

## **Vežba 5.**

### **Programiranje mikrokontrolera u programskom jeziku C. Specifičnosti u odnosu na ANSI C jezik. C51 prevodilac.**

#### **Uvod**

Programiranje mikrokontrolera u assembleru predstavlja optimalan način programiranja sa stanovišta iskorišćenja ograničenih ugrađenih resursa, kao na primer, programske memorije i dr. S druge strane za složenije projekte program napisan u assembleru može biti komplikovan za čitanje i razumevanje. Zato se danas mikrokontroleri programiraju u višim programskim jezicima, a najpoznatiji i najčešće korišćeni je programski jezik C. Ovaj jezik ne samo da uprošćava projektovanje i programiranje, nego i čini osnovu većine viših programskih jezika, pa je njegovo znanje potreba svakog inženjera.

Ova vežba podrazumeva da student poseduje osnovna znanja o programskom jeziku C i o njegovim osnovnim strukturama podataka. Takođe, pretpostavlja se znanje tzv. ANSI C-a.

Programski jezik C je veoma pogodan za programiranje hardvera jer je u odnosu na ostale više programske jezike hijerarhijski bliži hardveru tj. mogu se pronaći jednostavne sitaksne jednakosti u odnosu na mašinski jezik. Efikasnost programiranja je povećana i zbog upotrebe elemenata strikturalnog programiranja i širokog skupa operatora.

U odnosu na ANSI (*American National Standards Institute*) C programski jezik, C za mikrokontrolere je prilagođena varijanta koja koristi specijalizovan kompajler (prevodilac). Primer specijalizovanog kompajlera je C51 kompajler koji se koristi za prevođenje programa kod mikrokontrolera baziranih na 8051 mikroprocesorima. C jezik sam po sebi nije u mogućnosti da obavlja operacije (kao što su ulaz i izlaz) koje bi normalno zahtevale intervenciju operativnog sistema. Zato su ove mogućnosti obezbeđene kao deo standardne biblioteke. Zbog toga što su ove funkcije odvojene od samog jezika, C je naročito pogodan za generisanje programa lako prenosivih između različitih platformi. Ukratko, interakcija programa sa okolinom u programskom jeziku C je sadržana u njegovoj standardnoj biblioteci, a C51 kompajler je specijalizovan za ugrađene (embedded) mikroprocesore. Ukratko ćemo objasniti proširenja C-a za mikrokontrolere u odnosu na standardan ANSI C jezik.

#### **Proširenja standardnog C jezika za 8051 mikrokontrolere**

U cilju što lakšeg rada sa 8051 mikroprocesorima, C51 kompajler koristi brojne nove ključne reči koje ne postoje u standardnom C jeziku (sl. 1). Ukratko ćemo objasniti značenje najviše korišćenih ključnih reči.

Arhitektura 8051 podržava nekoliko fizički odvojenih memorijskih prostora i delova za smestanje programa. Svaki memorijski prostor nudi određene prednosti i mane. Postoje memorijski prostori koji omogućavaju:

- Čitanje ali ne i upis
- I upis i čitanje
- I upis i čitanje ali brže u odnosu na ostale memorije

<u><a href="#">_at</a></u>	<u><a href="#">far</a></u>	<u><a href="#">sbit</a></u>
<u><a href="#">alien</a></u>	<u><a href="#">idata</a></u>	<u><a href="#">sfr</a></u>
<u><a href="#">bdata</a></u>	<u><a href="#">interrupt</a></u>	<u><a href="#">sfr16</a></u>
<u><a href="#">bit</a></u>	<u><a href="#">large</a></u>	<u><a href="#">small</a></u>
<u><a href="#">code</a></u>	<u><a href="#">pdata</a></u>	<u><a href="#">_task</a></u>
<u><a href="#">compact</a></u>	<u><a href="#">_priority</a></u>	<u><a href="#">using</a></u>
<u><a href="#">data</a></u>	<u><a href="#">reentrant</a></u>	<u><a href="#">xdata</a></u>

*Slika 1. Ključne reči dodate ANSI C-u za lakši opis 8051 arhitekture*

Programska (code) memorija ima samo mogućnost čitanja i može biti po realizaciji unutrašnja, spoljašnja i kombinacija ove dve memorije. Programskoj memoriji se može pristupiti pomoću memorijskog tipa **code** definisanog u okviru C51 standardne biblioteke.

Unutrašnja memorija za podatke se nalazi na samom čipu i u nju se može i upisivati i čitati. Postoje razne varijante čipova u zavisnosti od količine unutrašnje memorije za podatke, ali u našem slučaju radićemo sa varijantama koje sadrže 256k unutrašnje memorije za podatke. Memoriji za podatke se brzo pristupa jer se koristi 8-bitna adresa. Za pristup internoj memoriji se koriste tri različita memorijska tipa:

- **data** - označava da se pristupa internoj memoriji podataka uz direktno adresiranje, što omogućava brz pristup (128B).
- **idata** - označava da se pristupa celoj memoriji za podatke (256B) uz indirektno adresiranje.
- **bdata** – označava da se pristupa lokacijama ukupne veličine od 16 bajtova koje se mogu adresirati po bitovima.

