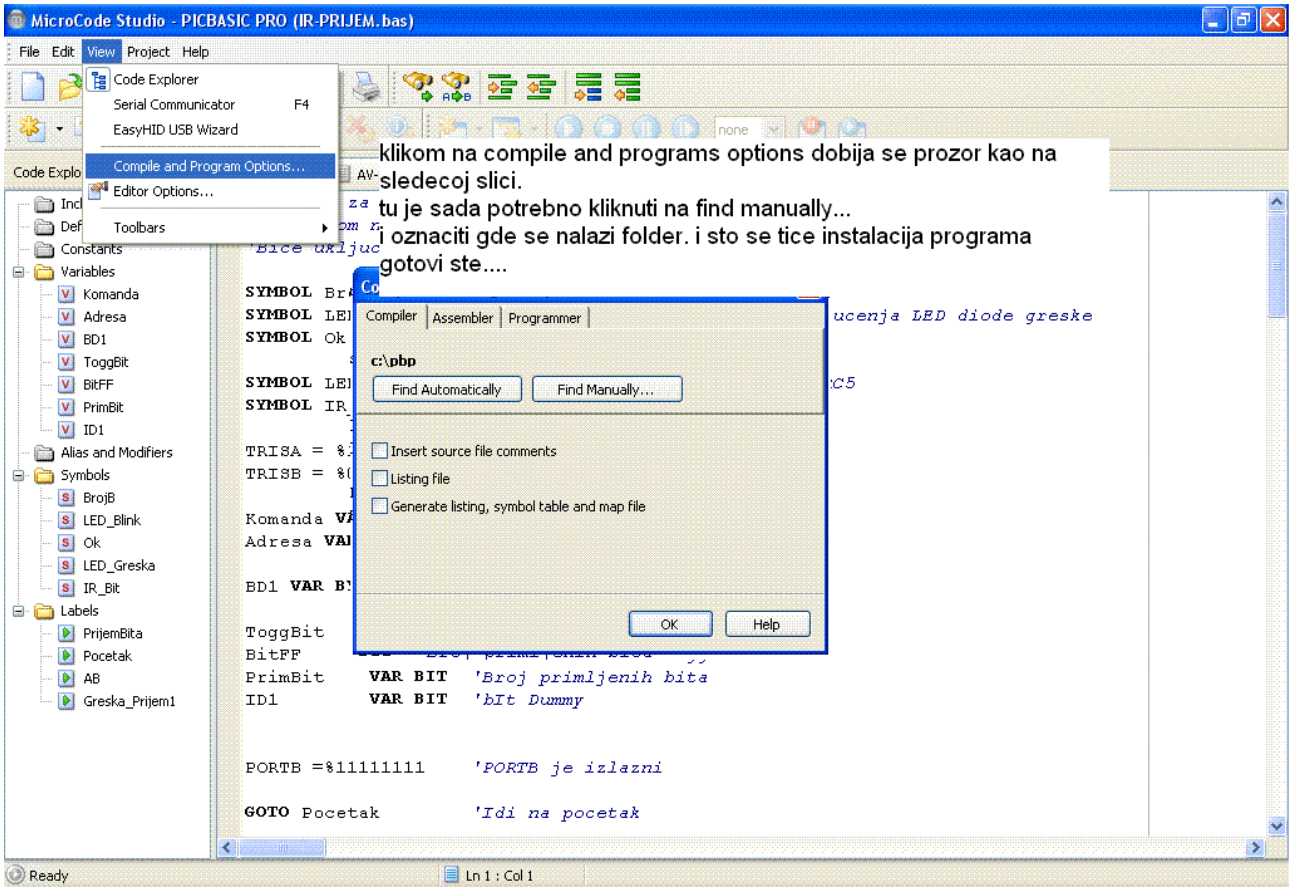
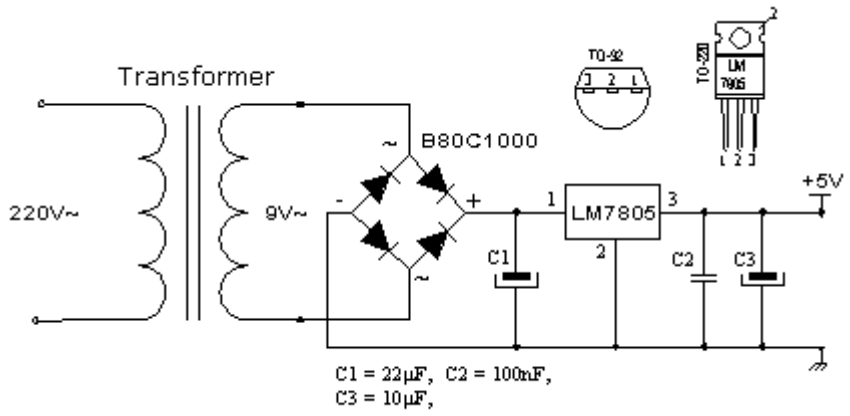
hardverski se pomazite kako znate i umete. za pocetak vam treba pic16F84A, kristal od 4MHZ-a(ili rezonator), stabilizator 7805 ili stabilizirani izvor napajanja na 5V, par ledica, i isto toliko otpornika od 220oma(ili sl 100-1000oma) nekoliko otpornika od 10k, nekoliko tastera, i ne bi lose dosao jedan lcd displej 16x2. i eventualno neka memorija 24c01 ili nesto slicno
e sad kad krenemo trudite se da ne postavljate pitanja ako bas ne morate.
e ovako za pocetak procitajte samo sve naredbe da ih imate u vidu, ne ocekujem da sve naucite zato cemo ici polako...
http://milan.milanovic.org/skola/mikkon/picbas-00.htm
i nemojte kad vidite ovo da kazete ja to znam i da okrenete glavu...
sutra cu vam sastaviti nesto o picu pinovima ulazi izlazi i napajanje itd...

http://rapidshare.com/files/142404491/sve_sto_treba_zapisanje_koda_u_picbasicu.rar.html

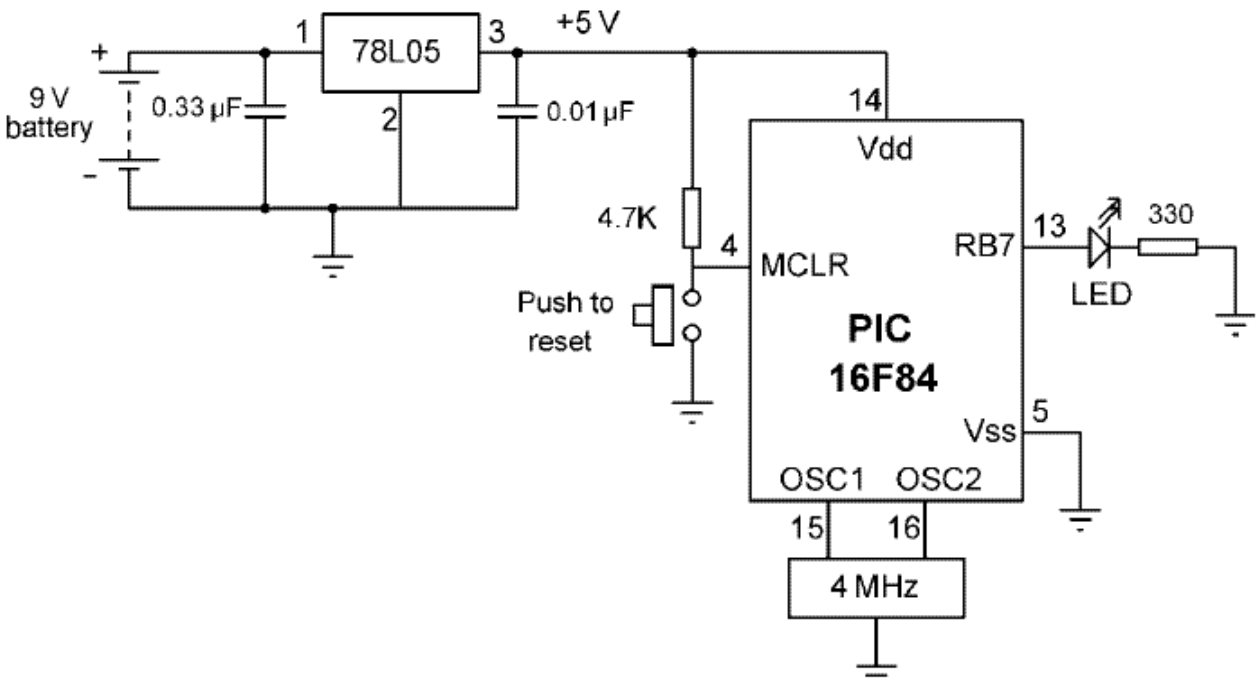
e ovako. u raru postoje 2 instalacije setup(instalira microcode studio) i pbp 2.44(to je kompajler). ne bitno kojim redom cete instalirati. samo nakon sto oboje instalirate preostaje vam povezati mcs sa kompajlerom.
uputstvo je na slici. znaci samo pokrenite mcs(micro code studio) i pratite uputstva sa slike



evo jednostavnog napajanja

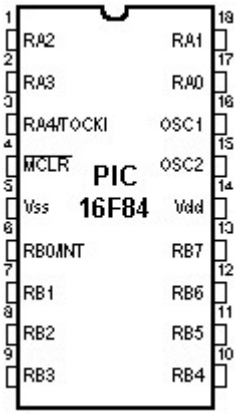


a evo jos jednostavnijeg



PIC16F84 ima 18 pinova od kojih čak 13 mogu da se koriste kao I/O linije (I=input - ulaz, O=output - izlaz), ima FLASH programsku memoriju od 1024 reči (svaka je dužine 14 bitova), RAM memoriju od 68 bajtova i 64 bajta internog EEPROM-a. Programska FLASH memorija može da se reprogramira oko 1000 puta i garantovano "cuva" program bar 40 godina. EEPROM može da se reprogramira cak 1 000 000 puta i takodje cuva podatke bar 40 godina. RAM memorije ima više od 68 bajtova, ali je jedan deo rezervisan za registre sa specijalnim funkcijama. Takodje postoji i STACK od 8 nivoa; STACK je, ukratko, neka vrsta skladišta za memorijsku adresu na koju program treba da se vrati nakon poziva potprograma, a takvih ugnježdenih poziva može da bude maksimalno 8, što je i više nego dovoljno za mikrokontroler sa ovakvim performansama. Napon napajanja se kreće od 2 do 6 V, a potrošnja je manja od 2 mA pri napajanju od 5V i taktu od 4MHz. Potrošnja pri 2 V i taktu od 32kHz je oko 15 uA (mikroampera), a kada je u SLEEP modu pri 2V pada ispod 1uA. Što se tice takta, on može da ide od 0 do 20 MHz; prvo su postojale dve verzije IC-a, do 4 i do 10MHz, ali se nedavno pojavila i verzija do 20 MHz. Arhitektura je RISC tipa (reduced instruction set) što znači da ima malo instrukcija (svega 35), ali se izvršavaju maksimalno brzo.

