Osnove programiranja PIC16FXXX družine mikrokontrolerjev

Kazalo vsebine

Prvi koraki v svet mikrokontrolerjev s PIC16F877

Mikrokontrolerji so v zadnjem desetletju postali glavni gradniki elektronskih naprav. To jim je uspelo, ker so prilagodljivi (njihovo delovanje določa program, ki ga lahko enostavno spreminjamo), enostavni za uporabo (ko poznamo njihovo delovanje in programiranje), in poceni (cene preprostih mikrokontrolerjev so primerljive s cenami TTL in CMOS vezij, ki opravljajo osnovne logične funkcije).

Začetniku v tem zanimivem svetu povzroča največ preglavic njihovo programiranje, v manjši meri pa še povezovanje z drugimi elektronskimi vezji. V tem delu se bomo seznanili z obojim. Priročnik sem napisal za začetnike in tiste, ki bi radi izvedeli kaj več o delovanju PIC16F87x družine mikrokontrolerjev.

Kaj pravzaprav je mikrokontroler?

Navzven je ploščica črne plastike iz katere moli veliko število kovinskih priključkov, ki jim bomo rekli "nožice". Nožice so povezava mikrokontrolerja z zunanjim svetom. Preko njih se pretakajo električni signali, ki se s pomočjo različnih električnih vezij, ki jih priključimo na mikrokontroler, spremenijo v obliko, zaznavno našim čutom: zvok, svetlobo, znake na prikazovalniku, vrtenje motorja...

V plastiki se skriva kompleksno vezje, ki je po zgradbi in načinu delovanja zelo podobno osebnemu računalniku (PC). Vezje je sestavljeno iz enostavnega mikroprocesorja, pomnilnika in vhodno/ izhodnega dela. Mikroprocesor je sestavljen iz aritmetično logične enote in registrov.

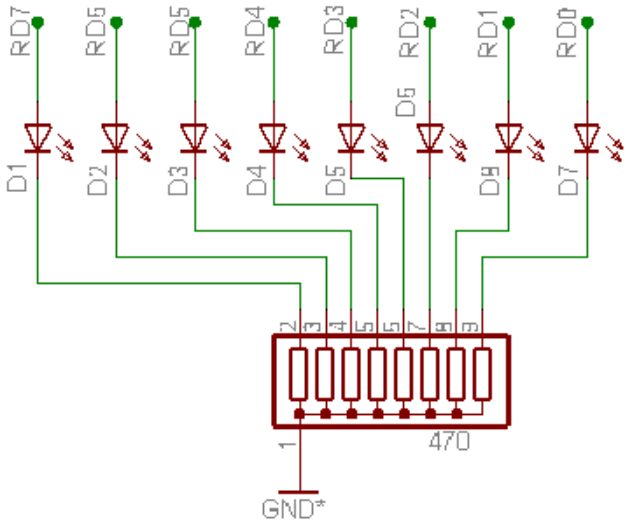
Aritmetično logična enota (ALE) je vezje, ki lahko opravlja enostavne aritmetične (seštevanje, odštevanje) in logične (ali, in, ne, izključno ali) operacije na enotah informacije, ki jim pravimo biti. Običajno se vse dogajanje vrti okrog skupine osmih bitov, ki ji pravimo "bajt". Zakaj osmih bitov? To število so si izbrali snovalci prvih računalnikov. Prav lahko bi si izbrali kakšno drugo manjše ali večje število (resnici na ljubo, med prvimi komercialnimi mikroprocesorji so bili taki, ki so obdelovali podatke dolžine štirih bitov). Mi se bomo ves čas ukvarjali z mikrokontrolerjem, ki lahko obdela po 8 bitov podatkov v enem koraku. Za tak mikrokontroler rečemo, da je "8 biten".

V mikroprocesorju se nahajajo še registri. To so pomnilniške lokacije, s katerimi si mikroprocesor pomaga pri obdelavi podatkov. Pri 8 bitnem mikroprocesorju imajo vedno velikost enega bajta. Nekateri mikroprocesorji znajo registre povezovati skupaj in podatke v njih obravnavati skupaj, na primer kot 16 bitne besede.

Pomnilnik je zgrajen iz pomnilniških lokacij, ki jim običajno pravimo besede. V našem primeru so to bajti. Le-ti so nanizani v zaporedju, vsak s svojim nespremenljivim naslovom, katerega vrednost se ob vsaki naslednji pomnilniški lokaciji poveča za ena. V vsako pomnilniško besedo lahko shranimo vrednosti bitov, ki mikroprocesorju pomenijo nekaj le v kombinaciji z ukazom, ki jih obdeluje. Digitalne vrednosti lahko pomenijo marsikaj. Običajno je programer tisti, ki določi njihov pomen. To so lahko binarne številke (s predznakom ali brez) ASCII kode znakov ali na primer naslovi lokacij, ki jih mikroprocesor potrebuje ob izvajanju ukazov. Mikroprocesor lahko iz pomnilnika bere in vanj shranjuje vrednosti. Pri PIC mikrokontrolerjih je pomnilnik ločen na dva dela: podatkovni (za podatke) in programski (za program). (Več o pomnilniku pri PIC16F877 na strani 13).

Prvi program!

Dovolj teorije. Posvetimo se bolj zanimivim temam. Napišimo program, ki bo prižgal 8 diod LED, ki smo jih po vezalni shemi na sliki priključili na mikrokontroler:



Kaj sploh je program?

Program je zaporedje ukazov, ki jih vpišemo v programski pomnilnik. Iz pomnilnika jih mikroprocesor jemlje enega za drugim in naredi to, kar mu ukažejo. Vsak mikroprocesor pozna le določeno število ukazov. V našem primeru je teh samo 35. Spoznali jih bomo postopoma skozi posamezne primere. Ukazi, ki jih bomo uporabljali, so ukazi zbirnika (ali asemblerja). Njihova oblika je taka, da imajo določen pomen v angleškem jeziku. Program, ki ga napišemo s pomočjo teh ukazov, moramo še prevesti v strojni jezik. To je zaporedje enic in ničel, ki ga razume mikroprocesor. V tej obliki ga lahko s pomočjo posebnega vezja z imenom programator (glej dodatek B) vpišemo v mikrokontrolerjev programski pomnilnik.

Za pisanje programov bomo uporabili razvojno okolje MPLAB, ki ga proizvajalec za svoje mikrokontrolerje daje v brezplačno uporabo. Dobimo ga lahko na naslovu http://www.microchip.com.

V MPLABu lahko pišemo programe za mikrokontroler v več jezikih. Mi bomo uporabljali MPASM; zbirnik za PIC družino mikrokontrolerjev. Več o uporabi tega programa si oglejte v dodatku A.

Kako začnemo s pisanjem programa?

Začetek vseh naših programov bo enak:

```
list  p=16F877    ; povemo, za kateri mikrokontroler pišemo program

org  0            ; na katero lokacijo v prog. pomnilniku sodi naslednji ukaz
goto  4           ; ukaz povzroči skok na lokacijo z naslovom 4
org  4            ; naslednji ukaz sodi na lokacijo 4
nop               ; ukaz nop
end               ; konec programa
```

Tak program ne naredi nič pametnega, si pa na njem lahko ogledamo nekaj pomembnih stvari v zvezi s pisanjem programov v zbirniku.

Program pišemo vedno v treh stolpcih. Levi stolpec je rezerviran izključno za naslove, srednji za ukaze in desni za operande. Za desni stolpec ni nujno potrebno, da ima poravnan levi rob, kot je to značilno za srednji stolpec. Levi stolpec se vedno začne na levem robu okna urejevalnika teksta. Za postavitev srednjega stolpca je smiselno uporabljati tipko "Tab":

"Tab"--------->

1. stolpec naslovi	2. stolp. ukazi	3. stolpec operandi ; komentarji

Naš program ima 6 vrstic. V njih sta samo dva ukaza v zbirniku (goto in nop); ostalo so vse ukazi prevajalniku (MPASM-ju). Vse, kar se nahaja v vrstici za znakom ";" prevajalnik ignorira. Znak ";" običajno označuje, da sledi komentar. Oglejmo si vsako vrstico po vrsti:

- "list" pove prevajalniku, da sledi ime mikrokontrolerja, za katerega pišemo program. Na podlagi tega podatka nas lahko prevajalnik obvesti, če v programu uporabljamo imena registrov, ki jih mikrokontroler ne vsebuje, ali naprimer prekoračimo pomnilnik.
- Ukazu prevajalniku "org " ("originate") vedno sledi naslov, ki pove, na katero lokacijo v programskem pomnilniku naj prevajalnik postavi ukaz, ki se nahaja v naslednji vrstici programa.
- "end" pove prevajalniku, da se na tem mestu program konča. Moramo ga uporabiti na koncu programa. V nasprotnem primeru nam le-ta javi napako.
- "goto" je ukaz v zbirniku, ki povzroči skok na novo lokacijo v programskem pomnilniku. Vedno mu sledi naslov nove lokacije.
- "nop" ("no opperation") je ukaz v zbirniku, ki ne naredi nič; zaseda le eno mesto v programskem pomnilniku in zanj mikroprocesor porabi določen čas, ki je potreben za izvršitev enega ukaza.

Dopolnimo sedaj naš program tako, da bo prižgal diode LED na vmesniku D!

Na vezalni shemi vidimo, da so diode priključene na nožice mikrokontrolerja z oznakami od RD0 do RD7. Te nožice pripadajo vmesniku D, ki lahko upravlja z osmimi nožicami. To počne s pomočjo registra z imenom PORTD, preko katerega lahko pošilja podatke iz mikrokontrolerja v zunanji svet ali pa ugotavlja, kakšna so logična stanja na teh nožicah, če jih vsiljujejo zunanje naprave. Obojega seveda ne more početi hkrati. Katero smer bodo imeli podatki, lahko nastavimo v registru TRISD. Če za pripadajočo nožico v tem registru vpišemo "0", bo nožica nastavljena kot izhod, v primeru logične enice "1", pa kot vhod. (Več o tem v poglavju o vmesnikih mikrokontrolerja!)

V našem primeru pošiljamo podatke iz mikrokontrolerja. Vrednost vseh bitov TRISD registra bo torej 0. Register PORTD se nahaja na lokaciji 0x08 in register TRISD na lokaciji 0x88 v podatkovnem pomnilniku.

```
list p=16F877
org 0
goto 4
org 4
bsf 0x03,5
clrf 0x88
bcf 0x03,5
movlw 0xff
movwf 0x08
end
```

Naj vas primer ne prestraši. Kot bomo takoj videli, se programe običajno piše na veliko bolj razumljiv način.

Program vsebuje kar nekaj novih ukazov, ki nekaj počnejo s pomnilniškimi lokacijami, katerih naslovi so zapisanimi v šestnajstiškem sistemu (oznaka »0x« pomeni, da sledi število zapisano v šestnajstiškem sistemu). Pa si jih poglejmo:

bsf: Ukaz »bsf« (»bit set file«) pomeni: postavi bit v »file« registru. Z besedo »file« se označujejo vsi registri, ki se nahajajo v podatkovnem pomnilniku. Registra PORTD in TRISD sta tudi »file« registra. Vsi »file« registri so 8-bitni. Ukaz bsf ima vedno obliko:

bsf ime_file_registra, številka_bita.

Biti so označeni s številkami od 0 do 7. V primeru registra PORTD:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
RD7	**RD6**	**RD5**	**RD4**	**RD3**	**RD2**	**RD1**	**RD0**

Z ukazom `bsf 0x03,5` smo postavili 5. bit v registru z naslovom 0x03. Na tem naslovu se nahaja register z imenom STATUS. Vse njegove bite si lahko ogledate na strani xx. 5. in 6. bit v registru STATUS se uporabljata za preklop med deli podatkovnega pomnilnika, ki ima 4 dele. Več o tem si lahko preberete v poglavju o podatkovnem pomnilniku.

Med posameznimi deli podatkovnega pomnilnika se premikamo s pomočjo vrednosti v naslednji tabeli:

RP1	RP0	del RAMa	naslovi v RAMu
0	0	Bank 0	0x00 – 0x7f
0	1	Bank 1	0x80 – 0xff
1	0	Bank 2	0x100 – 0x17f
1	1	Bank 3	0x180 – 0x1ff

Ker se register TRISD z naslovom 0x88 nahaja na drugem delu RAMa (»Bank 1«), moramo preden lahko uporabimo ta register postaviti 5. bit v registru STATUS. Ob priključitvi mikrokontrolerjeve napetosti imata oba bita vrednost 0, zato nam bita 6 ni potrebno spreminjati.

clrf: Ukaz »clrf« (»clear file register«) povzroči, da se vsi biti v registru, katerega naslov sledi ukazu, zbrišejo. V registru so po takem ukazu same ničle. V našem programu smo z ukazom

`clrf 0x88` dosegli, da so vse nožice vmesnika D nastavljene kot izhodi, saj smo v register TRISD na ta način vpisali same »0«.

bcf: Ukaz »bcf« (»bit clear file register«) je nasprotje ukaza »bsf«. Ta ukaz zbriše bit v registru, katerega naslov navedemo ob ukazu. Mi smo ga uporabili, da smo 5. bit v registru STATUS zbrisali in s tem omogočili dostop do prvega dela podatkovnega pomnilnika (»Bank 0«). V tem delu se namreč nahaja register PORTD, ki ga potrebujemo v nadaljevanju programa.

movlw: Ukaz »movlw« (»move literal to working«) naloži v register W (»working« ali delovni register) konstanto, ki ima lahko vrednost med nič in 255. V programu smo v register W vpisali vrednost 0xff, ki je v desetiškem sistemu 255, v binarnem pa 11111111. V W smo torej vpisali same enice.

movwf: Naslednji ukaz v našem primeru »movwf« (»move working to file register«) prekopira vsebino registra W v izbran register, katerega naslov se nahaja ob ukazu. S tem ukazom smo v programu dosegli, da so se v register PORTD iz registra W vpisale enice.

Ob vpisu enic v register PORTD se na nožicah vmesnika D pojavi napetost 5V, ki povzroči, da diode LED zasvetijo. S tem smo tudi dosegli cilj, ki smo ga zastavili za naš program.

Napišimo sedaj program v obliki, ki bo običajna ob pisanju vseh nadaljnjih programov:

```
            list p=16f877

STATUS      equ 0x03
PORTD       equ 0x08
TRISD       equ 0x88

            org 0
            goto zacetek
            org 4
zacetek     bsf STATUS,5
            clrf TRISD
            bcf STATUS,5
            movlw b'11111111'
            movwf PORTD
konec       goto konec
            end
```

Glavna razlika med prejšnjo in novo verzijo programa je, da smo na začetku programa naslovom 0x03, 0x08 in 0x88 dali imena STATUS, PORTD in TRISD. Program je tako veliko bolj čitljiv, saj imamo namesto številk pojme, ki nam nekaj pomenijo. Besedica »equ« ni ukaz, temveč le pove prevajalniku, naj besedam v prvem stolpcu priredi vrednosti iz tretjega. Za prevajalnik postanejo tako STATUS, PORTD in TRISD konstante z vrednostmi 0x03, 0x08 in 0x88.

Proti koncu programa smo naredili še eno spremembo: namesto zapisa »0xff« smo uporabili zapis »b'11111111'«. Prevajalniku je namreč vseeno, v katerem številskem sistemu mu podajamo vrednosti. Prav lahko bi tudi napisali d'255'. Vsi trije zapisi pomenijo isto vrednost, le da je prvič zapisana v šestnajstiškem, drugič dvojiškem in tretjič desetiškem številskem sistemu.

Lastnosti mikrokontrolerja PIC16F877

PIC je družina mikrokontrolerjev proizvajalca Microchip iz Arizone. Ime je kratica v angleščini: »Pheripheral Interface Controller«, kar bi v slovenščino lahko prevedli kot »vezje za nadzor ali upravljanje z zunanjimi napravami«.

Družina PIC mikrokontrolerjev je v desetih letih svojega obstoja postala ena najbolj popularnih in najbolje prodajanih družin 8-bitnih mikrokontrolerjev na svetovnem tržišču. Levji delež te popularnosti ji prav gotovo pripada na račun mikrokontrolerja z oznako PIC 16F84, za katerega lahko najdete na internetu veliko programov in najrazličnejših projektov.