Za razliku od unutrašnje, spoljašnja memorija za podatke je sporija, jer se pristup vrši preko pokazivača na podatke. Maksimalno je podržano 64k spoljašnje memorije za podatke. Vrlo je važno da se shvati da se ovaj adresni prostor ne mora koristiti samo za pristupanje memoriji. Naš hardverski dizajn može mapirati određene periferne sklopove unutar memorijskog prostora. C51 kompajler nudi dva memorijska tipa za pristup spoljašnjoj memoriji za podatke:

- **xdata** - da se može pristupiti bilo kojoj lokaciji unutar memorijskog prostora (64k) i koristi se kod *large* memorijskog modela (objašnjeno kasnije)
- **pdata** - označava da se pristupa samo jednoj stranici veličine 256 bajta od spoljašnje memorije za podatke. Ovaj tip se koristi kod *compact* memorijskog modela koji će takođe biti objašnjen kasnije.

Udaljena memorija (far memory) označava prošireni adresni prostor kod većine 8051 varijanti. C51 kompajler koristi generički trobajtni pokazivač za pristup ovom proširenom adresnom prostoru. Dva C51 memorijska tipa, **far** i **const far**, služe za pristup promenljivama u proširenom RAM prostoru i konstantama u proširenom ROM prostoru.

Memorijski model određuje tip memorije u upotrebi za argumente funkcija, automatske promenljive i deklaracije koje ne koriste eksplicitno navedeni tip memorije. C51 koristi tri memorijska modela.

- **Small Model** - Kod ovog modela, sve promenljive, podrazumevano, se nalaze u unutrašnjoj memoriji za podatke 8051 sistema tj. koristi se **data** memorijski tip. Kod ovog modela, promenljivama se pristupa na efikasan način. Međutim, svi objekti koji nisu eksplicitno postavljeni u nekom drugom memorijskom prostoru, moraju da se uklupe u okviru unutrašnjeg RAM-a koji je sam po sebi vrlo mali.
- **Compact Model** - Kod ovog modela, podrazumevano, sve promenljive se nalaze u jednoj stranici spoljašnje memorije za podatke tj. koristi se memorijski tip **pdata**. Ovaj model može obezbediti maksimalno 256 bajtova promenljivih, jer se koristi indirektno adresiranje kroz registre R0 i R1.
- **Large Model** - Kod ovog modela, sve promenljive se nalaze u spoljašnjoj memoriji za podatke (do 64k prostora) tj. koristi se **xdata** memorijski tip. Pristup memoriji kod ovog modela ja u odnosu na ostale modele najsporiji i neefikasan, naročito kod promenljivih koje sadrže nekoliko bajtova.

Evo nekoliko primera za korišćenje navedenih memorijskih tipova.

```
unsigned char bdata bdata_var;
unsigned char code code_constant;
unsigned char data fast_variable;
unsigned char far far_variable;
unsigned char idata variable;
unsigned char pdata variable;
unsigned char xdata variable;
```

Familija 8051 mikrokontrolera sadrži poseban memorijski prostor za pristup specijalnim funkcijskim registrima (Special Function Registers - SFRs). Ovi registri se koriste u programu za kontrolu tajmera, brojača, serijskog ulaza i izlaza, ulazno-izlaznih portova i periferija. Nalaze se na adresama od 0x80 do 0xFF (128 bajta) i može im se pristupati bitski, bajtovski, ili na nivou reči. Svi nazivi ovih registara su predefinisani u okviru C51 kompajlera. C51 kompajler obezbeđuje pristup specijalnim registrima preko tipova podataka:

- **sfr** - deklarise se u istom stilu kao i druge C promenljive. Portovi P0, P1, P2 i P3 koriste ovaj tip. Ovaj tip se ne može definisati u okviru tela funkcije.

```
sfr P0 = 0x80;    /* Port-0, address 80h */
sfr P1 = 0x90;    /* Port-1, address 90h */
sfr P2 = 0xA0;    /* Port-2, address 0A0h */
sfr P3 = 0xB0;    /* Port-3, address 0B0h */
```

- **sfr16** - neki od specijalnih registara su 16-bitni. Na primer, 8052 koristi adrese 0xCC i 0xCD za donji i gornji bajt tajmera/brojača 2.

```
sfr16 T2 = 0xCC;    /* Timer 2: T2L 0CCh, T2H 0CDh */
sfr16 RCAP2 = 0xCA; /* RCAP2L 0CAh, RCAP2H 0CBh */
```