RASPORED I FUNKCIJA PINOVA



pin 1 = RA2, I/O linija porta A, TTL tipa.

pin 2 = RA3, I/O linija porta A, TTL tipa

pin 3 = RA4/TOCKI, I/O linija porta A, takodje služi kao ulaz do TMR0 brojača. Kada je I, onda je ST tipa (schmitt trigger), a kada je O, onda je open- drain.

pin 4 = inv. MCLR (invertovan MCLR, obeležava se i sa -MCLR), reset pin (reset kada je na niskom nivou) ili se dovodi napon programiranja.

pin 5 = masa napajanja.

pin 6 = RB0/INT, I/O linija porta B, ili izvor spoljnog interapta, TTL/ST (ST kada je ulaz za interapt).

pin 7 = RB1, I/O linija porta B, TTL tipa.

pin 8 = RB2, I/O linija porta B, TTL tipa.

pin 9 = RB3, I/O linija porta B, TTL tipa.

pin 10 = RB4, I/O linija porta B, takodje izvor interapta pri promeni nivoa, TTL tipa.

pin 11 = RB5, I/O linija porta B, izvor interapta pri promeni nivoa, TTL tipa.

pin 12 = RB6, I/O linija porta B, izvor interapta pri promeni nivoa, takt pri programiranju, TTL/ST (ST pri programiranju).

pin 13 = RB7, I/O linija porta B, izvor interapta pri promeni nivoa, podaci pri programiranju, TTL/ST (ST pri programiranju).

pin 14 = pozitivan pol napajanja.

pin 15 = OSC2/CLKOUT, spaja se na jedan kraj kristala, a ako se koristi RC ocilator, na ovom pinu se pojavljuje 1/4 frekvence oscilatora.

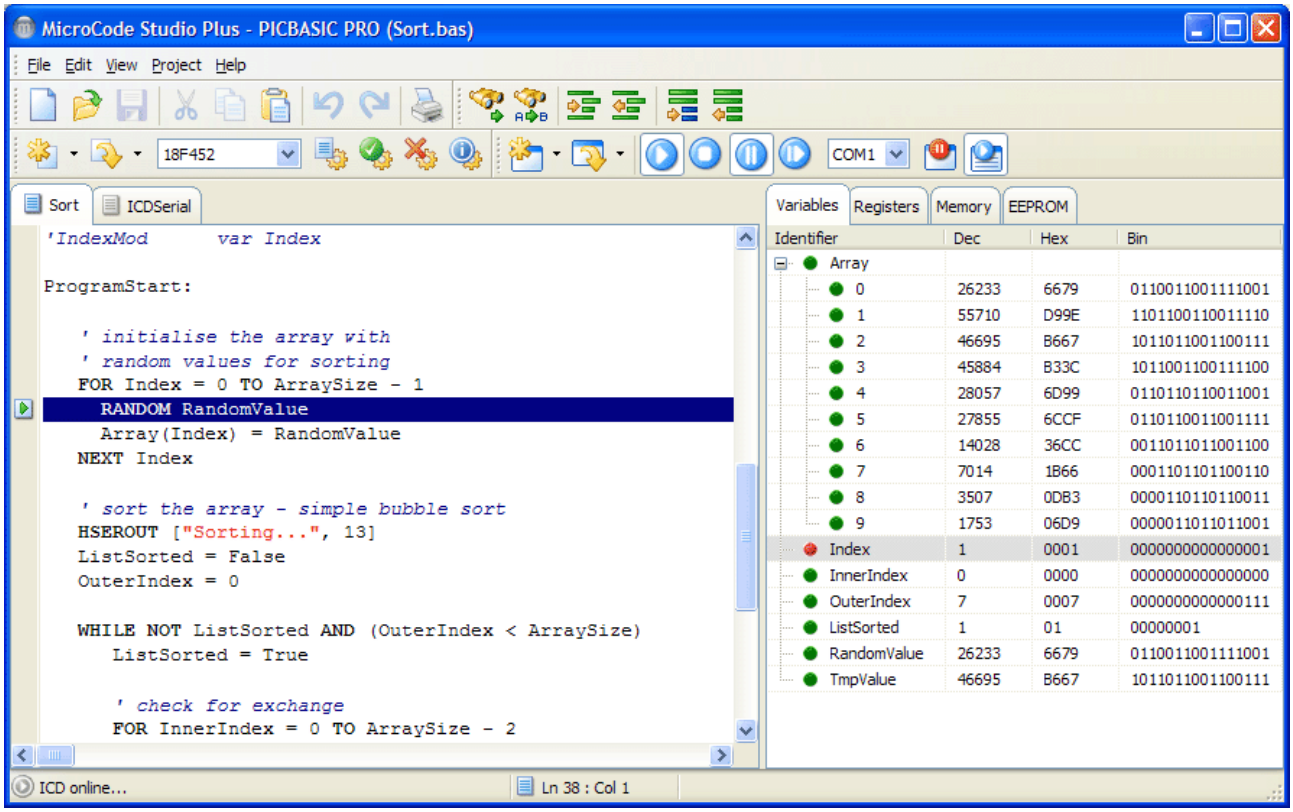
pin 16 = OSC1/CLKIN, spaja se jedan kraj kristala, ili RC, ili se dovodi spoljni takt, ST/CMOS tipa (ST kod RC oscilatora).

pin 17 = RA0, I/O linija porta A, TTL tipa.

pin 18 = RA1, I/O linija porta A, TTL tipa

Opis registara sa specijalnim funkcijama
Na adresi 00h nalazi se registar sa imenom INDF; to nije fizički registar već se koristi za indirektno adresiranje u sprezi sa
registrom SFR za indirektno adresiranje.
01h - TMR0, 8-bitni brojač/sat realnog vremena
02h - PCL, niski bajt PC-a (program counter-a)
03h - STATUS, sadrži bitove koji označavaju stanje aritmetičko-logičke jedinice (ALU), RESET status, i bitove za
adresiranje viših lokacija RAM-a.
04h - FSR, pointer za indirektno adresiranje RAM memorije
05h - PORTA, to su, u stvari, pinovi porta A
06h - PORTB, pinovi porta B
07h - ne koristi se
08h - EEDATA, registar podataka za EEPROM
09h - EEADR, registar adrese za EEPROM
0Ah - PCLATH, visoki bajt PC-a
0Bh - INTCON, registar za kontrolu interapta
80h - INDF, isto kao 00h
81h - OPTION, upravljački registar za preskaler, spoljni interapt, TMR0 i pull- up otpornike na portu B
82h - PCL, kao 02h
83h - STATUS, kao 03h
84h - FSR, kao 04h
85h - TRISA, registar za definisanje smera (ulaz ili izlaz) pinova na portu A
86h - TRISB, registar za definisanje smera pinova na portu B
87h - ne korisiti se
88h - EECON1, upravljački registar za rad sa EEPROM-om
89h - EECON2, drugi upravljački registar (nije fizički registar)
8Ah - PCLATH, kao 0Ah
8Bh - INTCON, kao 0Bh
detaljan opis svih registara mozete naci na linku:
http://www.geocities.com/sinelyu/pic/16F84-02_specijalni_registri1.html

ovde cemo se samo pozabaviti sledecim registrima TRIS,PORT, i OPTION registrom(OPTION_REG je ime u picbasicu)
pa sad da krenemo redom.
REGISTRI TRISA I TRISB. pomocu ovih registara odredjujemo dali je neki pin ulazni ili izlazni. oba registra su osmobitna.



posto porta ima samo 5 pinova na ovom kontroleru a registar je osmobitno, onda se koristi nizih 5 bita, a poslednja 3 biota se zanemaruju i ne bitno je dali su postavljeni na 0 ili 1.
koriscenje ovih registara bice kasnije objasnjeno na primeru.

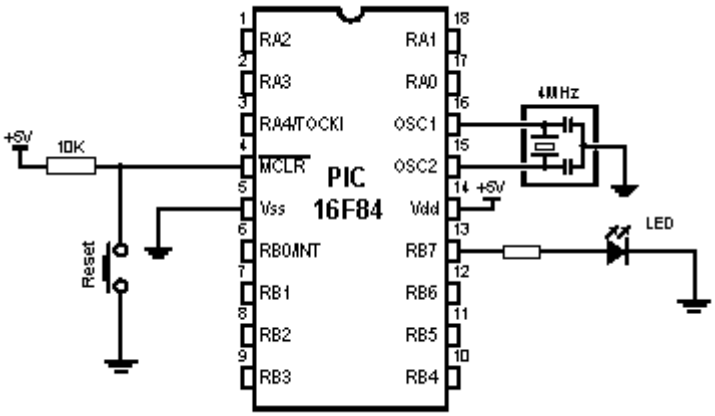
REGISTRI PORTA I PORTB ovi registri sluze citanje/postavljanje stanja na pinovima.
ako su tris registri postavljeni kao izlazni onda se stanje sa port registra "preslikava" na pinove. sto znaci ako je neki bit u registru na 0 onda je i na tom pinu 0, a ako je u registru neki bit na 1 onda je na tom pinu logicka jednica tj 5V.
a ako je tris postavljen kao ulazni oda ce se u port registae upisati stanje sa pina. ako je na pinu 0V onda se u registar upisuje 0, a ako je na pinu 5V onda se u registar upisuje 1. postoji mogucnost pristupa pojedinacnom bitu u port registru. a to se vrsi na sledeci nacin: biti su obelezeni od 0 do 7. i ako hocemo da pristupimo npr 5 bitu registra portb i da ga postavimo na 0 napisacemmo portb.5=0 ili ako hocemo da ga iscitamo u neki promenljivu A napisacemo A=portb.5

OPTION REGISTAR(OPTION_REG) nas konkretno zanima sedmi bit ovog registra. sedmi bit upravlja pulluup-om na portub.
stoga ako nam je potreban pull up na nekom ulaznom pinu umesto da ubacujemo otpornik dovoljno je samo postaviti sedmi bit option_reg na nulu. i imacemu pull up na svim ulaznim pinovima na portub. to je moguce uraditi na dva nacina
prvi OPTION_REG.7=0, ovako pristupamo samo sedmom bitu i postavljamo ga na nulu.
drugi nacin LOW OPTION_REG.7 ovde koristimo naredbu LOW koja postavlja vrednost na nulu.

