Sadržaj

Uvod

U priručniku za laboratorijske vežbe iz predmeta Mikroprocesorski Softver je na praktičan način, kroz jednu uvodnu i pet vežbi obrađena materija koja je izložena na predavanjima iz ovog predmeta.

Sve vežbe su realizovane korišćenjem Intelove serije šesnaestobitnih mikrokontrolera sa oznakom MCS96 koja je u drugoj polovini devedesetih godina postala industrijski standard i nekoliko godina držala preko 50% tržišta šesnaestobitnih mikrokontrolera.

Za potrebe vežbi realizovano je i kompletno hardversko okruženje u obliku razvojnog sistema μM196 podržano monitorskim programom za komunikaciju sa računarom opšte namene, recimo PC, tako da svaki student može samostalno da razvije program, prebaci u realni hardver i isproba program u realnim uslovima. Priručnik sadrži i uputstvo za korišćenje ovog razvojnog sistema.

U uvodnoj (prvoj) vežbi obrađen je način pisanja izvornog kôda, pozivanje asemblera, analiza izveštaja asemblera i povezivanje apsolutno asembliranog fajla, sa pogledom na izveštaj povezivača (linkera).

U drugoj vežbi detaljno su obrađeni načini adresiranja. Zatim je obrađeno definisanje globalnih i lokalnih promenljivih, da bi se moglo preći na razmatranje pisanja programa korišćenjem modula i povezivanja modula. Takođe je obrađen izgled izveštaja asemblera kod relativnog asembliranja i, na kraju, izveštaj povezivača posle povezivanja dva modula.

Treća vežba već zahteva od studenta određeno poznavanje asmeblera i linkera, kao i mogućnost samostalnog osmišljavanja algortima koji može da se implementira u asembleru. Obrađeno je kreiranje tabele u programskom segmentu gde je akcenat stavljen na indeksno adresiranje kao najbitnije za pristup i izmenu pojedinih članova tabele. Obrađen je delimično i paralelni port mikrokontrolera, kao i razvojni sistem na kome se vežbe sprovode.

U četvrtoj vežbi je najviše pažnje posvećeno radu sa stringovima, kao i osnovnim operacijama nad stringovima, kao jednoj od najčešće korišćenih dinamičkih struktura. U okviru ove vežbe, obrađena je i veza PC računara sa razvojnim sistemom. Takođe, serijska komunikacija je obrađena kroz praktične primere.

Peta vežba detaljnije pokriva serijski i paralelni port kao periferije, uz praktične primere za korišćenje.Uz ovo, skreće se pažnja i na primenu makroa, kroz praktičnu primenu, i na mogućnost i korisnost njihove upotrebe u programiranju na nižim programskim jezicima.

U šestoj vežbi, studenti se detaljnije upoznaju sa mehanizmom prekida, kroz korišćenje softverskog prekida u generisanju PWM signala (sa promenljivom srednjom vrednošću) na paralelnom portu kontrolera. Takođe, zahteva se i korišćenje High Speed Output jedinice koji izaziva prekid koristeći jedan od internih tajmera.

U dodatku 1 je ukratko opisan razvojni sistem sa svim periferijama kontrolera i dodatnim periferijama koje se nalaze u sklopu razvojnog sistema, a koje se na vežbama koriste, i koje bi student mogao koristiti u rešavanju praktičnih problema.

Dodatak 2 je sastavljen od spiska svih veza razvojnog sistema koje su dostupne spolja sa blok šemom razvojnog sistema.

Dodatak 3 je podsetnik na skup instrukcija sa kratkim opisom instrukcija pojedinačno.

U dodatku 4 se nalazi ASCII tabela.

U dodatku 5 se nalazi opis terminal emulator programa Tera Term Pro koji će se na vežbama koristiti.

Dodatak 6 opisuje podešavanja razvojnog okruženja (radi se o ConTEXT editoru-text editoru koji se koristi na vežbama).

U dodatku 7 je dat kratak opis batch fajla koji uprošćava prevođenje i povezivanje.

VEŽBA 1

- **Izrada izvornog fajla**
  - Deklaracije registara
  - Korišćenje različitih formata brojeva i karaktera
  - Korišćenje asemblerskog pokazivača adresa (location counter)
- **Pozivanje asemblera**
- **Analiza izveštaja asemblera**
  - Razmotriti dodelu adresa promenljivama
  - Primetiti redosled zapisa nižeg i višeg bajta (reči)
  - Razlika u operacionim kodovima iste instrukcije
- **Povezivanje apsolutno asembliranog fajla**
- **Letimičan pogled na izveštaj povezivača (linkera)**

ZADATAK:

Izraditi izvorni program u kome bi:
- U registarskom segmentu počev od adrese 1AH deklarisali sledeće promenljive:

bajt - promenljiva tipa bajt
bajt1 - promenljiva tipa bajt
lon - promenljiva tipa long
rec - promenljiva tipa reč
rec1 - promenljiva tipa reč
Zbog daljih komentara, promenljive deklarisati po navedenom redosledu!

- Simbolu **BROJ** dodeliti vrednost 27

Na primer, za deklaraciju bajta koristiti direktivu **`bajt: dsb 1`**, *a za dodeljivanje vrednosti simbola direktivu* **`BROJ equ 27`**
Asembler ne razlikuje mala i velika slova!

- U kod-segmentu, počev od adrese 2080H izraditi sledeći program (*komentari iza simbola **;** nisu deo programa već pitanja za razmišljanje na osnovu izveštaja asemblera*):

```
poc:  ldb   bajt, bajt1       ; šta će uraditi ova instrukcija?
      ldb   bajt, BROJ        ; po čemu se ova razlikuje od prethodne?
      ldb   bajt, #BROJ       ; … a ova? Obratiti pažnji na operacioni kod
      ldb   bajt, bajt+1      ; šta će uraditi ova instrukcija?
      ; Brojčani podaci u asembleru
      ldb   bajt, #26
      ldb   bajt, #26H
      ldb   bajt, #111001b
      ld rec, #111001b        ; obratiti pažnju na razliku u odnosu na prethodnu!
      ldb bajt, #-3           ; kako će asembler predstaviti broj -3?
      ld rec, #-3             ; kako će asembler predstaviti broj -3 u ovom slučaju?
      ldb bajt, #´A´          ; kako će asembler predstaviti slovo A, zašto?
      ld rec, #´AB´           ; string kao podatak. Obratiti pažnju i na redosled!
      ld rec, #$              ; kako je preveden simbol $, koliko iznosi?
      ld rec, #$              ; kako je sada preveden simbol $, koliko sada iznosi?
```

```
add rec, lon          ; šta će uraditi ova instrukcija?
add rec, lon+2        ; šta će uraditi ova instrukcija?
ld rec, #poc          ; šta će uraditi ova instrukcija?
; Program završiti direktivom end
```

Procedura:

1. Klikom na ikonu **MPSV1.bat**, automatski će, u ConTEXT editoru, biti otvoren (za editovanje) fajl **Vezba1.asm**. Izvorni kod treba napisati i sačuvati pod ovim imenom. Pritiskom na taster **F9** automatski biva pozvan izvršni (batch) fajl koji za redom poziva asembler, linker i hex translator. U slučaju greške u bilo kom stepenu izvršavanja, automatski biva pozvan (otvoren za editovanje u ConTEXT editoru) izveštaj u kome se može pronaći greška zbog koje program nije do kraja preveden. Takodje u "Output Console" prozoru se može videti kako je proteklo prevođenje, povezivanje ili hex-translacija. ConTEXT editor je pomoću **80c196Assembler** highlighter-a povezan i sa help fajlom Asmhyper.hlp. Poziv help fajla, tj. poziv pomoći se dobija pozicioniranjem kursora na instrukciju za koju se traži objašnjenje i pritiskom na taster **F1.**

2. Po završenom asembliranju, asembler na ekranu ispisuje broj grešaka. Izveštaj rada asemblera je u fajlu **Vezba1.lst**, koji je automatski otvoren u editoru tako da možete naći u kojoj liniji kôda se tačno nalazi greška. Greške koje je prijavio asembler treba ispraviti **editujući ponovo izvorni fajl Vezba1.asm** sve dok broj grešaka ne bude 0.
Svaka linija izveštaja **Vezba1.lst** u levom delu sadrži vrednost asemblerskog pokazivača adresa (location counter), zatim izvršni kôd (rezultat asembliranja), a u desnom delu odgovarajući izvorni kôd, onako kako je unet. Ostali sadržaji će biti komentarisani u sledećim vežbama

- Pronaći u izveštaju koje su adrese dodeljene promenljivama.
- Pronaći u izveštaju operacione kodove i operande za instrukcije koje su korišćene.
  Pažnja, operandi u izvršnom kodu idu obrnutim redom, prvo drugi, pa prvi operand!
- Šta se desilo sa vrednošću asemblerskog pokazivača adresa (lokation counter) posle definicije simbola BROJ. Premestiti ovu liniju među deklaracije varijabli i asemblirati ponovo. Da li ima razlike? Primetiti da se definicija simbola u izveštaju razlikuje od deklaracije promenljivih po tome što vrednost asemblerskog pokazivača adresa nije definisana za tu liniju. *Definicija simbola ne pomera asemblerski pokazivač adresa i može se naći u bilo kom segmentu.*
- Tabela simbola je data na kraju izveštaja. Pokušajte da protumačite značenja atributa.

Objektni kod (rezultat asembliranja) nalazi se u fajlu sa imenom **vezba1.obj** i ovaj fajl nije čitljiv (nije tekstualni pa se ne može videti pomoću standardnog editora teksta).
Povezivanje fajla je po definiciji nužno (prvenstveno zbog tipova fajlova) čak i ako postoji samo jedan izvorni fajl u kome su sve adrese dodeljene.

3. Izveštaj povezivača se nalazi u fajlu pod imenom **vezba1.m96** (ovaj fajl će napraviti povezivač i u njemu će biti naznačene i eventualne greške prilikom povezivanja). To je tekstualni fajl i može se čitati. Detalji ovog izveštaja će biti predmet narednih vežbi
Izvršni kod (rezultat povezivanja) nalazi se u fajlu sa imenom **vezba1. (bez ekstenzije)** i ovaj fajl nije čitljiv (nije tekstualni pa se ne može videti pomoću standardnog editora teksta).

4. Na kraju vežbi, odjaviti se sa računara.

VEŽBA 2

- **Načini adresiranja**
  - Sintaksa pojedinih načina adresiranja
  - Razlike u operacionim kodovima zavisno od načina adresiranja
- **Definisanje globalnih i spoljašnjih promenljivih**
- **Povezivanje modula**
- **Analiza izveštaja asemblera kod relativnog asembliranja**
  - Razlika u operacionim kodovima iste instrukcije
- **Analiza izveštaja povezivača**
  - Razmotriti dodelu adresa promenljivama

ZADATAK:

Izraditi izvorni program u kome bi:
- U registarskom segmentu počev od adrese 20H deklarisali sledeće promenljive:

Ax - promenljiva tipa reč
Pokaz - promenljiva tipa reč

Rezervisati prostor za pet podataka tipa reč sa početnom adresom **Podaci** (direktiva za ovo rezervisanje bi bila **`Podaci: dsw 5`**).
- U segmentu podataka počev od adrese 1000h rezervisati mesto za 100 podataka tipa reč sa početnom adresom **`Tabela`** (direktiva je ista kao za rezervaciju prostora u registarskom segmentu).

- U kod-segmentu, počev od adrese 2080H izradili sledeći program:

```
public     Pokaz, Podaci, Tabela, Pristup_pod, Pristup_tab
extrn      nazad                  ; Labela koja će biti definisana u drugom programu
     ld    Ax, Pokaz              ; INSTRUKCIJA 1
     ld    Ax, #Pokaz             ; INSTRUKCIJA 2
Pristup_pod:
     ld    Pokaz, # Podaci        ; inicijalizacija pokazivača na podatke
     ld    Ax, [Pokaz]            ; INSTRUKCIJA 3
     ld    Ax, [Pokaz]+           ; INSTRUKCIJA 4
     ld    Ax, 2[Pokaz]           ; INSTRUKCIJA 5
     br    nazad
Pristup_tab:
     ld    Ax, Tabela [Pokaz]     ; INSTRUKCIJA 6
     add   Pokaz, #2              ; način kretanja kroz tabelu
     ld    Ax, Tabela [Pokaz]     ; INSTRUKCIJA 7
     br    $
     ; Program završiti direktivom end
```

-Asemblirati napisani izvolni kôd. Na osnovu izveštaja asemblera (obratiti pažnju na operacioni kôd i operande) odgovoriti na sledeća pitanja:
Koje načine adresiranja prestavljaju INSTRUKCIJE 1 do 6 ?_____
Koji su operacioni kodovi instrukcije **`ld`** ?_____
Koja je razlika INSTRUKCIJA 1 i 2 ?_____
Koja je razlika INSTRUKCIJA 5 i 6 ?_____

Koji su atributi simbola definisanih u ovom programu ?_____

U drugom delu vežbe, izraditi novi program koji bi se povezao sa prvim, a u kome bi se inicijalizovali podaci.
- Otvoriti novi tekstualni fajl i izraditi sledeći program:

```
      CSEG
Public      nazad
      ld    Podaci, #1
      ld    Podaci+2, #2
      ld    Podaci+4, #3
      ld    Podaci+6, #4
p1:   br    Pristup_pod       ; skok na deo programa za pristup podacima u RSEG
nazad:

      ld    Bx, #10h          ; jedan od načina punjenja tabele u DATA segmentu
      st    Bx, Tabela        ; INSTRUKCIJA 8
      ld    Bx, #20h          ; novi podatak (Bx je lokalna, ponoćna promenljiva)
      st    Bx, Tabela+2      ; INSTRUKCIJA 9

      clr   Pokaz             ; inicijalizacija pokazivača na podatke u tabeli u DSEG
p2:   br    Pristup_tab       ; skok na deo programa za pristup podacima u DSEG
      end
```

Da bi program mogao uspešno da se asemblira nužno je samostalno definisati registarski segment i ubaciti EXTRN direktive (tamo gde je potrebno) za promenljive za koje je to nužno.

- Asemblirati i ovaj fajl. Kako sada izgleda izveštaj asemblera ?
- Povezati dobijeni objektni fajl sa objektnim fajlom prvog programa

Za ovaj slučaj, povezivač (linker) se poziva u DOS prozoru komandom

rl96 ime1.obj, ime2.obj

*...gde su **ime1** i **ime2** imena koja ste dali izvornim fajlovima prvog i drugog programa.*

Izveštaj povezivača se nalazi u fajlu pod imenom **ime1.m96** (ovaj fajl će napraviti povezivač i u njemu će biti naznačene i eventualne greške prilikom povezivanja). To je tekstualni fajl i moze se čitati.

Pitanja vezana za izveštaj povezivača:
- Od koje adrese je povezivač smestio kôd drugog programa ?_____
- Pronaći u izveštaju koliko je bajtova reg. prostora zauzeto u prvom programu _____
- Pronaći u izveštaju informacije o globalnim promenljivima

Pitanja vezana za adresiranja:
- Posle skoka označenog labelom **p1**…

koji je sadržaj registra Ax nakon izvršene INSTRUKCIJE 3 ?_____
koji je sadržaj registra Ax nakon izvršene INSTRUKCIJE 4 ?_____
koji je sadržaj registra Ax nakon izvršene INSTRUKCIJE 5 ?_____

Koji je efekat INSRUKCIJA 8 i 9 ?_____

- Posle skoka označenog labelom **p2**…
koji je sadržaj registra Ax nakon izvršenih INSTRUKCIJA 6 i 7 ?_____
- Kako bi se nastavio pristup ostalim podacima tabele ? _____

Prvi deo naredne, treće vežbe će se sastojati u proveri ovih odgovora na realnom hardveru. Izvršni kôd i listing će tada biti raspoloživi.

Procedura:

1. Klikom na ikonu **MPSV2.bat**, automatski će, u ConTEXT editoru, biti otvoren (za editovanje) fajl **Vezba2.asm**. Izvorni kod treba napisati i sačuvati pod ovim imenom. Pritiskom na taster **F9** automatski biva pozvan izvršni (batch) fajl koji za redom poziva asembler, linker i hex translator. U slučaju greške u bilo kom stepenu izvršavanja, automatski biva pozvan (otvoren za editovanje u ConTEXT editoru) izveštaj u kome se može pronaći greška zbog koje program nije do kraja preveden. Takodje u "Output Console" prozoru se može videti kako je proteklo prevođenje, povezivanje ili hex-translacija. ConTEXT editor je pomoću **80c196Assembler** highlighter-a povezan i sa help fajlom Asmhyper.hlp. Poziv help fajla, tj. poziv pomoći se dobija pozicioniranjem kursora na instrukciju za koju se traži objašnjenje i pritiskom na taster **F1.**

2. Po završenom asembliranju, asembler na ekranu ispisuje broj grešaka. Izveštaj rada asemblera je u fajlu **Vezba2.lst**, koji je automatski otvoren u editoru tako da možete naći u kojoj liniji kôda se tačno nalazi greška. Greške koje je prijavio asembler treba ispraviti **editujući ponovo izvorni fajl Vezba2.asm** sve dok broj grešaka ne bude 0.
Svaka linija izveštaja **Vezba2.lst** u levom delu sadrži vrednost asemblerskog pokazivača adresa (location counter), zatim izvršni kôd (rezultat asembliranja), a u desnom delu odgovarajući izvorni kôd, onako kako je unet. Ostali sadržaji će biti komentarisani u sledećim vežbama
Objektni kod (rezultat asembliranja) nalazi se u fajlu sa imenom **vezba2.obj** i ovaj fajl nije čitljiv (nije tekstualni pa se ne može videti pomoću standardnog editora teksta).
Povezivanje fajla je po definiciji nužno (prvenstveno zbog tipova fajlova) čak i ako postoji samo jedan izvorni fajl u kome su sve adrese dodeljene.

3. Izveštaj povezivača se nalazi u fajlu pod imenom **vezba2.m96** (ovaj fajl će napraviti povezivač i u njemu će biti naznačene i eventualne greške prilikom povezivanja). To je tekstualni fajl i može se čitati. Detalji ovog izveštaja će biti predmet narednih vežbi
Izvršni kod (rezultat povezivanja) nalazi se u fajlu sa imenom **vezba2. (bez ekstenzije)** i ovaj fajl nije čitljiv (nije tekstualni pa se ne može videti pomoću standardnog editora teksta).

4. Na kraju vežbi, odjaviti se sa računara.

VEŽBA 3

- **Izrada programske konstante u obliku tabele.**
  - Praktična primena indeksnog adresiranja.
  - Pristup tabeli
- **Ograničavanje kretanja kroz tabelu**
- **Korišćenje paralelnog porta.**
- **Upoznavanje sa razvojnim sistemom μM 196.**
- **Samostalna izrada, prevođenje i izvršavanje jednostavnog programa.**

ZADATAK1:

Izraditi program za paljenje različitih kombinacija svetlećih dioda vezanih za razvojni sistem. Svaku kombinaciju definisati kao poseban podatak u tabeli, a tabelu očitavati primenom indeksnog adresiranja. Između očitavanja napraviti pauzu željenog trajanja. Pauzu realizovati pomoću potprograma pri čemu treba da postoji mogućnost zadavanja promenljivog trajanja.

Osam svetlećih dioda je povezano na pasivni izlazni paralelni port sa adresom **8000h**. Pri tom je dioda označena sa **L0** vezana za bit 0 ovog porta, dioda **L1** za bit 1 i tako redom. Ukoliko je bit za koji je dioda vezana nula, ta dioda svetli, a ukoliko je bit za koji je vezana jedinica, dioda je ugašena. Upisom odgovarajućeg bajta (čine ga biti 0 do7) na adresu 8000h, programer može po želji da pali ili gasi diode.

Podatak upisan na paralelni port ostaje do sledećeg upisa. Izlazni paralelni port se ne može očitavati. Adresa 8000h nije u registarskom prostoru pa se upis podatka na tu adresu mora obaviti preko nekog pomoćnog registra. Pored toga, izlazni paralelni port je tako realizovan da se u njega može jedino upiasati podatak (očitavanje je nemoguće).

Ako se deklariše pomoćni registar, na primer, **led_pom** (tipa bajt) iz njega se ujedno može i očitati trenutno stanje svetlećih dioda (pod uslovom da se upis svakog novog stanja uvek vrši pomoću ovog registra)
Na primer, instrukcije:

```
LDB   led_pom, #00000000b
STB   led_pom, 8000h
```

će upaliti svih osam svetlećih dioda i one će ostati upaljene do sledećeg upisa na adresu 8000h.

Moguće je definisati simboličku adresu, na primer, LED (**LED equ 8000h**) tako da se ovaj simbol, definisan na jednom mestu, može koristiti umesto unošenja adrese svaki put što pored povećavanja jasnoće zapisa, olakšava eventualnu izmenu.
Tako će, na primer, instrukcije:

```
LDB   led_pom, #11111111b
STB   led_pom, LED
```

ugasiti svih osam svetlećih dioda i one će ostati ugašene do sledećeg upisa na adresu 8000h, a instrukcije:

```
LDB   led_pom, #01111110b
STB   led_pom, LED
```

će ugasiti sve diode osim L7 i L0 koje će upaliti.

Izvorni kod potprograma koji će napraviti pauzu u trajanju oko 1ms (pod pretpostavkom da postoji deklarisan registar **Ax** tipa reč) je sledeći:

```
Pauza:
        Ld Ax,#680        ; 680 prolazaka kroz petlju traje oko 1ms
lab:    dec Ax
        bne lab
ret
```

Samostalno modifikovati ovaj potprogram uvođenjem ulazne promenljive **traj_ms** (tipa reč) pomoću koje bi se moglo zadati željeno trajanje pauze u milisekundama (potrebno je pre izlaska iz potrprograma datu petlju ponoviti onoliko puta koliki je sadržaj u registru **traj_ms**).

Primer jednostavne tabele kombinacija upaljenih i ugašenih dioda sa samo tri kombinacije:

```
TAB:    dcb 11111110b
   dcb 11111101b
   dcb 11111011b
Duz     equ $-TAB              ; automatski određuje dužinu tabele
```

U prvoj kombinaciji upaljena je samo dioda L0, u drugoj samo L1, a u trećoj samo L2. Poslednja linija definiše simbol **Duz** i dodeljuje mu vrednost jednaku broju bajtova u tabeli (ako se tabela poveća, asembler će sam modifikovati vrednost simbola **Duz**).

Primer instrukcija pomoću kojih se može očitati prvi podatak iz tabele pod pretpostavkom da postoji deklarisan indeksni registar **red_br** (tipa reč) koji sadrži redni broj tekućeg podatka u tabeli:

```
clr     red_br                 ; inicijalizacija indeksnog registra
ldb     led_pom, TAB[red_br]   ; preuzimanje podatka pod red_br
```

Ovaj kôd puni pomoćni registar **led_pom** prvim podatkom iz tabele. Napredovanje kroz tabelu moguće je jednostavnim uvećavanjem sadržaja registra **red_br** i korišćenjem iste instrukcije za preuzimanje sledećeg podatka, na primer, u petlji koja će se ponoviti onoliko puta koliko podataka ima u tabeli.

Podsetnik na neke detalje važne za pisanje programa:

- Program koristi potprogram pa je neophodno inicijalizovati pokazivač steka (registar sa adresom 18h). Pokazivač steka treba inicijalizovati na vrednost 200h (kraj internog registarskog prostora upotrebljenog mikrokontrolera)
- Simboličko ime pokazivača steka može da bude bilo koje, na primer **SP**, ali je neophodno definisati vrednost tog simbola, na primer, **SP equ 18h.**
- Svi upotrebljeni registri treba da budu deklarisani u okviru registarskog segmenta.
- Reset adresa od koje treba da počne kôd koji se izvršava je **2080h**.
- Glavni program mora da ima beskonačnu petlje (ne postoji kraj programa). Iza te beskonačne petlje mogu se pisati potprogrami i tabele.
- Tabela se definiše u okviru kod-segmenta ali ne u delu programa koji treba da se izvršava.
- Izvorni kôd Programa se mora završiti direktivom **end**. To nije instrukcija mikroprocesoru već obaveštenje asembleru (koji izvršava PC) da je izvorni kôd završen. Iza ove direktive mora postojati prelaz u novi red jer bez toga asembler ne vidi završetak linije.

ZADACI 2 i 3:

Koristeći napisani program, skratiti tabelu na samo dve zadate kombinacije i ponoviti program sa zadatim trajanjima. Kobinacije koje treba napraviti su sledeće:

```
TAB:    dcb   00000000b
   dcb   10100101b
Duz     equ   $-TAB              ; automatski određuje dužinu tabele
```

Prva kombinacija su sve upaljene diode, a drugu čine naizmenično četiri upaljene i četiri ugašene.

- Napraviti program tako da prva kombinacija traje oko 200 ms, a druga 1000 ms. Prevesti i startovati program. Četiri diode bi trebalo da se pale i gase svake sekunde a ostale diode bi trebalo da svetle sve vreme.

Vratiti se nazad u izvorni i promeniti trajanje ovih istih kombinacija na sledeći način:

- Napraviti program tako da prva kombinacija traje oko 4 ms, a druga 20 ms. Prevesti i startovati program. Program je isti, kombinacije su iste, odnos vremenskih intervala kad diode svetle i kad su ugašene je isti (200:1000 = 4:20), jedino se razlikuje perioda ponavljanja. U prvom slučaju je 1200 ms, a u drugom 24 ms. Kako se sada ponašaju diode?

OKRUŽENJE:

Svaki razvojni sistem μM 196 je povezan sa dva računara tipa PC serijskom vezom. Sve poruke koje šalje razvojni sistem pojavljuju se na oba računara istovremeno. Studenti ne mogu istovremeno slati svoje programe razvojnom sistemu. Dok se jedan program šalje, drugi računar mora da čeka.

Sve razvojne alatke nalaze se na lokalnom disku C u poddirektorijumu **C:\ASM96**. Tu se nalaze programi asemblera (**asm96.exe**), povezivača (**rl96.exe**) i translatora u hex kod (**oh.exe**).

Na istom direktorijumu, u poddirektorijumu **C:\ASM96\UPUTSTVA** nalaze se i uputstva za razvojne alatke (**ASMHYPER.HLP**) i priručnik za korišćenje mikrokontrolera (**M196KC.HLP**). Ova uputstva su u windows help formatu

PC se pored alatke za razvoj programa koristi i kao tastatura i monitor razvojnog sistema. Da bi se ta uloga ostvarila na PC treba pokrenuti terminal-emulator program koji pretvara PC u tastaturu, vezanu za predjnu liniju serijske veze; i monitor, vezan za prijemnu liniju serijske veze. Sve što se otkuca na tastaturi, kao ASCII kod se prenosi do razvojnog sistema, i sve što razvojni sistem pošalje na serijski port pojavljuje se na monitoru računara.

