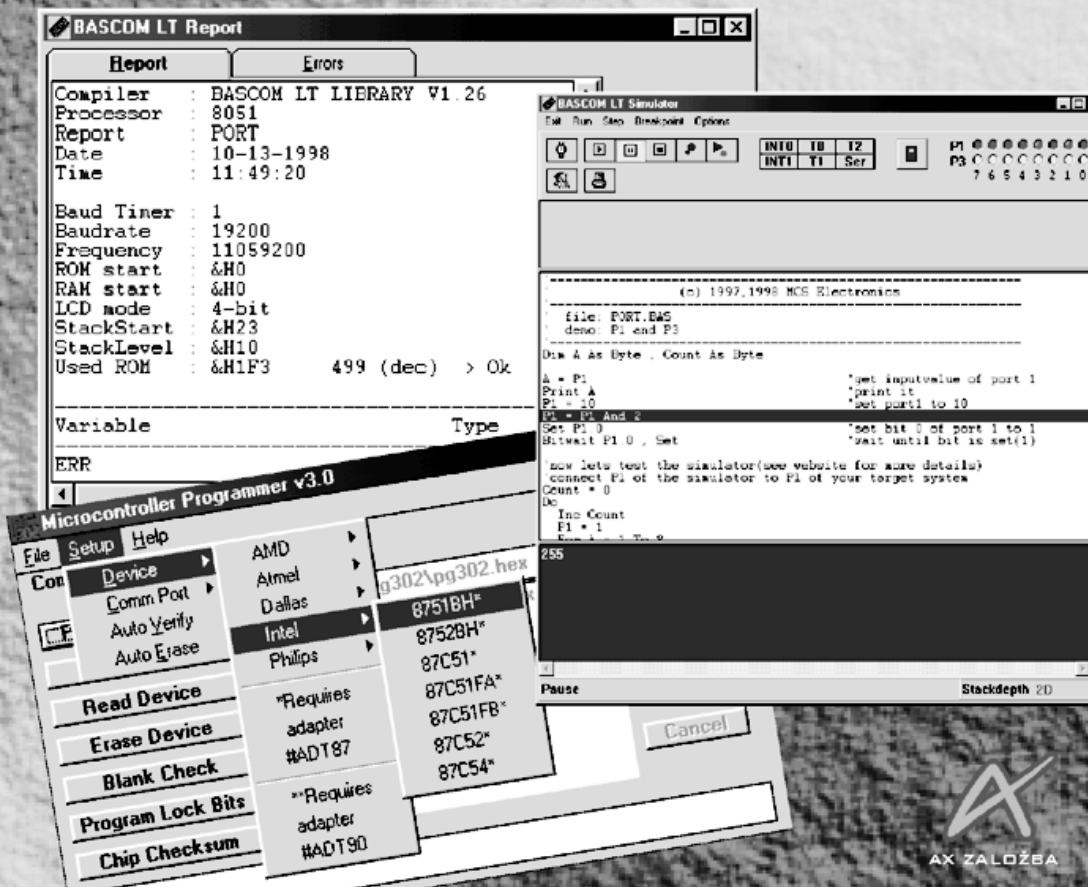
Jurij Mikeln

programski jezik BascomLT

Uvod v programiranje mikrokontrolerjev s programskim jezikom BascomLT



AX ZALOŽBA

BascomLT - uvod v programiranje mikrokontrolerjev s programskim jezikom BascomLT

Avtor: Jurij Mikeln
Uredniški odbor: Jurij Mikeln, mag. Vladimir Mitrović, Mirko Pelcl, Dragan Selan
Recenzija: mag. Vladimir Mitrović
Tehnični urednik: Mitja Zajc
Založnik: AX elektronika d.o.o., Ljubljana
Za založbo Jurij Mikeln
Oblikovanje in grafična priprava: AX elektronika d.o.o., Ljubljana
Tisk: SIDRA d.o.o., Ljubljana
Naklada: 1000 izvodov

Po mnenju Ministrstva za znanost in tehnologijo Republike Slovenije št. 415-01-59/99 z dne 15.4.1999, sodi publikacija med proizvode, za katere se plačuje 5% davek od prometa proizvodov.

CIP - Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

681.326-181.4
519.682

MIKELN, Jurij
Programski jezik BascomLT : uvod v programiranje mikrokontrolerjev s programskim jezikom BascomLT / Jurij Mikeln. - Ljubljana : AX elektronika, 1999

ISBN 961-90703-0-5

98508544

Jurij Mikeln

programski jezik BascomLT

Uvod v programiranje mikrokontrolerjev s programskim jezikom BascomLT

VSEBINA

BascomLT

Uvod v programiranje mikrokontrolerjev s programskim jezikom BascomLT

Uvod

Vsi se še spomnimo t.i. mavrice, Sinclairjevega Spectrum računalnika, ki je zelo pripomogel pri spoznavanju mikroprocesorjev in mikrokontrolerjev. Takrat so bili mikroprocesorji za marsikoga popolna neznanka in tako rekoč tabu, ki smo se ga v glavnem skoraj vsi izogibali. Razvoj elektronike pa tudi programskih jezikov in razvojnih orodij je seveda naredil svoje in sčasoma so mikroprocesorji postali vse bolj dostopni elektronikom. Danes si ne moremo več predstavljati resne naprave brez mikrokontrolerja. Namreč v času, ko so cene mikrokontrolerjev zelo nizke, dobavljivost v naši državi pa že nekaj časa ni več aktualno vprašanje, je potrebno mikrokontrolerje približati elektronikom na kar se da nevsiljiv način. In prav temu času je kot na kožo napisan programski jezik BascomLT.
Kljub temu bomo elektronika - analogista težko prepričali, da bodo mikrokontrolerji pomenili svež veter v njegovih izdelkih. Mikroprocesorjev namreč niso nikoli uspeli spoznati v takšni meri, da bi se jih lotili. Velika večina elektronikov pa nas vsaj od daleč pozna programski jezik Basic, bodisi iz časov Sinclairove mavrice, bodisi iz časov šolanja na srednji šoli. In prav tukaj je BascomLT programskemu jeziku uspelo podreti tabuje. Če bo nam s priročnikom vsaj približno tako dobro uspelo animirati bodoče uporabnike mikrokontrolerjev, potem bomo veseli, da smo vam odstrli zaveso znanja in vam odprli popolnoma nova obzorja.

BascomLT je razmeroma nov programski jezik, saj ga v Slovenij poznamo šele dobro leto dni, ko se je njegov prvi opis pojavil leta 1998 v

februarski številki revije Svet Elektronike. Od takrat naprej smo s pomočjo g.Pelcla in g.Okrožnika gradili in spoznavali BascomLT. Iz tega druženja je nastala tudi Bascom testna plošča, na kateri lahko preizkušamo naše programe, da ne omenjam primerov programov, kjer so razložene rutine programa. Avtor programa BascomLT je g. Mark Alberts, ki je vložil veliko truda in svojega časa, da je danes BascomLT zelo uporabno in cenovno sprejemljivo razvojno orodje. Avtor se je potrudil, da je v BascomLT integriral tiste ukaze in rutine, ki nas elektronike zelo zanimajo: prikaz na LCD displeju, RS232 komunkacija, I2C, 1Wire rutine in podobno. Vse omenjene rutine bi zahtevale kar sposobnega in rutiniranega programerja, da bi usposobil mikrokontroler za zahtevano nalogo. BascomLT pa nam je na razmeroma enostaven način omogočil prvo, začetno programiranje. Upamo, da bo ta priročnik večini njegovih bralcev pomenil prvo stopnico v programiranju. In želimo, da bi bil ta prvi korak kar se da zanimiv in seveda tudi poučen! Vsi, ki boste tudi sodelovali v tečaju programiranja z BascomLT, boste po tečaju znali kar solidno programirati mikrokontrolerje, med drugim boste znali tudi prikazovati podatke na LED prikazovalniku!

Na tem mestu bi se želel zahvaliti prav g. Albertsu, ki mi je pomagal pri nejasnostih programa in pokazal veliko mero potrpežljvosti z mano. Zahvaliti bi se želel tudi g.Mirku Pelclu in g.Gorazdu Okrožniku, ki sta mi pomagala premagati težave programerja - začetnika. Naš dolgoletni sodelavec mag. Vladimir Mitrović je pripomogel s svojimi nasveti profesionalnega predavatelja pa tudi z nasveti okoli samega priročnika. Za obilico nasvetov se zahvaljujem tudi g. Borisu Plutu, ki je pomagal tako organizacijsko kot tudi z nasveti pri izvedbi prvega tečaja. Na koncu pa bi se želel zahvaliti vsem sodelavcem podjetja AX Elektronika d.o.o., ki so pomagali pri izzidu tega priročnika.

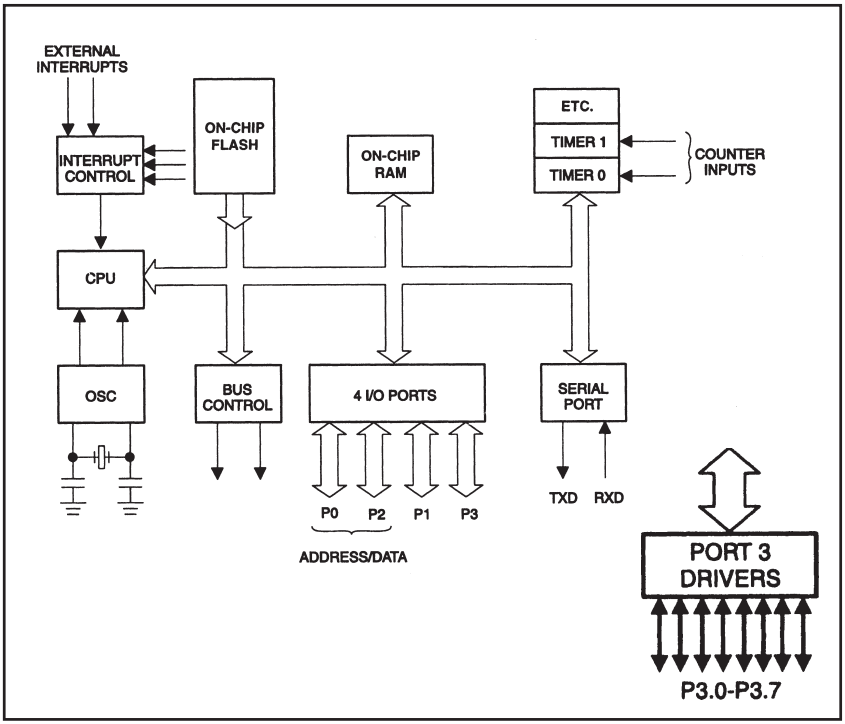
Vsem še enkrat prav lepa hvala!

Jurij Mikeln
Ljubljana, Marec 1999

Kratek uvod v mikrokontrolerje

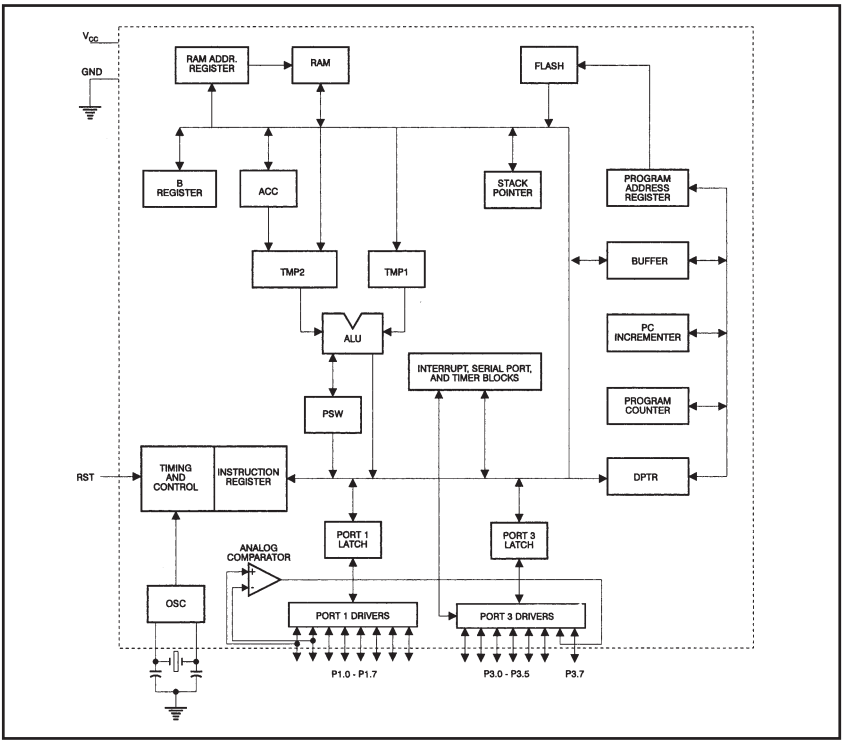
Veliko nas je, ki nam je beseda mikrokontroler zelo znana in pravzaprav zakaj bi sploh izgubljali besede o njej? No, morda pa le ni odveč, da na kratko ponovimo, kaj vse je v mikrokontrolerju in kako cela zadeva sploh deluje.

Morda nekaterim od vas zgradba mikrokontrolerja izgleda komplicirana, vendar verjemite mi, da za začetno programiranje sploh ni potrebno poznati celotnega mikrokontrolerja.
Vse, kar za začetek morate vedeti je to, da ima mikrokontroler t.i. vhodno/izhodne priključke, ki jim v programerskem žargonu rečemo porti. Večinoma ima eden port



Slika 1: Notranja zgradba mikrokontrolerja

8 priključkov, ki jih označujemo s številkami od 0 do 7. Recimo, da priključke porta P1 označujemo kot P1.0, P1.1, P1.2 itd. do P1.7. Vhodno/izhodni pomeni, da lahko cel port ali samo eden njegov priključek nastavimo kot vhod oziroma izhod. To, kako nastavimo ali "sprogramiramo" določen port kot vhod ali izhod, je odvisno od programa, ki smo ga napisali in shranili v mikrokontrolerju. Porti so v mikrokontrolerju preko vodila povezani z drugimi elementi mikrokontrolerja. Najpomembnejši od teh elementov je centralno procesna enota ali CPU, kjer se pravzaprav dogaja izvajanje programa, ki smo ga napisali. Na sliki 1 vidimo, da CPU za svoje delovanje potrebuje takt, ki mu ga zagotavlja vgrajeni oscilator. Običajno je ta oscilator narejen tako, da z minimalnim številom zunanjih delov zagotavlja stabilen in točen takt. Kajti samo če je takt stabilen in točen, bo delovanje mikrokontrolerja takšno, kot smo ga sprogramirali.



Slika 1a: Blok shema mikrokontrolerja AT89C2051

Bodi dovolj besed o zgradbi mikrokontrolerja. Ostale dele mikrokontrolerja bomo spoznali sproti. Čas je, da spoznamo prvi primer programa, s katerim bomo prižigali in ugašali LED diodo na enem od vhodno/izhodnih portov.

Več luči!

Skoraj vsi elektroniki smo si v začetku svojega druženja z elektroniko želeli narediti enostavno vezje z premikajočimi lučkami. Iz predala smo vzeli NE555, ki je služil za takt, temu smo dodali CD4017, ki je števec do 10. Kar nekaj elementov se je nabralo za samo eno možno kombinacijo prižiganja lučk. Kako narediti enostavno vezje, ki vam omogoča veliko možnosti prižiganja in premikanja lučk?

Vezje premikajočih lučk bo temeljilo na najbolj enostavni uporabi integriranega vezja AT89C2051, ki ga v našem prodajnem servisu dobite po akcijski ceni že za 399 SIT.

Prižgimo lučko

Lučke lahko prižigamo na več načinov. Najprej vam bom pokazal, kako sploh prižgati eno LED diodo na mikrokontrolerju. Ko boste prvič pognali BascomLT program, najprej odprite nov list z ukazom File New ali preko tipkovnice s pritiskom na Ctrl+N. Nato poimenujte vašo spremenljivko in jo definirajte.
Definirate jo lahko kot: **bit** *(vrednost spremenljivke je lahko samo 0 ali 1),* **byte** *(je 8 bitov, vrednost spremenljivke je lahko med 0 in 255),* **word** *(je dva byta, vrednost spremenljivke je lahko med 0 do 65535) ali* **integer** *(vrednost spremenljivke je lahko med -32767 do +32768). Spremenljivk je sicer še nekaj vendar jih zaenkrat ne bomo uporabljali. Znak' v BascomLT pomeni, da se za znakom' nahaja komentar, ki ne vpliva na izvajanje programa.*

Spremenljivke definiramo z ukazom **Dim**, *kot je prikazano v primeru.*

```
Dim Lucka as bit          'spremenljivka Lucka bo tipa Bit, ker se bo
                          'samo prizigala oziroma ugasala
```

*S tem ukazom smo definirali spremenljivko "***Lucka***" in ji definirali tip. Ker se bo* **Lucka** *samo prižigala oziroma ugašala (logična 0 oziroma logična 1), je spremenljivka lahko tipa* **Bit**. *Omenil sem, da se* **Lučka** *prižiga oziroma ugaša z logično 0 ali 1. Ne vem zakaj, ampak nekako smo vajeni, da LED diode prižigamo z logično 1, ne pa z logično 0.*

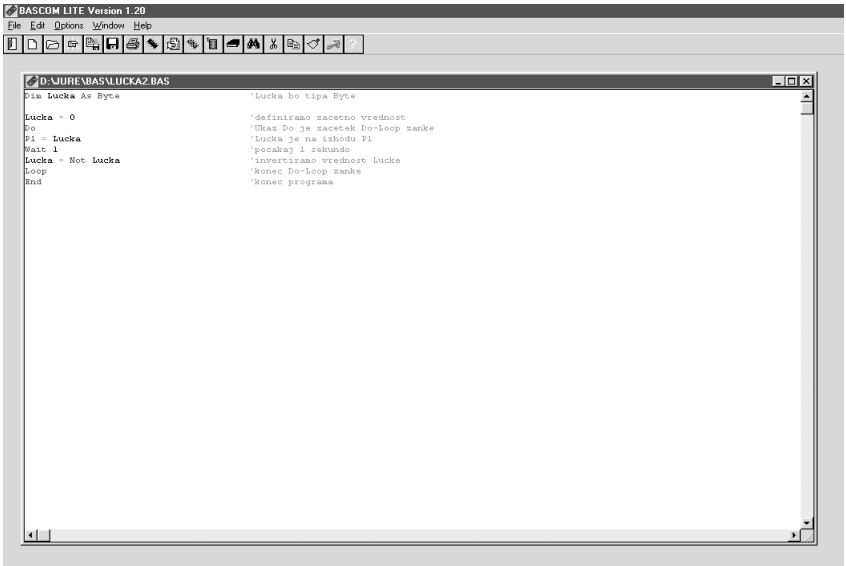
No, pri mikrokontrolerjih je v tem primeru drugače: namreč porti imajo zanimivo lastnost, da v stanju logične 1 zmorejo le nekaj µA toka, medtem ko z logično 0

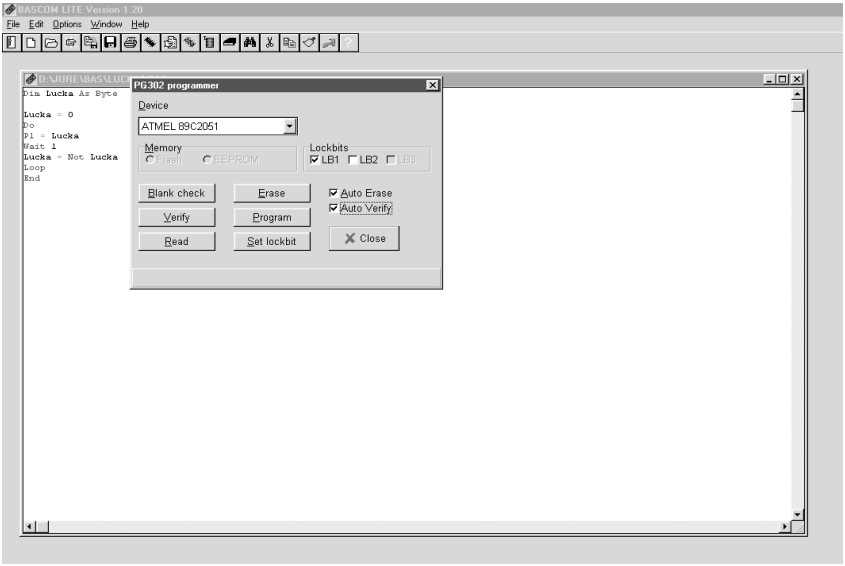
zmorejo do 25 mA toka. Zato LED diodo zvežemo tako, da je anoda stalno vezana na +5V napajanja, katoda pa je vezana na port mikrokontrolerja. V primeru, da 25 mA toka ni dovolj, moramo na izhod mikrokontrolerja vezati ojačevalnik ali buffer.