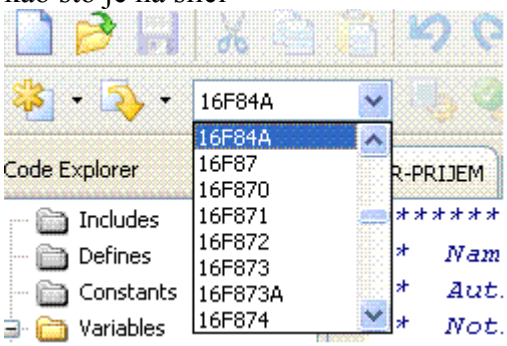
o vrstama oscilatora ako hocete procitajte na linku:
http://www.geocities.com/sinelyu/pic/16F84-04_vrste_oscilatora.html

mi cemo ovde koristiti samo xt oscilator
mislim da je ovo za pocetak sasvim dovoljno.
pitajte ako vam nesto nije jasno, ali nemojte glupostim da pitate. mislim da je text dosta razumljiv.
i ako ste ovo savladali mogli bi polako krenuti na pisanje programa u pbp.

ako imas flopi rasturi ga i unutara imas rezonator. krajnje dve idu na pic, srenjs na masu. u principu moze bilo koja vrsta oscilatora samo onda tako moras podesiti i u programatoru.
posto vidim da su svi glasovi za dalje i nema potrebe za zadrzavanjem, hajde da krenemo.
prvi program neka bude prgram za obicnu ledicu koja treperi.
sema:



milsim da je sema prosta tako da nema potrebe posebno je komentarisati
prvo sto treba da uradite e da izaberete 16F84A
kao sto je na slici



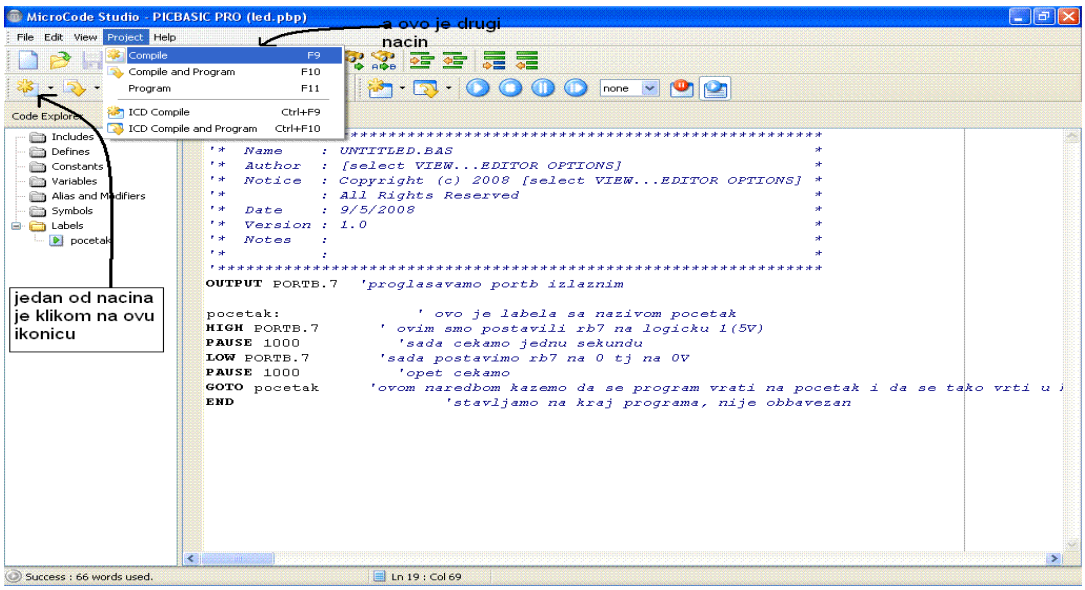
kada ste izabrali pic vreme je da se krene sa pisanjem
prvo sto treba uraditi je postaviti portb.7 kao izlaz.
to se moze raditi na vise nacina, jedan od nacina je pomocu tris registra.
evo kako bi to izgledalo
TRISB=%01111111(u pbp kada se stavi znak % to znaci da sledi broj ubinarnom zapisu)
ovako su svi pinovi postavljeni kao ulazni sem najviseg bita. kao sto se vidi iz primera rb7 je prvi bit, pa za njim dolazi rb6 itd sve do rb0 koji je najnizi tj krajnji desni bit.
drugi nacin za postavljanje pina kao izlaznog je koriscenje naredbe OUTPUT
ova naredba se koristi tako sto se posle nje napise pin koji zelimo proglasiti izlazom, ili cak ceo port.
npr ako napisemo OUTPUT PORTB tada cemo ceo portb proglasiti izlaznim
a ako napisemo OUTPUT PORTB.7 tada cemo samo rb7 proglasiti izlazom a ostali pinovi ce ostati nepromenjeni.
naredba za postavljanje pina kao ulazni je INPUT i sa njom cemo se pozabaviti kasnije...

sledeca naredba koja ce nam trebati je PAUSE
mislim da se ovde nema sta posebno reci sem da je maksimalna vrednost pauze 65,535 ms
pauza se koristi tako sto se napise PAUSE 50, to znaci da ce pic apraviti pauzu od 50ms. slicna naredba je PAUSEUS. na isti nacin se koristi, samo je za odabir duzine potrebno pogledati tabelu koja se nalazi u help fajlu.
naredbe HIGH i LOW su nam potrebne da bi menjali stanje na odredjenom pinu ili portu.
koriste se tako sto se napise HIGH pa pin ili port
u nasem slucaju HIGH PORTB.7
na isti nacin se koristi i naredba LOW
drugi nacin za menjanje stanja na portu je da napisemo PORTB.7=1 ili PORTB.7=0 ovo je ekvivalentno naredbama HIGH i LOW
e sada nam ostaje sve ovo sloziti u program
i sada nam je ostala jos jedna naredba GOSUB
ona se koristi da bi program skocio na neku labelu.
najlakse cete razumeti na primeru
i ostaje nam jos SYMBOL
ova naredba nije potrebna ali je zgodna.
sa njom mozemo dati ime nekom pinu.
npr ako napisemo SYMBOL LED=PORTB.7
u daljem programu necemo morati pisati portb.7 nego je dovoljno napisati led.
napomene: u toku pisanja programa kada zelimo da komentarisemo neki red stavi se znak ' i posle tog znaka u redu je komentar i on ne ulazi u program.
postoji jedna dosta zgodna stvar koja dosta olaksava posao.
a to je kada u toku pisanja programa ne mozemo necega da se setimo dovoljno je postaviti kursor na naredbu i stisnuti F1, tada se otvara help i odmah prikazuje pomoc oko te naredbe.
npr ako napisemo LOW i stavimo kursor na njega dobicemo opis te naredbe
evo kako bi izgledao program:

```
OUTPUT PORTB.7   'proglasavamo portb izlaznim

pocetak:         ' ovo je labela sa nazivom pocetak
HIGH PORTB.7     ' ovim smo postavili rb7 na logicku 1(5V)
PAUSE 1000       'sada cekamo jednu sekundu
LOW PORTB.7      'sada postavimo rb7 na 0 tj na 0V
PAUSE 1000       'opet cekamo
GOTO POCETAK     'ovom naredbom kazemo da se program vrati na
pocetak i da se tako vrti u krog do beskonacnosti
END              'stavljamo na kraj programa, nije obbavezan
```

kada smo ovo napisali potrebno je sacuvati program pa kompajlirati. u koliko nista sacuvali program a stisnuli dugme za kompajliranje mcs ce sam ponuditi da sacuvate program.
kompajliranje se moze izvesti na 3 nacina,pritiskom na taster 'F9' ili kao sto je pokazano na slici:



a sad evo isti program samo kada se koristi SYMBOL

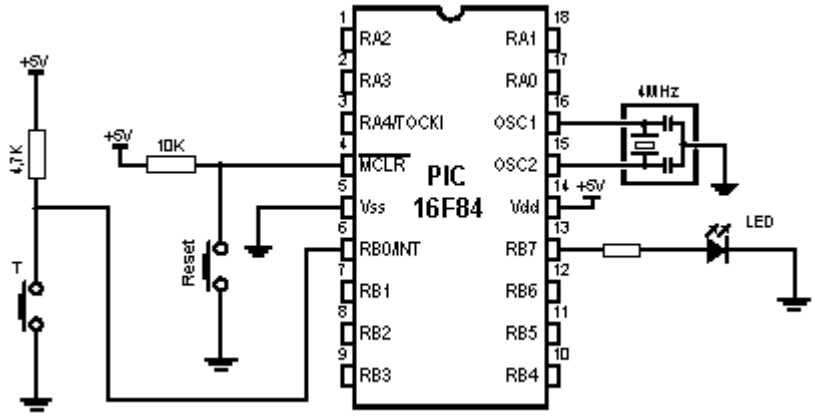
```
SYMBOL LED= PORTB.7
OUTPUT LED
pocetak:
HIGH LED
PAUSE 1000
LOW LED
PAUSE 1000
GOTO POCETAK
END
```

sada jos samo ostaje ubaciti hex faj u pic. hex fajl se nalazi u istom folderu gde ste sacuvali i program.
i sada kada to sklopite i upisete program u pic imate ledicu koja treperi
sad bih voleo da se vi potrudite pa napravite da se dve ledice naizmenicno pale i gas, kao svetla na rampama znaci vi crtate semu i vi pisete program.
pa ko prvi uradi to, i isproba neka uslika, po mogucnosti i snimi pa da vidimo svi kako mu to radi. e sad majstori lemilice u ruke i na posao
naravno ako ima pitanja pitajte.
e sad vidoh da nisam prokomentarisa tipku reset. o njoj ne treba nista posebno reci. kada se tipka stisne pic se resetuje i program krece ispocetka. tj kada je na pinu mclr logicka nula tada je reset aktivan. kada je logicka jedinica tada pic normalno radi

evo sad sam se setio jos jedne komande koju bi bilo zgodno pomenuti.
a to je TOGGLE. ova komanda menja stanje na pinu, ako je na pinu bila 1 on taj pin postavlja na 0, a ako je bila 0 onda se menja u jedan.
evo jos jedan primer

```
SYMBOL  LED=PORTB.7
OUTPUT LED
pocetak:
TOGGLE LED 'prvi put kada program naidje na naredbu toggle on ce
pin postaviti na 1, sledecim nailaskom ce ga vratiti na 0 i tako u
krug. i dobili ste ledicu koja blinka.
PAUSE 1000
GOTO POCETAK
END
```

evo sada bi mogli pokazati kako se koristi pin kao ulazni...