- **sbit** - zadaci koji se rešavaju 8051 mikrokontrolerima često zahtevaju jednobitne promenljive. Zato C51 podržava jednobitni tip podataka koji omogućava pristup specijalnim jednobitnim registrima.

```
sbit EA = IE^7;    //označava dozvolu prekida
```

```

sfr PSW = 0xD8; //specijalni registri
sfr IE = 0xA8; //sa mogućnošći bitskog adresiranja u tri varijante

sbit OV = PSW^2; //varijanta 1 - sbit name = sfr-name ^ bit-position
sbit CY = PSW^7;
sbit EA = IE^7;

sbit OV = 0xD0^2; //varijanta 2 - sbit name = sbit-address ^ bit-position
sbit CY = 0xD0^7;
sbit EA = 0xA8^7;

sbit OV = 0xD2; //varijanta 3 - sbit name = sbit-address
sbit CY = 0xD7;
sbit EA = 0xAF;

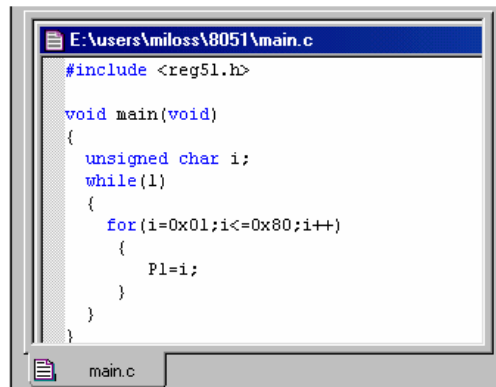
```

Pored ovih tipova specifičnih za 8051 mikrokontrolere, ostali tipovi u standardnom ANSI C-u su takođe podržani od strane C51 kompajlera. Koji su to tipovi i koliko zauzimaju memorije, može se videti u sledećoj tabeli.

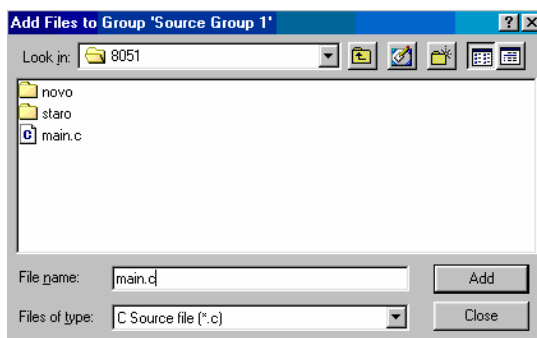
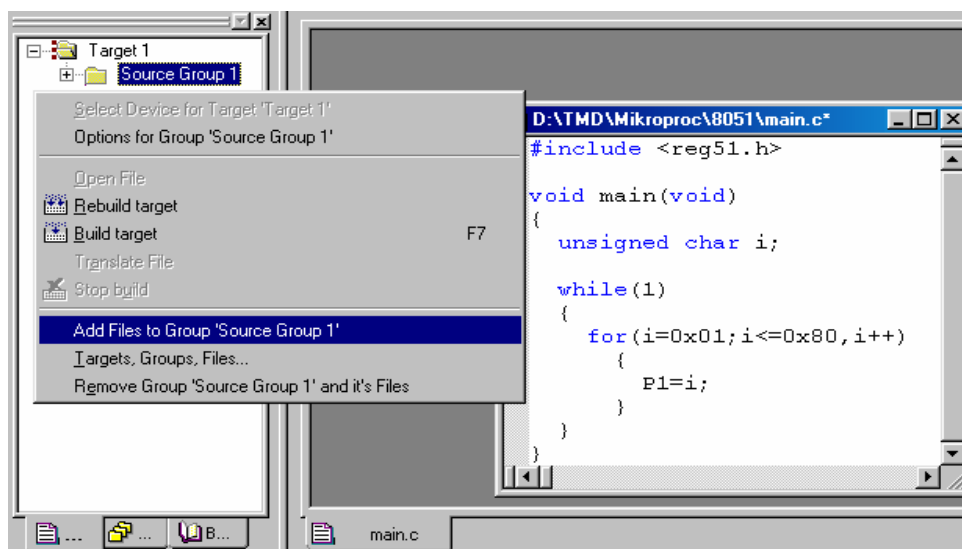
Tip podataka	Bitovi	Bajtovi	Opseg vrednosti
<a href="#">bit</a>	1		0 to 1
signed char	8	1	-128 to +127
unsigned char	8	1	0 to 255
enum	8 / 16	1 or 2	-128 to +127 or -32768 to +32767
signed short	16	2	-32768 to +32767
unsigned short	16	2	0 to 65535
signed int	16	2	-32768 to +32767
unsigned int	16	2	0 to 65535
signed long	32	4	-2147483648 to +2147483647
unsigned long	32	4	0 to 4294967295
float	32	4	±1.175494E-38 to ±3.402823E+38
<a href="#">sbit</a>	1		0 or 1
<a href="#">sfr</a>	8	1	0 to 255
<a href="#">sfr16</a>	16	2	0 to 65535