Zdaj poskusimo nadaljevati tako, da bomo našo spremenljivko **Lucka** *prižigali in ugašali. To naredimo takole:*

```
'program Lucka1.bas
Dim Lucka as bit          'Lucka bo tipa Bit, ker se bo samo prizigala
                          'oziroma ugasala
Lucka = 0                 'definiramo zacetno vrednost spremenljivke
Do                        'Ukaz Do je zacetek Do-Loop zanke
P1.0 = Lucka              'Lucka je na izhodu p1.0
wait 1                    'pocakaj 1 sekundo
Lucka = not Lucka         'invertiramo vrednost Lucke
Loop                      'konec Do-Loop zanke
End                       'konec programa
```

Najprej določimo vrednost spremenljivke **Lucka**. *To je potrebno zato, da se ob resetu oziroma vklopu mikrokontrolerja spremenljivka ne postavi na naključno vrednost. Zanka* **Do-Loop** *je neskončna zanka. Ukazi, ki so vsebovani v tej zanki, se izvajajo v neskončnost. Ukaz* **not** *pomeni, da invertiramo vrednost spremenljivke. Kaj to pome-*





ni v našem primeru? Recimo, da ima spremenljivka **Lucka** *vrednost 0. Z ukazom* **Lucka = not Lucka** *spremeni vrednost spremenljivke* **Lucka** *iz 0 v 1. Če pa bi spremenljivka* **Lucka** *imela vrednost 1, bi po izvrsitvi ukaza* **Lucka = not Lucka** *imela vrednost 0. Ukaz "***Do***" je začetek zanke, vmes je program, ki se izvaja korak za korakom, "***Loop***" pa konec zanke in vrne izvajanje programa na začetek zanke. Ko smo napisali program, pritisnemo* **F7***, ali z miško kliknemo na gumb* **Compile**. *Programator PG302 priklopimo na napajanje (+18 do +24V DC), pritisnemo* **Ctrl+A** *in odprlo se bo okno programatorja. V kolikor še nimamo nastavljen programator na PG302, to izberemo v meniju "***Options, Programmer***". Ko je programator nastavljen, pritisnemo* **Ctrl+A** *in izberemo tip mikrokontrolerja. To storimo s klikom miške v okno* **Device***., izberemo* **Atmel** *in v novem oknu izberemo* **AT89C2051**. *Končno lahko pritisnemo na tipko "***Program***", ki nam bo sprožila postopek programiranja. Sprogramiran kontroler vzamemo iz programatorja, ga vtaknemo v Bascom testno ploščo in priklopimo napajalno napetost (+5V DC).*

Tako! Zdaj nam LED dioda lepo utripa na izhodu P1.0. Če bi želeli, da bi **Lucka** *utripala na drugem izhodu - recimo na P1.2, potem v programu spremenimo samo del, kjer določamo izhod:*

```
P1.2 = Lucka
```

Vse ostalo ostane isto. Enostavno kajne!

Prižgimo več lučk hkrati

Podobno bomo prižigali in ugašali celo skupino lučk. Program spremenimo samo v dveh vrsticah.

```
'program Lucka2.bas
Dim Lucka as byte       'Lucka bo tipa Byte, ker se bo prizigala
                        'oziroma ugasala na portu P1 (vseh 8 luck)
                        'vrednost bo od 0 do 255
Lucka = 0               'definiramo zacetno vrednost
Do                      'Ukaz Do je zacetek Do-Loop zanke
P1 = Lucka              'Lucka je na izhodu p1
wait 1                  'pocakaj 1 sekundo
Lucka = not Lucka       'invertiramo vrednost Lucke
Loop                    'konec Do-Loop zanke
End                     'konec programa
```

V tem primeru je spremenljivka **Lucka** *tipa* **Byte**, *kar pomeni, da je njena binarna vrednost od 00000000 do 11111111. Tema binarnima vrednostima ustrezata desetiški 0 in 255 (vzemite kalkulator v roke in preverite, če ne verjamete).*

Če nimate kalkulatorja, si desetiško vrednost izračunate po sledeči formuli:

$$DEC = P3.0 \times 2^0 + P3.1 \times 2^1 + P3.2 \times 2^2 + P3.3 \times 2^3 + P3.4 \times 2^4 + P3.5 \times 2^5 + P3.6 \times 2^6 + P3.7 \times 2^7$$

Pri tem morate paziti, da je 2^0 vedno 1!

Premikajoče lučke

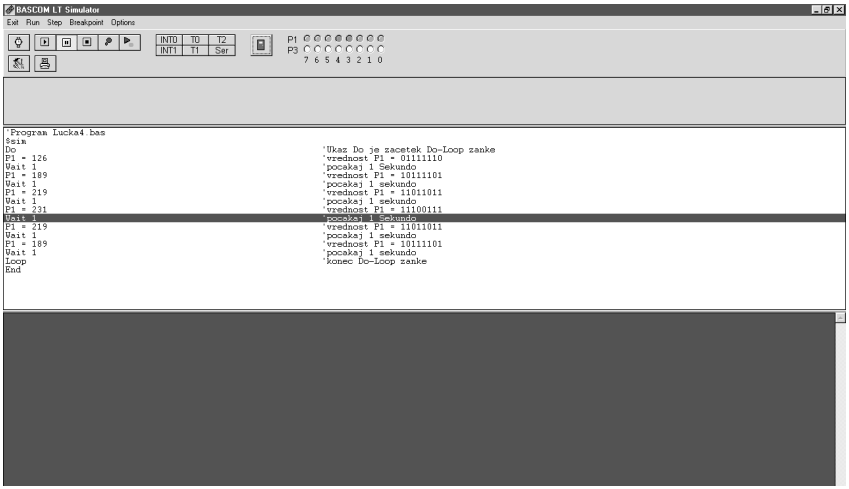
No, zdaj se pa že lahko izkaže vaša domišljija. Kdo pa pravi, da naj bo začetna vrednost spremenljivke **Lucka = 0***? Kaj pa, če bi bila začetna vrednost 146 (binarno 10010010)? Poskusimo. Na izhodu se bosta pojavljali kombinaciji 10010010 in njena invertirana vrednost 01101101.*

Popolnoma isto bi dosegli, če bi v program direktno vpisali, katere vrednosti naj bodo na izhodu P1. Poglejmo si primer:

```
'program  Lucka3.bas
Dim Lucka As Byte       'Lucka bo tipa Byte, ker se bo prizigala
                        'oziroma ugasala na portu P1 (vseh 8 luck)
                        'vrednost bo od 0 do 255
Lucka = 146             'postavimo na zacetno vrednost: 10010010
Do                      'Ukaz Do je zacetek Do-Loop zanke
P1 = Lucka              'Lucka je na izhodu P1
Wait 1                  'pocakaj 1 sekundo
Lucka = 36              'vrednost Lucke je 00100100
P1 = Lucka              'Lucka je na izhodu P1
Wait 1                  'pocakaj 1 sekundo
Lucka = 73              'vrednost Lucke je 01001001
P1 = Lucka              'Lucka je na izhodu P1
Wait 1                  'pocakaj 1 sekundo
Lucka = 146             'vrednost Lucke je 10010010
Loop                    'konec Do-Loop zanke
End                     'konec programa
```

In kaj smo dobili s primerom **Lucka3***? S simulatorjem, ki je že vgrajen v BascomLT, si lahko na hitro pogledate izvajanje programa. Pri tem vas moram spomniti, da na vrhu programa, ki ga želite spremljati s simulatorjem, napišete ukaz:* **$sim**. *Ta ukaz bo onemogočil časovne zanke (kot npr.* **wait 1***), ki bi upočasnile izvajanje in simulacijo na simulatorju. Isti ukaz pa morate izbrisati, ko prevajate program za programiranje v mikrokontroler. Torej, kot rečeno, napišite ukaz* **$sim**, *prevedite program s*

pritiskom **F7** *in ko je program preveden pritisnite* **Ctrl+M**. *S tem boste štartali simulator, ki ga vidite na sliki. Izvajanje simulacije programa se prične s pritiskom na* **RUN** *in ustavi s pritiskom na tipko* **STOP**.



Poglejte si še tabelo 1, kjer so po vrsti napisane binarne kombinacije za vrednost spremenljivke **Lucka**.

0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	0
0	1	0	0	1	0	0	1

Tabela 1: Vrednost spremenljivke Lucka

V tabeli 1 vidimo, da smo dosegli učinek premikajoče luči na zelo enostaven način. Ne smemo pozabiti, da 0 v tabeli pomeni, da LED dioda priključena na port P1 gori, 1 pa pomeni, da LED dioda ne gori. Naj pa vas ne moti, da na simulatorju lučke gorijo, kadar je na njih logična 1.
Kot ste videli, smo v programu vpisali določeno vrednost za spremenljivko in potem smo to vrednost prikazali na izhodnem portu P1.

Vam že dela domišljija?! Vas je uporaba mikrokontrolerja pritegnila? Če že, potem bom pokazal še kakšen primer premikajoče luči.

Recimo, da bi želeli da se lučke začno prižigati od zunaj navznoter. Predstavljajte si primer s pomočjo tabele 2.

```
0 1 1 1 1 1 1 0
1 0 1 1 1 1 0 1
1 1 0 1 1 0 1 1
1 1 1 0 0 1 1 1
1 1 0 1 1 0 1 1
1 0 1 1 1 1 0 1
0 1 1 1 1 1 1 0
```

Tabela 2: Primer Lucka4.bas

Ker želimo primer **Lučka4** *narediti podobno kot vse predhodne, moramo vzeti v roke kalkulator in izračunati desetiške vrednosti za tabelo 2.*

```
0 1 1 1 1 1 1 0   =   1 2 6
1 0 1 1 1 1 0 1   =   1 8 9
1 1 0 1 1 0 1 1   =   2 1 9
1 1 1 0 0 1 1 1   =   2 3 1
1 1 0 1 1 0 1 1   =   2 1 9
1 0 1 1 1 1 0 1   =   1 8 9
0 1 1 1 1 1 1 0   =   1 2 6
```

Tako, zdaj pa k pisanju programa. Tokrat vam bom pokazal, da včasih sploh ne potrebujemo spremenljivke. Pravzaprav bi lahko kar direktno vpisali vrednost na P1 skladno z našo tabelo 2. No pa dajmo.

```
'Program Lucka4.bas
Do                      'Ukaz Do je zacetek Do-Loop zanke
P1 = 126                'vrednost P1 = 01111110
Wait 1                  'pocakaj 1 sekundo
P1 = 189                'vrednost P1 = 10111101
Wait 1                  'pocakaj 1 sekundo
P1 = 219                'vrednost P1 = 11011011
Wait 1                  'pocakaj 1 sekundo
P1 = 231                'vrednost P1 = 11100111
Wait 1                  'pocakaj 1 sekundo
P1 = 219                'vrednost P1 = 11011011
Wait 1                  'pocakaj 1 sekundo
P1 = 189                'vrednost P1 = 10111101
Wait 1                  'pocakaj 1 sekundo
Loop                    'konec Do-Loop zanke
End
```

Naš program se bo odvijal v neskončnost, ker se nahaja v **Do-Loop** *zanki.*

Namesto konca pa še naloga

Napišite enostaven program, ki bo prižigal lučke po sledeči tabeli:

```
1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
```

Ne pozabite, da 0 v tabeli pomeni, da LED dioda na izhodu P1 gori in 1 pomeni, da LED dioda na izhodu P1 ne gori. Program naj vsebuje ukaze, ki smo jih spoznali v temu poglavju.

Vhodi in izhodi

V tem poglavju bomo spoznali, kako mikrokontroler ve, da je pritisnjena tipka, ki je vezana na enega od njegovih portov.

Kot sem omenil v začetku priročnika, so mikrokontrolerjevi porti lahko vhodi ali izhodi. To določa program, ki ga vpišemo v mikrokontroler. Kako bi torej dopovedali mikrokontrolerju, da sta na njegovih portih P3.2 in P3.3 vezani tipki? Enostavno! Poglejmo si primer **Lucke2.bas**:

```
**'primer, ko potujoca lucka ugasa od prve do zadnje**
**'UP = P3.2, DOWN = P3.3, tipki za gor in dol**

Dim Lucke As Byte , Up As Bit , Down As Bit

Lucke = 1               'postavimo zacetno vrednost spremenljivke Lucke
P1 = Lucke              'Lucke so na P1

Do                      'zacetek DO-LOOP zanke
Up = P3.2               'tipka Up je na portu P3.2
Down = P3.3             'tipka Down je na portu P3.3
```

Z ukazom **"Up = P3.2"** *mikrokontrolerju določimo, da se port P3.2 "preslika v spremenljivko z imenom* **Up**. *Zdaj že lahko s to spremenljivko počnemo vse, kar se da početi s spremenljivko tipa* **Bit**. *Eden od ukazov je t.i.* **IF...THEN** *stavek, kjer bomo ugotavljali, kaj se dogaja s spremenljivko* **Up**. *Stavek* **IF....THEN** *v prevodu pomeni:* **ČE (pogoj) POTEM**. *V mikrokontrolerju se pa odvija takole: ko pride CPU do ukaza*

```
If Up = 0 Then          'ce je Up = 0 potem na se lucke
```

primerja, če je spremenljivka **Up = 0**. *Če ni, nadaljuje izvajanje programa pri zaključku* **IF....THEN** *stavka, ki ga predstavlja ukaz* **END IF**. *Hm, v našem primeru sta kar dva* **END IF** *ukaza. Katerega upošteva CPU? Tukaj obstaja pravilo, da se upošteva prvi* **END IF** *stavek, na katerega naleti v izvajanju programa. V našem primeru se bo program izvajal takole: če CPU ugotovi, da je* **Up = 0**, *potem gre v izvajanje zanke, ki se začne z ukazom* **Rotate Lucke , Left** .

```
If Up = 0 Then          'ce je Up = 0 potem na se lucke
   Rotate Lucke , Left  'vrtijo v levo
   P1 = Lucke           'prikazi Lucke na P1
   Wait 1               'pocakaj 1 sekundo, drugace gre vse prehitro
   If Lucke = 0 Then    'ce so Lucke = 0 potem
      Wait 1            'pocakaj 1 sekundo
      Lucke = 1         'postavi vrednost spremenljivke Lucke
      P1 = Lucke        'postavi Lucke na P1
      Wait 1            'pocakaj 1 sekundo
   End If
End If
```

Ko pride do ponovnega **IF...THEN** *stavka, zopet pogleda, ali je pogoj izpolnjen. Če pogoj ni izpolnjen, se program nadaljuje na prvem* **END IF** *stavku, prva zanka pa se zaključi z drugim* **END IF** *stavkom.*

Naj pojasnim še ukaz **Rotate Lucke , Left** . **Rotate** *v angleščini pomeni* **(za)vrti**, *kar sprva ne zveni logično. Kaj naj bi se vrtelo? Poglejmo si recimo stanje na portu P1, ki je v nekem trenutku 01000010. Po izvedbi ukaza* **Rotate Lucke , Left** *, bo stanje na portu P1 sledeče: 10000100. In, če bi ukaz ponavljali, bi se stanje na portu P1 spreminjalo takole: 00001000, 00010000, 00100000, 01000000, 10000000, 00000000. Ko bi naslednjič izvedli ukaz* **Rotate Lucke , Left** *, se stanje ne bi spremenilo in bi ostalo 00000000. Ker želimo dobiti učinek vrtečih se lučk, je potrebno ugotoviti, kdaj je stanje na portu P1 = 00000000 in nato postaviti začetno vrednost P1 = 00000001 ter ponovno pognati ukaz* **Rotate Lucke , Left** .

