se imenuje področje **označb vrstice (Label)**. **Obvezno** se vpiše **v 1. stolpec** vrstice in se **začne s črko** (alfabetski znak brez šumnikov) ali s podčrtajem '_', nadaljuje se lahko tudi s številkami. Število znakov je omejeno na 32. Priporočljivo je uporabljati smiselna imena (npr.: `temperatura`), vendar se moramo izogibati rezerviranim imenom (imena registrov, imena ukazov). Program »**zbirnik«** ločuje velike in male črke.
- **Polje2** vsebuje **mnemonična imena ukazov procesorja** ali **makro ukaze** (za programsko izvedljive vrstice) oziroma **psevdoukazna imena** (za vrstice z zbirniškimi smernicami). **Mnemonična imena ukazov** definira proizvajalec procesorja (za Microchip mikrokrmilnike srednje kategorije PIC16Fxxx je potrebno poznati le 35 ukazov). Uporabljajo se lahko tako **velike kot male črke**.
- **Polje3** vsebuje **operande iz nabora načinov naslavljanja** (za programsko izvedljive vrstice) oziroma **definirana simbolna imena** ali **vrednosti** (za vrstice z zbirniškimi smernicami). Številčne vrednosti lahko zapišemo v različnih številskih formatih: desetiško `.39` ali `d'39'`, šestnajstiško `0x7E` ali `h'7E'` ali `7Eh`, dvojiško `b'10110001'`, ASCII znakovne konstante `'G'` ali `a'GcX'`.
- **Polje4** (ki ni obvezno) se začenja s podpičjem '`;`' in se nadaljuje s poljubnim komentarjem (vsekakor je **priporočljivo pisati smiselne komentarje** v kontekstu posledice izvršitve ukaza).

polje1	polje2	polje3	polje4
	list	P=16F876	; psevdoukaz za določitev tipa procesorja
sprem	equ	0x20	; psevdoukaz za prireditev naslova simbolu
	org	0	; psevdoukaz: določitev vrednosti programskega števca PC=0
	clrf	sprem	; ukaz za brisanje sprem
zanka	incf	sprem, f	; ukaz za povečanje sprem v registru f za 1
	goto	zanka	; ukaz za brezpogojni skok na označbo
	end		; psevdoukaz za konec programa
'TAB'	'TAB'	'TAB'	

Program 2-1: Splošni format izvornega programa v zbirnem jeziku

Če v 1. stolpcu (ali kasneje) vpišemo znak podpičje '`;`' , ki mu sledi poljuben tekst, dobimo vrstico z izključnim komentarjem. **Zaradi povečanja preglednosti in čitljivosti programa je smiselno vstavljati** tudi **prazne vrstice**.

Zbirnik (Assembler) (angl.: Assembling, »zbiranje«) pretvarja izvorno datoteko (npr.: progA.asm) sekvenčno – vrstico za vrstico. Vsako **programsko izvedljivo vrstico** pretvori v **14-bitno besedo (strojna koda),** pri čemer uporablja **vrstice s psevdoukazi (zbirniškimi smernicami)** kot navodilo za tvorjenje strojne kode. Postopek poteka v dveh korakih, kajti potrebno je upoštevati tudi vrednosti simbolov, ki so definirani kasneje. Če ni sintaktičnih napak, se tvori datoteka s šestnajstiško zapisanimi kodami (progA.hex), katere vsebino lahko prenesemo (programiramo) v FLASH ROM pomnilnik mikrokrmilnika v ciljnem sistemu.

Registrska struktura je (na prvi pogled) precej skromna, kajti v programskem modelu je na voljo **en sam 8-bitni splošno uporaben register z oznako W** (**W**ork). Vendar je potrebno upoštevati tudi, da je na voljo kar **368 datotečnih registrov** (**F**ile register), ki jih v ukazih lahko povečini enakovredno uporabljamo tako za izvorne kot ciljne operande.

Trajanje ukazov je tipično en strojni cikel (4 urini cikli, kar pomeni **200 ns** pri 20 MHz taktu), pri čemer je tudi nekaj ukazov, ki lahko pogojno trajajo dva cikla in nekaj, ki vedno trajajo dva cikla (zgoščen opis ukazov prikazuje Tabela 2-1).

2.2. Podrobnejši opis ukazov

Nabor ukazov zbirnega jezika obsega **35 mnemoničnih imen ukazov** v skladu z RISC arhitekturo [2][4][6][8]. Podrobno so opisani v nadaljevanju.

Vsak ukaz (Tabela 2-1) v **programsko izvedljivih vrsticah** se prične v **polju2** z zapisom mnemoničnega imena ukaza. Temu sledi **polje3**, kamor vpisujemo operande. Nekateri ukazi ne potrebujejo operandov in nekateri vsebujejo le en operand: izvorni operand (konstanta **k**) ali ciljni operand (datotečni register **f**).

Večina ukazov je takšnih, da je prvi operand izvorni (datotečni register **f**), drugi pa ciljni (bit **b** ali pa vrednost **d**, ki pomeni izbiro cilja med **W** in **f**). V splošnem je **»izvorni operand«** tisti, ki se **ne spreminja**, ampak samo bere, **»ciljni operand«** pa tisti, kamor se **shranjuje rezultat operacije**. Upoštevati je potrebno, da se v skladu z rezultatom operacije (predvsem pri aritmetičnih in logičnih ukazih) spreminjajo nekatere zastavice v registru **STATUS**.

2.2.1. Seznam in opisi ukazov v abecednem vrstnem redu:

Pomen uporabljenih simbolov v opisih posameznih ukazov:

k - konstanta
W - delovni register (**W**ork register)
f - pomnilniška lokacija (**f**ile register)
<b> - bit (mesto - pozicija v registru)

d - označba za izbiro cilja (W - 0 ali f - 1)
PC - programski števec
TOS - števec (vrh) sklada
WDT - Watchdog Timer
C, DC, Z - zastavice v **STATUS** registru (**C**arry, **D**ecimal **C**arry, **Z**ero)
[labela] - simbolna označba vrstice (začne se v 1. stolpcu polja1 in ni obvezna)

ADDLW Seštej konstanto in W

Sintaksa:	[labela] ADDLW k
Operandi:	$0 \le k \le 255$
Operacija:	(W) + k $\rightarrow$ (W)
Spreminja status:	C, DC, Z
Opis:	Vsebini registra W se prišteje osem bitna konstanta 'k', rezultat se shrani v register W.

ADDWF Seštej W in f

Sintaksa:	[labela] ADDWF f,d
Operandi:	$0 \le f \le 127$,
	$d \in [0,1]$
Operacija:	(W) + (f) $\rightarrow$ (cilj)
Spreminja status:	C, DC, Z
Opis:	Vsebini registra W se prišteje vsebina registra 'f'. Če je 'd' 0, se rezultat shrani v register W, če pa je 'd' 1, se rezultat shrani v register 'f'.

ANDLW Konstanta IN W

Sintaksa:	[labela] ANDLW k
Operandi:	$0 \le k \le 255$
Operacija:	(W) .IN. k $\rightarrow$ (W)
Spreminja status:	Z
Opis:	Na vsebini registra W je izveden logični bitni IN (AND) z osem bitno konstanto 'k', rezultat se shrani v register W.

ANDWF W IN f

Sintaksa:	[labela] ANDWF f,d
Operandi:	$0 \le f \le 127$,
	$d \in [0,1]$
Operacija:	(W) .IN. (f) $\rightarrow$ (cilj)
Spreminja status:	Z
Opis:	Na vsebini registra W je izveden bitni IN z vsebino registra 'f'. Če je 'd' 0, se rezultat shrani v register W, če pa je 'd' 1, se rezultat shrani v register 'f'.

BCF Zbriši bit b v f

Sintaksa:	[labela] BCF f,b
Operandi:	$0 \le f \le 127$,
	$0 \le b \le 7$
Operacija:	0 $\rightarrow$ (f<b>)
Spreminja status:	Nobenega
Opis:	Bit 'b' v registru 'f' je zbrisan.

BSF Postavi bit b v f

Sintaksa:	[labela] BSF f,b
Operandi:	$0 \le f \le 127$,
	$0 \le b \le 7$
Operacija:	1 $\rightarrow$ (f<b>)
Spreminja status:	Nobenega
Opis:	Bit 'b' v registru 'f' je postavljen.

BTFSS	**Testiraj bit, preskoči če je '1'**
Sintaksa:	[labela] BTFSS f,b
Operandi:	$0 \le f \le 127$, $0 \le b \le 7$
Operacija:	Preskoči naslednji ukaz, če je (f<b>) = 1
Spreminja status:	Nobenega
Opis:	Če je bit 'b' v registru 'f' enak '0', se izvede naslednji ukaz. Če je bit 'b' enak '1', je naslednji ukaz preskočen in se namesto njega izvede NOP, kar povzroči, da se ukaz izvaja dva urina cikla.

BTFSC	**Testiraj bit, preskoči če je '0'**
Sintaksa:	[labela] BTFSC f,b
Operandi:	$0 \le f \le 127$, $0 \le b \le 7$
Operacija:	Preskoči naslednji ukaz, če je (f<b>) = 0
Spreminja status:	Nobenega
Opis:	Če je bit 'b' v registru 'f' enak '1', se izvede naslednji ukaz. Če je bit 'b' enak '0', je naslednji ukaz preskočen in se namesto njega izvede NOP, kar povzroči, da se ukaz izvaja dva urina cikla.

CALL	**Klic podprograma**
Sintaksa:	[labela] CALL k
Operandi:	$0 \le f \le 2047$
Operacija:	(PC) + 1 $\rightarrow$ TOS, k $\rightarrow$ PC<10:0>, (PCLATH<4:3>)$\rightarrow$ (PC<12:11>)
Spreminja status:	Nobenega
Opis:	Kliče podprogram. Najprej je naslov povratka iz podprograma (PC+1) shranjen v sklad. Enajst bitni naslov je naložen v bite PC <10:0>. Zgornja bita PC sta naložena iz PCLATH. Ukaz CALL traja dva cikla.

CLRF	**Zbriši f**
Sintaksa:	[labela] CLRF f
Operandi:	$0 \le f \le 127$
Operacija:	00h $\rightarrow$ (f), 1 $\rightarrow$ Z
Spreminja status:	Z
Opis:	Vsebina registra 'f' je zbrisana in bit (Z) v statusnem registru je postavljen na 1.

CLRW	**Zbriši W**
Sintaksa:	[labela] CLRW
Operandi:	Nima
Operacija:	00h $\rightarrow$ (W), 1 $\rightarrow$ Z
Spreminja status:	Z
Opis:	Vsebina registra W je zbrisana in bit (Z) v statusnem registru je postavljen na 1.

CLRWDT Zbriši Watchdog Timer

Sintaksa: [labela] CLRWDT

Operandi: Nima

Operacija: 00h → WDT,

0 → WDT prescaler,

1 → $\overline{\text{TO}}$

1 → $\overline{\text{PD}}$

Spreminja status: $\overline{\text{TO}}$, $\overline{\text{PD}}$

Opis: Ukaz CLRWDT resetira Watchdog Timer. Resetira tudi njegov prescaler. Statusna bita $\overline{\text{TO}}$ in $\overline{\text{PD}}$ sta postavljena.

COMF Komplement f

Sintaksa: [labela] COMF f,d

Operandi: $0 \le f \le 127$,

$d \in [0,1]$

Operacija: $(\overline{f})$ → (cilj)

Spreminja status: Z

Opis: Izračun eniškega komplementa registra 'f'. Če je 'd' 0, se rezultat shrani v register W, če pa je 'd' 1, se rezultat shrani v register 'f'.

DECF Zmanjšaj f za 1

Sintaksa: [labela] DECF f,d

Operandi: $0 \le f \le 127$,

$d \in [0,1]$

Operacija: (f) - 1 → (cilj)

Spreminja status: Z

Opis: Zmanjšaj register 'f' za 1. Če je 'd' 0, se rezultat shrani v register W, če pa je 'd' 1, se rezultat shrani v register 'f'.

DECFSZ Zmanjšaj f za 1, preskoči če je 0

Sintaksa: [labela] DECFSZ f,d

Operandi: $0 \le f \le 127$,

$d \in [0,1]$

Operacija: (f) - 1 → (cilj);

preskoči, če je rezultat enak 0

Spreminja status: Nobenega

Opis: Zmanjšaj register 'f' za 1. Če je 'd' 0, se rezultat shrani v register W, če pa je 'd' 1, se rezultat shrani v register 'f'.

Če je rezultat enak 1, se izvede naslednji ukaz. Če je rezultat enak 0, je naslednji ukaz preskočen in se namesto njega izvede NOP, kar povzroči, da se ukaz izvaja dva urina cikla.

GOTO Brezpogojni skok

Sintaksa: [labela] GOTO k

Operandi: $0 \le f \le 2047$

Operacija: k → PC<10:0>,

(PCLATH<4:3>)→ (PC<12:11>)

Spreminja status: Nobenega

Opis: GOTO sproži brezpogojni skok (vejitev). Enajst bitni naslov je naložen v bite PC <10:0>. Zgornja bita PC sta naložena iz PCLATH. Ukaz GOTO traja dva cikla.

INCF	**Povečaj f za 1**
Sintaksa:	[labela] INCF f,d
Operandi:	$0 \le f \le 127$,
	$d \in [0,1]$
Operacija:	(f) + 1 $\rightarrow$ (cilj)
Spreminja status:	Z
Opis:	Povečaj register 'f' za 1. Če je 'd' 0, se rezultat shrani v register W, če pa je 'd' 1, se rezultat shrani v register 'f'.

INCFSZ	**Povečaj f za 1, preskoči če je 0**
Sintaksa:	[labela] INCFSZ f,d
Operandi:	$0 \le f \le 127$,
	$d \in [0,1]$
Operacija:	(f) + 1 $\rightarrow$ (cilj);
	preskoči, če je rezultat enak 0
Spreminja status:	Nobenega
Opis:	Poveča register 'f' za 1. Če je 'd' 0, se rezultat shrani v register W, če pa je 'd' 1, se rezultat shrani v register 'f'.
	Če je rezultat enak 1, se izvede naslednji ukaz. Če je rezultat enak 0, je naslednji ukaz preskočen in se namesto njega izvede NOP, kar povzroči, da se ukaz izvaja dva urina cikla.

IORLW	**Konstanta ALI W**
Sintaksa:	[labela] IORLW k
Operandi:	$0 \le k \le 255$
Operacija:	(W) .ALI. k $\rightarrow$ (W)
Spreminja status:	Z
Opis:	Na vsebini registra W je izveden bitni logični ALI (OR) z osem bitno konstanto 'k', rezultat se shrani v register W.

IORWF	**W ALI f**
Sintaksa:	[labela] IORWF f,d
Operandi:	$0 \le f \le 127$,
	$d \in [0,1]$
Operacija:	(W) .ALI. (f) $\rightarrow$ (cilj)
Spreminja status:	Z
Opis:	Na vsebini registra W je izveden bitni logični ALI (OR) z vsebino registra 'f'. Če je 'd' 0, se rezultat shrani v register W, če pa je 'd' 1, se rezultat shrani v register 'f'.

MOVF	**Shrani f**
Sintaksa:	[labela] MOVF f,d
Operandi:	$0 \le f \le 127$,
	$d \in [0,1]$
Operacija:	(f) $\rightarrow$ (cilj)
Spreminja status:	Z
Opis:	Vsebina registra f je pomaknjena v ciljni register. Če je d=0, se rezultat shrani v register W, če pa je d=1, se rezultat shrani v register 'f'. Ukaz z d=1 je običajno uporabljen za testiranje registra f, saj vpliva na statusno zastavico Z.

MOVLW	**Shrani k v W**
Sintaksa:	[labela] MOVLW k
Operandi:	$0 \leq k \leq 255$
Operacija:	$k \rightarrow (W)$
Spreminja status:	Nobenega
Opis:	Osem bitna konstanta 'k' se shrani v register W.

MOVWF	**Shrani W v f**
Sintaksa:	[labela] MOVWF f
Operandi:	$0 \leq k \leq 127$
Operacija:	$(W) \rightarrow (f)$
Spreminja status:	Nobenega
Opis:	Vsebina registra W se shrani v register f.

NOP	**Ni operacije**
Sintaksa:	[labela] NOP
Operandi:	Nima
Operacija:	Ni operacije
Spreminja status:	Nobenega
Opis:	ni operacije, traja 1 cikel

RETFIE	**Vrni se iz interrupta**
Sintaksa:	[labela] RETFIE
Operandi:	Nima
Operacija:	TOS $\rightarrow$ PC, 1 $\rightarrow$ GIE
Spreminja status:	Nobenega
Opis:	Vrnitev iz prekinitvene rutine (interrupt).

RETLW	**RETURN s konstanto v W**
Sintaksa:	[labela] RETLW k
Operandi:	$0 \leq k \leq 255$
Operacija:	$k \rightarrow (W)$, TOS $\rightarrow$ PC
Spreminja status:	Nobenega
Opis:	V register W se naloži konstanta 'k'. V programski števec se naloži vrednost z vrha sklada (naslov za vrnitev iz podprograma). Operacija traja dva cikla.

RETURN	**Vrnitev iz podprograma**
Sintaksa:	[labela] RETURN
Operandi:	Nima
Operacija:	TOS $\rightarrow$ PC
Spreminja status:	Nobenega
Opis:	Vrnitev iz podprograma. V programski števec se naloži vrednost z vrha sklada (naslov za vrnitev iz podprograma). Operacija traja dva cikla.

RLF — Rotiraj f v levo skozi Carry

Sintaksa: [labela] RLF f,d

Operandi: $0 \le f \le 127$,

$d \in [0,1]$

Operacija:

Spreminja status: C

Opis: Vsebina registra f je pomaknjena za eno mesto v levo skozi zastavico statusnega registra Carry. Če je d=0, se rezultat shrani v register W, če pa je d=1, se rezultat shrani v register 'f'.