## Rad sa Keil korisničkim interfejsom

Rad sa Keil okruženjem je veoma sličan kao u prethodni vežbama. Sada ćemo objasniti razliku u radu sa programima napisanim u assembleru i u programskom jeziku C. Pokretanje programa i stvaranje novog projekta je nepromenjeno i ostaje isto kao i kod rada sa assemblerom opisanog u prethodnim vežbama. Razdvajanje dolazi kod kreiranja izvornog (source) koda. Kreiranje novog izvornog fajla se vrši izborom opcije **File-New**, pri čemu se otvara prazan editorski prozor u koji se unosi izvorni kod u C-u. Zatim se izabere opcija **File-Save As** i zada se ime fajla sa ekstenzijom **.c**, npr. *main.c* kao na slici (sl. 2). Sada treba taj fajl dodati u projekat. To se može uraditi tako što se prvo otvori **Project Window (View - Project Window)**, ako već nije otvoren. Zatim se klikne desnim tasterom miša na **Source Group1**, a potom izabere **Add Files to Group 'Source Group1'** (sl. 3).



Slika 2. Editor u okviru Keil okruženja za pisanje teksta programa



Slika 3. Postupak dodavanja izvornog koda u tekući projekat

Prvi red programa (sadrži direktivu **#include** za čitanje (ubacivanje) izvornog fajla sa deklaracijama za odgovarajući mikrokontroler. U toku kompajliranja, fajl naveden u direktivi **#include** se ubacuje u osnovni fajl (u ovom slučaju u main.c). Pored **#include** značajna direktiva je i **#define** kojom se definiše makro ili konstanta. Na primer:

```

#include<math.h>      //uključenje matematičkih funkcija
#include<reg51.h>     //uključenje adresa portova, registara i specijalnih
bita u memoriji

```

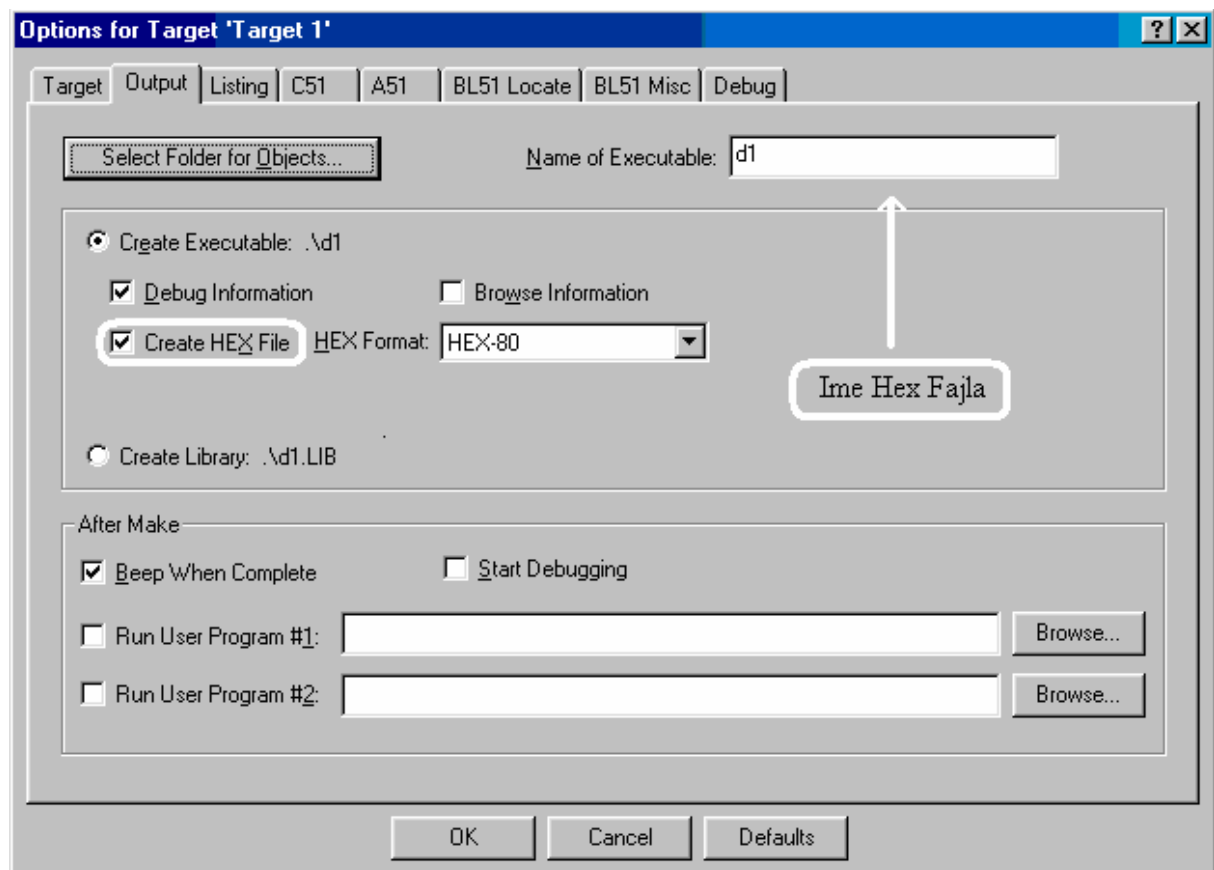
```
#define PI 3.14159 /* definisanje konstante PI, sada se u programu
svuda piše PI a kompajler zna da je to broj 3.14159 */
#define Timer0H 0xE4 //Konstanta u kodu Timer0H se zameni sa vrednošću
=0xE4
#define Timer0L 0xF0
```