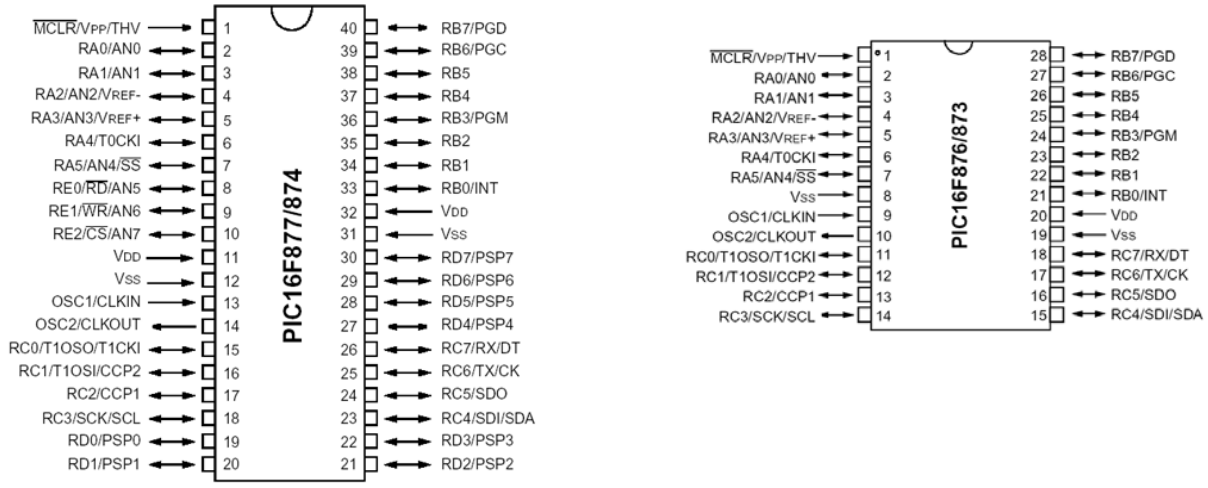
Uspešno pot nadaljuje skupina 16F87x, katere člani imajo vgrajene pretvonike AD, serijski in paralelni vmesnik, PWM krmilnik, tri časovne registre…

X iz oznake 16F87x nadomešča katerokoli število iz skupine: 0, 1, 2, 3, 4, 6 in 7. Mi si bomo ogledali 16F877 mikrokontroler, ki je po svojih lastnostih »najmočnejši« član v skupini.

Črka »F« v imenu označuje, da ima mikrokontroler pomnilnik tipa »flash«, kar pomeni, da je realiziran v obliki EEPROMa, ki ga lahko električno brišemo in v njega ponovno vpisujemo. Ta lastnost omogoča izdelavo enostavnih programatorjev in programiranje mikrokontrolerja direktno v vezju, v katerem se nahaja.

Vse, kar bomo povedali o PIC16F877, velja tudi za mikrokontrolerje PIC16F873, F874, F876. Med sabo se razlikujejo le v količini programskega pomnilnika in številu priključnih nožic: 16F873 in 16F876 nimata paralelnega vmesnika in zato 28 namesto 40 nožic. (glej prilogo A)

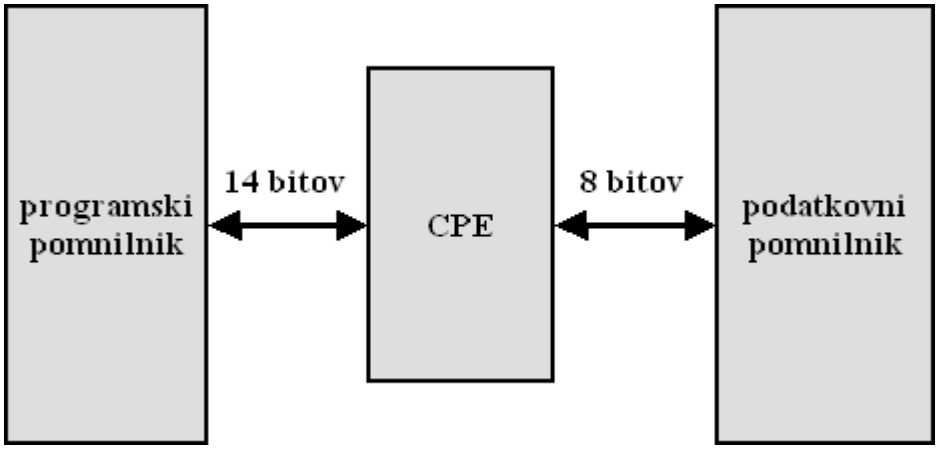
Vse 8-bitne PIC16xxx mikrokontrolerje povezuje tudi enotni zbirnik, ki pozna le 35 ukazov. Pridobljeno znanje na tem področju bo tako uporabno na celotni družini.



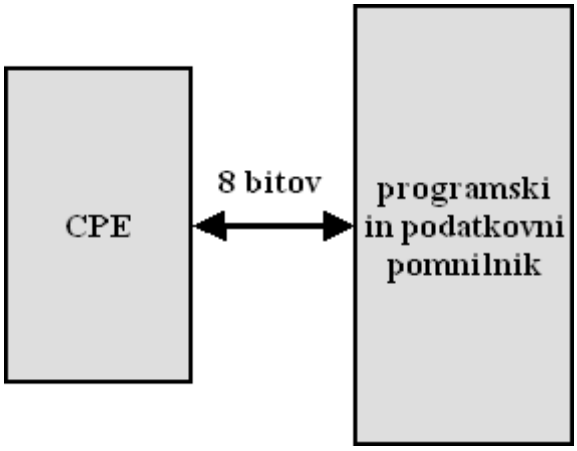
Slika 1: Prikaz razporeditve priključnih nožic in signalov družine PIC 16F87x

PIC16F877

Vsi PIC mikrokontrolerji imajo Harvard arhitekturo pomnilnika, za katero je značilno, da sta podatkovni in programski pomnilnik med sabo ločena. CPE (centralna procesna enota) zato lahko dostopa do ukazov v programskem in do podatkov v podatkovnem pomnilniku v enem ukaznem ciklu. Posledica tega je hitrejše delovanja mikrokontrolerja v primerjavi z Von Neumann ali Princeton arhitekturo, kjer si oba tipa pomnilnika delita isti prostor in isto vodilo do CPE.



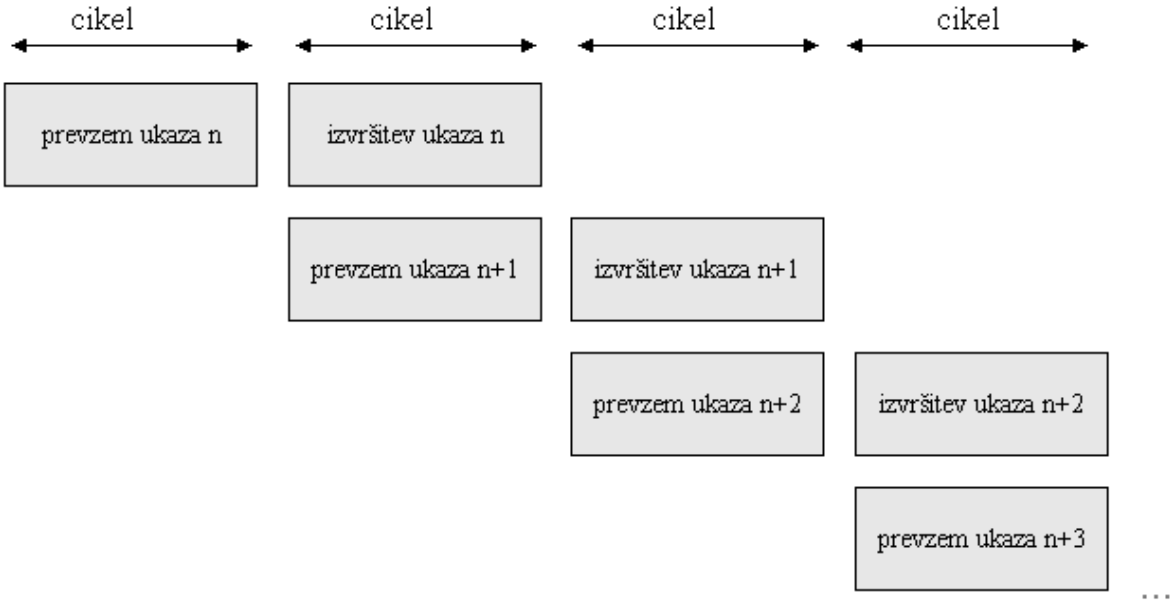
Slika 2: Harvard arhitektura. Omogoča različno širino podatkovnega in programskega vodila. Pri PIC16F877 se ukazi k CPE prenašajo po ukaznem vodilu, ki je 14-bitno, podatki pa po 8-bitnem podatkovnem vodilu.



Slika 3: Princeton arhitektura ima samo en pomnilniški prostor, ki si ga delita programska koda in podatki. CPE zato potrebuje med izvedbo enega ukaza več dostopov do pomnilnika. Prvi dostop prebere programski ukaz z naslednjimi dostopi, pa prebere podatke, ki jih ukaz potrebuje. Ta arhitektura ima nižjo hitrost izvajanja programa kot Harvard arhitektura, saj mora CPE pred branjem podatkov počakati, da se najprej prebere ukaz. Ta značilnost se označuje kot »Von Neumannovo« ozko grlo«.

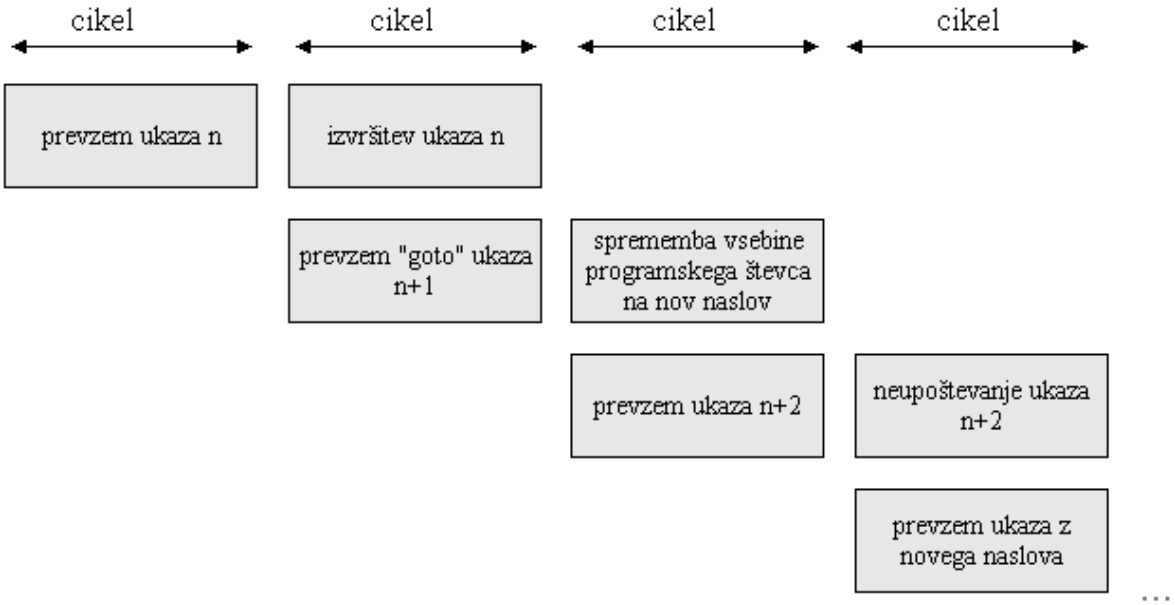
Pri izvajanju programa si vedno sledita dva koraka: prevzem in izvedba ukaza. PIC ima za izvajanje ukazov posebno »cevovodno« arhitekturo. S pomočjo le-te lahko v enem ciklu

izvede ukaz in prevzame naslednjega. V naslednjem ciklu izvede prej prevzeti ukaz in prevzame iz pomnilnika novega. Na ta način se lahko v vsakem ciklu izvede po en ukaz, kar pohitri delovanje mikrokontrolerja.



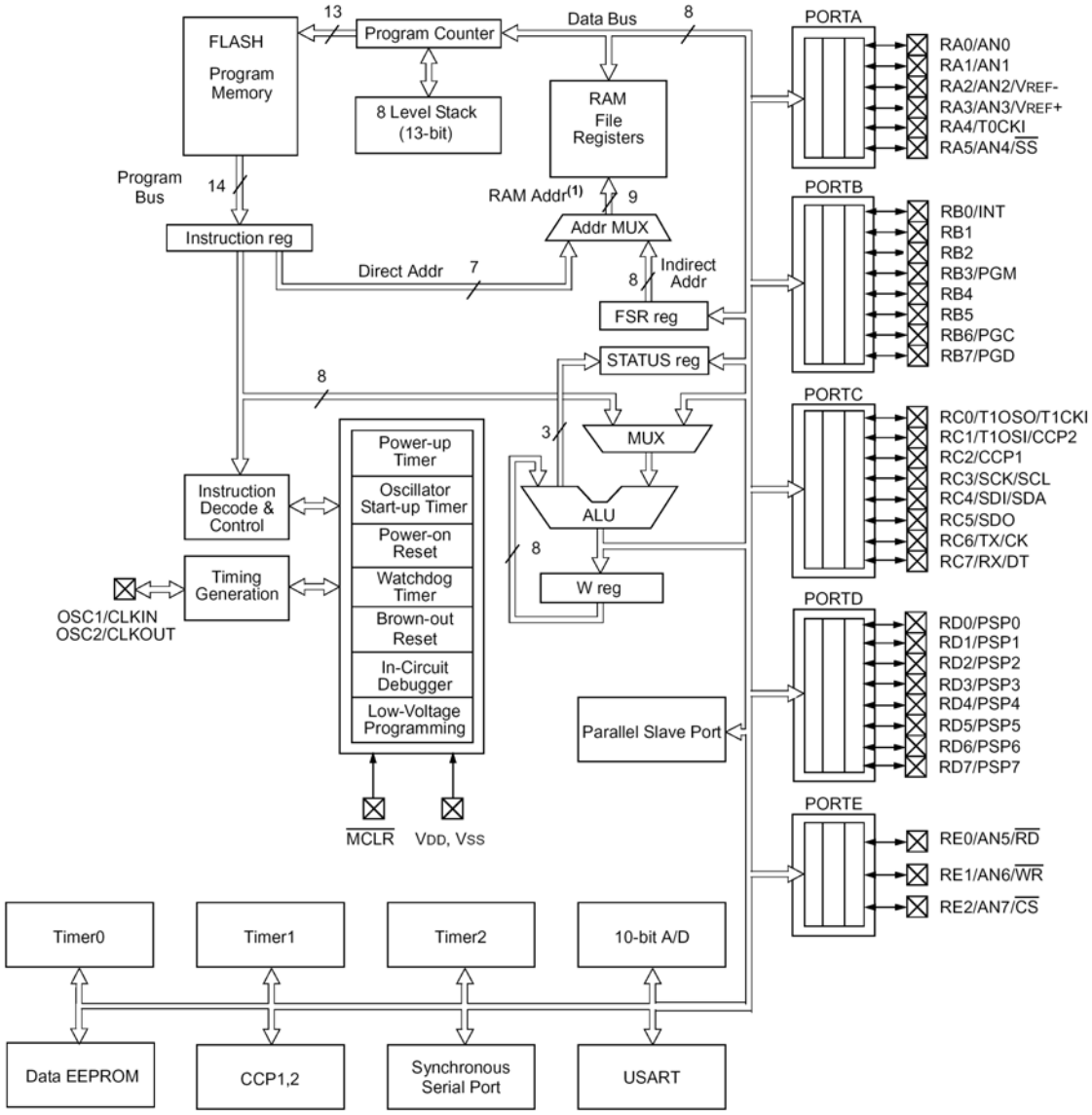
Slika 4: Potek izvajanja ukazov

Slika se spremeni, ko pride na vrsto programska vejitev. Zaradi skoka ob vejitvi na novo pomnilniško lokacijo je prevzeti ukaz napačen. Ukaz se zavrže in prevzame ukaz na novi lokaciji. Ukazi, ki povzročajo skoke v programu ali vejitve, se zato vedno izvedejo v dveh ciklih. Razmere ponazarja naslednja slika:



Slika 5: Izvajanje programa ob skoku

Notranja zgradba PIC16F877



Slika 6: Notranja zgradba PIC16F877

Srce mikrokontrolerja je CPE, ki jo sestavljata ALE (aritmetično logična enota) in register W. Na sliki lahko lepo vidimo opisano ločitev programskega in podatkovnega pomnilnika. Označene so tudi širine vseh vodil med posameznimi deli mikrokontrolerja.