S tipko **Down** *naredimo podobno proceduro, kot za tipko* **Up**.

```
If Down = 0 Then                'podobna procedura, kot je za Up tipko,
je tudi
   Rotate Lucke , Right
   P1 = Lucke                   'za Down
   Wait 1
   If Lucke = 0 Then

   ' Wait 1                     primer napacnega vrstnega reda

      Lucke = 128
      P1 = Lucke
```

```
      Wait 1                    'ta wait 1 je na pravem mestu
   End If
End If
Loop

End                             'konec programa
```

Tako, primer uporabe tipke nam je zdaj znan, morda ostanimo še nekaj časa pri temu primeru, kajti pri programiranju se nam lahko vtihotapi napaka, ki jo bomo težko odpravili. Poglejmo si izsek programa, ki smo ga ravnokar napisali:

```
P1 = Lucke                      'za Down
   Wait 1
   If Lucke = 0 Then

    ' Wait 1                    primer napacnega vrstnega reda

    Lucke = 128
    P1 = Lucke
    Wait 1                      'tale Wait 1 je na pravem mestu
    End if
```

Ukaz **P1 = Lucke** *preslika vrednost spremenljivke* **Lucke** *na port P1. Potem počakamo 1 sekundo, nakar gremo v* **IF-THEN** *zanko. Smisel vseh zakasnitev v tem programu je to, da sploh opazimo gibanje lučk.*
Če se lučke prehitro prižigajo oziroma ugašajo, ne bi videli nič drugega kot brlenje lučk. Zato sem pri programiranju spregledal napako, ki se mi je vtihotapila v program. Spregledal sem dejstvo, da program ne bo deloval, tako kot bi želel. V primeru napačnega **Wait 1** *ukaza bi program najprej preveril, če je izpolnjen pogoj* **Lucke = 0**. *Če je pogoj izpolnjen, bi počakal 1 sekundo, v spremenljivko* **Lucke** *bi postavil vrednost 128 (1000000 binarno) in* **Lucke** *preslikal na P1.To bi v praksi pomenilo, da bi eno zaporedje preskočili.*

Vam ni jasno zakaj? Vam verjamem. Tudi meni ni bilo, dokler nisem videl, kako se LED diode prižigajo in ugašajo. Zdaj je čas, da to tudi vi sami ugotovite. Pri tem si pomagajte z Bascom testno ploščo, kjer boste program preizkusili.

VEDNO si moramo zapomniti, da mikrokontroler izvaja ukaze lepo po vrsti in točno tako, kot smo jih napisali.

Zaigrajmo melodijo

V tretjem poglavju bomo spoznali, da lahko naš mikrokontroler zaigra poljubno melodijo. Celo sami bomo lahko igrali nanj.

Za primer, na katerem bom razložil uporabljene ukaze, bomo vzeli program **Klavir.bas**. *S tem programom bomo naredili "klavir" z devetimi tipkami. Če hočemo, da se bo naš mikrokontroler obnašal kot klavir, moramo definirati, kateri port bo služil kot tipke. Ker ima vsak port 8 bitov, bomo potrebovali še 1 bit drugega porta, če želimo imeti klavir z 9 tipkami. Na enem od prostih bitov drugega porta bomo zvezali zvočnik.*

Zdaj že kar dobro poznamo programski jezik BascomLT in verjetno ni potrebno zelo podrobno razlagati prav vseh ukazov. Na začetku programa **Klavir.bas** *dimenzioniramo spremenljivke in definiramo vhodne tipke.*

Program nadaljujemo s serijo **IF-THEN** *stavkov, kjer predpišemo vsaki tipki svoj ton. Program deluje takole: če je pritisnjena tipka* **Ti1**, *potem nastavi spremenljivko* **Traj = 500** , *ki pomeni dolžino trajanja tona (500 ms) in nato pojdi v subrutino* **Zvok1**. *Če je pritisnjena tipka* **Ti2** *nastavi* **Traj = 500** *in pojdi v subrutino* **Zvok2** *itd.*

```
'primer klavirja
'dimenzioniranje spremenljivk
Dim Frekvenca As Word , Pom As Word , Traj As Word , Pom1 As Word
Dim Ti1 As Bit , Ti2 As Bit , Ti3 As Bit , Ti4 As Bit , Ti5 As Bit
Dim Ti6 As Bit , Ti7 As Bit , Ti8 As Bit , Ti0 As Bit

Do

Ti0 = P3.0              'definicija portov in tipk
Ti5 = P3.5
Ti1 = P3.1
Ti2 = P3.2
Ti4 = P3.4
Ti3 = P3.3
Ti6 = P1.7
Ti7 = P1.6
Ti8 = P1.5
```

```
                        'določitev tona vsake posamezne tipke
If Ti1 = 0 Then
   Traj = 500
Gosub Zvok1
End If

If Ti2 = 0 Then
   Traj = 500
Gosub Zvok2
End If
.
.
.
If Ti8 = 0 Then
   Traj = 500
Gosub Zvok8
End If
```

V omenjenih **IF-THEN** *stavkih ste opazili nov ukaz:* **GOSUB <ime subrutine>**. *Ukaz* **GOSUB** *pomeni, da se izvrševanja programa nadaljuje v t.i.* **subrutini**. *Ko se* **subrutina** *zaključi, se izvajanje programa nadaljuje tam, kjer je skočil v* **subrutino**. *Ukaz* **Loop** *pa že poznamo od prej.*

```
Loop

End
```

Zdaj si pa še poglejmo eno od **subrutin** *za določanje posameznega tona. Tudi tukaj bomo spoznali nekaj novih in praktičnih ukazov. Eden od njih je ukaz* **Restore <ime tabele>**. *Prevod besede* **restore** *pomeni* **obnovi**. *V našem primeru se program odvija takole: ko pride do ukaza* **Restore Tabela0**, *shrani* **Tabelo0** *v svoj spomin in jo po vrsti uporablja pri ukazu* **Read <ime tabele>**. *Oboje poteka zaporedno: ko ukaz* **restore** *obnovi podatke iz tabele in jih shrani v spomin, jih ukaz* **read** *prebere iz spomina po vrstnem redu od prvega do zadnjega.*

```
Zvok0:                          'subrutina za Zvok0
Restore Tabela0
      Read Frekvenca
         Sound P3.7 , Traj , Frekvenca
Return
```

Še en nov ukaz je v subrutini: to je ukaz **Sound <ime porta>, <trajanje>, <frekvenca>**. *Ukaz* **Sound** *nam precej olajša programiranje. Prevod besede* **sound** *pomeni* **zvok** *oz.* **ton**. *V bistvu s tem ukazom generirano kvadratni signal na katerem koli prostem portu, določimo čas trajanja tega signala in njegovo frekvenco.*

```
Zvok1:
Restore Tabela1
      Read Frekvenca
         Sound P3.7 , Traj , Frekvenca
Return

.
.
.
.

Zvok8:
Restore Tabela8
      Read Frekvenca
         Sound P3.7 , Traj , Frekvenca
Return
```

Na vrsti je še zadnji del programa. To so **tabele** *za vsak ton posebej, kjer določimo frekvenco tona. Sintaksa je:*

```
Tabela:
Data <frekvenca> %
```

Naj vas znak **%** *ne moti. Tukaj je zato, ker je takšna pač sintaksa BascomLT jezika.*

Predlagam, da zdaj sprogramirate kontroler s programom klavir, povežete zvočnik na izhodni port tako, kot je predlagano v **help** *dokumentaciji programa BascomLT v poglavju* **Apendix D** *in si zaigrate melodijo po želji.*

```
'definicije posameznih tonov

Tabela0:
Data 250%

Tabela1:
Data 230%

Tabela2:
Data 200%
.
.
Tabela8:
Data 105%
```

Velikokrat bomo v svojih aplikacijah potrebovali takšno ali drugačno signalizacijo. Ponavadi je že kratek pisk dovolj. Če pa temu pisku lahko še relativno enostavno spreminjamo frekvenco in trajanje, potem je to še toliko bolje. Z ukazom **Sound** *si kar precej olajšamo programiranje.*

Kot logično nadaljevanje je program, ki nam bo služil kot enostaven zvonec. V programu Zvonec5.bas boste opazili, da nisem uporabil nič novega. Uporabil sem ukaze, ki jih že poznamo. Le to je novo, da ima vsaka **subrutina** *dodano* **FOR-NEXT** *zanko, ki jo bom razložil zdajle:*

```
Zvok0:                          'subrutine za razlicne melodije
   Restore Tabela0
   For Pom = 1 To 21
      Read Frekvenca
      Sound P3.7 , Traj , Frekvenca
   Next
```

V **FOR-NEXT** *zanki sem uporabil pomožno spremenljivko* **Pom**, *ki je tipa* **Word** *in nam služi kot števec.* **FOR-NEXT** *zanka namreč ponavlja ukaz ali ukaze, ki so vsebovani v njej. V našem primeru izvajanja* **subrutine Zvok0** *najprej shranimo* **Tabelo0** *v interni spomin, nakar se spremenljivka* **Pom** *postavi na vrednost 1, preberemo spremenljivko* **Frekvenca** *in jo uporabimo v ukazu* **Sound**, *nakar povečamo vrednost spremenljivke* **Pom** *za 1, zopet preberemo spremenljivko* **Frekvenca** *in jo uporabimo v ukazu* **Sound**. *To delamo toliko časa, dokler ni spremenljivka* **Pom = 21** *in s tem je* **FOR-NEXT** *zanka zaključena.*

Prikaz podatkov na LED displeju

V četrtem poglavju bomo postali že kar primerno dobri programerji. Od nekdaj smo si elektroniki želeli prikazovati različne spremenljivke na LED displeju. Bodisi smo želeli izmeriti napetost na napajalniku, ali tok porabe bremena, do tega, da smo recimo želeli izmeriti frekvenco.

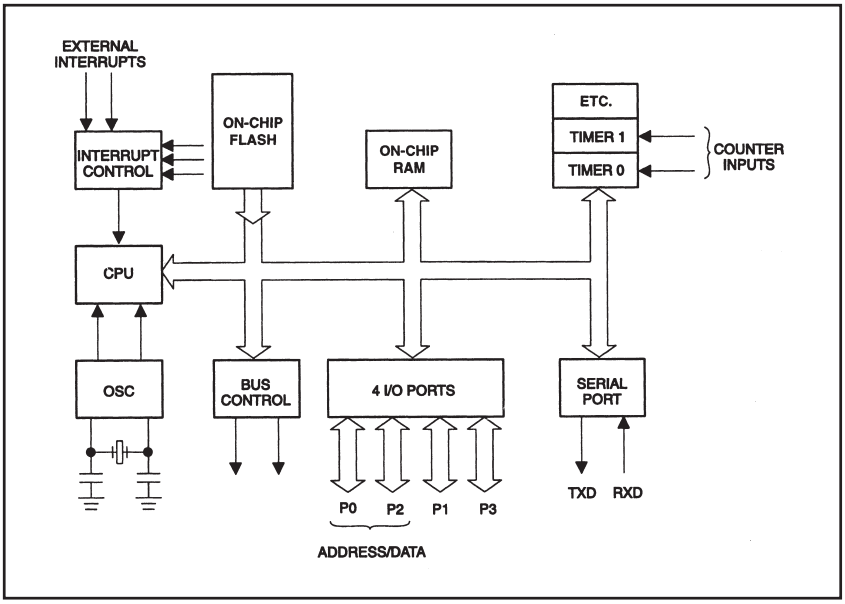
To poglavje nam že približa vse te - do sedaj nedosegljive želje. Mislim, da največ problemov pri začetniku-programerju predstavlja prikaz podatkov na displeju. Zato bom za razlaganje uporabil program **Clock0.bas**, *ki ga je v originalu napisal g.Mirko Pelcl, jaz sem pa ga malo predelal.*

Kot že običajno, se program začne z definicijo spremenljivk.

```
Dim Clock As Byte , Clock1 As Byte , Mux As Byte , Sekunde As Byte , X As Byte
Dim Pomozna_v As Byte , Segmenti As Byte , Enice As Byte , Desetice As Byte
Dim Prikaz As Bit , Izracun As Bit

Config Timer0 = Timer , Gate = Internal , Mode = 2     'konfiguriramo Timer

'Timer0 uporabimo timer 0
'Gate = Internal  brez zunanje prekinitve(interupt)
'Mode = 2         8 bit auto reload
```

Pazljivi bralci seveda niste spregledali nove strukture. To je ukaz **Config**. *Z ukazom* **Config** *določimo, kaj se bo dogajalo s* **Timerjem**. *In kaj sploh je* **Timer***? Poglejmo si še enkrat notranjo blok shemo mikrokontrolerja (slika 2).*

Zgoraj desno v shemi vidimo **Timer1**, **Timer2** *itd. s pripadajočima vhodoma. Beseda* **Timer** *v slovenščini pomeni časovnik. Hm, malo neroden prevod, ki ga bo potrebno malce razložiti. Recimo, da je* **Timer** *števec, ki ga prednastavimo, da šteje do 100. Pri štetju naj uporablja notranji CPU takt. Ko bo preštel do 100, naj nam z enim bitom na izhodu signalizira, da je preštel do 100, nakar naj se ponovno postavi na 0 in šteje ponovno do 100. Tisti en "signalni" bit na izhodu imenujemo* **Interrupt** *ali* **prekinitev** *po slovensko. Zakaj jo imenujemo* **prekinitev***? Zato, ker ima* **Timer**



Slika 2: Blok shema mikrokontrolerja

prioriteto pred ostalim izvajanjem programa. To pomeni, da se bo izvajanje programa prekinilo v trenutku, ko se bo pojavil prekinitveni bit. Program se bo nadaljeval v prekinitveni rutini, jo zaključil in se vrnil v izvajanje prvotnega programa točno tja, kjer se je program prekinil.
Približno tako deluje **Timer**. *Poznamo pa še drugo vrsto delovanja, ko* **Timer** *deluje kot* **Counter** *ali števec. V tem primeru pa* **Counterja** *ne nastavimo do katerega števila naj šteje, ampak mu določimo naj samo šteje in ko bo preštel vseh 16 bitov, kolikor jih ima Counter na voljo (do* 2^{16} *= 65536) naj nam to tudi sporoči z enim bitom na izhodu.* **Counter** *naj šteje impulze, ki jih pripeljemo od zunaj na enega od portov. Tudi* **Counter** *generira prekinitveni bit.*
Tako, približno sem opisal **Timer** *in* **Counter**. *V našem primeru bom uporabili* **Timer**.

Sledi ukaz:

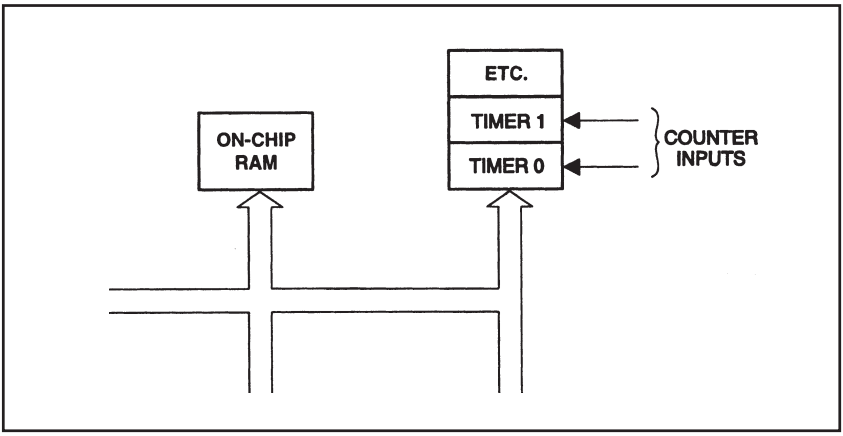
```
On Timer0 Timer_0_int                    'prekinitvena rutina
```

Kar pomeni, da ko bo **Timer0** *preštel do nastavljene vrednosti, naj program vsakič, ko bo* **Timer0** *generiral prekinitveni bit, v vsakem primeru skoči na rutino* **Timer_0_int**.

Nadaljujemo z naložitvijo vrednosti, do katere naj **Timer0** *šteje:*

```
Load Timer0 , 250              'nalozimo v timer0 vrednost 250 µsek
Priority Set Timer0            'prioriteta dolocena timerju0
Enable Interrupts              'omogocimo prekinitve
Enable Timer0                  'omogocimo delovanje timerja0

Start Timer0                   'startamo timer0
```

zdaj, ko smo štartali **Timer0** *si že lahko pogledamo pobližje prekinitveno rutino.*



Slika 3: Blok vezava Timerjev v mikrokontrolerju

V rutini **Timer_0_int** *bomo generirali 1-sekundni impulz. Pa ne približno 1 sekundnega ampak kar se da natančnega.*
Vsaj tako natančnega, kakor je kvarčni kristal lahko natančen. To naredimo tako, da si pomagamo z interno uro mikrokontrolerja. Ta ura ali CPU takt se odvija z 1 MHz, če kot zunanji element uporabimo 12 MHz kvarčni kristal. **Timer0** *smo nastavili na 250. To pomeni, da bo* **Timer0** *signaliziral prekinitveni bit vsakič, ko bo pri štetju CPU ure prešel do 250. Ker je CPU ura 1 MHz, bo prekinitveni bit generiran vsakih 250 mikro sekund.*
Torej moramo imeti 4000 prekinitvenih bitov, da dobimo 1 sekundo (4000 x 250 µs). Nič lažjega torej, samo štejmo te prekinitvene bite in ko prešteje do 4000, vklopimo LED diodo ali povečamo števec sekund ipd. V tem programu smo uporabili dve pomožni spremenljivki **Clock** *in* **Clock1**. *Ker sta obe tipa* **Byte**, *je največja vrednost vsake lahko največ 255. Torej, recimo, prva naj šteje do 20, druga pa do 200. 20 x 200 = 4000.*

Poglejmo si prekinitveno rutino:

```
'prekinitvena rutina
Timer_0_int:

    Incr Clock                      'povecaj Clock za 1
      If Clock > 19 Then            'ko bo Clock vecji od 19
         Clock = 0                  'ga resetiraj na 0

         Prikaz = 1                 'omogoci prikaz

         Incr Clock1                'povecaj Clock1 za 1
            If Clock1 > 199 Then    'ko bo Clock1 vecji od 199
               Clock1 = 0           'ga resetiraj na 0
               P1.7 = Not P1.7      'kontrolni bit, utripne vsako sekundo
               Izracun = 1          'postavi Izracun na 1
               Incr Sekunde
               If Sekunde > 59 Then
                  Sekunde = 0
               End If
            End If
         End If
Return
```

Sekundne impulze zdaj imamo, vse skupaj pa je potrebno prikazati na LED displeju, če želimo imeti kolikor toliko praktično uro. Poglejmo si glavni program:

```
Clock = 0       'dolocimo zacetne vrednosti spremenljivk
Clock1 = 0      'dolocimo zacetne vrednosti spremenljivk
Sekunde = 0     'dolocimo zacetne vrednosti spremenljivk

Do     'zacetek DO-LOOP neskoncne zanke
   If Izracun = 1 Then                  'Izracun = vsako sekundo
      Izracun = 0
'_____________________________________________________________________
'Rutina za izračun desetic in enic
      Desetice = Sekunde / 10
      Pomozna_v = Desetice * 10
      Enice = Sekunde - Pomozna_v
'_____________________________________________________________________
```

```
   End If

   If Prikaz = 1 Then           'prikaz na displeju samo, ko je Prikaz=1
      Prikaz = 0
      P1.0 = 1                  'postavi P1.0 na 1, vklopi tranzistor

'_____________________________________________________________________
'rutina za prikaz enic
        Pomozna_v = Enice
        Gosub Prikaz            'pojdi na subrutino Prikaz
        P1.0 = 0
'_____________________________________________________________________
   End If
Loop                            'konec DO-LOOP zanke
End                             'konec programa
```

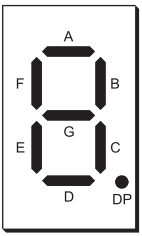
V glavnem programu izračunavamo desetice in enice. To potrebujemo zato, da uporabljamo samo 1 port za prikaz večih številk. Zdajle vam še ni čisto jasno, kmalu pa se vam bo prižgala lučka.... V kolikor ne želimo potratno trošiti I/O porte, jih je potrebno kar se da koristno in smiselno uporabiti. Zakaj bi 100% časa porabili za prikazovanje recimo enic, če pa lahko 50% časa porabimo za prikazovanje enic na prvem displeju, 50% časa pa za prikazovanje desetic na drugem displeju. S tem nismo nič poslabšali prikaza. Vsi namreč vemo, da oči ne zmorejo slediti hitrim spremembam. Zaradi tega sploh lahko spremljamo recimo televizijo in tudi gledamo na računalniški monitor. Če bomo dovolj hitro prikazovali tako enice kot tudi desetice, hkrati pa vklapljali oziroma izklapljali ustrezen LED displej, bomo videli tako desetice kot tudi enice.
Ampak, malo smo začeli prebitevati. Saj še niti enic, niti enega LED displeja ne znamo prižgati! Najprej se naučimo to, potem pa pojdemo naprej.