RRF — Rotiraj f v desno skozi Carry

Sintaksa: [labela] RRF f,d

Operandi: $0 \le f \le 127$,

$d \in [0,1]$

Operacija:

Spreminja status: C

Opis: Vsebina registra f je pomaknjena za eno mesto v desno skozi zastavico statusnega registra Carry. Če je d=0, se rezultat shrani v register W, če pa je d=1, se rezultat shrani v register 'f'.

SLEEP — SLEEP način delovanja

Sintaksa: [labela] SLEEP

Operandi: Nima

Operacija: $00h \rightarrow WDT$,

$0 \rightarrow$ WDT prescaler,

$1 \rightarrow \overline{TO}$

$0 \rightarrow \overline{PD}$

Spreminja status: $\overline{TO}$, $\overline{PD}$

Opis: Statusni bit $\overline{PD}$, ki označuje status napajanja je zbrisan, statusni bit $\overline{TO}$, ki označuje pretek časa, je postavljen. Watchdog Timer in njegov »prescaler« (preddelilnik) sta zbrisana. Procesor se preklopi v način delovanja SLEEP z zaustavljenim oscilatorjem.

SUBLW — Odštej W od konstante

Sintaksa: [labela] SUBLW k

Operandi: $0 \le k \le 255$

Operacija: $k - (W) \rightarrow (W)$

Spreminja status: C, DC, Z

Opis: Vsebina registra W se odšteje (po metodi dvojiškega komplementa) od osem bitne konstante 'k', rezultat se shrani v register W.

SUBWF	**Odštej W od f**
Sintaksa:	[labela] SUBWF f,d
Operandi:	$0 \le f \le 127$, $d \in [0,1]$
Operacija:	(f) - (W) → (cilj)
Spreminja status:	C, DC, Z
Opis:	Vsebina registra W se odšteje (po metodi dvojiškega komplementa) od vsebine registra 'f'. Če je 'd' 0, se rezultat shrani v register W, če pa je 'd' 1, se rezultat shrani v register 'f'.

SWAPF	**Zamenjaj »nibbla« v f**
Sintaksa:	[labela] SWAPF f,d
Operandi:	$0 \le f \le 127$, $d \in [0,1]$
Operacija:	(f<3:0>) → (cilj<7:4>), (f<7:4>) → (cilj<3:0>)
Spreminja status:	-
Opis:	Zgornji in spodnji »nibble« (štirje biti) sta zamenjana. Če je 'd' 0, se rezultat shrani v register W, če pa je 'd' 1, se rezultat shrani v register 'f'.

XORLW	**Konstanta XOR W**
Sintaksa:	[labela] XORLW k
Operandi:	$0 \le k \le 255$
Operacija:	(W) .XOR. k → (W)
Spreminja status:	Z
Opis:	Na vsebini registra W je izveden bitni logični ekskluzivni ALI (XOR) z osem bitno konstanto 'k', rezultat se shrani v register W.

XORWF	**W IN f**
Sintaksa:	[labela] XORWF f,d
Operandi:	$0 \le f \le 127$, $d \in [0,1]$
Operacija:	(W) .XOR. (f) → (cilj)
Spreminja status:	Z
Opis:	Na vsebini registra W je izveden bitni bitni logični ekskluzivni ALI (XOR) z vsebino registra 'f'. Če je 'd' 0, se rezultat shrani v register W, če pa je 'd' 1, se rezultat shrani v register 'f'.

Za razumevanje tematike je potrebno tudi predznanje iz osnov digitalne elektronike (številski sistemi, logične operacije) [3] in osnov programiranja v zbirnem jeziku: [2] in [8]. Dobri opisi programskih ukazov PIC mikrokrmilnikov srednje kategorije so na voljo v [2] in [6] ter na spletnem portalu: http://www.interq.or.jp/japan/se-inoue/e_pic.htm.

Vprašanja za utrjevanje:

S katerim ukazom postavimo (na 1) bite 0, 1 in 2 v delovnem registru W ?

Rešitev:

```
IORLW b'00000111' ;'ALI' log. oper. med istoleznimi biti, visjih 5 bitov v W se ohrani,
                  ;spodnji trije biti v W se postavijo na 1
```

1. *Koliko različnih ukazov je na voljo ?*
2. *Kakšna polja vsebuje programska vrstica v zbirnem jeziku ?*
3. *S katerim ukazom zbrišemo bit 3 v registru f na naslovu 0x2F ?*
4. *S katerim ukazom opravimo logični IN (AND) med registrom W in f na naslovu 0x22 ?*
5. *S katerim ukazom zaključimo podprogram ?*
6. *S katerim ukazom zbrišem bite 5, 6, in 7 v registru W ?*

2.2.2. Kratek opis ukazov mikrokrmilnika PIC 16F87x

ukaz, operand	opis ukaza	STATUS	cikli
UKAZI, KI SO POVEZANI Z ZLOGI (BYTE)			
ADDWF f, d	seštej **W** in register **f**	C, DC, Z	1
ANDWF f, d	»logična in« operacija med **W** in registrom **f**	Z	1
CLRF f	register **f** naj dobi vrednost 0	Z	1
CLRW -	register **W** naj dobi vrednost 0	Z	1
COMF f, d	eniški komplement registra **f** (zamenjava 0 in 1)	Z	1
DECF f, d	zmanjšaj za ena vsebino registra **f**	Z	1
DECFSZ f, d	zmanjšaj vsebino reg. **f** za ena in preskoči naslednji ukaz, če **f**=0	-	1(2)
INCF f, d	povečaj za ena vsebino registra **f**	Z	1
INCFSZ f, d	povečaj vsebino reg. **f** za ena in preskoči naslednji ukaz, če **f**=0	-	1(2)
IORWF f, d	»logična ali« operacija med **W** in registrom f	Z	1
MOVF f, d	kopiraj vsebino registra **f** (d=0 ->**W**; d=1 ->**f**)	Z	1
MOVWF f	kopiraj vsebino **W** v register **f**	-	1
NOP -	ne naredi ničesar, potroši 1 urin cikel	-	1
RLF f, d	krožno premakni vse bite v registru **f** za eno mesto v levo	C	1
RRF f, d	krožno premakni vse bite v registru **f** za eno mesto v desno	C	1
SUBWF f, d	odštej vsebino reg. **W** od vsebine v reg. **f** (d=**f**-**W**)	C, DC, Z	1
SWAPF f, d	zamenjaj zgornje in spodnje 4 bite v registru **f**	-	1
XORWF f, d	»izključni ali« logična operacija med **W** in reg. **f**	Z	1
UKAZI, KI SO POVEZANI Z BITI			
BCF f, b	briši bit b v registru **f**	-	1
BSF f, b	postavi bit b v registru **f**	-	1
BTFSC f, b	testiraj bit b v registru **f** in preskoči naslednji ukaz, če je bit b=0	-	1(2)
BTFSS f, b	testiraj bit b v registru **f** in preskoči naslednji ukaz, če je bit b=1	-	1(2)
SISTEMSKI IN UKAZI, KI SO POVEZANI S KONSTANTAMI			
ADDLW k	registru **W** prištej konstanto **k**	C, DC, Z	1
ANDLW k	»logična in« operacija med **W** in konstanto **k**	Z	1
CALL k	klic podprograma	-	2
CLRWDT -	vsebina »časovnega stražnika« (**W**atch **D**og **T**imer) naj bo enaka nič	TO, PD	1
GOTO k	skoči na naslov **k** (11-bitni naslov)	-	2
IORLW k	»logična ali« operacija med **W** in konstanto **k**	Z	1
MOVLW k	vpiši konstantno vrednost k v **W** (**W**=k)	-	1
RETFIE -	vrni se iz podprograma za strežbo prekinitvene zahteve	-	2
RETLW k	vrni se iz podprograma s konstantno vrednostjo **k** v registru **W**	-	2
RETURN -	vrni se iz podprograma	-	2
SLEEP -	postavi mikrokrmilnik v stanje mirovanja - »sleep« način	TO, PD	1
SUBLW k	odštej vsebino **W** od konstantne vrednosti **k** (**W**=k-**W**)	C, DC, Z	1
XORLW k	»izključno ali« logična operacija med **W** in konstanto **k**	Z	1

Tabela 2-1: Tabela ukazov s kratkimi opisi

f - ime ali naslov datotečnega registra (**f**ile register)

k - konstantna (8-bitna) vrednost med 0 in 255

d - cilj hrambe rezultata operacije (d=0 - shrani se v **W**, d=1 – shrani se v **f**)

b – pozicija ali oznaka bita med 0 in 7

1(2) – trajanje ukaza je 1 (ali 2) ukazna cikla (pri 20 MHz taktu je trajanje enega ukaznega cikla (4*50) ns = 0,2 µs)

3. Organizacija pomnilnika

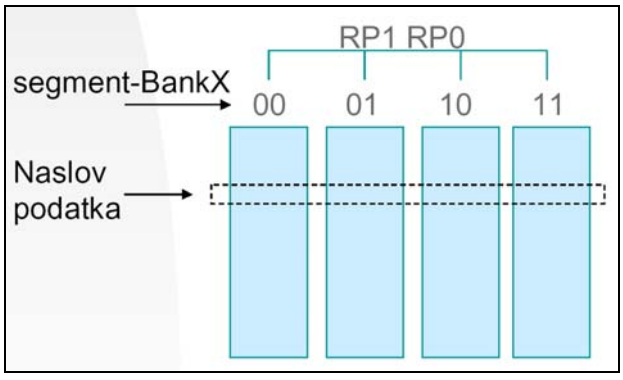
*Spoznali boste model in organizacijo podatkovnega pomnilnika (segmenti) in model programskega pomnilnika ter v zvezi s tem najpomembnejši datotečni register **STATUS**. Naučili se boste ločevati med podatkovnim pomnilnikom (RAM) in programskim pomnilnikom (FLASH ROM).*

3.1. Podatkovni pomnilnik (namenski in splošni registri)

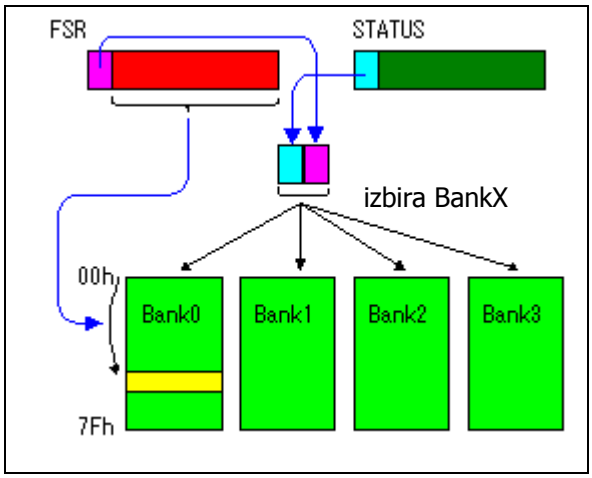
Podatkovni pomnilnik (Slika 3-1) mikrokrmilnikov PIC16F87x [6][8][16] je **razdeljen na štiri segmente** (Bank0, Bank1, Bank2, Bank3). Vsak od segmentov obsega do **128 zlogov** (8-bitnih besed) ali lokacij – datotečni registri (**F**ile registers) z naslovi od 0 do 7Fh (Slika 3-2).

Datotečne registre SFR (Slika 3-3), ki zasedajo začetni del (pri prvih dveh segmentih na naslovih: 0 do 1Fh), imenujemo tudi **namenski funkcijski registri** (angl.: **S**pecial **F**unction **R**egisters), ker so **namenjeni posebnim funkcijam**: sistemski registri in registri vhodno/izhodnih vmesnikov. Na voljo je nekaj **nad 50** različnih **SFR** registrov, vsak pa ima specifično (rezervirano) ime (npr.: **PORTB**), katero lahko uporabimo kot **operand pri programiranju**. Nekateri (najpomembnejši) registri (npr.: **STATUS**, **FSR**) se pojavljajo v vseh štirih segmentih.

Datotečne registre GPR (Slika 3-3), ki zasedajo drugi del (pri prvih dveh segmentih na naslovih: 20h do 7Fh) imenujemo tudi **splošno uporabni registri** (angl.: **G**eneral **P**urpose **R**egisters), ker so **namenjeni splošni funkciji**, kot je shranjevanje spremenljivk. V vseh štirih segmentih je pri PIC16F876/877 na voljo **368 RAM pomnilniških lokacij**, pri čemer se zadnjih 16 lokacij (naslovi od 70h do 7Fh) pojavi v vseh štirih segmentih.



Slika 3-1: Segmentacija pomnilnika



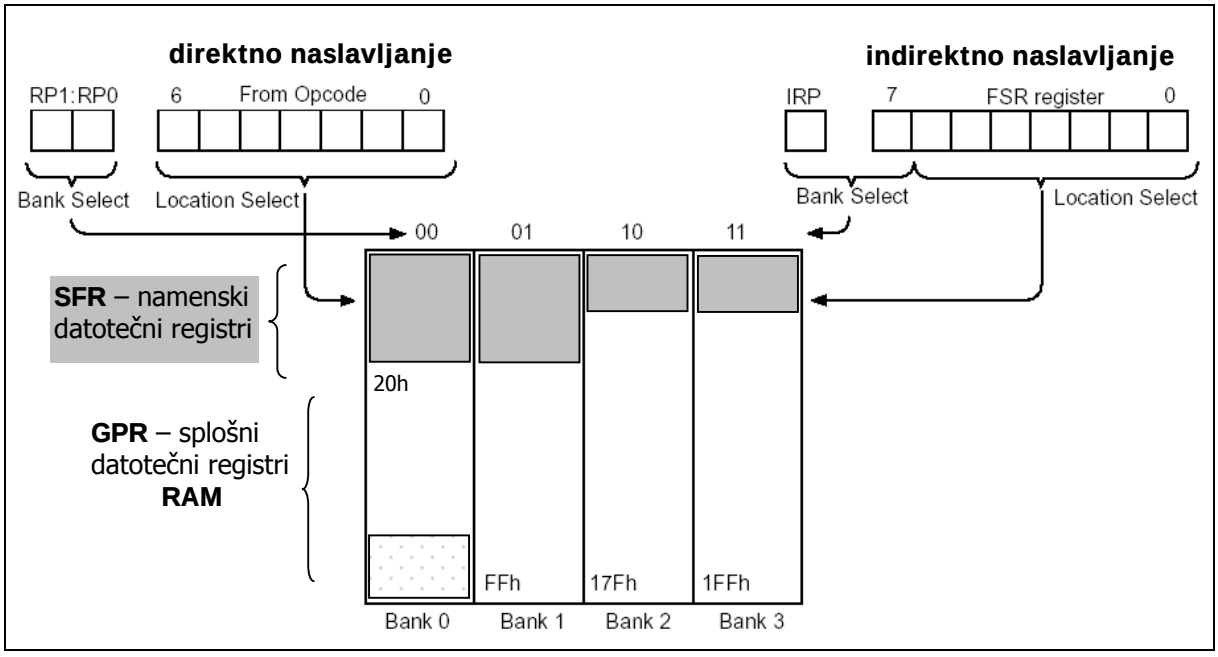
Slika 3-2: Izbira registrov v segmentih

Pri **»direktnem« načinu naslavljanja** (Slika 3-1, Slika 3-2, Slika 3-3) imamo programski dostop do posameznega **SFR** registra preko imena **f** registra, dostop do **GPR** registra pa preko izbire naslova (med 1Fh in 7Fh). **Pred tem moramo izbrati**

ustrezni segment, kar dosežemo z vpisom 2-bitne informacije v register **STATUS** (bita RP1, RP0).

Pri **»indirektnem« načinu naslavljanja** (Slika 3-2, Slika 3-3) imamo enoten programski dostop do **SFR** in **GPR** registrov preko izbire naslova (med 0Fh in FFh) s pomočjo »indeksnega registra« **FSR**. **Pred tem moramo izbrati ustrezni par segmentov**, kar dosežemo z vpisom 1-bitne informacije v register **STATUS** (bit z oznako IRP).

Za (manj vešče) uporabnike je na začetku vsekakor **priporočljivo uporabljati »direktni« način naslavljanja**.