Program za komunikaciju sa PC (monitorski program) koji se izvršava na razvojnom sistemu po njegovom uključenju šalje PC-u kao znak spremnosti za rad (prompt) zvezdu (*****). Komande koje ovaj progam raspoznaju su:
L - prenos programa iz PC-a u razvojni sistem.
G - startovanje prenetog programa.

Procedura:

1. i 2. Tačke 1. i 2. su identične kao u drugoj vežbi. Treba pokrenuti batch fajl MPSV3.bat čija se ikonica nalazi na dekstopu.

3. Editovati fajl **vezba3.asm** u prozoru ConTEXT editora, gde je on već i otvoren, i u njega upisati program. Fajl **vezba3.asm** već sadrži uvodne direktive, direktive za priključivanje teksta makroa i teksta deklaracije nekih registara. Izgled fajla je sledeći:

```
                $ep
                Vezba3    MODULE     main

RSEG at 1ah             ; Pocetak registarskog segmenta
;*********************************************************
;*          Ovde treba deklarisati registre koji ce se koristiti          *
;*********************************************************

CSEG at 2080h           ; Pocetak kod-segmenta
; ***********************************
; *       Ovde treba napisati program       *
; ***********************************

; **** Deo programa koji sledi obezbeđuje rad razvojnog okruženja ****
                CSEG      AT     7FFAH
                LDB       0Fh,  #11111110B;
                LJMP      2080H
                END
```

Priključeni fajl **sis_reg2.inc** sadrži tekst deklaracija registara koji se koriste u makroima. Studenti mogu koristiti ove registre, mada se preporučuje da sami deklarišu registre koje će koristiti u svom programu kako bi se izbeglo da neki makro promeni sadržaj registra. Sadržaj ovog fajla je sledeći:

```
        OFFSET:     DSW       1
        pom:        DSW       1
        mac_pom:    DSW       1
        poruka:     DSW       1
        indx:       DSW       1
        mac_bajt:   DSB       1
        port2_pom:  DSB       1
        led_pom:    DSB       1
        slovo:      DSB       1
        brojac:     DSB       1
        BAJT:       DSB       1
        CHR:        DSB       1
        SP_POM:     DSB       1
```

4. Radi jednostavnijeg prevođenja i povezivanja izrađenog programa napravljen je fajl **prevedi.bat** koji sam poziva redom asembler, linker, i hex-translator. Ako se u bilo kom koraku pojavi greška, izvršavanje batch fajla **prevedi** će se prekinuti. Pre poziva terminal-emulatora, preporučuje se da se pogledom u prozor **"Console Output"** uverite da nije bilo grešaka tokom prevođenja. Fajl **prevedi.bat** je napisan tako da u slučaju da je tokom prevođenja bilo grešaka,

batch file otvoriti **.lst** fajl (što je ustvari izveštaj asemblera o procesu asembliranja), ili **.m96** fajl (što je izveštaj linkera) u prozoru ConTEXT editora, gde je moguće ozbiljnije analizirati uzroke grešaka. Na osnovu ove analize, treba se vratiti u izvorni (source) kôd koji je u fajlu sa ekstenzijom **.asm**, i ispraviti greške koje su asembler ili linker prijavili.

5. U ovoj vežbi se za prenos programa na razvojni sistem koristi Tera Term Pro terminal-emulator koji radi u grafičkom okruženju. Studenti kojima to više odgovara, mogu koristiti i neki drugi terminal-emulator, kao što je Microsoft-ov Hyper Terminal ili neki drugi. Suštinska razlika ovih programa ne postoji. Posle pokretanja Tera Term Pro terminal emulatora treba proveriti da li je razvojni sistem "na vezi" **uključivanjem razvojnog sistema ako je bio isključen**, ili **pritiskom na reset taster,** u slučaju da je razvojni sistem već bio uključen. Ako je komunikacija od razvojnog sistema ka PC-u ispravna, na ekranu će se pojaviti sinmbol "*" (prompt). U slučaju da terminal emulator nije podešen kako treba ili u slučaju da je vod za komunikaciju u prekidu, pojaviće se neki drugi simbol, ili se neće pojaviti ništa.

6..Ako je sve u redu, tj ako se pojavio prompt (simbol "*"), potrebno je pripremiti razvojni sistem za primanje novog programa (programa koji je napisan i preveden). Da bi se to uradilo, potrebno je pritisnuti taster **L** na tastaturi (skraćenica od "Load", što znači napuni). Razvojni sistem bi trebao da odgovori sa "**Čekam fajl…**" na monitoru PC-a. Ako se ovakav odgovor ne pojavi, postoji nekakav problem u komunikaciji od PC-a ka razvojnom sistemu. Ako se ovaj odgovor pojavio na monitoru PC-a, može se početi sa slanjem **.hex** fajla.

Da bi to uradili potrebno je otvoriti **"File"** meni i u njemu odabrati opciju "**Send file…"**, dalje, naći fajl na hard disku (najverovatnije je direktorijum u kome se nalazi **.hex** fajl, koji je rezultat prevođenja kôda, po pravilu otvoren kada kliknete na opciju "Send file…"), u poddirektorijumu **student** direktorijuma **ASM96** na disku **C**.Tokom prenosa terminal emulator će na monitoru ispisivati ono što prenosi ka razvojnom sistemu (sadržaj .hex fajla). Za vreme prenosa svetleće (LED) diode razvojnog sistema trepere.

7. Na kraju prenosa, sam razvojni sistem će na svom LCD displeju ispisati **"Prenos uspesan"** ako je prenos zaista bio uspešan. Sada je samo potrebno pokrenuti program, a to se postiže pritiskom na taster **G** na tastaturi (skraćeno od "**Go"**, što je eng.reč za "kreni"). Program koji se izvršava na razvojnom sistemu se prekida resetovanjem. Ako je program neispravan, ili ako ne radi ono za šta je bio pisan, potrebno je prepraviti ga i ponoviti ovu proceduru za prenos kako bi novo-napisani program mogao da bude pokrenut na razvojnom sistemu. Zarad toga, potrebno je vratiti se u ConTEXT editor, ispraviti izvorni kôd, ponovo ga prevesti pritiskom na taster **F9**, i tada ponoviti prenos na razvojni sistem.

8. Na kraju vežbe potrebno je zatvoriti sve pokrenute programe i odjaviti se sa računara.

VEŽBA 4

- **Rad sa stringovima.**
  - Načini definisanja stringa.
  - Pristup stringu.
  - Operacije nad stringovima.
- **Primer veze racunara (PC) sa mikrokontrolerom.**
- **Upoznavanje sa serijskom vezom kroz prakticnu primenu.**
- **Razlika pristupu stringu iz glavnog programa i potprograma.**
- **Samostalna izrada, prevođenje i izvršavanje jednostavnog programa.**

ZADATAK 1:

Definisati jedan string po "Paskal" standardu i jedan po "C" standardu i ispisati ih na ekran monitora jedan ispod drugog.

Stringovi su programske konstante sastavljene od uređenog niza podataka tipa bajt. Svaki bajt niza predstavlja po jedno slovo (karakter) kodovano ASCII kodom. Pored alfa-numeričkih znakova (slova i cifre) stringovi mogu sadržati i znakove interpunkcije kao i specijalne karaktere namenjene formatiranju ispisa (prelaz u novi red, nova stranica, povratak na početak reda...).

Stringovi se definišu korišćenjem direktive za definisanje programskih konstanti tipa bajt **(DCB)**. Sintaksa ove direktive, direktno preuzeta iz priručnika (str. 23), je sledeća:

[labela:] **DCB** {izraz | string} [**,** …]

Ova direktiva definiše niz bajtova koje treba smestiti u ROM, počev od trenutne vrednosti programskog brojača adresa. **Labela** ispred direktive je neobavezna. Ako postoji dobija vrednost asemblerskog pokazivača adresa.

Svaki operand DCB direktive može biti **izraz** ili **string**. Izraz može da bude i relativan, tada vrednost u ROM upisuje linker. String je niz slova (karaktera) uokviren jednostrukim navodnicima. Operandi su odvojeni zarezima.

Direktiva DCB se sme pojaviti jedino u programskom segmentu.

Primer:

```
Cseg at 3000h
Podaci:    DCB  Duz, 'Slova', 0Ah, (28 shr 1)-1, 'A', 00001111b
Duz  equ 5
```

Počev od adrese 3000h u ROM će biti upisan sledeći niz bajtova: 5, 53h (ASCII kôd slova S),6Ch,6Fh,76h,61h, 0Ah, 13 to jest (28/2-1), 41h (ASCII kôd slova A), 0Fh. Simbol (labela) ***Podaci*** *dobiće vrednost 3000h.*

Koriste se dva standarda pomoću kojih se na neki način određuje dužina stringa:

- Po **C** standardu iza poslednjeg slova stringa se ubacuje nula (**00000000b**) koja označava kraj. Obratiti pažnj na rzliku ove nule od ASCII koda za cifru nula (30h = **00110000b**). Često se ova nula naziva **NULL** karakterom (karakter koji ima ASCII kôd 00000000b). Na primer direktiva

  ```
  Poruka1:   dcb   'Ovo je string po C standardu',0
  ```

  definiše string po C standardu sa simboličkom adresom **Poruka1**.

- Po **Paskal** standardu pre prvog slova upisuje se bajt sa podatkom o ukupnoj dužini stringa. Taj prvi bajt je sastavni deo stringa. Na primer,

  ```
  Poruka2:   dcb    33, 'Ovo je string po Paskal standardu'
  ```

  definiše string po **Paskal** standardu sa simboličkom adresom **Poruka2**. Dužinu stringa (33 u primeru) može da preračuna i asembler korišćenjem asemblerskog pokazivača adresa, na primer:

  ```
  Poruka2:   dcb    DUZINA, 'Ovo je string po Paskal standardu'
  DUZINA     equ    $-Poruka2-1
  ```

  U ovom primeru, simbol **DUZINA** dobija vrednost 33, ali je ta vrednost upisana i u ROM na adresi **Poruka2**. Tako se informacija o dužini stringa može dobiti bilo pristupanjem bajtu na adresi **Poruka2,** bilo korišćenjem simbola **DUZINA**, na primer:
  ldb brojac, #DUZINA.

1.1 Korišćenje specijalnih karaktera

Deo stringa mogu biti i karakteri koji nisu alfanumerički kao i specijalni karakteri za formatiranje. ASCII kodovi često korišćenih specijalnih karaktera su sledeći:

0	(00h)	Oznaka kraja stringa u C standardu, NULL karakter
7	(07h)	Prozvodi bip kad se pošalje na ekran, BELL
10	(0Ah)	Prelaz u novi red, često označen sa LF (*line feed*)
12	(0Ch)	Prelazak na novu stranicu (čišćenje ekrana) označen sa FF (*forme feed*)
13	(0Dh)	Povratak na početak reda često označen sa CR (*carriage return*)

Neke od specijalnih karaktera, kao što su BELL ili FF, neki terminal-emulatori ne prihvataju kao specijalne karaktere. Desktop (preporučeni terminal-emulator) prihvata.

Ako se string:

```
Strng: dcb 'Iza ove poruke novi red',10,13,0
```

koji je definisan po C standardu, ispiše na ekran, kursor će preći u novi red i vratiti se na početak tog novog reda, odmah ispod slova I. Kursor određuje mesto ispisa sledećeg slova koje stigne. Bez LF karaktera (10) sledeća poruka koja bi stigla na ekran bi se ispisala preko ove, a bez CR karaktera, sledeća poruka bi bila ispisana u novom redu ali ne od početka već iza slova d (... red). Primetiti da je NULL karakter iza karaktera za prelazak u novi red i povratak kursora. Dakle, ovi specijalni karakteri se tretitaju kao sva ostala slova u stringu iako nisu obuhvaćeni navodnicima.

1.2 Pristup pojedinačnim karakterima

Za pristup pojedinačnim karakterima u ovom slučaju može se koristiti indeksno adresiranje, na primer:

```
ldb slovo, Poruka1[red_br]
```

pod pretpostavkom da je **slovo** deklarisano kao registar tipa bajt, a **red_br** kao indeksni registar tipa reč. **Red_br** sadrži redni broj slova u stringu, počev od nule do dužine stringa, a **slovo** će posle izvršene ove instrukcije sadržati slovo iz stringa koje se nalazi na poziciji određenoj sadržajem registra **red_br**.

1.3 Inicijalizacija i korišćenje priloženog potprograma

Priloženi potprogram **putchar** šalje jedan karakter (slovo) na serijski port razvojnog sistema, a samim tim i na ektan monitora. Ulazni podatak je **chr** (registar tipa bajt) i treba da sadrži ASCII kôd slova koje se šalje na ekran.

Labela pomoću koje se potprogram poziva je **putchar**. Pre poziva, u registar **chr** treba upisati ASCII kôd slova koje se šalje na ekran. Na primer, instrukcije

```
ldb   chr, #'A'
call  putchar
```

će poslati veliko slovo A na ekran monitora. Slovo će se ispisati na poziciji određenoj trenutnim položajem kursora a kursor će se posle ispisivanja pomeriti za jedno mesto udesno.
Da bi se ispisao ceo string potrebno je u petlji pristupati jednom po jednom karakteru (slovu) iz stringa kao što je opisano ranije (u poglavlju 1.2) i pozivati potprogram za štampanje slova. Petlju treba ponoviti onoliko puta koliko slova ima string.

Izvorni kôd potprograma **putchar** ne treba pisati. On se nalazi u tekst-fajlu **potprogrami4.inc** koji je priključen izvornom kodu **v4.asm**. Ovaj kôd nije od interesa za ovu vežbu.

Ulazni registar **chr** treba deklarisati kao registar tipa bajt.

ZADATAK 2:

Izraditi potprogram pod imenom `unazad` koji bi neki string štampao počev od poslednjeg slova do prvog (unazad) i pozvati ga iz glavnog programa. Ulazni podatak treba da bude registar **`Adr_str`** i treba da sadrži adresu stringa koji treba odštampati unazad. To može biti bilo koji od stringova definisanih u zadatku 1.

Poziv potprograma čija je izrada tema ovog zadatka treba obaviti na sledeći način:

```
ld   Adr_str, #Poruka1
call unazad
```

ukoliko je potprogram namenjen stringovima definisanim po C standardu.

Prilikom izrade potprograma obratiti pažnju da se sada početna adresa stringa nalazi u registru, dakle, data je kroz sadržaj registra, a ne kao labela (koja je konstantna u toku izvršavanja programa).

Jedan od osnovnih principa indeksnog adresiranja je da ofset (veličina ipred uglastih zagrada) ne sme biti promenljiv u toku izvršavanja, dakle ne sme biti sadržaj nekog registra već konstanta. Ovo je opšti princip i važi za sve tipove mikroprocesora. U skladu sa tim principom potprogram ne može koristiti indeksno adresiranje za pristup slovima kakav je opisan u 1.2 jer se registar **`Adr_str`** ne sme upotrebiti kao ofset. Instrukcija **`ldb slovo, Adr_str[red_br]`**, iako sintaksno ispravna, ne bi radila ono što se od nje očekuje (videti priručnik, kontra-primer u poglavlju 10.6 na strani 73 i poglavlje 10.7.2, primer 3 na strani 79).

Umesto indeksnog treba upotrebiti posredno (indirektno) adresiranje. Kod indirektnog adresiranja umesto indeksnog registra koji sadrži redni broj podatka u tabeli (relativnu adresu u odnosu na početak), **kod posrednog adresiranja se koristi pokazivač koji sadrži apsolutnu adresu podatka**. Samim tim je i faza inicijalizacije drugačija. Inicijalizacija i pristup podatku u stringu bi se mogli realizovati na sledeći način:

```
ld   pokaz, Adr_str        ; inicijalizacija pokazivača
...
ldb  slovo, [pokaz]        ; pristup slovu
```

Registar **`pokaz`** mora biti deklarisan kao reč, a napredovanje kroz string može se obaviti uvećavanjem (ili umanjivanjem) sadržaja tog registra bilo pomoću **`inc`** (ili **`dec`**) instrukcije, bilo korišćenjem posrednog adresiranja sa auto post-inkrementiranjem tamo gde to odgovara.

Jedan primer kostura dela kod-segmenta za ovaj zadatak bi bio:

```
...
     ld   Adr_str, #Poruka1    ; inicijalizacija poziva potprograma
     call unazad               ; poziv potprograma
     br   $                    ; beskonačna petlja, kraj programa
```

```
;-------------------------------------------------------------------
;  POTPROGRAM UNAZAD
unazad:
      ld    pokaz, Adr_str          ;  inicijalizacija pokazivača
      petlja:
      ;************************************************
      ;         Ovde realizovati petlju za ispisivanje stringa unazad
      ;         (sa inicijalizacijom i pozivom p.prog putchar)
      ;************************************************
ret                                 ; povratak iz potprograma
;-------------------------------------------------------------------
; DEFINICIJA STRINGA
      ;*********************************
      ;     Ovde definisati string Poruka1
      ;*********************************
...
```

Ovaj deo kod-segmenta treba ubaciti u priloženi izvorni kôd **v4.asm**.

Ako se realizuje potprogram za string pisan po C-standardu, najpre treba pronaći kraj stringa (uvećavati pokazivač, i proveravati da li je slovo na koje ukazuje pokazivač NULL karakter). Potom treba preuzimati jedan po jedan podatak napredujući sa pokazivam unazad (umanjujući ga).

Za string pisan po Paskal standardu potprogram se razlikuje jedino po tome što je podatak o dužini stringa već raspoloživ pa se može odmah dodati početnoj adresi i tako pokazivač pomeriti na kraj stringa. Treba voditi računa o tome da to sabiranje mora da obavi mikroprocesor (jer su svi ti podaci promenljive, sadržaji registara) a ne asembler! Potpuno je pogrešno napisati

```
      ld    pokaz, pokaz+duz        ; POGREŠNO !!!
```

Ako bi zamenili simbole brojevima postalo bi jasno zašto ova instrukcija ne radi ono što želimo.

OKRUŽENJE:

Svaki razvojni sistem μM 196 je povezan sa dva računara tipa PC serijskom vezom. Sve poruke koje šalje razvojni sistem pojavljuju se na oba računara istovremeno. Studenti ne mogu istovremeno slati svoje programe razvojnom sistemu. Dok se jedan program šalje, drugi računar mora da čeka.

Sve razvojne alatke nalaze se na lokalnom disku C u poddirektorijumu **C:\ASM96**. Tu se nalaze programi asemblera (**asm96.exe**), povezivača (**rl96.exe**) i translatora u hex kod (**oh.exe**). U DOS prozoru ovaj direktorijum treba da bude u putanji za traženje fajlova (*path*) ili da bude tekući.

Na istom direktorijumu, u poddirektorijumu **C:\ASM96\UPUTSTVA** nalaze se i uputstva za razvojne alatke (**ASMHYPER.HLP**) i priručnik za korišćenje mikrokontrolera (**M196KC.HLP**). Ova uputstva su u windows help formatu

PC se pored alatke za razvoj programa koristi i kao tastatura i monitor razvojnog sistema. Da bi se ta uloga ostvarila na PC treba pokrenuti terminal-emulator program koji pretvara PC u tastaturu, vezanu za predjnu liniju serijske veze; i monitor, vezan za prijemnu liniju serijske veze. Sve što se

otkuca na tastaturi, kao ASCII kod se prenosi do razvojnog sistema, i sve sto razvojni sistem pošalje na serijski port pojavljuje se na monitoru računara.

Program za komunikaciju sa PC (monitorski program) koji se izvršava na razvojnom sistemu po njegovom uključenju šalje PC-u kao znak spremnosti za rad (prompt) zvezdu (*****). Komande koje ovaj progam raspoznaju su:
L - prenos programa iz PC-a u razvojni sistem.
G - startovanje prenetog programa.

Procedura:

1. Procedura je identična trećoj vežbi. Treba kliknuti na ikonu **MPSV4** i u prozoru ConTEXT editora editovati fajl **vezba4.asm** i u njega upisati program. Kada je program napisan treba pritisnuti taster **F9** koji sam poziva redom asembler, linker, hex-translator. Na kraju treba pokrenuti terminal emulator program (Tera Term Pro) čija se ikonica nalazi na desktopu.

2. Editovati fajl **vezba4.asm** i u njega upisati program. Fajl **vezba4.asm** postoji na disku i već sadrži neophodne uvodne direktive. Izgled fajla je sledeći:

```
$ep
Vezba4   MODULE    main
$INCLUDE (makroi4.inc)              ; Ubacivanje tekstova makroa

RSEG at 1ah                         ; Pocetak registarskog segmenta
$INCLUDE (sis_reg4.inc)             ; Ubacivanje teksta deklaracije sistemskih registara;
;*************************************************************
;*          Ovde treba deklarisati registre koji ce se koristiti          *
;*************************************************************

      CSEG at 2080h                 ; Pocetak kod-segmenta
; ********************************
; *       Ovde treba napisati program       *
; ********************************

$INCLUDE (potprogrami4.inc)         ;Ubacivanje teksta unapred definisanih potprograma;
DEBUG                               ; Makro obezbedjuje rad sa razvojnim okruzenjem
                                    ; Poziv je OBAVEZAN na kraju izvornog koda
END
```

VEŽBA 5

- **Primer serijske veze racunara (PC) sa mikrokontrolerom.**
  - Prijem karaktera sa računara
  - Prevođenje ASCII zapisa cifara u binarni
- **Primena paralelnog porta za povezivanje bipera.**
- **Upoznavanje sa makroima kroz primenu.**
- **Izrada potprograma za prijem brojeva do 65535 sa tastature**
- **Samostalna izrada, prevođenje i izvršavanje jednostavnih programa.**

ZADATAK 1:

Izraditi potprogram `cekaj_cif` za prijem jedne cifre sa serijskog porta sa čekanjem i proverom. Primiti karakter, proveriti da li je cifra, ako nije, samo generisati kratki bip, a ako jeste pretvoriti je u binarni kôd i smestiti u izlazni registar **`cifra`** tipa bajt.

Ovaj zadatak se sastoji iz nekoliko delova. U prvom delu treba pozvati potprogram za prijem karaktera sa serijskog porta. U drugom proveriti da li je pirimljeni karakter cifra, i generisati kratki bip ako nije (što predstavlje treći deo zadatka). Četvrti deo je prevođenje ASCII kôda cifre u binarni.

1.1 Korišćenje potprograma za prijem sa serijskog porta.

Razvojni sistem je povezan sa računarom (PC) preko standardne serijske veze RS232. Svaki taster pritisnut na tastaturi šalje ogovarajući ASCII kôd razvojnom sistemu.

Pozivom potprograma **`getchar`** čiji se izvorni kôd nalazi u priključenom fajlu **potprg5.inc** mikrokontroler prelazi u stanje čekanja da nešto stigne na serijski port. Kada se to desi, ASCII kôd karaktera koji je stigao smešta se u registar CHR tipa bajt, a potprogram se završava i vraća kontrolu glavnom programu koji ga je zvao.

Dakle, ako u nekom programu pozovemo potprogram **`getchar`**, program će čekati pritisak nekog tastera na PC-tastaturi, a po zaršetku ovog, registar **`CHR`** će sadržati ASCII kôd pritisnutog tastera.

```
call getchar      ; Čeka na pritisak tastera na PC tastaturi
                  ; Po završetku, CHR sadrži ASCII kôd ;tastera
```

Registar **`CHR`** je već deklarisan u fajlu **sis_reg5.inc**, a izvorni kôd potprograma **`getchar`** se nalazi u fajlu **potprg5.inc**. Tekstovi oba fajla su ubačeni u kostur programa **v5.asm** pomoću **`include`** direktive, dakle, već su raspoloživi i ne treba ih pisati. Sadržaji ovih fajlova se mogu videti standardnim tekst-editorom.

1.2 Provera da li je primljeni karakter cifra

ASCII kodovi cifara 0 do 9 su 30h do 39h, respektivno. Ako je karakter koji se nalazi u CHR registru u ovom opsegu, otkucana je cifra na tastaturi, ako je van opsega, otkucano je nečto drugo. Da bi se proverilo da li je u CHR cifra treba proveriti da li je njegov veći od 30h i da li je manji od 39h. Moguće je koristiti i zapis **'0'** umesto 30h i **'9'** umesto 39h čime se puno dobija na jasnoći algoritma.

Provera se može obaviti korišćenjem **`cmpb`** instrukcije i iza nje nekog od grananja za neoznačene brojeve **`bc`**, **`bh`**, **`bnh`**, ili **`bnc`**. Sintaksa instrukcije **`cmpb`** kao i tabela uslova grananja preuzeta je direktno iz priručnika (strane 97 i 129):

****

CMPB - uporedi BAJTe

Drugi BAJT operand se oduzima od prvog BAJT operanda i na osnovu rezultata postavljaju se flegovi u PSW. Rezultat oduzimanja ostaje u internom registru i nije dostupan korisniku. Fleg prenosa (C) se briše ako ima pozajmice prilikom oduzimanja, a postavlja ako pozajmice nema.

CMPB S1breg, S2baop

Flegovi se postavljaju na osnovu rezultata (S1 - S2).

Uticaj na flegove **Z, N, C, V, VT↑**

****

Posle instrukcije **CMP U, V** gde su U i V **NEOZNAČENE** veličine:

BC	skoči ako je **U ≥ V**
BH	skoči ako je **U > V**
BNH	skoči ako je **U ≤ V**
BNC	skoči ako je **U < V**
BE	skoči ako je **U = V**
BNE	skoči ako je **U ≠ V**

1.3 Generisanje tonskog signala (bipa)

Biper je minijaturni zvučnik koji daje jednoličan ton sve vreme dok mu se dovodi napajenje. Napajenje bipera se kontroliše pomoću bita 7 paralelnog porta IOPORT2 tako da kad je taj bit na nuli, biper je uključen, kad je na jedinici, biper je isključen
IOPORT2 je registar u registarskom prostoru (na adresi 07) i u njega se moze upisati željena vrednost kao u svaki drugi registar. Na primer, instrukcija:

```
LDB     IOPORT2, #10000000b
```

setuje bit7 (postavlja ga na nivo logičke jedinice) a ostale bite postavlja na nivo logičke nule. Tom naredbom bi se isključio biper (bit 7 registra IOPORT2), ali bi brisanje ostalih bita (bit 0 do bita 6) moglo da smeta pošto se drugi biti paralelnog porta koriste u druge svrhe.