Treba zapaziti da je u programu *main.c* promenljiva **i** definisana kao **unsigned char**, što znači da je osmobitna. Kako port P1 ima osam nožica, vrednost koja mu se pridružuje mora biti osmobitna tj. u intervalu od 0 do 255. Pojedinin nožicama porta P1 pristupa se pisanjem  $P1^x$  gde x označava broj nožice (npr.  $P1^2=0$ ), gde je x u opsegu od 0 do 7.

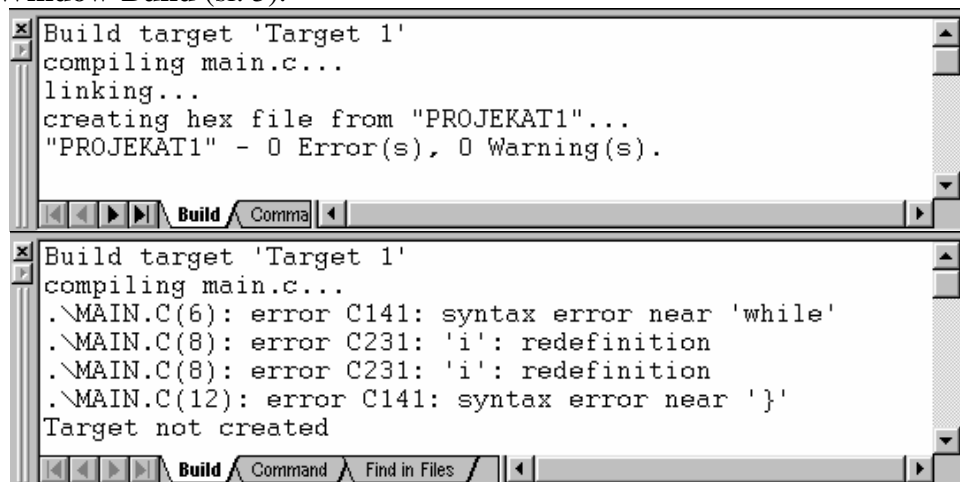
Ovo je bio jednostavan primer programa napisanog u C programu. Možemo primetiti da se telo programa izvršava u okviru beskonačne petlje, što znači da će se program na mikrokontroleru u normalnim okolnostima izvršavati do trenutka kada se prekine napajanje. Ovakav način programiranja nije obavezan, ali je poželjan u smislu da na pravi način opisuje osnovnu upotrebu mikrokontrolera u kontrolnim i nadzornim sistemima u realnom vremenu.

## Testiranje projekta i simulacija

Pošto je projekat napravljen, može se pristupiti testiranju njegove ispravnosti i kreiranju odgovarajućeg HEX fajla, pomoću koga se vrši programiranje mikrokontrolera preko programatora. Prvo se izabere opcija **Project-Options for target 'Target1'-Output** gde se uključi opcija **Create HEX File** (sl. 4):