Slika 3-3: Nazornejši prikaz izbire segmentov in naslavljanja registrov

3.1.1. Podroben prikaz segmentne organizacije podatkovnega pomnilnika

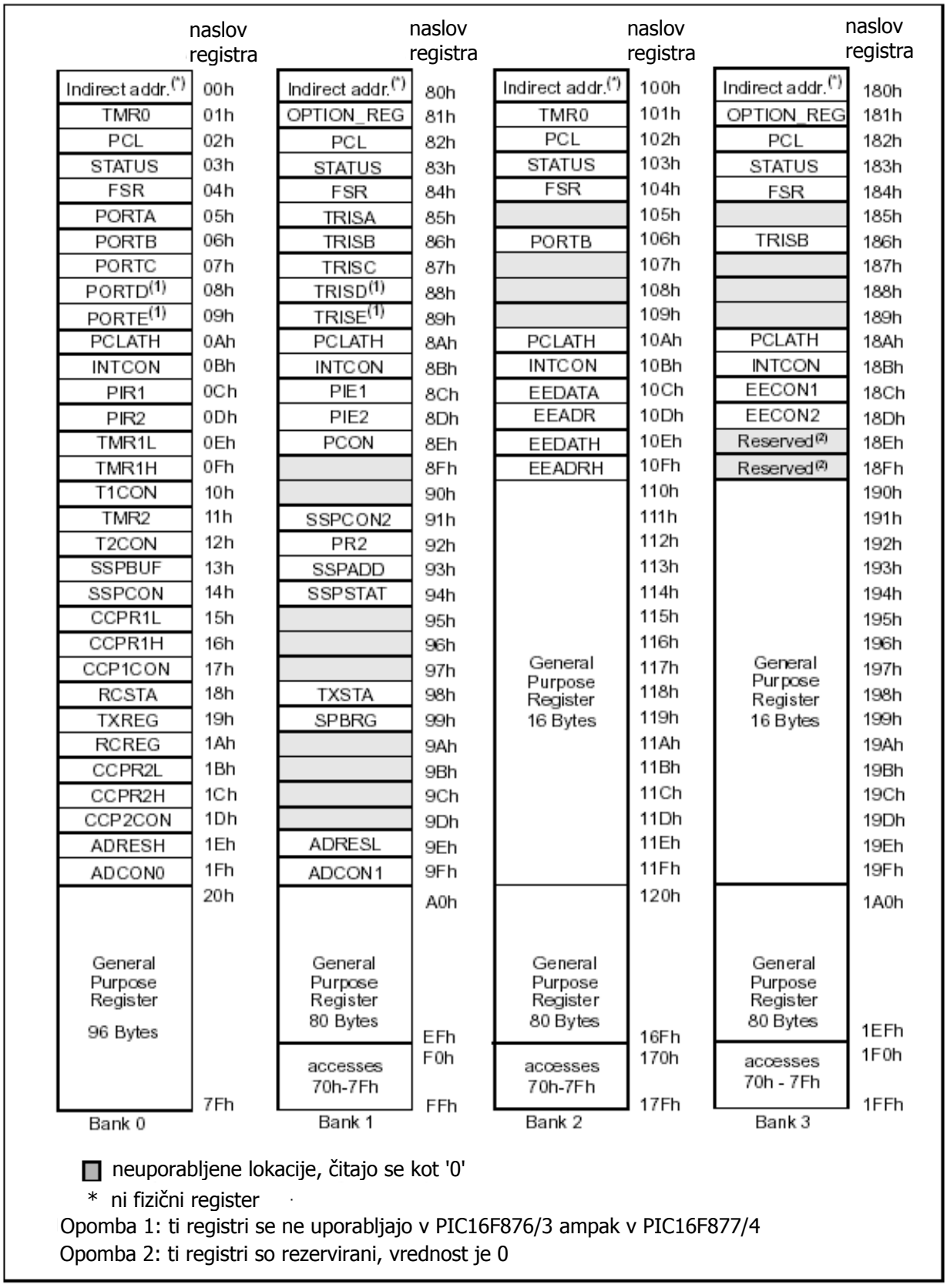


Tabela 3-1: Razporeditev registrov po pomnilniških segmentih

3.1.2. Register STATUS

Register **STATUS** vsebuje bitni status aritmetične logične enote (ALU), resetni status in kontrolne **bite za izbiro segmenta (banke)** v pomnilniku. Statusni register lahko uporabimo kot cilj v katerem koli ukazu, če je ukaz takšen, da vpliva na bite Z, C ali DC; le-teh ni mogoče spreminjati, bitov $\overline{\text{TO}}$ in $\overline{\text{PD}}$ pa ni mogoče prepisati.

Naslov	Ime	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Vrednost pri: POR,BOR	Vrednost pri vseh drugih resetih
03h	STATUS	IRP	RP1	RP0	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C	0001 1xxx	000q quuu

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	**RP1**	**RP0**	**TO**	**PD**	**Z**	**DC**	**C**
bit 7							bit 0

Legenda: R = omogočeno branje bita, W = omogočeno pisanje v bit, U = neuporabljen bit, beri kot '0', -n = vrednost ob POR resetu

bit 7: **IRP:** Bit za izbiro bank registrov **(**Register Bank Select bit, uporabljen za posredno naslavljanje)
- 1 = Bank 2, 3 (100h - 1FFh)
- 0 = Bank 0, 1 (00h - FFh)

bit 6-5: **RP1:RP0**: Bit za izbiro banke registrov **(**Register Bank Select bits, uporabljen za neposredno naslavljanje)
- **00 = Bank 0 (00h - 7Fh)**
- **01 = Bank 1 (80h - FFh)**
- 10 = Bank 2 (100h - 17Fh)
- 11 = Bank 3 (180h - 1FFh)

Vsaka segment (Bank) ima maksimalno 128 zlogov (bytov).

bit 4: $\overline{\text{TO}}$: Bit za prekoračitev časa (Time-out bit)
- 1 = Po vklopu napajanja, ukazu `CLRWDT ali SLEEP`
- 0 = Pojavila se je prekoračitev časa WDT

bit 3: $\overline{\text{PD}}$: Bit za izklop napajanja (Power-down bit)
- 1 = Po vklopu ali z ukazom `CLRWDT`
- 0 = Z izvedbo ukaza `SLEEP`

bit 2: **Z**: Ničelni bit (Zero bit)
- 1 = rezultat aritmetične ali logične operacije je nič
- 0 = rezultat aritmetične ali logične operacije ni nič

bit 1: **DC**: Bit za prenos med nibbloma (Digit Carry/$\overline{\text{borrow}}$ bit, ukazi `ADDWF, ADDLW,SUBLW,SUBWF`)
(za $\overline{\text{borrow}}$ je polariteta obrnjena)
- 1 = Pojavil se je prenos s četrtega bita.
- 0 = Prenos s četrtega bita se ni pojavil.

bit 0: **C**: Bit za prenos/$\overline{\text{izposojo}}$ (Carry/$\overline{\text{borrow}}$ bit, ukazi `ADDWF, ADDLW, SUBLW, SUBWF`)
- 1 = Prišlo je do prenosa iz najvišjega bita (MSB)
- 0 = Ni prišlo do prenosa iz najvišjega bita (MSB)

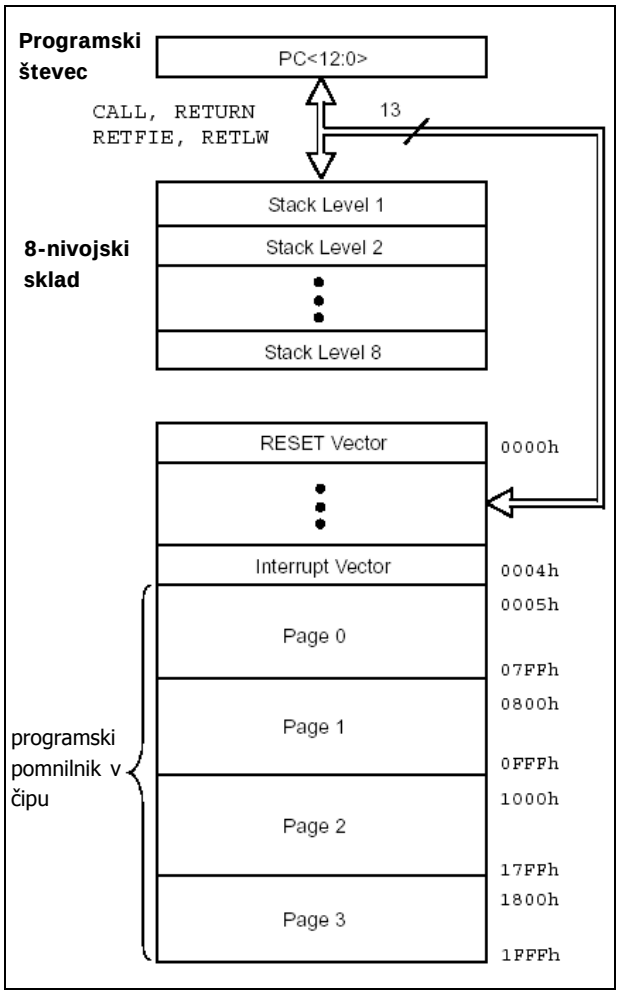
Opomba: Za $\overline{\text{borrow}}$ je polariteta obrnjena. Odštevanje je izvedeno s prištevanjem dvojiškega komplementa drugega operanda. Za ukaze za rotacije (`RRF`, `RLF`) se ta bit naloži z bodisi najvišjim ali najnižjim bitom registra, ki ga rotiramo.

3.2. Programski pomnilnik

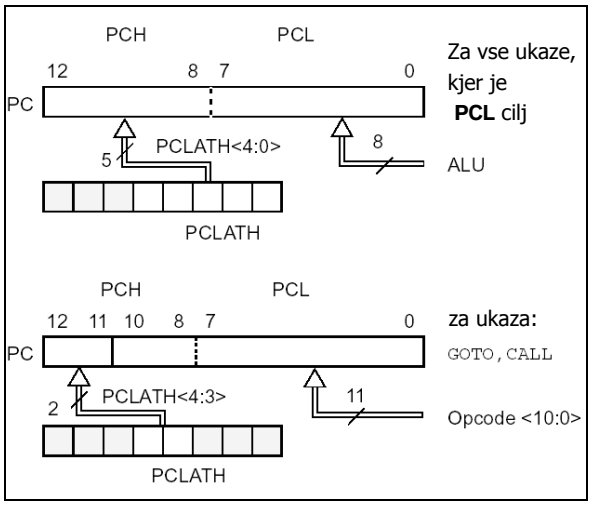
Programski pomnilnik (Slika 3-4) v mikrokrmilniku PIC16F876/877 [6][8][16] vsebuje **8k besed** (14-bitne celice FLASH ROM), kamor se **naloži** (programira, »zapeče«) zaporedje **strojnih ukazov - program**. Pomnilnik je sicer razdeljen na segmente (strani, angl.: Page) po 2k, vendar to ne predstavlja večjih težav, razen v primeru, ko se izvajajo skoki izven segmenta.

Za vsak program je pomembno, da se **prvi ukaz nahaja na začetnem naslovu 0** (**RESET Vector**), kajti procesor začne ob vklopu napajanja (Reset) izvajati program prav iz začetne lokacije.

Pomemben je še **prekinitveni vektor** (**Interrupt Vector**), ki se nahaja **na naslovu 4**. Če so prekinitve omogočene, se mora na navedenem naslovu nahajati **1. ukaz prekinitvenega strežnega programa** (angl.: Interrupt Service Routine).



Slika 3-4: Mapa programskega pomnilnika in sklada



Slika 3-5: Programski števec iz dveh delov

Sklad (angl.: Stack) procesne enote PIC mikrokrmilnikov je posebnost, kajti na voljo je le 8 nivojev (angl.: Stack Level) za shranjevanje trenutne vrednosti (13 bitov) programskega števca **PC** (Program Counter) ob **skokih v podprograme** ali ob **prekinitvenih zahtevah**.

Programski števec (PC) je sicer 13-bitni, vendar se v 8-bitnih **SFR** registrih podatkovnega pomnilnika hrani ločeno višjih 5 bitov (**PCH**) in nižjih 8 bitov (**PCL**). Kot pomožni register za polnjenje **PCH** (Slika 3-5) služi »zadrževalni register« **PCLATH**.

Za razumevanje tematike je potrebno tudi predznanje iz osnov digitalne elektronike in gradnikov mikroprocesorskih sistemov (pomnilniški elementi), kar je na voljo med drugim v [3]. Dobri opisi zgradbe in opisa pomnilniškega modela PIC mikrokrmilnikov so tudi v [2] in [6] ter na spletnem portalu: http://www.interq.or.jp/japan/se-inoue/e_pic.htm.

Vprašanja za utrjevanje:

Napišite zaporedje ukazov za izbiro segmenta Bank1

Rešitev:

```
BCF    STATUS,RP1    ; 0 → RP1 (brisi bit 6 v reg. STATUS)
BSF    STATUS,RP0    ; 1 → RP0 (postavi bit 5 v reg. STATUS)
```

1. *Napišite zaporedje ukazov za izbiro segmenta Bank2*
2. *Kakšen pomen imajo zastavice v registru STATUS ?*
3. *Kateri pomnilniški segment (Bank0, 1, 2, 3) se največ uporablja ?*
4. *Koliko bitne so celice v programskem pomnilniku ?*
5. *Koliko nivojev ima sklad (stack) ?*
6. *S katerega naslova v programskem pomnilniku se začne izvajati program ?*
7. *Kateri datotečni registri se pojavijo v vseh štirih segmentih ?*

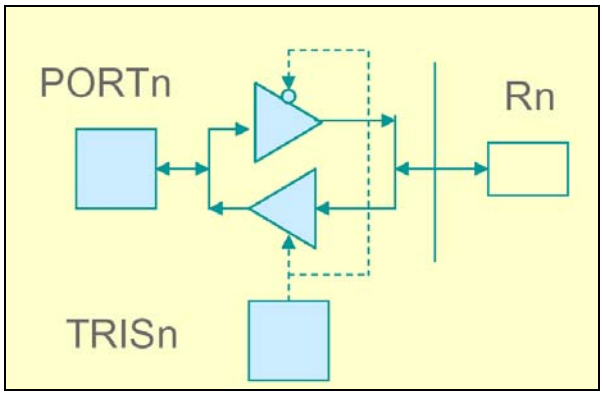
4. Osnovne vhodno/izhodne enote

Spoznali boste osnovne vhodno/izhodne enote – digitalne (binarne) ali logične vhode/izhode. Naučili se boste konfigurirati posamezne priključke vrat A, B in C ter programsko krmiliti izhode in ugotavljati stanja vhodov.

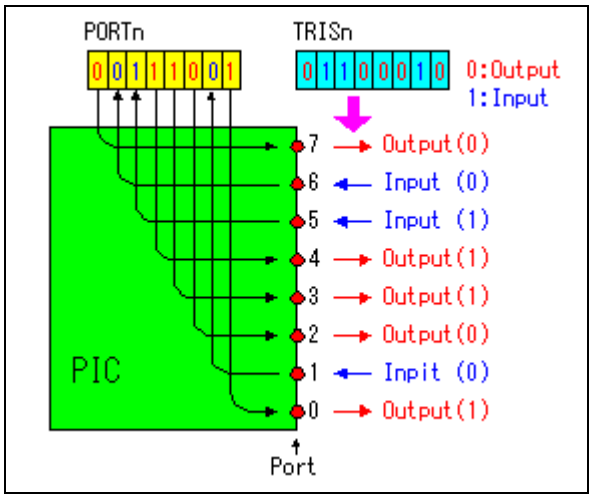
Mikrokrmilnik PIC16F876 (Slika 1-5) ima tri vhodno/izhodna vrata (angl.: port): **PORTA**, **PORTB** in **PORTC**, preko katerih so dostopni vhodno/izhodni signali vsebovanih vmesnikov. Vrednosti na priključkih lahko zajemamo oz. spreminjamo s čitanjem/vpisovanjem vrednosti istoimenskih registrov. Priključke je potrebno predhodno programsko konfigurirati (izbrati vhod/izhod) z nastavitvijo vrednosti v registrih **TRISA**, **TRISB** in **TRISC**. Ta postopek imenujemo tudi »inicializacija«.

4.1. Vrata PORTA, PORTB in PORTC

Osnovni način delovanja (Slika 4-1) **PIC mikrokrmilnikov** je, da se linije (priključki) vrat **PORTn**: **PORTA** (RA0 do RA5), **PORTB** (RB0 do RB7) in **PORTC** (RC0 do RC7) **uporabljajo kot digitalni (binarni, logični, preklopni) vhodi ali izhodi**. Določanje orientacije (Slika 4-2) ali smeri delovanja (**0: izhod** - Output, **1: vhod** - Input) poteka z vpisom bitne informacije v ustrezni register **TRISn**, stanje vhodne/izhodne linije pa določamo s čitanjem/vpisom v bitne celice pripadajočega registra **PORTn**.



Slika 4-1: Poenostavljena zgradba elementa vrat



Slika 4-2: Primer konfiguriranja priključkov vrat

Upoštevati je potrebno [8], da je **večina vhodno/izhodnih priključkov multipleksirana** (deljena raba) s preostalimi funkcijami mikrokrmilnika (analogni vhodi, Timer0,1,2, UART, idr.).

4.1.1. PORTA

PORTA ima 6 vhodno/izhodnih priključkov, katerih obnašanje (logični vhod ali izhod) nastavljamo z registrom **TRISA**. Postavitev ustreznega **bita TRISA na 1** bo iz priključka na **PORTA naredila vhod**, **postavitev na 0 pa izhod**. Priključek **RA4** je multipleksiran z vhodom časovnika Timer0. **Ostali priključki PORTA** so **multipleksirani z analognimi vhodi** oziroma z analognima referenčnima vhodoma VREF. Delovanje priključkov je določeno z nastavitvijo/brisanjem ustreznih bitov v registru **ADCON1**. Register **TRISA** določa smer delovanja RA priključkov tudi, ko so le-ti konfigurirani kot analogni vhodi, torej je treba zagotoviti, da so priključki tudi v tem primeru orientirani kot vhodi (ustrezni biti v **TRISA** so postavljeni na 1).

Ob vklopu se priključki konfigurirajo kot analogni vhodi in imajo vrednost 0.