Zato bi trebalo uticati samo na bit7 a ostale bite ostaviti nepromenjene što je moguće ostvariti logičkim funkcijama AND, OR i XOR. Medjutim, očitavanje izlaznih bita paralelnog porta (nužno za obavljanje logičkih funkcija) u nekim implementacijama hardvera ne daje vrednost koja je bila poslednja upisana. Zbog toga se kod svakog upisa u IOPORT2, željena vrednost upiše i u standardni registar opšte namene (za to je deklarisan **port2_pom** kao registar tipa bajt). Deklaracija se već nalazi u fajlu **sis_reg5.inc**.

Set instrukcija:

```
andb  port2_pom, #01111111b       ; samo ioport2.7 na nulu
stb   port2_pom, ioport2
```

ukjučuje biper i biper svira sve dok se ne isključi.

Set instrukcija:

```
orb port2_pom, #10000000b         ; samo port2.7 na jedinicu
stb port2_pom, ioport2
```

isključuje biper.

Generisanje pauze

Između uključivanja i isključivanja bipera potrebno je napraviti pauzu. Pauza je u ovoj vežbi realizovana pomoću makroa radi ilustracije ovog mehanizma. Tekst makroa je namerno izdvojen od teksta drugih i ubačen u kostur v5.asm kako bi se mogao analizirati. Princip rada je identičan principu rada potprograma izrađenog u okviru prve vežbe. Jedna od prednosti makroa nad potprogramom je u tome što poziv makroa može sadržati parametar (slično pozivima potprograma u višim programskim jezicima. Tako se ponuđeni makro poziva instrukcijom:

```
pauza koliko
```

i pravi pauzu približno ***koliko*** milisekundi.Tako je

```
pauza  10          ;Pauza oko 10ms
pauza  6*10        ;Pauza oko 60ms
```

Tekst ovog makroa je sledeći (već postoji, ne treba ga pisati):

```
; .....
; Makro pauza je pauza opste namene. Parametar koliko je priblizno u ms
pauza macro koliko
      local lab
      ld mac_pom, #koliko              ; pauza
      lab: ldb mac_bajt,#680
           dbnz  mac_bajt, $           ; kasnjenje od oko 1ms
           dbnzw mac_pom, lab
endm
; .....
```

1.4 Pretvaranje ASCII kôda cifre u binarni

Direktno iz ASCII kodova cifara proizilazi da se binarni kôd cifre može dobiti jednostavnim oduzimanjem 30h (ili ’0’) od ASCII kôda.

Instrukcija:

```
subb cifra, CHR, #'0'          ; umesto #'0', može se pisati i #30h
```

u bajt-registar **cifra** ubacuje binarni kôd cifre čiji je ASCII kôd u **CHR**. Registar **cifra** treba deklarisati u okviru registarskog segmenta.

1.5 Neki detalji i provera traženog potprograma

Posle prijema sa serijskog porta treba proveriti da li je primljeni karakter cifra i ako jeste u izlazni registar **cifra** upisati binarni kôd primljene cifre. Ukoliko nije cifra, pre povratka iz potprograma treba generisati kratki bip trajanja 60-100ms, a refistar **cifra** napuniti sadržajem **0FFh** kako bi to služilo kao informacija da primljeni karakter nije cifra.

Da bi se proverio rad potprograma treba izraditi glavni program sa pozivom ovog potprograma. Na raspolaganju je makro **print_bajt *podatak*** koji na ekran PC ispisuje sadržaj zadatog bajt-registra kao heksadecimalni broj.

```
print_bajt CHR                 ; šalje na ekran PC sadržaj registra CHR
```

Tekst ovog makroa nije predmet ove vežbe, zainteresovani studenti ga mogu videti u fajlu **makroi5.inc**.

Glavni program bi mogao biti, na primer:

```
petlja:
      call primi_cif
      print_bajt cifra             ; ili print_bajt CHR, ako hoćete da stampate CHR
      br petlja
```

Potprogram treba napisati iza glavnog programa jer je glavni program taj koji mora da počne od 2080h.

ZADATAK 2:

Izraditi glavni program koji bi koristio potprogram `cekaj_cif` za prijem petocifrenog broja do 65535 sa serijskog porta. Treba primati jednu po jednu cifru dok se ne pritisne taster <Enter> i koristeći predloženi algoritam izračunati primljeni broj i smestiti ga u registar **`broj`** tipa reč.

Standardni kraj unosa cifara je pritisak na taster <Enter> ASCII kôd koji se pri tom šalje mikrokontroleru je **13** decimalno (ili **0Dh**).

Da bi sa tastature uneli broj 1234 treba otkucati taster 1, pa taster 2, pa 3, pa 4 i na kraju taster <Enter>. Podaci koji se šalju ka mikrokontroleru su 31h, 32h, 33h, 34h i 0Dh, redom. Od ovih podataka treba napraviti broj 1234 (04D2h ili 0000 0100 1101 0010b) i smestiti ga u registar **`broj`** tipa reč.

Očigledan algoritam je da prvu cifru treba pomnožiti sa 1000, drugu sa 100, treću sa 10, četvrtu sa 1 i ove rezultate sabrati. Nedostatak ovog algoritma je što se ne zna unapred koliko će cifara broj imati. Zato se u praksi koristi algoritam u kome se pođe od toga da je rezultat nula, pa ukoliko stigne neka cifra rezultat se pomnoži sa 10 i doda mu se ta cifra. Dobijena vrednost postaje novi rezultat. Ako stigne nova cifra, algoritam se ponavlja sve dok se ne pritisne <Enter> taster.

Rezultat bi po ovom algoritmu za dati primer bio 1 posle prve otkucane cifre, 12 (1*10+2) posle druge, 123 (12*10+3) posle treće i 1234 (123*10+4) posle četvrte otkucane cifre. Tu bi se algoritam prekinuo jer bi mikrokontroler dobio ASCII kôd <Enter> tastera umesto pete cifre. Algoritam bi radio dobro i za brojeve sa manjim ili većim brojem cifara.

Uprošćeno, algoritam se može predstaviti na sledeći način:

rezultat ← 0
primi cifru
dok se ne pritisne <Enter>, ponavljaj petlju
    rezultat ← rezultat * 10 + cifra
    primi cifru
kraj petlje

Množenje sa 10 mora se obaviti MULU instrukcijom koja kao izlazni operand ima registar tipa long. Zbog toga je najjednostavnije promenljivu **`rezultat`** deklarisati kao long-registar. Tada bi množenje i sabiranje bi izgledalo:

```
mulu rezultat, #10
addb rezultat, cifra   ; Dodavanje cifre najnižem bajtu rezultat
addcb rezultat+1, #0   ; Dodavanje prenosa iz najnižeg bajta
                       ; Sada je u rezultat zbir i moguć je povratak u petlju
```

Ovakva petlja podrazumeva da je promenljiva **`rezultat`** inicijalizovana na 0 pre prijema prve cifre.

Kompletna petlja bi se sastojala od sledećih celina:

1. Inicijalizuj rezultat na 0
2. Poziv potprograma za prijem cifre;
3. Ako je **cifra** = 0FFh, proveri da li je **CHR**=0Dh i ako jeste izađi iz petlje, a ako nije idi na 2
4. Pomnoži rezultat sa 10 i dodaj **cifra**
5. Idi na 2

Kada se izađe iz petlje (izlaz je samo u koraku 3), primljeni broj je u nižoj reči **lon**, i može se prebaciti u registar **broj**.

Provera primljenog broja je moguća korišćenjem makroa **`print_rec` *`podatak`*** koji na ekran PC ispisuje sadržaj zadatog reč-registra kao heksadecimalni broj.

```
      print_rec broj                ; šalje na ekran PC sadržaj registra broj
```

PROCEDURA:

1. Procedura je identična trećoj i četvrtoj vežbi. Treba kliknuti na ikonu **MPSV5** i u prozoru ConTEXT editora editovati fajl **vezba5.asm** i u njega upisati program. Kada je program napisan treba pritisnuti taster **F9** koji sam poziva redom asembler, linker, hex-translator. Na kraju treba pokrenuti terminal emulator program (Tera Term Pro) čija se ikonica nalazi na desktopu.

2. Editovati fajl **vezba5.asm** i u njega upisati program. Fajl **vezba5.asm** postoji na disku i već sadrži neophodne uvodne direktive. Izgled fajla je sledeći:

```
      $ep
      Vezba5 MODULE main
$INCLUDE (makroi5.inc)        ; Ubacivanje tekstova makroa
;-------------------------------------------------------------------------------------------------------------------------
; Makro pauza je pauza opste namene. Parametar koliko je priblizno u ms
pauza macro koliko
      local lab
      ld mac_pom, #koliko           ; pauza
      lab: ldb mac_bajt,#680
            dbnz  mac_bajt, $       ; kasnjenje od oko 1ms
            dbnzw mac_pom, lab
endm
;-------------------------------------------------------------------------------------------------------------------------
```

```
        RSEG at 1ah                     ; Pocetak registarskog segmenta
        $INCLUDE (sis_reg5.inc)         ; Ubacivanje teksta deklaracije sistemskih registara;
;*********************************************************
;*          Ovde treba deklarisati registre koji ce se koristiti          *
;*********************************************************

        CSEG at 2080h               ; Pocetak kod-segmenta
; ***********************************
; *       Ovde treba napisati program       *
; ***********************************

        $INCLUDE (potprg5.inc)          ; Ubacivanje teksta unapred definisanih potprograma;
        DEBUG                           ; Makro obezbedjuje rad sa razvojnim okruzenjem
                                        ; Poziv je OBAVEZAN na kraju izvornog koda
        END
```

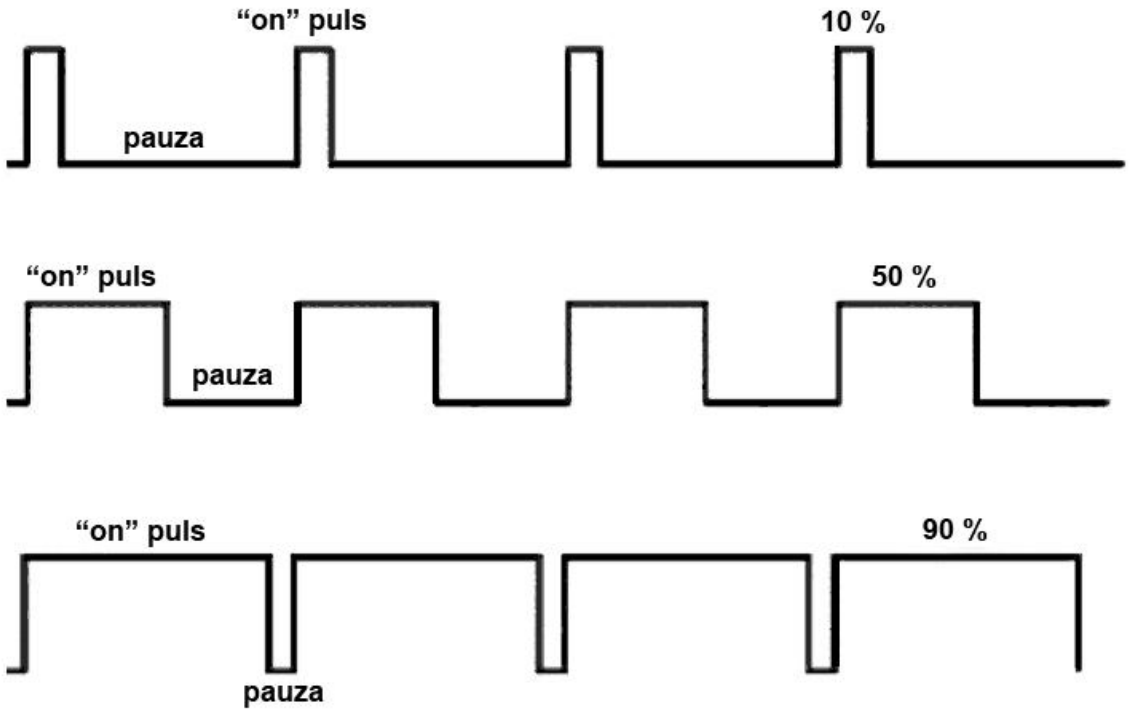
VEŽBA 6

- **Kratak uvod u PWM.**
- **Moguća primena softverskog prekida u PWM.**
  - Generisanje periodičnog softverskog prekida.
  - Izrada prekidne rutine
- **Korišćenje pasivnog paralelnog porta kao izlaza.**
- **Upoznavanje sa makroima kroz primenu.**
- **Menjanje parametara PWM**
  - Korišćenje paralelnog porta kao ulaza - tastature
- **Samostalna izrada, prevođenje i izvršavanje jednostavnog programa.**

Uvod.

Pulse Width Modulation (PWM) – Impulsno-širinska modulacija

PWM je tehnika za upravljanje kolima pomoću digitalnih izlaza procesora. Analogni signal je neprekidan signal (signal čija se vrednost-amplituda menja kontinualno), tj. čija amplituda uzima vrednosti iz beskonačnog skupa, što znači da amplituda analognog signala može biti bilo koji realan broj iz skupa, recimo {0,9}. Za razliku od analognog signala, amplituda digitalnog signala može da ima samo dve vrednosti, na primer 0V i 5V.
Zamenom analognih signala digitalnim drastično se smanjuje potrošnja energije i cena uređaja, a upravo je PWM jedan od načina na koji se to ostvaruje. Kod impulsno-širinske modulacije potrošačima se napajanje dovodi u vidu serije “on” i “off” pulseva. “On” puls je deo periode u kome izvor jednosmernog napajanja napaja potrošač, a “off”puls, ili pauza, je deo u kome je potrošač “otkačen” od izvora.
Na slici su prikazana tri različita PWM signala. Prvi signal je signal kod kojeg “on” puls traje 10% periode, kod drugog 50%, a kod trećeg 90%.



ZADATAK 1:

Napisati program koji će na svaku drugu svetleću (LED) diodu izbacivati Impulsno-širinski modulisan signal (PWM), dok će ostale držati konstanto upaljene.

Širinski modulisan signal moguće je generisati na jednom pinu pasivnog izlaznog paralelnog porta. Da bi se to realizovalo, potrebno je menjati stanje tog pina, tj. potrebno je da određeno vreme ovaj pin bude na nivou logičke jedinice, nakon čega treba da pređe u stanje logičke 0. Ako se ove promene dešavaju periodično, generisali smo širinski modulisan signal. Trajanje periode zavisi od potrošača kojim je potrebno upravljati. U ovoj vežbi, zbog jednostavnosti, izabrana je perioda od 25.6ms, što je 256 * 100 μs. Ako bi svakih 100 μs izvršavali isti kôd i brojali koliko je puta taj kôd ponovljen, i u zavisnosti od tog broja menjali stanje na izlaznom portu, mogli bismo lako realizovati PWM signal. Na primer, u prvom prolasku možemo postaviti izlaz paralelnog porta na jedinicu, u 128. prolasku ga vratiti na nulu, a u 256. prolasku resetovati brojač i sve početi ispočetka. Tako bi dobili signal koji ima odnos "on" puls - pauza 1:1 (12.8ms na jedinici i isto toliko na nuli).
Ako umesto u 128. prolasku, signal vratimo na nulu, na primer, u drugom prolasku, a sve ostalo ostane isto, dobili bi uzan impuls od 100 μs i pauzu od 255*100 μs što čini odnos impuls-pauza 1:255. Na taj način možemo dobiti signale čiji odnosi impuls-pauza variraju od 1:255 do 255:1.

Najbolji način na koji možemo saopštiti procesoru na svakih x μs da treba da izvrši određenu sekvencu je pomoću prekida (interrupt), koji bi morao u tom slučaju da se dešava periodično. Kako ranije procesori nisu imali integrisane tajmere, koristila su se eksterna integrisana kola koja su imala 16-bitni ili 8- bitni izlaz, i čiji se izlaz uvećavao za jedan (inkrementirao) posle određenog vremena. Npr. ako je tajmer bio 16-bitni, i ako se njegova vrednost uvećavala svake μs, onda je njegov "pun kug" trajao 65.536 ms. Kako sadašnji procesori (a naročito kontroleri, zbog primene u industriji i

česte potrebe za merenjem vremena) imaju integrisane tajmere, to je moguće bez mnogo petljanja sa hardverom izazivati periodičan softverski ili tajmerski prekid sa proizvoljnom dužinom periode, na jako jednostavan način.

1.1 Generisanje periodičnog prekida

Da bi kontroler generisao softverski prekid periodično, koristi se HSO izlaz. HSO (High Speed Output) je izlaz koji je kontrolisan od strane internog tajmera i od strane programera. Ovaj izlaz će u zavisnosi od vrednosti koja je upisana u registar **hso_time** koji se nalazi na adreesi 04h i 05h (2 registra dužine jedan bajt, odnosno jedan registar dužine jedne reči), generisati **HSO događaj** kada interni tajmer "odbroji" do one vrednosti koja je upisana u ovaj registar. Rezolucija ovog tajmera je 1 μs. Upisom vrednosti u registar **hso_command** definiše se komanda koju HSO treba da izvršava. Budući da je nama potreban softverski prekid, u ovaj registar je potrebno upisati vrednost **00011000b**, ili **018h**. Ovaj registar je tipa bajt, i nalazi se na adresi 06h. Pošto je po svakom izvršenom HSO događaju potrebno ponovo upisati vrednost u registre **hso_command** i **hso_time**, zgodno je koristiti makro koji bi mogao da izgleda ovako:

```
navi_sw_IRQ macro koliko
    add irq_tren, timer1, koliko ;saberi trenutnu vrednost iz timer1 sa koliko, i
                                 ; kada timer1 dođe na tu vrednost, izazovi događaj
    ldb hso_command, #018h       ; koristi interni timer1i izazovi softverski interrupt
    ld  hso_time, irq_tren       ; upiši vrednost iz irq_tren u HSO_TIME
endm
```

Potrebno je definisati reč (word) registar **irq_tren**, čiji se sadržaj upisuje u registar **hso_time.** Ovaj makro se poziva sa **navi_sw_IRQ koliko**, gde ***koliko*** može biti registar dužine jedne reči (word) ili konkretna vrednost, recimo **#100**. U slučaju: **navi_sw_IRQ #100** , za 100μs od poziva makroa HSO će generisati softverski prekid (i ništa više), što se može iskoristiti za merenje vremena. Kako je trajanje periode 256*100μs = 25ms, frekvencija je 40Hz, što je dovoljno za našu primenu (zbog tromosti oka), možemo raditi sa ovom vrednošću.

1.2 Izrada prekidnog potprograma (prekidne rutine)

Kako će se svakih 100μs generisati softverski prekid, za PWM je dalje potrebno definisati dužinu trajanja aktivnog i neaktivnog dela (on i off pulsa), kao i dužinu periode. Kako smo već usvojili da perioda traje 256 * 100μs, to za dužinu periode ne moramo koristiti poseban registar. Takođe, što se trajanja aktivnog i neaktivnog dela tiče, dovoljno je tretirati samo jedno od ova dva kao promenljivu, tj. ako definišemo trajanje on pulsa (registar **vreme_on**) jasno je da je trajanje off pulsa, tj. pauze, jednako razlici periode i vremena trajanja on pulsa.

Dalje, da bi se menjala vrednost na izlazu na svakih **vreme_on*200μs**, ili **vreme_off*200μs** potrebno je pri svakom prekidu uvećavati (inkrementirati) vrednost nekakvog brojača, koji će brojati koliko puta se ušlo u prekid, i koji treba upoređivati sa vrednošću **vreme_on**, i prema tome vršiti određen ispis na izlaz. To znači sledeće: ako je sadržaj **vreme_on=128** tada treba porediti vrednost brojača sa **vreme_on**, i proveravati da li je **brojač<vreme_on**, i ako jeste, na izlaz (pasivni paralelni port) treba ispisivati **#00000000b**, a ako nije, onda **#10101010b**. Ispis na pasivni paralelni port je detaljnije bio obrađen u vežbi 3, a kratko uputstvo za ispisivanje na ovaj port se nalazi na kraju uputstva za zadatak1, na ovoj i sledećoj strani.

U izradi programa i prekidne rutine obratiti pažnju na sledeće stvari:

- Vektor softverskog prekida se nalazi na adresi 200Ah, pa je na toj adresi potrebno upisati adresu gde se nalazi prekidni potprogram:

```
Cseg at 200Ah
      dcw softverski_prekid
```

gde je ***softverski_prekid*** labela koja pokazuje na početak rutine za softverski prekid.

- Kôd koji bi se izvršavao pri softverskom prekidu treba da ima ovakvu strukturu:

```
Softverski_prekid:
      pushf                    ; flegovi na stek, zabrana prekida
      navi_sw_IRQ #200         ; navijanje sledećeg prekida
            *
            *
         program
            *
            *
      popf                     ; flegovi sa steka u PSW, dozvola prekida
ret                            ; povratak u program
```

1.3 Korišćenje pasivnog paralelnog porta kao izlaza

Dalje, da bi se menjala vrednost na izlazu na svakih **vreme_on*200μs,** potrebno je pri svakom prekidu uvećavati nekakav brojač, koji će brojati koliko puta se ušlo u prekid, i koji treba upoređivati sa vrednošću **vreme_on** u zavisnosti u kom delu ciklusa se nalazi izlaz. Da bi se pratilo u kom delu ciklusa se nalazi izlaz, tj. da li je izlaz aktivan ili neaktivan, može se koristiti neki registar, koji može biti tipa bajt, koji bi služio kao indikator stanja u kome se nalazi izlaz, i koji bi se invertovao pri svakom ispisu na izlaz. U drugoj varijanti, može se očitavati ono što je trenutno na izlazu, i prema tome doneti sud da li brojač treba porediti sa **vreme_on** ili **vreme_off**. Naravno brojač je na početku programa neophodno obrisati, i naravno, brojač mora biti istog tipa kao **vreme_on**.

Potrošač koji ćemo ovom prilikom koristiti su svetleće (LED) diode koje su vezane na paralelni pasivni port na adresi 8000h, koje su već bile predmet vežbe. Pri tom je dioda označena sa **L0** vezana za bit 0 ovog porta, dioda **L1** za bit 1 i tako redom. Ukoliko je bit za koji je dioda vezana nula, ta dioda svetli, a ukoliko je bit za koji je dioda vezana jedinica, dioda je ugašena. Upisom odgovarajućeg bajta (čine ga biti 0 do7) na adresu 8000h, programer može po želji da pali ili gasi diode.

Podatak upisan na paralelni port ostaje do sledećeg upisa. Izlazni paralelni port se ne može očitavati. Adresa 8000h nije u registarskom prostoru pa se upis podatka na tu adresu mora obaviti preko nekog pomoćnog registra. Pored toga, izlazni paralelni port je tako realizovan da se u njega može jedino upisati podatak (očitavanje je nemoguće).

Ako se deklariše pomoćni registar, na primer, **pom_led** (tipa bajt) iz njega se ujedno može i očitati trenutno stanje svetlećih dioda (pod uslovom da se upis svakog novog stanja uvek vrši pomoću ovog registra)
Na primer, instrukcije:

```
LDB   pom_led, #00000000b
STB   pom_led, 8000h
```

će upaliti svih osam svetlećih dioda i one će ostati upaljene do sledećeg upisa na adresu 8000h.

Moguće je definisati simboličku adresu, na primer, LED (**LED equ 8000h**) tako da se ovaj simbol, definisan na jednom mestu, može koristiti umesto unošenja adrese svaki put što pored povećavanja jasnoće zapisa, olakšava eventualnu izmenu.
Tako će, na primer, instrukcije:

```
LDB   pom_led, #11111111b
STB   pom_led, LED
```

ugasiti svih osam svetlećih dioda i one će ostati ugašene do sledećeg upisa na adresu 8000h, a instrukcije:

```
LDB   pom_led, #01111110b
STB   pom_led, LED
```

će ugasiti sve diode osim L7 i L0 koje će upaliti.

Zadatak 2:

Na osnovu prethodnog programa, napisati program koji će očitavati tastere sa porta IOPORT1, i koji će na osnovu pritiska na tastere smanjivati ili povećavati odnos vreme_on / vreme_off.

2.1 Korišćenje paralelnog porta kao ulaza-tastature

Očitavanje tastera se vrši direktnim proveravanjem sadržaja porta, što znači da će instrukcija

bbc IOPORT1.2 labela

skočiti na labelu **labela** ako je taster 2 porta 1 pritisnut. Ono što se javlja kod mehaničkih tastera je takozvani baunsing (bouncing) efekat, tj. efekat odskakanja ili treperenja. Ako bi port čitali na gore navedeni način, zbog brzine izršavanja programa, očitali bi da je taster pritisnut verovatno više desetina puta, jer odskakanje traje nekoliko milisekundi, zavisno od tastera. Zbog toga je ovaj efekat neophodno anulirati, tj. izvršiti debauns (debounce). To se izvodi tako što se posle očitavanja tastera, ako je detektovan pritisak tastera, uvodi pauza od nekoliko milisekundi, nakon čega se opet očitava taster, testira da li je otpušten, a ako jeste, uvodi se još jedna pauza, jer se bouncing efekat javlja i pri pritisku i pri otpuštanju tastera. Za generisanje pauze koristićemo već napisan makro **pauza** koji izgleda ovako:

```
pauza macro koliko
      local      lab
      ld         mac_pom, #koliko      ; pauza
      lab: ldb        mac_bajt,#100
      dbnz       mac_bajt, $           ; kasnjenje od oko 1ms
      dbnzw      mac_pom, lab
endm
```

Makro **pauza** ima ulazni argument ***koliko*** koji mu se mora obezbediti. Pauza koja je dovoljna kod debounce procedure je 10ms, a kako je makro **pauza** napisan tako da generiše pauzu od ***koliko*** ms, treba ga pozvati sa argumentom 10, tj. **pauza 10.**

Kako se debounce procedura često ponavlja, korisno je napisati makro i za nju:

```
debounce macro koji
      pauza 10                      ; pauza 10ms, za smirenje treperenja (bouncing)
      bbc   koji, $                 ; ceka puštanje tastera
      pauza 10                      ; pauza 10ms, za smirenje treperenja (bouncing)
endm
```

Ovaj makro ima argument ***koji***, a on predstavlja taster koji testiramo, što znači da ga pri testiranju tastera 1 porta 1 treba pozvati na sledeći način: **debounce IOPORT1.1**, a pri testiranju tastera 2 porta 1, kao:

```
debounce IOPORT1.2.
```

U zavisnosti od toga koji je taster pritisnut, treba povećavati ili smanjivati trajanje aktivnog i neaktivnog dela periode PWM-a. Ovo se postiže dodavanjem nekog broja na aktuelni sadržaj registra **vreme_on**. Kasnije, u prekidnoj proceduri je brojač potrebno porediti sa ovom vrednošću.