evo seme. potrebno je samo na prethodnu semu dodati jedan taster i otpornik. otpornik nije kritican mose od 2k2, pa sve do 10K...
ako nemate taster pri ruci dovoljno je i komad zice zalemiti na pin.
zadatak:
kada je taster stisnut ledica treba da treperi, kada se taster pusti ledica treba da bude ugasena.
kao prvo treba na prethodni program dodati jedan ulaz i dati ime ulaznom pinu
za ovo ce nam biti potrebna naredba INPUT koja je vec objasnjena.
jos jedna od naredbi koja ce nam trebati je IF... THEN...
ova nareba sluzi za ispitivanje tacnosti izraza i ako je izraz tacan izvrsava se naredba koja se nalazi iza then.
prvi i najjednostavniji oblik je kada se sve nalazi u istom redu i za to postoji sledeca sintaksa.
IF izraz THEN naredba.
izraz moze da pretstavlja bilo koju jednakost ili nejednakost.
npr u nasem slucaju
IF TASTER=0 THEN ... sada smo ispitai da li je ulaz jednak nuli. a ulaz ce biti jednak nuli kada je taster stisnut.
naredba moze biti bilo koja iz PBP-a
postoji jos i slucaj kada postoji vise izraza koji se mogu povezati sa OR ili AND.
npr: IF TASTER1=0 AND TASTER2=0 THEN... u ovom slucaju oba uslova moraju biti jednaka da bi se izvrsila naredba posle THEN
drugi slucaj je kada se koristi OR
npr: IF TASTER1=0 OR TASTER2=0 THEN... u ovom slucaju ce se naredba izvrsiti kada je ispunjen bilo koji od ta dva uslova...
sledeci slucaj je slucaj kada posle THEN treba da se izvrsi skup naredbi a ne samo jedna.
sintaksa za to je sledeca
IF TASTER=0 THEN
naredba1
naredba2
.
.
.
naredba n
ENDIF
ukoliko je izraz tacan tada ce se preci na izvrsavanje naredbi redom kako su pisane.
i u ovom slucaju se mogu koristiti i OR ili AND.
i jos jedan oblik naredba IF je kada se koristi ELSE.
sintaksa je slicna kao iz proslog primera samo sto sada postoji deo posle ELSE koji se izvrsava ukoliko izraz nije tacan

IF TASTER=0 THEN
naredba1
naredba2
.
.
naredba n
ELSE
naredba1
naredba2
.
.
naredba n
ENDIF
toliko IF...
sada da pokazemo na primerima

```
SYMBOL  TASTER= PORTB.0
SYMBOL  LED=PORTB.7
OUTPUT LED
INPUT TASTER             ' sada smo taster proglasili ulazom
pocetak:
IF TASTER=0 THEN TOGGLE LED  ' sada se ispituje dali je taster
stisnut, ukoliko je stisnut promenice se stanje na led, posle
svakog ispitivanja pravi se pauza od 1s
PAUSE 1000
GOTO pocetak    ' program se vraca na pocetak
END
```

sada primer za drugi slucaj koriscenja naredbe

```
SYMBOL TASTER= PORTB.0
SYMBOL  LED=PORTB.7
OUTPUT LED
INPUT TASTER
pocetak:
IF TASTER=0 THEN       'u ovom slucaju ukoliko je taster stisnut
izvrsavace se blok narebi
HIGH LED
PAUSE 1000
LOW LED
PAUSE 1000
ENDIF
GOTO POCETAK     'posle izvrsenog bloka naredbi program se vraca
na pocetak i ponovo se ispituje stanje TASTER-a
END
```

sledeci tip je if...then....else

```
SYMBOL TASTER= PORTB.0
SYMBOL  LED=PORTB.7
OUTPUT LED
INPUT TASTER
pocetak:
IF TASTER=0 THEN
HIGH LED       'ako je uslov ispunjen izvrsava se ovaj blok
naredbi(moze da stoji i jedna naredba)
PAUSE 1000
ELSE
LOW LED       'ukoliko uslov nije ispunjen izvrsava se ovajh blok
naredbi (ili naredba)
PAUSE 1000
ENDIF
GOTO POCETAK     'posle izvrsenog bloka naredbi program se vraca
na pocetak i ponovo se ispituje stanje TASTER-a
END
```

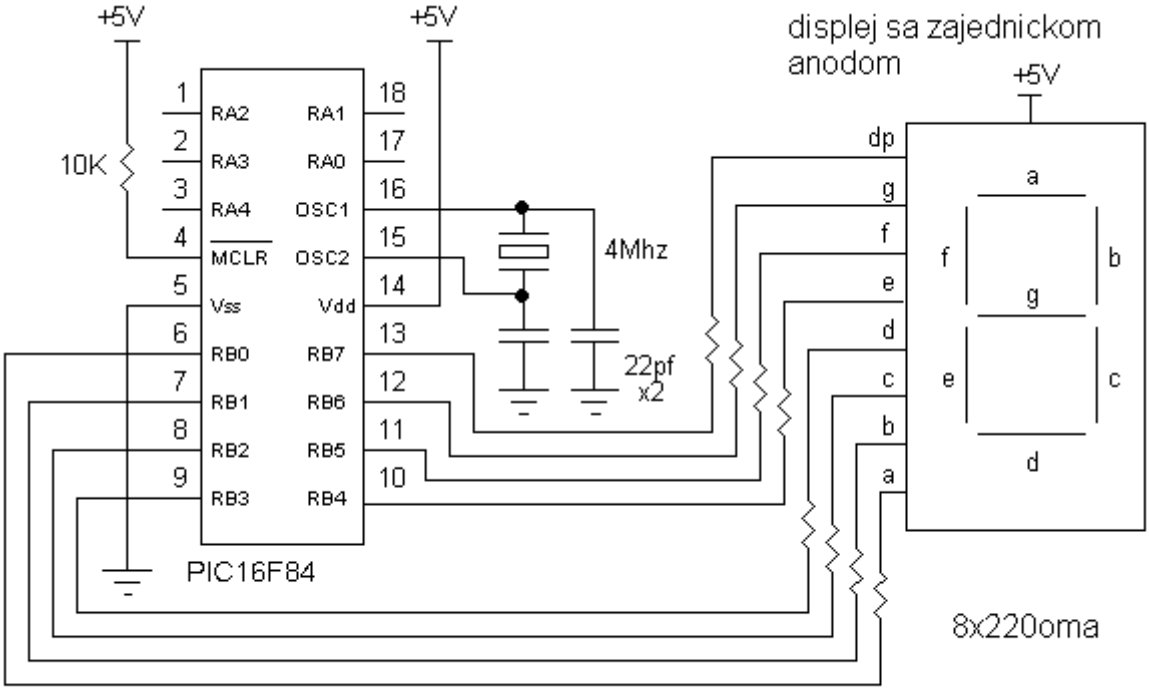
e sada da se vratimo na sedmi bit OPTION REGISTRA
da bi se izbegla upotreba pullup otpornika na portub dovoljno je samo postaviti sedmi bit OPTION_REG na 0.
sada je potrebno samo ubaciti na sam vrh programa LOW OPTION_REG.7 i tada vam vise nije potreban pullu otpornik... NAPMENA ovo vazi samo za portb
evo i jedan primer sa upotrebom OPTION REGISTRA

```
LOW OPTION_REG.7
SYMBOL TASTER= PORTB.0  'dali smo ime pinu portb.0
SYMBOL  LED=PORTB.7
OUTPUT LED
INPUT TASTER             ' sada smo taster proglasili ulazom
pocetak:
IF TASTER=0 THEN TOGGLE LED  ' sada se ispituje dali je taster
stisnut, ukoliko je stisnut promenice se stanje na led, posle
svakog ispitivanja pravi se pauza od 1s
PAUSE 1000
GOTO pocetak    ' program se vraca na pocetak
END
```

e sada bi mogli uvesti promenljive i uciniti stvari mal okomplikovanijim i zanimljivijim...
promenljive su mesto gde se podaci privremeno cuvaju. podacima u njuma se moze pristupiti bilo kada u toku izvrsavanja programa. i one su uobicajno rezultat necega, neke funkcije, ili cak stanja na portu ili samo na pojedinom bitu ili pinu...
kada kazem privremeno mislim na to da se one gube po nestanku napajanja.
stoga ako neke promenljive treba da imaju neku pocetnu vrednost onda se na pocetku programa uvek posle definisanja promenljivih dodeljuje i njihova vrednost.
preporucljivo je staviti i da je promenljiva jednaka nuli ako ona ne treba da sadrzi nista, jer nekada se desi da pic zapamti nesto na toj lokaciji.
sintaksa za definisanje promenljivih"
ime promenljive VAR velicina
ime promenljive je obavezno jedna rec. i nije dozvoljeno koristiti znakove kao sto su tacka ili zarez. uostalom ako napisete pogresno ime kompajler ce javiti gresku i zacrvenece se red u kojem je greska...
postoje 3 velicine promenljivih a to su BIT, BYTE i WORD
promenljiva BIT je velicine bita sto znaci da se u njoj moze zapamtiti samo 0 ili 1.
promenljiva BYTE je velicine 8biti. tju nju se moze sacuvati broj od 0 do 255
promenljiva WORD je velivine 2 bajta tj 16bita.
i u nju se moze sacuvati broj od 0 do 65535
evo par primara:
Pin VAR BIT ' ovim smo definisali promenljivu pin i ona moze biti samo 1 ili 0
Broj VAR BYTE ' ovim smo definisali promenljivu broj i u nju mozemo smestiti broj do 255
VelikiBroaj VAR WORD ' promenljiva je veliki broj i u nju se moze smestiti broj do 65535
sada da se to primeni na prakticnom primetu.
Zadatak:
treba napraviti program koji ce po pritisku tastera upaliti led zadrzati je upaljenju pola sekunde, zatim je ugasiti i ako je taster i dalje stisnut ledica ne treba ponovo da se pali. znaci ledica treba da se upali tek kada se taster pusti pa ponovo stisne.
a to cemo uraditi na sledeci nacin.

```
SYMBOL LED=PORTB.7
SYMBOL TASTER=PORTB.0
OUTPUT LED
INPUT TASTER
NovoStanje var bit
StaroStanje var bit
starostanje = taster ' ovde se uzima pocetno stanje, tj stanje
kada je taster pusten
Pocetak:
novostanje = taster   'OVDE SE UZIMA TRENUTNO STANJE NA TASTERU
if novostanje<starostanje then ' OVDE SE UPOREDJUJU STANJA. AKO JE
TASTER STISNUT TADA JE NOVO STANJE JEDNAKO 0 I SAMIM TIM JE MANJE
OD STAROG STANJA
HIGH led                  'PALI LED
PAUSE 500                 'PAUZA POLA SEKUNDE
LOW LED                   'GASI LED
ENDIF
STAROSTANJE=NOVOSTANJE 'OVAJ RED JE POTREBAN IZ RAZLOGA DA PRI
PONOVNOM NAILASKU NA IF PROMENLJIVA NNOVO STANJE NE BI BILA MANJA
OD STAROG STANJA. U OVOM SLUCAJU SE IZJEDNACAVAJU. I TAK KADA SE
PROMENI STANJE SA 1 NA 0 ONDA CE SE LEDICA UPALITI
GOTO POCETAK
END
```

evo sada je na red dosao i sedmo segmentni displej.