Funkcije PORTA:

Ime	Bit#	Buffer	Funkcija
RA0/AN0	bit0	TTL	Vhod/izhod ali analogni vhod
RA1/AN1	bit1	TTL	Vhod/izhod ali analogni vhod
RA2/AN2	bit2	TTL	Vhod/izhod ali analogni vhod
RA3/AN3/VREF	bit3	TTL	Vhod/izhod ali analogni vhod ali VREF
RA4/TOCKI	bit4	ST	Vhod/izhod ali vhod za zunanji takt za Timer0 Izhod je z odprtim ponorom
RA5/$\overline{SS}$/AN4	bit5	TTL	Vhod/izhod ali analogni vhod

Legenda: TTL = TTL vhod, ST = vhod s Schmittovim Triggerjem

S **PORTA** povezani registri:

Naslov	Ime	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Vrednost ob POR, BOR	Vrednost ob drugih resetih
05h	**PORTA**	-	-	RA5	RA4	RA3	RA2	RA1	RA0	--0x 0000	--0u 0000
85h	**TRISA**	-	-	**PORTA** register smeri podatkov						--11 1111	--11 1111
9Fh	**ADCON1**	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0	--0- 0000	--0- 0000
1Fh	**ADCON0**	ADCS1	ADCS0	CHS2	CHS1	CHS0	Go/Done	-	ADON	000 00-0	0000 00-0

Legenda: x = neznan, u = nespremenjen, - = neuporabljene lokacije, beri kot '0', PORTA ne uporablja osenčenih lokacij

4.1.2. PORTB

PORTB ima 8 vhodno/izhodnih priključkov, katerih obnašanje (logični vhod ali izhod) nastavljamo z registrom **TRISB**. Postavitev ustreznega bita **TRISB** na 1 bo iz priključka na **PORTB** naredila vhod, postavitev na **0** pa **izhod**. Trije biti so multipleksirani s funkcijo za programiranje pri nizki napetosti (**RB3/PGM)** in signaloma za komunikacijo z **ICD** modulom: **RB6/PGC** in **RB7/PGD**). Priključek **RB0** je multipleksiran (podvojen) z zunanjim prekinitvenim vhodom **INT** (Interrupt).

Funkcije PORTB:

Ime	Bit#	Buffer	Funkcija
RB0/INT	bit0	TTL/ST	Vhod/izhod ali zunanji priključek za »interrupt«
RB1	bit1	TTL	Vhod/izhod
RB2	bit2	TTL	Vhod/izhod
RB3/PGM	bit3	TTL	Vhod/izhod ali priključek za programiranje pri nizki napetosti
RB4	bit4	TTL	Vhod/izhod
RB5	bit5	TTL	Vhod/izhod
RB6/PGC	bit6	TTL/ST	Vhod/izhod ali priključek za programiranje pri nizki napetosti
RB7/PGD	bit7	TTL/ST	Vhod/izhod ali priključek za programiranje pri nizki napetosti

Legenda: TTL = TTL vhod, ST = vhod s Schmittovim Triggerjem

S **PORTB** povezani registri:

Naslov	Ime	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Vrednost ob POR, BOR	Vrednost ob drugih resetih
06h, 106h	**PORTB**	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
86h, 186h	**TRISB**	PORTB register smeri podatkov								1111 1111	1111 1111
81h, 181h	**OPTION_REG**	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legenda: x = neznan, u = nespremenjen, - = neuporabljene lokacije, beri kot '0', PORTB ne uporablja osenčenih lokacij

4.1.3. PORTC

PORTC ima 8 vhodno/izhodnih priključkov, katerih obnašanje (logični vhod ali izhod) nastavljamo z registrom **TRISC**. Postavitev ustreznega bita **TRISC** na **1** bo iz priključka na **PORTC** naredila **vhod**, postavitev na **0** pa **izhod**.

Funkcije PORTC:

Ime	Bit#	Buffer	Funkcija
RC0/T1OSO/T1CKI	bit0	ST	Vhod/izhod ali izhod oscilatorja za Timer1 / vhod takta za Timer1
RC1/T1OSI/CCP2	bit1	ST	Vhod/izhod ali vhod oscilatorja za Timer1 ali vhod Capture2 / izhod Compare 2 / izhod PWM2
RC2/CCP1	bit2	ST	Vhod/izhod ali vhod Capture1 / izhod Compare 1 / izhod PWM1
RC3/SCK/SCL	bit3	ST	Vhod/izhod ali sinhronski serijski CLOCK za SPI in I^2C
RC4/SDI/SDA	bit4	ST	Vhod/izhod ali podatkovni vhod SPI / I^2C
RC5/SDO	bit5	ST	Vhod/izhod ali sinhronski serijski podatkovni izhod
RC6/TX/CK	bit6	ST	Vhod/izhod ali USART asinhronski Transmit ali sinhronski clock
RC7/RX/DtT	bit7	ST	Vhod/izhod ali USART asinhronski Receive ali sinhronski podatki

Legenda: TTL = TTL vhod, ST = vhod s Schmittovim Triggerjem

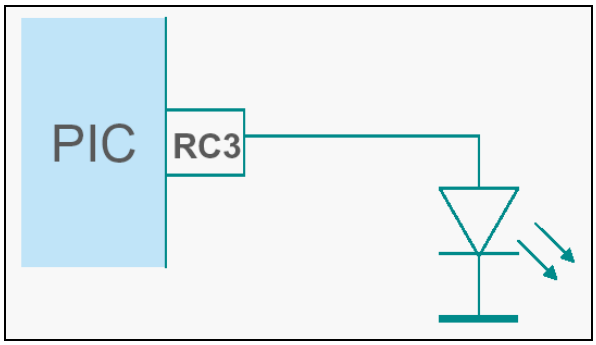
S **PORTC** povezani registri:

Naslov	Ime	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Vrednost ob POR, BOR	Vrednost ob drugih resetih
07h	**PORTC**	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
87h	**TRISC**	PORTC register smeri podatkov								1111 1111	1111 1111

Legenda: x = neznan, u = nespremenjen

4.2. Preprosti zgledi programov za binarne digitalne vhode/izhode

Vzemimo **zgled: krmiljenje svetlobnega indikatorja** (LED dioda), ki je priključen na logični izhod **RC3** (Slika 4-3, dejansko je potrebno dodati zaporedni upor npr.: 330 Ω za omejitev toka na pribl. 10 mA) [3].



Slika 4-3: Priključitev LED diode na izhod RC3

Poglejmo si izsek programa (Program 4-1), ki opravi inicializacijo (**RC3** kot izhod) in zatem vklopi (**RC3** postavi na 1) oziroma izklopi LED diodo (**RC3** postavi na 0):

```
; Inicializacija: vrata PORTC (RC3 izhod) in postavitev zacetnega stanja na RC3 izhod
       movlw  b'10011111'    ;0x9f->W, priprava maskirne konstante za brisanje bitov:6,5
       andwf  STATUS,f       ;bitni AND, 0->RP1,RP0, izbira Bank0 zaradi dostopa do PORTA
       bcf    PORTC,3        ;zacetno stanje RC3=0, ker stanja ob RESET-u niso definirana!
       bsf    STATUS,RP0     ;1->RP0, kodna stran Bank1 zaradi dostopa do TRISC registra
       bcf    TRISC,3        ;0->PORTC.3, RC3: izhod
; Postavitev stanja na vrata PORTC, bit 3
       bcf    STATUS,RP0     ;kodna stran Bank0 zaradi dostopa do PORTC registra!
ponovi bsf    PORTC,3        ;1->RC3, vklopi LED
; Brisanje stanja na izhodu RC3
       bcf    PORTC,3        ;0->RC3, izklopi LED
       goto   ponovi         ;neskoncna zanka
```

Program 4-1: Vklop/izklop LED diode v zbirnem jeziku

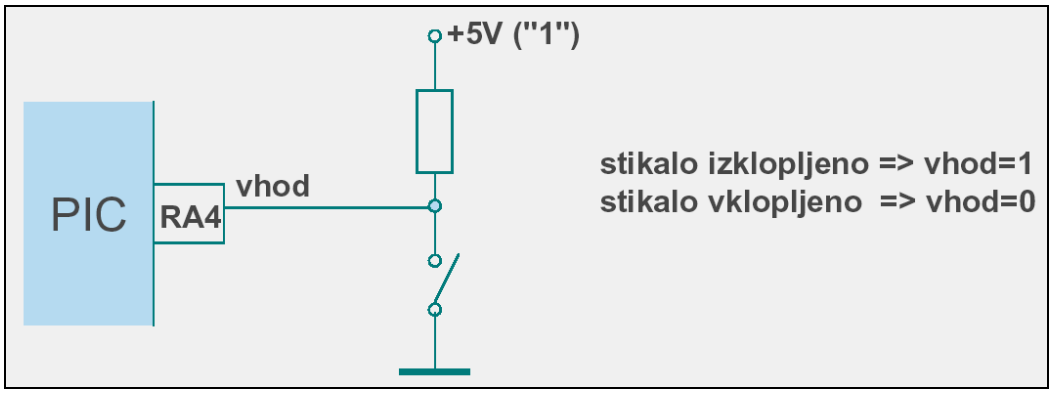
V zgornjem primeru **prvi trije ukazi niso nujno potrebni**, koristni pa so v primeru, ko stanja izhodov (odvisno od naprav, ki so nanje priključene) niso definirana ob prvem vklopu mikrokrmilnika.

Za ilustracijo si še poglejmo, kako je lahko kodiran **enak algoritem v C-jeziku** (Program 4-2) za prevajalnik HI-TEC PICC:

```
main()
{
// Inicializacija: vrata PORTC (RC3 izhod) in postavitev zacetnega stanja na RC3 izhod
PORTC=PORTC&0xf7;     //bitni AND med PORTC in '11110111', zacetno stanje RC3=0,
//ker stanja ob RESET-u niso definirana!
TRISC=0xF7;           //'11110111' -> TRISC, RC3 je izhod
       while(1)       //neskoncna zanka
       {              //zacetek zanke
// Postavitev stanja na vrata PORTC, bit 3
       RC3=1; //1->RC3, vklopi LED
// Brisanje stanja na izhodu RC3
       RC3=0;         //0->RC3, izklopi LED
       }              //konec zanke
}
```

Program 4-2: Vklop/izklop LED diode v C-jeziku

Vzemimo **zgled: program za ugotavljanje stanja stikala**, ki je priključeno na vhod **RA4** (Slika 4-4).



Slika 4-4: Priključitev stikala ali tipke na vhod RA4

```
Stikalo equ   0xF0          ;spremenljivka Stikalo v GPR registru na naslovu 0xF0
; Inicializacija: vrata PORTA (RA4 vhod)
       bsf    STATUS,RP0    ;1->RP0, kodna stran Bank1 zaradi dostopa do TRISA registra
       bsf    TRISA,4       ;1->PORTA.4, RA4: vhod
       bcf    STATUS,RP0    ;kodna stran Bank0 zaradi dostopa do PORTA registra!
; Ugotavljanje stanja PORTA, bit 4, rezultat v registru W
ponovi movf   PORTA,w       ;PORTA->W
       andlw  b'00010000'   ;bitni AND med W in 0x10, maskiranje bita 4, W=000x0000, x=st.
; ce je W=0, je stikalo sklenjeno!, ce je W razlicen od 0, je stikalo odprto!
       movwf  Stikalo       ;W->Stikalo, shrani stanje v spre. Stikalo
       goto   ponovi        ;neskoncna zanka
```

Program 4-3: Ugotavljanje stanja stikala v zbirnem jeziku

V algoritmu za ugotavljanje stanja stikala je potrebno upoštevati, da se **stikalo** smatra kot **idealno**. Dejansko prihaja pri mehanskih stikalih do **pojava »odskakovanja«**, zato je potrebno ustrezno ukrepati (angl.: debounce) [6], bodisi v vezju, bodisi v programskem algoritmu. Jedro zanke **`ponovi`** traja (Program 4-3) le okrog 1 μs, pojav odskakovanja (spreminjanje logičnega stanja med '0' in '1') pa od nekaj ms do nekaj 10 ms.

Za ilustracijo si še poglejmo, kako je lahko kodiran **enak algoritem v C-jeziku** (Program 4-4) za prevajalnik HI-TEC PICC:

```
main()
{
unsigned char Stikalo; //spremenljivka Stikalo je 8-bitno celo stevilo brez predznaka
// Inicializacija PORTA, RA4 kot vhod
TRISA=TRISA|0x10;     // 1 -> bit4, RA4 je vhod
       while(1)       //neskoncna zanka
       {      //zacetek zanke
// Ugotavljanje stanja PORTA, bit 4, rezultat v spremenljivki Stikalo
       Stikalo=RA4;  //RA4->Stikalo
// ce je Stikalo=0 je stikalo sklenjeno!, ce je Stikalo razlicen od 0, je stikalo odprto!
       }              //konec zanke
}
```

Program 4-4: Ugotavljanje stanja stikala v C- jeziku

Obsežnejši primer (Program 4-5) konfiguracije vrat in uporabe vhodov/izhodov:

```
; ********* Program za inicializacijo vrat A, B, C in brisanje RB4 ter postavitev RA5 **

       include <p16f876.inc>; vkljucitev datoteke z definicijami simbolov registrov
       list   p=16f876       ; izbira tipa cipa

; Inicializacija: vrata A (RA2, RA5 izhoda), B (vsi izhodi), C (vsi izhodi)
       clrf   PORTB  ; zacetno stanje vrat B =0, ker stanja ob RESET-u niso definirana!
       clrf   PORTC  ; zacetno stanje vrat C =0, ker stanja ob RESET-u niso definirana!

       bsf    STATUS,RP0     ; kodna stran Bank1 zaradi dostopa do TRISn registrov
       movlw  0x00           ; PORTB in PORTC - vsi izhodi
       movwf  TRISB          ; PORTB - RB(0:7) : izhodi
       movwf  TRISC          ; PORTC - RC(0:7) : izhodi
       movlw  B'00011011'    ; 0x1B ->W,  RA(2, 5): izhodi, RA(0, 1, 3, 4): vhodi
       movwf  TRISA          ; W->TRISA
                             ; za PORTA je potrebno v ADCON1 vpisati B'10000100' ali 0x84
       movlw  B'10000100'    ; B'10000100' -> W, 0x84
       movwf  ADCON1         ; W->ADCON1, desno poravnan zapis, RA(0,1,3): analogni vhodi
; Postavitev zacetnih stanj na vratih B in C
       bcf    STATUS,RP0     ; kodna stran Bank0 zaradi dostopa do PORTn registrov!
       movlw  B'11111111'    ; nastavitev stanja 0xFF na izhodih PORTC
       movwf  PORTC          ; na PORTC
       movlw  B'00001111'    ; nastavitev stanja 0x0F na izhodih PORTB
       movwf  PORTB          ; na PORTB
; Ponavljajoca se zanka
label1 movlw  0xEF           ; B'11101111'
       movwf  PORTB          ; zapiši na PORTB, RB4=0, ostali 1
       bsf    PORTA,5        ; postavi bit 5 PORTA, RA5=1
       goto   label1         ; izvajaj zanko, skoci na label1

       end                   ;psevdoukaz za zakljucek programske kode
```

Program 4-5: Popoln primer programa v zbirnem jeziku za logične signale

Popoln zgled programa za izvajanje na MPU-PIC16F876:

Primer programa v zbirnem jeziku

```
;-------------------------------------------------------------------------------
; Demo program za vklop LED diod: LD1 na RC0 in LD3 na RC3
;-------------------------------------------------------------------------------

            list        p=16f876  ;izbira tipa cipa
            include <p16f876.inc> ;vkljucitev datoteke z definicijami
                                  ;simbolov
;-------------------------------------------------------------------------
; Nastavitev vektorjev

            org         0x00      ;RESET vektor
            goto        Main      ;skok na glavni program Main
            org         0x05      ;zacetek programske kode
Main
;--------- Inicializacija registrov vhodno/izhodnih vmesnikov
            bsf         STATUS,RP0          ;banka 1
            movlw       0x06      ;v delovni register nalozi vrednost 0x06
            movwf       ADCON1    ;ne uporabljamo analognih vhodov
                                  ; (00000110)
            movlw       0x10      ;0 - izhod, 1 – vhod, b'00010000'
            movwf       TRISA     ;vsi pini porta A so izhodni, razen RA4

            clrf        TRISB     ;vsi pini porta B so izhodni
            clrf        TRISC     ;vsi pini porta C so izhodni
            bcf         STATUS,RP0          ;banka 0
            clrf        INTCON    ;prekinitev ne bomo uporabljali
            clrf        PORTC     ;izhode porta C postavi na 0

;-------- jedro programa
Zanka
            bsf         PORTC,0   ;vklop LED diode LD1 na RC0
            bsf         PORTC,3   ;vklop LED diode LD3 na RC3
            goto        Zanka     ;skok na zacetek

            end                   ;psevdo ukaz za konec programa
```

Primer programa v C-jeziku

```
/* Demo program za vklop LED diod: LD1 na RC0 in LD3 na RC3 */

#include <pic1687x.h> // vkljucitev datoteke z definicijami simbolov

void main(void)
{

/* --------- Inicializacija registrov vhodno/izhodnih vmesnikov */
ADCON1=0x06;   // ne uporabljamo analognih vhodov, 0b00000110
TRISA=0x10;  // vsi pini porta A so izhodni, razen RA4,
                // 0b00010000)
TRISB=0;       // vsi pini porta B so izhodni
TRISC=0;       // vsi pini porta C so izhodni
PORTC=0;       // izhode porta C postavi na 0
INTCON=0;      // prekinitev ne bomo uporabljali

/* -------- jedro programa */

            while (1)   // neskoncna zanka
            {
            RC0=1;     // vklop LED diode LD1 na RC0
            RC3=1;     // vklop LED diode LD3 na RC3
            }

}
```

Program 4-6: Popoln primer programa v zbirnem in C-jeziku za logične izhode

Za razumevanje tematike je potrebno tudi predznanje iz osnov programiranja. Dodatna znanja iz programiranja v zbirnem jeziku so v: [2][4][6][7]. Za razumevanje programov, ki so kodirani v C-jeziku, je potrebno poznavanje osnov ANSI C strukturnega programiranja: [15][11][13][5]. Veliko rešitev praktičnih nalog z uporabo PIC mikrokrmilnikov je na spletnem portalu http://www.interq.or.jp/japan/se-inoue/e_pic6.htm.

Vprašanja za utrjevanje:

Napišite podprogram za inicializacijo vseh linij vrat PORTB kot izhodi.