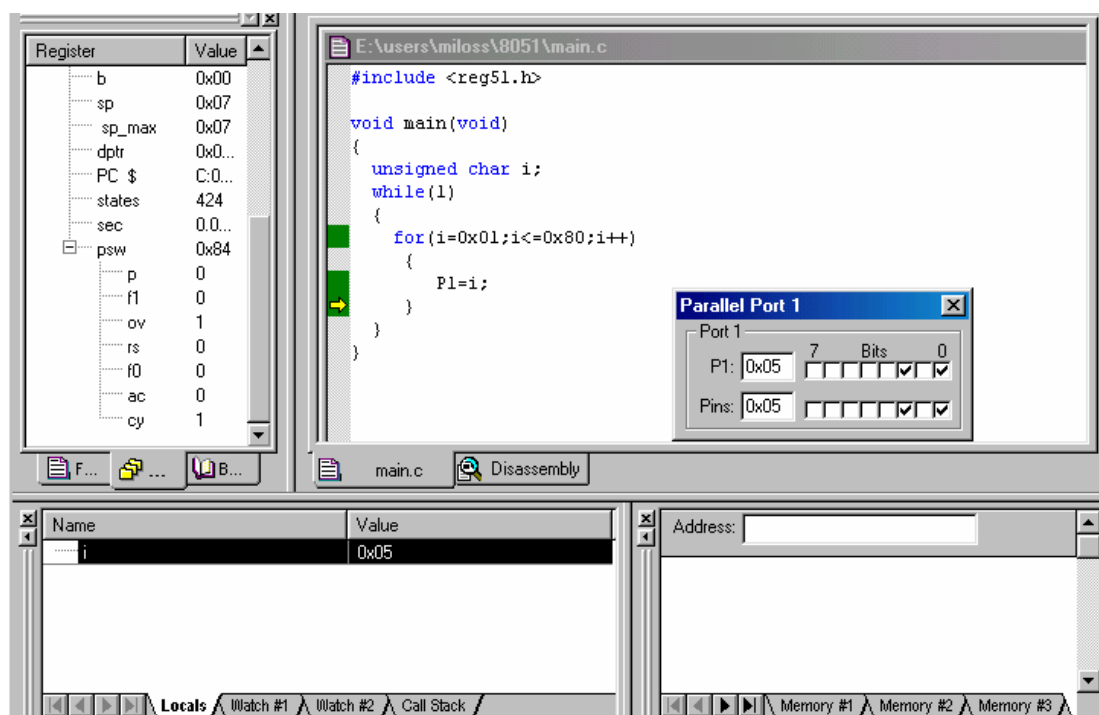
Slika 4. Podešavanja za tekući projekat



*Slika 5. Izveštaj o uspešnom (gore) i neuspešnom (dole) kompajliranju*

Na gornjoj slici je prikazan program bez grešaka, dok na donjoj postoje 4 greške. Kada se dva puta klikne na poruku o datoj greški, strelica označi u editoru red u kome se ta greška nalazi.

Tek kada su ispravljene sve greške u programu može se pristupiti simulaciji rada mikrokontrolera. Postoje mnogi programi za simulaciju, a ovde će u kratkim crtama biti opisane mogućnosti **µVision2 debugger-a**. On simulira memorijsku mapu promenljivih, lokalne promenljive i periferije (serijski port, spoljašnje I/O nožice i tajmere). Startovanje se vrši izborom opcije **Debug-Start/Stop Debug Session**. Ako se uključi opcija **View-Disassembly Window** dobija se izvorni kod sa svojim asemblerskim prevodom. Ipak preglednije je pratiti izvršavanje programa u izvornom C programu (sl. 6).