u principu on se sastoji od 8dioda. sedam dioda su segmenti i osma dioda je tacka.
znaci ako hocemo da napravio da se na displeju menjaju redom brojevi samo je potrebno da palimo odredjene diode. isti je slucaj kod displeja sa zajednickom anodom i katodom.
naravno samo je potrebnno izmeniti ono sto se salje na port.
za pocrtak napisacemo program da ispise broj 1 na displeju.
posto je ovo displej sa zajednickom anodom segmenti se pale kada se na njih dovede 0.
da bi se pokazao broj 1 na displeju potrebno je ukljuciti segmente b i c.
a tocemo uraditi tako sto cemo poslati broj %1111 1001
evo ptimera.
trisb=0 'port je izlazni
portb=%1111 1001 'ukljucili smo jedan
ili mozemo napisati i portb=249' to je isti broj kao gore samo u decimalnom obliku
ili mozemo napisati broj u heksadecimalnom obliku portb=$F9
kada se pise broj u heksadecimalnom obliku onda se ispred broja stavlja znak $
Broj Segment Hex Dec
0 %1100 0000 $C0 192
1 %1111 1001 $F9 249
2 %1010 0100 $A4 164
3 %1011 0000 $B0 176
4 %1001 1001 $99 153
5 %1001 0010 $92 146
6 %1000 0010 $82 130
7 %1111 1000 $F8 248
8 %1000 0000 $80 128
9 %1001 1000 $98 152
evo ga. a na vama je sada da napisete da broji sekunde od 0 do 9, i na kraju displej treba da se ugasi.
znaci da ne svetli ni jedan segment na njemu

evo i jedne ciklusne naredbe

FOR Index=Start TO End {STEP {-} Inc}
{naredbe}
NEXT {Index}
indeks je promenljiva tipa byte ili word
Start - je početna vrednost promenljive Index.
End - je krajnja vrednost promenljive Index sa čijim se dostizanjem završava FOR ciklus.
Inc - je vrednost sa kojom se u svakom prolazu povecava ili smanjuje promenljiva Index. Ako nije navedenao {STEP {-} Inc} podrazumeva se da je 1.
npr jedan primer

```
i VAR BYTE
SIMBOL LED=PORTB.1
OUTPUT LED
FOR i=1 TO 10            'broj prolaza 10
TOGGLE LED
NEXT i
```

ovaj programcic ce 10 puta promeniti stanje na ledici.
ovo je zgodno kada nesto treba da se ponovi odredjeni broj puta...
e sada da pogledamo jednu naredbu koja je jako korisna kod 7 segmetnog displeja.
npr kada bi hteli da napravimo program koji bi broio od 0 do 9 koristeci for.
to bi izgledalo

```
FOR i=0 TO 9
if i=0 then portb= 192
.
.
.
.
if i=9 then portb= 152
NEXT i
```

mao naporno zar ne...
ajde sad da pogledamo drugi nacin koristeci select case
prvo nesto da kazemo o toj naredbi

```
SELECT CASE promenljiva
CASE izraz1,izraz..
naredba
CASE izraz2 ,izraz
naredba
{CASE ELSE naredba}
END SELECT
```

promenljiva moze biti bilo kog tipa, mada ako je promenljiva bit onda ova naredba bas i nema smisla.
izraz je ono sa cime se uporedjuje promenljiva, i ako je tacna onda se izvrsava naredba ispod.
CASE ELSE naredba - ovaj red je opcioni.
znaci moze da se stavi a i nemora.
a naredba koja stoji iza case else ce se izvrsiti ukoliko

promenljiva nije bila jednaka sa bilo kojim izrazom navedenom posle case.
evo jedan primer iz help fajla.

```
SELECT CASE x
CASE 1
y = 10
CASE 2, 3
y = 20
CASE IS > 5
y = 100
CASE ELSE
y = 0
END SELECT
```

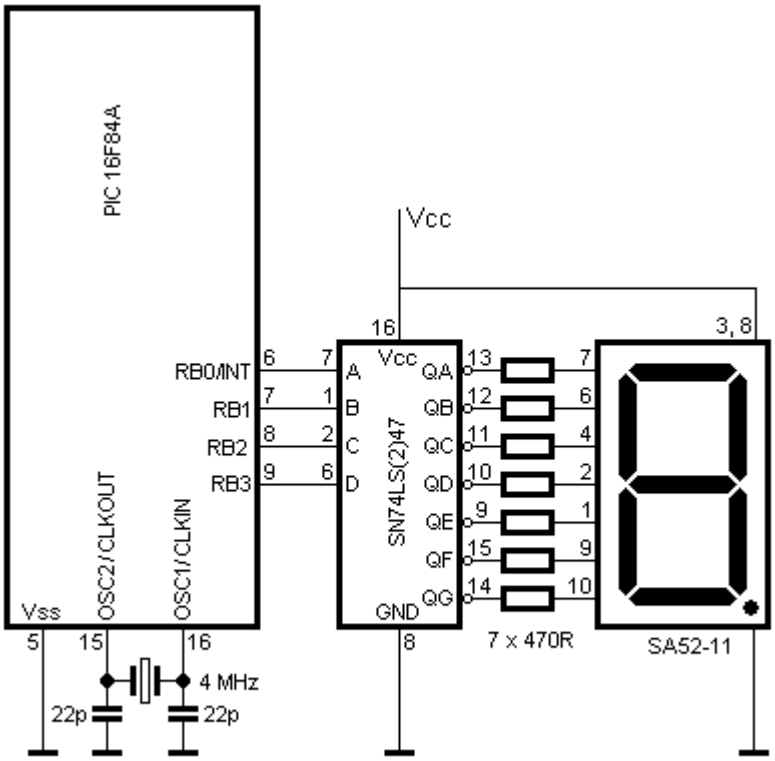
mislim da ga nema potrebe nesto posebno objasnjavati.
i ovo smo mogli iskoristiti za ispis brojeva na displeju ali nista puno nam ne skracuje posao u odnosu na naredbu if...
sto se mene tice case slobodno zaboravite. najcesce se koristi if i sledeca naredba.
a to je LOOKUP
sintaksa:
LOOKUP Indeks,[Konstanta,Konstanta...],Promenljiva
indeks predstavlja broj od nule, pa se moze menjati do onog broja koliko imamo konstanti u zagradi
promenljiva je mesto gde se smesta konstanta.
Ova naredbe koristi se za čitanje vrednosti iz tabele konstani na osnovu zadatog Indeksa. Kada je 0 to je indeks prvog člana tabele. Vrednost člana sa datim Indeksom se smešta u promenljivu. kada je indeks 1 onda se uzima vrednost druge konstante i smesta se u promenljivu.
U slučaju ako je vrednost Indeksa veća ili jednaka broju članova tabele promenljiva Var zadržava prethodnu vrednost.
ista je stvar i sa lookup2 samo sto kod te naredbe konstante mogu biti velicine word
evo primera:
Lookup i, [192,249, 164,176,153, 146, 130, 248, 128, 152], portb
u ovom slucaju ce poslati odgovarajuci broj na portb.
ako obratite paznu videcete da su to brojevi iz tabele za 7 seg. displej.
pa program za brojenje bi izgledao ovako

```
i VAR BYTE
OUTPUT PORTB
FOR i=0 TO 9
Lookup i, [192,249, 164,176,153, 146, 130, 248, 128, 152], portb
PAUSE 500
NEXT i
```

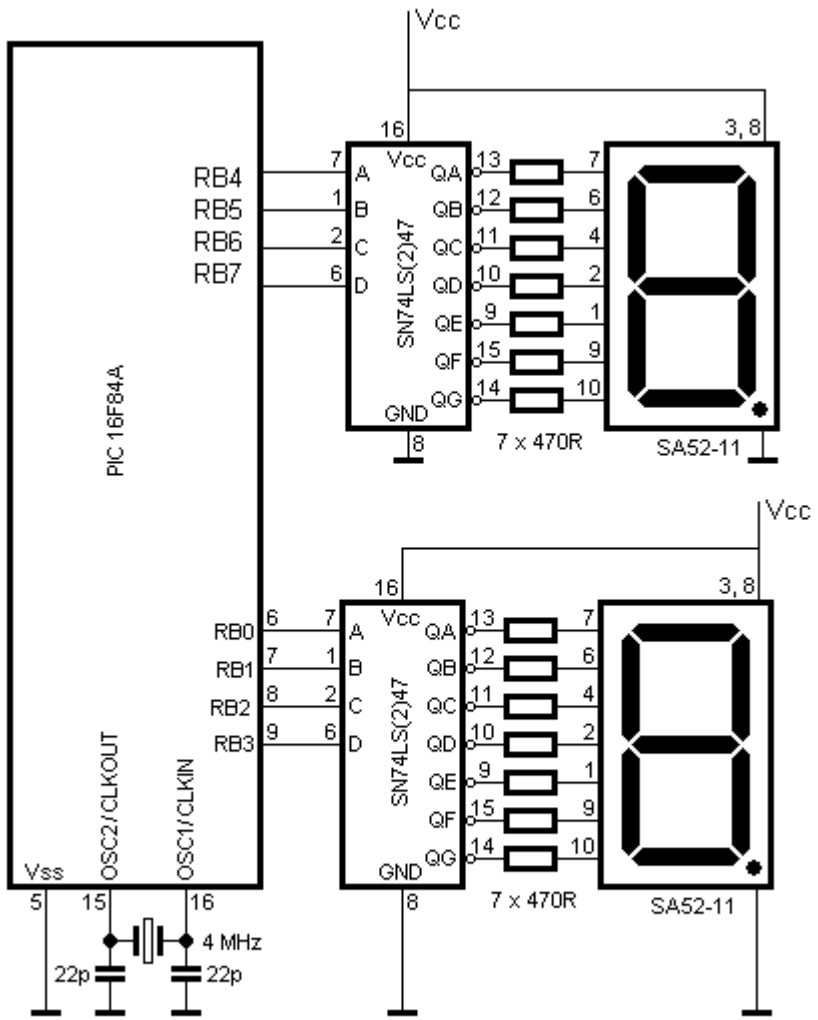
ovo je mnogo jednostavnije od bilo cega navedenog, zar ne?
sto se tice 7 seg displeja ostale su jos 2 stvari da objasnim. a to je multipleksiranje displeja, i koriscenje dekodera.
sto se tice multipleksiranja mozda bi bilo najbolje da svi instalirate proteus da nemorate plocice praviti. samo jedna napomena kod proteusa ne rade sve naredbe. npr kada se napise

output portb, proteus nece prepozndati da je portb izlazni. nego se mora koristi trisb registar. to sam tek sada primetio.
posle toga dolazi lcd displej, za koji se isto moze koristiti proteus. ali ja bi vam preporucio da nabavite displej da bi mogli videti prenos podataka sa racunara na pic. i time bi se polako priveli kraju...
pa ako imate pitanja u vezi ovoga pitajte
a evo malo domaceg
voleo bi sad da neko napravi program koji ce brojati pritiske tastera, i kada izbroji do 9, pa se ponovo taster stisne treba da se vrati na 0 i da sve ide ispocetka
ne bi trebalo da vam predstavlja problem

sema:



ovo je sedmo segmentni displej sa dekoderom. kao sto se vidi na semi prednost ovako nacina vezivanja displeja je u tome sto je za 1 displej potrebno manje pinova. tj potrebna su 4 pina...
koriscenje je manje vise jako prosto. dovoljno je poslati broj od 0-16 i na displeju ce se pojaviti broj od 0-9, a za vece brojeve uglavnom se pojavljuju brljotine na displeju.
a neki dekoderi podrzavaju i heksa decimalne brojeve pa za brojeve vece od 10 prikazace se odgovarajuca slova od A-F.
sledeci nacin je da se koriste 2 dekodera
sema:



sada je situacija nesto slozenija. ker imamo 2 broja od 0-16 koja treba sloziti u jedan bajt.
za prikazivanje na displeju ciji je dekoder prikopcan na rb0-rb3, je ista prica kao od malopre. znaci treba poslati broj od 0 do 16.