Pomemben sestavni del vsakega mikrokontrolerja so vhodno/izhodna vrata (V/I). Pri PIC družini so poimenovana s črkami A, B, C… PIC16F877 ima vrata A, B, C, D in E (PORTA, PORTB, PORTC, PORTD, PORTE). Vrata so običajno sestavljena iz 8 nožic, s katerimi je mikrokontroler povezan v zunanji svet. Vsaka V/I vrata so povezana s pripadajočim registrom z istim imenom, s pomočjo katerega preko nožic prenašamo digitalne vrednosti. Število nožic in s tem tudi bitov v registru je lahko tudi različno od 8. Vrata A imajo tako na primer 6 nožic in vrata E samo 3.

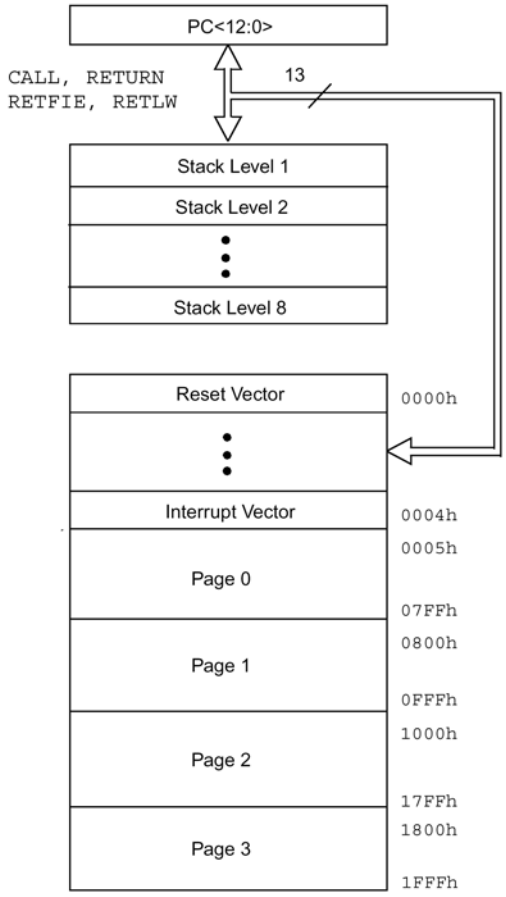
V mikrokontrolerju so vgrajena še druga vezja, ki smo jih včasih lahko kupili samo kot samostojne izdelke v svojih ohišjih. Le-ta na prvi pogled nimajo kakšne posebne vloge v

mikroprocesorskem sistemu. Šele z njihovo uporabo pa se izkaže zmogljivost, uporabnost in prilagodljivost mikrokontrolerja. Ti deli so: števci, časovna vezja, pretvorniki AD, serijska in paralelna vrata, krmilniki PWM. Mikrokontroler s pomočjo njih lahko nastopa v raznovrstnih aplikacijah.

Programski pomnilnik

Pri PIC16F877 je programski pomnilnik sestavljen iz 8K besed s po 14 biti. Do njih lahko dostopa s svojim 13 bitnim PC (»program counter« - programskim števcem). V povezavi s programskim števcem je tudi 8-nivojski sklad, v katerega se shranjujejo vrednosti PC ob programskih skokih. Značilnost programskega pomnilnika PIC mikrokontrolerjev je tudi, da ima na lokaciji 0000h »vektor reset« in na lokaciji 0004 »vektor prekinitev«. Na lokaciji vektorja reset se začne izvajanje programa ob resetu, na lokaciji vektorja prekinitev pa se nadaljuje izvajanje programa ob vsaki prekinitvi. Če uporabljamo prekinitve, se mora na lokaciji 0004h začeti programska rutina, ki se požene ob prekinitvi. V nasprotnem primeru se program ne bo pravilno izvedel.

Naj omenim še, da je programski pomnilnik razdeljen v 4 strani, katerih vlogo bomo podrobneje spoznali ob obravnavi programskega števca.



Slika 7: Programski pomnilnik

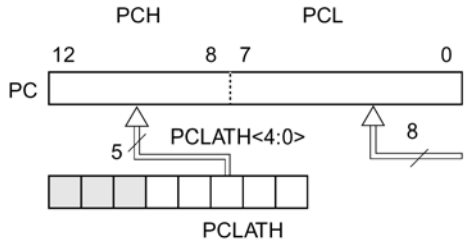
Programski števec

Ko se mikrokontroler prebudi iz reseta ob priključitvi napajalne napetosti, začne z izvajanjem programa. Da se ukazi izvajajo eden za drugim skrbi programski števec (PC), ki se po vsakem ukazu poveča za ena in vedno s svojo številsko vrednostjo kaže na naslednji ukaz. Programski števec se lahko spremeni tudi za vrednost, ki je različna od ena. To se zgodi ob »skočnih« ukazih in ob prekinitvi.

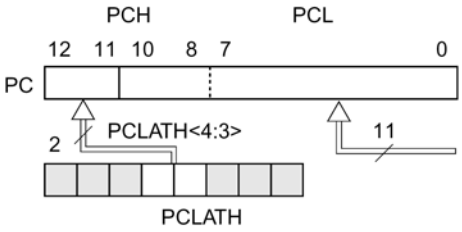
PC je 13-bitni števec. Sestavljen je iz dveh delov:

- Spodnjega 8-bitnega PCL registra, ki se nahaja na lokaciji 0x02 v podatkovnem pomnilniku. V PCL register lahko vpisujemo ali njegovo vrednost preberemo.
- 5 bitov, ki jih ne moremo brati, lahko pa v njih indirektno vpisujemo s pomočjo PCLATH registra, ki se nahaja na lokaciji 0ah v podatkovnem RAMu.

Kako se vzpostavi vrednost v PC ob vpisu v PCL ali ob skokih prikazujeta spodnji sliki.



Slika 8: Vzpostavitev naslova v PC ob vpisu v PCL



Slika 9: Vzpostavitev naslova ob izvedbi skočnega ukaza (goto, call)

Pri ukazih, ki direktno spreminjajo vrednost PCL, moramo biti pazljivi, da ne prekoračimo bloka z 256 bajti, v katerem se program trenutno nahaja, saj je le znotraj njega zaradi 8-bitnega PCL ukaz dosegljiv.

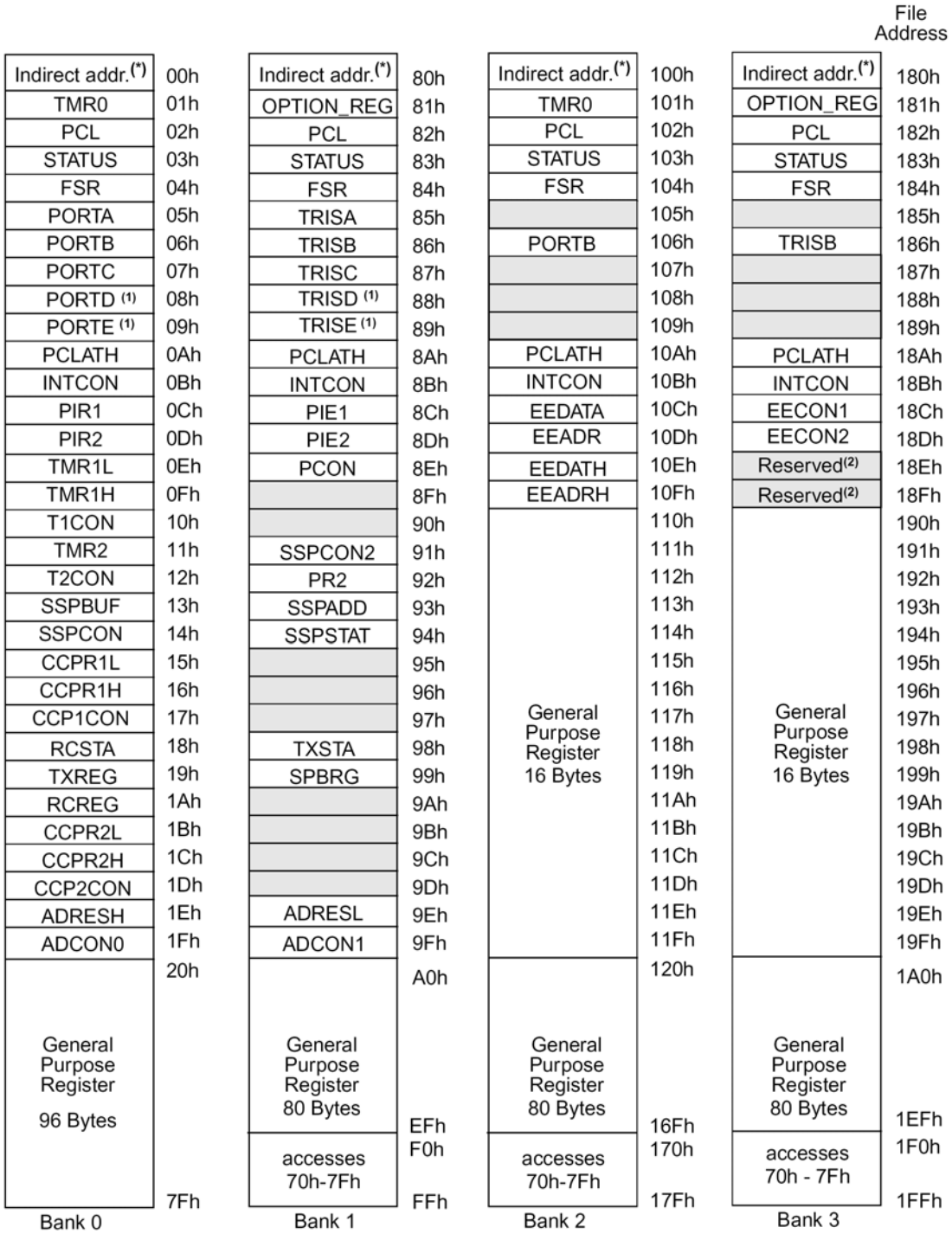
Omeniti velja še skoke med stranmi v programskem pomnilniku. Skočni ukazi namreč programskemu števcu zagotavljajo le 11 bitov naslova (glej sliko 9). Vse je v najlepšem redu, če ne preskočimo meje 2K (preko 2K, 4K in 6K), saj le ob teh prehodih pride tudi do

spremembe najvišjih dveh bitov PC. Če imamo v programu tak primer, moramo sami postaviti bita 3 in 4 v registru PCLATH. S tem programu omogočimo, da lahko skoči na želeno stran v programskem pomnilniku.

Ker se ob vrnitvi iz podprograma vrednost programskega števca prebere s sklada, nam za pravilno postavitev zgornjih dveh bitov ni potrebno skrbeti, saj je na skladu vedno shranjena celotna 13 bitna vrednost PC.

Podatkovni pomnilnik

Podatkovni pomnilnik je razdeljen v štiri dele s po 128 bajti. Na vsakem delu se nahajajo tako registri, ki imajo poseben pomen za CPE in registri oziroma pomnilniške lokacije, ki jih lahko poljubno uporabimo za svoje potrebe pri programiranju. Vsi registri v podatkovnem pomnilniku se imenujejo »file« registri. Med posameznimi deli v pomnilniku se sprehajamo s pomočjo dveh bitov: RP0 in RP1, ki se nahajata v registru status.



Slika 10: Podatkovni pomnilnik

V primeru, da bi radi nekaj vpisali na primer v register TRISA, ki se nahaja v drugem delu pomnilnika (Bank1), moramo v bit 6 (RP1) v registru STATUS vpisati 0, v bit 5 (RP0) pa 1.

Vse možne vrednosti prikazuje tabela:

RP1	RP0	del RAMa
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

Za prejšnji primer bi lahko v programu uporabili ukaza:

```
bcf STATUS,6
bsf STATUS,5
```

Lahko uporabimo pa tudi macro "banksel", ki naredi isto:

```
banksel TRISA
```

V tem primeru moramo le navesti ime enega registra na delu RAMa, ki ga hočemo uporabljati. Mpasm poskrbi, da se bita postavita pravilno namesto nas.

Določeni registri s posebno funkcijo ("special function registers"), ki so pomembnejši za delovanje mikrokontrolerja, so prisotni na vseh delih pomnilnika. Register STATUS se nahaja na lokacijah 0x03, 0x83, 0x103 in 0x183. Vedeti moramo, da imamo preko teh naslovov vedno dostop do enega samega registra.

V nadaljevanju si bomo ogledali nekaj pomembnejših registrov.

Register STATUS

R = bit lahko preberemo
W = bit lahko spreminjamo
-n = vrednost ob POR (zagonskem resetu)

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 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

IRP .. bit za izbiro dela podatkovnega pomnilnika pri posrednem naslavljanju
1 = Bank 1,2
0 = Bank 3,4

RP1 : RP0 .. bita za izbiro dela podatkovnega pomnilnika
1 1 = Bank 3
1 0 = Bank 2
0 1 = Bank 1
0 0 = Bank 0

TO .. bit "timeout"
1 = po priključitvi napetosti ali po izvedbi clrwdt ali sleep ukaza
0 = prišlo je do prekoračitve "kuža pazi" timerja

PD .. bit "power down"
1 = po priključitvi napetosti ali izvršitvi ukaza clrwdt
0 = po izvršitvi ukaza sleep

Z .. bit "zero"
1 = rezultat aritmetične ali logične operacije je bil 0
0 = rezultat aritmetične ali logične operacije ni bil 0

DC .. bit "digit carry
1 = generiral se je prenos s 3. na 4. bit v bajtu (pri ukazih: addwf, addlw, sublw, subwf)
0 = ni prišlo do prenosa s 3. na 4. bit

C .. bit "carry"
1 = generiral se je prenos s 7. bita v bajtu (pri ukazih: addwf, addlw, sublw, subwf)
0 = ni prišlo do prenosa s 7. bita

PS2 : PS1 : PS0 .. nastavitev preddelilnika

PS2 PS1 PS0	*TMR0*	*WDT*
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

Register OPTION_REG

R/W-1	*R/W-1*	*R/W-1*	*R/W-1*	*R/W-1*	*R/W-1*	*R/W-1*	*R/W-1*
RBPU	**INTEDG**	**T0CS**	**T0SE**	**PSA**	**PS2**	**PS1**	**PS0**
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

RBPU .. omogoči "pull-up" upore na vratih B
1 = upori so onemogočeni
0 = upori so omogočeni

INTEDG .. nivo proženja zunanje prekinitve
1 = prekinitev ob naraščajoči fronti na RB0/INT pinu
0 = prekinitev ob padajoči fronti na RB0/INT pinu

T0CS .. izbira vira ure za TIMER0
1 = ura na RA4/T0CKI pinu
0 = notranja ura procesorja

T0SE .. izbira nivoja proženja pri zunanji uri
1 = povečanje ob padajoči fronti
0 = povečanje ob naraščajoči fronti

PSA .. dodelitev pred delilnika
1 = pred delilnik dodeljen WDT ("kuža pazi tajmerju")
0 = pred delilnik dodeljen TIMER0

Vhodno – izhodni vmesniki

Vmesniki mikrokontrolerja ali tudi "vrata" kot jih včasih poimenujemo, so sestavljeni iz skupine nožic (priključkov), ki opravljajo podobno funkcijo. Pri PIC16F877 mikrokontrolerju imamo 5 vhodno/izhodnih vmesnikov, ki so označena s črkami A, B, C, D in E. Z vmesniki lahko upravljamo s pomočjo registra z imenom, ki je sestavljeno iz besede PORT in črke vmesnika (na primer PORTB). Posamezni priključki, ki sestavljajo vmesnik, so označeni s črko R, črko imena vmesnika in številko priključka znotraj vmesnika (na primer RC3).

Preko vsakega priključka lahko v nekem trenutku prenašamo podatke samo v eni smeri: v ali iz mikrokontrolerja. Katero smer uporabljamo, povemo v TRIS registru. Če bit v TRIS registru, ki upravlja s priključkom, postavimo na logično "1", je priključek nastavljen kot vhod, v obratnem primeru pa kot izhod.

Primer za vrata B:

TRISB

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

PORTB

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

priključki

RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

Vsi priključki mikrokontrolerja se lahko uporabljajo kot običajni digitalni vhodi ali izhodi. Veliko jih ima tudi druge možnosti uporabe; te si bomo, za vsaka vrata posebej, ogledali v nadaljevanju.