PROCEDURA:

1. Procedura je identična trećoj, četvrtoj i petoj vežbi. Treba kliknuti na ikonu **MPSV6** i u prozoru ConTEXT editora editovati fajl **vezba6.asm** i u njega upisati program. Kada je program napisan treba pritisnuti taster **F9** koji sam poziva redom asembler, linker, hex-translator. Na kraju treba pokrenuti terminal emulator program (Tera Term Pro) čija se ikonica nalazi na desktopu.

2. Editovati fajl **vezba6.asm** i u njega upisati program. Fajl **vezba6.asm** postoji na disku i već sadrži neophodne uvodne direktive. Izgled fajla je sledeći:

```
$ep
Vezba6   MODULE    main
$INCLUDE (makroi6.inc)        ; Ubacivanje tekstova makroa

RSEG at 1ah                   ; Pocetak registarskog segmenta
$INCLUDE (sis_reg6.inc)       ; Ubacivanje teksta deklaracije sistemskih registara;
;*************************************************************
;*         Ovde treba deklarisati registre koji ce se koristiti          *
;*************************************************************

CSEG at 2080h                 ; Pocetak kod-segmenta
```

```
; **********************************
; *       Ovde treba napisati program       *
; **********************************

DEBUG                   ; Makro obezbedjuje rad sa razvojnim okruzenjem
                        ; Poziv je OBAVEZAN na kraju izvornog koda
END
```

Dodatak 1. Opis razvojnog sistema

Razvojni sistem na kome će se odvijati vežbe iz predmeta Mikroprocesorski Softver je bitno upoznati kako bi razumeli mogućnosti i ograničenja koja će postojati pri programiranju, kako samog mikrokontrolera i njegovih pefirerija, tako i periferija koje su vezane za mikrokontroler.

Razvojni sistem je takoreći podeljen u više "nezavisnih" celina (nezavisnih u onoj meri u kojoj je to moguće reći za jedan mikroračunarski sistem koji je organizovan oko jednog centralnog mikroprocesora, odnosno mikrokontrolera).
Sve ove celine su povezane na adresnu i/ili magistralu podataka, i na taj način je svima omogućena komunikacija sa mikrokontrolerom.
Ove celine su:

1. Adresni dekoder
2. Eksterna ROM (RAM) Memorija
3. Osmobitni Leč registri čijim se izlazima može pristupiti i sa spoljne strane kućišta
4. Inteligentni Displej (2X16 karaktera)
5. Osmobitni leč registar na koji su povezane LED
6. Serijska komunikacija sa ugrađenim kolom za prilagođavanje naponskih nivoa
7. Po tri PWM i HSO izlaza (može im se pristupiti sa spoljne strane kućišta)
8. Osam tastera koji su vezani na paralelne portove mikrokontrolera (sa pull-up otpornicima)
9. Četiri analogna ulaza (kojima se može pristupiti sa spoljne strane kućišta)
10. Biper (za zvučnu signalizaciju) koji je vezan za jedan bit porta 2

Naravno, za rad nekih od ovde nabrojanih celina koriste se same periferije mikrokontrolera, kao što je serijska komunikacija ili PWM izlazi.

O internim periferijama mikrokontrolera

- Standardni (paralelni) I/O Portovi
- Serijski I/O Port
- High-Speed Input/Output (HSIO) Jedinica
- Analogno-Digitalni Konvertor
- Pulse Width Modulator – kraće PWM (Impulsno Širinski Modulator)

Standardni I/O Portovi

80C196KC ima pet 8-bitnih I/O portova. Neki od njih su strogo ulazni, neki strogo izlazni, neki su bidirekcioni (dvosmerni), a neki, opet, podržavaju više funkcija. Port 0 je ulazni port
koji je takođe i analogni ulaz za A/D konvertor. Port 1 je kvazi-bidirekcioni port. Pinovi Porta 1 su multipleksirani sa kontrolnim signalima magistrale i dva izlaza PWM-a. Port 2 sadrži tri vrste linija: kvazi-bidirekcione, ulazne i izlazne. Neke funkcije procesora koriste ulazne i izlazne linije Porta 2. Portovi 3 i 4 su sa otvorenim drejnom bidirekcioni portovi sa otvorenim drejnom (open-drain) koji dele pinove sa adresnom i magistralom podataka.

Serijski I/O Port

Serijski I/O port je asinhroni/sinhroni port na koji je priključen UART (Universal Asynchronous Receiver and Transmitter). UART ima jedan sinhroni način (mod) rada (mod 0), i tri asinhrona moda (modovi 1, 2 i 3). Asinhroni modovi su full-duplex, što znači da može da prima i šalje podatke istovremeno. Prijemni deo je sa duplim baferom, što omogućava primanje drugog bita pre nego što je prvi očitan. Predajni deo je takođe sa duplim baferom, i može generisati kontinualnu predaju.

High-Speed Input/Output (HSIO) Jedinica

HSIO jedinica sadrži četiri zasebna periferna modula: Timer 1, Timer 2, High-Speed Input, i High-Speed Output.
Timer1 je slobodan tajmer koji se inkrementira na svako osmo stanje (stanje je vremenski interval koji je duplo duži od jedne periode eksternog oscilatora, i to je osnovna vremenska jedinica za ovaj mikrokontroler). On je vremenska osnova na kojoj se bazira rad High-Speed Input modula, i opciono se može koristiti u radu High-Speed Output modula.
Timer2 broji uzlazne i opadajuće ivice na ulazu. Može se koristiti kod High-Speed Output modula kao up/down brojač, ili kao dodatni tajmer.

High-Speed Input (HSI)

HSI modul može beležiti vremena spoljnih događaja sa rezolucijom od 8 stanja (sa oscilatorom na 16MHz to je 1μs). On može pratiti četiri ulaza od kojih svi mogu biti konfigurisani zasebno, i beležiti vrednost tajmera Timer1 kada se definisani događaj desio. Četiri tipa događaja se mogu pratiti: uzlazna ivica, silazna ivica, uzlazna i silazna ivica, ili svaka osma uzlazna ivica. HSI modul može držati do osam zabeleženih vrednosti Timer1 tajmera.

High-Speed Output (HSO)

HSO modul može izazvati osam događaja na osnovu vrednosti Timer1 i Timer2. Ovi programabilni događaji su: start A/D konverzije, resetovanje tajmera Timer1 ili Timer2, generisanje do četiri softverska vremenska zakašnjenja, i brisanje ili setovanje jednog ili više od ukupno šest HSO izlaznih linija. HSO jedinica sadrži događaje "na čekanju" i definisane trenutke u CAM (Content Addressable Memory) fajlu. Ovaj fajl sadrži do osam komandi. Svaka komanda definiše vreme akcije, vrstu akcije, potrebu za generisanjem prekida, i podatak o tome koji tajmer je referentni.

Analogno-Digitalni Konvertor

Analogno-digitalni (A/D) konvertor pretvara ulazni analogni signal u digitalni. Rezolucija je 8 ili 10 bita, vreme odabiranja (semplovanja) i konverzije su programabilni. Automatizovana A/D konverzija i smeštanje rezultata je dosta olakšana korišćenjem PTS-a (Peripheral Transfer Server). Glavni delovi A/D konvertora su kolo zadrške nultog reda, tzv. "sample and hold" kolo, 8-kanalni multiplekser, i 8-bitni ili 10-bitni A/D konvertor sa sukcesivnom aproksimacijom. On se sastoji od lestvice od 256 otpornika koji su vezani za zasebne komparatore napona. Otpornici su takvih vrednosti da obezbeđuju rezoluciju od 20mV, a dodatna preciznost se postiže pomoću kondenzatora

koji uparuju po dve linije i pomoću kojih se korak od 20mV deli na korake od po 5mV, čime se postiže 10-bitna preciznost.
Sukcesivna aproksimacija je algoritam koji je zasnovan na sledećoj ideji:
Prvo se testira ½ punog opsega A/D konvertora, što odgovara vrednosti 0111111111 b. Ako je analogni signal manji od ½ opsega, bit 10 registra u kome će se nalaziti rezultat na kraju konverzije (SAR – Successive Approximation Register) se ostavlja na vrednosti 0, i proverava se ¼ opsega (0011111111 b). Ako je signal veći od ove vrednosti, bit 9 SAR-a se postavlja na 1. Bit 8 se zatim postavlja na 0 kako bi se pripremio za sledeći test – 0101111111 b. Ova pretraga se ponavlja 10 ili 8 puta (8 za osmobitnu konverziju), posle čega se u SAR-u nalazi validan rezultat konverzije, odakle se softverksi može pročitati. Opseg A/D konvertora je 0-5V, i rezultat je jednak odnosu ulaznog (merenog) signala i punom opsegu A/D konvertora. A/D konverzija može biti pokrenuta softverski (direktno) ili je može pokrenuti HSO.

Pulse Width Modulator - PWM (Impulsno Širinski Modulator)

80C196KC ima tri PWM modula. Izlazni signal svakog od njih je puls sa promenljivom širinom koji se startuje svakih 256 ili 512 stanja (1 stanje sa oscilatorom na 16MHz traje 0,125μs), a izbor između ove dve opcije je prepušten programeru. Posle filtriranja, PWM signal će biti jednak srednjoj vrednosti signala pre filtriranja:

$$\frac{1}{T}\int_0^T f(t)dt\ .$$

Ovako je moguće generisati DC signal u opsegu od 0 – 5 V, sa rezolucijom od 256 ili 512 nivoa, s tim što je potrebno voditi računa i o prirodi potrošača koji se napaja.

O delovima i periferijama samog razvojnog sistema

Adresni dekoder

Dekoduje memorijski prostor na sledeći način:

Opseg adresa	Perfierija
0-7FFFh	Eksterni ROM i RAM
8000h-87FFh	LED (leč na čiji je izlaz vezano 8 LED)
8800h-8FFFh	Leč 1
9000h-97FFh	Leč 2
C000h-C7FFh	Inteligentni displej (2X16)

Tabela 1.1

Memorija (0000h - 7FFFh)

U memorijski prostor od 32Kb je smešten ROM modul od 32Kb, koji je preklopljen sa RAM modulom iste veličine (piggy-back).U ROM –u je smešten monitorski program koji je zadužen za prebacivanje programa koji se testira u RAM memoriju. Po završenom prebacivanju, monitorski program očitava serijski port i u slučaju da očita određenu sekvencu karaktera, on će startovati program iz RAM-a i to tako što će izbacivanjem logičke jedinice na određenu liniju jednog od paralelnih portova promeniti stanje memorijskog selektora (hardvera koji je zadužen da na **OE** ulaz RAM-a umesto na isti ulaz ROM-a dovede aktivan signal). U RESET stanju ovaj "preklopnik" selektuje ROM, čime je obezbeđeno da se po startovanju razvojnog sistema izvršava monitorski program. Monitorski program je u stanju da vrši predaju i prijem karaktera, prijem hex koda, kao i pretvaranje hex koda u mašinski kod.

LED (leč registar sa svetlećim diodama na izlazu – 8000h - 87FFh)

Na adresama 8000h do 87FFh je vezan jedan leč registar čiji je izlaz povezan na svetleće diode koje su na prednjoj strani kućišta razvojnog sistema. To znači da se pristupom bilo kojoj adresi u ovom opsegu pristupa istom leč registru (tzv. delimično dekodovanje adresnog prostora). Leč registar ostaje u istom stanju sve do narednog upisa, čime je na neki način realizovan pasivni paralelni port. Na vežbama ćemo LED diode često koristiti, i za tu namenu je u okviru inicijalizacije u .inc fajlu asemblerskom direktivom LED equ 8000h definisan simbol LED čija je vrednost 8000h. Tako je sada moguće umesto pisanja (i pamćenja) adrese na kojoj se LED leč nalazi, u instrukcijama samo pisati simbol LED (napomena: asembler nije case-senzitivan, pa su tako simboli **LED**, **LEd**, **Led** i **led** potpuno jednaki).

Dva osmobitna leč registra Leč1 i Leč2 (8800h – 8FFFh i 9000h – 97FFh)

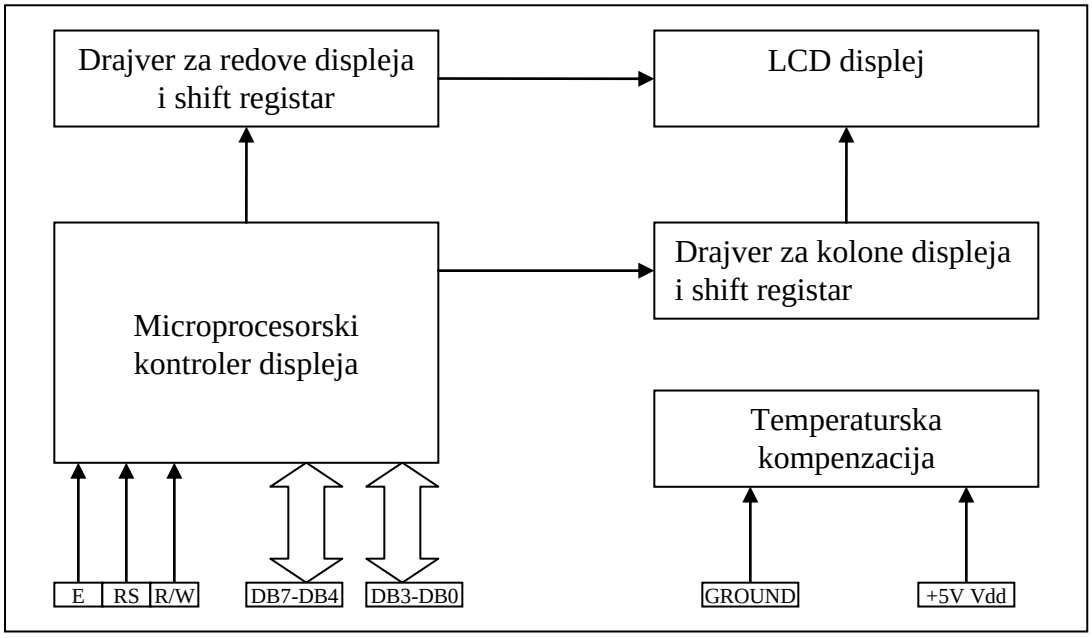
Na adresama 8800h-97FFh se nalaze dva leč registra, zauzimajući po pola od ovog celog opsega, kako je i naznačeno u tabeli adresnog dekodera; leč1 se nalazi na svim adresama od 8800h do 8FFFh, a leč 2 na adresama od 9000h do 97FFh (i u ovom slučaju je u pitanju delimično adresiranje adresnog prostora). Bitno je primetiti da su oba leč registra jednosmerna i da je u njih moguće samo upisivati sadržaj. Kako oba ova leča imaju zasebne izvode na ploči, i kako su ti izvodi spojeni na konektor koji se nalazi na zadnjoj strani kućišta razvojnog sistema, moguće je koristiti ih kao pasivne izlazne paralelne portove. Kako se oni (kao ni LED leč) naravno ne nalaze u registarskom adresnom prostoru, da bi se na njih ispisalo, potrebno je prvo neki registar napuniti vrednošću koja se želi ispisati, and onda sadržaj tog registra smestiti na jednu od ovih adresnih lokacija, čime će sadržaj biti ispisan i na izlazu izabranog leča.

Opis Inteligentnog LCD Displeja

Serija 3803 LCD displeja postoji u obliku sa 1 ili 2 linije od 16 karaktera u liniji.Veliki, laki za čitanje, karakteri su formirani u 5x7 tačkaste matrice koje dozvoljavaju prikazivanje velikih i malih slova, brojeva, simbola i znakova interpunkcije. Svaki od displeja ima poseban model karaktera zavisno od njegovog formata i dozvoljenog ugla gledanja.

Module karakterišu CMOS VLSI mikroprocesorski kontroleri koji sadrže 80 karaktera u RAM-u, koji zahtevaju generator karaktera RAM-a i ROM-a i sve signale obnavljanja, kontrole i merenja vremena. Jedino što je nužno da korisnik obezbedi je +5VDC, TTL nivoe ASCII kodova i onda mu je jednostavno rukovanje i sređivanje podataka na displeju. Prenos podataka je olakšan sa većim 4 ili 8-bitnim mikroprocesorima sa minimalnim povećanjem hardvera i sa dodatnim kontrolnim funkcijama kao što su početna pozicija kursora, adresiranje kursora, promena displeja i definisanje karaktera od strane korisnika za lakšu implementaciju.

Na slici 1.1 je predstavljena blok šema inteligentnog LCD displeja



Slika 1.1: Funkcionalna blok šema displeja

Displej ima dva registra preko kojih se njime upravlja. Jedan je statusni, a drugi je kontrolni. Pored toga, displej ima dve vrste RAM memorije, DDRAM i CGRAM. CGRAM je memorija u koju biva smešten svaki karakter ponaosob (prebačen iz ROM-a displeja po uključivanju), a DDRAM je memorija u kojoj se nalazi trenutna pozicija kursora.
Tako je, kreiranjem svojih karaktera, i njihovim smeštanjem u CGRAM, moguće displej prilagoditi potrebama uređaja u kome će displej biti primenjen, kao i potrebama podneblja gde će se taj uređaj koristiti.

Interfejs displeja sa mikroračunarskim sistemom

Interfejs serije 3803 sa mikroprocesorskom magistralom ne zahteva velike intervencije na hardveru. Bitovi podatataka 0-7 displeja direktno se vezuju na magistralu podataka, dok se za različite mikroprocesore, kontrolni i statusni registar, kao i kontrolni signali $\overline{R}$ i $\overline{W}$ sa magistralom podataka i kontrolnim linijama sprežu uz određene modifikacije hardvera. Logički nivoi RS-a (Register Select) moraju biti validni najmanje 140ns pre nego što Enable postane visok. Kako je LCD displej pravljen za direktno sprezanje sa magistralom Motorola mikroprocesora, intervencija koju je potrebno izvršiti se tiče kontrolnih linija koje, tako reći, treba naterati da se ponašaju kao kontrolne linije nekog drugog procesora. Jedan od načina da se ovo postigne je izveden i na razvojnom sistemu; linija $R/\overline{W}$ je zamenjena adresnom linijom A1, a linija RS adresnom linijom A0. Tako se sada dva registra inteligentnog displeja kojima se pristupa spolja, adresiraju kao četiri, zavisno od toga da li se u određeni registar želi pisati ili iz njega čitati.

Pisanje na displej

Kada se podatak ili komanda upisuju u modul displeja oni zahtevaju određeno vreme tzv. "vreme izvršenja". Vreme učitavanja karaktera određuje maksimalnu brzinu kojom se karakteri mogu ubacivati.Novi podatak se ne šalje dok se ne završi tekući proces.To se utvrđuje očitavanjem BUSY flag-a (Data bit 7). Za čitanje BUSY flag-a postavlja se RS=0 i $READ/\overline{WRITE} = 1$, i onda je signal ENABLE na visokom logičkom nivou. BF=1 (Busy Flag) pokazuje zauzetost kontrolera, a BF=0 - kontroler je spreman za novi proces.

Inicijalizacija displeja pri uključenju

Displej automatski izvršava inicijalizaciju (RESET) kada se uključi. Sledeće instrukcije se izvrše u toku inicijalizacije:

(1) Brisanje displeja (clear display)
    BUSY flag se drži na visokom nivou do kraja inicijalizacije (15ms)
(2) Postavljanje funkcija (function set)
    DL=1: za interfejs sa 8-bitnom magistralom podataka
    N=1: podešavanje za 2x16 displej
    F=0: za 5x7 taškasti karakter
(3) Kontrola paljenja/gašenja (display ON/OFF control)
    D=0: ugašen displej
    C=0: ugašen kursor
    B=0: ugašeno blinkanje
(4) Postavljanje ulaznog moda (entry mode set)
    I/D=1: +1(inkrementiranje)
    S=0: bez pomeranja
(5) Displej RAM podataka je selektovan.

Pošto su svi podaci korektno postavljeni displej se uključuje i podaci mogu biti ispisani. Inicijalizacija može biti nekompletirana ako vreme napajanja nije isto kao i vreme uključenja.

Sekvenca insrukcija paljenja, prikazana na sledećoj tabeli, je preporučena za sistemske rutine inicijalizacije. Spoj ovih pet koraka inicijalizacije unutar sistemskog softvera bi eliminisali ove ni u kom slučaju obavezne početne parametre, koji su nepodesni za podešavanje, a time bi se smanjilo vreme paljenja. Podešavanje parametara (N,DL,I/D,S,D,C i B) je zahtev displeja i aplikacije.

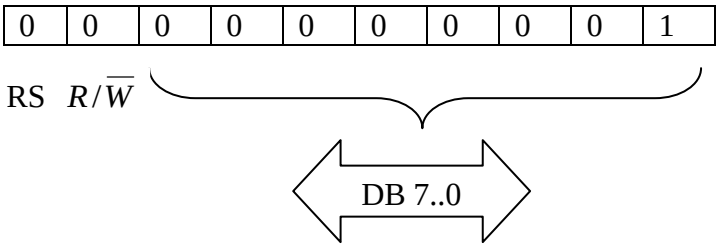
SEKVENCA PALJENJA

	Operacija	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	Čitanje BUSY flega	0	1	BF	⊢			AC			⊣
2	Postavljanje funkcija	0	0	0	0	1	DL	N	0	*	*
3	Brisanje displeja	0	0	0	0	0	0	0	0	0	1
4	Postavljanje ulaznog moda	0	0	0	0	0	0	0	1	I/D	S
5	Kontrola paljenja/gašenja	0	0	0	0	0	0	1	D	C	B

Tabela1.2. Sekvenca paljenja

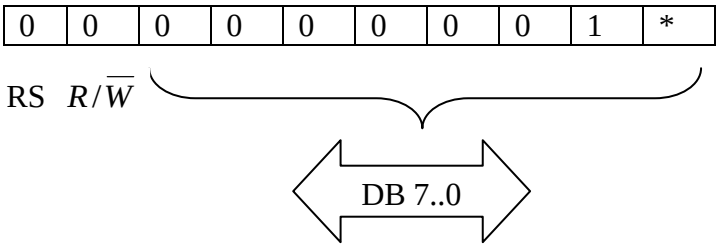
Opis instrukcija displeja

(1) **Brisanje displeja**



Upisivanje ASCII ”space” (20 HEX) u DD RAM.Kursor se vraća na Adresu 0 (ADD=”00”) i displej ako je bio promenjen vraća se na početno stanje.

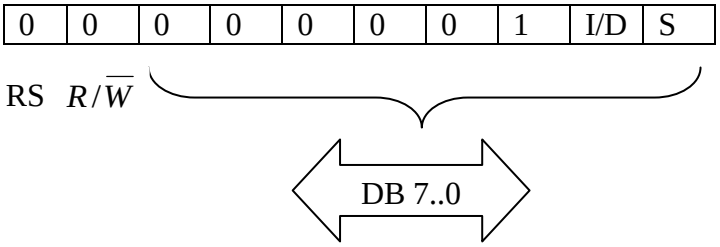
(2) Vraćanje kursora na početnu poziciju



(* = nije bitno)

Vraća kursor na Adresu 0(ADD=”0”) i displej ako je menjan vraća na početno stanje.DD RAM sadrži ostale ne izvršene promene.

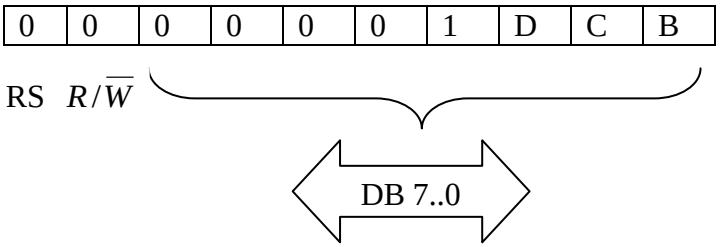
(3) Postavljanje ulaznog moda



I/D: Ovaj mod automatski inkrementira (I/D=1) ili dekrementira (I/D=0) DD RAM adresu svaki put kad se karakter upiše ili čita iz DD RAM-a.Isto se dešava kada se upisuje i/ili čita podatak iz CG RAM-A.

S (Shift): Automatski pomera ceo displej na desnu ili levu stranu posle svakog čitanja karaktera,ako je S=1,u levo ako je I/D=1 ili u desno ako je I/D=0.Zbog toga displej izgleda kao da kursor stoji i samo se karakteri na displeju pomeraju.Displej se ne menja kada se čita iz DD RAM-a ili ako je S=0.

(4) Kontrola paljenja/gašenja

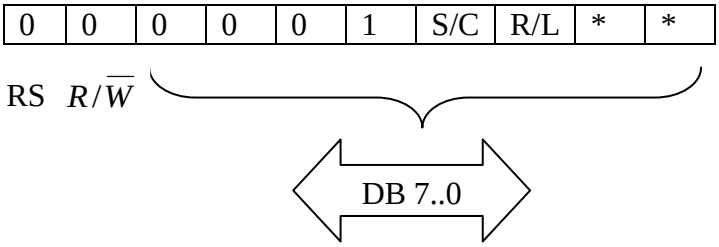


D (Display): Displej se uključuje kad je D=1,a gasi kad je D=0.Kada je displej isključen uz D=0, podaci se pamte u DD RAM-u i mogu biti prikazani odmah pošto D postane “1”.

C (Cursor): Kursor je prikazan kao underbar kada je C=1, a nema ga na displeju kada je C=0.Čak i kad kursor nije prikazan, funkcija I/D se i dalje ne menja tokom ispisa podatka.Kursor je prikazan sa 5 tačaka u osmoj liniji 5x7 matričnom karakter fontu.

B (Blink): Karakter na poziciji kursora blinka kada je B=1.Blinkanje je urađeno kao prekidanja između svih crnih tačaka i prikazanog karaktera u intervalu od 0.4 sec. Kursor i blink karakter ne mogu se videti istovremeno.