a da bi smo nesto prikazali na displeju ciji je dekoder prikopcan na rb4-rb7 moramo nekako pomeriti broj za 4 bina na gore.
ovako bi trebao da izgleda broj poslat na portb u binarnom obliku
GGGG DDDD
D-biti donjeg displeja
G-biti gornjega displeja.
a mi imamo 2 broja u sledecem obliku
0000 DDDD
0000 GGGG
---
da bismo dobili oblik kao gore najjednostavnije je pomeriti bite GGGG u levo da se dobije
GGGG 0000
kada imamo broj u tom obliku onda ga je dovoljno sabrati sa drugim brojem

```
0000 DDDD
+GGGG 0000
-----------------
GGGG DDDD
```

i dobili smo oblik koji smo zeleli.
pomeranje bita GGGG mozemo izvesti na 2 nacina.
jedan je koriscenjem matematickog operatora za siftovanje bita.
mat operator se koristi na sledeci nacin
promenljiva << broj za koliko se bita pomera sadrzaj u levo.
isti je slucaj i kod siftovanja u desno.
evo konkretan primer:

```
TRISB=0
BROJ1 VAR BYTE
BROJ2 VAR BYTE

BROJ1=5  ' donji displej treba da prikaze 5
BROJ2=2  ' gornji displej treba da prikaze 2

BROJ2=BROJ2<<4 ' ovde pomeramo donja 4 bita, u levo da donju na
mesto gornja 4 bita, sada samo treba sabrati brojeve i upisati ih
u portb
PORTB=BROJ1+BROJ2
```

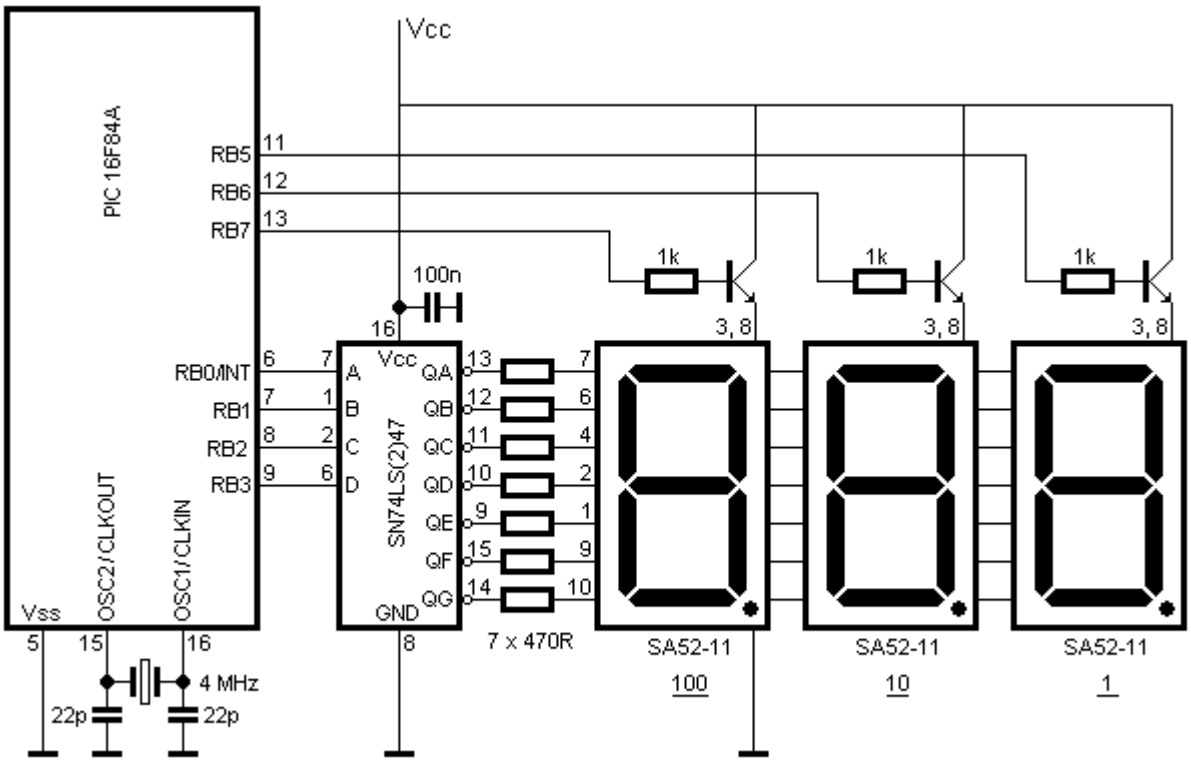
drugi nacin pomeranja u levu stranu je mnozenje sa brojem 16(ako je potrebno 4 mesta)

```
TRISB=0
BROJ1 VAR BYTE
BROJ2 VAR BYTE

BROJ1=5  ' donji displej treba da prikaze 5
BROJ2=2  ' gornji displej treba da prikaze 2

BROJ2=BROJ2*16 ' ovde pomeramo donja 4 bita, u levo da donju na
mesto gornja 4 bita, sada samo treba sabrati brojeve i upisati ih
u portb
PORTB=BROJ1+BROJ2
```

sada dolaze na red displeji u multipleksu.
sema:



ovde su svi displeji paralelno spojeni. ali postoje tranzistori pomocu kojih ukljucujemo samo odredjeni displej.
ovde se sve svodi na to da se pali prvi drugi pa treci displej... znaci ako hocemo da upalimo skroz levi displej koji nam pokazuje stotine onda cemo postaviti rb7 na 1.
evo konkretno kako se upravlja displejima.
prvo je potrebno na portb poslati broj koji zelimo da se vidi na odredjenom displeju, zatim, je potrebno ukljuciti taj displej(postaviti odgovarajuci pin na 1) drzatiga ukljucenog neko vreme, zatim ga iskljuciti , pa isto to ponoviti za sledeci displej. i ako ovo dovoljno brzo ponavljamo dobicemo utisak da displeji stalno svetle.
evo jedan kratak program koji bi trebao ispisati trocifreni broj:

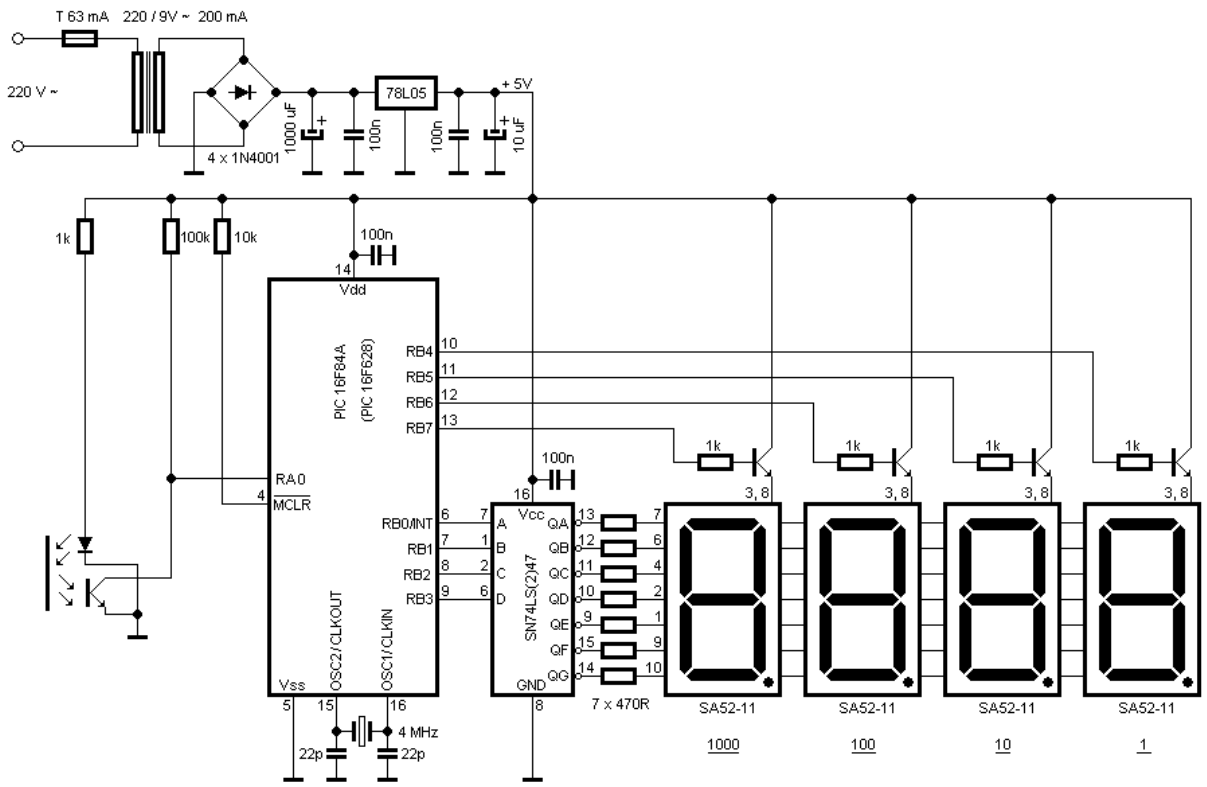
```
TRISB=0
BROJ1 VAR BYTE
BROJ2 VAR BYTE
BROJ3 VAR BYTE
BROJ1=5
BROJ2=2
BROJ2=6
POCETAK:
PORTB=BROJ1   'postavljanje prve vrednosti na port b
HIGH PORTB.7  'ukljucivanje displeja
PAUSE 5       ' zadrzavanje prikaza
LOW PORTB.7   'iskljucivanje displeja
PORTB=BROJ2  'postavljanje druge vrednosti....
HIGH PORTB.6
PAUSE 5
LOW PORTB.6
```

```
PORTB=BROJ3
HIGH PORTB.5
PAUSE 5
LOW PORTB.5
GOTO POCETAK
```