PORTA

Vmesnik A je sestavljen iz 6 nožic:

PORTA

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
-	**-**	**RA5**	**RA4**	**RA3**	**RA2**	**RA1**	**RA0**

RA4 se lahko uporablja kot vhodna nožica za TIMER0.

Ostali priključki vmesnika A se lahko uporabljajo kot vhodi za pretvornik AD. Ob priključitvi napajalne napetosti so tudi tako nastavljeni.

PORTB

Vmesnik B ima 8 nožic:

PORTB

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
RB7	**RB6**	**RB5**	**RB4**	**RB3**	**RB2**	**RB1**	**RB0**

Na RB0 nožici je vhod za zunanje prekinitve. Tudi priključki RB4, RB5, RB6 in RB7 lahko sprožijo prekinitev, saj so povezani skupaj preko ALI logičnega vezja, katerega izhod lahko sproži prekinitev ob spremembi logičnega nivoja, če so nastavljene kot vhodi.

PORTC

PORTC

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
RC7	**RC6**	**RC5**	**RC4**	**RC3**	**RC2**	**RC1**	**RC0**

Vsi priključki vmesnika C imajo posebne možnosti uporabe. Prikazuje jih naslednja tabela:

RC7	- sprejemna nožica asinhronega serijskega vmesnika - podatki sinhronega serijskega vmesnika
RC6	- oddajna nožica asinhronega serijskega vmesnika - ura sinhronega serijskega vmesnika
RC5	- izhod za sinhroni serijski vmesnik
RC4	- vhod za sinhroni serijski vmesnik - vhod/izhod za I^2C vmesnik
RC3	- ura za sinhroni serijski in I^2C vmesnik
RC2	- vhod/izhod za CCP2 ali izhod PWM2
RC1	- vhod oscilatorja TIMER1 - vhod/izhod za CCP1 ali izhod PWM1
RC0	- izhod oscilatorja TIMER1 -vhod za števec TIMER1

PORTD

Vmesnik D lahko poleg običajnega digitalnega vhodnega/izhodnega načina delovanja, v povezavi z vmesnikom E, uporabimo kot mikrokontrolerjeva paralelna vrata.

PORTD

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
RD7	**RD6**	**RD5**	**RD4**	**RD3**	**RD2**	**RD1**	**RD0**

PORTE

Vmesnik E ima samo 3 nožice:

- **RE0/RD/AN5**
- **RE1/WR/AN6**
- **RE2/CS/AN7**

Vsaka izmed njih se lahko uporablja kot običajni TTL vhod/izhod, kot kontrolna nožica za paralelni prenos podatkov preko vmesnika D ali kot vhod analogno digitalnega pretvornika. V TRISE registru lahko za nožice nastavimo smer v kateri se bodo pretakali podatki. Poleg tega lahko s preostalimi biti nastavimo delovanje paralelnega vmesnika.

TRISE

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
IBF	**OBF**	**IBOV**	**PSPMODE**	**-**	**Smer RD2**	**Smer RD1**	**Smer RD0**

IBF	- na paralelnem vmesniku smo sprejeli bajt - na paralelnem vmesniku nismo sprejeli novih podatkov
OBF	- v oddajnem registru paralelnega vmesnika so še podatki - oddajni register je prazen
IBOV	- do ponovnega sprejema prišlo preden smo uspeli prebrati prejšnjo vrednost - ni prišlo do prekoračitve sprejemnega pomnilnika
PSPMODE	- vmesnik D nastavljen kot paralelni vmesnik - vmesnik D kot običajen vhodno/izhodni vmesnik

Ukazi

PIC16XXX družina mikrokontrolerjev pozna 35 ukazov. Našteti in z nekaj besedami opisani so v spodnji tabeli. Z opisom delovanja ukazov se bomo ukvarjali v naslednjem poglavju.

Ukazi so zaradi večje preglednosti združeni v tri skupine:

			STATUS	CIKLI
Ukazi v povezavi z bajti				
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	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**	skopiraj vsebino registra f (d=0 ->W; d=1 ->f)	Z	1
MOVWF	**f**	skopiraj vsebino W v register f		1
NOP	-	ne naredi ničesar		1
RLF	**f, d**	premakni vse bite v registru f za eno mesto v levo	C	1
RRF	**f, d**	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 (f=f-W)	C, DC, Z	1
SWAPF	**f, d**	zamenjaj zgornje in spodnje 4 bite v registru f		1
XORWF	**f, d**	izključno ali logična operacija med W in reg. f	Z	1
Ukazi v povezavi z biti				
BCF	**f, b**	zbriši 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 enak nič		1(2)
BTFSS	**f, b**	testiraj bit b v registru f in preskoči naslednji ukaz, če je bit b enak ena		1(2)
Ukazi v povezavi s konstantami				
ADDLW	**k**	seštej W in 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 »kuža pazi« časovnika naj bo enaka nič	TO, PD	1
GOTO	**k**	skoči na naslov k		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 servisiranje prekinitve		2
RETLW	**k**	vrni se s konstantno vrednostjo k v W		2
RETURN	-	vrni se iz podprograma		2
SLEEP	-	postavi mikrokontroler v »sleep« način delovanja	TO, PD	1
SUBLW	**k**	odštej vsebino W od konstantne vrednosti k (k=k-w)	C, DC, Z	1
XORLW	**k**	izključno ali logična operacija med W in konstanto k	Z	1

f.. običajno je to ime registra
k.. konstantna vrednost večja ali enaka 0 in manjša ali enaka 255
d.. kam se shrani rezultat operacije: d=0 .. shrani se v W
d=1 .. shrani se v f
b.. številka bita od 0 do 7

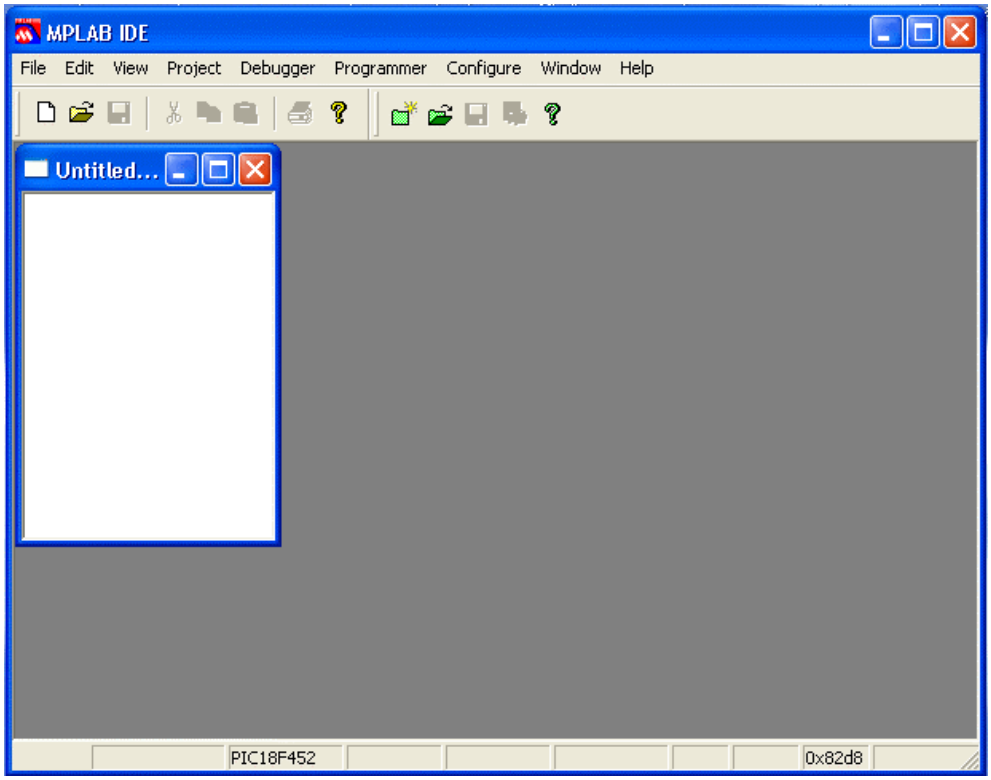
Dodatek A

Navodila za uporabo programa MPLAB IDE

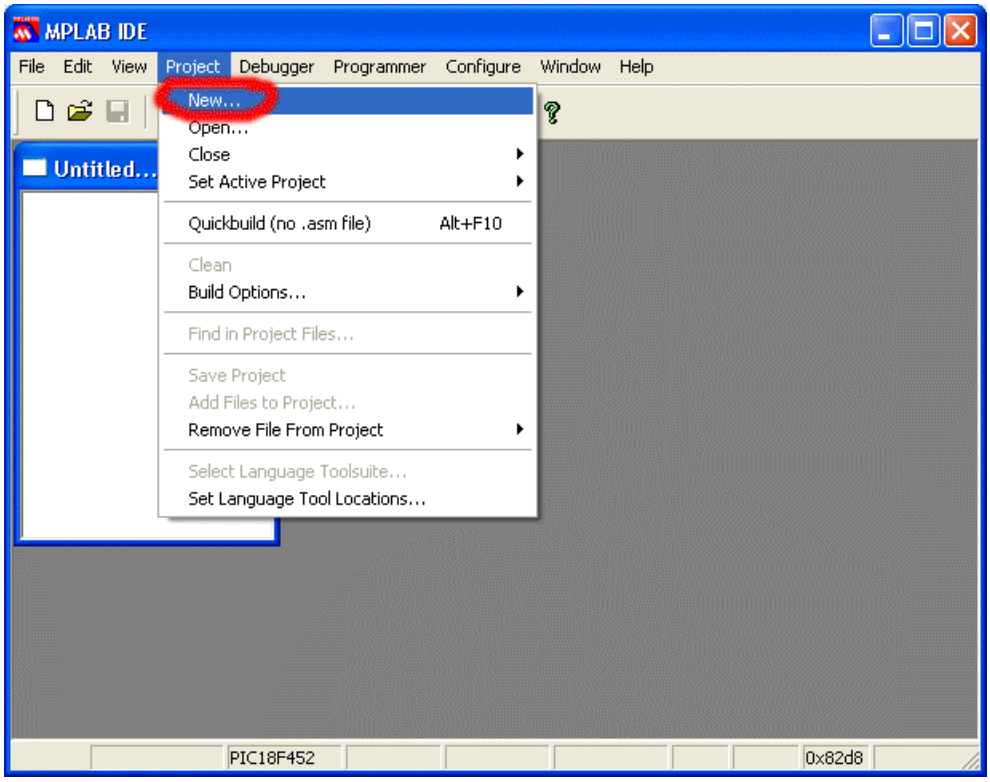
Mplab je razvojno okolje za PIC družino mikrokontrolerjev, ki ga podjetje Microchip ponuja uporabnikom svojih izdelkov zastonj. Program si lahko prenesemo na svoj računalnik z internetne strani http://www.microchip.com.

Ta navodila so napisana za program verzije 6.10. Prikazujejo, kako program napišemo, prevedemo in izvedemo preprosto simulacijo.

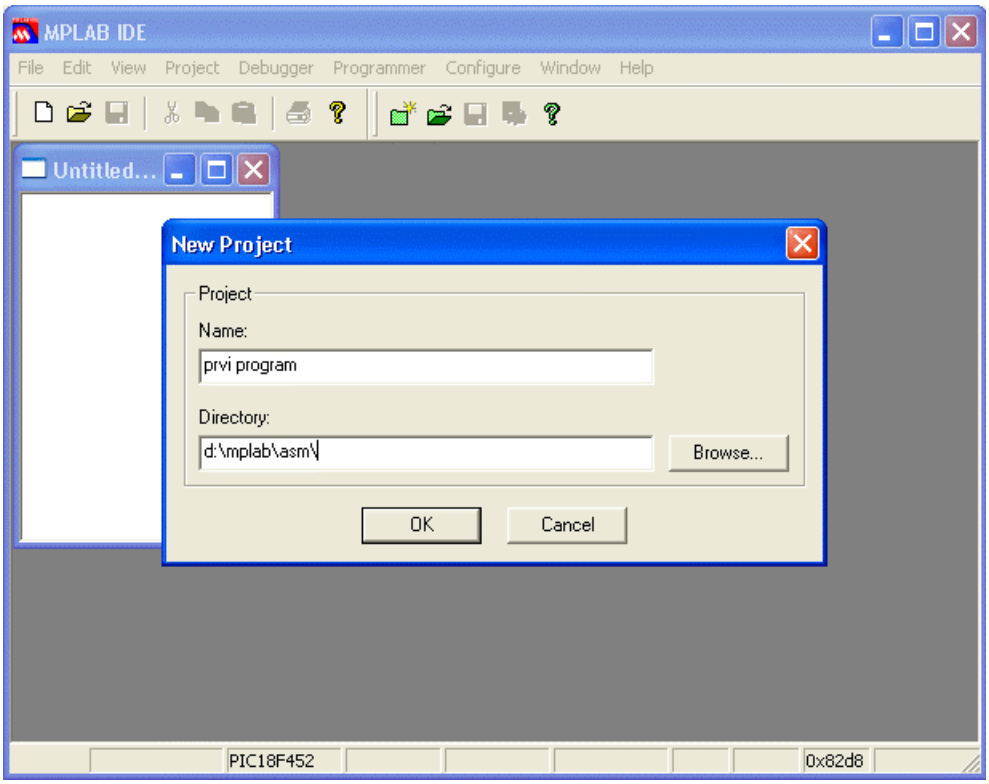
Ko poženemo program Mplab, se nam javi glavno okno programa:



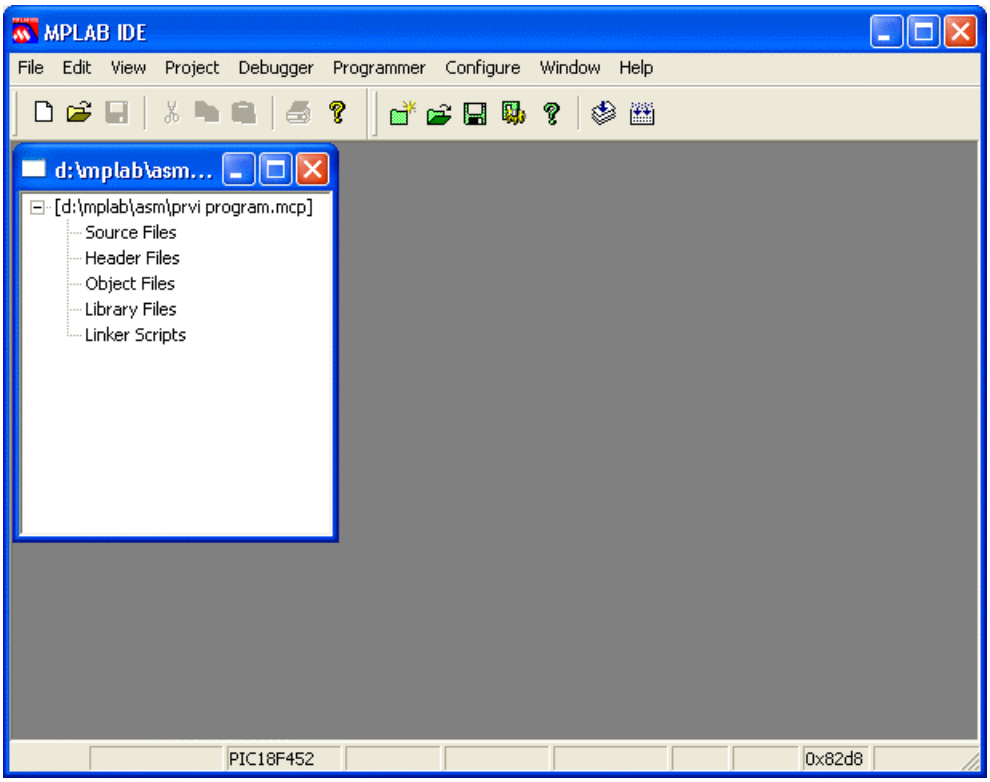
Najprej ustvarimo nov projekt:



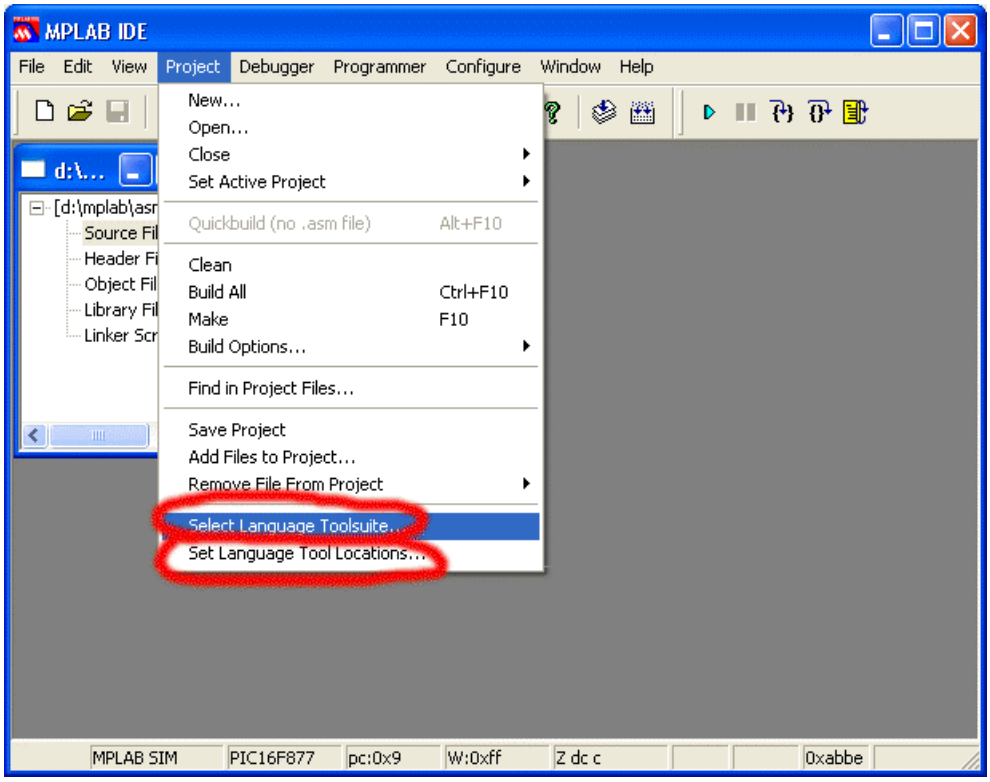
Projekt poimenujemo in shranimo v mapo z našimi programi:



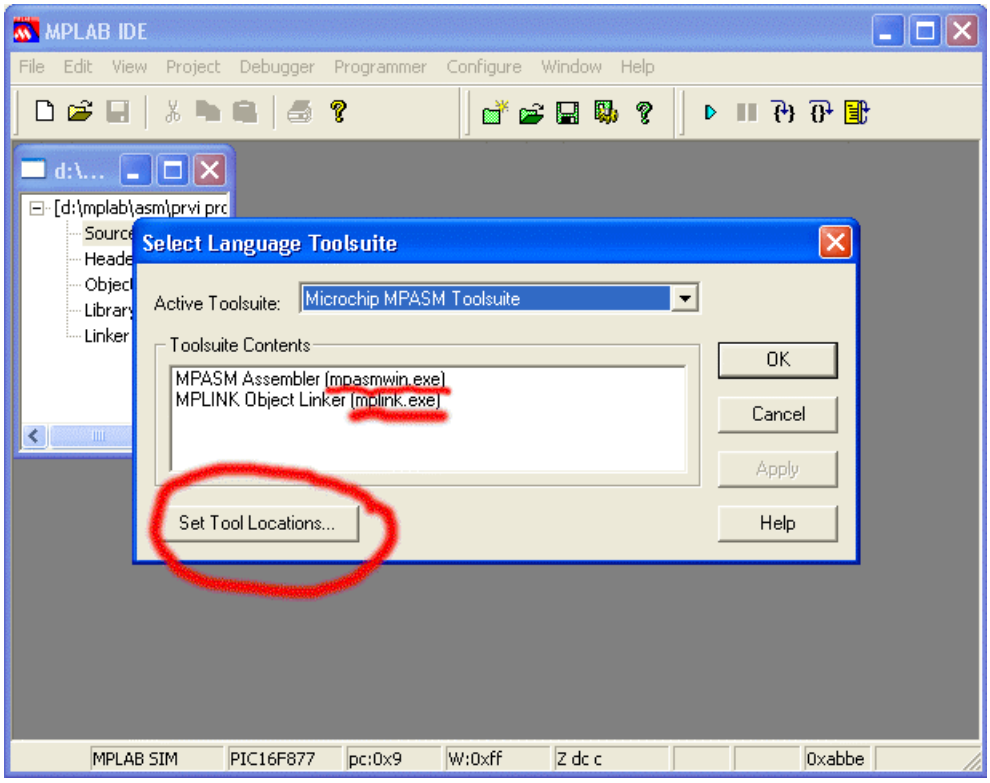
V oknu z lastnostmi projekta lahko sedaj vidimo ime našega novega projekta:



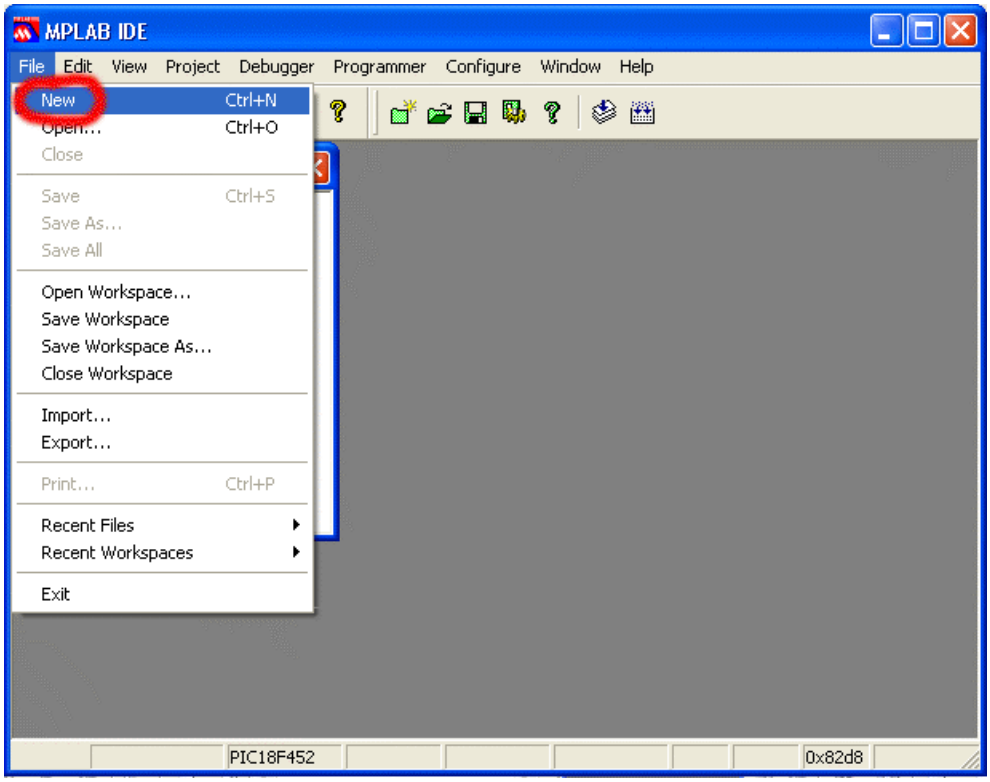
Če tega nismo še nikdar storili (npr. Uporabljamo novo nameščen program Mplab), moramo programu



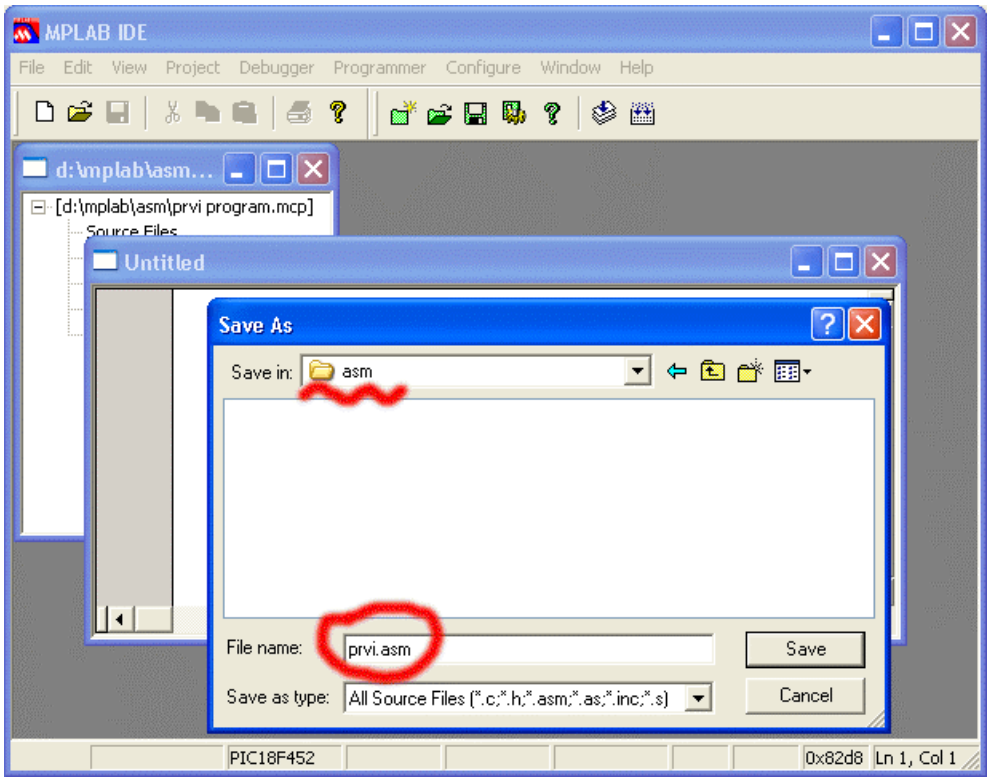
povedati, kateri jezik uporabljamo (mpasm) in pokazati pot do programov, ki jih potrebujemo za prevajanje naše kode:



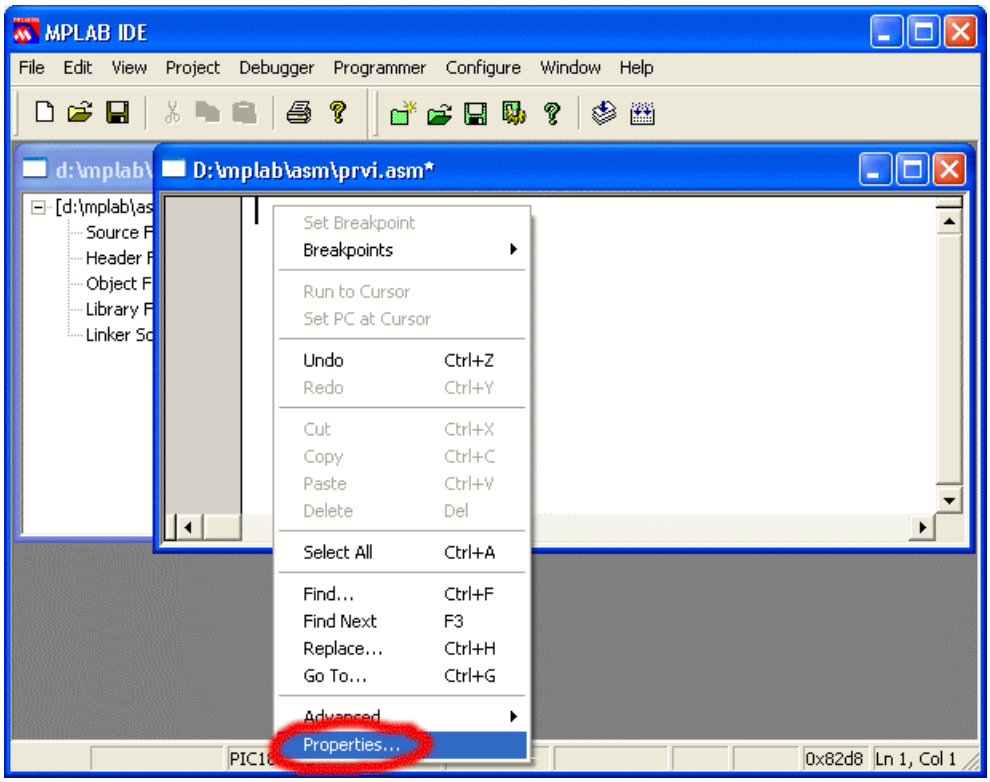
Sedaj odpremo novo datoteko, v kateri bo shranjena naša izvorna koda:

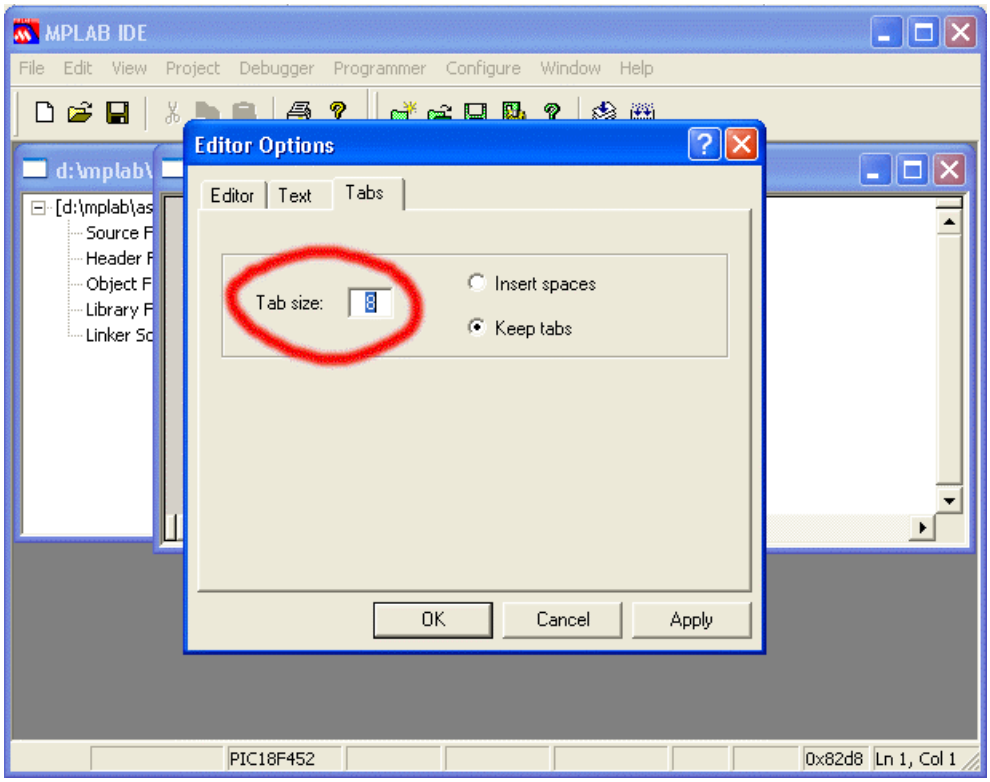


Našo datoteko poimenujemo in ji dodamo končnico ".asm". Shranimo jo lahko v mapo, kjer imamo shranjen projekt, ali katerokoli drugo mapo.