Poglejmo si subrutino **Prikaz***:*

```
'rutina za prikaz na LED displeju

Prikaz:

Restore Tabela                  'nalozi Tabelo v spomin
   For X = 0 To 9               'od X = 0 do X = 9
      Read Segmenti             'preberi iz Tabele vrednost
                                'in jo preslikaj v spremenljivko
```

```
                                ‘z imenom Segmenti
      If X = Pomozna_v Then     ‘ce je X = Pomozna_v potem
         P3 = Segmenti          ‘prikazi Segmenti na portu P3
         Exit For               ‘konec FOR zanke
      End If
   Next
Return

‘— podatki za pravilen prikaz stevilk na LED displeju ——
Tabela:
Data 128 , 249 , 36 , 48 , 25 , 18 , 2 , 248 , 0 , 16
```

V rutini **Prikaz** *delamo to, da gledamo vrednost spremenljivke* **Pomozna_v** *in za to vrednost prečitamo ustrezno vrednost v* **Tabeli**. *To vrednost potem prenesemo na port P3, kjer je priklopljen LED displej. Recimo, da je spremenljivka* **Pomozna_v** = *3, se bo* **FOR-NEXT** *zanka zavrtela 3x in ko bo X =* **Pomozna_v**, *bo v internem spominu tretja vrednost iz* **Tabele**, *nakar bo program zapusti* **FOR-NEXT** *zanko. Kar moramo poskrbeti je le to, da je vrednosti v Tabeli ustrezajo dejanskim številkam na LED prikazovalniku. Kako izračunamo te vrednosti, bom razložil s pomočjo spodnje tabele. V tabeli imamo označene segmente od A do G plus DP (decimalna pika) in številke od 0 do 9. Kako izračunamo vrednost za številko 1? Najprej moramo ugotoviti, kakšen razpored priključkov ima LED displej. Recimo, da morata za prikaz številke 1 goreti segmenta B in C. Vsi ostali segmenti so ugasnjeni. Nato moramo določiti, na katere priključke porta P3 bomo priključili posamezni segment LED displeja. To naredimo poljubno, kakor nam najbolj ustreza. Recimo, da jih priklopimo takole:*
G=P3.7, DP=P3.6, F=P3.5, E=P3.4, D=P3.3, C=P3.2, B=P3.1, A=P3.0



Slika 4: LED displej

Ker že vemo, da moramo “prižigati” LED diode z logično ničlo, bodo logična stanja za prikaz enice na LED prikazovalniku takšna:

	3.7	3.6	3.5	3.4	3.3	3.2	3.1	3.0	DEC
0	1	1	0	0	0	0	0	0	192
1	1	1	1	1	1	0	0	1	249
2	0	0	1	0	0	1	0	0	36

Dodal sem še primera za številko 0 in 2, zato, da bo razlaga še bolj razumljiva. Kolona DEC pa pomeni decimalno vrednost 8-bitnega števila. Za takšno pretvorbo oziroma preračun potrebujemo malce bolj zmogljiv kalkulator, ki nam bo binarna števila pretvoril v decimalna. Če tega nimamo, lahko to naredimo "peš" po sledečem postopku:

recimo, da bomo pretvorili število 0 iz **Tabele**.

Formula je sledeča:

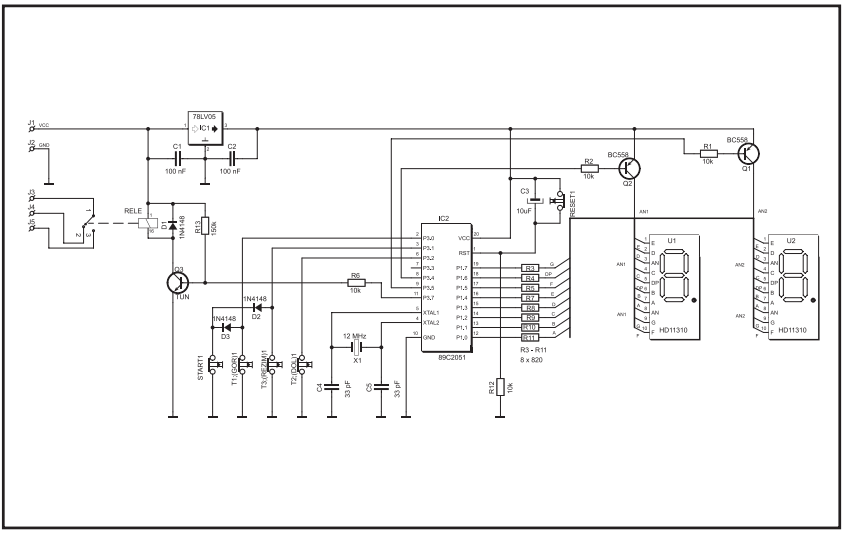
$DEC = P3.0 \times 2^0 + P3.1 \times 2^1 + P3.2 \times 2^2 + P3.3 \times 2^3 + P3.4 \times 2^4 + P3.5 \times 2^5 + P3.6 \times 2^6 + P3.7 \times 2^7$

V formuli bomo namesto P3.0, P3.1, P3.2 itd. vnesli binarno vrednost iz **Tabele**.

$DEC = 0 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 0 \times 2^5 + 1 \times 2^6 + 1 \times 2^7$

$DEC = 64 + 128 = 192$

Ostane vam samo še, da ga prevedete, "zapečete" v mikrokontroler in ga vložite v Bascom testno ploščo. Opazili boste, da dela samo displej za enice. Res je. Za prvič bo kar dobro, če prikazujemo samo na enem displeju. Za prikaz na dveh (ali celo večih) displejih, bo pa potrebno pokukati v zadnje poglavje.



Slika 5: Primer vezave LED displejev

```
'_____________________________________________________________________
'       (c)1997 Mirko Pelcl & Jure Mikeln
'       Slovenija
'_____________________________________________________________________
'       Prikaz z LED displejem
'       Brez Multiplex-a
'       Demo projekt
'_____________________________________________________________________
'Dolocanje tipa spremenljivk
Dim Clock As Byte , Clock1 As Byte , Mux As Byte , Sekunde As Byte , X As Byte
Dim Pomozna_v As Byte , Segmenti As Byte , Enice As Byte , Desetice As Byte
Dim Prikaz As Bit , Izracun As Bit

Config Timer0 = Timer , Gate = Internal , Mode = 2'konfiguriramo Timer

'Timer0             uporabimo timer 0
'Gate = Internal    brez zunanje prekinitve(interupt)
'Mode = 2           8 bit auto reload

On Timer0 Timer_0_int           'prekinitvena rutina
Load Timer0 , 250               'nalozimo v timer0 vrednost 250 µsek
Priority Set Timer0             'prioriteta dolocena timerju0
Enable Interrupts               'omogocimo prekinitve
Enable Timer0                   'omogocimo delovanje timerja0

Start Timer0                    'startamo timer0

Clock = 0                       'dolocimo zacetne vrednosti spremenljivk
Clock1 = 0                      'dolocimo zacetne vrednosti spremenljivk
Sekunde = 0                     'dolocimo zacetne vrednosti spremenljivk

Do                              'zacetek DO-LOOP neskoncne zanke
   If Izracun = 1 Then          'Izracun = vsako sekundo
      Izracun = 0
'_____________________________________________________________________
'Rutina za dolocanje desetic in enic
      Desetice = Sekunde / 10
      Pomozna_v = Desetice * 10
      Enice = Sekunde - Pomozna_v
'_____________________________________________________________________
```

```
   End If

   If Prikaz = 1 Then            'prikaz na displeju samo, ko je Prikaz=1
      Prikaz = 0
      P1.0 = 1                   'postavi P1.0 na 1, vklopi tranzistor

'_____________________________________________________________________

'rutina za prikaz enic
        Pomozna_v = Enice
        Gosub Prikaz             'pojdi na subrutino Prikaz
        P1.0 = 0
'_____________________________________________________________________
   End If
Loop                             'konec DO-LOOP zanke
End                              'konec programa

'_____________________________________________________________________
'prekinitvena rutina

Timer_0_int:

    Incr Clock                   'povecaj Clock za 1
      If Clock > 19 Then         'ko bo Clock vecji od 19
         Clock = 0               'ga resetiraj na 0

         Prikaz = 1              'omogoci prikaz

         Incr Clock1             'povecaj Clock1 za 1
            If Clock1 > 199 Then 'ko bo Clock1 vecji od 199
               Clock1 = 0        'ga resetiraj na 0
               P1.7 = Not P1.7   'kontrolni bit
               Izracun = 1       'postavi Izracun na 1
               Incr Sekunde
               If Sekunde > 59 Then
                  Sekunde = 0
               End If
            End If
         End If
Return
```

```
‘_____________________________________________________________________
‘rutina za prikaz na LED displeju

Prikaz:                                 ‘pazi na napako:dvopicje mora
biti zraven brez presledka
Restore Tabela                          ‘nalozi Tabelo v spomin
   For X = 0 To 9                       ‘od X = 0 do X = 9
      Read Segmenti                     ‘preberi iz Tabele vrednost
                                        ‘in jo preslikaj v spremenljivko
                                        ‘z imenom Segmenti
      If X = Pomozna_v Then             ‘ce je X = pomozna_v potem
         P3 = Segmenti                  ‘prikazi Segmenti na portu P3
         Exit For                       ‘konec FOR zanke
      End If
   Next
Return

‘— podatki za pravilen prikaz stevilk na LED displeju ——
Tabela:
Data 128 , 249 , 36 , 48 , 25 , 18 , 2 , 248 , 0 , 16
```

	A	B	C	D	E	F	G	DP

	3.7	3.6	3.5	3.4	3.3	3.2	3.1	3.0	DEC
0									
1									
2									
3									
4									
5									
6									
7									
8									
9									

Prikaz podatkov z multipleksiranimi LED displeji

Velikokrat imamo opravka s prikazom dveh ali večih številk, bodisi na LCD ali LED prikazovalniku. Videl sem že poizkuse prikaza 4 številk na osmih LED diodah (binarni prikaz), ali celo zvočni prikaz z morsejevo kodo! Kam takšne "kriptične" metode prikazovanja peljejo ne vem. Moram pa reči, da je še vedno daleč najbolj primeren prikaz na LED ali LCD displeju. V zadnjem poglavju bom pokazal, kako več števil prikažemo na LED displeju.

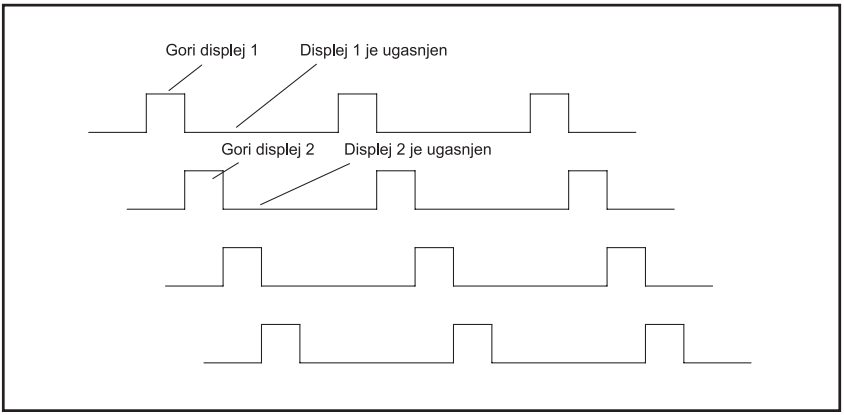
Program, na katerem bom prikazal uporabo, je v originalu napisal Mirko Pelcl. Pravzaprav me je s tem programom "zastrupil" do te mere, da sem se z BascomLT programom začel precej bolj ukvarjati.

Program **Clock1.bas** *je v bistvu identičen programu* **Clock0.bas**, *ki smo ga obdelali v predzadnjem poglavju. S to razliko, da je program* **Clock0.bas** *prikazoval podatke na enem LED displeju, tokrat pa bomo te podatke prikazovali na dveh LED displejih.*

Preden pa začnem, moram pojasniti besedo **MULTIPLEKS**, *ki morda zveni zelo strokovno. V resnici pomeni to, da skupno anodo displejev* **preklapljamo** *ali* **multipleksiramo**. *Torej slovenski izraz bi bil preklapljanje, vendar ne vem zakaj vsi elektroniki uporabljamo izraz* **multipleks**.

Bodi kakor hoče, če hočemo prikazati dve številki na dveh displejih, potrebujemo za to vsaj 2 porta mikrokontrolerja (7 bitov za en displej + 1 bit za decimalno piko (DP) in še 7 bitov za drugi displej + 1 bit za drugo decimalno piko). Skupaj torej dva porta za prikaz dveh številk. Po analogiji bi za prikaz petih številk potrebovali 5 portov! Takšnih mikrokontrolerjev pa nimamo na voljo oziroma, če bi jih že imeli, bi bili kar primerno dragi da ne govorim, kako veliki in okorni bi bili. Pomislite samo, kako ogromni bi bili, če bi morali prikazovati 12 številk! Zato so elektroniki uporabili možgane in si izmislili drugačen prikaz številk: rodil se je multipleksni prikaz. Kaj delamo pri multipleksnemu prikazu? Vsako številko prikazujemo na njenem displeju samo določen čas, ostali čas je displej ugasnjen (slika 6). Ker so ti intervali ko je displej ugasnjen oziroma vklopljen zelo kratki, jih oko sploh ne zazna in pravzaprav dobimo občutek, kot da gorijo prav vsi displeji hkrati. Če torej sekundo razdelimo na desetinke lahko poenostavljeno rečem, da prvo številko prikazujemo v prvih 10 dese-

tinkah sekunde, drugo v drugih 10 desetinkah, tretjo številko prikazujemo tretjih 10 *desetinkah ipd. Ostali čas so displeji ugasnjeni.*



Slika 6: Diagram stanj LED displejev

Poglejmo, kako je to narejeno v BascomLT programu.