(5) Pomeranje kursora i promena displeja

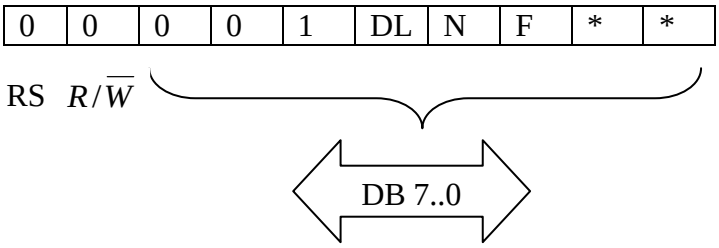


(* nije bitno)

Pomera kursor i displej u desno ili levo bez ispisa ili čitanja podatka.Ova funkcija se koristi za ispravku ili pretraživanje displeja.

S/C	R/L	
0	0	Pomera kursor u levo (AC se dekrementira za jedan).
0	1	Pomera kursor u desno (AC se inkrementira za jedan).
1	0	Pomera ceo displej u levo.Kursor prati pomeranje displeja.
1	1	Pomera ceo displej u desno.Kursor prati pomeranje displeja.

(6) Postavljanje funkcija



(* nije bitno)

DL (Data Length): Postavlja dužinu podatka.Podatak je poslat ili primljen u dužini od 8 bita (DB0 do DB7) kada je DL=1, i u dužini od 4 bita (DB4 do DB7) kada je DL=0.Kada je selektovana dužina od 4 bita podatak mora biti ispisan ili pročitan dva puta za svaki 8-bitni bajt podatka.Prvo se šalju 4 viša bita (od linije DB4 do DB7),zatim se šalju 4 preostala niža bita (od linije DB4 do DB7).

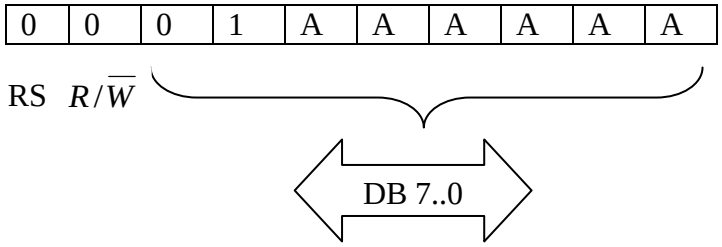
N: Postavlja kontroler za odabrani model/multiplex odnos što je prikazano na donjoj tabeli.

F (Font): Postavlja font karaktera na 5x7 tačkastu matricu.F mora biti 0.

N	F	Broj modela displeja	Font karaktera	Multiplex odnos	Format karaktera
0	0	3803-13-016	5x7 tačkasta matrica	1/8	1x16
0	0	3803-14-016			
1	0	3803-11-032	5x7 tačkasta matrica	1/16	2x16
1	0	3803-12-032			

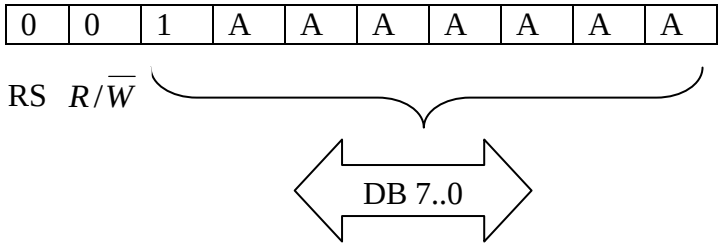
Tabela 1.3 Mogući načini komunikacije sa displejom

(7) Postavljanje adrese CG RAM-a



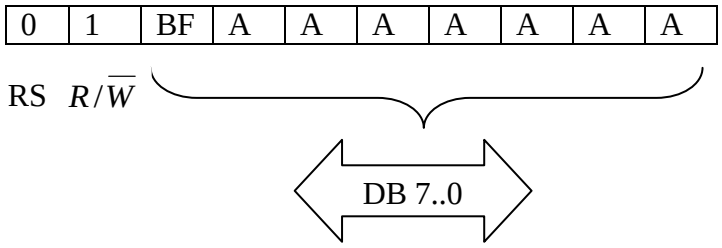
Punjenje adrese CG RAM-a (binarni broj ”AAAAAA”) u AC. Podatak koji pripada CG RAM-u je napisan ili pročitan od strane glavnog sistema pošto je komanda izvršena.

(8) Postavljanje adrese DD RAM-a



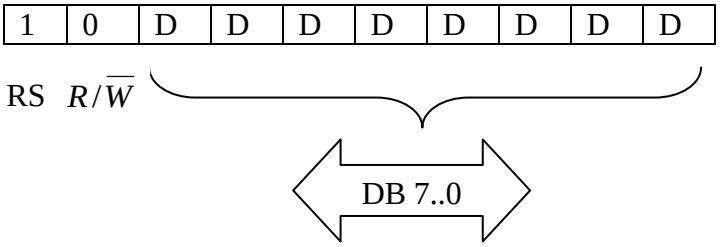
Punjenje adrese DD RAM-a (binarni broj ”AAAAAAA”) u AC.Podaci koji pripadaju DD RAM-u su ispisani na displej ili pročitani iz glavnog sistema pošto je komanda izvršena.Kada je N=1 vrednost “AAAAAAA” se kreće od “00” do “27”(HEX) za gornju liniju karaktera na displeju, a od “40” do “67” (HEX) za donju liniju karaktera na displeju.
Mada displej ima 16 karaktera u liniji adrese prikazanih lokacija zavise od toga kako je displej pomeren tj. kako su definisane instrukcije za kontrolu displeja.

(9) Čitanje Busy flag - a i adresa



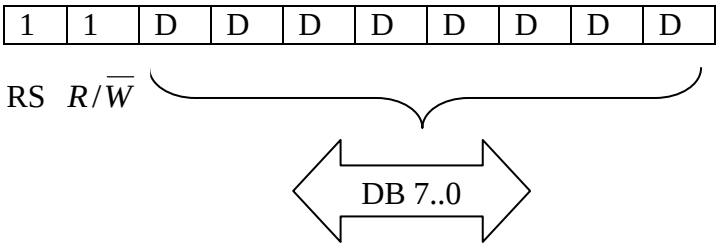
Čita Busy flag (BF). Kada je BF=1 sistem još uvek radi neku operaciju za ranije primljenu instrukciju. Sledeća instrukcija ne može biti prihvaćena dok se BF ne vrati na nulu.Provera BF je pre svakog upisivanja.
U isto vreme vrednost AC hitno šalje binarni broj “AAAAAAA”.AC se koristi i za CG i DD RAM, i te vrednosti su određene tekućom instrukcijom.Sadržaj adresa je isti kao i za (7) i (8).

(10) Ispis podatka u CG ili DD RAM



Ispisuje 8-bitni podatak “DDDDDDDD” u CG ili DD RAM. Gde će se podatak upisati određuje ranije poslata komanda.Posle upisa adresa se automatski inkrementira ili dekrementira uz saglasnost ulaznog moda. Displej se takođe pomera prateći ulazni mod. Prekid CG RAM programskog moda se izvršava ubacivanjem 'Clear display' instrukcije ili slanjem validne DD RAM adrese (adrese podatka na displeju).

(11) Čitanje podatka iz CG ili DD RAM-a



Čita 8-bitni podatak “DDDDDDDD” iz CG ili DD RAM-a. Odakle će se podatak čitati određuje ranije poslata instrukcija. Prioritet ubacivanja ove READ instrukcije u jedan od “SET CG RAM ADDRESS INSTRUCTION” ili “SET DD RAM ADDRESS INSTRUCTION” mora biti određen. Posle učitavanja podatka, adresa se automatski inkrementira ili dekrementira uz saglasnost ulaznog moda. Kako god, pomeranje displeja nije izvršeno bez obzira na podešavanje ulaznog moda.

Generator karaktera ROM (CG ROM)

CG ROM generiše 192 karaktera u obliku 5x7 tačaka za osmobitni kod karaktera. Tabela 1 pokazuje vezu između kodova karaktera i modela karaktera.Zapisi karaktera u poslednjoj koloni prostiru se i u koloni kursora i veličine su 5x8 tačaka.

Generator karaktera RAM (CG RAM)

CG RAM je RAM u koji korisnik može isprogramirati model karaktera.Osam 5x7 tačkastih karaktera može biti isprogramirano.Lista kodova karaktera prikazana na levoj strani Tabele 1.6 za prikaz modela karaktera smešta se u CG RAM.
Tabela 1.6 prikazuje vezu između adresa CG RAM-a, karakter koda (DD RAM) i izgleda karaktera.Kako je prikazano u Tabeli 1.6,bitovi podataka CG RAM-a se ne koriste za prikaz modela karaktera (markirani kao “nije bitno”) a mogu se upotrebljavati kao običan podatak RAM-a.

Tabela 1.4 Skup karaktera koji se nalaze u ROM-u displeja

Kod karaktera (DD RAM podatak)									CG RAM Adresa							Primer karaktera (CG RAM podatak)								
7	6	5	4	3	2	1	0		5	4	3	2	1	0		7	6	5	4	3	2	1	0	
←Najviši bit Najniži bit→									← Najviši bit Najniži bit →							← Najviši bit Najniži bit →								
												0	0	0		*	*	*	1	1	1	1	0	
												0	0	1					1	0	0	0	1	
												0	1	0					1	0	0	0	1	
												0	1	1					1	1	1	1	0	Primer karaktera
0	0	0	0	*	0	0	0		0	0	0	1	0	0					1	0	1	0	0	
												1	0	1					1	0	0	1	0	
												1	1	0					1	0	0	0	1	
												1	1	1		*	*	*	0	0	0	0	0	←Kursor

* nije bitno

Tabela 1.5 Generisanje karaktera

Implementacija displeja na razvojnom sistemu

Kako je interfejs između mikrokontrolera i displeja drugačiji od onoga za koji je displej predviđen (za Motorola mikroračunare), to se instrukcije navedene u tabeli 1.3 realizuju na nešto drugačiji način. Budući da je adresni prostor dekodovan na već naveden način, i da se interni registri displeja adresiraju na nešto dugačiji način, to je moguće uvesti nešto drugačiju predstavu o broju registara displeja i njihovim adresama. Naime, kako displej ima dva registra kojima se spolja pristupa, a koji se zbog implementacije na razvojnom sistemu nalaze na različitim adresama kada im se pristupa radi pisanja odnosno radi čitanja, to ih je moguće zameniti sa četiri različita registra sa sledećim adresama:

Komandni registar, u koji je moguć samo upis se nalazi na adresi 0c000h. Statusni registar iz koga je moguće samo čitanje je na adresi 0c002h. Upis podatka u DDRAM se vrši pisanjem na dresu 0c001h, a čitanje podatka iz DDRAM-a se vrši čitanjem adrese 0c001h. Po ovome je formiran i deo .inc fajla u kome se inicijalizuju simbolička imena ovih registara sa imenima, koji izgleda ovako:

```
disp equ 0c000h          ;komandni registar - upis
disp_wdata equ 0c001h    ;upis podataka
disp_read equ 0c002h     ;status registar - citanje
disp_rdata equ 0c003h    ;citanje podataka
```

U Tabeli 1.6 dat je pregled napred objašnjenih instukcija za upravljanje displejom kada je displej povezan sa nekim od Motorolinih procesora.

INSTRUKCIJE	KOD										OPIS
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
Brisanje displeja	0	0	0	0	0	0	0	0	0	1	Briše displej i vraća kursor na početnu poziciju.
Vraćanje kursora na početnu poziciju	0	0	0	0	0	0	0	0	1	*	Vraća kursor na početnu poziciju. Takođe vraća izmenjen kursor na početnu poziciju.
Postavljanje ulaznog moda	0	0	0	0	0	0	0	1	I/D	S	Postavlja upravljanje kursorom i vrstu da li će se ili neće pomerati po displeju.Ove operacije se izvršavaju u toku čitenja ili ispisa pod.
Kontrola paljenja/gašenja	0	0	0	0	0	0	1	D	C	B	Postavlja ON/OFFdispleja(D),ON/OFF kursora (C) i treptući karakter kao poziciju kursora(B).
Pomeranje kursora i promena displeja	0	0	0	0	0	1	S/C	R/L	*	*	Pomera kursor i menja displej bez promena sadržaja DD RAM.
Postavljanje funkcija	0	0	0	0	1	DL	N	F	*	*	Postavlja dužinu pod. na magistralu podataka (DL),podešava kontroler za odgovarajući model/multiplex koeficient (N) i postavlja font karaktera za5x7 tačkastu matricu.
Postavljanje Adrese CG RAM-a	0	0	0	1	A_{CG}						Postavlja CG RAM adrese.CG RAM podatak je poslat i primljen posle tog postavljanja.
Postavljanje Adrese DD RAM-a	0	0	1	A_{DD}							Postavlja DD RAM adrese.DD RAM podatak je poslat i primljen posle tog postavljanja.
Čitanje Busy flag-a i adresa	0	1	BF	AC							Čita BUSY flag kada se podatak ne prihvata,čita sadržaj adresnog brojača (AC).
Ispis podatka u CG ili DD RAM	1	0	Ispis podatka								Piše podatak u DD RAM ili CG RAM i inkrementira (ili dekrementira) AC.
Čitanje podatka iz CG ili DD RAM-a	1	1	Čitanje podatka								Čita podatak iz DD RAM-a ili CG RAM-a i inkrementira (ili dekrementira) AC.
	I/D=1: Inkrementiranje S=1:Nemoguća promena displeja S/C=1:Promena na displeju R/L=1:Pomeranje u desno DL=1: 8 bita N=1: 2 linije F=1: 5x10 tačaka BF=1: Interna operacija					I/D=0: Dekrementiranje S=0: Moguće promena displeja S/C=0: Pomeranje kursora R/L=0:Pomeranje u levo DL=0: 4 bita N=0: 1 linija F=0: 5x7 tačaka BF=0: Može prihvatiti instrukciju					Vreme izvršenja Brisanje displeja i Vraćanje kursora na početnu poziciju je $1.64ms$,svih ostalih komandi je $40\mu s$.

Tabela 1.6 Instrukcije za upravljanje displejem (za Motoroline procesore)

Iz navedenih razloga, sada će instrukcije displeja izgledati nešto drugačije u samom kôdu. Tako, na ovom razvojnom sistemu, instrukcije navedene u Tabeli 1.3 treba modifikovati na sledeći način.

Brisanje displeja se vrši upisivanjem podatka 00000001b na adresu 0c000h. Na primer, ako koristimo pomoćni registar **pomoc** , biće:

```
ldb pomoc, #00000001b      ;puni pomoćni registar podatkom
stb pomoc, disp            ;vrši upis sadržaja pomoćnog registra u komandni registar displeja
```

Vraćanje kursora na početnu poziciju se vrši upisom podatka 0000001* na adresu 0c00h,gde "*" znači da vrednost bita nije bitna. To bi, slično kao u prethodnom primeru, izgledalo ovako:

```
ldb pomoc, #00000010b      ;puni pomoćni registar podatkom
stb pomoc, disp            ;vrši upis sadržaja pomoćnog registra u komandni registar displeja
```

s tim što umesto podatka 00000010b može biti upisan i podatak 00000011b.

Postavljanje ulaznog moda se sastoji od sledećeg: u tabeli 1.3 se pod "Postavljanje ulaznog moda" može videti šta koji bit u podatku koji treba biti upisan u kontrolni registar (isto kao u prethodnim primerima) predstavlja. Bit najmanje težine, označen u tabeli sa "S" određuje da li će se po ispisu karaktera ceo displej šiftovati za jedan karakter ili ne, a bit označen sa "I/D" u tabeli, određuje da li će se kursor pomerati (ili ceo displej) šiftovati ulevo ili udesno, tj. da li će se pozicija kursora inkrementirati ili dekrementirati. Tako, na primer, ako se želi da se pozicija kursora inkrementira po ispisu karaktera, a da se ceo displej ne šiftuje pri tome, potrebno je upisati vrednost 00000110b na adresu 0c00h:

```
ldb pomoc, #00000110b
stb pomoc, disp
```

Kontrola paljenja/gašenja i upravljanje ponašanjem kursora

U tabeli 1.3 pod tačkom 4 je naznačeno koji bit u ulaznom podatku ima koju ulogu (objašnjeno u tački 4 opisa instrukcija displeja) a ovde će biti na primeru pokazano kako se isključuju kursor i blinkanje, dok se displej ostavlja uključenim, a slično je i sa ostalim bitima, tj. ostalim funkcijama. Za to je potrebno upisati vrednost 00001100b na adresu 0c00h, tj:

```
ldb pomoc, #00001100b
stb pomoc, disp
```

Pomeranje kursora i promena displeja

Pomera kursor i displej u desno ili levo bez ispisa ili čitanja podatka.Ova funkcija se koristi za ispravku ili pretraživanje displeja, i detljno je objašnjena u tački 5 glave "Opis instrukcija displeja". Radi pojašnjavanja dat je primer gde se kursor pomera za jednu poziciju unazad, dok displej miruje, što može biti korisno u praksi. Potrebno je upisati podatak 000100**b u komandni registar, npr:

```
ldb pomoc, #00010000b
stb pomoc, disp
```

Postavljanje adrese CG-RAM-a i DD-RAM-a

Za objašnjenje funkcija ove dve instrukcije, pogledati deo u kome su detaljno opisane pojedinačne instrukcije.

Za postavljanje adrese CG RAM-a potrebno je upisati adresu koja se želi postaviti, kao podatak, na adresu **disp**. Adresa treba da bude u sledećem formatu: 01XXXXXXb gde X označava bit adrese, pa je recimo za upis adrese 011101 potrebno uraditi sledeće:

```
ldb pomoc, #01011101b
stb pomoc, disp
```

Isto važi i za upis adrese u DD-RAM, s tim što adresa treba da bude u formatu 1XXXXXXXb, pa je za upis adrese 0110010b potrebo uraditi sledeće:

```
ldb pomoc, #10110010b
stb pomoc, disp
```

Čitanje Busy Flag-a

Potreba za čitanjem Busy Flag-a se javlja često, a za to je potrebo očitati statusni registar displeja, čiji bit 5 je Busy Flag, a to se izvodi na sledeći način:

```
ldb pomoc,disp_read
```

Sada, kada je vrednost status registra displeja u registru pomoc, vrednost Busy Flag-a može biti izdvojena maskiranjem:

```
andb pomoc,#00100000b
```

posle čega će u registru pomoc ostati samo vrednost Busy Flag-a na mestu bita sa težinom 5, ili je moguće odmah donositi odluku na osnovu vrednosti bita 5, čak i bez maskiranja, na sledeći način:

```
bbc pomoc,5,labela
```

Ovako će se odmah skočiti na ono mesto u programu gde je to potrebno ako je Busy Flag jednak 0.

Upis podatka u CG RAM ili DD RAM se vrši upisivanjem željenog podatka na adresu disp_wdata, kada je prethodno postavljena adresa u koju se želi upisati, i to na sledeći način:

```
ldb pomoc, #10110010b
stb pomoc, disp_wdata
```

Čitanje podatka iz CG RAM-a ili DD RAM-a

Da bi se ovo učinilo, ponovo je potrebno prethodno upisati adresu sa koje se želi čitati, a potomo očitati lokaciju **disp_rdata**.

PWM i HSO

PWM i HSO izlazi su na ploči razvojnog sistema realiozovani na sledeći način:

PWM izlazi mikrokontrolera su vezani na RC filtar (integrator), odakle je izveden izvod na spoljni deo kućišta razojnog sistema. Dva od ta tri izvoda su "upravljani" pomoću dva HSO izlaza kontrolera tako što HSO upravlja prekidačem koji zatvara strujno kolo. Time se dobija, moglo bi se reći, PWM izlaz sa promenljivom amplitudom, tj amplitudno modulisan PWM signal, a time se opet dobija izuzetno velika fleksibilnost i preciznost u radu.
Svi HSO izlazi su vezani i na konektor na spoljnoj strani kućišta i moguće im je pristupiti (moguće ih je koristiti kao klasične HSO izlaze) bez obzira što se dva gore pomenuta izlaza mogu koristiti i u sprezi sa PWM izlazima kao množači, koji filtriran PWM signal množe sa 0 i 1 kada je izlaz HSO na niskom nivou i kada je na visokom, respektivno.

Trećim PWM izlazom ne upravlja HSO izlaz na gore opisan način, već četvrti PWM izlaz , i to na sledeći način: četvrti PWM izlaz sa kontrolera je povezan na jedan prekidač direktno a na drugi preko invertora, tako da oba prekidača ne mogu biti zatvorena u isto vreme, ali uvek jedan mora biti zatvoren., Tako imamo dva izlaza jednog amplitudno modulisanog PWM signala koji su jedan sa drugim u kontrafazi.

Analogni ulazi

Četiri analogna ulaza su pristupačni sa spoljne strane kućišta kontrolera, i to preko pinova konektora 5, a vezani su sa mikrokontrolerom preko Anti-Aliasing Filtra (AAF) sa limiterom. Dalje veza ide do porta 0 mikrokontrolera koji radi kao ulaz internog A/D konvertora kontrolera. Interni A/D konvertor je 10-bitni konvertor sa sukcesivnom aproksimacijom.

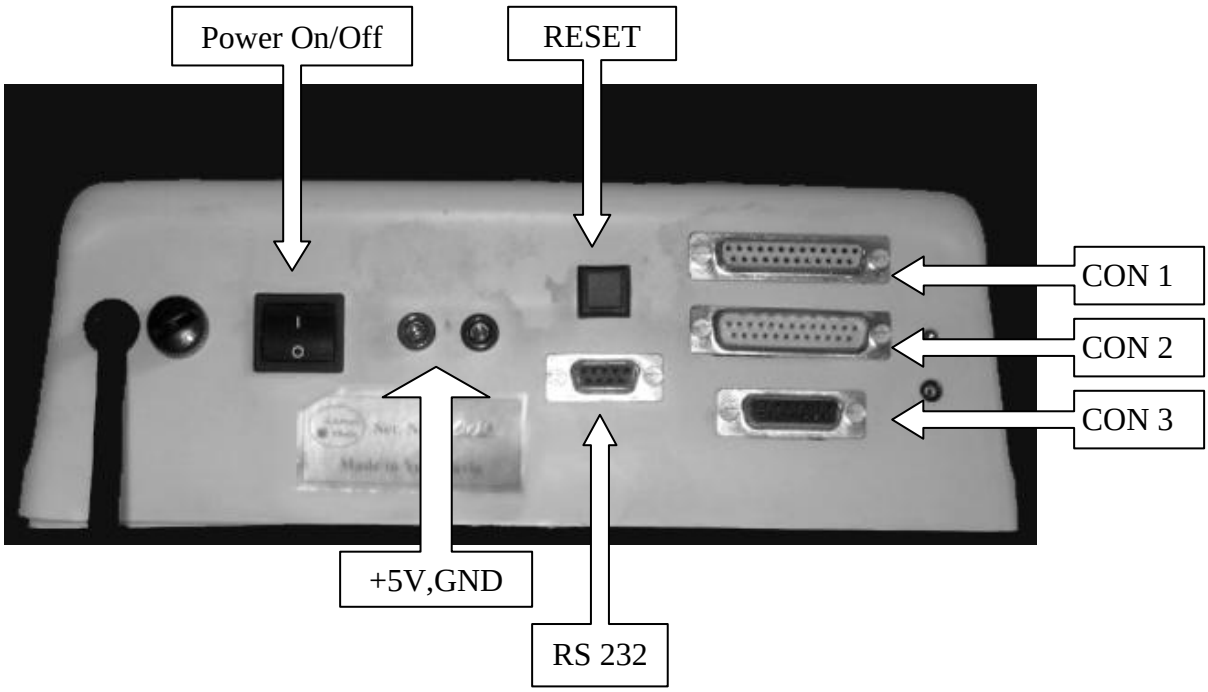
Tasteri

Osam tastera koji se nalaze sa prednje strane kućišta razvojnog sistema su povezani preko Pull-Up otpornika na pinove 1, 2, 5, 6 i 7 porta 1 i na pinove 2, 3 i 4 porta 2. Razlika između pinova ova dva porta, kada su oba konfigurisani kao ulazni, je to što port 1 ima integrisane Pull-Up otpornike a port 2 ne, pa su ti otpornici dodati na samoj ploči razvojnog sistema. To znači da svaki od ovih osam tastera, kada nije pritisnut, daje visok nivo na pinu porta na koji je vezan, dok kada je pritisnut, daje nizak nivo.

Biper

Biper je povezan na pin 7 porta 2 (koji je deklarisan kao izlazni), i aktivan je na niskom naponskom nivou, tj. pištaće onda kada je na portu 2.7 nizak naponski nivo.

Dodatak 2. Spisak spoljnih veza i blok šema razvojnog sistema



Spisak pinova konektora:

Na konektor CON1 su vezane određene linije paralelnih portova 0, 1 i 2, kao i tri HSO izlaza. Na konektor CON2 je vezan Leč1, a na konektor CON3 Leč2. Na konektoru CON4 je serijska veza (RS232).