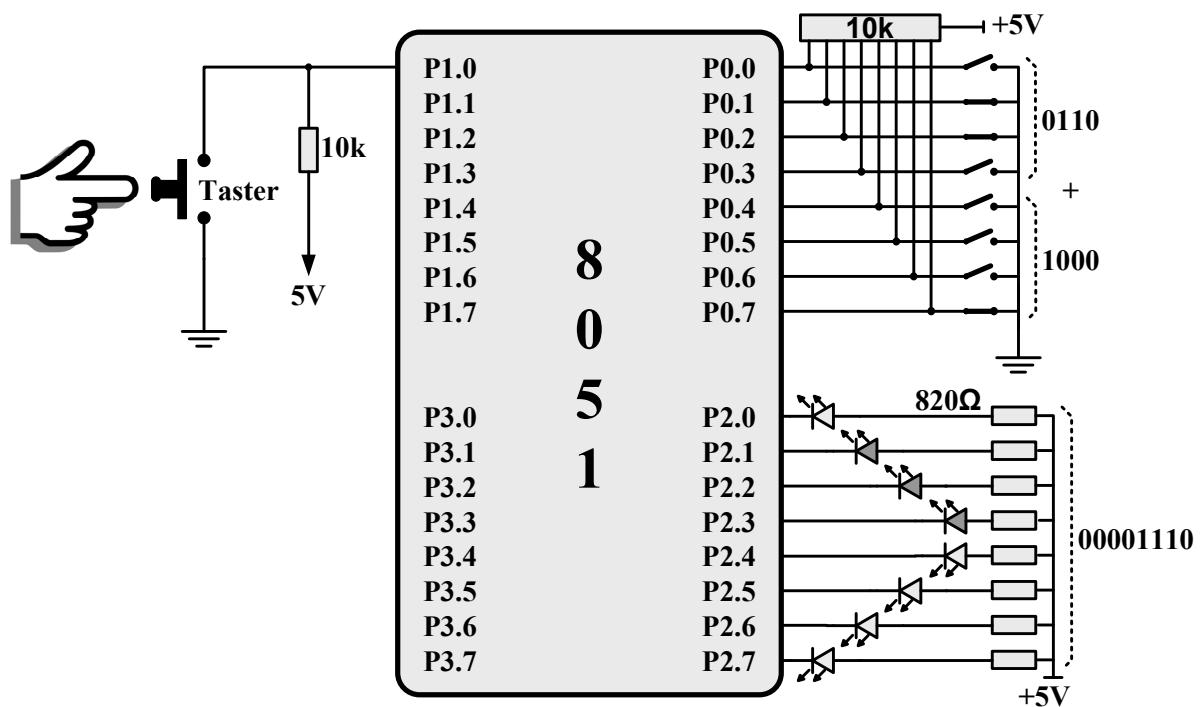


*Slika 6. Okruženje za testiranje programa i nalaženje grešaka*

Sa leve strane se vide registri u koje su smeštene promenljive, a takođe se može videti i sadržaj **PSW** registra. Pritiskom na taster **F11** izvršavaju se instrukcije jedna po jedna, pri čemu žuta strelica prati redosled izvršavanja programa. Istovremeno, sadržaj registara se menja zavisno od izvršene instrukcije. Mogu se posmatrati i lokalne promenljive (ovde je to **i**) uključenjem opcije **View-Watch&Call Stack Window**, kao i stanje na portovima izborom opcije **Peripherals-I/O-Ports** (ovde se posmatra samo port P1). Prekidanje simulacije se vrši izborom opcije **Debug-Stop Running (Debug-Start/Stop Debug Session)**. Ovaj vid simulacije je pogodan za proveru redosleda izvršavanja programa, matematičkih izračunavanja i osnovnog prikaza periferija.

## Primeri

Prost sabirač. Napisati program koji sabira prvi i drugi nibl porta P0 i rezultat prikazuje na port P2 tako što se uključuju odgovarajuće diode (sl. 7). Sabiranje se radi samo u slučaju kada se pritisne taster vezan na nožicu P1.0. Rad programa proveriti u Keil simulatoru (debageru).



Slika 7. Hardverska realizacija prostog sabirača

Rešenje:

```
#include <reg51.h>

//def. globalnih promenljivih

#define ulaz P0
#define izlaz P2

typedef unsigned char byte;

sbit saberi = P1^0;
bit taster_pritisnut;
byte prvi;
```



```

byte drugi;

void Inicijalizacija (void)
{
    ulaz = 0xFF; //port P0 je ulazni
    izlaz = 0xFF; //port P2 je rezultat
    saberi = 1 ;
    taster_pritisnut = 0;
    prvi = drugi = 0;
}


void main(void)                                //glavni program
{
    Inicijalizacija();                          // poziv potprograma za inicijalizaciju
    while(1)                                    // glavni program se vrti u ovoj petlji i
    {

        if(!saberu && !taster_pritisnut)
        {
            taster_pritisnut = 1;
            prvi = ulaz & 0x0F; // treba nam donji nibl
            drugi = ulaz >> 4;   // šiftovanje bajta za 4 mesta
            izlaz = ~(prvi+drugi); // ~ invertovanje jer je aktivna nula
        }
        if (saberu) taster_pritisnut = 0;
    }
}

```

Prvi korak u pisanju programa je uključivanje zaglavlja (`#include`), koje sadrži informacije specifične za korišćeni mikrokontroler. U ovom slučaju se koristi mikrokontroler 8051 koji kao zaglavlje koristi *reg51.h* dokument. Sledi definisanje promenljivih i ono se može odraditi na načine svojstvene sintaksi C jezika. Mogu se, na primer, koristiti makroi (`#define`), a mogu se definisati i korisnički tipovi (`typedef`). Ulazne promenljive su stanje porta P0 i prekidač vezan na nožicu P1.0, dok je izlaz vezan na port P2. Dozvoljeno je naravno koristiti i pomoćne promenljive u obradi.

Prva funkcija postavlja početne vrednosti i ovde je treba obratiti pažnju na način definisanja ulaznog porta (upisom FF), što je ranije već napomenuto. Posle inicijalizacije, program ulazi u beskonačnu petlju u kojoj proverava stanje prekidača na nožici P1.0 i kada je tester pritisnut vrši se sabiranje gornjeg i donjeg nibla porta P0. Do rezultata se stiže prostom obradom i on se ispisuje na port P2 u invertovanom obliku zato što su ledovke priključene na ovaj port aktivne na nizak nivo.