Program **Clock1.bas** *se začne z običajnim definiranjem spremenljivk in timerjev. Nakar se začne znana* **Do - Loop** *zanka. Ker v predzadnjem poglavju nisem posebej opisal rutine za prikaz enic in desetic, bom to naredil zdajle.*

V programu najprej pogledamo, če je kontrolna spremenljivka z imenom **Izracun** *enaka 1. Ta spremenljivka nam pove, če se je spremenila vrednost spremenljivk, ki jih prikazujemo. Če se ni, potem ne bomo izgubljali časa v rutinah za preračun enic in desetic. Če je* **Izracun = 1**, *potem gremo naprej v rutino za izračun enic in desetic. Zakaj izračun desetic in enic se boste vprašali? Mikrokontroler vendar ima število, ki ga hočemo prikazati, v svojem spominu! To je res, vendar ste morda pozabili, da mikrokontroler ne operira z desetiškim sistemom. Številka 12 za njega ni 10x1 + 2 ampak je to za njega zaporedje enic in ničel, v tem primeru 1100.*

Zato torej moramo najprej želeno število deliti z 10, da dobimo spremenljivko **Desetice**. *Potem v spremenljivko* **Pomozna_v** *shranimo ravnokar izračunane* **Desetice**, *ki jih pomnožimo z 10 in jih odštejemo od spremenljivke* **Sekunde**. *Naredimo primer s številom 12:*

Desetice: 12 / 10 = 1
Pomozna_v: 1 x 10 = 10
Sekunde: 12 - 10 = 2

Poglejmo si ta del programa:

```
If Izracun = 1 Then                    'Izracun = vsako sekundo
      Izracun = 0
'_____________________________________________________________________
'Rutina za dolocanje desetic in enic
      Desetice = Sekunde / 10
      Pomozna_v = Desetice * 10
      Enice = Sekunde - Pomozna_v
'_____________________________________________________________________
   End If
```

Ves ostali program **Clock1.bas** *je identičen programu iz prejšnjega poglavja. Le, da smo dodali nekaj rutin: V prekinitveni rutini* **Timer_0_int** *smo dodali števec od 0 do 3, kjer je uporabljena spremenljivka* **Mux** *in kontrolna spremenljivka* **Prikaz**. *Dodali pa smo tudi rutino za prikaz desetic.*

```
Incr Mux                        'povecaj Mux za 1
         If Mux > 3 Then        'ko bo Mux vecji od 3
            Mux = 0             'ga resetiraj na 0
         End If
         Prikaz = 1             'omogoci prikaz
```

Rutine štetja do 3 verjetno ni potrebno posebej razlagati. Bolj pomembno je, zakaj potrebujemo to rutino. S to rutino kontroliramo, kdaj bo točno določen displej prižgan: ko je **Mux = 2**, *potem prikazujemo desetice, ko je* **Mux** *= 0, prikazujemo enice. Tu bi za prikaz večih številk lahko dodali števec do recimo 6 in bi lahko prikazovali do 6 števil ipd.*

Zdaj vam bo tudi jasna rutina za prikaz desetic:

```
'_____________________________________________________________________
'rutina za prikaz desetic
      If Mux = 2 Then
         Pomozna_v = Desetice
         'P1.0 = 1       'v tem primeru ni nujno, da ugasnemo ta segment
         Gosub Prikaz    'pojdi na subrutino Prikaz
         P1.3 = 0
      End If
'_____________________________________________________________________
```

Ko je **Mux = 2**, *v spremenljivko* **Pomozna_v** *vpišemo vrednost za* **Desetice**, *prižgemo ustrezen bit za vklop skupne anode (P1.0 = 1), nakar skočimo v subrutino* **Prikaz**, *kjer na displeju prikažemo vrednost spremenljivke* **Pomozna_v**. *Potem, ko to naredimo, se vklopi prikaz na displeju za enice. V spremenljivko* **Pomozna_v** *vpišemo vrednost za* **Enice** *in ponovimo postopek s to razliko, da vklopimo P1.3 za skupno anodo displeja enic.*

In tako nam bo naša rutina prikazovala enice in desetice. Če se vam zdi, da displeji utripajo, spremenite vrednosti števca v rutini **Timer_0_int**, *kjer* **Mux** *šteje do 3. Pravzaprav se zdaj lahko igrate po mili volji, saj v glavnem veste, kaj počenjate. Verjetno boste želeli razširiti prikaz na vsaj 3 številke, vendar pa se boste morali že malo potruditi in to sami realizirati. S primeri, ki sem jih razložil bo verjetno šlo kar hitro In enostavno.*

Za konec vas pa vabim, da si pogledate, kako je prikaz na LED prikazovalniku izvedel naš dolgoletni sodelavec mag. Vladimir Mitrović v programu za digitalni kontroler.

Rutina za **Lookup**

```
Display:                          ' ***  prikaz na displeju  ***
      Pom = P3 And &B1111         ' zajemi stanje števca
      P1 = Lookup(pom , Segmenti)  ' in ažuriraj izpis na displeju
Return
Segmenti: 'tabela vrednosti za 8-segmentni LED displej (prikaz 1-16)
Data &B11111001, &B10100100, &B10110000, &B10011001, &B10010010,
&B10000010
Data &B11111000, &B10000000, &B10010000, &B01000000, &B01111001,
&B00100100
Data &B00110000, &B00011001, &B00010010, &B00000010
```

Mag. Mitrović je za prikaz na LED displeju uporabil ukaz **Lookup**, *ki ga v naših izvajanjih nisem omenil. Pravzaprav moram priznati, da je njegov način prikaza na LED prikazovalniku še precej enostavnejši kot tisti, ki sem ga razložil v zadnjih dveh poglavjih. Preglednost programa z uporabo* **Lookup** *ukaza je tudi precej boljša. Tudi ukaz za zaznavanje pritisnjenosti tipke je drugačen od tistih, ki smo jih poznali do sedaj.*

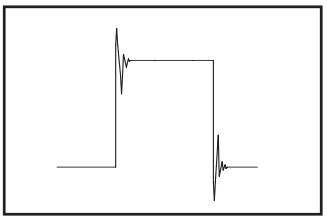
*Pri prikazu na displeju ima ključni pomen tabela "***Segmenti***" . v njej so v binarni obliki (znak* **&B** *pomeni, da je podatek v binarni obliki) zapisane vrednosti segmen-*

tov a-i (opomba: v primeru mag.Mitrovića je uporabljen 1+1/2 displej, glej S.E. št.48, stran 40). Tabelo beremo s pomočjo **Lookup** *funkcije za razliko od ukaza* **Read**, *s katerim beremo tabelo podatek za podatkom, omogoča* **Lookup** *neposreden pristop do iskanega podatka. Če je vrednost spremenljivke* **Pom = 7**, *bo ukaz* **Lookup** *prebral 8. podatek iz tabele in ga prenesel na port P1 ter na ta način aktiviral prikazovalnik. Ali bi lahko bilo še bolj preprosto?*

Še nekaj besed o tabeli **Segmenti**: *v njej je vpisanih 32 podatkov, čeprav lahko* **Lookup** *doseže samo prvih 16 (Pom = 0 do 15). Teh prvih 16 vpisanih vrednosti dekodira stanje števca "0000" do "1111" v prikaz od "1" do "16". Če želite, da bo števec štel od 0 do 15, potem uporabite tiste podatke, ki so ustrezno označeni v tabeli* **Segmenti**.

Mag. Mitrović je za zaznavanje pritisnjenosti tipke uporabil ukaz **Debounce** *in prav je tako, da mikrokontroler tipko zazna malo bolj natančno, kot smo to počeli do sedaj. Zakaj? Tipke imajo ob pritisku t.i. prehodni pojav, kar pomeni, da napetost na priključku malo zaniha ob preklopu iz 0 v 1 in obratno.*

Seveda takšne prenihaje zazna tudi naš mikrokontroler in nam utegne signalizirati pritisnjeno tipko takrat, ko tipka sploh ni pritisnjena. Zato je mag.Mitrović vsvojem programu uporabil ukaz **Debounce**, *ki je že izdelana rutina za zaznavanje pritisnjenosti tipke.*



Slika 7: slika prehodnega pojava

```
Debounce P3.4 , 0 , Up , Sub      ' Tipka UP pritisnjena? Izvrši subrutino UP!
Debounce P3.5 , 0 , Down , Sub    ' Tipka DOWN pritisnjena? Izvrši subrutino DOWN!
```

Pravilna rutina za tipke deluje približno takole: mikrokontroler v trenutku pritiska tipke zazna, da se je nekaj dogodilo na portu, kjer je priklopljena tipka. Ker mikrokontroler seveda ne ve, ali je bil ta signal samo motnja ali je se je res dogodil pritisk tipke, čez nekaj desetink sekunde še enkrat preveri, ali je tipka res pritisnjena.

Če dobi signal, da je tipka pritisnjena, potem izvede rutino za pritisnjeno tipko, drugače pa mikrokontroler ta signal ignorira. S tem se znebimo "lažnih" pritiskov tipk in

seveda tudi motenj. V naših primerih sem se namenoma izognil tem problemom, ki nas čakajo v svetu mikrokontrolerjev. Tudi naši primeri namenoma niso bili kritični glede pritiska tipke, nekaj povsem drugega pa pomeni napačen signal recimo pri krmiljenju strojev, kjer lahko napačen signal povzroči veliko škodo. V takšnih primerih pa moramo možnost napake zmanjšati na minimum.

Namesto konca

Prav pri koncu pa ne morem mimo zelo elegantne rešitve naslednjega problema: ko za kratek čas pritisnemo tipko "UP", želimo, da se števec povečuje, če kratko pritisnemo tipko "DOWN" želimo, da se števec zmanjšuje. Bolj napredni uporabniki pa velikokrat želijo, da števec hitreje šteje (gor ali dol). Za izvedbo te funkcije, recimo ji "FAST" imamo na voljo dve rešitvi: ena je, da dodamo tipko "FAST". Ko pritisnemo tipko "UP", števec šteje navzgor po določenih korakih. Če hkrati pritisnemo še tipko "FAST", bo števec štel hitreje. Vendar je s stališča porabe portov in tudi tipk to neracionalna rešitev. Mag.Mitrović je svojo rešitev naredil kar se da elegantno: če dalj časa (več kot 1 sekundo) držimo katero koli tipko, bo števec začel hitreje šteti.

```
Up:                                 ' ***  ***  Štetje navzgor ***  ***

   Do                               'To počni dokler je tipka UP
                                    'pritisnjena
      Set P3.7                      'postavi MUTE
      Pom = P3 And &B1111           'preberi stanje števca
      If Pom < 15 Then
         Incr P3                    'in ga povečaj za 1
      Else                          'ali nadaljuj šteti od 0
         P3 = P3 And &B11110000     'če je števec bil na 15
         Set P3.4
      End If
      Gosub Display                 'ažuriraj prikaz na displaju
      For X = 1 To Koliko Step Korak 'zadrži stanje števca določen čas
         If P3.4 = 1 Then           'če je bila tipka UP medtem sproščena
            Exit For                'takoj nadaljuj
         End If
      Next X
      Korak = 3                     'skrajšaj trajanje naslednjega impulza
   Loop Until P3.4 = 1              'če je tipka pritisnjena ponovi
```

BASCOM LT BASic COMpiler za mikrokontrolerje družine 8051

Pregled ukazov

BASCOM BASIC je zelo podoben Microsoft-ovemu QBASIC-u, saj sta 99% kompatibilna, poleg tega pa BASCOM s posebnimi ukazi podpira standardne LCD prikazovalnike in I²C serijski protokol, ki ga razumejo najrazličnejša zanimiva integrirana vezja (npr. AD/DA pretvorniki, ura realnega časa, PLL sintetizatorji, port expanderji...), Dallasov 1Wire protokol je tudi zanimiv zaradi Dallasovih komponent kot je DS1820 – senzor temperature z integriranim A/D pretvornikom.

V Tabeli 2 vidimo večino razpoložljivih ukazov BASCOM prevajalnika. Omogočajo nam pregledno, strukturirano programiranje z ukazi IF-THEN-ELSE-END IF, DO-LOOP ali WHILE-WEND. Ker se program še vedno dograjuje, ta seznam ni dokončen in lahko pričakujemo še več novosti. Vsak kupec programa je seveda upravičen do brezplačnega update-a, ki ga lahko naloži kar preko interneta!
serijskega kanala

Spremenljivke in označbe programskih delov (labele) so lahko dolžine do 32 znakov. Uporabljamo lahko BIT, BYTE ali INTEGER spremenljivke. Poleg tega lahko s posebnimi ukazi uporabljamo interno periferijo 8051 krmilnikov, kot so TIMER, COUNTER ali INTERRUPT. Dosegamo lahko tudi direktno do vsebine internih registrov krmilnika ali pa v program vključimo tudi del v strojni kodi oz. asemblerju.

UKAZ	Pomen
Deklaracije	
DIM var AS type	Definicija spremenljivke
DIM symbolAS CONST value	Definicija konstante
BIT, BYTE, INTEGER	Tipi podatkov
DEFBIT, DEFBYTE, DEFINT	Default deklaracija spremenljivk
Logične in aritmetične operacije	
AND, MOD, NOT, OR, XOR	
DEC var, INC var	Zmanjšanje, povečanje za 1
SET bit, RESET bit	Postavi ali briše vrednost bita
ROTATE var, LEFT / RIGHT	Rotiranje bitov neke spremenljivke v levo/desno
SWAP var1, var2	Zamenjava vrednosti dveh spremenljivk
GETDATA, SETDATA	Vmesni pomnilnik
Pretvorbe	
CHR, BCD	Pretvorba iz binarne vrednosti v Character ali BCD
MAKEBCD, MAKEDEC	Pretvorba med BCD in decimalno vrednostjo

UKAZ	Pomen
Ukazi za tvorbo programske strukture	
DO ..stavki..	DO zanka
LOOP UNTIL izraz	Zanka
FOR var=start TO/DOWNTO end ...stavki... NEXT	FOR-NEXT zanka
WHILE izraz .. stavki.. WEND	WHILE zanka
EXIT FOR/DO/WHILE	Neposredni izhod iz zank
IF izraz THEN stavki	Pogojni skoki
CALL (do 9 parametrov)	Klic rutin s prenosom parametrov
GOSUB, RETURN	Klic podprograma in vrnitev
GOTO	Skok na označbo
ON var GOTO/GOSUB	Vejitev glede na vrednost spremenljivke
STOP, END	Zaustavitev, konec programa
Vhodno-izhodni stavki	
INKEY	Prinese ASCII vrednost s serijskega kanala
INPUT ("tekst"), var	
INPUTHEX("tekst"), var	Vnos preko RS-232
PRINT var; "konstanta"	
PRINTHEX var	Izhod preko RS-232
Prekinitve (interrupt)	
ON INTx, SERIAL, TIMERx	Prekinitve npr.: ON TIMER0 prekinitvena_rutina Skok na prek. rutino, ko nastopi TIMER0
PRIORITY, DISABLE, ENABLE	Kontrola prekinitev
Časovnik/števec	
LOAD timer,vrednost	Inicializacija časovnika
START timer, STOP timer	Zagon, zaustavitev časovnika
COUNTER	Šteje impulze na P3.4
Splošno	
REM	Komentar
SOUND pin,trajanje,frekvenca	Generiranje enostavnih tonov
DELAY	Kratkočasna zakasnitev
WAIT sekunde, WAIT ms	Zakasnitev poljubne dolžine
BITWAIT x, SET/RESET	Čakanje, dokler nek bit ni postavljen/zbrisan
Mode - načini delovanja CPU	
IDLE, POWERDOWN	Postavitev mikrokontrolerja v stanje manjše porabe
I²C-bus ukazi	
I2CRECEIVE,I2CSEND	Sprejem, pošiljanje podatkov neki I2C enoti
I2 CSTART, I2CSTOP	Start oz. STOP pogoj na I2C vodilu
I2CRBYTE, I2CWBYTE	Sprejem, pošiljanje enega zloga
Ukazi za prikaz na LCD	
CLS	Brisanje displeja in kurzor v začetni položaj
CURSOR ON/OFF	Krmiljenje kurzorja in displeja
BLINK/NOBLINK	
HOME UPPER/LOWER	
DISPLAY ON/OFF	
LCD var, LCDHEX var	Zapis vrednosti spremenljivke na LCD
LOCATE y,x	Pozicioniranje kurzorja
LOWERLINE, UPPERLINE	
SHIFTLCD LEFT/RIGHT	Premikanje vsebine LCD-ja v levo/desno
SHIFTCURSOR LEFT/RIGHT	Premikanje kurzorja levo/desno

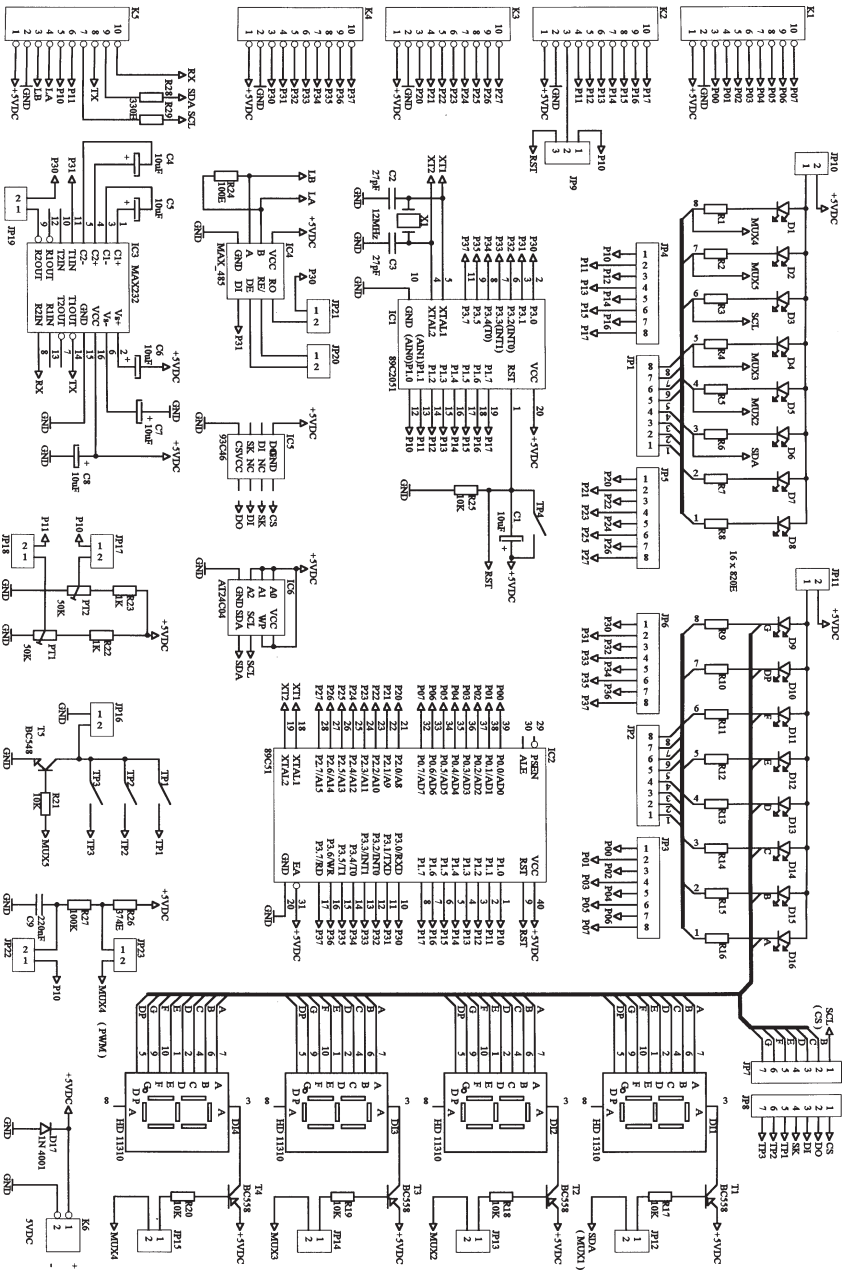
Tabela 2: Seznam ukazov BASIC prevajalnika BASCOM LT

Bascom – Testna plošča za družino 8051

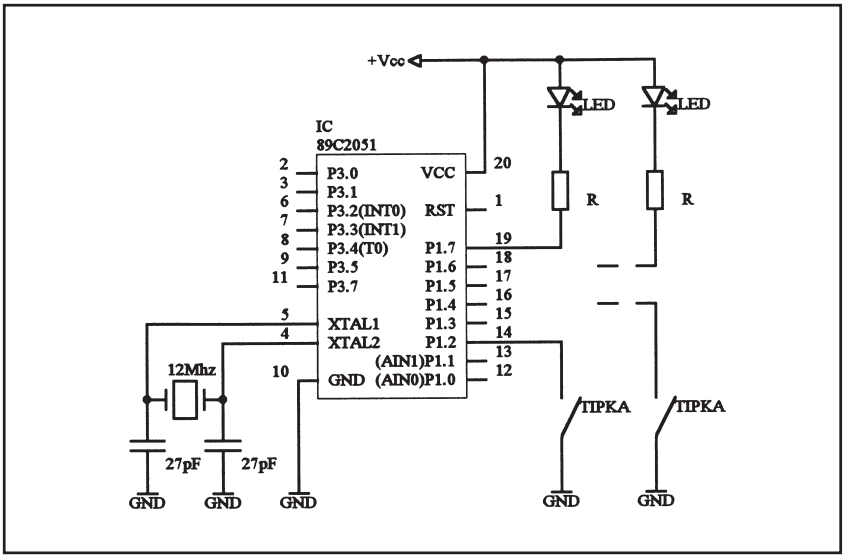
Delo in programiranje z BASCOM LITE prevajalnikom bo lažje, če bomo imeli univerzalno preizkuševalno mesto. V tem poglavju vam bomo predstavili Bascom Testno ploščo.