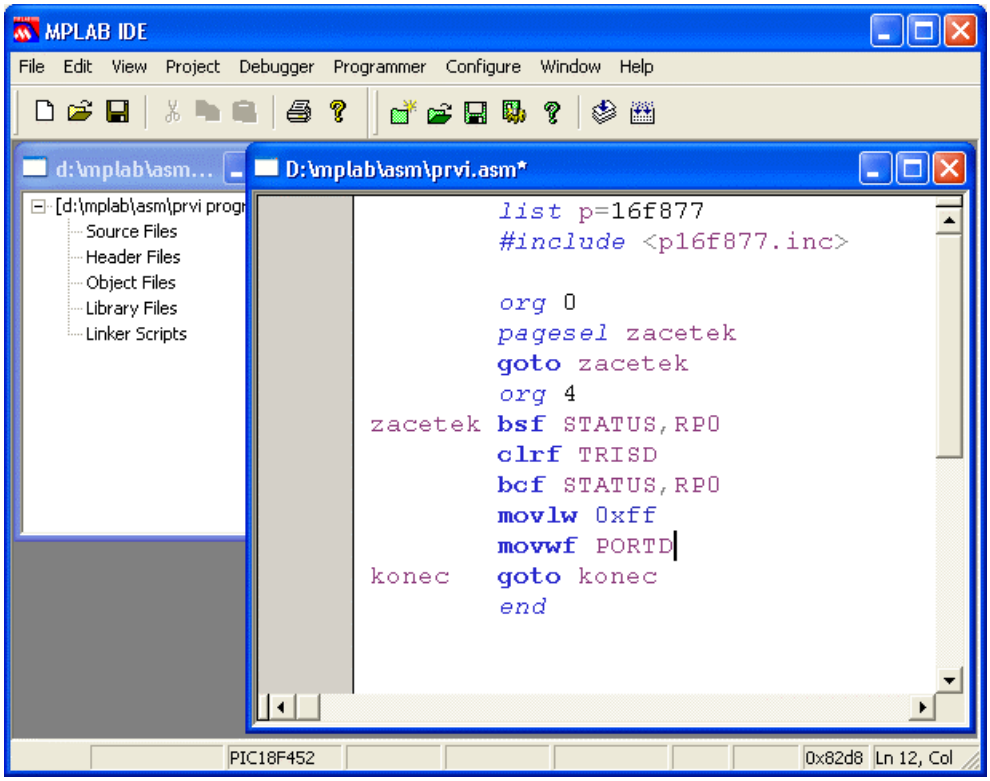


Odpre se nam okno urejevalnika izvorne kode. V naslovni vrstici lahko vidimo pot do datoteke našega programa. Preden začnemo s pisanjem programa bomo nastavili še odmik ("tabulator"), saj je privzeta vrednost 4 običajno premajhna. To naredimo tako, da z desno miškino tipko kliknemo na okno urejevalnika in iz priročnega menija, ki se pojavi, izberemo lastnosti ("properties"):

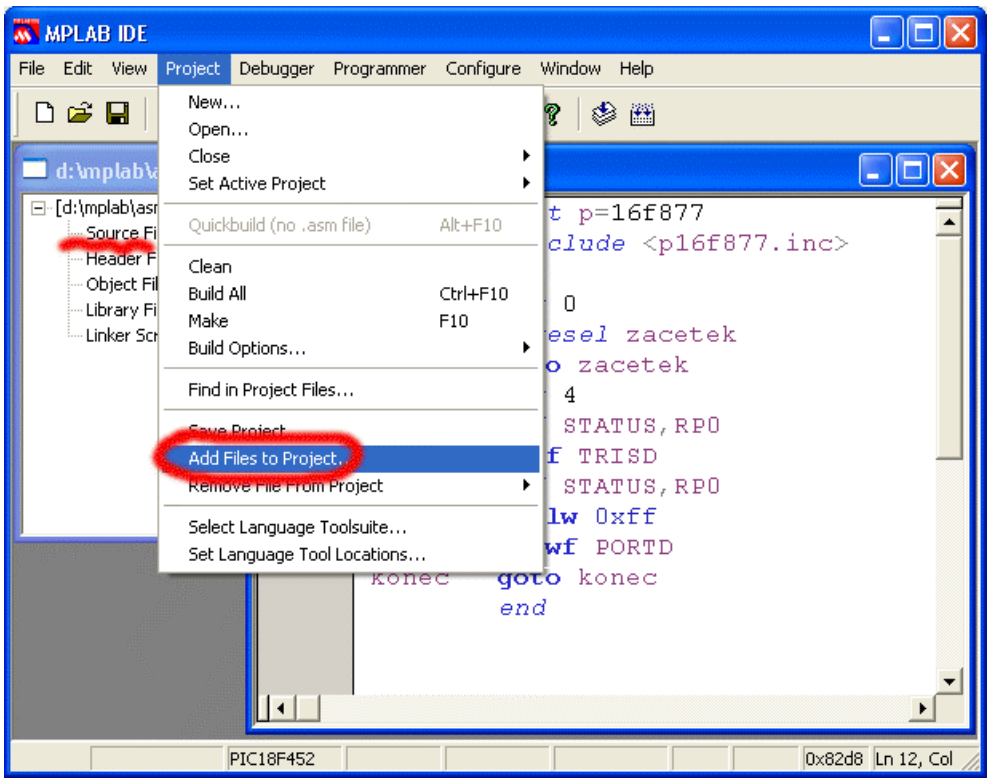




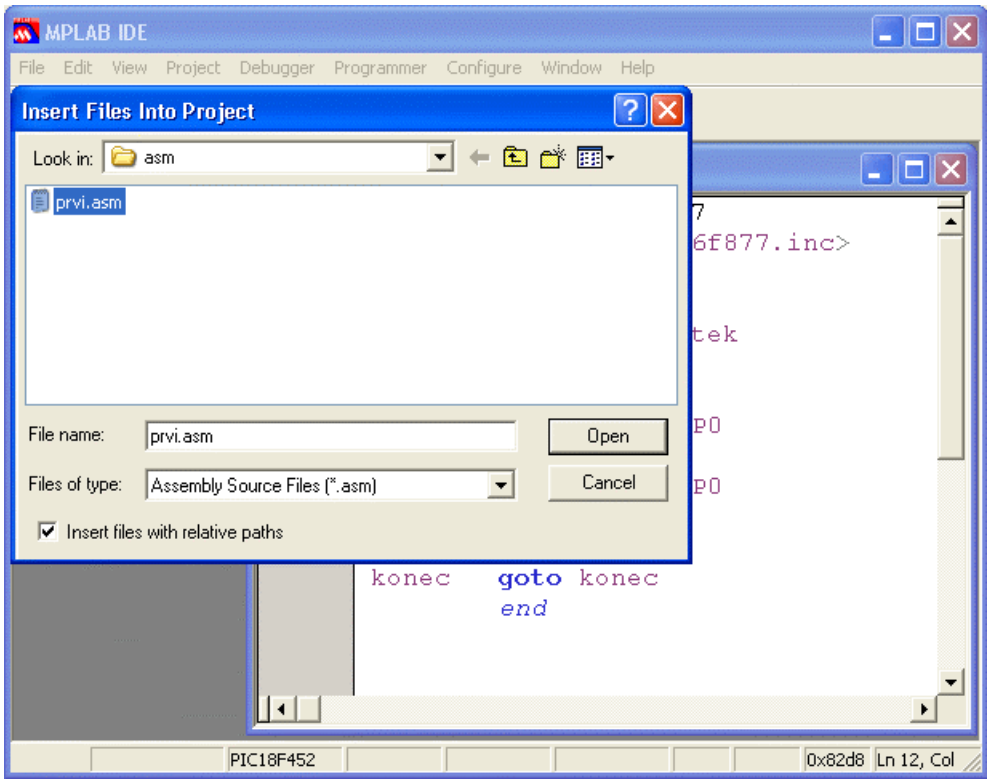
V urejevalniku sedaj lahko napišemo naš program:



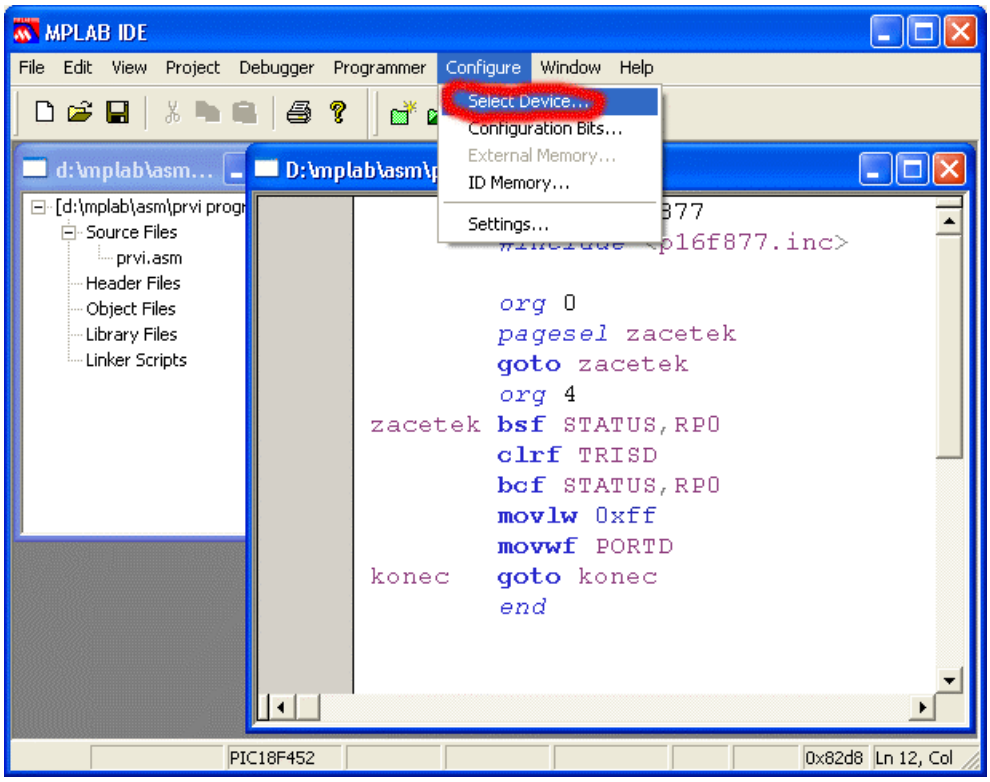
Program, ki smo ga pravkar napisali, moramo dodati k našemu projektu. To lahko naredimo tako kot kaže spodnja slika ali pa kliknemo z desno miškino tipko na vrstico kjer piše "Source fils" v oknu z lastnostmi projekta. Iz priročnega menija lahko tedaj izberemo "Add files".



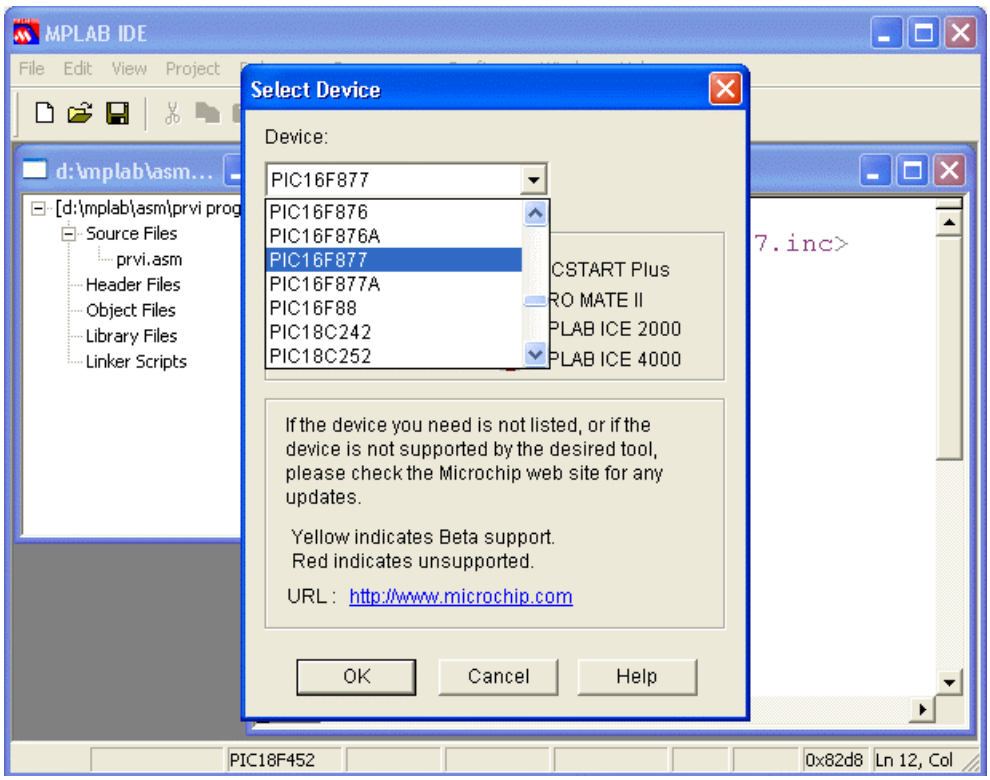
Odpre se nam novo okno, v katerem poiščemo datoteko z programom, ki smo ga pravkar napisali:



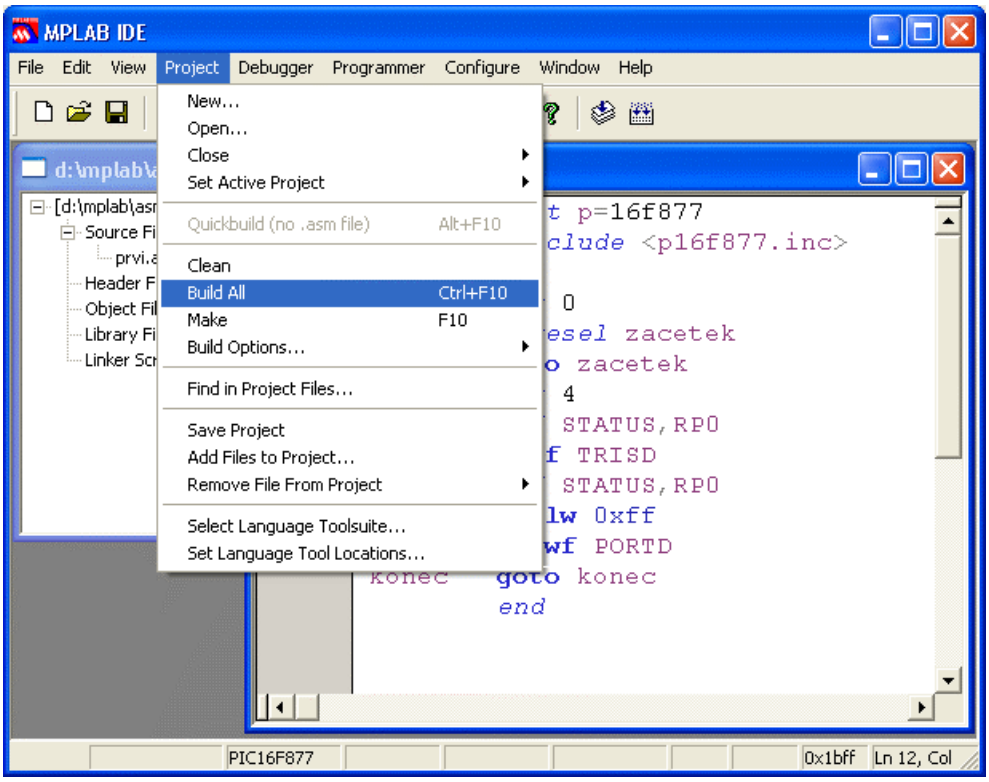
Preden program prevedemo, moramo nastaviti še procesor, v katerega bomo vpisali program:



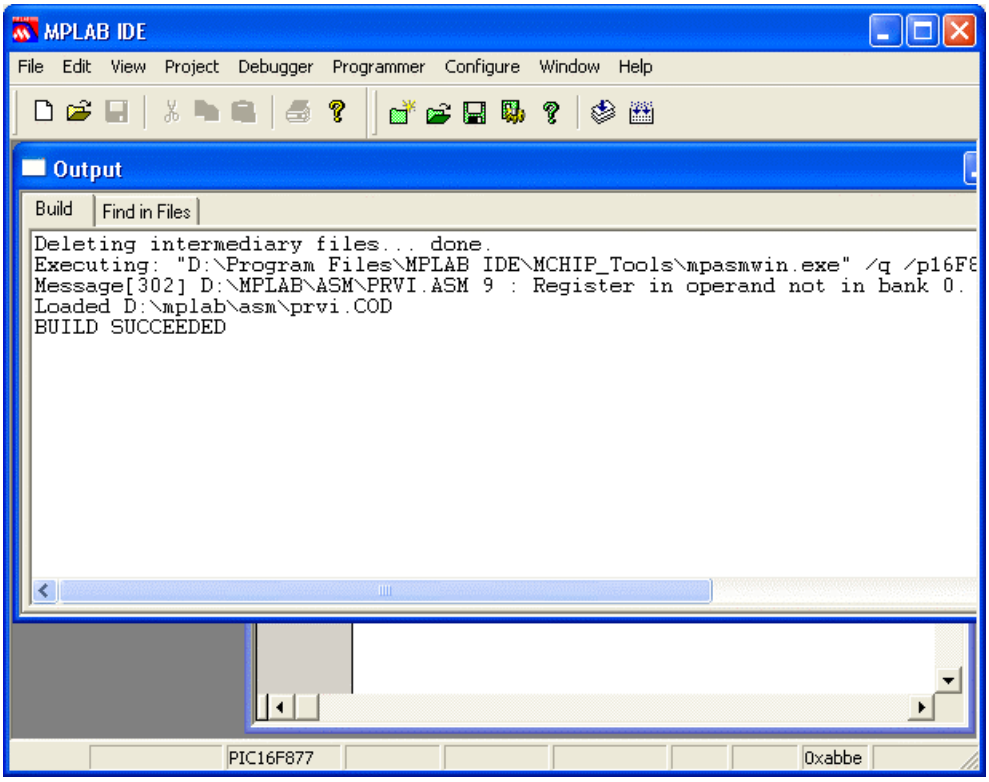
Iz padajočega menija izberemo ustrezni mikrokontroler:



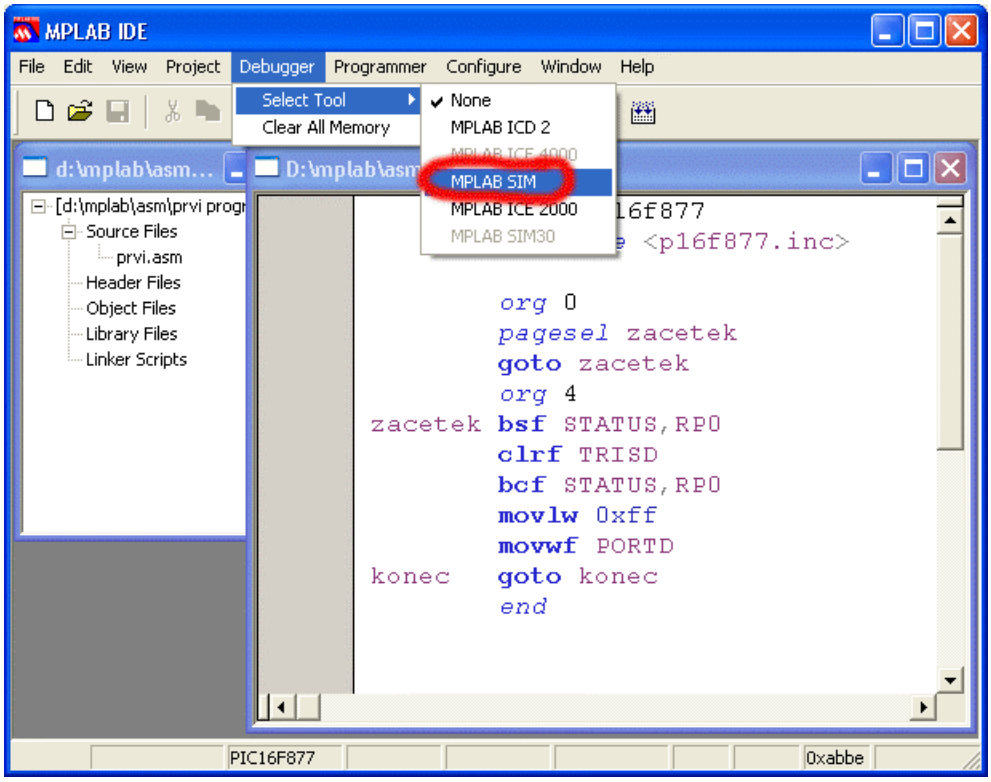
Sedaj lahko sprožimo prevajanje programa:



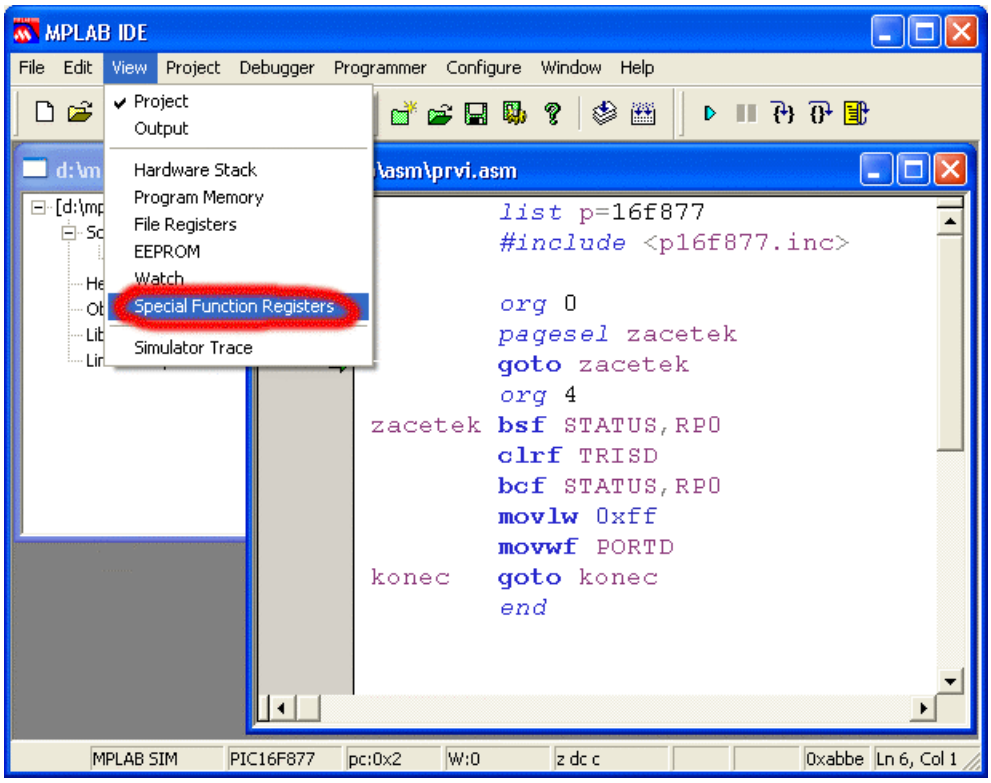
Prevajalnik nam odpre novo okno, v katerem nam prikaže nekaj podatkov o prevajanju in morebitne napake v naši kodi. Če je bilo prevajanje brez napak nam izpiše "BUILD SUCCEEDED".



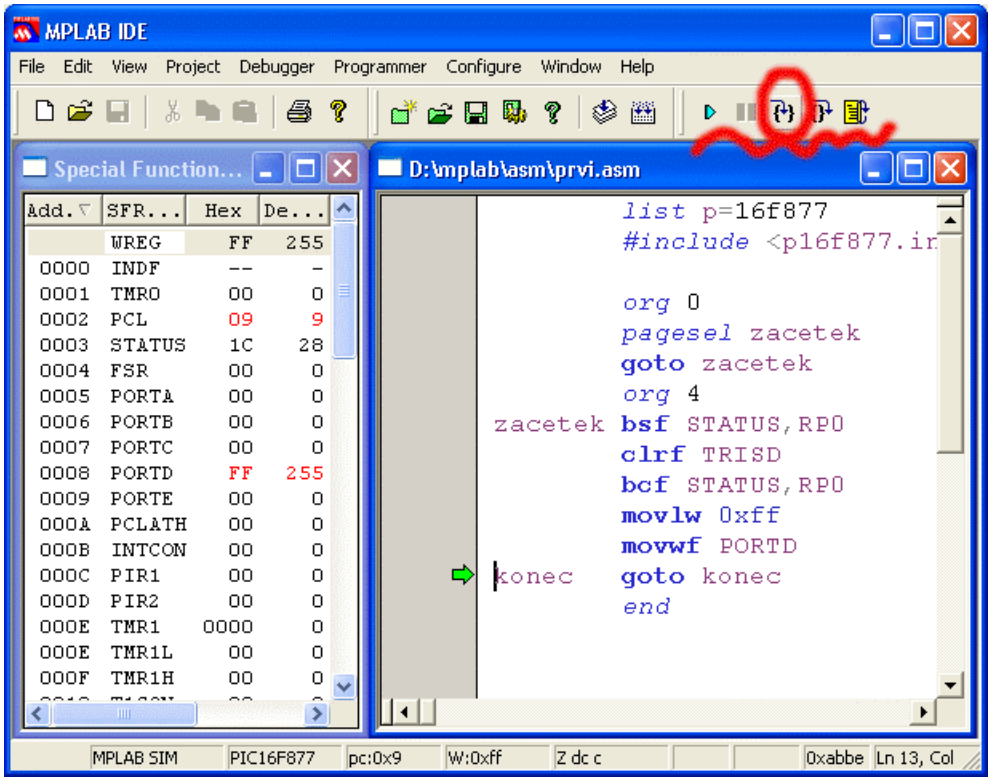
Kako program deluje, si lahko ogledamo v simulatorju:



Odpremo okno z registri, ki bi jih radi opazovali:

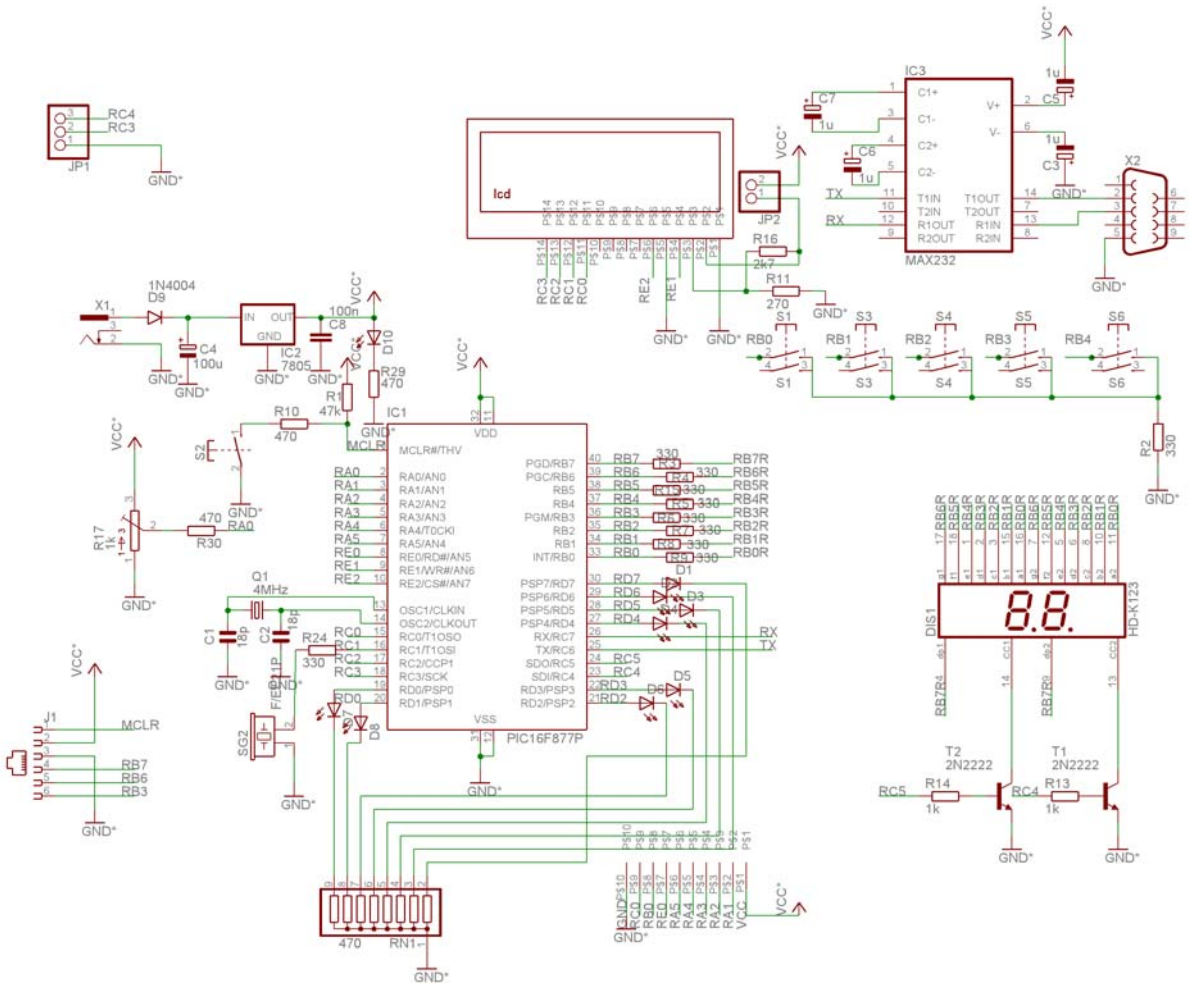


S klikanjem na označen gumb lahko opazujemo izvajanje programa po korakih:



Dodatek B

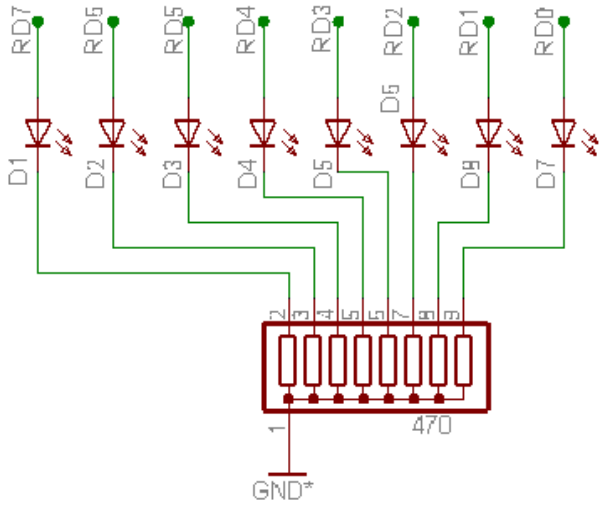
Vezje »PicZaHec«



Programiranje posameznih delov vezja »PicZaHec«

Ker bi bila predstavitev celotne sheme v enem delu nepregledna, bomo celotno vezje predstavili po posameznih sklopih. Pri tistih delih vezja, ki si med sabo delijo priključne nožice mikrokontrolerja, bo to posebej poudarjeno.

Diode LED



8 diod LED je priključenih na vrata D mikrokontrolerja. Z anodami so povezane z nožicami mikrokontrolerja, s katodo pa preko 470Ω upora na negativno napajalno napetost vezja.

Na ploščici tiskanega vezja so postavljene tako, da dioda na skrajni desni strani predstavlja LSB bit PORTD registra, dioda na skrajni levi pa MSB.

Če vrata D mikrokontrolerja uporabimo kot digitalna izhodna vrata (to naredimo z ukazom `clrf TRISD`), lahko s postavitvijo bitov v PORTD registru na logično vrednost »1«, prižgemo posamezne diode.

Primer: napiši zaporedje ukazov, ki bo prižgalo diode priključene na pine RD0, RD1 in RD2. Predpostavi, da je vrednost PORTD registra na začetku enaka 0x00.

Rešitev: Posamezne nožice mikrokontrolerja, ki pripadajo vratom D, lahko krmilimo s pripadajočimi biti v PORTD registru. V našem primeru mora biti v PORTD registru vpisana vrednost b'00000111'. To lahko naredimo na dva načina:

a) spremenimo posamezne vrednosti tistih bitov, ki se razlikujejo od začetnega stanja:

```
bsf PORTD,0
bsf PORTD,1
bsf PORTD,2
```

b) vpišemo celotno želeno vrednost v PORTD register:

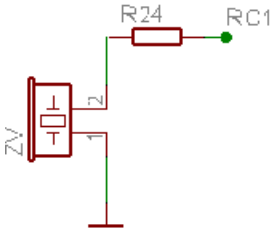
```
movlw b'00000111'
movwf PORTD
```

Zvočnik

V vezju najdemo zvočnik, ki deluje s pomočjo piezoelektričnega efekta. Zvočnik je narejen zelo enostavno: okrogla kovinska ploščica, na katero je nanesenega nekaj kvarca (SiO_2). Za ta kristal je značilno, da ob priključitvi izmenične napetosti niha. Nihanje se prenese na kovinsko ploščico, ki ustvari zvok. Tak zvočnik se obnaša podobno kot kondenzator (SiO_2 je dober električni izolator).

Zvok je torej odvisen od frekvence vzbujalne napetosti. Če generiramo napetost s frekvenco npr. 1kHz in jo priključimo na zvočnik, bomo dobili zvočno valovanje iste frekvence. Za kvarčne kristale je značilno, da geometrija kristala določa resonančno frekvenco – frekvenco ,pri kateri kristal najbolj niha. To se v našem primeru kaže v jakosti zvoka, ki je za različne frekvence različna, kar pa pri generiranju enostavnih zvokov ne igra pomembne vloge.

Pri generiranju zvoka ne tečejo veliki tokovi. Da bi se ognili okvari zvočnika zaradi tokov, ki bi se lahko pojavili ob resonanci je tok še dodatno omejen z upornostjo R_{24}.



Zvočnik je priključen na nožico RC1 torej nožico, ki pripada vratom C.

Pripadajoči bit v registru TRISC moramo zbrisati na logično »0«, da ga lahko krmilimo.

Zvok generiramo tako, da RC1 nožico preklapljamo med logično »1« in »0«, z želeno frekvenco.