Pošto nemamo mogućnost da program isprobamo u realnom hardveru, proveru ćemo uraditi u simulatoru Keil okruženja (Debugger). Startujte simulator pritiskom na . Pritiskom na taster F11 pojavljuje se strelica koja pokazuje koji red programa se izvršava sledeći (sl. 8).

```

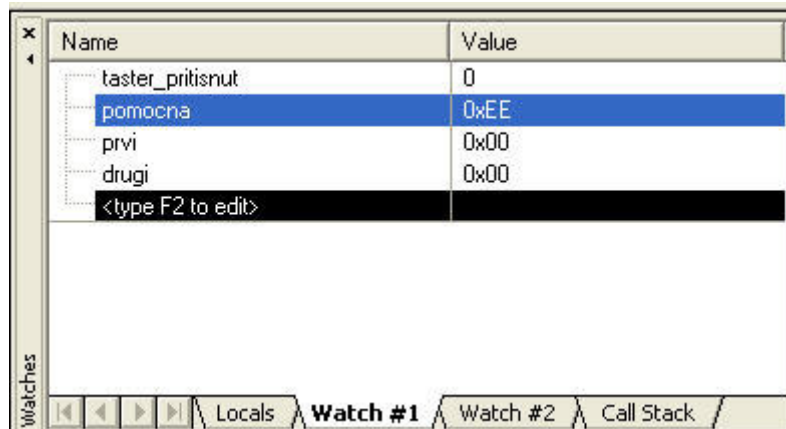
15 void Inicijalizacija (void)
16 {
17     ulaz = 0xFF; //port P0 je ulazni
18     izlaz = 00; //port P2 je rezultat
19     saberi = 0;
20     taster_pritisnut = 0;
21     pomocna = 0xee;
22     prvi = drugi = 0;
23 }

```

Slika 8. Korak po korak izvršavanje programa

Ukoliko postoje delovi programa koji nisu interesantni za posmatranje, možemo ih jednostavno preskočiti dodavanjem prekidnih tačaka (breakpoint). Na primer, nije nam interesantno da vidimo kako se izvršava funkcija inicijalizacije. Ali pre nego što se uvede prekidna tačka interesantno je pogledati kako se vrši inicijalizacija promenljivih. Za posmatranje promenljivih koristi se *Watch Window* koji se aktivira izborom opcije *Watch&Call Stack Window* iz menija *View*. Pojaviće se prozor u donjem desnom uglu radnog okruženja. U ovom prozoru se mogu posmatrati lokalne promenljive, stek, ali i promenljive koje smo mi definisali. Klikom na *Watch #1* jezičak, pojavljuje se novi prozor u kome se pritiskom na tipku F2 može uneti bilo koja definisana promenljiva. U toku simulacije od posebne koristi je mogućnost menjanja vrednosti promenljivih čime se testiranje ubrzava i pojednostavljuje.

U našem primeru imamo 4 definisane promenljive i možemo ih sve posmatrati. Kada dođe do promene vrednosti, promenljiva se označi plavom bojom (sl. 9).



Slika 9. Watch prozor sa posmatranje promenljivih

Postavlja se pitanje kako posmatrati portove mikrokontrolera, tj. kako postaviti ulazne signale i očitati rezultat? Keil okruženje poseduje *Peripherals* meni koji služi za rad sa periferijama. Ako izaberemo port P0, možemo podesiti ulazne signale (sl. 10). Takođe, na portu P2 posmatramo rezultat. Ukoliko želimo da saberemo donji i gornji nibl porta P0, potrebno je pritisnuti taster koji je vezan za nožicu P1.0, a to se takođe može odraditi u ovom meniju. Ako još napomenemo da se u ovom meniju mogu posmatrati i menjati još i prekidi, tajmeri, serijski port, onda posedujemo zadovoljavajući alat za simuliranje periferija.



*Slika 10. Rad sa portovima u Keil simulatoru*

Na nožici P1.0 se nalazi taster i jednostavno ukoliko želimo da pritisnemo taster, označimo kućicu kao na slici (sl. 10). Na slici je prikazan pritisnut taster. Na portu P0 postavimo gornji i donji nibl kao na slici (sl. 7). Vidimo da se dobija tačan rezultat ako se svi bitovi invertuju na portu P2. Invertovanje je potrebno zato što se ledovke uključuju logičkom nulom.

Kod komplikovanijih programa pojaviće se potreba da želimo da se uverimo u ispravnost rada samo određene linije ili dela programa. Izvršavanje programa korak po korak samo povećava ukupno vreme potrebno za projektovanje. Zato se koriste prekidne tačke. U našem primeru možemo postaviti prekidnu tačku u delu programa gde se ispituje da li je taster pritisnuti na taj način se preskače inicijalizacija pritiskom na tipku F5. Program će se izvršavati do trenutka kada se dostigne prekidna tačka. U ovom trenutku možemo posmatrati registre mikrokontrolera, korisničke promenljive ili „pritisnuti taster“. Ponovnim pokretanjem nastavlja se rad programa sve dok se ne dostigne nova prekidna tačka ili se dostigne kraj programa. U našem slučaju program je u beskonačnoj petlji pa će se on beskonačno izvršavati.