Pri pisanju programov je dobro vedeti tudi nekaj o zgradbi mikrokontrolerja. To smo že na kratko predstavili na začetku tega priročnika. Lep opis zgradbe 8051 je tudi bil objavljen v prvih številkah revije Svet elektronike in bo objavljen v posebnem priročniku za mikrokontrolerje. Tisti, ki vas to zanima, pokličite v uredništvo revije Svet Elektronike.

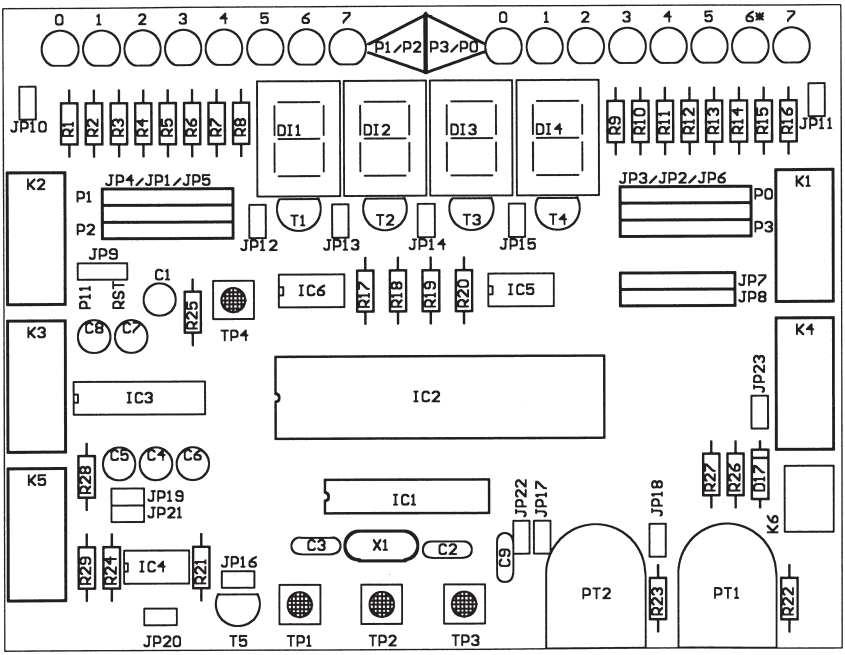
Nadalje si je potrebno priskrbeti nekaj osnovne opreme za delo. Med prvimi je prav gotovo programator. Poceni rešitev je PG302, ki je bil objavljen v reviji in bo opisan v nadaljevanju tega priročnika. Lahko pa seveda uporabite kateri koli drugi programator za MCS51 mikrokontrolerje. Drugi koristen pripomoček je razvojni sistem, na katerem preizkušamo programske rešitve. Sam BascomLT program ima v svoji dokumentaciji opisan sistem za preizkušanje. Tukaj pa vam predstavljamo nekoliko kompleksnejši razvojni sistem, namenjenega "malim" in "velikim" mikrokontrolerjem. Iz Atmelove družine so "mali" tisti z dvajsetimi nožicami: AT89C2051, AT90S1300, AT90S2312. Veliki pa imajo enkrat več nožic AT89C51, AT89C52.. ta družina je tudi nekoliko večja. Razvojni sistem je izdelan tako, da podpira čim več funkcij, ki so navadno zajete pri mikrokontrolerskih napravah. Prikaz, tipke, komunikacija.... Seveda si lahko vsak sam izdela sistem, kot ga želi imeti. Vsi porti mikrokontrolerja so dostopni na konektorjih K1 do K4, kar omogoča razširitev in priklop zunanjih enot, npr. LCD. Majhna izjema je K2. S pomočjo kratkostičnika (jumperja) lahko njegov pin št. 3 preklopimo med P10 in RST. Preklop nam bo omogočil, da lahko preko K2 programiramo procesorje z SPI vodilom. Na K5 so priklopljeni signali za komunikacijo. RX in TX za RS232, LA in LB za RS485, ter SDA in SCL za I2 C. Poleg tega sta na K5 dodatno dostopna P10 in P11, ki ju bomo kasneje uporabili za analogni vhod in analogni izhod. Za A/D konverzijo bodo skrbeli R26, R27 in C9. Analogno meritev bomo izvedli kot meritev napetosti na PT1, omenjeni trimer in PT2 služita tudi kot vhoda komparatorja pri "malih" procesorjih. Kratkostičniki so namenjeni konfiguraciji hardverja. Razvojni sistem vseh funkcij, ki jih omogoča, ne more opravljati hkrati oziroma v celoti, predvsem zaradi tega, ker je vso delo naloženu portoma P1 in P3. Omenjena rešitev je narejena zaradi lažje uporabe enakega programa v "malih" in "velikih" mikrokontrolerjih. Kljub temu pa nobeden pin mikrokontrolerja ni trajno



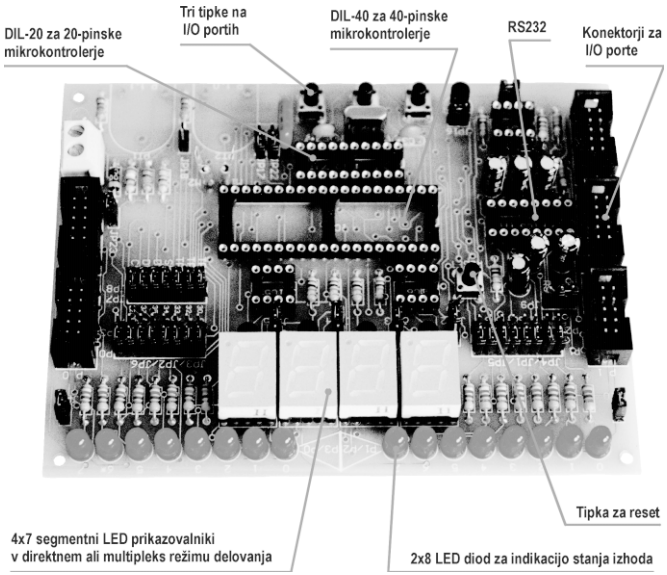
Slika 8: Električna shema Bascom Testne plošče



Slika 9: Shema priključitve LED diod in tipk



Slika 10: Montažna shema Bascom Testne plošče



vezan na določen hardver. To nam omogoča, da lahko hardverske povezave s pomočjo žičk izvedemo kakorkoli želimo. Stanje pinov na portih spremljamo s pomočjo LED diod. Ko uporabljamo velike mikrokontrolerje lahko LED-ice D1 do D8 preklapljamo tako, da prikazujejo stanja na P2 ali P1. Celo skupino pa vključimo s kratkostičnikom JP10. Enak način velja za LED-ice D9 do D16, ki služijo za porta P0 in P3. Na zadnjo skupino so priključeni štirje LED displeji. Posamezne displeje vključimo s kratkostičniki JP12 do JP15. Predvidene tipke TP1, TP2 in TP3 so povezane v multipleks (MUX5). To bi na začetku verjetno komu povzročalo težave pri pisanju programa, zato se s pomočjo JP16 lahko izključijo iz multipleksa. TP4 služi za reset (RST) mikrokontrolerja. Sistem podpira dve najbolj razširjeni različici EEPROMA, 93C46 in 24C04 z I2 C vodilom, lahko se uprabi tudi kakšen od bližnjih sorodnikov prvega ali drugega. Zaradi hardverskih povezav se ne moreta uporabljati oba hkrati. Preden napišemo prvi program, še nekaj o tem, kako smemo priključiti pin porta. Velja za družino 8051. Ko je na pinu stanje logične enice lahko daje tok le nekaj µA, če ga obremenimo napetost na njem pade. (Ioh = -30µA, Voh = 0.75*Vcc minimalna vrednost. Atmelovi podatki za AT89C2051) Takšen pin lahko brez posledic staknemo z maso. Tako tipamo stanja zunanjih enot: tipke, razni digitalni vhodi..... Drugače je, ko je pin na nivoju logične ničle, takrat lahko teče precej večji tok. (Iol = 20mA, Vcc = 5V, Vol = 0.5V maksimalno, Atmel za AT89C2051) Paziti moramo, da mu ne vsiljujemo visokega nivoja, ker bomo iz mikrokontrolerja naredili pokojni mikrokontroler. Če bremenu omejimo tok, tako da ne presega Iol maksimalnega, ga lahko zvežemo direktno med napajalno napetost in pin mikrokontrolerja.

PROGRAMATOR PG302 ZA PROGRAMIRANJE ČLANOV DRUŽINE 89C51 IN AVR

Prišli so novi mikrokontroleri in našemu staremu programatorju je pošla sapa. Zaradi tega smo poiskali novo poceni možnost programiranja mikrokontrolerjev družine 8051. Pričujoči programator je v osnovi namenjen programiranju popularnih malih Atmelovih mikrokontrolerjev serije 89C1051 in 89C2051 v DIL 20 ohišju, takoj za tem pa novim RISC mikrokontrolerjem AVR serije 90SXXXX.

Seveda zna programator PG302 programirati poleg ATMELovih mikrokontrolerjev tudi mikrokontrolerje proizvajalcev AMD, Dallas, Intel in Philips.

Koncept

Resnejši razvijalci programske opreme za mikrokontrolerje uporabljajo pri programiranju emulatorje. Pri tem lahko sledijo toku podatkov, preverjajo lahko stanja akumulatorjev in ostalih registrov, postavljajo lahko tudi prekinitvene točke. Vse to jim pomaga pri hitrem pisanju kode in kar je najbolj važno, pri razhroščevanju kode. Ker pa je naš programator prvenstveno namenjen mikroprocesorjem z majhnim pomnilnikom (89C1051 ima le 1Kb) emulatorja mikrokontrolerov niti ne potrebujemo, saj je naša koda zelo kratka in lahko sledimo toku podatkov tudi na zaslonu ali na listingu programa.

Z metodo poizkušanja lahko program kar hitro razhroščimo, nakar ga še zadnjič sprogramiramo, zaklenemo ***lock* bite** in ga vstavimo v naše ciljno vezje.
Še enostavnejša stvar je pri programiranju AVR mikrokontrolerov z SPI vodilom, preko katerega lahko procesor vpišemo kar v ciljnem sistemu in to lahko storimo do 10.000-krat.

Programator PG302 sodi v razred amaterskih in polprofesionalnih programatorjev in je primeren tako za domače delo, kot za programiranje manjših serij mikrokontrolerov. Programira precejšen del družine mikrokontrolerjev 80C51, odlikuje pa ga tudi dovolj nizka cena. Kljub temu pa programira zanesljivo in hitro, uporabniški vmesnik na PCju pa deluje v prijaznem Windows okolju.

ELEKTRONIKA

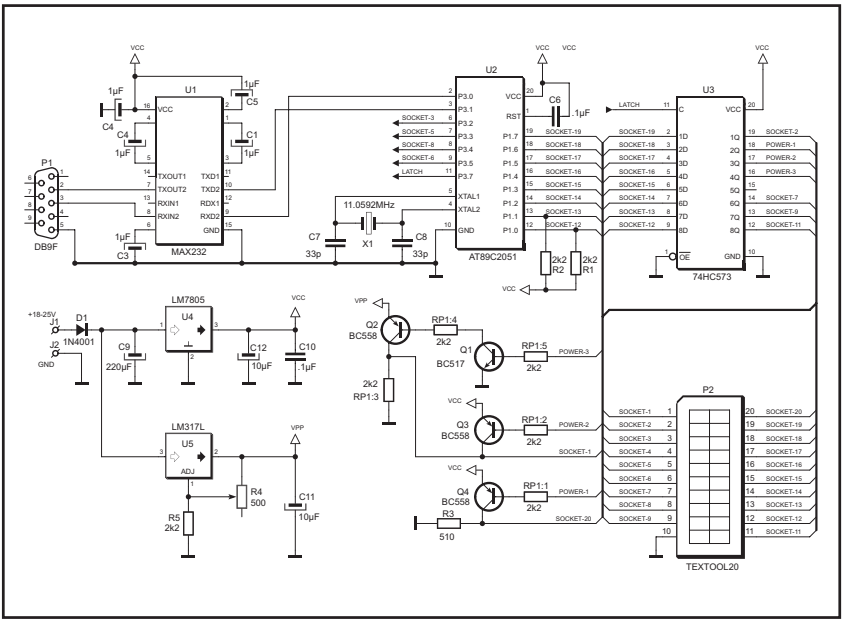
Električna shema

Programator je pravzaprav enostavna naprava. Datoteko s strojno kodo mora pač vpisati v izbrani mikrokontroler na enega od dveh možnih načinov: normalno preko porta P1 in P3 (za običajne krmilnike s Flash ali PROM pomnilnikom) ali preko SPI vodila (AVR-ji).

Glavna komponenta programatorja je mikrokontroler Atmel 89C2051, ki deluje s taktom 11,0592MHz. Mikrokontroler skrbi za to, da pravzame kodo iz PC-ja in jo po določenem programirnem algoritmu shrani v pomnilnik mikrokontrolerja, ki ga programiramo. PG302 je na PC povezan po navadnem serijskem kanalu RS232. Zato je uporabljen MAX232ACPE, znani pretvornik nivojev TTL/±9V, ki uporablja kondenzatorje vrednosti 1µF, napajamo pa ga le s petimi volti.

Ker ima mali AT89C2051 premalo portov za programiranje vstavljenega mikrokontrolerja, je za preklapljanje med programiranjem in normalnim delovanjem ter vključevanje delovne napetosti uporabljen dodaten *latch*, ki "demultipleksira" port P1. Kot *Chip select* za 74HC573 je uporabljen izhod P3.7 (LATCH).

Programator napajate z napetostjo od +18 do +25Vdc max. Uporabimo lahko kar standardni adapter-napajalnik, ki ga lahko kupimo za nekaj 100 tolarjev v skoraj



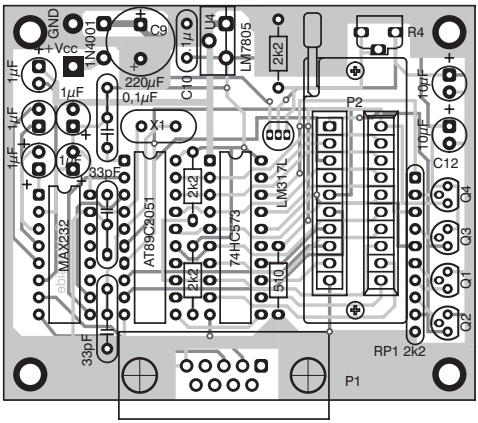
Slika 11: Električna shema programatorja PG302

vsaki trgovini z elektromaterialom. Napajalnik mora zagotoviti vsaj 200mA ali več toka, da bo programator pravilno deloval.
Napajalni del mora zagotoviti delovno napetost V_{cc} in programirno napetost V_{pp}. Za stabilizacijo Vcc je uporabljen klasični 5-voltni regulator LM7805. Zaradi precejšnje disipacije je uporabljen 1-amperski. Programirno napetost nastavimo z regulatorjem LM317LZA. To je nastavljiv regulator v plastičnem ohišju, ki zagotavlja cca 100mA toka, kar pa povsem zadošča. Napetost lahko nastavljamo s trimer potenciomatrom vrednosti 500 ohmov, lahko pa vgradimo kar navaden upor, ki ga prej umerimo.
Mikrokontroler IC1 preklaplja med programirno in reset napetostjo s signali Power_2 in Power_3 z vključevanjem tranzistorja Q1 ali Q3. Nizek signal Power_2 pomeni reset napetost, visok signal Power_3 pa programirno napetost.
S signalom Power_1 vključujemo in izključujemo delovno napetost programiranega mikrokontrolerja.

Gradnja in umerjanje

Samogradnja programatorja je enostavna. Po ustaljenem redu najprej prispajkajte vse nizke elemente, nato vse višje, čisto na koncu pa še konektorje. Vendar POZOR! Pred vgradnjo TEXTOOL podnožja moramo najprej preveriti delovanje **po naslednjih točkah:**

- Programator priključimo na napajalno napetost in z voltmetrom pomerite na testnih točkah GND in Vcc. Voltmeter mora pokazati 5V.
- Voltmeter priključimo na točki GND in Vpp. Tokrat moramo nameriti 12.75V. Če temu ni tako, moramo s trimer potenciometrom TP1 nastaviti napetost tako, da bo voltmeter pokazal 12.75V.
- Programator izključimo in dokončamo gradnjo programatorja.



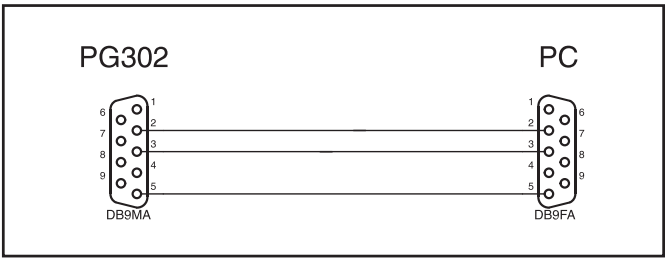
Slika 12: Montažna shema programatorja PG302

Če želite vezje vgraditi v ohišje, je najbolje, da pod TEXTOOL podnožje vtaknete še eno DIL podnožje. S tem boste pridobili na višini. Ročica TEXTOOL podnožja mora namreč gledati iz ohišja tudi takrat, ko je podnožje zaprto. Ker je najvišja komponenta stabilizator, jo lahko prispajkate na spodnji strani tiskanine.