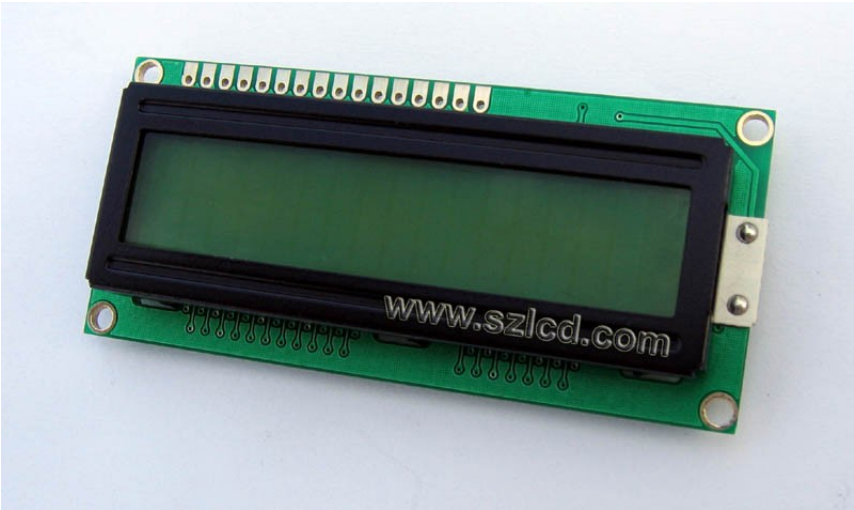
isto je ovo moguce i bez dekodera. stim sto se onda brojevi na portb salju kao kada je prikljucen samo 1 displej(najlakse koriscenjem lookup naredbe).
ovo je bilo lako jer imamo rastavljene cifre.
ali npr ako imamo neki rezultat u bajtu, tada ga je potrebno rastaviti na stotine desetice i jedinice...
za to cemo koristiti naredbu DIG. ova naredba izdvaja odredjenu cifru iz nekog broja.
evo primer
BROJ1= 123 DIG 1 ' ovo ce u promenljivu broj1 staviti vrednost cifre koja se nalazi na drugom mestu kada brojima sa desna na levo.
broj posle DIG govori koja se cifra izdvaja iz broja koji je ispred naveden. ako stoji dig 0 onda se izdvaja krajnja desna cifra, tj jedinice, ako stoji dig 4 onda se izdvaja krajnja leva cifra...
evo primera:

```
TRISB=0
BROJ VAR BYTE
BROJ1 VAR BYTE
BROJ2 VAR BYTE
BROJ3 VAR BYTE
BROJ=123
POCETAK:
BROJ1 = BROJ DIG 2' izdvajamo stotine
BROJ2 = BROJ DIG 1'desetice
BROJ3 = BROJ DIG 0'jedinice
PORTB=BROJ1
HIGH PORTB.7
PAUSE 5
LOW PORTB.7
PORTB=BROJ2
HIGH PORTB.6
PAUSE 5
LOW PORTB.6
PORTB=BROJ3
HIGH PORTB.5
PAUSE 5
LOW PORTB.5
GOTO POCETAK
```

sema:



ako je sve jasno moglo bi se reci da smo zavrsili sa sedmosegmentnim displejima....
sada bi mogli malo preci na lcd displej.
za pocetak mozete koristiti i proteus.



to bi bio primjer jednog paralelnog 16x2 LCD-a, još postoje i

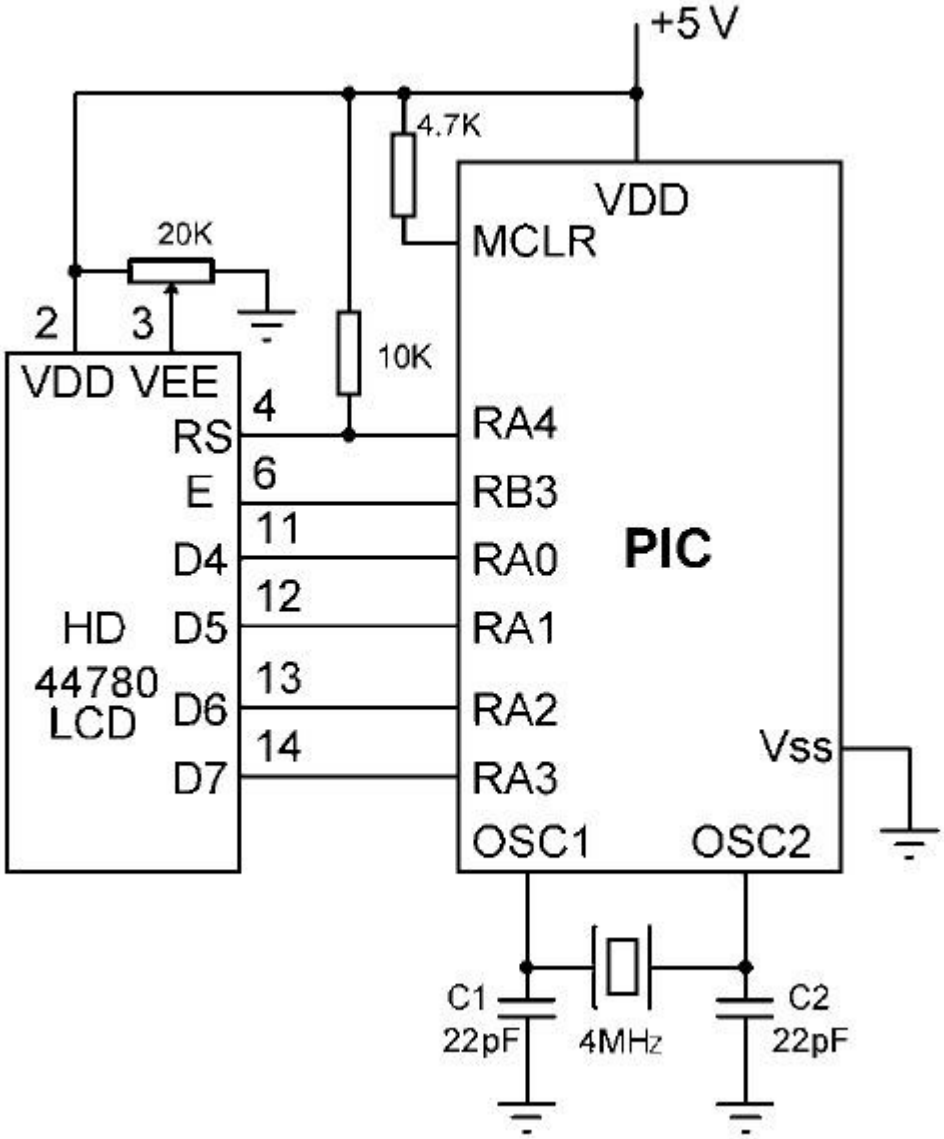
serijski al oni su kompliciranij i skuplji pa njih odmah zaboravimo 😀

za PBP koriste se LCD-ei koji u sebi koriste HD44780 kontoler, mada ima i drugih koji se mogu koristit (jedan takav je u mene)

http://home.iae.nl/users/pouweha/lcd/lcd0.shtml#pin_assignment

tu imamo raspored pinova na jednom takvom LCD-u

Kod spajanja LCD-a sa PIC-om, ako koristimo pbp trebamo koristiti odgovarajuće pinove.
Na slici je prikazano spajanje LCD-a i PIC-a:



No spajanje na ovaj način nije nužno mogu se koristiti i drugi pinovi PIC-a ali se onda se moraju definirati pinovi koji se koriste

to bi bilo malo od hardware-skog dijela sada da pređemo na pisanja programa.

za slanje podtaka na LCD koristi se jednostavna naredba LCDOUT.

Ispod imamo tabelu komandi koje se koriste uz naredbu LCDOUT.

KOMANDA OPERACIJA NA DISPLAY-u
$FE,1 obriši displej
$FE,2 vrati se na početak prve linije
$FE,$0C isključi kursor
$FE,$0E uključi underline kursor
$FE,$0F uključi blinkajući kursor
$FE,$10 pomeri kursor u levo za 1 mesto
$FE,$14 pomeri kursor u desno za 1 mesto
$FE,$80 vrati kursor na početak prve linije
$FE,$C0 vrati kursor na početak druge linije
$FE,$94 vrati kursor na početak treće linije (za one displaye koji imaju 3 linije)
$FE,$D4 vrati kursor na početak četvrte linije (za one displaye koji imaju 4 linije)

ove komande pišu se iz LCDOUT!

```
LCDOUT $FE, 1 ;briše sve sa LCD-a
```

Kod pisanja programa na početku potrebno je napraviti pausu od 0.5sek da se LCD inicializira.

```
pause 500
LCDOUT $FE, 1 ;briše sve sa LCD-a i inicijelizira ga
```

ako uzmemo

```
LCDOUT $FE, $C0  ;početak druge linije
```

onda se podatci upisuju u 2. red.

neki od načina slanja podataka na LCD:

```
LCDOUT $FE, 1, "elektronika.ba"  ; ispisuje u prvu liniju
elektronika.ba
LCDOUT $FE, $C0, "elktrophreak ; u drugu liniju ispisuje
elektrophreak
LCDOUT $FE, i ; ispisuje neku vrijednost koju smo dodjelili i
```

ako npr. želimo da podatak bude zapisan od pete pozicije u prvom redu koristit ćemo sljedeće:

```
LCDOUT $FE, $80+5, "elektro"
```

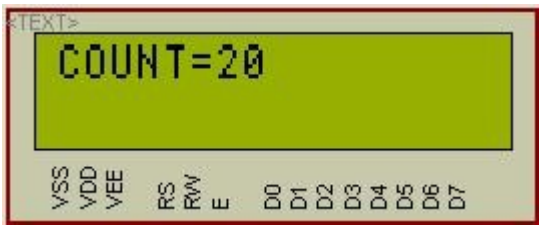
da bi vrijednosti neke konstante ili promjenjive na LCD-u predstavili u binearnom obliku koristi se naredba LCDOUT $FE, BIN i
odnosdno da bi bila u heksadekadnom ili dekadnom obliku umjesto BIN koristit će se HEX ili DEC.

ako oćemo npravi ti niz nekog znaka određeni broj puta koristit ćemo REP,
npr. ako želimo da na LCD-u imamo ispisano ***** , naredba bi izgledala LCDOUT $FE, REP "*"\5
znači gdje je "*" upisuje se znak koji se želi ponavaljati i MORA biti pod novodnicima, a 5 predstavlja broj ponvaljanja.

Ovo su bile neke osnovne naznake kod pisanja naredbih za LCD, prilično su jednostavno tako da nebi trebali imati problema sa njima!