Rešitev:

```
PBizh   bsf     STATUS,RP0      ; 1 -> RP0 , izbira Bank1 za dostop do TRISB
        clrf    TRISB           ; 0 -> TRISB, vsi izhodi
        bcf     STATUS,RP0      ; 0 -> RP0 , izbira Bank0 za dostop do PORTB
        return                  ; konec podprograma
```

1. *Napišite podprogram za inicializacijo vseh linij vrat PORTC kot vhodi.*
2. *Kakšen pomen imajo linije vrat PORTA ?*
3. *Napišite programček za ugotavljanje stanj na vhodih: RA0, RA1 in RA3 ?*
4. *Stanje stikala na vhodu RA4 prenesi (programsko) na izhod RC3.*
5. *Izhode na RC0 in RC1 preklapljaj izmenoma vsakih (pribl.) 100 ms.*

5. Analogni vhodi

Spoznali boste vmesnik mikrokrmilnika, ki omogoča uporabo analognih vhodov mikrokrmilnika. Naučili se boste konfigurirati posamezne priključke vrat PORTA za izbiro analognih vhodov. Naučili se boste uporabljati 10 ali 8-bitni rezultat in ločevati med desno oz. levo poravnavo ter interpretirati številčni rezultat AD pretvorbe.

A/D pretvornik je 10-bitni, vsebuje 5-kanalni multiplekser, modul pa ima tudi pozitivni in negativni referenčni vhod [4].

Registri A/D modula so:

- register višjih bitov rezultata A/D pretvorbe (A/D Result High Register, **ADRESH**)
- register nižjih bitov rezultata A/D pretvorbe (A/D Result Low Register, **ADRESL**)
- A/D kontrolni register 0 (A/D Control Register 0, **ADCON0**)
- A/D kontrolni register 1 (A/D Control Register 1, **ADCON1**)

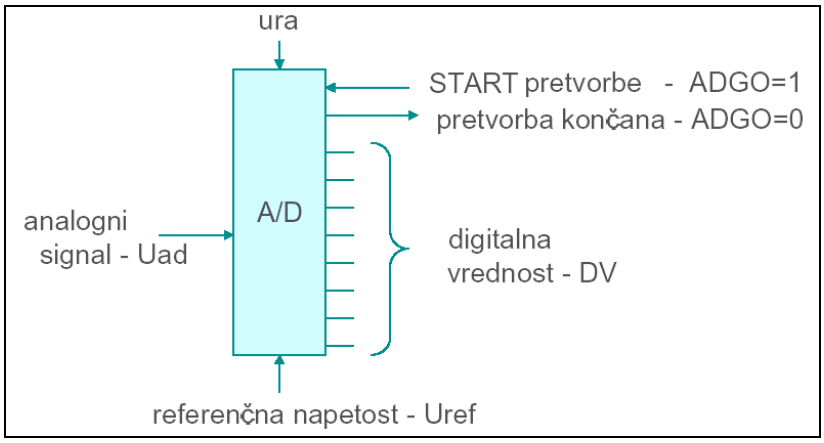
ADCON0 nadzoruje delovanje A/D modula, **ADCON1** pa nastavi funkcije priključkov. Registra **ADRESH:ADRESL** vsebujeta 10-bitni rezultat pretvorbe. Ko je A/D pretvorba zaključena, se rezultat naloži v oba registra, bit GO/$\overline{\text{DONE}}$ (**ADCON0**<2>) se postavi na '0', bit ADIF se postavi na '1'.

5.1. Delovanje in uporaba AD pretvornika

Postopek uporabe pretvornika je opisan v naslednjih točkah:

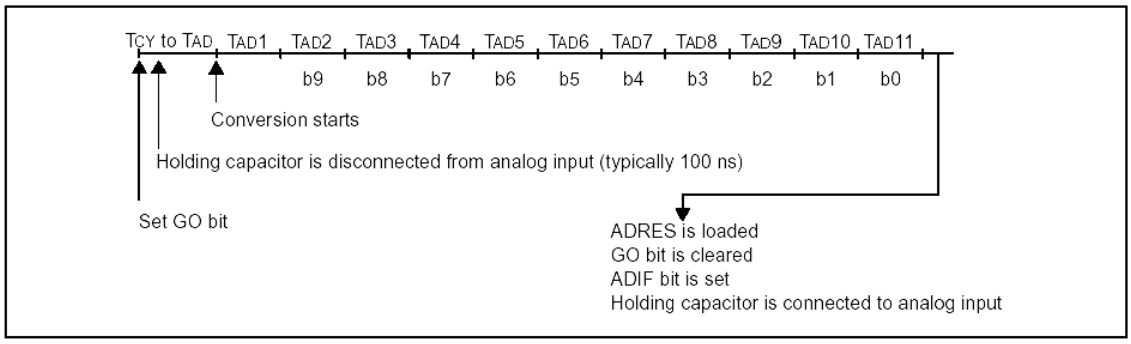
1. **Konfiguriraj A/D modul:**
   - Konfiguriraj analogne priključke/napetostne reference in digitalne vhode/izhode (**ADCON1**)
   - Izberi vhodni kanal A/D pretvornika (**ADCON0**)
   - Izberi takt A/D pretvorbe (**ADCON0**)
   - Vključi A/D modul (**ADCON0**)
2. Po potrebi konfiguriraj prekinitve, ki jih proži A/D pretvornik
   - Zbriši bit **ADIF**
   - Postavi bit **ADIE**
   - Postavi bit **GIE**
3. Počakaj, da se inicializacija izvede
4. **Poženi pretvorbo**
   - Postavi bit **GO/$\overline{\text{DONE}}$ oz. ADGO** (**ADCON0**)
5. **Počakaj, da se A/D pretvorba konča, obstajata dva načina:**
   - **Čakaj, da se zbriše bit GO/$\overline{\text{DONE}}$ oz. ADGO ali**
   - Čakaj na A/D prekinitev
6. **Preberi rezultat v registrih (ADRESH:ADRESL), po potrebi zbriši bit ADIF.**
7. Za naslednjo pretvorbo se vrni v korak 1 ali 2 (kateri ustreza).

Običajno uporabljamo A/D pretvornik tako, da ga »programsko startamo« (postavitev bita 2 v registru **ADCON1**) in zatem počakamo na zaključek pretvorbe, tako da programsko testiramo isti statusni bit **ADGO** (Slika 5-1).



Slika 5-1: Simbolični prikaz delovanja A/D pretvornika

Kratek čas (največ 2 µs) po postavitvi bita **ADGO** (GO/DONE), se **zajame** (**trenutno otipa) vrednost na analognem vhodu** (širina okna je približno 100 ns) in se shrani v »zadrževalni kondenzator«. Zatem se prične proces pretvorbe (Slika 5-2) po postopku »zaporednega približevanja«.



Slika 5-2: Časovni potek delovanja A/D pretvorbe

Analogno digitalna pretvorba traja okrog 20 µs ($12T_{AD}$, pri čemer je interval T_{AD} enak 1,6 µs). Ko se zaključi A/D pretvorba, **mora miniti vsaj $2T_{AD}$ (3,2 µs)**, preden lahko sprožimo naslednje zajemanje analognega vhoda.

Rezultat (digitalna vrednost – DV) je seveda **odvisen od referenčne napetosti** (izbiramo lahko med notranjo – napajanje 5,0 V ali zunanjo - na vhodih V_{ref-} in V_{ref+}), kakor tidi od kvalitete tiskanega vezja, dolžine in oblike povezav ter od **preciznosti** samega pretvornika.

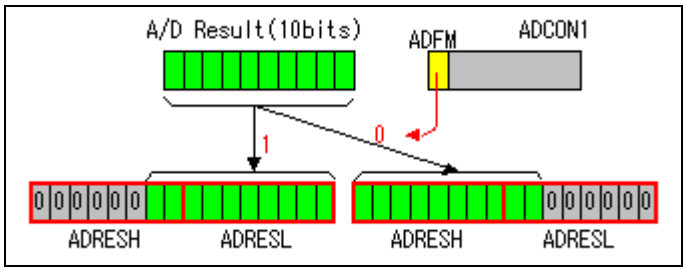
Napetost na vhodu izračunamo na osnovi 8 ali 10-bitne vrednosti na naslednji način:

U_{ad} = DV*q, pri čemer je: **$q=U_{ref}/256$** (8-bitni rezultat), **$q=U_{ref}/1024$** (10-bitni rezultat);

DV je pri **8-bitni** vrednosti rezultata med **0 in 255**, pri **10-bitni** pa med **0 in 1023**.

Vrednost kvanta: **q=4,883 mV** (10-bitni rezultat) oz. **q=19,53 mV** (8-bitni rezultat) ob predpostavki, da je referenčna napetost natanko **U_{ref}=5,00 V**.

Način shranjevanja rezultata AD pretvorbe (Slika 5-3):

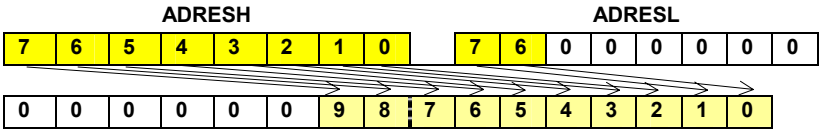


Slika 5-3: Desna ali leva poravnava rezultata A/D pretvorbe

V kolikor se zadovoljimo z **8-bitno vrednostjo rezultata (0 do 255)**, izberemo levo poravnavo in rezultat prenesemo iz registra **ADRESH** v 8-bitno spremenljivko. Pri tem je ločljivost A/D pretvorbe enaka 19,53 mV (pri območju - referenci 5,0 V).

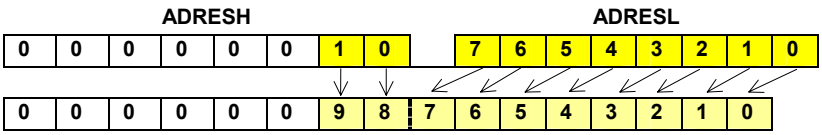
Če je pomemben popoln **10-bitni rezultat (0 do 1023)**, je primernejša izbira **desne poravnave**, postopek shranjevanja pa je prikazan spodaj. Pri tem je ločljivost A/D pretvorbe enaka 4,88 mV (pri območju - referenci 5,0 V).

Postopek za prenos levo poravnanega rezultata v 16-bitno spremenljivko (Slika 5-4):



Slika 5-4: Shranjevanje levo poravnanega rezultata v 16-bitno spremenljivko

Postopek za prenos desno poravnanega rezultata v 16-bitno spremenljivko (Slika 5-5):



Slika 5-5: Shranjevanje desno poravnanega rezultata v 8 ali 16-bitno sprem.

5.2. Registri AD pretvornika in primer programa

5.2.1. Register ADCON0

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	**ADCS0**	**CHS2**	**CHS1**	**CHS0**	**GO/$\overline{\text{DONE}}$**	**—**	**ADON**
bit7	6	5	4	3	2	1	bit0

Legenda: R = omogočeno branje bita, W = omogočeno pisanje v bit, U = neuporabljen bit, beri kot '0', -n = vrednost ob POR resetu

bit 7-6: **ADCS1:ADCS0: bita za izbiro takta A/D pretvorbe**

00 = FOSC/2
01 = FOSC/8
10 = FOSC/32 (priporočljiva izbira pri 20 Mhz taktu)
11 = FRC (zunanji RC oscilator)

bit 5-3: **CHS2:CHS0: biti za izbiro analognega kanala**

000 = kanal 0, (RA0/AN0)
001 = kanal 1, (RA1/AN1)
010 = kanal 2, (RA2/AN2)
011 = kanal 3, (RA3/AN3)
100 = kanal 4, (RA5/AN4)

bit 2: **GO/$\overline{\text{DONE}}$: Statusni bit A/D pretvorbe (**bit se lahko imenuje tudi **ADGO)**

Če je ADON = 1
1 = A/D pretvorba poteka (postavitev tega bita sproži A/D pretvorbo)
0 = A/D pretvorba ne poteka (ta bit se avtomatsko postavi na '0' ko je A/D pretvorba končana)

bit 1: **Neuporabljen**: beri kot '0'

bit 0: **ADON**: A/D On bit

1 = A/D pretvornik deluje
0 = A/D pretvornik je izključen

5.2.2. Register ADCON1

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	**-**	**-**	**-**	**PCFG3**	**PCFG2**	**PCFG1**	**PCFG0**
bit7							bit0

Legenda: R = omogočeno branje bita, W = omogočeno pisanje v bit, U = neuporabljen bit, beri kot '0', -n = vrednost ob POR resetu

bit 7: **ADFM: Izbira formata rezultata A/D pretvorbe**

1 = Desno poravnan. 6 najbolj pomembnih (most significant) bitov **ADRESH** se prebere kot '0'.
0 = Levo poravnan. 6 najmanj pomembnih (least significant) bitov **ADRESL** se prebere kot '0'.

bit 6-4: **Neuporabljeni: Beri kot '0'**

bit 3-0: **PCFG3:PCFG0:** Biti za konfiguracijo A/D vmesnika (Configuration Control bits)

PCFG3: PCFG0	**AN7 (1) RE2**	**AN6(1) RE1**	**AN5 (1) RE0**	**AN4 RA5**	**AN3 RA3**	**AN2 RA2**	**AN1 RA1**	**AN0 RA0**	**VREF+**	**VREF-**	**CHAN / Refs (2)**
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	VSS	4/1
0100(3)	**D**	**D**	**D**	**D**	**A**	**D**	**A**	**A**	**VDD**	**VSS**	**3/0**
0101	D	D	D	D	VREF+	D	A	A	RA3	VSS	2/1
011x(3)	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**VDD**	**VSS**	**0/0**
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

Legenda: A = analogni vhod, D = digitalni vhod/izhod

Opomba 1: Ti kanali na čipih z 28 nožicami (16F876) niso na voljo, ampak pri PIC mikrokrmilnikih s 40 nožicami (16F877)

2: Stolpec označuje število analognih vhodov in število analognih vhodov, ki jih uporabljamo kot vhode za referenčno napetost

3: V krepkem tisku sta vrstici, ki pomenita najbolj verjetno izbiro.

Popoln zgled programa za A/D pretvornik za izvajanje na MPU-PIC16F876:

Primer programa v zbirnem jeziku

```
;-------------------------------------------------------------------------------
; Naziv programa: Demo program za AD pretvorbo: AN3 shrani v RezH
; in   RezL ,   v casu AD pretvorbe vklopljena LD4 na RA5 izhodu
;-------------------------------------------------------------------------------

            list        p=16f876  ;izbira tipa cipa
            include <p16f876.inc> ;vkljucitev datoteke z definicijami
                                  ; simbolov
;-------------------------------------------------------------------------------
;--- spremenljivke
RezH        equ         0x20      ;zgornjih 8 bitov rezultata
RezL        equ         0x21      ;spodnjih 8 bitov rezultata
;--- Nastavitev vektorjev
            org         0x00      ;RESET vektor
            goto        Main      ;skok na glavni program Main
            org         0x05      ;zacetek programske kode
Main
;--------- Inicializacija registrov vhodno/izhodnih vmesnikov
            bsf         STATUS,RP0          ;banka 1
            movlw       b'00011011'         ;0 - izhod, 1 - vhod
            movwf       TRISA               ;RA5-izhod,    RA4-vhod,
            ; RA3/AN3-vhod, RA2-izhod, RA1/AN1-vhod, RA0/AN0-vh.
            clrf        TRISB     ;vsi pini porta B so izhodni
            clrf        TRISC     ;vsi pini porta c so izhodni
;--- inicializacija A/D - AN3
            movlw       b'10000100'  ;desna poravnava, vhod AN3 je
                                     ; analogni, tudi AN1 in AN0
            movwf       ADCON1    ;bit7=1 -> desna poravnava 10-
                                  ; bitnega rezultata
            bcf         STATUS,RP0          ;nazaj v banko 0
            movlw       b'10011001'   ;takt (f/32,takt=20MHz), izbira
                                  ; anal. kan. (AN3), AD pretv. je omogocen
            movwf       ADCON0
;----------------------
            clrf        INTCON    ;prekinitev ne bomo uporabljali
            clrf        PORTC     ;izhode porta C postavi na 0

;-------- jedro programa
Zanka
            bsf         PORTA,5   ;vklopi LD4 na RA5
;--- Start AD pretvorbe in cakanje na rezultat
            bsf         ADCON0,2 ;start AD pretvorbe (postavi bit
                                 2 (GO_DONE) na 1
pocakaj     btfsc       ADCON0,2;testiraj GO_DONE
            goto        pocakaj   ;cakanje na izvedbo AD
                                  pretvorbe (pribl. 20 us)
;--- pretvorba je koncana in rezultat pripravljen
            bcf         PORTA,5   ;izklopi LD4 na RA5
            movf        ADRESH,w     ;zgornjih 8-bitov (dejansko le 2
                                          bita!) rezultata -> W
            movwf       RezH      ;premik visjih osmih bitov iz
                                  delovnega registra w v sprem RezH
            bsf         STATUS,RP0  ;v banko1, kjer je ADRESL
            movf        ADRESL,w    ;spodnjih 8-bitov rezultata -> w
            bcf         STATUS,RP0  ;nazaj v banko0
            movwf       RezL        ;premik nizjih 8-bitov v   RezL
Test        nop                     ;«Test« simbol za Breakpoint
            goto        Zanka       ;ponovitev AD pretvorbe

            end                     ;psevdo ukaz za konec programa
```

Primer programa v C-jeziku

```
/*      ------------------------------------------------------------------------------
Demo program za AD pretvorbo vhoda AN3, rezultat shrani v 16-bit
sprem. Rez ; v casu AD pretvorbe je vklopljena LD4 na RA5 izhodu
------------------------------------------------------------------------------*/


#include <pic1687x.h>  /* vkljucitev datoteke z definicijami simbolov */

unsigned int Rez;   /* sprem. za rezultat pretvorbe */

void main(void)
{

/* --------- Inicializacija registrov vhodno/izhodnih vmesnikov */

TRISA=0x1b;  /*b'00011011' RA5-izhod, RA4-vhod, RA3/AN3-vhod,
          RA2-izhod, RA1/AN1-vhod, RA0/AN0-vhod */
TRISB=0;              /* vsi pini porta B so izhodni */
TRISC=0;              /* vsi pini porta C so izhodni */
/*--- inicializacija A/D - AN3 */
ADCON1=0x84;       /* b'10000100' desna poravnava, vhod AN3 je
                                  analogni, tudi AN1 in AN0 */
ADCON0=0x99; /* b'10011001' takt(f/32,takt=20 MHz), izbira anal. kan.
                          (AN3) */

PORTC=0;        /* izhode porta C postavi na 0 */
INTCON=0;       /* prekinitev ne bomo uporabljali */

/* -------- jedro programa */

          while (1)   /* neskoncna zanka */
          {
          RA5=1;     /* vklop LD4 */
          /*--- Start AD pretvorbe in cakanje na rezultat */
          ADGO=1;  /* start AD pretvorbe, postavi bit 2 (GO_DONE)
na 1 */
          while (ADGO==1); /* cakanje na izvedbo AD pretvorbe (~
20 us) */

          RA5=0;     /* izklop LD4 */
          Rez = ADRESL|(ADRESH<<8);    /* rezultat AD pretvorbe */

          asm("test nop");    /* moznost nastav. Breakpoint na simb.
"test" */

          }          /* ponovitev AD pretvorbe */

}
```

Program 5-1: Popoln primer programa v zbirnem in C-jeziku za A/D pretvornik

Za razumevanje tematike je potrebno tudi predznanje iz osnov programiranja in številskih sistemov. Dodatna znanja iz programiranja v zbirnem jeziku so v: [2][4][6][7]. Za razumevanje programov, ki so kodirani v C-jeziku, je potrebno poznavanje osnov ANSI C strukturnega programiranja: [15][11][13][5]. Veliko rešitev praktičnih nalog z uporabo PIC mikrokrmilnikov, je na spletnem portalu http://www.interq.or.jp/japan/se-inoue/e_pic6.htm. Podrobnejši opis delovanja A/D pretvornika s primeri uporabe v PIC mikrokrmilnikih najdemo tudi v: [8][16][17][18].