Adapter 220V/18Vdc do 25Vdc lahko vgradite v ohišje programatorja. Če programatorja ne boste vgradili v ohišje, pa je najbolje, da je adapter v solidnem plastičnem ohišju. Že zaradi varnosti!

Izdelava serijskega kabla za povezavo s PC in napajalnega kabla

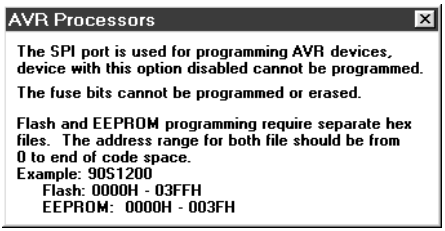
Izdelava kabla je tudi enostavna: po spodnji shemi sestavite dva konektorja DB9, na strani programatorja je moški in na strani PC-ja je ženski. Namesto na DB9 lahko na PC strani namestite konektor DB25. V tem primeru se držite sheme na sliki 12. Uporabljene so le tri žice: TXD, RXD in GND.



Slika 13: Razpored priključkov na serijskem kablu s konektorji DB9

ISP - programiranje mikrokontrolerjev v ciljnem sistemu

Če uporabljate mikrokontrolerje z SPI vodilom, potem tudi praktično ne potrebujete emulatorja. Mikrokontroler lahko vstavimo v tiskano vezje in celo prispajkamo, nato pa ga preko kabla zvežemo z našim programatorjem, tako kot je narisano v spodnji shemi. Jasno pa je, da si morate konektor za ISP programiranje pripravit tudi v ciljnem sistemu in ga seveda povezati na ustrezne priključke mikrokontrolerja!



Slika 14: Opozorilo za ISP programiranje preko SPI vodila. Žal program še ne zna zakleniti lock bitov AVR mikrokontrolerjev.

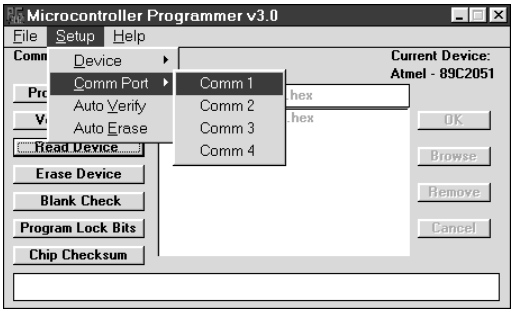
PROGRAMSKA OPREMA

Nameščanje programa v PC

Krmilni program v PC-ju teče v Windows okolju. Deluje tako v Win 3.11 kot tudi v Win95 ali Windows NT operacijskem sistemu. Program za namestitev dobite na 3.5 palčni disketi. Pri instalaciji naložite namestitveni program ***setup3_0.exe*** v svoj direktorij oz. mapo in ga zaženite. Ker je to samo razširitveni program, sam razpakira programe, ki so potrebni za delovanje programatorja. Ko je to narejeno, lahko zaženete program ***Pg302.exe***.

Nastavitve programa (*Setup*)

Ko se odpre okno programa, najprej izberite proizvajalca in vrsto mikrokontrolerja, ki ga želite programirati. To naredite tako, da pritisnete **Setup, Device** in izberete proizvajalca ter nato še ustrezni mikrokontroler.



Slika 15: Nastavitev komunikacijskih vrat

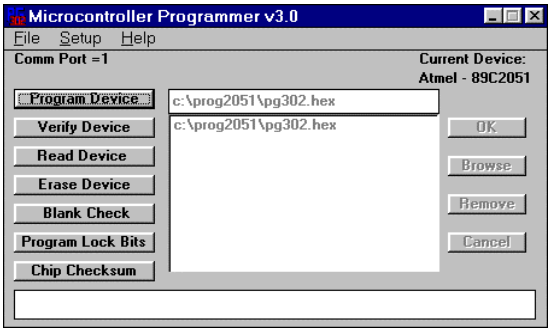
Nadalje svetujemo, da si v menuju **Setup** odkljukate **Auto Verify** in **Auto Erase**. Tako bo programiranje teklo tekoče. Lahko tudi poizkusite drugače, vendar se utegne vaš računalnik v ekstremnih primerih tudi "obesiti".

Programator preko zgoraj opisanega standardnega kabla za serijsko komunikacijo priklopimo na eno od prostih serijskih komunikacijskih vrat. Nato morate določiti komunikacijska vrata (slika 15), preko katerih bo potekalo programiranje. V kolikor ne veste, na katerih komunikacijskih vratih je vaš programator priklopljen, lahko poizkušate tako, da v meniju **Setup.Comms** najprej izberete **Comm1**, izpraznete TEXTOOL podnožje programatorja, priklopite napajalno napetost in pritisnite **Blank Check** ter počakate na odgovor programatorja. V kolikor bo odgovor "**Programmer not responding**" zamenjajte komunikacijska vrata v meniju **Setup.Comms** ali celo preklopite serijski kabel na druga komunikacijska vrata vašega računalnika. To počenjajte toliko časa da dobite odgovor "**Device Blank**".

Mikrokontroler vložimo v TEXTOOL podnožje programatorja. Mikrokontroler naj bo obrnjen tako, da ima nogico 1 pri ročici TEXTOOL podnožja oziroma vsa integrirana vezja na programatorju naj imajo nogico 1 na isti strani!.

Programiranje

Sedaj že lahko programirate mikrokontrolerje, jih berete in shranite njihovo vsebino v datoteko. Seveda ne morete prebrati vsebine mikrokontrolerja, ki so ga pri programiranju "**zaklenili**" s t.i. **Lock biti**.

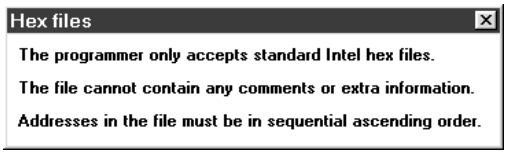


Slika 16: Programiranje mikrokontrolerja

Samo programiranje poteka tako, da najprej z ukazom "**Browse**" poiščemo, kje se nahaja datoteka, ki jo želimo programirati. Ko jo najdemo, jo kliknemo z miško in še enkrat kliknemo na OK. Program bo v primeru programiranja v spodnjem oknu napisal, kakšen je status programiranja.

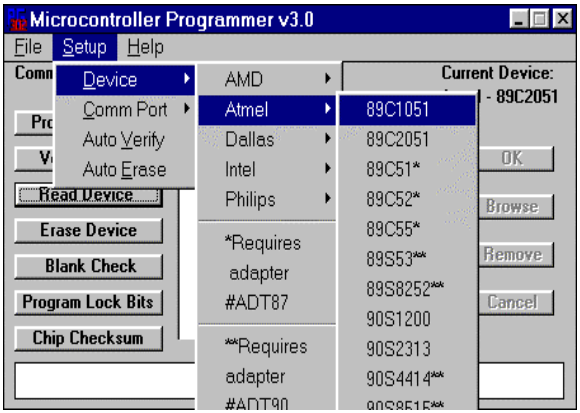
Po uspešnem programiranju lahko zaklenemo **Lock bite** tako, da pritisnemo "**Program Lock Bits**" in z miško odkljukamo želene **Lock bite**.

Pri programiranju moramo upoštevati, da programator programira **SAMO** Intelovo HEX kodo! Vse druge oblike programske kode bodo napačno sprogramirane in mikrokontroler ne bo deloval po programu. Pri programiranju morate paziti, da ni hex koda prevelika glede na spominski prostor mikrokontrolerja. Predolga koda za spominski prostor določenega mikrokontrolerja bo pomenila, da bo programator pri preverjanju vsebine javil napako. Enako velja za EEPROM pri AVR procesorjih.



Slika 17: Opozorilo pred napačnim formatom hex datoteke

Jasno je, da se mora oznaka mikrokontrolerja, ki ga želimo programirati ujemati z nastavljeno oznako mikrokontrolerja na programatorju. Tako bo programiranje npr. 89C2051 neuspešno, če je programator nastavljen na 89C51 in podobno.



Slika 18: Izbira procesorjev, ki jih želimo programirati

Za programiranje 40 pinskih mikrokontrolerjev je potrebno dodatno podnožje-adapter. V meniju ***Setup.Device*** so mikrokontrolerji brez oznake programirljivi brez dodatnih adapterjev, tisti označeni z eno zvezdico (*) se programirajo v adapterju označenim z **#ADT87**, tisti označeni z dvema zvezdicama (**) pa se programirajo v adapterju označenem z **#ADT90**. Na voljo pa je tudi ISP programirni kabel, s katerim lahko programiramo AVR mikrokotrolerje kar v končnem vezju.

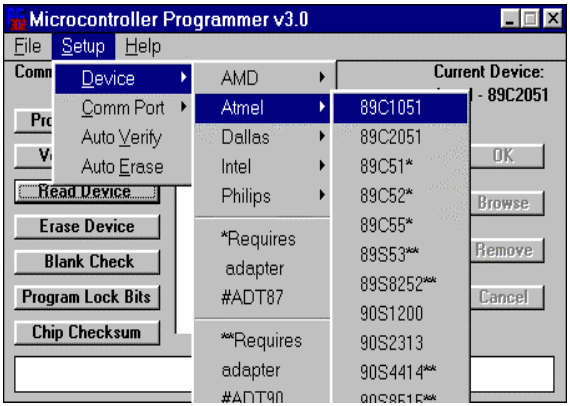
Kosov	Opis	Vrednost	Oznaka
5	Kondenzator elektrolitski	1µF	C1, C2, C3, C4, C5
2	Kondenzator keramični	33pF	C6, C7
2	Kondenzator multilayer	0.1µF	C8, C10
1	Kondenzator elektrolitski	220µF	C9
2	Kondenzator elektrolitski	10µF	C11, C12
1	DC Power Jack		P1
1	Konektor ženski kotni	DB9FA	P2
1	Podnožje za DIL20	TEXTOOL20	P3
1	Tranzistor univerzalni NPN	BC517	Q1
3	Tranzistor univerzalni PNP	BC558	Q2, Q3, Q4
1	Pretvornik TTL/RS232	MAX232ACPE	U1
1	Mikrokrmilnik s programom	AT89C2051	U2
1	Osemkratni D-Flip Flop (Latch)	74HC573	U3
1	Stabilizator 5V	LM7805	U4
1	Nastavljiv stabilizator	LM317L	U5
1	Kristal	11.0592 MHz	X1
3	Upor 1/4 W	2k2	R1, R2, R5
1	Upor 1/4 W	510	R3
1	Upor multiturn	500	R4
1	Uporovna lestvica 5x2 v SIL10	5 x 2k2	RP1

Tabela 3: Seznam uporabljenih elementov

ADAPTERJI ZA PROGRAMATOR PG302

V tem poglavju si bomo pobliže ogledali adapterje za 40-pinske procesorje, saj ima osnovni model PG302 le 20-pinsko podnožje.

No, če želite programirati recimo AT89S53, se vam bo v krmilnem programu poleg oznake naprave pojavila še dodatna oznaka - dve zvezdici, kar pa pomeni, da potrebujemo za adapter ADT90. Kot lahko vidite iz slike 20, obstajata dva tipa adapterjev: ADT87 in ADT90. Prvi je namenjen programiranju klasičnih 40-pinskih procesorjev družine 8051, drugi pa vezjem, ki imajo SPI vodilo.



Slika 19: Izbira mikrokontrolerja, ki ga želimo programirati

AMD	Atmel	Dallas	Intel	Philips	Adapter
87C51*	89C1051	87C520*	8751BH*	87C51*	Brez dodatne
87C52*	89C2051		8752BH*	87C51FA*	oznake adapter
87C521*	89C51*		87C51*	87C51FB*	ni potreben
87C541*	89C52*		87C51FA*	87C52*	* Potreben je
	89C55*		87C51FB*	87C504*	adapter za
	89S53**		87C52*	87C524*	40-pinska
	89S8252**		87C54*	87C528*	podno ja
	90S1200			87C550*	**ADT87**
	90S2313			87C575*	** Potreben je
	90S4414**			87C576*	adapter za SPI
	90S8515**			87C652*	vodilo
				87C654*	**ADT90**

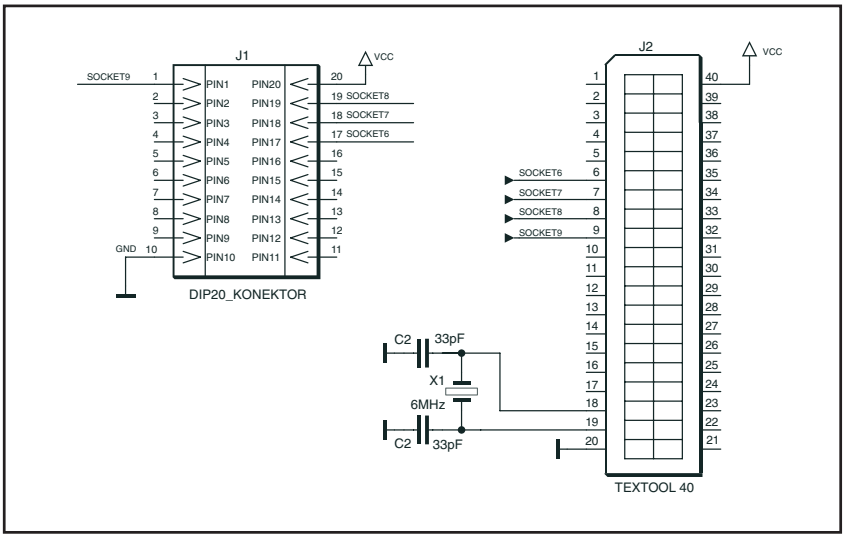
Tabela 4: Seznam komponent, ki jih lahko programira PG302 za adapterjema ADT87 in ADT90

Tabela 4 prikazuje vse tipe mikrokontrolerjev, ki jih zna programirati programator PG302 skupaj z obema adapterjema.

Kot vidite, so posebej podprti mikrokontrolerji Atmel serije AVR. To so hitri RISC procesorji, ki se bodo v prihodnosti vse več uporabljali, prihajajo pa tudi vedno novi člani družine.

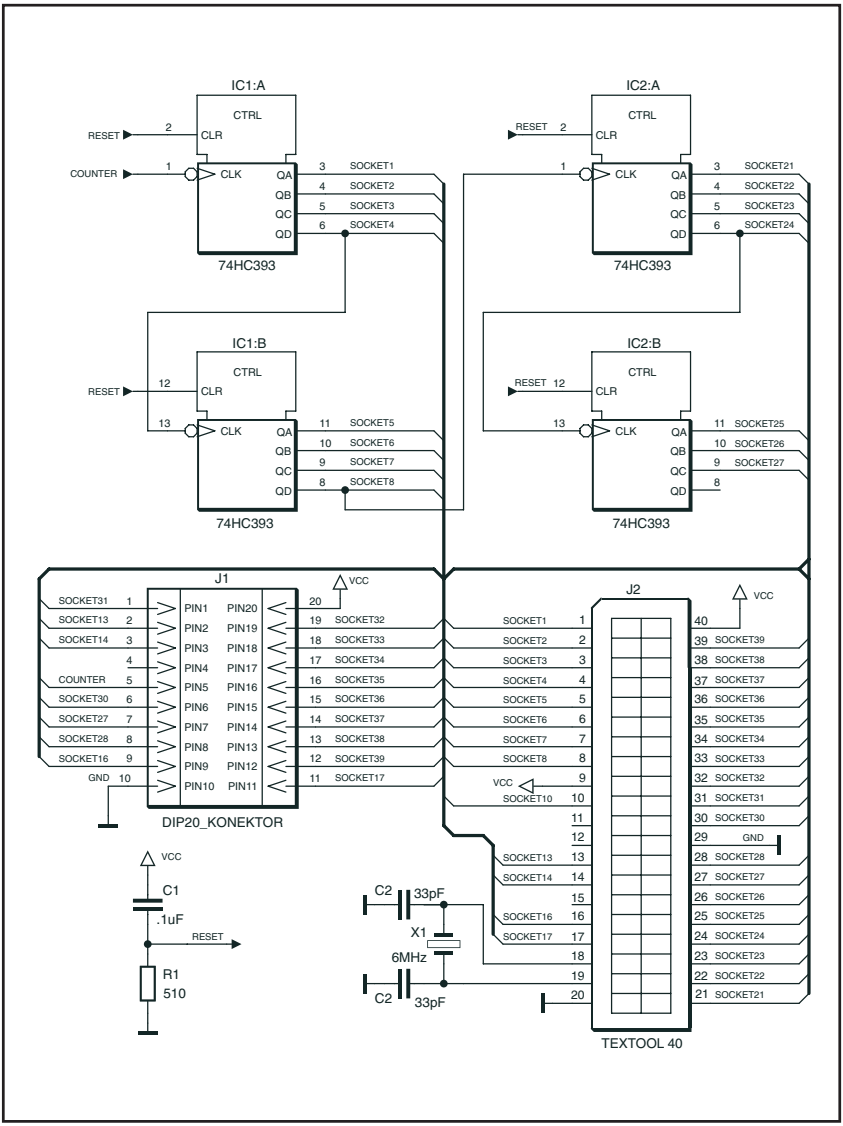
Električna shema

Sliki 20 in 21 prikazujeta električni shemi adapterjev. ADT90 je zelo enostaven, saj je vsa umetnost v tem, da pravilno prevežemo pine SPI vodila. Dodan je še kristal s pripadajočima kondenzatorjema za oscilator frekvence 6MHz. Ta frekvenca je potrebna za programiranje preko SPI vodila. Časovne diagrame programiranja bi sicer lahko predstavil v tem priročniku vendar mislim, da je to tematika, ki ne sodi v ta priročnik. Sicer si pa jih lahko ogledate pri opisu Atmelovega RISC procesorja z SPI vodilom - AVR 90S1200 v eni od starejših številk revije Svet elektronike.