Sada ćemo samo pokazati primjer jednog jednostavnog programa koji mjeri frekvenciju na koju dovedemo na željeni ulaz (RB1)

```
TRISB.1=1
TRISA=0
BROJ VAR WORD
pause 500
lcdout $FE, 1
POCETAK:
COUNT PORTB.1,1000,BROJ ' broji impulse na pinu RB1, u periodu od
1S i rezultat  smesta u "BROJ"
lcdout $FE, $80, "COUNT=", dec broj ; ispisuje na LCD-u COUNT= i
broj prikazuje kao decimalnu veličinu
pause 10
GOTO POCETAK
```



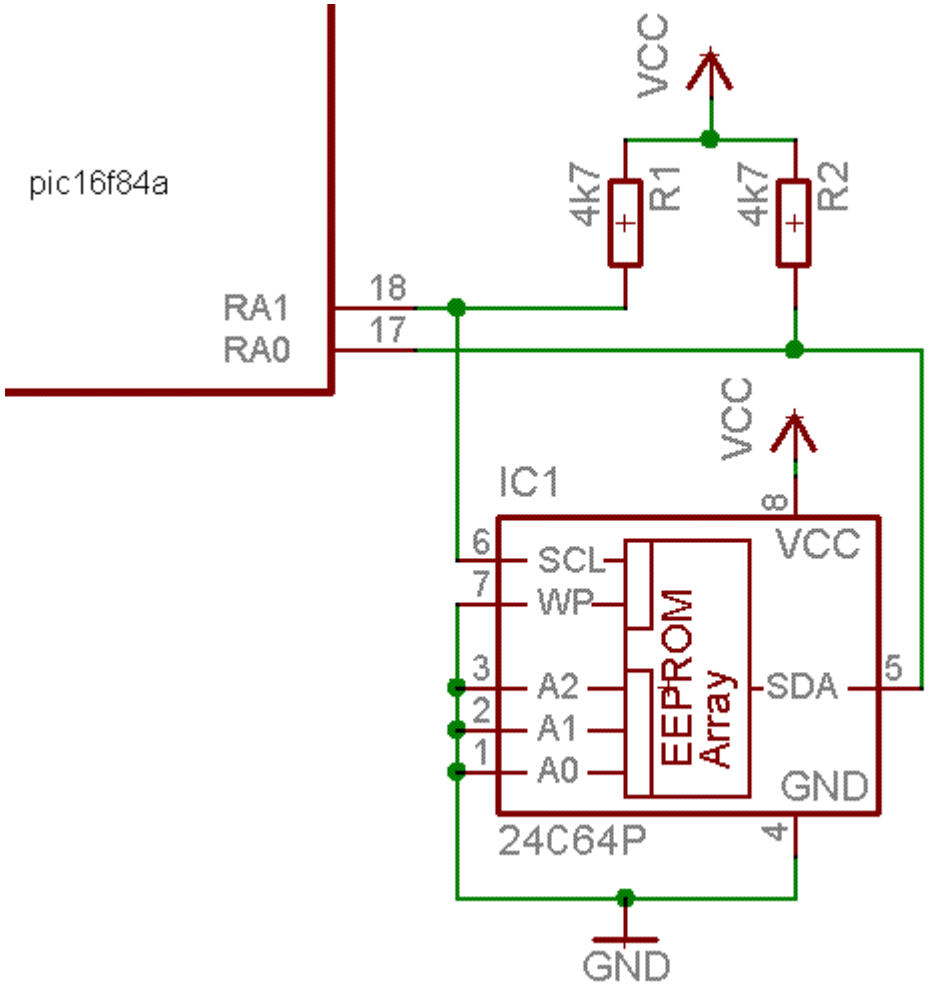
kod koriscenja define upotreba naredbi lcdout se ne menja. define samo definise kako je spojen lcd sa picem. i dodaje se na sam pocetak programa.

```
DEFINE LCD_BITS 4 'bira se da li lcd radi u 4 bitnom ili 8 bitnom modu
DEFINE LCD_DREG PORTB 'bira se port na koji su zakaceni data biti(d4-d7)
DEFINE LCD_DBIT 0 'pocetni bit data porta. 0 ili 4. ako je postavljen na 0 onda se d4 spaja na rb0,d5-rb1, d6-rb2, d7-rb3, a ako je postavljen na 4 d4 se spaja na rb4 itd..
DEFINE LCD_RSREG PORTB 'port na koji je zakacena linija RS
DEFINE LCD_RSBIT 4 'bit porta na kome je zakacena linija RS. u ovom slucaju je zakacena na RB4
DEFINE LCD_EREG PORTB 'port na kome je zakacena E linija displeja
DEFINE LCD_EBIT 3 'bit porta na kome je zakacena E linija.
DEFINE LCD_RWREG PORTE 'podesavanje sa RW liniju. mada ovo se ne ubacuje jer se RW vezuje na masu. i time je odabran upis podataka u displej.
DEFINE LCD_RWBIT 2 'RW bit
DEFINE LCD_LINES 2 'broj linija displeja. 1,2,4
DEFINE LCD_COMMANDUS 2000 'kasnjenje komande u ms. po meni je ne potrebno posebno definisati, jer je vec definisano u pbp
DEFINE LCD_DATAUS 50 'kasnjenje podataka u ms.
```

jos par napomena. umesto potenciometra moze se ubaciti otpornik(2k2) izmedju mase i pina 3. a ako se koristi ra4 sa lcd-om pull up otpornik uglavnom nije potreban jer lcd ima pull up na svim ulasnim pinovima.
kada se radi simulacija u proteusu, tada pull up mora ici...
evo jedan primer spajanja lcd-a na portb.

```
DEFINE LCD_BITS 4
DEFINE LCD_DREG PORTB
DEFINE LCD_DBIT 0
DEFINE LCD_RSREG PORTB
DEFINE LCD_RSBIT 4
DEFINE LCD_EREG PORTB
DEFINE LCD_EBIT 5
DEFINE LCD_LINES 2
pause 100 'cekanje inicijalizacije displeja
lcdout $fe,1, "pozdrav svima"
end
```

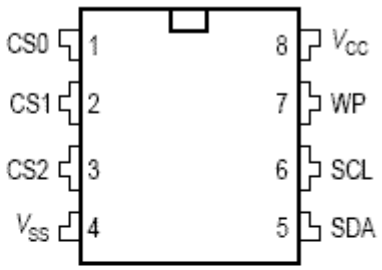
evo primer kako se koriste i2c memorije tipa 24cXX
detalj seme povezivanja eeproma:



potrebno je spojiti jos ledice na portb, i napajanje, kristal
pullup na mclr..
koja rec o memoriji.

Device	Capacity	Control	Address size
24LC01B	128 bytes	%1010xxx0	BYTE
24LC02B	256 bytes	%1010xxx0	BYTE
24LC04B	512 bytes	%1010xxb0	BYTE
24LC08B	1K bytes	%1010xbb0	BYTE
24LC16B	2K bytes	%1010bbb0	BYTE
24LC32B	4K bytes	%1010ddd0	WORD
24LC65	8K bytes	%1010ddd0	WORD

bbb = selektovanje bloka u memoriji
ddd = biti za selektovanje memorije(na memoriji pinovi od cs1-cs3)
xxx = ne uzimaju se u obzir



WP(write protection) je ulaz kojim se onemogucuje pisanje podataka u memoriju. kada je na logickoj 1 tada je moguce samo citati podatke

u tablici su dati kapaciteti memorija, od cega se sastoje kontrolni bajtovi, i koja je duzina adrese(byte ili word)
sintaksa za naredbu upisa u memoriju izgleda ovako:
I2CWRITE SDA,SCL, Kontrolni_bajt,{Adresa,}[Prom{,Prom...}]
sda, i scl predstavljaju data i clock pinove. na semi su porta.0 i porta.1, i iz tog razloga na pocetku je potrebno definisatite pinove. to se radi na sledeci nacin

```
SYMBOL SDA=porta.0
SYMBOL SCL= pota.1
DEFINE I2C_SDA PORTA,0
DEFINE I2C_SCL PORTA,1
```

pin SDA mora biti pin sa otvorenim kolektorom(scl nije obavezan da bude sa otvorenim kolektorom)
kontrolna rec se uzima iz tabele.
moze da se pise u binarnom obliku kao sto je u tabeli. ili krace u heksa decimalnom.
posto su leva 4 bita 1010 to je decimalni broj 10 ili heksadecimalni A.
stoga za kontrolnu rec cemo pisati $a0 ako su cs0-cs3 na masi.
u tablici se vidi da u kontolnu rec ulaze bitovi oznaceni sa b, i napomenuto je da sluze za odabir blokova u memoriji. npr memorija 24c04 je kapaciteta 512 bajtova. a adresa je duzine bajta, stoga vidimo da samo pomocu adrese ne mozemo pristupiti celoj memoriji. i iz tog razloga postoji ono b u kontrolnoj reci, koje sluzi dali da se selektuje jedna ili druga polovina memorije.
to je najlakse zamisliti kao da su u jednu memoriju spakovali 2 memorije od 256bajtova.
pa ako je b0 onda se pristupa jednom delu a ako je b1 onda drugom.
meni je uvek mrsku da se zezam sa time, pa ako mi je 24c02 mala, onda koristim neku 16-tobitnu memoriju(ovo 16-bitna znaci da je adresa duzine 16bita, a ne podatak)
posle svakog upisa podataka potrebno je sacekati oko 10ms, pre nego sto se krene sa upisom narednog podatka.
primer upisa u memoriju:

```
SYMBOL SDA=porta.0
SYMBOL SCL= porta.1
DEFINE I2C_SDA PORTA.0
DEFINE I2C_SCL PORTA.1
i var byte
Adr var word'(ili byte  zavisi koja se memorija koristi
for i=0 to 255
i2cwrite sda,scl,$a0,adr,[i]
pause 10
next i
```

naredba za citanje memorije
I2CWRITE SDA, SCL, Kontrolni_bajt,{Adesa,}[Prom{,Prom...}]
jedina razlika izmedju i2cwrite i i2cread je u tome sto sa i2c write pisemo u memoriju podatke a sa i2cread iscitavamo podatak sa adrese u promenljive.
postoje jos 2 stvari vezane za i2c a to su

DEFINE I2C_SCLOUT 1 'ovom naredbom se pin kloka postavlja u bipolarni mod umesto u mod sa otvorenim kolektorom. i u tom slucaju pullup na scl nije potreban
DEFINE I2C_SLOW 1 'ovo se koristi kada se na mikrokontroleru koriste oscilatori brzi od 8MHz-a
evo primer pisanja i citanja podataka iz memorije:

```
SYMBOL SDA=porta.0
SYMBOL SCL= porta.1
DEFINE I2C_SDA PORTA.0
DEFINE I2C_SCL PORTA.1
trisb=0
portb=0
i var byte
Adr var word'(ili byte  zavisi koja se memorija koristi
for i=0 to 255 step 2
adr=i
i2cwrite sda,scl,$a0,adr,[i]
pause 10
next i
for i=0 to 255
i2cread sda,scl,$a0,adr,portb  'iscitava podatak iz memorije i
smesta ga na portb
pause 500
next i
```

program upisuje na svaku drugu adresu vrednost promenljive i u drugom delu program iscitava svaku adresu i njenu vrednost smesta na portb. tako da ce se na portub pojaviti vrednost u binarnom obliku.