Vprašanja za utrjevanje:

Napišite podprogram z imenom InAD1 za inicializacijo AD pretvornika (vhod AN1) z desno poravnavo rezultata.

Rešitev:

```
;--------- Inicializacija registrov vhodno/izhodnih vmesnikov
InAD1  bsf    STATUS,RP0     ; 1 -> RP0, Bank1 za dostop do TRISA in ADCON1
       bsf    TRISA,1        ; 1 -> TRISA.1, RA1/AN1- vhod
;--- inicializacija A/D - AN1
       movlw  b'10000100'    ;priprava vredn. za ADCON1, vhod AN1-analogni, tudi AN3 in AN0
       movwf  ADCON1         ;bit7=1 -> desna poravnava 10-bitnega rezultata
       bcf    STATUS,RP0     ;0 ->RP0, nazaj v banko 0
       movlw  b'10001001'    ;takt (f/32,Fosc=20MHz), izbira anal. kan. AN1, AD  omogocen
       movwf  ADCON0         ;konec inicializacije
       return                ;konec podprograma
```

1. *Napišite podprogram z imenom InAD3 za inicializacijo AD pretvornika (vhod AN3) z levo poravnavo rezultata.*
2. *Napišite podprogram (ki se navezuje na točko 1) za Start AD pretvorbe vhoda AN3 in shranjevanje 8-bitnega rezultata v register W.*
3. *Kolikšna napetost je na analognem vhodu, če smo pri 8-bitni pretvorbi dobili rezultat 7Fh ?*
4. *Kolikšna napetost je na analognem vhodu, če smo pri 10-bitni pretvorbi dobili rezultat 300h in je V_{ref+} = 4,00 V ?*
5. *Kolikšen je čas AD pretvorbe (približno) in po kolikem času lahko sprožimo ponovno AD pretvorbo ?*

6. Časovnik Timer0 in periodično generiranje prekinitev

Spoznali boste osnovni 8-bitni časovnik. Naučili se ga boste konfigurirati in generirati natančne časovne intervale s pomočjo prekinitev.

Modul **Timer0** je 8-bitni časovnik/števec [2][6][8] z naslednjimi lastnostmi:

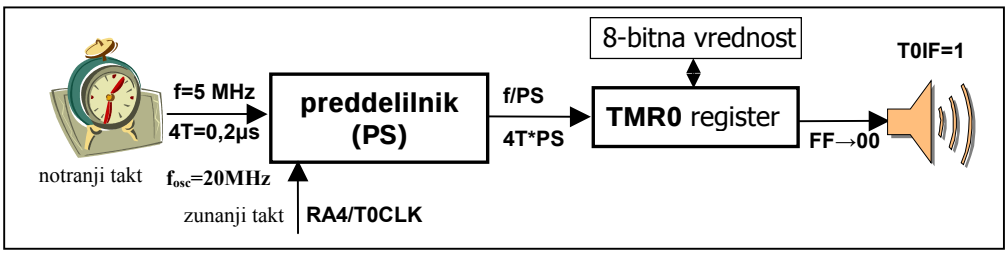
- vsebuje 8-bitni programsko izbirljiv preddelilnik (prescaler PS);
- ima možnost izbire med notranjim (urin takt, npr.: 20 MHz) in zunanjim taktom (na voljo je tudi možnost izbire aktivnega prehoda);
- sproži zastavico (**T0IF**) in (če so prekinitve omogočene) zahtevo za prekinitev (**Interrupt**) ob napolnitvi števca (prehod iz FFh na 00);
- omogoča štetje dogodkov (prehodov signala) na zunanjem priključku **RA4/T0CLK**
- je standardna enota večine Microchip PIC mikrokrmilnikov (npr.: 16F84) srednje kategorije (Mid Range).

Z modulom **Timer0** so povezani registri:

- **TMR0** (01h) – 8-bitni števni register (vpišemo lahko 8-bitno število: 256 vrednosti, register vedno **prišteva** prehode takta do napolnitve, zatem se register inkrementira ali povečuje od začetne vrednosti 00 oz. vpisane vrednosti);
- **OPTION_REG** (81h) – register z biti za nastavitev preddelilnika in kontrolnimi biti timerja;
- **INTCON** (0Bh) – register s kontrolnimi in statusnimi biti prekinitvenih zahtev.

6.1. Splošno o časovniku Timer0

Izbiramo lahko (Slika 6-1) med **notranjim taktom** (četrtina frekvence urinega takta) in **zunanjim taktom**, ki ga pripeljemo na vhod **RA4/T0CLK**.

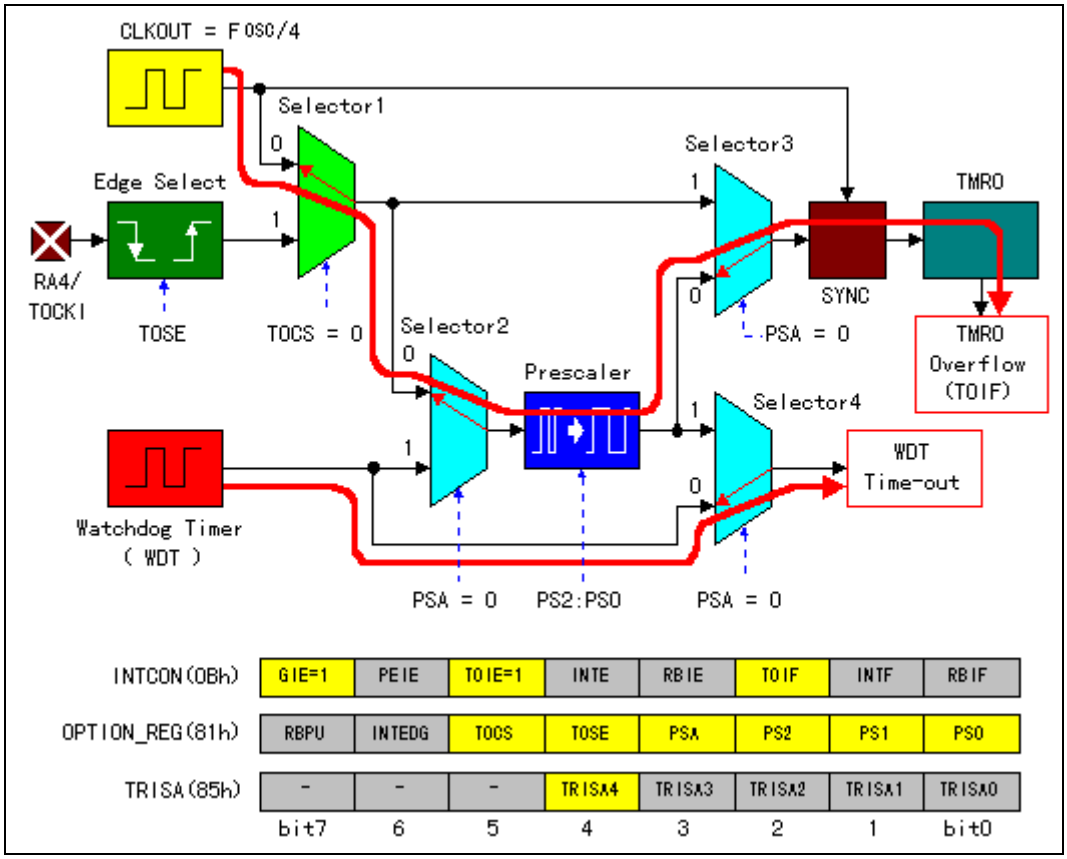


Slika 6-1: Simbolična zgradba časovnika Timer0

Največji možni doseg (interval) časovnika **Timer0** je: (256*256)*0,2 µs=**13,1 ms** (pri izbiri preddelilnika 1:256, začetni vrednosti **TMR0**=0 in taktu 20 MHz). Večjo (ugodnejšo) ločljivost nastavitve časovnega intervala dosežemo pri čim manjši vrednosti preddelilnika.

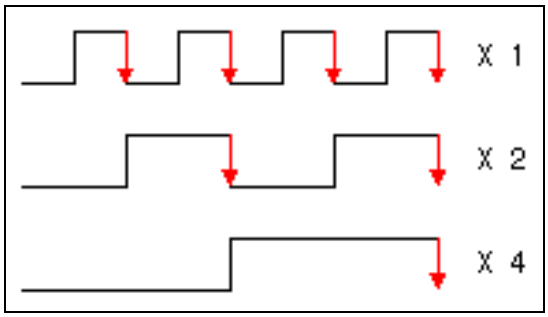
Ob vsaki napolnitvi (prelivu) števca (Slika 6-2) se samodejno aktivira zastavica (**T0IF**=1) in hkrati tudi sproži prekinitvena zahteva (če je omogočena). V prekinitvenem strežnem programu je potrebno programsko zbrisati zastavico (**T0IF**=0), preden ponovno omogočimo prekinitve.

$Timer0_{Interval}$ = (256-**TMR0**) * PS * 4 * T_{osc}, pri čemer je T_{osc} = 50 ns, če je frekvenca urinega takta 20 MHz.

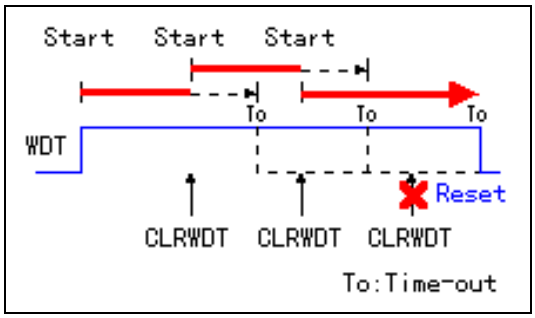


Slika 6-2: Podrobnejši grafični prikaz delovanja časovnika Timer0

Preddelilnik (Prescaler) ima pomembno funkcijo (Slika 6-3), kajti osnovni takt (frekvenco) deli z izbranim faktorjem, kar posredno pomeni povečanje števila bitov števca (dejansko se perioda signala v števec pomnoži z vrednostjo preddelilnika).



Slika 6-3: Delovanje preddelilnika (Prescaler)



Slika 6-4: »Časovni stražnik« (Watch Dog Timer)

Časovni stražnik (Watch Dog Timer) si ekskluzivno deli (en sam) preddelilnik s časovnikom Timer0 (Slika 6-4).

Časovni stražnike sicer uporabljamo v **vgrajenih sistemih**, kjer skrbijo, da se aplikativni programi ne bi zazankali [1][6]. V PIC mikrokrmilnikih ima **WDT** (Slika 6-2) svoj takt (RC oscilator, ki je neodvisen od sistemskega kvarčnega oscilatorja). **Največji interval** do izteka **WDT** je približno **18 ms** (pri preddelilniku 1) in 2,3 s (pri preddelilniku 1:128). Uporabniški program mora pred iztekom **WDT** izvesti namenski ukaz **CLRWDT**, ki zbriše števec. **Če se to ne zgodi, povzroči časovni stražnik RESET mikrokrmilnika.**

6.1.1. Register OPTION_REG

Nahaja se na naslovu 81h oz 01h v segmentu Bank1.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	**INTEDG**	**T0CS**	**T0SE**	**PSA**	**PS2**	**PS1**	**PS0**
bit7	6	5	4	3	2	1	bit0

Legenda: R = omogočeno branje bita, W = omogočeno pisanje v bit, U = neuporabljen bit, beri kot '0', -n = vrednost ob POR resetu

bit 7: **$\overline{\text{RBPU}}$**: **PORTB** "Pull-up" upori - izbira (1 – upori niso vključeni, 0 – upori so vključeni)

bit 6: **INTEDG**: izbira prehoda signala za prekinitev preko RB0/INT vhoda (1 – naraščajoči, 0 – padajoči)

bit 5: **T0CS**: izbira takta za **TMR0** (1 – zunanji takt na RA4/T0CKL, **0** – notranji takt iz CLKOUT)

bit 4: **T0SE**: izbira prehoda signala za **TMR0** (1 – padajoči, **0** – naraščajoči na RA4/T0CKL zunanjem taktu)

bit 3: **PSA**: dodelitev preddelilnika (1 – **WDT** "Watch Dog Timer" časovni stražnik1, **0** – **Timer0**)

biti 2-0: **PS2:PS0**: biti za izbiro preddelilnika

PS2,PS1,PS0	TMR0 (PS)	WDT vrednost
0 0 0	1:2	1:1
0 0 1	1:4	1:2
0 1 0	1:8	1:4
0 1 1	1:16	1:8
1 0 0	1:32	1:16
1 0 1	1:64	1:32
1 1 0	1:128	1:64
1 1 1	1:256	1:128

Table 6-1: Vrednosti preddelilnika za TMRO in WDT

6.1.2. Register INTCON

Nahaja se na naslovu 0Bh v vseh segmentih.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	**PEIE**	**T0IE**	**INTE**	**RBIE**	**T0IF**	**INTF**	**RBIF**
bit7	6	5	4	3	2	1	bit0

Legenda: R = omogočeno branje bita, W = omogočeno pisanje v bit, U = neuporabljen bit, beri kot '0', -n = vrednost ob POR resetu

bit 7: **GIE**: Globalni prekinitveni bit (**1** – omogoči vse nemaskirane prekinitve, **0** – onemogoči vse prekinitve)

bit 6: **PEIE**: bit za omogočanje prekinitev periferije (1 – omogoči vse nemaskirane prekinitve periferije, **0** – onemogoči)

bit 5: **T0IE**: omogočanje prekinitev **TMR0** (**1** – omogoči TMR0 prekinitev, 0 – onemogoči ali maskiraj)

bit 4: **INTE**: omogoči zunanjo prekinitev RB0/INT (1 – omogoči, **0** – onemogoči ali maskiraj)

bit 3: **RBIE**: omogoči prekinitve ob spremembi na RB vratih (1 – omogoči, **0** – maskiraj)

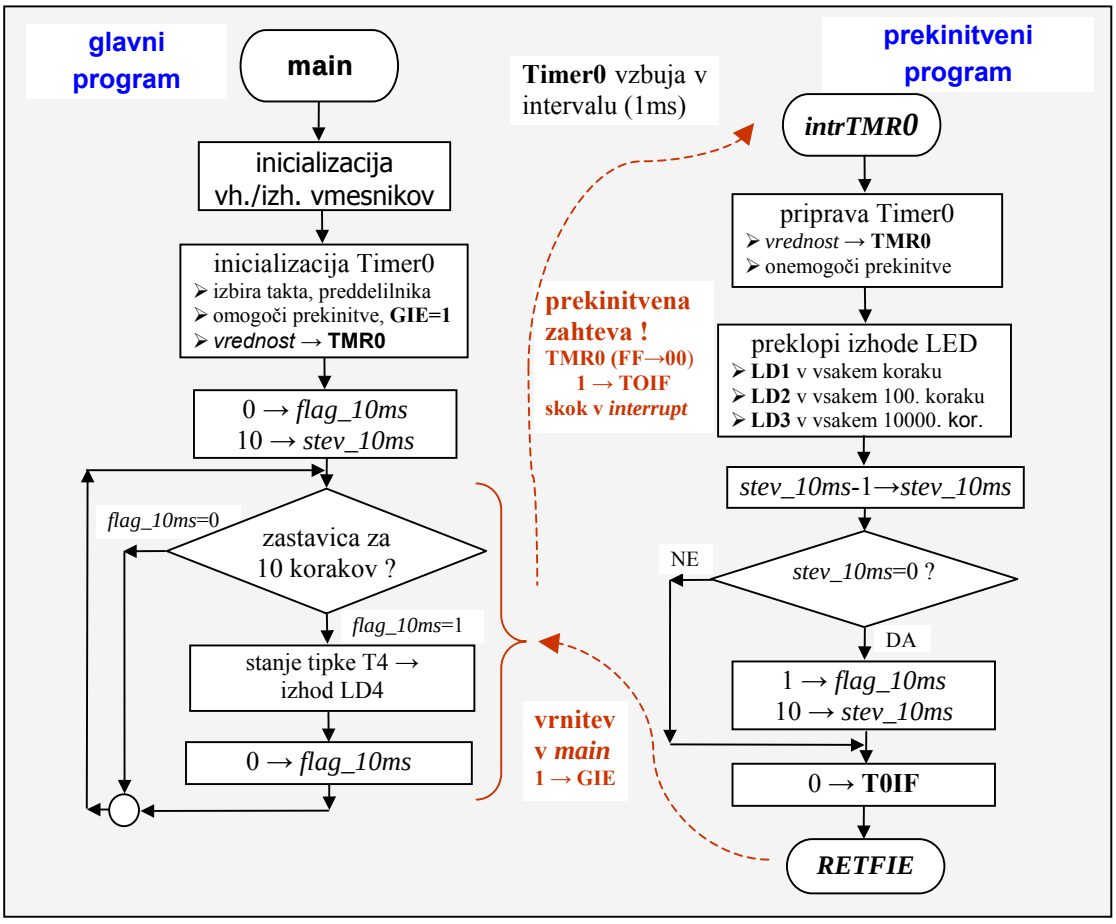
bit 2: **T0IF**: zastavica prekoračitve števca **TMR0** (**1** – prekoračitev ali prehod iz FFh na 0 dosežena, 0 – ni prekoračitve)

bit 1: **INTF**: zastavica prekinitve zaradi RB0/INT vhoda (1 – se je zgodila, 0 – ni)

bit 0: **RBIF**: zastavica prekinitve na RB4 - RB7 vhodih (1 – se je zgodila, 0 – ni)