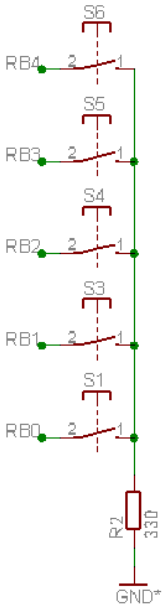
V programu to naredimo tako, da merimo čas polovice periode želene frekvence. Po preteku časa preklopimo logično vrednost na nožici RC1.

Program bi lahko izgledal tako:

```
ponovi      bsf PORTC,1
            call pocakaj_pol_periode
            bcf PORTC,1
            call pocakaj_pol_periode
            goto ponovi
```

`pocakaj_pol_periode` mora biti podprogram, ki »porabi« potreben čas. Tak podprogram lahko napišemo z uporabo katerega od časovnikov ali pa z rekurzivno zanko.

Naj povdarim, da so vse rešitve, ki jih dobimo pri časovnem krmiljenju odvisne od zunanje ure mikrokontrolerja – torej hitrosti izvajanja ukazov. Program, ki ga prenašamo na procesor z drugačno zunanjo uro, moramo popraviti!



Tipke

Na ploščici se nahaja 5 tipk, ki so priključene med spodnjimi petimi nožicami vmesnika B in negativnim polom napajalne napetosti.

Preden začnemo uporabljati tipke, moramo pripadajoče nožice nastaviti kot vhode. To naredimo v TRISD registru:

```
movlw b'00011111'
movwf TRISD
```

Najvišje tri bite vmesnika B lahko nastavimo kot vhode ali izhode. V vezju so skupaj z vsemi ostalimi signali vmesnika B uporabljeni za krmiljenje dveh 7- segmentnih prikazovalnikov.

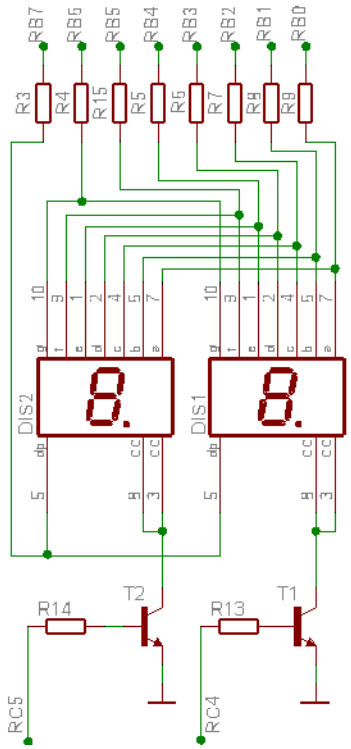
Ker v vezju ni upornikov, ki bi na nožicah ob ne-pritisnjenih tipkah definirali logični nivo, moramo vključiti notranje upore (v mikrokontrolerju), ki so vgrajeni v vmesniku B. Ti upori povežejo nožice proti pozitivni napajalni napetosti. Ko ni pritisnjena nobena tipka, je tako na vhodih vmesnika B visok logični nivo.

Notranje upore vključimo tako, da zbrišemo najvišji bit v opcijskem registru:

```
bcf OPTION_REG,7
```

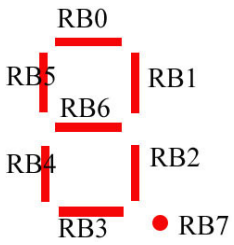
Paziti je treba le na pravilno nastavitev dela RAMa, saj je OPTION_REG tako kot TRISD v drugem delu.

7-segmentni prikazovalnik



V vezju sta dva 7-segmentna prikazovalnika, ki ju vključimo s pomočjo bitov RC5 in RC4 vmesnika C. Visok logični nivo na pripadajočih nožicah vključi transistorja T1 in T2. Ta prepuščata tok, ki teče iz nožic vmesnika B proti masi. Če damo na nožice vmesnika B visok logični nivo – diode LED svetijo.

Diode so povezane z nožicami po sliki:

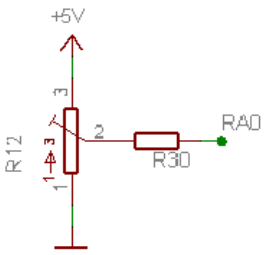


Z naslednjim koščkom kode lahko prižgemo vse diode obeh prikazovalnikov:

```
bsf STATUS,RP0
bcf TRISC,5
bcf TRISC,4
clrf TRISB
bcf STATUS,RP0

bsf PORTC,5
bsf PORTC,4
movlw b'11111111'
movwf PORTB
```

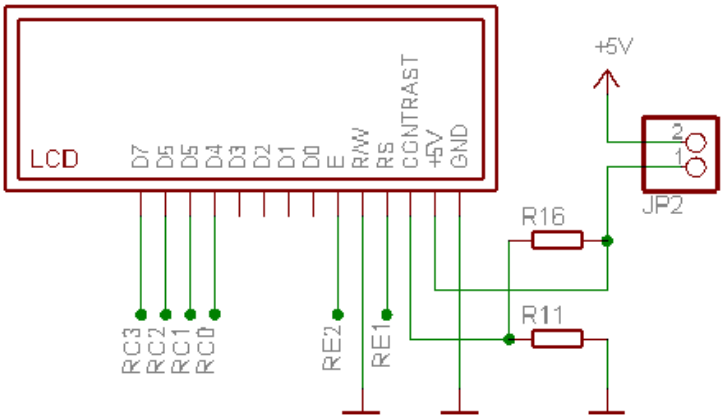
Pretvornik AD



V vezju se nahaja potenciometer, s pomočjo katerega je mogoče na analognem vhodu RA0 mikrokontrolerja nastaviti napetost med 0 in 5 volti.

Preden začnemo uporabljati pretvornik AD moramo RA0 nastaviti kot vhod in v registru ADCON1 povedati, da bomo uporabljali analogni vhod. Vse ostale vrednosti (vklop AD, start pretvorbe, izbira vhoda) imamo dostopne v registru ADCON0.

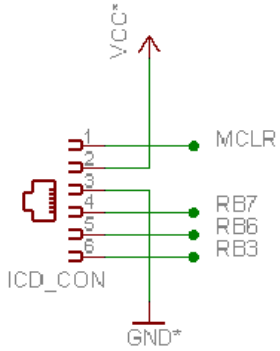
Prikazovalnik LCD



V vezju je priključen prikazovalnik LCD s krmilnikom HD44780, ki ga krmilimo s 6 signali mikrokontrolerja. Podatke in ukaze prenašamo preko spodnjih štirih bitov vmesnika C, vpis in branje pa omogočimo s kontrolnima signaloma E in RS, ki sta povezana na nožici RE2 in RE1.

Da prikazovalnik po nepotrebnem ne troši, toka skrbi mostiček JP2, ki ga moramo ob uporabi LCD kratkostičiti.

ICD »razhroščevalnik«



Na levem robu ploščice pod LCD je tudi priključek za Microchip-ov ICD.

Če imamo ICD vezje, ga lahko tu priključimo namesto serijske povezave s PC. ICD omogoča programiranje mikrokontrolerja, postavljanje prekinitev v program, ki teče v mikrokontrolerju, ogled vsebine registrov…

Primeri programov

Diode

```
            list p=16f877
            #include <p16f877.inc>

            org 0x00
            goto glavni
            ORG 0x04

glavni      nop
            banksel TRISD     ; To je isto kot da bi napisal bsf status,rp0

                              ; in bcf status,rp1
                              ; izbrati moras pac stran v ramu, z zeljenim
                              ;registrom
            clrf  TRISD
            banksel PORTD     ; pa smo spet na strani 0
                              ; iz "banksel portb" prevajalnik naredi:
                              ; bcf status,rp0
                              ; bcf status,rp1
            movlw 0x55
            movwf PORTD
      konec goto  konec
            end
```

Diode – izboljšano

```
            ;miganje diod na vrath d
            list p=16f877
            include <p16f877.inc>

pepi1       equ 0x20
pepi2       equ 0x21

            org 0x00
            goto glavni
            org 0x04

glavni      bsf STATUS,RP0
            clrf  TRISD
            bcf STATUS,RP0
naprej      movlw 0x55
            movwf PORTD
            call  zakasni
            movlw 0xaa
            movwf PORTD
            call  zakasni
            goto  naprej


zakasni     decfsz pepi1,1    ;((1+2)*255+2+(1+2))*255+((1+2)*255+2+2)+2=197121
            goto zakasni
decfsz      pepi2,1
            goto zakasni
            return

            end
```

Generiranje zvoka 1kHz

```
      ;1kHz pisk pri 4MHz zunanjem kristalu

      list p=16f877
      #include <p16f877.inc>

pepi1 equ 0x20
pepi2 equ 0x21

      org 0x000
      goto glavni
      org 0x04

glavni
      bsf STATUS,RP0
      clrf TRISC
      bcf STATUS,RP0
naprej
      bsf PORTC,1
      call zakasni
      bcf PORTC,1
      call zakasni
      goto naprej


zakasni
      movlw d'165'
      movwf pepi1
malo  decfsz pepi1,1    ; 1+1+(1+2)*164+2+2=498
      goto  malo        ;
      return

      end
```

7 segmentni prikazovalnik

```
            list      p=16f877
            #include <p16f877.inc>

pepi1       equ   0x20
pepi2       equ   0x21

            org   0x000
            goto  glavni
            org   0x004

glavni      nop
            banksel TRISC
            clrf  TRISC
            clrf    TRISB
            banksel PORTB

            bsf   PORTC,5
            bsf   PORTC,4
ponovi      movlw 0x01
            movwf PORTB
naprej      call  zakasni
            rlf   PORTB,1
            btfsc PORTB,6
            goto  ponovi
            goto  naprej


zakasni     movlw 0x30
            movwf pepi2
malo        decfsz  pepi1,1
            goto  malo
            decfsz  pepi2,1
            goto  malo
            return

            end
```

Uporaba tipk

```
            ;tipke
            list      p=16f877
            #include <p16f877.inc>

pepi1       equ 0x20
pepi2       equ 0x21

            org 0x00
            goto glavni
            ORG 0x04

glavni      nop
            banksel TRISC
            movlw 0xff
            movwf TRISB
            clrf TRISD
            bcf OPTION_REG,7     ;vkljuciti moras RBPU
            banksel PORTD
            movwf PORTD

naprej      comf PORTB,w
            movwf PORTD
            goto naprej

            end
```

URA – 1

```
      ;utripanje LED na RD0 z 1Hz
      list p=16f877
      #include <p16f877.inc>

stot  equ 0x20
prvi  equ 0x21
drugi equ 0x22

      org 0
      pagesel zacetek
      goto zacetek
      org 4
zacetek
      bsf STATUS,RP0
      bcf STATUS,RP1
      clrf TRISD
      bcf STATUS,RP0

      clrf PORTD
      movlw d'50'
      movwf stot

ponovi
      call stotinka
      decfsz stot,f
      goto ponovi
      movlw d'50'
      movwf stot
      movlw 0x01
      xorwf PORTD,f
      goto ponovi

stotinka
      nop
      movlw d'13'
      movwf drugi
      movlw d'250'
      movwf prvi
se_malo
      decfsz prvi,f
      goto se_malo
      decfsz drugi,f
      goto se_malo
      return

      end
```

URA - 2

```
;TIMER1 v povezavi s CCP modulom - prikaz sekund na dveh 7-segmentnih LED
;prikazovalnikih


            list p=16f877
            #include <p16f877.inc>

stot        equ 0x20
en_sek      equ 0x21
des_sek     equ 0x22
indeks      equ 0x23

            org 0
            pagesel zacetek
            goto zacetek
            org 4
zacetek     bsf STATUS,RP0
            bcf STATUS,RP1
            clrf TRISB
            bcf TRISC,4
            bcf TRISC,5
            bcf STATUS,RP0

            clrf stot
            clrf en_sek       ;na niclo
            clrf des_sek

            movlw 0x27        ;0x2710 = 10000
            movwf CCPR1H
            movlw 0x10
            movwf CCPR1L
            movlw b'00001011'
            movwf CCP1CON
            movlw 0x01
            movwf T1CON
            movlw 0x10
            movwf PORTC

cakaj       btfss PIR1,CCP1IF ;pocakaj 10000 us
            goto cakaj
            bcf PIR1,CCP1IF
            call prikazi

            incf stot,1
            movlw d'100'            ;test stotink
            subwf stot,w
            btfss STATUS,Z
            goto cakaj
            clrf stot

            movlw 0x80        ;vkljuci piko sekund
            xorwf PORTB,1

            incf en_sek,1           ;test enic sekund
            movlw 0x0a
            subwf en_sek,w
            btfss STATUS,Z
            goto cakaj
            clrf en_sek
```

```
            incf des_sek,1          ;test desetic sekund
            movlw 0x06
            subwf des_sek,w
            btfss STATUS,Z
            goto cakaj
            clrf des_sek
            goto cakaj

prikazi     movlw 0x30
            xorwf PORTC,1
            btfss PORTC,4
            goto pok_des
            movf en_sek,w
            call tabela
            movwf PORTB
            return
pok_des     movf des_sek,w
            call tabela
            movwf PORTB
            return

tabela      addwf PCL,f
            retlw b'00111111' ;0
            retlw b'00000110' ;1
            retlw b'01011011' ;2
            retlw b'01001111' ;3
            retlw b'01100110' ;4
            retlw b'01101101' ;5
            retlw b'01111101' ;6
            retlw b'00000111' ;7
            retlw b'01111111' ;8
            retlw b'01101111' ;9

            end
```

Pretvornik AD

```
                ;pretvornik AD s prikazom vrednosti
                ;na diodah LED na vmesniku D

                list      p=16f877
                #include <p16f877.inc>

pepi1           equ     0x20
pepi2           equ     0x21

                org      0x00
                goto     glavni
                ORG      0x004

glavni          bsf STATUS,RP0
                movlw 0x0e
                movwf ADCON1
                movlw 0x01
                movwf TRISA
                clrf TRISD
                bcf STATUS,RP0

                movlw 0x41
                movwf ADCON0

naprej          bsf ADCON0,2
pocakaj         btfsc ADCON0,2
                goto pocakaj
                movf ADRESH,w
                movwf PORTD
                goto naprej

                end
```

Prikazovalnik LCD

```
            ;Prikaz inicializacije in izpisa na LCD
            list      p=16f877
            #include <p16f877.inc>

pepi1       equ   0x20
pepi2       equ   0x21
temp        equ   0x22

            org   0x00
            goto  glavni
            ORG   0x004
glavni      bsf   status,rp0
            clrf  trisc
            movlw 0x07
            movwf adcon1
            clrf  trise
            bcf   status,rp0

            clrf  porte
            call  zakasni2

init        clrf  porte
            movlw 0x03
            movwf portc
            call  puls_E
            call  zakasni
            call  puls_E
            call  zakasni
            call  puls_E
            call  zakasni

            movlw 0x02
            movwf portc
            call  puls_E
            call  zakasni

            movlw 0x28
            call  LCD_ukaz
            movlw 0x08
            call  LCD_ukaz
            movlw 0x0f
            call  LCD_ukaz
            movlw 0x06
            call  LCD_ukaz

            call  LCD_zbrisi

            movlw 'Z'
            call  LCD_znak
            movlw 'i'
            call  LCD_znak
            movlw 'v'
            call  LCD_znak
            movlw 'i'
            call  LCD_znak
            movlw 'j'
            call  LCD_znak
            movlw 'o'
            call  LCD_znak
```

```
            movlw '!'
            call  LCD_znak

konec       goto  konec

LCD_ukaz    bcf   porte,1
            movwf temp
            swapf temp,w
            movwf portc
            call  puls_E
            movf  temp,w
            movwf portc
            call  puls_E
            call  zakasni
            return

LCD_znak    bsf   porte,1
            movwf temp
            swapf temp,w
            movwf portc
            call  puls_E
            movf  temp,w
            movwf portc
            call  puls_E
            call  zakasni
            return

LCD_zbrisi  bcf   porte,1
            movlw 0x01
            call  LCD_ukaz
            call  zakasni
            return

puls_E      bsf   porte,2
            nop
            bcf   porte,2
            return

zakasni     decfsz pepi1,1 ; zakasnitev 1.5ms
            goto  zakasni
se_malo     decfsz pepi1,1
            goto  se_malo
            return

zakasni2    movlw d'250'      ; zakasnitev 20ms
            movwf pepi1
            movlw d'25'
            movwf pepi2
ponovno     decfsz pepi1,1
            goto  ponovno
            decfsz pepi2,1
            goto  ponovno
            return

            end
```