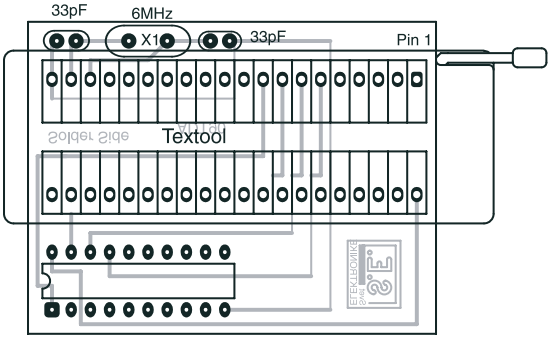
Slika 20: Električna shema adapterja ADT90

Drug problem je pri adapterju ADT87, kjer že številka v imenu pove, da gre za družino mikrokontrolerjev 89C51 in nekatere druge derivate, ki imajo 40-pinsko DIL podnožje. Tukaj potrebujemo tako podatkovne kot naslovne linije. Zato sta uporabljena dva dvojna 4-bitna binarna števca, kjer so posamezni števci vezani serijsko tako, da dobimo 16-bitno naslovno področje. Števec inkrementiramo z vhodom "Counter", ki ga krmili mikrokontroler na programatorju. Števce postavimo na 0 tako, da prekinemo napajanje

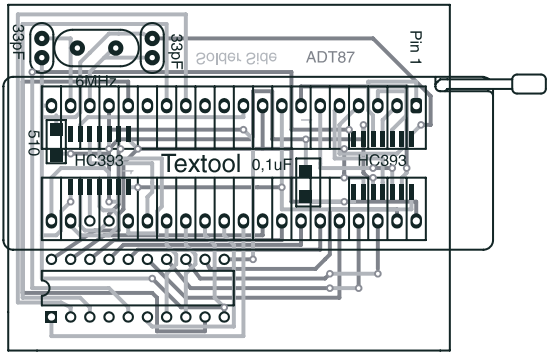


Slika 21: Električna shema adapterja ADT87

Vcc in ga ponovno vrnemo. Ves ostali Handshake je izveden preko ostalih linij neposredno preko 20-pinskega konektorja.
V uredništvu revije smo pripravili tudi adapter za PLCC podnožje, vendar o tem kaj več, če pokličete v uredništvo revije.



Slika 22: Montažna shema adapterja ADT90



Slika 23: Adapter ADT87 - pogled na zgornjo stran, ostali elementi se spajkajo na spodnji strani

SESTAVLJANJE

Na slikah 22 in 23 vidimo razpored elementov na ploščicah obeh adapterjev. 20-polni DIL konektor se v obeh primerih prispajka na spodnjo stran, na adapterju ADT87 pa tudi nekaj SMD komponent. Le-te so uporabljene zaradi čim manjše višine. Ja, še to. Za DIL20 konektor lahko uporabite kar moško kontaktno letvico, ki jo lahko poljubno nalomite. Ker je ADT90 enostransko vezje, morate paziti pri spajkanju. Konektor morate namreč prispajkati s strani, saj so povezave na isti strani kot konektor.

Prižiganje osmih LED diod

```
Dim Lucka As Byte                ‘Lucka bo tipa Byte
Lucka = 0                        ‘definiramo zacetno vrednost
Do                               ‘Ukaz Do je zacetek Do-Loop zanke
P1 = Lucka                       ‘Lucka je na izhodu P1
Wait 1                           ‘pocakaj 1 sekundo
Lucka = Not Lucka                ‘invertiramo vrednost Lucke
Loop                             ‘konec Do-Loop zanke
End                              ‘konec programa
```

Primer prižiganja lučk za božično drevo

```
Dim Clock As Byte
Clock = 0

Do
For Clock = 1 To 3
Gosub Kratko
Waitms 250
Next
For Clock = 1 To 3
Gosub Dolgo
Waitms 250
Next
For Clock = 1 To 1
Gosub Kratko
Waitms 250
Next
For Clock = 1 To 4
Gosub Dolgo
Waitms 250
Next
For Clock = 1 To 5
Gosub Kratko
Waitms 250
Next
For Clock = 1 To 2
Gosub Dolgo
Waitms 250
Next
For Clock = 1 To 4
Gosub Kratko
Waitms 250
Next
For Clock = 1 To 3
Gosub Dolgo
Waitms 250
Next
Loop
End

Dolgo:
P1 = 255
Wait 1
P1 = 0
Wait 1
Return

Kratko:
P1 = 255
Waitms 250
P1 = 0
Waitms 250
Return
```

Primer klavirja

```
'dimenzioniranje spremenljivk
Dim Frekvenca As Word , Pom As Word , Traj As Word , Pom1 As Word
Dim Ti1 As Bit , Ti2 As Bit , Ti3 As Bit , Ti4 As Bit , Ti5 As Bit
Dim Ti6 As Bit , Ti7 As Bit , Ti8 As Bit , Ti0 As Bit

Do

Ti0 = P3.0                'tipke za posamezni ton
Ti5 = P3.5
Ti1 = P3.1
Ti2 = P3.2
Ti4 = P3.4
Ti3 = P3.3
Ti6 = P1.7
Ti7 = P1.6
Ti8 = P1.5

If Ti1 = 0 Then
   Traj = 500
Gosub Zvok1
End If
If Ti2 = 0 Then
   Traj = 500
Gosub Zvok2
End If
If Ti3 = 0 Then
   Traj = 500
Gosub Zvok3
End If
If Ti4 = 0 Then
   Traj = 500
Gosub Zvok4
End If
If Ti5 = 0 Then
   Traj = 500
Gosub Zvok5
End If
If Ti0 = 0 Then
   Traj = 500
Gosub Zvok0
End If
If Ti6 = 0 Then
   Traj = 500
Gosub Zvok6
End If
If Ti7 = 0 Then
   Traj = 500
Gosub Zvok7
End If
If Ti8 = 0 Then
   Traj = 500
Gosub Zvok8
End If
Loop
End

Zvok0:
Restore Tabela0
       Read Frekvenca
          Sound P3.7 , Traj , Frekvenca
Return
Zvok1:
Restore Tabela1
       Read Frekvenca
          Sound P3.7 , Traj , Frekvenca
Return
```

```
Zvok2:
Restore Tabela2
       Read Frekvenca
          Sound P3.7 , Traj , Frekvenca
Return
Zvok3:
Restore Tabela3
       Read Frekvenca
          Sound P3.7 , Traj , Frekvenca
Return
Zvok4:
Restore Tabela4
       Read Frekvenca
          Sound P3.7 , Traj , Frekvenca
Return
Zvok5:
Restore Tabela5
       Read Frekvenca
          Sound P3.7 , Traj , Frekvenca
Return
Zvok6:
Restore Tabela6
       Read Frekvenca
          Sound P3.7 , Traj , Frekvenca
Return
Zvok7:
Restore Tabela7
       Read Frekvenca
          Sound P3.7 , Traj , Frekvenca
Return
Zvok8:
Restore Tabela8
       Read Frekvenca
          Sound P3.7 , Traj , Frekvenca
Return

'definicije posameznih tonov
Tabela0:
Data 250%
Tabela1:
Data 230%
Tabela2:
Data 200%
Tabela3:
Data 190%
Tabela4:
Data 170%
Tabela5:
Data 150%
Tabela6:
Data 135%
Tabela7:
Data 125%
Tabela8:
Data 105%
```

Primer "zvonca" s 5 tipkami in različnimi melodijami

```
'izhod je na portu 3.7
Dim Frekvenca As Word                          'frekvenca
Dim Pom As Word                                'pomozna spremenljivka
Dim Traj As Word                               'trajanje igranja zvoka
Dim Pom1 As Word                               'pomozna spremenljivka
Dim Ti0 As Bit , Ti1 As Bit , Ti2 As Bit , Ti3 As Bit , Ti4 As Bit

Do                                             'zacetek DO-LOOP zanke
Ti0 = P3.0                                     'definicija vhodnih tipk
```

```
Ti1 = P3.1
Ti2 = P3.2
Ti3 = P3.3
Ti4 = P3.4
Traj = 500                  'dolocitev trajanja igranja zvoka

If Ti0 = 0 Then             'ce je pritisnjena tipka Ti1, potem zaigraj Zvok1
   Traj = 300               'nekaterim melodijam je potrebno prilagoditi trajanje
   Gosub Zvok0
End If
If Ti1 = 0 Then
   Gosub Zvok1
End If
If Ti2 = 0 Then
   Gosub Zvok2
End If
If Ti3 = 0 Then
   Traj = 1000
   Gosub Zvok3
End If
If Ti4 = 0 Then
   Traj = 300
   Gosub Zvok4
End If
Loop                        'konec DO-LOOP zanke
End
'_____________________________________________________________________________
'Subrutine za razlicne melodije
Zvok0:                      'subrutine za razlicne melodije
   Restore Tabela0
   For Pom = 1 To 21
      Read Frekvenca
         Sound P3.7 , Traj , Frekvenca
   Next
Return

Zvok1:
Restore Tabela1
   For Pom = 1 To 15
      Read Frekvenca
         Sound P3.7 , Traj , Frekvenca
   Next
Return

Zvok2:
Restore Tabela2
   For Pom = 1 To 15
      Read Frekvenca
         Sound P3.7 , Traj , Frekvenca
   Next
Return

Zvok3:
Restore Tabela3
   For Pom = 1 To 9
      Read Frekvenca
         Sound P3.7 , Traj , Frekvenca
   Next
Return

Zvok4:
Restore Tabela4
   For Pom = 1 To 21
      Read Frekvenca
         Sound P3.7 , Traj , Frekvenca
   Next
Return
```

```
'podatki za razlicne melodije
'1% je tukaj v funkciji takta ali kratke pavze
Tabela0:
Data 150% , 1% , 150% , 1% , 150% , 1% , 190% , 190% , 190% , 190% , 190% , 190%
, 170% , 1% , 170% , 1% , 170% , 1% , 200% , 200% , 200% , 200%
Tabela1:
Data 255% , 190% , 190% , 150% , 150% , 250% , 250% , 190% , 200% , 200% , 140%
, 140% , 1% , 140% , 140% , 140%
Tabela2:
Data 170% , 170% , 200% , 255% , 170% , 170% , 200% , 255% , 230% , 230% , 200%
, 190% , 200% , 230% , 230
Tabela3:
Data 150% , 190% , 170% , 255% , 1% , 255% , 170% , 150% , 190
Tabela4:
Data 125% , 125% , 125% , 1% , 125% , 125% , 125% , 1% , 125% , 125% , 125% ,
150% , 190% , 200% , 200% , 1% , 170% , 110% , 110% , 125% , 125
```

Prikaz z LED displejem brez multipleksa

```
'_____________________________________________________________________________
'                      (c)1997 Mirko Pelcl & Jure Mikeln
'_____________________________________________________________________________
'Dolocanje tipa spremenljivk
Dim Clock As Byte , Clock1 As Byte , Mux As Byte , Sekunde As Byte , X As Byte
Dim Pomozna_v As Byte , Segmenti As Byte , Enice As Byte , Desetice As Byte
Dim Prikaz As Bit , Izracun As Bit
Config Timer0 = Timer , Gate = Internal , Mode = 2'konfiguriramo Timer
'Timer0              uporabimo timer 0
'Gate = Internal     brez zunanje prekinitve(interupt)
'Mode = 2            8 bit auto reload
On Timer0 Timer_0_int                'prekinitvena  rutina
Load Timer0 , 250                    'nalozimo v timer0 vrednost 250 µsek
Priority Set Timer0                  'prioriteta dolocena timerju0
Enable  Interrupts                   'omogocimo  prekinitve
Enable  Timer0                       'omogocimo  delovanje timerja0
Start Timer0                         'startamo  timer0
Clock = 0                            'dolocimo  zacetne  vrednosti  spremenljivk
Clock1 = 0                           'dolocimo  zacetne  vrednosti  spremenljivk
Sekunde = 0                          'dolocimo  zacetne  vrednosti  spremenljivk

Do                                   'zacetek DO-LOOP neskoncne zanke
   If Izracun = 1 Then               'Izracun = vsako sekundo
      Izracun = 0
'_____________________________________________________________________________
'Rutina za dolocanje desetic in enic
      Desetice = Sekunde / 10
      Pomozna_v = Desetice * 10
      Enice = Sekunde - Pomozna_v
'_____________________________________________________________________________
   End If
   If Prikaz = 1 Then                'prikaz na displeju samo, ko je Prikaz=1
      Prikaz = 0
      P1.0 = 1                       'postavi P1.0 na 1, vklopi tranzistor
'_____________________________________________________________________________
'rutina  za  prikaz  enic
         Pomozna_v = Enice
         Gosub Prikaz                'pojdi  na  subrutino Prikaz
         P1.0 = 0
'_____________________________________________________________________________
   End If
Loop                                 'konec  DO-LOOP  zanke
End                                  'konec  programa
'_____________________________________________________________________________
'prekinitvena  rutina
Timer_0_int:
    Incr Clock                                 'povecaj Clock za 1
```

```
    If Clock > 19 Then                          'ko bo Clock vecji od 19
         Clock = 0                              'ga resetiraj na 0
         Prikaz = 1                             'omogoci prikaz
         Incr Clock1                            'povecaj Clock1 za 1
            If Clock1 > 199 Then                'ko bo Clock1 vecji od 199
               Clock1 = 0                       'ga resetiraj na 0
               P1.7 = Not P1.7                  'kontrolni bit
               Izracun = 1                      'postavi Izracun na 1
               Incr Sekunde
               If Sekunde > 59 Then
                  Sekunde = 0
               End If
            End If
         End If
Return
'-------------------------------------------------------------------------------
'rutina za prikaz na LED displeju
Prikaz:                                 'pazi na napako:dvopicje mora biti
                                        'zraven brez presledka
Restore Tabela                          'nalozi Tabelo v spomin
   For X = 0 To 9                       'od X = 0 do X = 9
      Read Segmenti                     'preberi iz Tabele vrednost in jo
                                        'preslikaj v spremenljivko z imenom Segmenti
      If X = Pomozna_v Then             'ce je X = pomozna_v potem
         P3 = Segmenti                  'prikazi Segmenti na portu P3
         Exit For                       'konec FOR zanke
      End If
   Next
Return
'-- podatki za pravilen prikaz stevilk na LED displeju ---
Tabela:
Data 3, 159, 38, 14, 154, 74, 66, 31, 2, 10
```

Prikaz z LED displejem z multipleksom

```
'-------------------------------------------------------------------------------
'                      (c)1997 Mirko Pelcl & Jure Mikeln
'-------------------------------------------------------------------------------
'Dolocanje tipa spremenljivk
Dim Clock As Byte , Clock1 As Byte , Mux As Byte , Sekunde As Byte , X As Byte
Dim Pomozna_v As Byte , Segmenti As Byte , Enice As Byte , Desetice As Byte
Dim Prikaz As Bit , Izracun As Bit

Config Timer0 = Timer , Gate = Internal , Mode = 2
'Timer0              uporabimo timer 0
'Gate = Internal     brez zunanje prekinitve(interupt)
'Mode = 2            8 bit auto reload
On Timer0 Timer_0_int                   'prekinitvena  rutina

Load Timer0 , 250                       'nalozimo v timer0 vrednost 250 µsek
Priority Set Timer0                     'prioriteta dolocena timerju0
Enable Interrupts                       'omogocimo prekinitve
Enable Timer0                           'omogocimo delovanje timerja0
                                        'startamo timer0
Clock = 0                               'dolocimo zacetne vrednosti spremenljivk
Clock1 = 0                              'dolocimo zacetne vrednosti spremenljivk
Sekunde = 0                             'dolocimo zacetne vrednosti spremenljivk
Do                                      'zacetek DO-LOOP neskoncne zanke
   If Izracun = 1 Then                  'Izracun = vsako sekundo
      Izracun = 0
'-------------------------------------------------------------------------------
'Rutina za dolocanje desetic in enic
      Desetice = Sekunde / 10
      Pomozna_v = Desetice * 10
      Enice = Sekunde - Pomozna_v
'-------------------------------------------------------------------------------
```

```
  End If
   If Prikaz = 1 Then
      Prikaz = 0                    'multipleks hitrost je  5ms (20*250us)
      P1.3 = 1                      'P1.3 in P1.7 sta driverja za digite za
                                    'enice in desetice
      P1.0 = 1                      '1, 1  oba digita sta ugasnjena
'_____________________________________________________________________________
'rutina za prikaz desetic
      If Mux = 2 Then
         Pomozna_v = Desetice
         'P1.0 = 1                  'ni nujno, da ugasnemo ta segment
         Gosub Prikaz               'pojdi na subrutino Prikaz
         P1.3 = 0
      End If
'_____________________________________________________________________________
'rutina za prikaz enic
      If Mux = 0 Then
         Pomozna_v = Enice
         'P1.3 = 1                  'ni nujno, da ugasnemo ta segment
         Gosub Prikaz               'pojdi na subrutino Prikaz
         P1.0 = 0
      End If
   End If
'_____________________________________________________________________________
Loop                                'konec DO-LOOP zanke
End                                 'konec programa
'_____________________________________________________________________________
'prekinitvena  rutina
Timer_0_int:
    Incr Clock                      'povecaj Clock za 1
      If Clock > 19 Then            'ko bo Clock vecji od 19
         Clock = 0                  'ga resetiraj na 0
         Incr Mux                   'povecaj Mux za 1
         If Mux > 3 Then            'ko bo Mux vecji od 3
             Mux = 0                'ga resetiraj na 0
         End If
         Prikaz = 1                 'omogoci  prikaz
         Incr Clock1                'povecaj Clock1 za 1
            If Clock1 > 199 Then    'ko bo Clock1 vecji od 199
               Clock1 = 0           'ga resetiraj na 0
               P1.7 = Not P1.7      'kontrolni  bit
               Izracun = 1          'postavi Izracun na 1
               Incr Sekunde         'povecaj sekunde za 1
               If Sekunde > 59 Then 'ko so sekunde >59
                  Sekunde = 0       'jih postavi na 0
               End If
            End If
         End If
Return
'_____________________________________________________________________________
'rutina za prikaz na LED displeju
Prikaz:                             'pazi na napako:dvopicje mora biti zraven
                                    'brez presledka
Restore Tabela                      'nalozi Tabelo v spomin
   For X = 0 To 9                   'od X = 0 do X = 9
      Read Segmenti                 'preberi iz Tabele vrednost in jo preslikaj
                                    'v spremenljivko z imenom Segmenti
      If X = Pomozna_v Then         'ce je X = pomozna_v potem
         P3 = Segmenti              'prikazi Segmenti na portu P3
         Exit For                   'konec FOR zanke
      End If
   Next
Return
'—— podatki za pravilen prikaz stevilk na LED displeju ——
Tabela:
Data 3, 159, 38, 14, 154, 74, 66, 31, 2, 10
```