6.2. Timer0 in prekinitve

Poglejmo si diagram poteka **tipičnega glavnega (*main*)** in prekinitvenega **strežnega programa (*interrupt*)** za periodično generiranje različnih časovnih intervalov (Slika 6-5):



Slika 6-5: Diagram poteka glavnega in prekinitvenega strežnega programa

Ob začetku prekinitvenega strežnega programa (***interrupt***) je priporočljivo (v kolikor ne uporabljamo drugih prekinitvenih virov) **zbrisati bit za omogočanje prekinitev** (**GIE=0**), zato da so med izvajanjem prekinitvenega programa vse prekinitve blokirane in s tem ni nevarnosti ponovnega aktiviranja prekinitvene zahteve, še preden se strežni program konča. Prekinitveni strežni program mora biti **vedno zaključen z ukazom RETFIE**, ki povzroči vrnitev v prej prekinjeni program in ponovno omogočanje prekinitev (ob tem se samodejno postavi bit **GIE=1**).

Programske koda algoritma v zbirnem jeziku (Slika 6-5) je nekoliko daljša, zato je v nadaljevanju prikazan pripadajoč program le v izvorni kodi C-jezika (Hi-TECH PICC Lite).

Izvorni program v C-jeziku, ki ustreza kodiranju algoritma (Slika 6-5):

```
// Demo program - utripanje LED diod: LD1 na 1ms, LD2 na 100ms, LD3 na 10s
// stanje tipke T4 (RC4 -> RA4) se prenese na LD4 (Ra5) vsakih ~10ms
//    avtor: Janez Pogorelc, 3.5.04

#include <pic1687x.h>

// definicija konstant (macro)
#define Timer0  100  //zacetna vrednost za TMR0
#define STO_ms  100  //konstanta 100 za stevec - 100ms
#define DESET_s 100  //konstanta 100 za stevec - 10s

// deklaracija globalnih spremenljivk
unsigned char flag_10ms, stev_10ms, sto_ms, deset_s;

// --------- prekinitveni strezni program se izvaja v intervalu 10ms
void interrupt intTMR0(void)
{
TMR0=Timer0;  // vrednost stevca: 100=256-156, int: 156*32*0,2us=998,4us ~1ms
GIE=0;        // blokiraj vse prekinitve med izvajanjem strezbe

// LD1 preklopi v vsakem prekin. intervalu (1ms)
RC0=!RC0;     // izmenicni vklop LED diode LD1 na RC0

// LD2 preklopi na vsakih 100ms
  sto_ms--;   // dekrement stevca ms
  if (sto_ms==0)
  {
  RC1=!RC1;   // izmenicni vklop LED diode LD2 na RC1
  sto_ms=STO_ms;     //zacetna vrednost stevca za 100ms
  // ---- LD3 preklopi na vsakih 10s
       deset_s--;    // dekrement stevca s
       if (deset_s==0)
       {
       RC3=!RC3;     // izmenicni vklop LED diode LD3 na RC3
       deset_s=DESET_s; //zacetna vrednost stevca za 10s
       }
  }

  // postavitev zastavice za 10ms v vsakem 10. koraku
  stev_10ms--;// dekrement stevca
  if(stev_10ms==0) //ali je 10 korakov ?
  {
  flag_10ms=1;// postavitev zastavice
  stev_10ms=10;      // zacetna vrednost stevca ms
  }

T0IF=0;       // brisi zastavico Timer0 prekinitev
} //------- konec prekinitv. streznega programa


// ---------------- glavni program
void main(void)
{

// deklaracija lokalnih spremenljivk
char tmpPORTC;

// --------- Inicializacija registrov vhodno/izhodnih vmesnikov
ADCON1=0x06;  // 0b00000110, ne uporabljamo analognih vhodov
TRISA=0x10;   // 0b00010000, vsi pini porta A so izhodni, razen RA4
```

```
TRISB=0;      // vsi pini porta B so izhodni
TRISC=0;      // vsi pini porta c so izhodni
PORTC=0;      // izhode porta C postavi na 0
PORTA=0;      // izhode porta A postavi na 0

// inicializacija TMR0
OPTION=0x04;  // 0b00000100, notranji takt, PSA=32,
INTCON=0xA0;  // 0b10100000, GIE=1, T0IE=1,
TMR0=Timer0;  // vrednost stevca: 100=256-156, int: 156*32*0,2us=998,4us ~1ms

sto_ms=STO_ms;// zacetna vrednost stevca za 100ms
deset_s=DESET_s; // zacetna vrednost stevca za 10s
flag_10ms=0;  // brisanje zastavice
stev_10ms=10; // zacetna vrednost stevca ms

// -------- jedro programa

  while(1)    // neskoncna zanka
  {

       if (flag_10ms==1)     // ali je zastavica za 10. korak
       {
  //------ vklop LD4 ob pritisku na T4, izvede se vsakih 10ms
       tmpPORTC=PORTC; // shrani trenutno vrednost PORTC
       // testiraj T4
       PORTC=(PORTC&0x0f)|0X10; // postavi 1 na RC4, ohrani RC0 do RC3
       RA5=RA4;              // stanje vhoda RA4 je enako stanju T4
       PORTC=(PORTC&0x0F)|(tmpPORTC&0xF0); //ohrani PORTC in stanja LD

       flag_10ms=0;  // brisanje zastavice
       }
  }
}
// dolzina programa (brez optimizacij!): 126 Words, pomnilnik: 8 Bytes
// interval prekinitev: 5.002c (1000,4us)
// trajanje prekinitv. str. programa (min.): 23c (4,6us), (maks.): 36c (7,2us)
// trajanje zanke v main(): 132c (26,4us)
// izvajanje programa po Resetu do main(): 53c (10,6us)
// stevilo vrstic (brez komentarjev): 44 vrstic
```

Program 6-1: Popoln zgled programa v C-jeziku za prekinitve Timer0

Prekinitveni strežni program naj bo v splošnem čim krajši in učinkovitejši (brez zank). V kolikor so omogočeni (razen Timer0) tudi drugi prekinitveni viri (npr.: vhod **RB0/INT**, vhodi na **PORTB** vratih **RB4** do **RB7**, Timer1, Timer2, A/D pretvornik, programiranje EEPROM lokacij, ...), je potrebno v začetku prekinitvenega strežnega programa (programsko) preveriti (angl.: polling), katera enota je sprožila prekinitveno zahtevo. Namreč, vse enote (možnih je 13 različnih virov) sporočajo status o dogodku – »prekinitvena zahteva« v registrih: **OPTION_REG**, **PIR1**, **PIE2** in **PIR2**.

Izjemno **pomembno je, da programsko poskrbimo za »kontekst«** - vsebina registrov **W** in **STATUS** v glavnem programu. **Poskrbeti moramo, da registra ohranita vrednost tudi ob vrnitvi v glavni program**.

Strukturo »glavnega« in »prekinitvenega« programa (izpis ni primeren za izvajanje, ker ni popoln!) prikazuje zgled izvornega programa v zbirnem jeziku (Program 6-2).

```
;*******************************************************************************
```

```
list   p=16f876      ; izbira tipa čipa
       include <p16f876.inc>; vključitev datoteke z definicijami simbolov registrov

BCKW   equ    0x20   ; definicija naslova spremenljivke - rezerva (backup za shranj. W)
BCKST  equ    0x21   ; definicija naslova spremenljivke - (backup za shranj. STATUS)

;-------------------------------------------------------------------------------------
; Tukaj nastavim Reset vektor
;-------------------------------------------------------------------------------------
       org    0x0    ; RESET vektor
       goto   Main   ; skok na glavni program, Main
       org    0x4    ; prekinitveni vektor, Interrupt
       goto   Int    ; skok v prekinitveno rutino - Int
       org    0x5    ; začetek programske kode

;-------------------------------------------------------------------------------------
; Prekinitveni program
;-------------------------------------------------------------------------------------
RefDisp
       ;---------------------------------------------------------------------------
              ; Vpiši program, ki se izvede ob prekinitvi - časovno zahtevne operacije
       ;---------------------------------------------------------------------------
       return

;-------------------------------------------------------------------------------------
; Prekinitvena rutina
;-------------------------------------------------------------------------------------
Int                          ; Shranjevanje stanja delovnega in statusnega registra
       movwf  BCKW           ; backup za delovni register W
       swapf  STATUS, W      ;
       clrf   STATUS         ; brisanje statusnega registra
       movwf  BCKST          ; backup za STATUS
       ;---------------------------------------------------------------------------
       ; Prekinitveni program
       ;---------------------------------------------------------------------------
       call   RefDisp        ; skok v uporabnikovo rutino prekinitvenega programa
       bcf    INTCON, T0IF   ; prekinitveni program je bil izveden
       ;---------------------------------------------------------------------------
                             ; Vzpostavitev starega stanja
       swapf  BCKST, W       ;
       movwf  STATUS         ; vzpostavi stari STATUS
       swapf  BCKW, F        ;
       swapf  BCKW, W        ; vzpostavi staro stanje delovnega registra W
       retfie

;-------------------------------------------------------------------------------------
; Glavni program
;-------------------------------------------------------------------------------------
Main
       ;---------------------------------------------------------------------------
                     ; Vpiši ukaze za inicializacijo (vhodno/izhodna enota, ...)
       ;---------------------------------------------------------------------------
       ;--------------------; Konfiguracija OPTION_REG (za prekinitve)
       bcf    STATUS,RP1     ;
       bsf    STATUS,RP0     ; BANK 1
       movlw  0x3            ; preddelilnik 1/16, pull-up vklopljeni, notranji pulzi
       movwf  OPTION_REG     ; inicializacija Timer0
       movlw  B'10100000'    ; nastavitev prekinitev ( GIE  = 1 , T0IE = 1 )
       movwf  INTCON         ; omogočitev prekinitev Timer0
       bcf    STATUS,RP0     ; BANK 0
       ;---------------------------------------------------------------------------
       ;---------------------------------------------------------------------------
                             ; Vpiši glavni program - časovno nezahtevne operacije
```

```
        ;-------------------------------------------------------------------------------
        end                     ; konec programa (psevdoukaz)
;-------------------------------------------------------------------------------
```

Program 6-2: Zgled ogrodja programa v zbirnem jeziku pri uporabi prekinitev

Za razumevanje tematike je potrebno tudi predznanje iz osnov programiranja in številskih sistemov. Dodatna znanja iz programiranja v zbirnem jeziku so v: [2][4][6][7]. Za razumevanje programov, ki so kodirani v C-jeziku, je potrebno poznavanje osnov ANSI C strukturnega programiranja: [15][11][13][5]. Veliko rešitev praktičnih nalog z uporabo PIC mikrokrmilnikov je na spletnem portalu http://www.interq.or.jp/japan/se-inoue/e_pic6.htm. Podrobnejši opis delovanja časovnika Timer0 s primeri uporabe v PIC mikrokrmilnikih je na voljo v: [8][16][17][18].

Vprašanja za utrjevanje:

Napišite podprogram iz imenom InTMR0 za inicializacijo časovnika Timer0, ki bo periodično prožil prekinitve na 0,2 ms.

Rešitev:

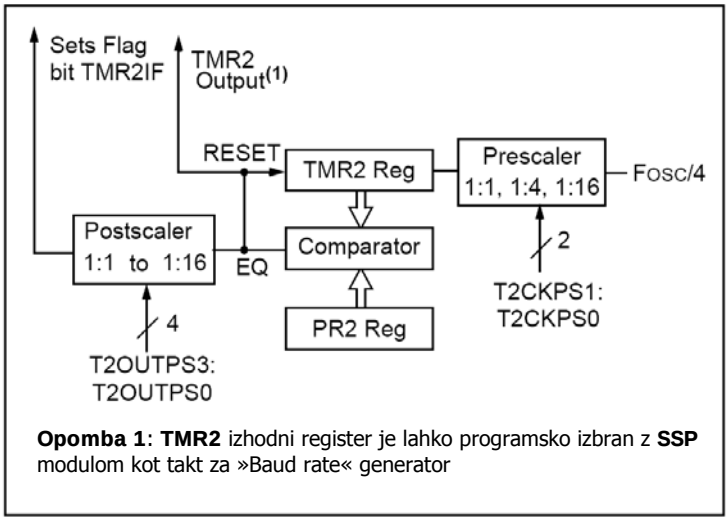
```
InTMR0 bsf     STATUS,RP0      ; 1 -> RP0 , izbira Bank1 za dostop do OPTION_REG
       movlw   b'00000001'     ;priprava vrednosti 0x01 za OPTION_REG
       movwf   OPTION_REG      ;preddelilnik casovnika Timer0 nastavimo na 4:1
       bcf     STATUS,RP0      ; 0 -> RP0, nazaj v Bank0
       movlw   b'10100000'     ;priprava vrednosti 0xA0 za INTCON, GIE=1, T0IE=1
       movwf   INTCON          ;Omogocimo prekinitve, ki jih sproza casovnik Timer0
       movlw   d'6'            ;priprava vrednosti (desetisko) 6 za TMR0
       movwf   TMR0            ;vrednost intervala: 0,2*(256-6)*4=200us=0,2ms
       return                  ;konec podprograma
```

1. *Napišite podprogram z imenom InTMR0_2 za inicializacijo časovnika Timer0, ki bo periodično prožil prekinitve na 2,0 ms.*
2. *Koliko bitni je števec TMR0 in kakšne vrednosti preddelilnika lahko izbiramo ?*
3. *Kakšna je funkcija časovnega stražnika (Watch Dog Timer) ?*
4. *Ali lahko Timer0 uporabljamo, ne da bi prožil prekinitve ?*
5. *Kakšen interval bi dosegli v zgornjem rešenem zgledu, če ne bi vpisali nobene vrednosti v register TMR0 ?*

7. Generiranje pulzno-širinskih signalov (PWM)

Spoznali boste vmesnik mikrokrmilnika, ki omogoča generiranje pulznoširinskih signalov (PWM). Naučili se boste konfigurirati registre za izbiro periode (frekvence) in širine pulza vmesnikov PWM1 in PWM2.

Modul **Timer2** je 8-bitni časovnik/števec z vgrajenim preddelilnikom (Prescaler) in podelilnikom (Postscaler). Lahko se **uporablja tudi kot PWM časovna baza za generiranje dveh PWM signalov (z enako periodo) v okviru CCP podsklopa**.

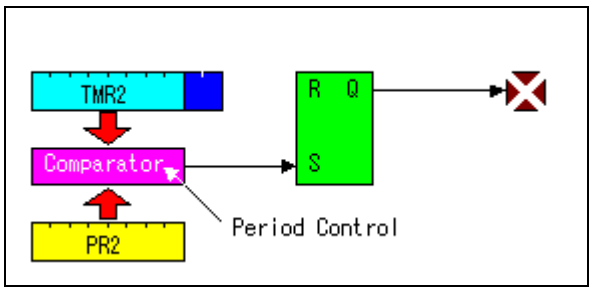


Slika 7-1: Zgradba modula Timer2

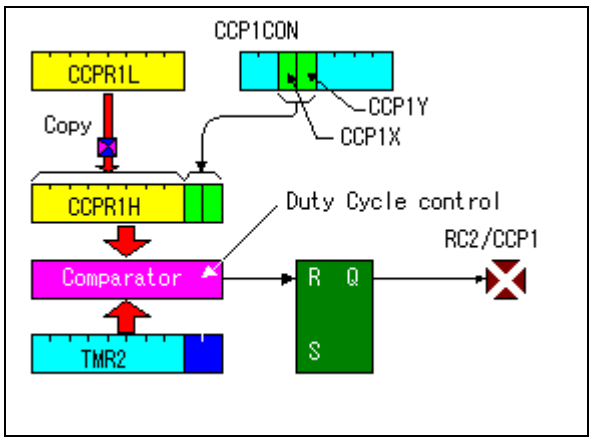
7.1. Zgradba in delovanje PWM vmesnika

CAPTURE/COMPARE/PWM (**CCP1** in **CCP2**) je podsklop modula **Timer2**, ki vsebuje 16-bitne registre in omogoča delovanje v naslednjih načinih:

- 16-bitni zajemalni (Capture) način;
- 16-bitni primerjalni (Compare) način;
- **PWM register za nastavitev širine pulza** (Master/Slave Duty cycle).