CON1:

1. P0.0
2. P0.1
3. P0.2
4. P0.3
5. P1.1
6. P1.2
7. P1.3
8. P1.4
9. P1.4
10. P1.5
11. P1.5
12. P1.7
13. P2.2
14. P2.3
15. P2.4
16. P2.5
17. P2.6
18. P2.6
19. HSO.0
20. HSO.1
21. HSO.2
22. +5V
23. +5V
24. GND
25. GND

CON2:

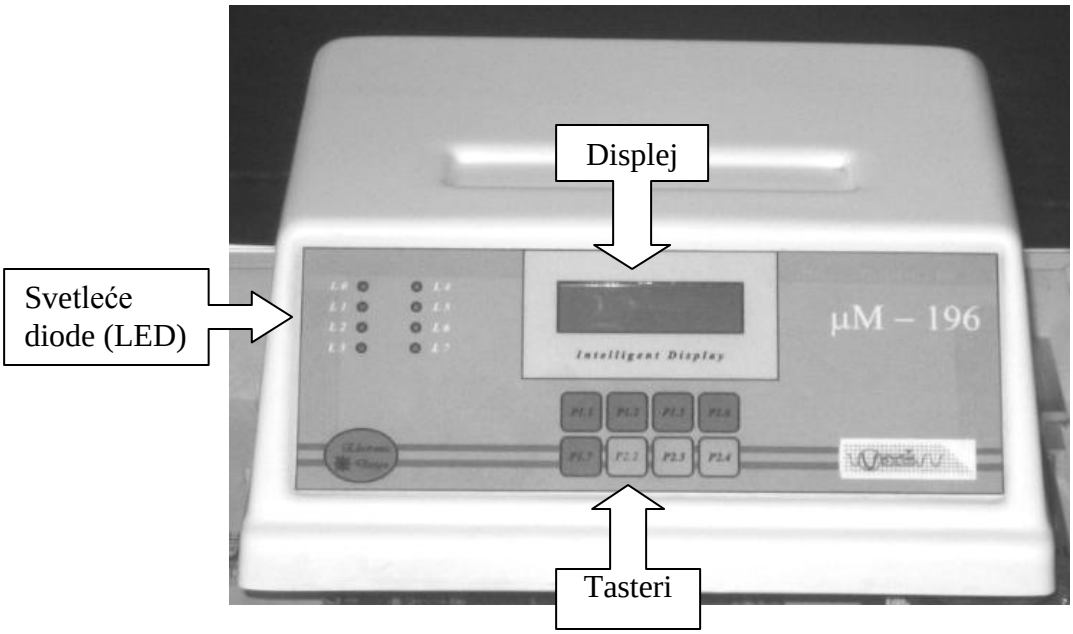
1. Leč1.D0
2. Leč1.D1
3. Leč1.D2
4. Leč1.D3
5. Leč1.D4
6. Leč1.D5
7. Leč1.D6
8. Leč1.D7
9. ------
10. ------
11. ------
12. ------
13. ------
14. +5V
15. +5V
16. +5V
17. +5V
18. ------
19. ------
20. ------
21. GND
22. GND
23. GND
24. GND
25. GND

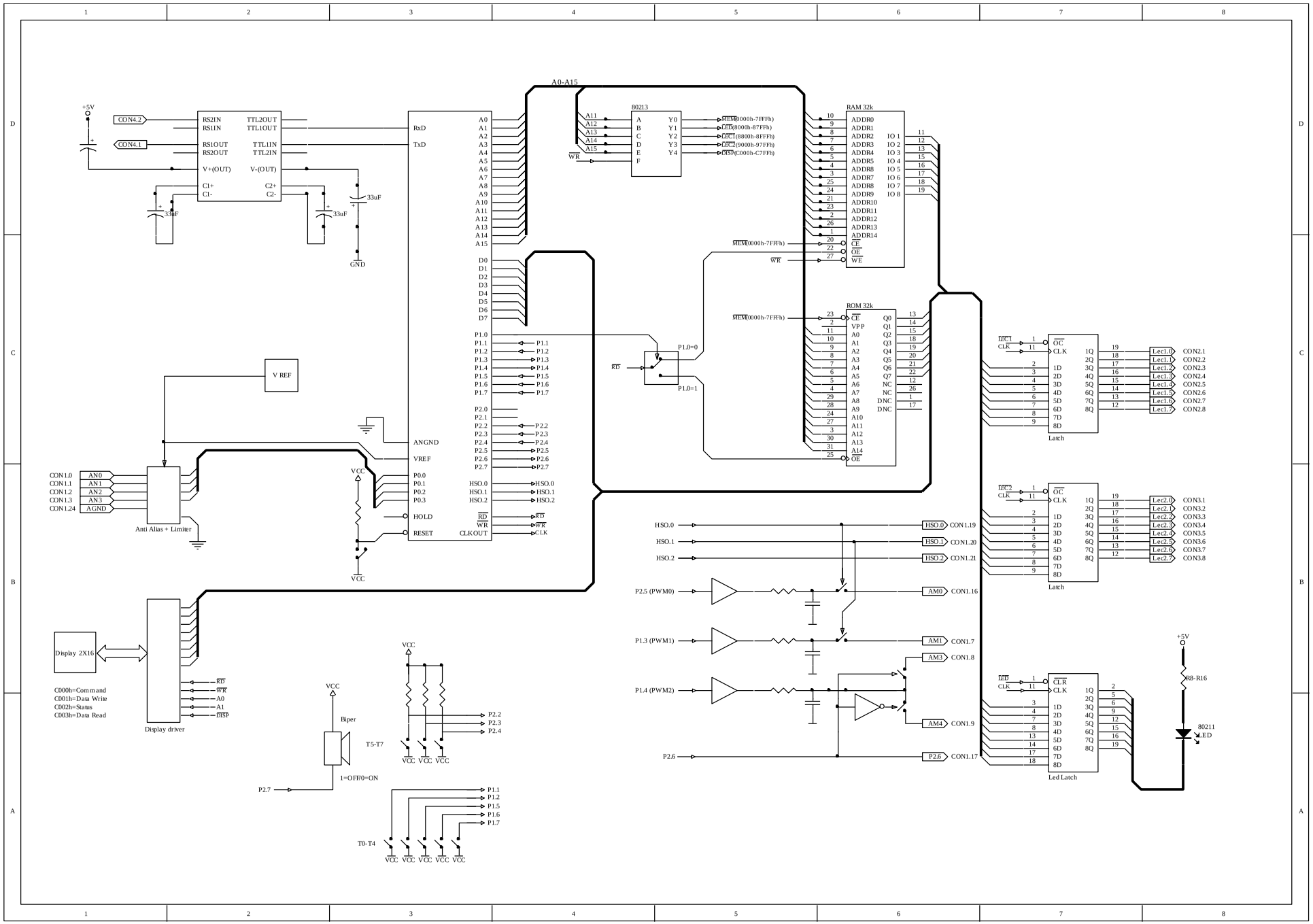
CON3:

1. Leč2.D0
2. Leč2.D1
3. Leč2.D2
4. Leč2.D3
5. Leč2.D4
6. Leč2.D5
7. Leč2.D6
8. Leč2.D7
9. +5V
10. +5V
11. +5V
12. ------
13. GND
14. GND
15. GND

CON4 (RS232):

1,4,6,8 kratko spojeni
2. Rx (mikrokontrolera)
3. Tx (mikrokontrolera)
5. GND


Displej
Svetleće diode (LED)
µM – 196
Intelligent Display
Tasteri


A0-A15
+5V
CON4.2
CON4.1
RS2IN
RS1IN
RS1OUT
RS2OUT
V+(OUT)
C1+
C1-
TTL2OUT
TTL1OUT
TTL1IN
TTL2IN
V-(OUT)
C2+
C2-
33uF
GND
RxD
TxD
ANGND
VREF
HOLD
RESET
CLKOUT
80213
RAM 32k
ROM 32k
ADDR0
ADDR1
ADDR2
ADDR3
ADDR4
ADDR5
ADDR6
ADDR7
ADDR8
ADDR9
ADDR10
ADDR11
ADDR12
ADDR13
ADDR14
IO 1
IO 2
IO 3
IO 4
IO 5
IO 6
IO 7
IO 8
MEM(0000h-7FFFh)
LEC1(8800h-8FFFh)
LEC2(9000h-97FFh)
DISP(C000h-C7FFh)
P1.0=0
P1.0=1
V REF
Anti Alias + Limiter
CON1.0
CON1.1
CON1.2
CON1.3
CON1.24
AN0
AN1
AN2
AN3
AGND
Display 2X16
Display driver
C000h=Command
C001h=Data Write
C002h=Status
C003h=Data Read
Biper
1=OFF/0=ON
T5-T7
T0-T4
Latch
Led Latch
Lec1.0
Lec1.1
Lec1.2
Lec1.3
Lec1.4
Lec1.5
Lec1.6
Lec1.7
Lec2.0
Lec2.1
Lec2.2
Lec2.3
Lec2.4
Lec2.5
Lec2.6
Lec2.7
CON2.1
CON2.2
CON2.3
CON2.4
CON2.5
CON2.6
CON2.7
CON2.8
CON3.1
CON3.2
CON3.3
CON3.4
CON3.5
CON3.6
CON3.7
CON3.8
HSO.0
HSO.1
HSO.2
CON1.19
CON1.20
CON1.21
P2.5 (PWM0)
P1.3 (PWM1)
P1.4 (PWM2)
AM0
AM1
AM3
AM4
CON1.16
CON1.7
CON1.8
CON1.9
P2.6
CON1.17
R8-R16
80211
LED

Dodatak 3. Spisak instrukcija

A. TABELA SKOKOVA POSLE POREĐENJA

Posle instrukcije CMP A, B gde su A i B **OZNAČENE** veličine:

JGE	skoči ako je **A ≥ B**
JGT	skoči ako je **A > B**
JLE	skoči ako je **A ≤ B**
JLT	skoči ako je **A < B**
JE	skoči ako je **A = B**
JNE	skoči ako je **A ≠ B**

Posle instrukcije CMP U, V gde su U i V **NEOZNAČENE** veličine:

JC	skoči ako je **U ≥ V**
JH	skoči ako je **U > V**
JNH	skoči ako je **U ≤ V**
JNC	skoči ako je **U < V**
JE	skoči ako je **U = V**
JNE	skoči ako je **U ≠ V**

B. SINTAKSE ASEMBLERSKIH DIREKTIVA I KONTROLA

Asemblerske direktive:

Ime_modula **MODULE** [Atribut] Atribut = { **MAIN | STACKSIZE(**n**)**}
EXTRN {Simbol [**:** tip_podatka]} [**,** …] Tip_podatka = { **BYTE** | **WORD** | **LONG** | **ENTRY** | **REAL** | **NULL** }
PUBLIC {Simbol} [**,** …]
{**CSEG | DSEG | RSEG | SSEG | OSEG**} [**REL | AT** Adresa]
ORG Nova_adresa
[labela:] **DCB** {izraz | string} [**,** …]
[labela:] **DCW** {izraz} [**,** …]
[labela:] **DCL** {Long_konstanta} [**,** …]
Simbol {**EQU** | **SET**} izraz [**:** tip_podatka] Tip_podatka = { **BYTE** | **WORD** | **LONG** | **ENTRY** | **REAL** | **NULL** }
[Simbol:] {**DSB | DSW | DSL | DSR**} aps_izraz
IF uslov uslov= aps_izraz [**ELSE**] **ENDIF**

Primarne kontrole izveštaja:

ERRORPRINT[**(**ime_fajla**)**]/**NOERRORPRINT**	**EP/NOEP**	**NOEP**
PAGELENGTH (n**)**	**PL**	**PL(60)**
PAGEWIDTH (n**)**	**PW**	**PW(120)**
PRINT [**(**ime_fajla**)**]/**NOPRINT**	**PR/NOPR**	**PR(**izvorni.LST**)**
SYMBOLS/NOSYMBOLS	**SB/NOSB**	**SB**
XREF/NOXREF	**XR/NOXR**	**NOXR**

Primarne kontrole objekta:

DEBUG/NODEBUG	**DB/NODB**	**NODB**
OBJECT [**(**ime_fajla**)**]/**NOOBJECT**	**OJ/NOOJ**	**OJ**(izvorni.OBJ)

Opšte kontrole:

COND/NOCOND	**CO/NOCO**	**CO**
EJECT	**EJ**	**_**
GEN/NOGEN	**GE/NOGE**	**NOGE**
INCLUDE (ime_fajla**)**	**IC**	**_**
LIST/NOLIST	**LI/NOLI**	**LI**
SAVE/RESTORE	**SA/RS**	**_**
TITLE ('string'**)**	**TT**	**TT(**ime_modula**)**

C. **KRATAK PREGLED INSTRUKCIJA**

U prilogu je data kopija pregleda svih instrukcija iz priručnika. Prva kolona je mnemonik instrukcije. Sufiks B iza mnemonika znači da je operacija osmobitna. Druga kolona prikazuje broj operanada koji slede mnemonik. Skraćen opis instrukcije je u trećoj koloni. Oznake **D** i **B** predstavljaju operande koji moraju biti u registarskom prostoru i kojima se može pristupiti jedino direktnim registarskim načinom adresiranja. **A** je operand koji može biti bilo gde u adresnom prostoru i može mu se pristupiti bilo kojim načinom adresiranja. Pretposlednja kolona pokazuje na koje flegove iz PSW deluje instrukcija.

Zagrade ukazuju na lokaciju čija se adresa nalazi u registru u zagradama. Na primer, (SP) označava lokaciju čija adresa se nalazi u registru SP. Sa **PC** je označen programski brojač, a sa **SP** pokazivač steka (REČ registar na adresi 18h). Sa **msb** i **lsb** su označeni najznačajniji i najmenje značajan bit u podatku, respektivno. **IMASK1** i **WSR** su oznake dodnatnog registra pojedinačnih dozvola prekida i registar koji određuje trenutnu vrednost registarskog "prozora".

Značenje oznaka (primedba 2) je sledeće:

✓ znači da instrukcija ili postavlja, ili briše fleg, u zavisnosti od rezultata operacije.

↑ znači da instrukcija postavlja fleg u zavisnosti od rezultata operacije ali ga nikad ne briše.

↓ znači da instrukcija briše fleg u zavisnosti od rezultata operacije ali ga nikad ne postavlja.

- (ili prazan prostor) znači da instrukcija ne utiče na fleg.

0 znači da instrukcija briše fleg.

1 znači da instrukcija briše fleg.

? znači da je stanje flega posle instrukcije nedefinisano ili da je ponašanje instrukcije u odnosu na taj fleg specifično.

Iste oznake važe i za prilog **D**

Primedbe u poslednjoj koloni su sledeće:

1. Ako se mnemonik završava sa B operacija je nad bajtima, u suprotnom nad šesnaestobitnim operandom. Operandi D, B i A moraju da zadovolje pravila lociranja u pogledu parnosti zavisno od tipa instrukcije. D i B su lokacije u registarskom prostoru; A može biti bilo gde u memoriji.
2. Značenje simbola koji opisuju uticaj na flegove je dato u prethodnom poglavlju
3. D,D+2 su uzastopne REČi u adresnom prostoru.
4. D,D+1 su uzastopne BAJTi u adresnom prostoru; D je na parnoj adresi.
5. Pretvara BAJT u REČ
6. Ofset je označeni broj (u drugom komplementu)
7. Specificirani bit je jedan od 2048 bita registarskog prostora
8. Sufiks L označava operacije nad tridesetdvobitnim podacima
9. Resetuje mikroračunar i postavlja RESET pin na nulu. Softver startuje od 2080h.
10. Asembler ne prihvata ovaj mnemonik
11. I = Dozvola prekida (PSW.9)

Mnemonik	Operandi	Operacija ([1])	Fiegovi ([2]) Z	N	C	V	VT	ST	Primedbe
ADD/ADDB	2	D = D + A	✓	✓	✓	✓	↑		
ADD/ADDB	3	D = B + A	✓	✓	✓	✓	↑		
ADDC/ADDCB	2	D = D + A + C	↓	✓	✓	✓	↑		
SUB/SUBB	2	D = D - A	✓	✓	✓	✓	↑		
SUB/SUBB	3	D = B - A	✓	✓	✓	✓	↑		
SUBC/SUBCB	2	D = D - A + C - 1	↓	✓	✓	✓	↑		
CMP/CMPB/CMPL	2	D - A	✓	✓	✓	✓	↑		
MUL/MULU	2	D,D + 2 = D * A							3
MUL/MULU	3	D,D + 2 = D * A							3
MULB/MULUB	2	D,D + 1 = D * A							4
MULB/MULUB	3	D,D + 1 = B * A							4
DIVU	2	D = (D,D + 2)/A, D + 2 = ostatak				✓	↑		3
DIVUB	2	D = (D,D + 1)/A, D + 1 = ostatak				✓	↑		4
DIV	2	D = (D,D + 2)/A, D + 2 = ostatak				✓	↑		
DIVB	2	D = (D,D + 1)/A, D + 1 = ostatak				✓	↑		
AND/ANDB	2	D = D & A (logičko ***i*** , bit po bit)	✓	✓	0	0			
AND/ANDB	3	D = B & A (logičko ***i*** , bit po bit)	✓	✓	0	0			
OR/ORB	2	D = D ili B (logičko ***ili,*** bit po bit)	✓	✓	0	0			
XOR/XORB	2	D = D (ex ili) A	✓	✓	0	0			
LD/LDB	2	D = A							
ST/STB	2	A = D							
XCH *	2	D ↔ A; D + 1 ↔ A + 1							
XCHB *	2	D ↔ A							
BMOV, BMOVI *	2	(PTR-HI)+ = (PTR-LOW)+; Dok COUNT ≠ 0							
LDBSE	2	D = A; D + 1 = Sign (A)							4, 5
LDBZE	2	D = A; D + 1 = 0							4, 5
PUSH	1	SP = SP - 2; (SP) = A							
POP	1	A = (SP); SP = SP + 2							
PUSHF	0	SP = SP - 2; (SP) = PSW; PSW=0; I=O; PSE=0	0	0	0	0	0	0	11
POPF	0	PSW = (SP) ; SP = SP + 2; I → ✓	✓	✓	✓	✓	✓	✓	11
PUSHA	0	SP=SP--2; (SP)=PSW; PSW=0000h; SP=SP–2; (SP) = IMASK1/WSR; IMASK1 = 00h; I = 0; PSE = 0	0	0	0	0	0	0	11
POPA	0	IMASK1/WSR = (SP); SP = SP + 2; PSW = (SP); SP = SP + 2	✓	✓	✓	✓	✓	✓	
SJMP	1	PC = PC + 11-bit ofset							6
LJMP	1	PC = PC + 16-bit ofset							6
BR[Indirect]	1	PC = (A)							
TIJMP *	3	PC = ([index] i MASK)2 + (Tabela)							
TRAP	0	SP = SP-2; (SP) = PC; PC = (2010h)							10
SCALL	1	SP = SP-2; (SP) = PC; PC = PC + 11-bit ofset							6
LCALL	1	SP = SP-2; (SP) = PC; PC = PC + 16-bit ofset							6
RET	0	PC = (SP); SP = SP + 2							
J (conditional)	1	PC = PC + 8-bit ofset (ako skače)							6
JC	1	Skoči ako je C = 1							6
JNC	1	Skoči ako je C = 0							6
JE	1	Skoči ako je Z = 1							6
JNE	1	Skoči ako je Z = 0							6
JGE	1	Skoči ako je N = 0							6
JLT	1	Skoči ako je N = 1							6
JGT	1	Skoči ako je N = 0 i Z = 0							6
JLE	1	Skoči ako je N = 1 ili Z = 1							6
JH	1	Skoči ako je C = 1 i Z = 0							6
JNH	1	Skoči ako je C = 0 ili Z = 1							6
JV	1	Skoči ako je V = 1							6
JNV	1	Skoči ako je V = 0							6
JVT	1	Skoči ako je VT = 1; Obriši VT					0		6
JNVT	1	Skoči ako je VT = 0; Obriši VT					0		6
JST	1	Skoči ako je ST = 1							6
JNST	1	Skoči ako je ST = 0							6
JBS	3	Skoči ako je naznačeni bit = 1							6,7
JBC	3	Skoči ako je naznačeni bit = 0							6,7
DJNZ/DJNZW	1	D = D - 1; Ako je D≠0, tada PC=PC + 8-bit ofset							6
DEC/DECB	1	D = D - 1	✓	✓	✓	✓	↑		
NEG/NEGB	1	D = 0 - D	✓	✓	✓	✓	↑		

Mnemonik	Operandi	Operacija ([1])	Fiegovi ([2]) Z N C V VT ST	Primedbe
INC/INCB	1	D = D + 1	✓ ✓ ✓ ✓ ↑	
EXT	1	D = D; D + 2 = Sign (D)	✓ ✓ 0 0	3
EXTB	1	D = D; D + 1 = Sign (D)	✓ ✓ 0 0	4
NOT/NOTB	1	D = Logičko **ne** (D)	✓ ✓ 0 0	
CLR/CLRB	1	D = 0	1 0 0 0	
SHL/SHLB/SHLL	2	C ← msb … lsb ← 0	✓ ✓ ✓ ✓ ↑	8
SHR/SHRB/SHRL	2	0 → msb ... lsb → C	✓ ✓ ✓ 0 ✓	8
SHRA/SHRAB/SHRAL	2	msb → msb ... lsb → C	✓ ✓ ✓ 0 ✓	8
NORML	2	Šiftuj nalevo dok je msb ≠ 1; D = Br. šift.	✓ ✓ 0	8
SETC	0	C = 1	1	
CLRC	0	C = 0	0	
CLRVT	0	VT = 0	0	
RST	0	PC = 2080H	0 0 0 0 0 0	9
DI	0	Zabrani sve interapte (I = 0)		
EI	0	Dozvoli interapte (I = 1)		
DPTS *	0	Zabrani sve PTS interapte (PSE = 0)		
EPTS *	0	Dozvoli PTS interapte (PSE = 1)		
NOP	0	PC = PC + 1		
SKIP	0	PC = PC + 2		
IPLPD	1	*Idle* mode ako je operand = 1; *Powerdown* mode ako je operand = 2; RESET za druge vrednosti		

* Instrukcije koje su dodate seriji MCS96 počev od modela 80c196KC pa nadalje.

D. UTICAJ INSTRUKCIJA NA FLEGOVE

Sledeća tabela daje samo pregled uticaja instrukcija na flegove. Instrukcije su date po abecednom redu radi lakšeg pronalaženja .

	Z	N	C	V	VT	ST
ADD, ADDB	✓	✓	✓	✓	↑	--
ADDC, ADDCB	↓	✓	✓	✓	↑	--
AND, ANDB	✓	✓	**0**	**0**	--	--
BMOV, BMOVI*	--	--	--	--	--	--
BR (Indirect)	--	--	--	--	--	--
CLR, CLRB	**1**	**0**	**0**	**0**	--	--
CLRC	--	--	**0**	--	--	--
CLRVT	--	--	--	--	**0**	--
CMP, CMPB	✓	✓	✓	✓	↑	--
CMPL	✓	✓	✓	✓	↑	--
DEC, DECB	✓	✓	✓	✓	↑	--
DI	--	--	--	--	--	--
DIV, DIVB,						
DIVU, DIVUB	--	--	--	✓	↑	--
DJNZ, DJNZW	--	--	--	--	--	--
DPTS	--	--	--	--	--	--
EI	--	--	--	--	--	--
EPTS	--	--	--	--	--	--
EXT, EXTB	✓	✓	**0**	**0**	--	--
IDLPD						
Ispravni operand	--	--	--	--	--	--
Neispravni operand	**0**	**0**	**0**	**0**	**0**	**0**
INC	✓	✓	✓	✓	↑	--
INCB	✓	✓	✓	✓	↑	--
JBC, JBS, JC, JE, JGE, JGT, JH, JLE, JLT, JNC, JNE, JNH, JNST	--	--	--	--	--	--
JNV	--	--	--	--	--	--

	Z	N	C	V	VT	ST
JNVT	--	--	--	--	**0**	--
JST, JV	--	--	--	--	--	--
JVT	--	--	--	--	**0**	--
LCALL, LD, LDB, LDBSE, LDBZE	--	--	--	--	--	--
LJMP	--	--	--	--	--	**?**
MUL, MULB, MULU, MULUB	--	--	--	--	--	**?**
NEG, NEGB	✓	✓	✓	✓	↑	--
NOP	--	--	--	--	--	--
NORML	✓	**?**	**0**	--	--	--
NOT, NOTB	✓	✓	**0**	**0**	--	--
OR, ORB	✓	✓	**0**	**0**	--	--
POP	--	--	--	--	--	--
POPA, POPF	✓	✓	✓	✓	✓	✓
PUSH	--	--	--	--	--	--
PUSHA, PUSHF	**0**	**0**	**0**	**0**	**0**	**0**
RET	--	--	--	--	--	--
RST	**0**	**0**	**0**	**0**	**0**	**0**
SCALL	--	--	--	--	--	--
SETC	--	--	**1**	--	--	--
SHL, SHLB, SHLL	✓	**?**	✓	✓	↑	--
SHR	✓	**0**	✓	**0**	--	✓
SHRA, SHRAB, SHRAL	✓	✓	✓	**0**	--	✓
SHRB, SHRL	✓	**0**	✓	**0**	--	✓
SJMP	--	--	--	--	--	--
SKIP	--	--	--	--	--	--
ST, STB	--	--	--	--	--	--
SUB, SUBB	✓	✓	✓	✓	↑	--
SUBC, SUBCB	↓	✓	✓	✓	↑	--
TIJMP*	--	--	--	--	--	--
TRAP	--	--	--	--	--	--
XCH*, XCHB*	--	--	--	--	--	--
XOR, XORB	✓	✓	**0**	**0**	--	--

* Instrukcije koje su dodate seriji MCS96 počev od modela 80c196KC pa nadalje.