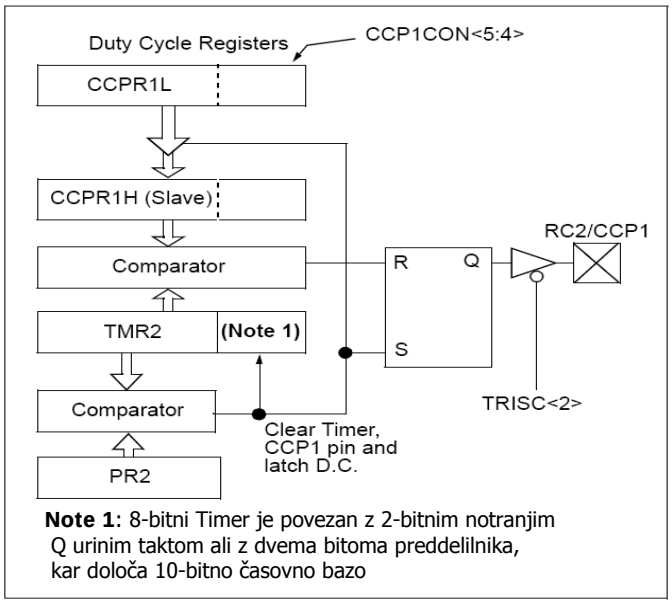


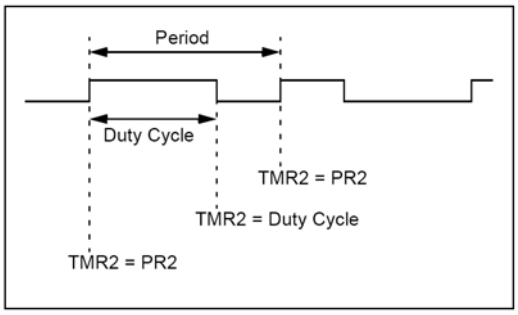
Slika 7-2: Nastavitev periode



Slika 7-3: Nastavitev širine pulza

PWM način delovanja omogoča, da na izhodih **CCP1** in **CCP2** programsko generiramo pulznoširinska signala z nastavljivo (največ 10-bitno) širino pulzov (angl.: Duty cycle) pri vnaprej izbrani frekvenci (le-ta je enaka za oba sklopa).





*Slika 7-5: **PWM** izhodni signal*

*Slika 7-4: Poenostavljena shema **PWM1** sklopa*

PWM1 izhod je na priključku **RC2/CCP1.**

PWM2 izhod je na priključku **RC1/CCP2.**

Periodo PWM signala nastavimo z vpisom 8-bitne vrednosti v register **PR2**. Izračunamo jo po naslednji formuli:

$PWM_{Perioda}$ = (**PR2**+1) * 4 * T_{osc} * $TMR2_{Prescaler}$, pri čemer je T_{osc} = 50 ns, če je frekvenca urinega takta 20 MHz.

Frekvenco (obratna vrednost $PWM_{perioda}$) lahko izbiramo od 1,22 kHz do nekaj 100 kHz (spodnja tabela), pri čemer se zmanjša ločljivost nastavitve širine pulza pod 10-bitov, če je frekvenca večja od 20 kHz.

PWM frekvenca	1.22 kHz	4.88 kHz	19.53 kHz	78.12kHz	156.3 kHz	208.3 kHz
Vrednost preddelilnika (1, 4, 16)	16	4	1	1	1	1
PR2 vrednost	0xFFh	0xFFh	0xFFh	0x3Fh	0x1Fh	0x17h
Največja ločljivost (biti)	10	10	10	8	7	5.5

Tabela 7-1: Vrednosti ločljivosti pri različnih frekvencah in periodah

Širino pulza (Duty cycle) nastavimo z vpisom višjih 8-bitov v register CCPR1L in spodnjih dveh bitov 10-bitne vrednosti v register **CCP1CON** (bita 5 in 4) v skladu z naslednjo formulo:

$PWM_{Duty\ cycle}$ = (**CCPR1L:CCP1CON<**5:4**>**) * T_{osc} * $TMR2_{Prescaler}$

Za čim večjo ločljivost se priporoča upoštevanje vrednosti **PR2**, kot jih prikazuje Tabela 7-1!

7.2. Opis registrov in primer programa

7.2.1. Opis registrov CCP1CON oz. CCP2CON

Nahaja se na naslovih 17h oz. 1Dh.

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	**-**	**CCPxX**	**CCPxY**	**CCPxM3**	**CCPxM2**	**CCPxM1**	**CCPxM0**
bit7	6	5	4	3	2	1	bit0

Legenda: R = omogočeno branje bita, W = omogočeno pisanje v bit, U = neuporabljen bit, beri kot '0', -n = vrednost ob POR resetu

bita 7-6: neuporabljena

bita 5-4: **CCPxX: CCPxY**: **spodnja dva bita 10-bitne vrednosti širine pulza PWM sklopa**

biti 3-0: **CCPxM3- CCPxM0**: izbira načina delovanja CCPx sklopa, za **PWM način: 11xx**

7.2.2. Opis registra T2CON

Nahaja se na naslovu 12h.

u-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	**TOUTPS3**	**TOUTPS2**	**TOUTPS1**	**TOUTPS0**	**TMR2ON**	**T2CKPS1**	**T2CKPS0**
bit7	6	5	4	3	2	1	bit0

Legenda: R = omogočeno branje bita, W = omogočeno pisanje v bit, U = neuporabljen bit, beri kot '0', -n = vrednost ob POR resetu

bit 7: neuporabljen

biti 6-3: **TOUTPS3: TOUTPS0**: delilnik frekvence za izhod Timer2 (od 0000 - 1:1 do 1111 - 1:16) – **nima vpliva na PWM**

biti 2: **TMR2ON**: **omogočitev Timer2** (**1 - Timer2 aktiven**, 0 - neaktiven)

bita 1-0: **T2CKPS1- T2CKPS0**: **izbira preddelilnika** za Timer2 v **PWM načinu: 00 – 1x,** 01 – 4x, 10 ali 11 – 16x

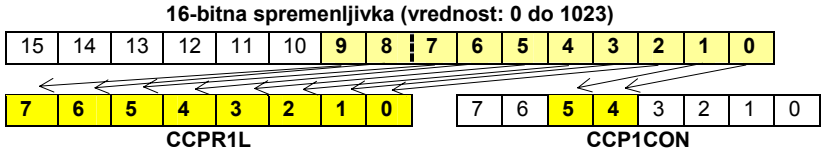
Postopek konfiguriranja CCP modula za PWM način delovanja:

1. Nastavitev PWM periode (frekvence) z vpisom **8-bitne vrednosti** v register **PR2**;
2. Nastavitev PWM širine pulza z vpisom višjih 8-bitov v register **CCPR1L** in spodnjih dveh bitov **10-bitne vrednosti** v register **CCP1CON** (bita 5 in 4)
3. Nastavitev **CCP1 priključka kot izhod** z brisanjem bita 2 (RC2=0) v registru **TRISC**;
4. Nastavitev **TMR2** preddelilnika in omogočitev Timer2 z vpisom ustrezne vrednosti (npr.: 04h) v register **T2CON**;
5. Konfiguriranje CCP1 modula za PWM način delovanja z vpisom (npr.: 0Ch) v register **CCP1CON**;
6. Ponavljanje (izvajanje) točke 2, ko želimo spremeniti širino pulza.

Navedeni postopek velja za **PWM1** sklop, ki ima izhod na priključku **CCP1/RC2**. Če želimo uporabiti tudi **PWM2** sklop, dobimo izhodni signal na priključku **CCP2/RC1**, registra v točki 2 in 5 (oz. formuli) pa nadomestimo z **CCPR2L** in **CCP2CON**.

V kolikor smo zadovoljni z **8-bitno ločljivostjo**, je dovolj da 8-bitno vrednost (med 0 in 255) prenesemo v register **CCPR1L**, medtem ko bita 5 in 4 v registru CCP1CON pustimo na izhodiščni vrednosti (0).

Če želimo generirati PWM signal s polno **10-bitno ločljivostjo** (med 0 in 1023), naložimo vrednost iz 16-bitne spremenljivke **v oba registra po postopku, kot je to prikazano** (Slika 7-6)**:**



Slika 7-6: Vrednost 16-bitne spremenljivke shrani v CCPR1L in CCP1CON

Primer programa v C-jeziku za generiranje signalov na izhodih PWM1 in PWM2:

```
// Demo program za prenos 10-bitne vred. na PWM1 in PWM2 (LD2 na RC1)
// Vrednost povecuj za 1 kv. (~5mV) ob vsakem pritisku na tipke

#include <pic1687x.h> // vkljucitev datoteke z definicijami simbolov
unsigned int Vredn;  // sprem. za vrednost PWM: 10-bit 0-1023

void main(void)
{
unsigned i;    // stevec za zakasnitev
//--------- Inicializacija registrov vhodno/izhodnih vmesnikov
TRISC=0;       // vsi pini porta C so izhodni
TRISA=0x10;    // vsi pini porta A so izhodni, razen RA4
INTCON=0;      // prekinitev ne bomo uporabljali
PORTC=0;       // izhode porta C postavi na 0
PORTA=0;       // izhode porta A postavi na 0

//--- inicializacija PWM1 ali/in PWM2
PR2=249;       // Perioda=(249+1)*4*50ns*1=50us ali 20kHZ pri 20Mhz takt, preddelilnik=1
CCPR1L=0;      // zacetna vrednost Duty cikla=0, moznost nastav: 0 do 1023 po 50ns
T2CON=0x04;    // bit 2: TMR2ON=1, aktiviraj Timer2, preddel.=1, b'00000100'
CCP1CON=0X0C; // za Timer 2 izberemo PWM nacin bit3=1, bit2=1, b'00001100'

Vredn=0;       // zacetna vrednost PWM1
// -------- jedro programa
       while(1)
       {
       PORTC=PORTC|0xF0;     // postavi RC7 do RC4 za test tipk
         if(RA4==1)          // ali je pritisnjena katerakoli tipka ?
         {
         Vredn=Vredn+1;      // DA, povecaj vrednost za 1 kvant
         RC0=1;              // vklopi LD1
  // ---- prenesi 10-bitno vrednost iz 16-bitne sprem. Vredn v PWM1 in PWM2 registre
         CCPR2L=CCPR1L=(unsigned char)(Vredn>>2); //zgornjih 8 bitov -> CCPR1L in CCPR2L
         CCP2CON=CCP1CON=(CCP1CON&0xCF)|((unsigned char)(Vredn<<4)); //spodnja 2 bita ->
       // CCPxCON<5,4>
         }
         else
         RC0=0;              // NE, izklopi LD1
       PORTC=PORTC&0x0f;           // brisi RC7 do RC4 za test tipk
       for(i=0; i<3333; i++);      // zakasnitev: 3333*3,0us=~10ms
       }
}
```

Program 7-1: Popoln primer programa v C-jeziku za vmesnik PWM

Za razumevanje tematike je potrebno tudi predznanje iz osnov programiranja in številskih sistemov. Dodatna znanja iz programiranja v zbirnem jeziku so v: [2][4][6][7]. Za razumevanje programov, ki so kodirani v C-jeziku, je potrebno poznavanje osnov ANSI C strukturnega programiranja: [15][11][13][5]. Veliko rešitev praktičnih nalog z uporabo PIC mikrokrmilnikov je na spletnem portalu http://www.interq.or.jp/japan/se-inoue/e_pic6.htm. Podrobnejši opis delovanja časovnika Timer2 in PWM vmesnika s primeri uporabe v PIC mikrokrmilnikih je v: [16][17].

Vprašanja za utrjevanje:

Napišite podprogram iz imenom InPWM1 za inicializacijo izhoda PWM1 s frekvenco 2,00 kHz.

Rešitev:

```
InPWM1 bsf    STATUS,RP0     ; 1 -> RP0 , izbira Bank1 za dostop do TRISC in PR2
       bcf    TRISC,2        ; RC2/CCP1 port za PWM1 definiramo kot izhod
       movlw  d'155'         ; desetisko 155 -> Za nastavitev frekvence signala 2kHz
       movwf  PR2            ; Perioda=(155+1)0.2us*16 = 499,2us -> f=1/499,2 = 2,003KHz
       bcf    STATUS,RP0     ; Nazaj v banko0
       clrf   CCPR1L         ; zacetna vrednost sirine pulza: CCPR1L = 0
       bsf    CCP1CON,3      ; 1 -> CCP1CON.3
       bsf    CCP1CON,2      ; 1 -> CCP1CON.2, bit3 in 2 na 1 v CCP1CON-PWM nacin delovanja
       movlw  b'00000111'    ; vrednost za T2CON, bit1,0 -> 1, preddelilnik je 1:16
       movwf  T2CON          ; 1 -> bit2, start casovnika TMR2
       return                ; konec podprograma
```

Napišite podprogram z imenom InPWM_2 za inicializacijo časovnika izhoda PWM2 s frekvenco 5,00 kHz.

1. *Koliko bitno vrednost lahko vpišemo kot vrednost za širino pulza ?*
2. *Kakšne vrednosti preddelilnika lahko izbiramo ?*
3. *Kakšna je ločljivost nastavitve širine pulza (najkrajša in najdaljša širina pulza v ns) pri frekvenci 20 kHz ?*
4. *Ali PWM vmesnik proži prekinitve ?*
5. *Kakšna je najmanjša širina pulza (razen 0) v zgornjem (rešenem) primeru ?*

8. Literatura

[1] James M. Sibigtroth: Understanding small microcontrollers; Prentice Hall, 1993; ISBN 0-13-089129-0, 250 strani

[2] Nebojša Matić: PIC mikrokontroleri; Mikroelektronika, Beograd 2002; ISBN 86-902189-4-7; 276 strani

[3] Matjaž Colnarič: Osnove digitalne tehnike v računalništvu; UM-FERI, Maribor, 2002; ISBN 86-435-0435-8; 138 strani

[4] Miran Rodič, Janez Pogorelc: Navodila za delo z modulom MPU-PIC16F876 (interno gradivo), [http://www.ro.feri.uni-mb.si/predmeti/mikro_el/nav_mpu_pic.pdf]; UM-FERI, Maribor, 2002

[5] Darko Dužanec: Programiranje PIC-ev v C-ju (serija člankov), Svet elektronike, maj 2004 do julij 2004, številke 109-111, ISSN 1318-4679

[6] Jernej Škvarč: PIC od začetka (serija 9 člankov), Svet elektronike, marec 2003 do januar 2004, številke 96-105, ISSN 1318-4679

[7] Microchip: spletna stran proizvajalca PIC mikrokrmilnikov, [http://www.microchip.com/]

[8] Microchip: Priročnik za mikrokrmilnike PIC16F87x (angl.), [http://www.ro.feri.uni-mb.si/predmeti/mikro_el/FTP/PIC16F876_30292b.pdf], 2000, 200 strani

[9] Microchip: Priročnik za MPLAB ICD (angl.), [http://www.ro.feri.uni-mb.si/predmeti/mikro_el/FTP/ICD_51184d.pdf], 2000, 104 strani

[10] Microchip: MPLAB V5.70 integrirano programsko okolje (program), [http://www.ro.feri.uni-mb.si/predmeti/mikro_el/FTP/mp57full.zip], 2003, 13 MB

[11] Brian W. Kernicham, Dennis M. Ritchie: Programski jezik C (slovenski prevod), 1990, UM-FERI Ljubljana, 240 strani

[12] HI-TECH Software: spletna stran proizvajalca C-prevajalnika, [http://www.htsoft.com/]

[13] HI-TECH Software: priročnik C-prevajalnika HI-TECH PICC Lite (angl.), [http://www.htsoft.com/files/demo/piccliteman.zip], 2002, 358 strani, 1,4 MB

[14] HI-TECH Software: C-prevajalnik HI-TECH PICC Lite (program), [http://www.htsoft.com/files/demo/picclite-setup.exe], 2,4 MB

[15] Miran Rodič: Uvod v programski jezik C, (interno gradivo) [http://www.ro.feri.uni-mb.si/predmeti/sis_meh/vaje/UvodC.pdf], UM-FERI, 2001, 14 strani

[16] Hobby Electronics: spletna navodila za uporabo PIC mikrokrmilnikov (angl.), [http://www.interq.or.jp/japan/se-inoue/e_pic.htm]

[17] Hobby Electronics: spletni primeri uporabe PIC mikrokrmilnikov (angl.), [http://www.interq.or.jp/japan/se-inoue/e_pic6.htm]

[18] MicrochipC.com: spletni primeri C-programov za PIC-e (angl.), [http://www.microchipc.com/]

[19] MicrochipC.com: Bootloader – najpreprostejši programator PICev (angl.), [http://www.microchipc.com/PIC16bootload/]

[20] A. Tetičkovič, B. Brečko, B. Gračner: Mobilni robot za sledenje črti, Uvodni seminar skupine študentov 3.l. Mehatronika, [http://www.ro.feri.uni-mb.si/predmeti/skup_sem/projektu/MobRob_Tet_Bre_Gra.pdf], UM-FERI, 2003

[21] J. Horvat, I. Kodrič: Poročilo o izdelavi mobilnega robota, Uvodni seminar skupine študentov 3.l. Mehatronika, [http://www.ro.feri.uni-mb.si/predmeti/skup_sem/projektu/protokol Kodric_meha-rudi.pdf], UM-FERI, 2003, 14 strani

[22] Tekmovanje RoboT200X: spletni članki z opisi mini mobilnih robotov na osnovi PIC mikrokrmilnikov, [http://www.ro.feri.uni-mb.si/tekma/dodatne_informacije_nasveti.htm]