Dodatak 4. ASCII Tabela

Decimal	Octal	Hex	Binary	Value	
000	000	000	00000000	NUL	(Null char.)
001	001	001	00000001	SOH	(Start of Header)
002	002	002	00000010	STX	(Start of Text)
003	003	003	00000011	ETX	(End of Text)
004	004	004	00000100	EOT	(End of Transmission)
005	005	005	00000101	ENQ	(Enquiry)
006	006	006	00000110	ACK	(Acknowledgment)
007	007	007	00000111	BEL	(Bell)
008	010	008	00001000	BS	(Backspace)
009	011	009	00001001	HT	(Horizontal Tab)
010	012	00A	00001010	LF	(Line Feed)
011	013	00B	00001011	VT	(Vertical Tab)
012	014	00C	00001100	FF	(Form Feed)
013	015	00D	00001101	CR	(Carriage Return)
014	016	00E	00001110	SO	(Shift Out)
015	017	00F	00001111	SI	(Shift In)
016	020	010	00010000	DLE	(Data Link Escape)
017	021	011	00010001	DC1	(XON) (Device Control 1)
018	022	012	00010010	DC2	(Device Control 2)
019	023	013	00010011	DC3	(XOFF)(Device Control 3)
020	024	014	00010100	DC4	(Device Control 4)
021	025	015	00010101	NAK	(Negative Acknowledgement)
022	026	016	00010110	SYN	(Synchronous Idle)
023	027	017	00010111	ETB	(End of Trans. Block)
024	030	018	00011000	CAN	(Cancel)
025	031	019	00011001	EM	(End of Medium)
026	032	01A	00011010	SUB	(Substitute)
027	033	01B	00011011	ESC	(Escape)
028	034	01C	00011100	FS	(File Separator)
029	035	01D	00011101	GS	(Group Separator)
030	036	01E	00011110	RS	(Request to Send)(Rec. Separator)
031	037	01F	00011111	US	(Unit Separator)
032	040	020	00100000	SP	(Space)
033	041	021	00100001	!	(exclamation mark)
034	042	022	00100010	"	(double quote)
035	043	023	00100011	#	(number sign)
036	044	024	00100100	$	(dollar sign)
037	045	025	00100101	%	(percent)
038	046	026	00100110	&	(ampersand)
039	047	027	00100111	'	(single quote)
040	050	028	00101000	(	(left/opening parenthesis)
041	051	029	00101001	)	(right/closing parenthesis)
042	052	02A	00101010	*	(asterisk)
043	053	02B	00101011	+	(plus)
044	054	02C	00101100	,	(comma)
045	055	02D	00101101	-	(minus or dash)
046	056	02E	00101110	.	(dot)
047	057	02F	00101111	/	(forward slash)
048	060	030	00110000	0	
049	061	031	00110001	1	
050	062	032	00110010	2	
051	063	033	00110011	3	
052	064	034	00110100	4	

Decimal	Octal	Hex	Binary	Value	
053	065	035	00110101	5	
054	066	036	00110110	6	
055	067	037	00110111	7	
056	070	038	00111000	8	
057	071	039	00111001	9	
058	072	03A	00111010	:	(colon)
059	073	03B	00111011	;	(semi-colon)
060	074	03C	00111100	<	(less than)
061	075	03D	00111101	=	(equal sign)
062	076	03E	00111110	>	(greater than)
063	077	03F	00111111	?	(question mark)
064	100	040	01000000	@	(AT symbol)
065	101	041	01000001	A	
066	102	042	01000010	B	
067	103	043	01000011	C	
068	104	044	01000100	D	
069	105	045	01000101	E	
070	106	046	01000110	F	
071	107	047	01000111	G	
072	110	048	01001000	H	
073	111	049	01001001	I	
074	112	04A	01001010	J	
075	113	04B	01001011	K	
076	114	04C	01001100	L	
077	115	04D	01001101	M	
078	116	04E	01001110	N	
079	117	04F	01001111	O	
080	120	050	01010000	P	
081	121	051	01010001	Q	
082	122	052	01010010	R	
083	123	053	01010011	S	
084	124	054	01010100	T	
085	125	055	01010101	U	
086	126	056	01010110	V	
087	127	057	01010111	W	
088	130	058	01011000	X	
089	131	059	01011001	Y	
090	132	05A	01011010	Z	
091	133	05B	01011011	[	(left/opening bracket)
092	134	05C	01011100	\	(back slash)
093	135	05D	01011101	]	(right/closing bracket)
094	136	05E	01011110	^	(caret/cirumflex)
095	137	05F	01011111	_	(underscore)
096	140	060	01100000	`	
097	141	061	01100001	a	
098	142	062	01100010	b	
099	143	063	01100011	c	
100	144	064	01100100	d	
101	145	065	01100101	e	
102	146	066	01100110	f	
103	147	067	01100111	g	
104	150	068	01101000	h	
105	151	069	01101001	i	
106	152	06A	01101010	j	
107	153	06B	01101011	k	
108	154	06C	01101100	l	
109	155	06D	01101101	m	

Decimal	Octal	Hex	Binary	Value	
110	156	06E	01101110	n	
111	157	06F	01101111	o	
112	160	070	01110000	p	
113	161	071	01110001	q	
114	162	072	01110010	r	
115	163	073	01110011	s	
116	164	074	01110100	t	
117	165	075	01110101	u	
118	166	076	01110110	v	
119	167	077	01110111	w	
120	170	078	01111000	x	
121	171	079	01111001	y	
122	172	07A	01111010	z	
123	173	07B	01111011	{	(left/opening brace)
124	174	07C	01111100	|	(vertical bar)
125	175	07D	01111101	}	(right/closing brace)
126	176	07E	01111110	~	(tilde)
127	177	07F	01111111	DEL	(delete)

Dodatak 5. Tera Term Pro Terminal emulator

Opis korisničkog interfejsa

Tera Term Pro terminal emulator se odlikuje velikim brojem opcija koje se mogu podesiti i koje se zatim mogu sačuvati u fajl za inicijalizaciju, kako bi se program startovao sa vrednostima koja su ranije određena i koja odgovaraju korisniku.
Ova podešavanja je moguće izvršiti korišćenjem menija iz zaglavlja glavnog prozora.

Spisak menija i opcija koje se mogu podesiti kroz menije:

[File] menu

New Connection – pokreće novu "vezu" (konekciju). Po izboru ove opcije otvara se prozor u kome je ponuđen izbor mogućih, tj. preostalih "veza". Moguće je uspostaviti vezu sa udaljenim računarom preko TCP/IP protokola, kao i preko serijskog porta. Da bi se uspostavila veza preko serijskog porta, potrebno je "izabrati" (čekirati) opciju **"Serial"** i izabrati preko kog **Com** porta želite da uspostavite vezu. U slučaju da je veza sa udaljenim računarom već ostvarena preko nekog **Com** porta, pri otvaranju nove konekcije ovaj **Com** port neće biti ponuđen. Da bi se uspostavila veza sa razvojnim sistemom za **µC196** potrebno je izabrati onaj **Com** port **PC** računara za koji je razojni sistem vezan (po pravilu je to **Com2**).

Log… – izborom ove opcije biće Vam ponuđeno da izaberete fajl u kome želite da terminal emulator zapisuje ono što prima za vreme trajanja konekcije:

Binary – Ako je odabrana ova opcija, primljeni karakteri će biti zapisani u log fajl bez ikakvih izmena. U suprotnom, karakteri komandnog tipa (novi red) će biti konvertovani , a escape sekvence će biti izbačene.

Append – Ako je odabrana ova opcija, i log fajl koji ste odabrali već postoji, primljeni karakteri će biti "zalepljeni" na kraj fajla. U suprotnom (ako "Append" opcija nije odabrana), zapisivanje u log fajl će se vršiti od početka, i njegov prethodni sadržaj će biti obrisan.

"Log" dijalog prozor (prikazan je samo dok se vrši upis u log fajl)

Close – Prekida upis u log fajl i zatvara ga.

Pause/Start – Pauzira/ponovo započinje upis u log fajl

Send File – Pokreće dijalog prozor za izbor fajla koji želite da pošaljete na udaljeni "računar" (u našem slučaju, razvojni sistem MCS196) bez ikakvog standardizovanog protokola za kontrolu toka i kontrolu integriteta prenetih podataka. Po izboru fajla, potrebno je pritisnuti taster **Open** i prenos će biti započet.

Transfer – pokreće prenos fajla koristeći neki od protokola (koji Vi izaberete) za kontrolu toka prenosa podataka.

Change directory – Podešavanje direktorijuma koji će ubuduće automatski bivati otvaran kada se pokrene prenos fajla. Ova opcija je prilično zgodna u slučaju kada se prenosi program koji treba da bude testiran na razvojni sistem, jer, kako je reč o programu koji se **testira**, svakako će biti potrebe za čestim prepravkama i ponovnim slanjem, a da ne biste gubili vreme kod svakog slanja programa na razvojni sistem, upišite u prazno polje sa imenom **New dir:** putanju do direktorijuma u kome se nalazi program koji pišete; po pravilu je to **C:\ASM96\student.**

Print – Pokreće se štampanje sadržaja glavnog prozora terminal emulatora.

Disonnect – Prekida se aktivna konekcija (veza).

[Edit] menu

Copy – Kopira prethodno označen tekst iz glavnog prozora terminal emulatora u memoriju (clipboard).

Copy table - Kopira prethodno označen tekst iz glavnog prozora terminal emulatora u memoriju, s tim da ako se tekst sastoji od reči, svaki razmak između reči će biti konvertovan u Tab karakter. Ako na ovaj način kopirate tekst, možete ga prebaciti u Excel tabelu.

Paste – Tekst koji se nalazi u memoriji (clipboard), tj. tekst koji je ranije bio Kopiran pomoću naredbe "Copy", biva poslan udaljenom računaru (u našem slučaju, razvojom sistemu).

Paste<CR> - Isto kao i prethodna opcija, s tim da na kraju teksta, program ubacuje komandni karakter za prelazak u novi red.

Clear screen – Ova komanda briše sadržaj glavnog prozora terminal emulatora.

Clear buffer – Ova komanda briše sadržaj scroll bafera (tj. onaj sadržaj koji se u datom trenutku nije nalazio na ekranu, a bio je ranije primljen), a takođe briše i onaj sadržaj glavnog prozora koji se nalazio na ekranu.

[Setup] menu

Terminal... – Izborom ove opcije, otvara se prozor "Terminal setup" u kome se mogu podesiti parametri koji su u vezi sa emulacijom terminala:

Terminal size – Stvarna veličina terminala (broj redova i kolona). Veličina prozora i veličina terminala ne moraju da budu jednake.

Term size = win size – Ako je ova opcija izabrana, program će u svakom trenutku držati veličinu terminala jednakom veličini prozora.

Auto window resize - Izborom ove opcije osiguraćete da se veličina prozora menja kako se menja veličina terminala.

New-line (receive) – Ako je izabrana opcija CR+LF, primljeni CR karakteri (komandni karakteri za prelaz u novi red) će biti konvertovani u CR+LF (novi red+početak reda).

New-line (transmit) - – Ako je izabrana opcija CR+LF, CR karakteri koji trebaju da budu poslati će biti konvertovani u CR+LF.

Terminal ID – Opcija za identifikaciju terminala na zahtev host računara.

Local echo – Omogućava tzv. Lokal eho, tj. ispis karaktera sa korisničke tastature (i uopšte karaktera koji se šalju (na ekranu terminal emulatora (ako ova opcija nije odabrana, na ekranu terminal emulatora će se povideti samo karakteri koje terminal prima)

Answerback – string koji će biti poslat hostu na prijem ENQ (05h) karaktera.

Auto switch (VT<->TEK) – Dozvoljava automatsku izmenu između VT i TEK emulacije.

Window... – Izborom ove opcije, otvara se prozor "Window setup" u kome se mogu podesiti parametri koji su u vezi sa samim prozorom terminal emulatora:

Title – Tekst koje će biti ispisan kao ime prozora.

Cursor shape – Oblik kursora: Blok, Vertikalna ili Horizontalna linija.

Hide title/menu bar – Ako su ove opcije izabrane, deo prozora gde se nalazi ime/meniji neće biti prikazan. Tada se meniji pojavljuju ako držeći Ctrl tater na tastaturi pritisnut, pritisnete levi taster miša.

Full color – Omogućava emulaciju u "punom koloru", koristeći bold (blink) atribut kao kolor atribut za tekst (pozadinu), i povećava broj boja sa 8 na 16.

Scroll buffer – Omogućava rad Scroll bafera. Ako je rad omogućen, broj linija (koji se računa uključujući i ono što je trenutno prikazano u prozoru terminal emulatora) može biti dodatno određen. Maksimalan broj je 10000.

Color

Text – Boja teksta

Background – Boja pozadine

Attribute – Atribut karaktera (normal, bold ili blink,tj. trepćući) čije boje će biti podložne promeni

Reverse – Boja teksta postaje boja pozadine, i obrnuto

Font… – Izborom ove opcije, otvara se prozor "Font setup" u kome se mogu podesiti parametri koji su u vezi sa fontom teksta koje će terminal emulator prikazivati, i uopšte, sa načinom prikaza teksta

Font – Dijalog prozor za izbor fonta. Treba Izabrati font sa liste.

Size – Veličina slova

Enable bold style – Kada je ova opcija uključena, dozvoljeno je da karakteri koji su primljeni kao bold, budu i i spisani kao bold.

Napomena: Tera Special font, koji se nalazi u paketu sa programom ne može biti biran u ovom dijalog prozoru. On biva izabran automatski od strane programa.

Keyboard… – Izborom ove opcije, otvara se prozor "Keyboard setup" u kome se mogu podesiti parametri koji su u vezi sa tastaturom računara i pojedinim njenim tasterima, koje terminal emulator može da tumači na različite načine.

Transmit DEL by:

Backspace key – Ako je ova opcija uključena, pritisak na "Backspace" taster će poslati DEL karakter (ASCII 0x7f), a u suprtonom BS karakter (ASCII 0x08)

Delete key – Ako je ova opcija uključena, pritisak "Delete" tastera će biti poslat kao DEL karakter, a u suprotnom, funkcija "Delete" tastera je određena posebnim setup fajlom za tastaturu

Meta key – Dozvoljava se upotreba meta-key načina rada, a "Alt" taster se koristi kao meta taster. Npr. Sekvenca ESC A se šalje istovremenim pritiskom na tastere "Alt" i A.

Serial port… – Izborom ove opcije, otvara se prozor "Serial port setup" u kome se mogu podesiti parametri koji su u vezi sa komunikacijom preko serijskog porta.

Port – Serijski port koji se koristi.

Data, Stop, Baud rate, Flow control, Parity – Parametri vezani za serisjku komunikaciju. Treba izabrati odgovarajuće parametre. U našem slučaju, to su:

Baud rate – 19200
Data – 8 bit
Parity – none
Stop – 1 bit
Flow control – hardware

Transmit delay – Vremenski interval u milisekundama između slanja karaktera/linija.

General…– Izborom ove opcije, otvara se dijalog prozor "General setup" u kome se mogu podesiti sledeći parametri:

Default port – Tip porta koji će se ubuduće podrazumevati kada se startuje terminal emulator

Language – Izbor jezika za emulaciju, tj. karakter-seta (Engleski, Ruski ili Japanski)

Save setup... – Pokretanjem ove komande opcije koje su podešene u prethodnim koracima će biti sačuvane u Tera Term setup fajl. U prozoru koji se otvori nakon pokretanja ove komande treba uneti novo ili izabrati neko od već postojećih setup fajl imena. Ako fajl već postoji biće prebrisan. Ako izostavite ekstenziju, ekstenzija .INI se podrazumeva.

Restore setup… - Pokretanjem ove komande otvoriće se prozor u kome treba izabrati fajl u kome se sadržana podešavanja koja đelite da koristite. Ako izostavite ekstenziju, u prozoru će biti izlistani samo fajlovi sa ekstenzijom .INI.

Load key map… - Pokretanjem ove opcije biće otvoren prozor u kome treba odabrati konfiguacioni keyboard fajl (fajl u kome je definisano koja akcija će se na koju kombinaciju tastera izvršiti, npr. Alt+N će pokrenuti novu konekciju). Aklo izostavite eksenziju, biće prikazani samo fajlovi sa .CNF ekstenzijom

[Control] menu

Reset terminal - Resetuje terminal

Are you there – Šalje AYT (Are You There) telnet signal. Obično host šalje nazad neku poruku da bi odgovorio. Ova komanda je dostupna samo kod telnet konekcija.

Send break – Šalje break signal hostu

Reset port – Resetuje serijski port. Ova opcija je dostupna samo kod serijske veze.

Open TEK – Otvara TEK prozor (na ovaj način terminal emulator omogućava više konekcija).

Close TEK – Zatvara TEK prozor.

Macro – Otvara prozor za izbor fajla u komse su definisani makroi.

[Window] menu

Možete aktivirati jedan od postojećih (već otvorenih) prozora terminal emulatora birajući ga na listi u ovom meniju. U meniju najviše može biti izlistanu 9 postojećih prozora. Ako broj otvorenih prozora premašuje 9, i Vi hoćete da aktivirate prozor koji se ne nalazi na listi, pozovite komandu "Window…" sa dna ovog menija, koja otvara ili zatvara određeni prozor.

[Help] menu

Index – Otvara prozor za pomoć koji je indeksiran po temama

Using help – Objašnjenje kako kosristiti Windows Help fajlove

About Tera Term... – Informacije o Tera Term terminal emulatoru

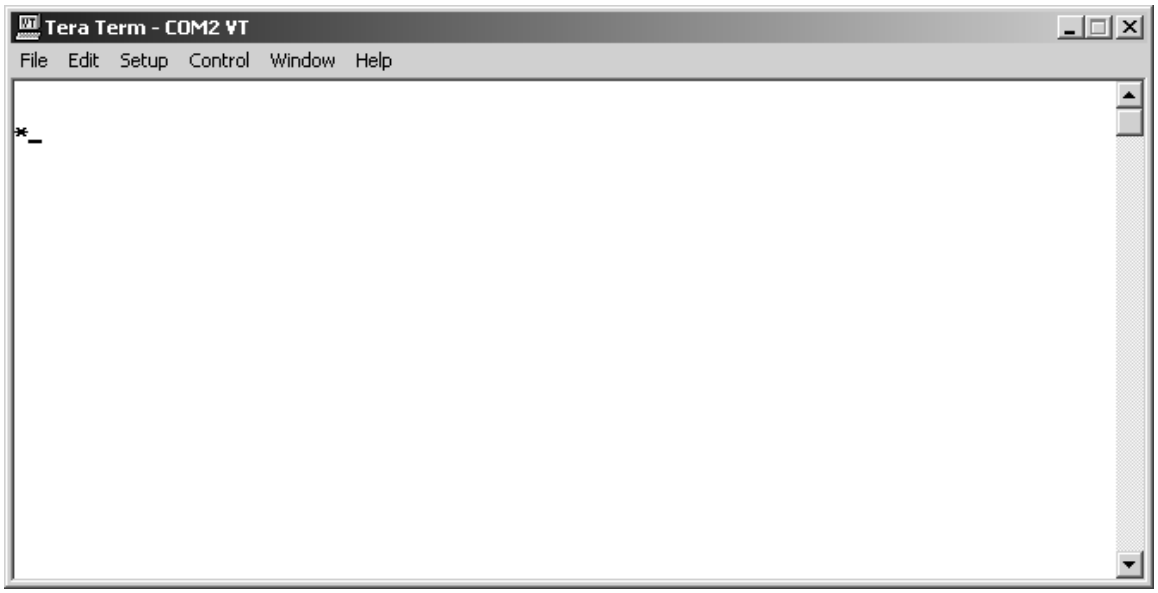
Procedura za korišćenje terminal emulatora na vežbama

Da bi napisan i preveden program preneli na razvojni sistem na kome će program biti testiran, potrebno je naravno, prvo startovati terminal emulator. Ukoliko upravo prvi put startujete terminal emulator, potrebno je konfigurisati ga prema svojim potrebama, a u skladu sa ranije navedenim mogućnostima. Postoje neka podešavanja koja svakako treba uraditi, koja ne zavise toliko od individualnih potreba korisnika, i ta podešavanja su sledeća:

- Po startovanju terminal emulatora izabrati serijsku vezu, umesto podrazumevane TCP/IP veze, i izabrati COM port koji se koristi
- Podesiti Brzinu protoka (Baud Rate) serijskog porta na 19200kb/s, Podatak (data) na 8 bita, Parnost (parity) na NONE, Stop bit na 1, Kontrolu toka (Flow cnotrol) na NONE.
- Povećati Font radi bolje vidljivosti
- Sačuvati podešavanja

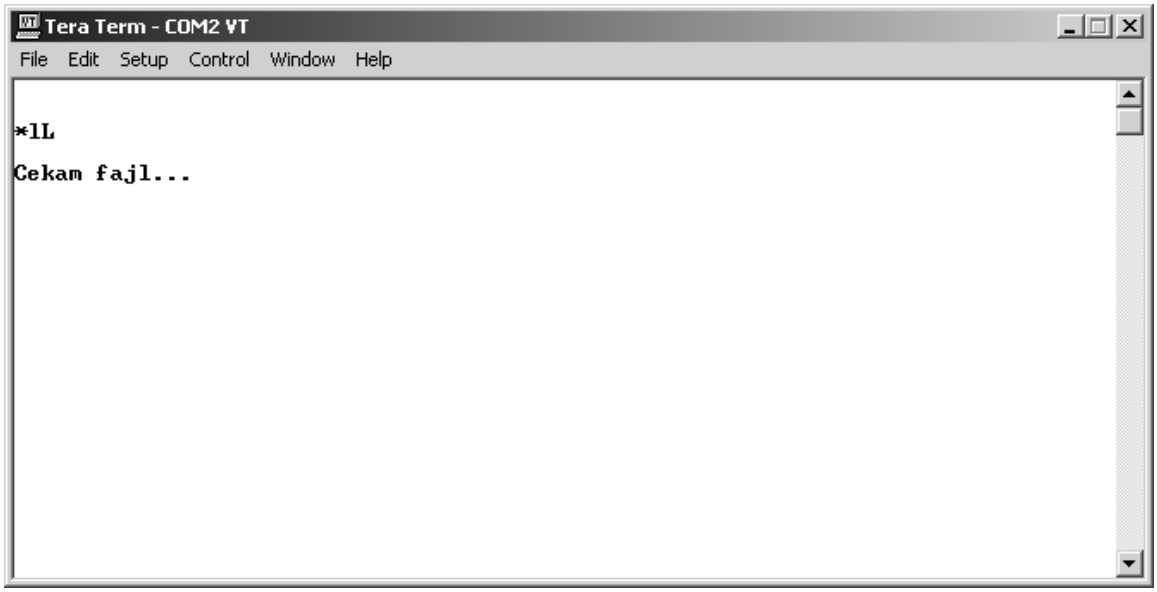
Sva ostala podešavanja, kao na primer Local echo, su po želji korisnika i nisu neophodna za ispravan rad razvojnog okruženja.

Sledeći korak je prenos podataka na razvojni sistem. Naravno, razvojni sistem je prvo potrebno uključiti. Ako je terminal emulator bio startovan i pravilno konfigurisan pre uključenja razvojnog sistema, u prozoru terminal emulatora bi trebalo da se pojavi "*" (slika 2.1), što je prompt signal razvojnog sistema. Ako terminal emulator nije bio startovan u vreme uključenja razvojnog sistema, potrebno je resetovati razvojni sistem (reset taster je četvrtasti taster crvene boje na poleđini kućišta razvojnog sistema), kako bi utvrdili da li je veza ispravna. Ako se na pojavi prompt signal ni posle reseta, a parametri serijske komunikacije su podešeni na ranije naveden način, treba proveriti fizičku vezu između razvojnog sistema i računara, što ne bi trebalo da se radi bez prisustva laboranta. Kada se prompt signal pojavi, može se preći na sledeći korak.



Slika 5.1 Prompt odziv razvojnog sistema

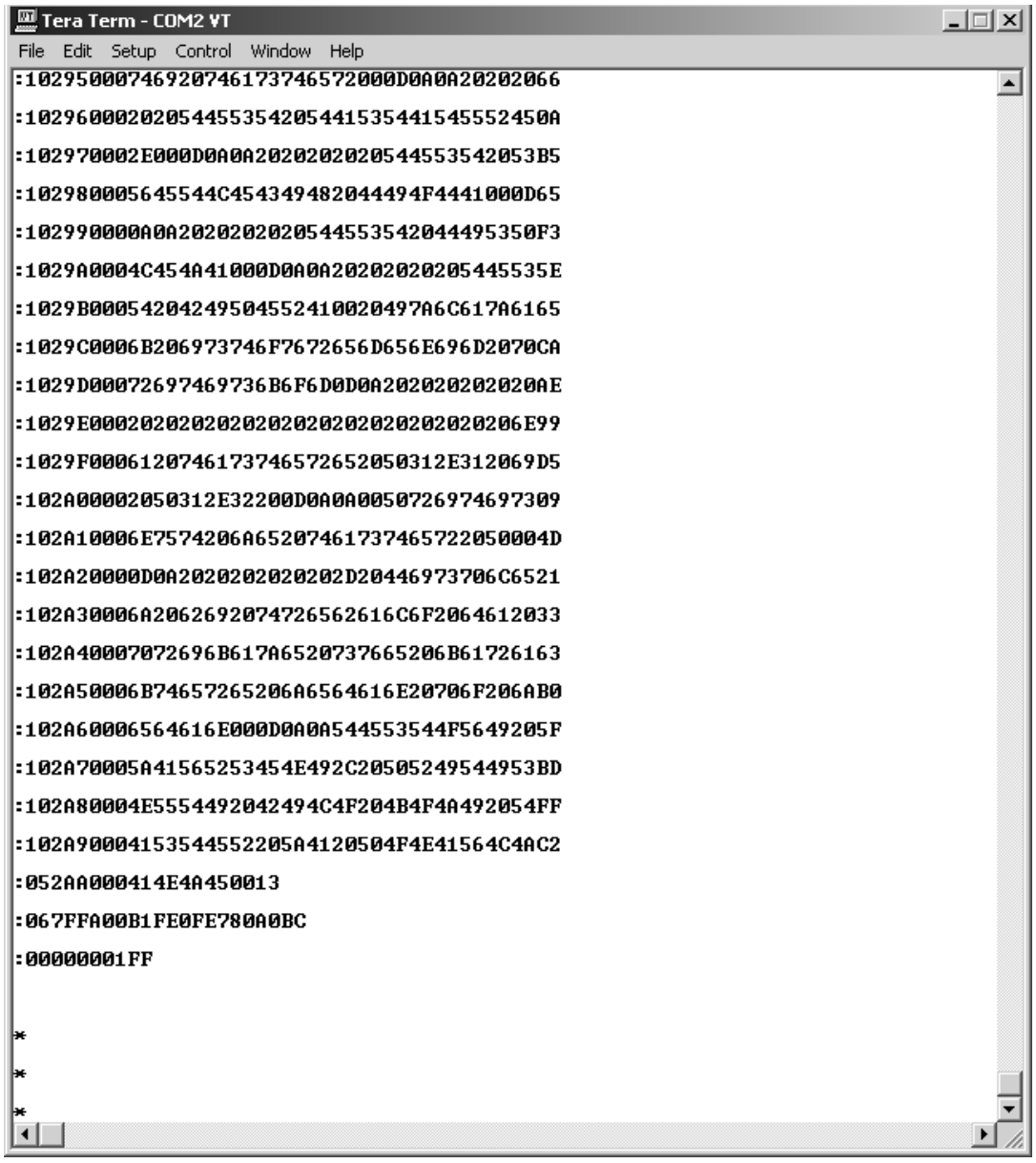
Sledeći korak je izdavanje komande razvojnom sistemu da se spremi za prijem novog programa. Ova komanda se naravno izdaje preko serijskog porta, što znači kroz terminal emulator. Komanda razvojnom sistemu za pokretanje prijema novog programa je “L” ili “l”, tj. malo ili veliko l, od engleske reči “Load”, što znači napuni. Po prijemu ove komande, razvojni sistem daje odgovor ,tj. potvrđuje prijem komande, i prjavljuje da je spreman za prijem (slika 2.2).



Slika 5.2 Odziv na “Load” komandu,tj. na komandu za prijem novog programa

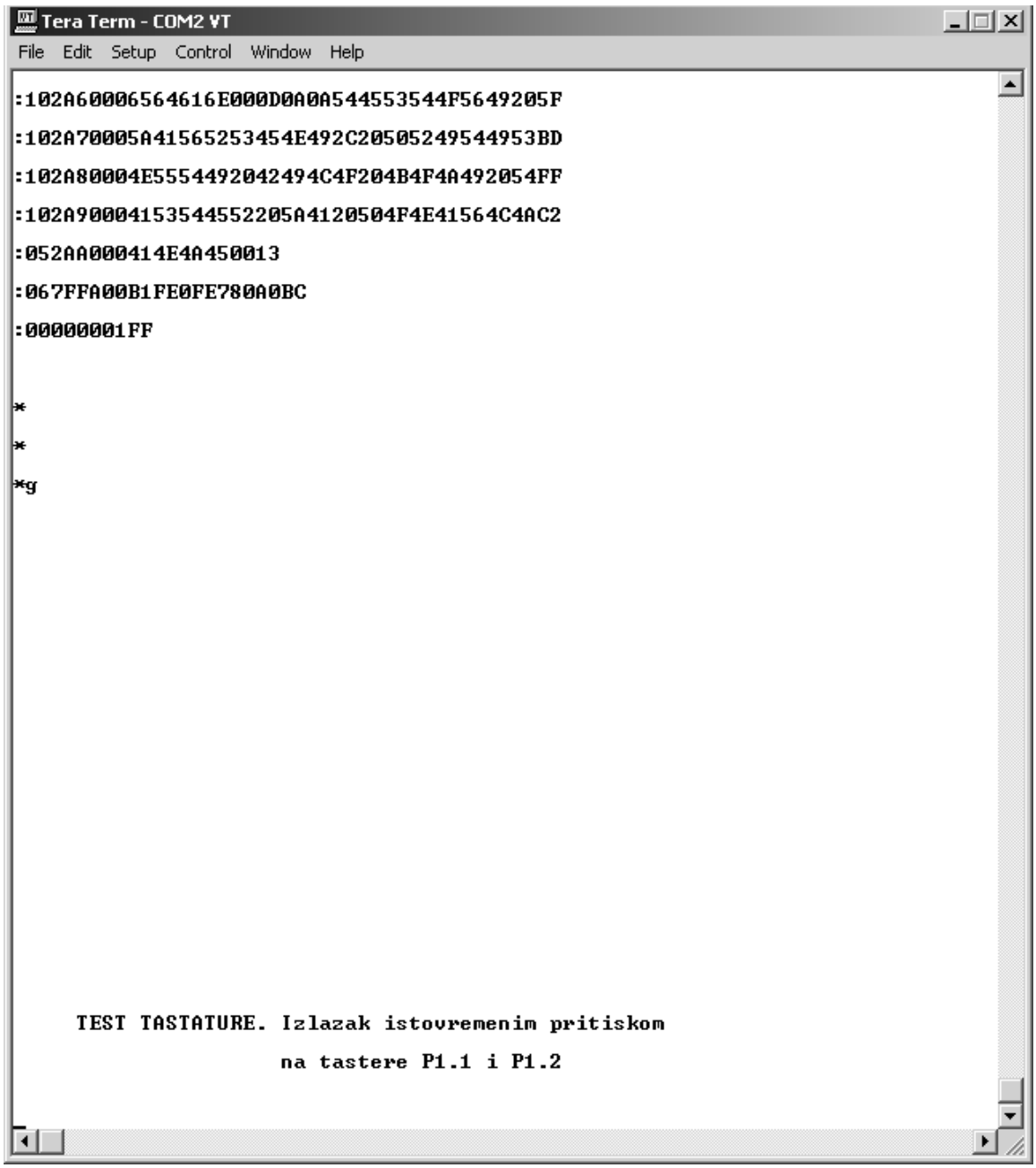
Sledeće što je potrebno uraditi je i stvarno poslati fajl u kome se nalazi preveden i linkovan program. Komanda terminal emulatoru da počne sa slanjem fajla (Send file) se nalazi u “File”-meniju. Takođe, definisanjem prečica na tastaturi, moguće joj je pristupiti i istovremenim pritiskom na tastere Ctrl+F5. Sada je potrebno izabrati fajl, a ako je za default direktorijum postavljen direktorijum u kome se nalazi hex fajl koji je rezultat prevođenja, linkovanja, i hex-translacije, tada je samo potrebno izabrati taj fajl u listi fajlova u direktorijumu. Ako ovo nije slučaj, potrebno je prvo naći odgovarajući direktorijum, a onda naći ovaj fajl, i na kraju, poslati ga biranjem komande “Open”.

Po startovanju prenosa, terminal emulator će primati od razvojnog sistema kao povratnu informaciju svaki karakter koji je poslat. po završenom prenosu, na displeju razvojnog sistema će se pojaviti natpis “prenos uspešan”, a u prozoru terminal emulatora pojaviće se tri prompt znaka (slika 2.3).



Slika 5.3 Izgled prozora terminal emulatora po uspešnom prenosu

Po uspešnom prijemu novog programa, razvojni sistem je spreman da pokrene taj isti program. Ono što je potrebno učiniti je izdaati mu komandu za pokretanje programa (koji se sada nalazi u RAM memoriji), slanjem karaktera “G” ili “g” na serijski port (tj. pritiskom na taster “G” ili “g” dok je prozor terminal emulatora aktivan prozor). Sada je program pokrenut i izvršava se (slika 2.4).



Slika 5.4 Pokretanje i izvršavanje programa

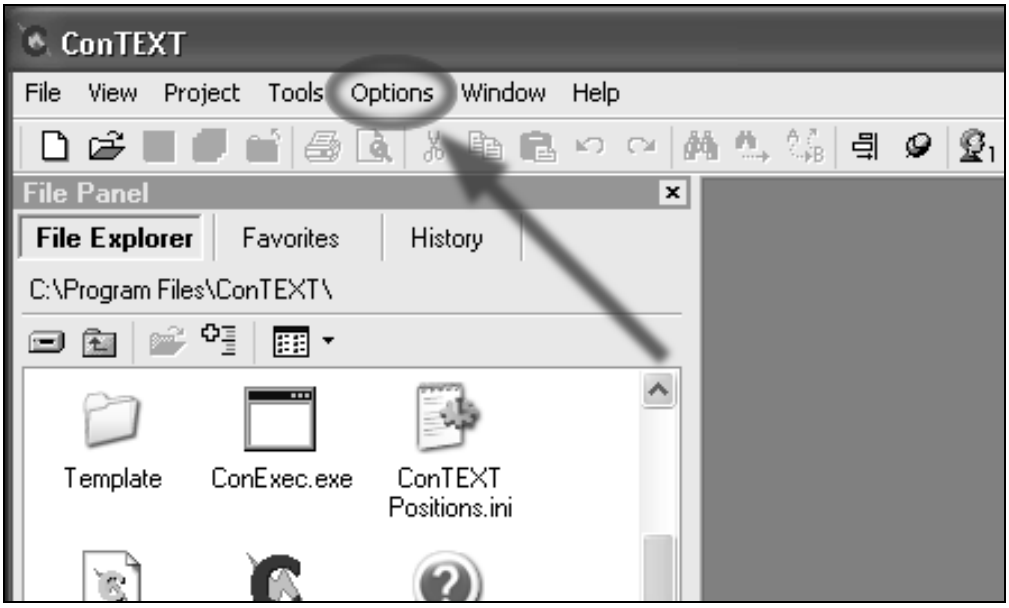
Po uspešnom (ili neuspešnom) testiranju programa, potrebno je resetovati razvojni sistem.

Dodatak 6. Uputstvo za podešavanje ConTEXT editora

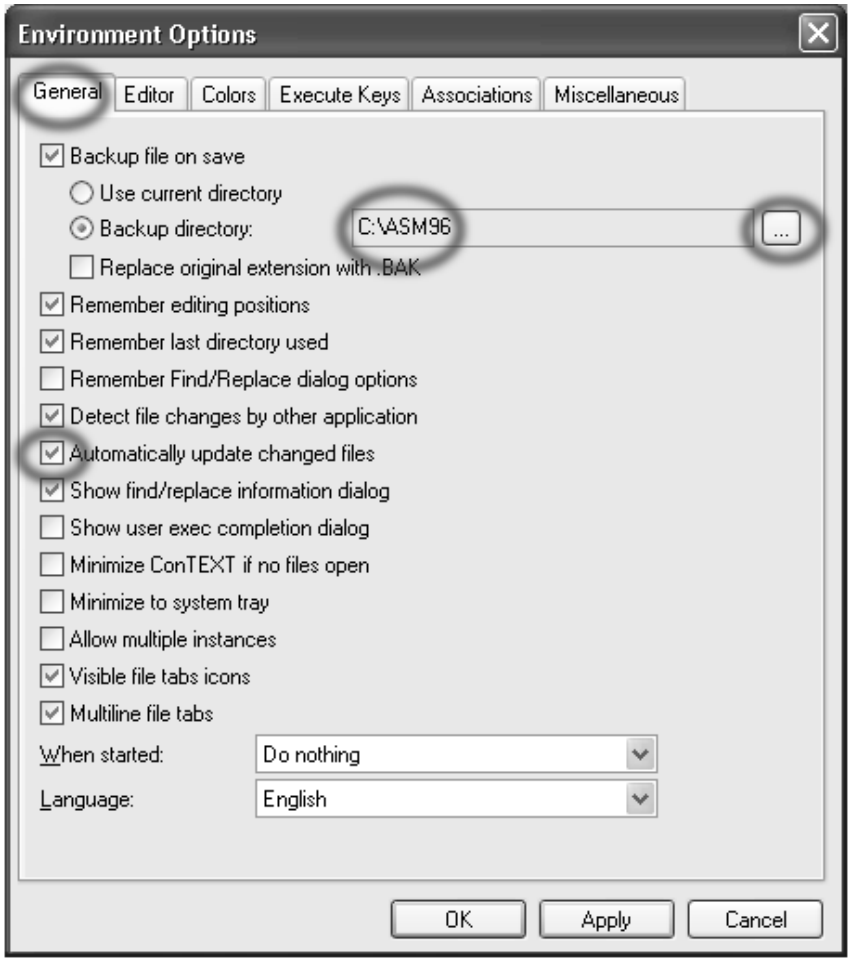
Okruženje koje je očekivano na disku računara je sledeće:

Postojanje direktorijuma **ASM96** gde se nalazi sam asembler, i u kome postoje poddirektorijumi **Uputstva** (u kome se nalazi fajl **Asmhyper.hlp**), i poddirektorijum **student** u kome će se smeštati fajlovi vezani za pisanje kôda, kao što je sam izvorni kôd, izveštaji asemblera i linkera, itd. Takođe, pretpostavlja se da se **highlighter 80c196** nalazi u poddirektorijumu **Highlighters** direktorijuma **C:\Program Files\ConTEXT** ConTEXT editora.
Dalje, potrebno je da postoji fajl **prevedi.bat** (koji može da se zove i drugačije, ali je bitan njegov sadržaj) a čiji sadržaj je naveden i delimično objašnjen u dodatku.
Po startovanju fajla za pokretanje vežbi, ili po startovanju ConTEXT editora, potrebno je izvršiti sledeće izmene u odnosu na one opcije koje su po pravilu uključene u ovom editoru:

1. Otvoriti **Options** menu i u njemu izabrati **"Environment Options…"**



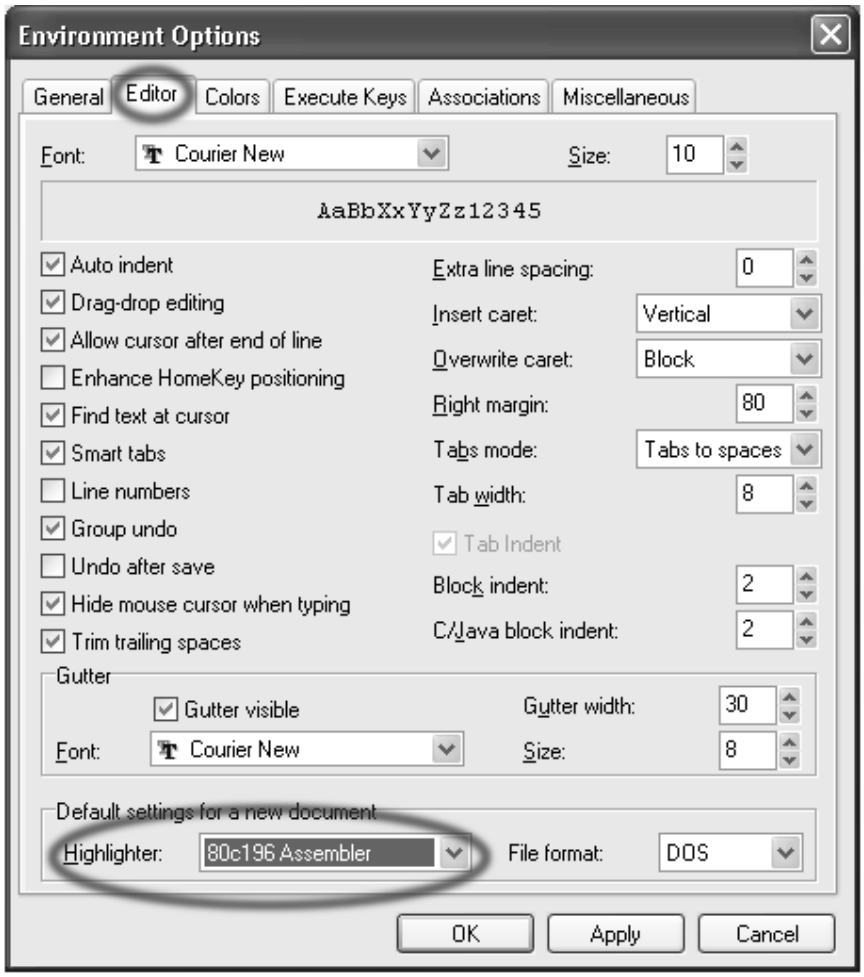
2. U prozoru koji se otvori, treba izvršiti sledeća podešavanja za opšte opcije, kojima se može pristupiti u **General** odeljku:



Kao **Backup directory** pronaći putanju do direktorijuma **C:\ASM96** (ili do nekog drugog direktorijuma u kome je dozvoljeno pisanje, a koji ne može biti obrisan kao posledica, recimo, pozivanja batch fajla koji služi za pripremu direktorijuma **student** na početku vežbi).

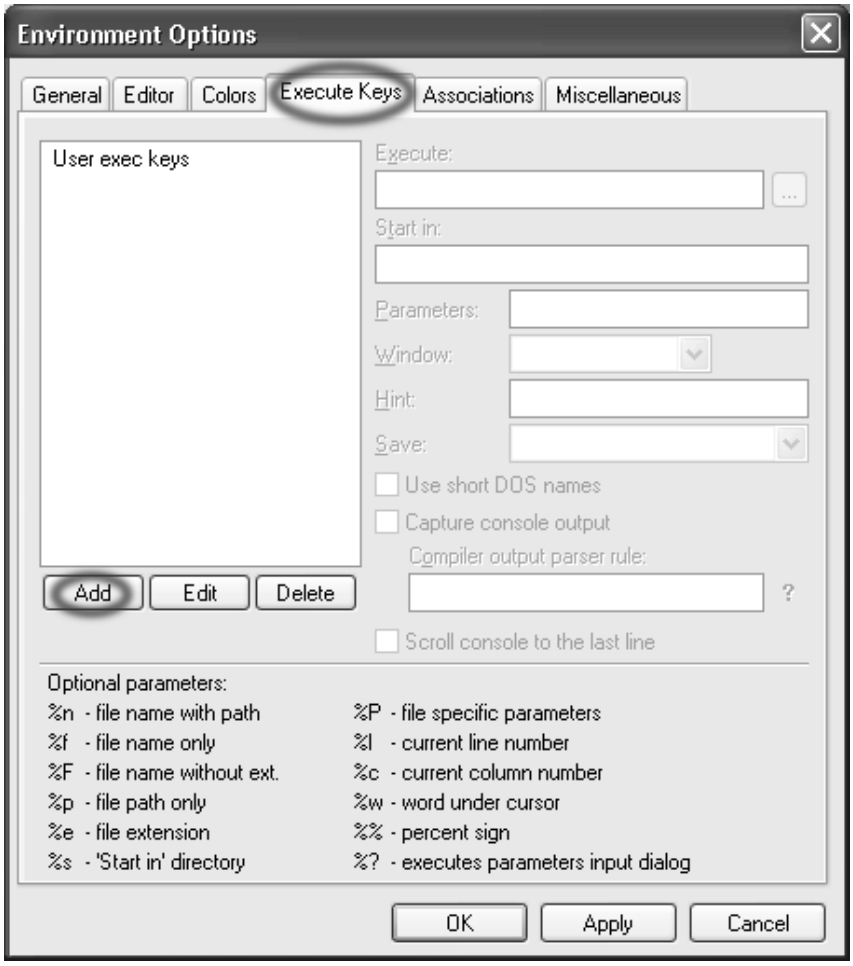
Treba uključiti opciju **"Automatically update changed files"**

3. Preći na podešavanja vezana za sam editor, izborom "zalistka" **Editor** koji se nalazi pored "zalistka" **General**, i tu podesiti , u delu "**Default settings for a new document" Highlighter** na **80c196 Assembler.**

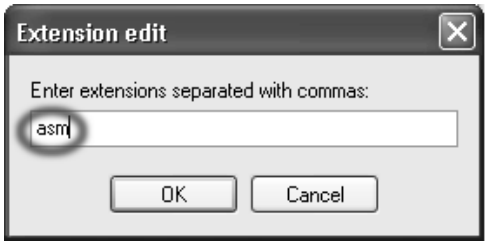


4. Dalje, što se tiče podešavanja koja su vezana za boje, tj. **"Colors"** potrebno je samo odabrati **Highlighter** na osnovu koga će editor postavljati boje, a on opet treba da bude, kao i u prethodnom slučaju, **80c196 Assembler.**

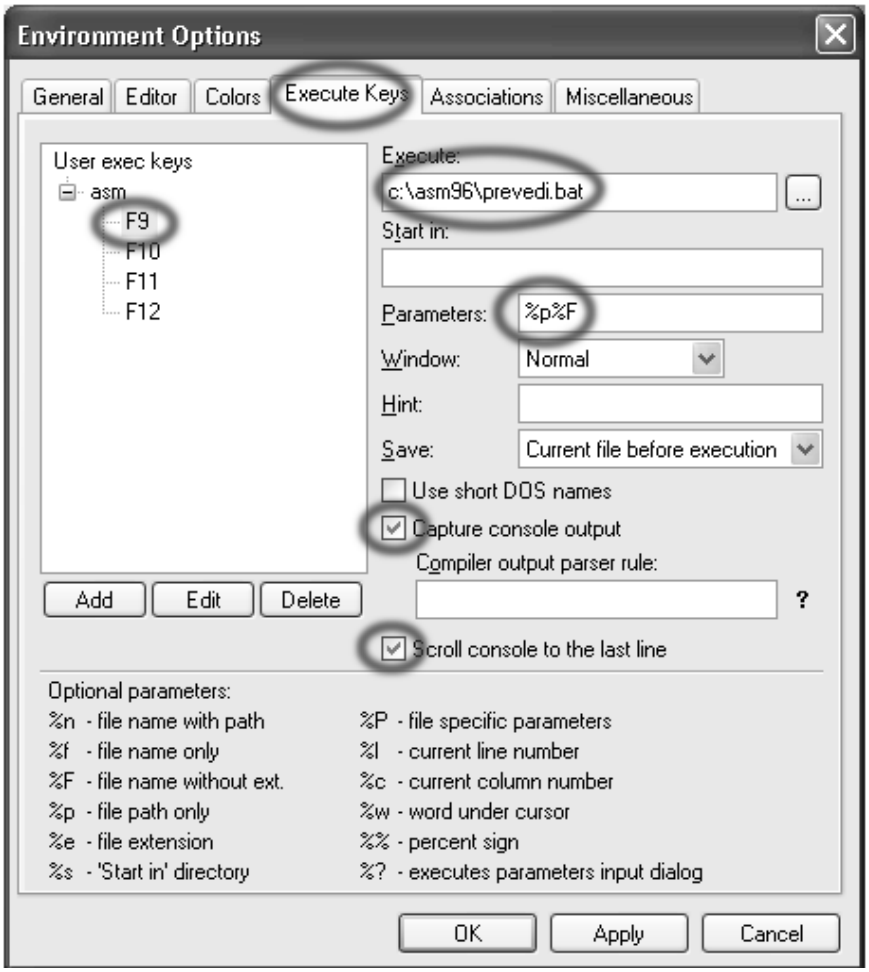
5. Sledeće što je potrebno podesiti je batch fajl koji će se koristiti za prevođenje, kao i opcije vezane za pozivanje "spoljnih" programa od strane editora. Potrebno je preći na **"Execute Keys"** i tamo, odabrati opciju **User exec keys** i pritisnuti taster **Add**.



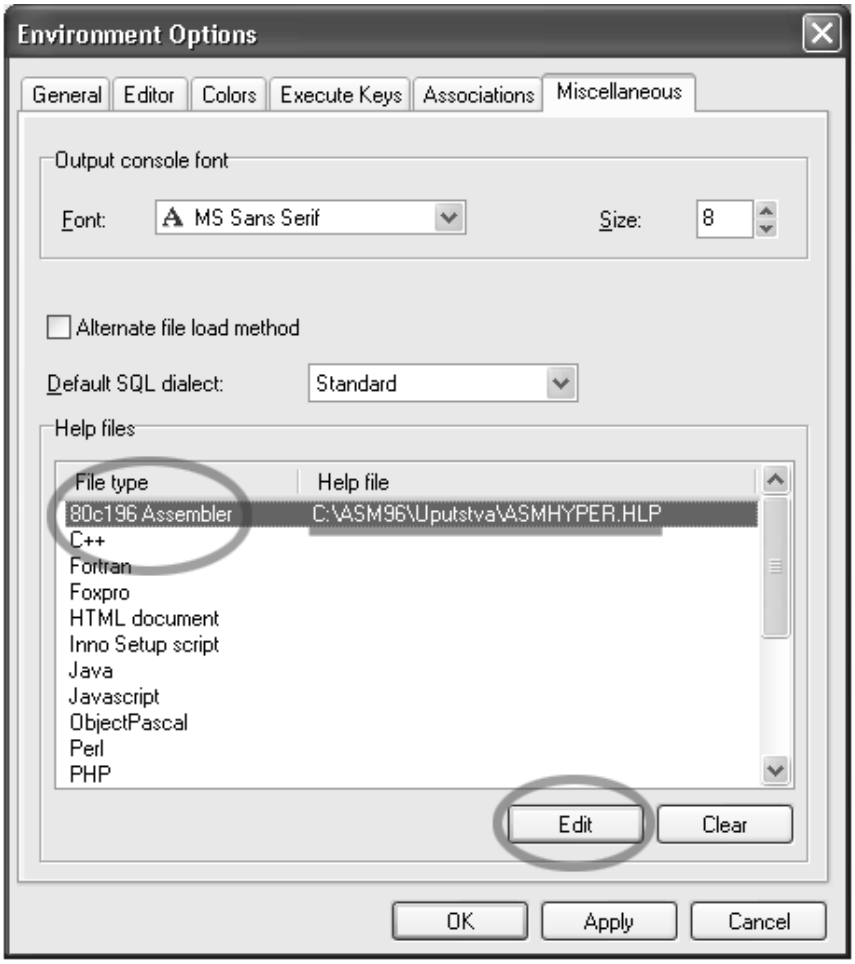
5.1 U prozoru koji se pojavi treba upisati asm, što je ekstenzija fajlova za koje će se pozivati pomenuti batch fajl, i pritisnuti **OK** taster.



5.2 Potom, treba podesiti akciju za svaki taster, a pošto je nama dovoljan samo jedan taster, jer imamo samo jedan spoljni program koji pozivamo, treba odabrati **F9**, u **Execute** polju pronaći (ili upisati njegovu putanju) fajl **prevedi.bat** (koji bi trebao da bude, a ne mora, u direktorijumu **C:\ASM96**). U polju **Parameters** potrebno je upisati **%p%F** (malo **p** veiko **F**), i uključiti opcije **"Capture console output"** i **"Scroll console to the last line"**.



6. Poslednja stavka je **"Miscellaneous"** gde se editor povezuje sa spoljnim **help** fajlom. Potrebno je odabrati **80c196 Assembler**, pritisnuti taster **Edit,** i ukazati editoru na putanju do fajla **ASMHYPER.HLP**, koji se nalazi u direktorijumu **C:ASM96\Uputstva.**



Kada su ova podešavanja završena, može se nastaviti sa radom.

Dodatak 7. Opis batch fajla

```
1:  @echo off
2:  cls
3:  C:\asm96\asm96 %1.asm
4:  IF ERRORLEVEL 1 GOTO GG1:
5:  echo .
6:  echo .
7:  echo ASEMBLIRANJE USPESNO,
8:  C:\asm96\rl96 %1.obj > null
9:  IF ERRORLEVEL 1 GOTO GG2:
10: echo .
11: echo .
12: echo LINKOVANJE USPESNO,
13: C:\asm96\oh %1 >null
14: echo .
15: echo *****  Izlazni HEX fajl je u %1.HEX *****
16: GOTO KRAJ
17: :GG1
18: C:\progra~1\context\context %1.LST
19: goto gg:
20: :GG2
21: C:\progra~1\context\context %1.m96
22: :GG
23: echo ***** GRESKA, Videti %1.lst i %1.m96 *****
24: :KRAJ
```

Stavke koje su bitne za razmatranje u ovom batch fajlu su sledeće:

3: Poziv asemblera koji izvršava ConTEXT editor, gde se kao argument u pozivu asemblera koristi ulazni parametar prosleđen batch fajlu od strane ConTEXT editora

4: U slučaju da je došlo do incidenta pri asembliranju, u izvršavanju batch fajl skače na labelu GG1 (linija 17) i poziva ConTEXT editor sa parametrom koji je jednak ulaznom ali je sa ekstenzijm .lst, čime ustvari biva otvoren (za editovanje) izveštaj asemblera

7: Ako je asembliranje prošlo bez incidenata, batch fajl nastavlja sa izvršavanjem, ispisuje poruku “ASEMBLIRANJE USPESNO”

8: Sada se poziva povezivač (linker) za isti ulazni argument koji je bio prosleđen asembleru, ali sa različitom ekstenzijom, tj. sa ekstenzijom .obj

9: U slučaju incidenta pri linkovanju, izvršavanje batch fajla se nastavlja na od labele GG2 (linija 20) i biva otvoren u ConTEXT editoru (za editovanje) izveštaj linkera (ime fajla je isto, ekstenzija je .m96)

12: U slučaju uspešnog linkovanja ispisuje se poruka “LINKOVANJE USPESNO”

13: Na ovoj liniji, na koju će se doći samo u slučaju da je program uspešno asembliran i linkovan, biva pozvan hex translator sa argumentom koji je isti ulaznom argumentu i za linker i asembler, ali u ovom slučaju bez ekstenzije

15: Ispisuje se poruka gde se nalazi potpuno preveden .hex fajl i skače se na labelu KRAJ, posle koje nema nikakvih naredbi.