Izdelava mobilnega robota z razvojnim sistemom roboPIC (4)

Avtor: Silvan Bucik
E-pošta: silvan.bucik@tscng.net

Do sedaj smo se naučili prižigati led diodo, utripati z ledico ter uporabljati tipko. Opazili smo, da so prvi članki o razvojnem sistemu roboPIC v vas sprožili veliko zanimanje, saj ste se hitro odzvali. Da bi bilo naše sodelovanje čim boljše, bi ob tej priložnosti bralce povabil na spletne strani foruma (http://www.svet-el.si/phpBB2/index.php), kjer lahko predstavite svoje uspehe pri delu, s skupnimi močmi pa lahko rešimo marsikatero nastalo težavo.

Kaj bomo spoznali v tem prispevku? Posvetili bi se podrobnostim programskega jezika C, kajti premnogokrat se nam dogaja, da nam pri realizaciji lastnih idej primanjkuje poznavanje orodij. Spoznali bomo tipe spremenljivk ter njihovo rabo, vrste matematičnih in logičnih operatorjev jezika C ter ukaze za nadzor toka programa. Prepričani smo, da vam bo gradivo zelo koristilo.

Rešitve »domačih nalog«

Sedaj pa k nalogam. Pogledali si bomo rešitve »domačih nalog«, rešitve nagradnih nalog pa prepustimo za prihodnjo številko.
Naloga 1: Spremenite program tako, da bo led dioda na priključku RB3 svetila, ko bo tipka RA0 nepritisnjena. Ob pritisku nanjo pa naj led dioda ugasne.

```
// INICIALIZACIJA
void main (void)
{
TRISA = 0B11111111;    //PORTA = vhod, tudi priključek RA0
ADCON1 = 0x07;         //PORTA definiramo kot digitalna
                       vhodno-izhodna vrata
TRISB = 0B11110111;    //**** priključek RB3 definiramo kot
                         izhod
TRISC = 0B11111111;    //PORTC = vhod, ga ne uporabljamo
TRISD = 0B11111111;    //PORTD = vhod, ga ne uporabljamo
TRISE = 0B00000111;    //PORTE = vhod, ga ne uporabljamo

// GLAVNI PROGRAM
do
                       //uporabimo neskončno do-while zanko
{
if (RA0 != 0)          //**** ko tipka ni pritisnjena, je
                         logično stanje različno od 0
{
RB3 = 0;               //če je tipka nepritisnjena, led dioda
                         sveti
}
else                   //**** ko je tipka pritisnjena, je
                         logično stanje enako 0
{
RB3 = 1;               //če je tipka pritisnjena, led dioda
                         ugasne
}
}
while (1>0);           //pogledamo še enkrat, če ni sedaj kaj
                         drugače
}
```

Naloga 2: Spremenite program tako, da bo ob pritisku tipke RA4 gorela led dioda na izhodu RB5. Ob spustu tipke pa naj zasveti led dioda na izhodu RB3.

```
// INICIALIZACIJA
vois main (void)
{
TRISA = 0B11111111;    //PORTA = vhod, tudi priključek RA4
ADCON1 = 0x07;         //PORTA definiramo kot digitalna
                         vhodno-izhodna vrata
TRISB = 0B11010111;    //**** priključka RB3 in RB5 določimo
                         kot izhoda
TRISC = 0B11111111;    //PORTC = vhod, ga ne uporabljamo
TRISD = 0B11111111;    //PORTD = vhod, ga ne uporabljamo
TRISE = 0B00000111;    //PORTE = vhod, ga ne uporabljamo

// GLAVNI PROGRAM
while (1>0)            //uporabimo neskončno while zanko
{
if (RA4 == 0)          //ko je tipka pritisnjena, je logično
                         stanje enako 0
{
RB5 = 0;               //če je tipka pritisnjena, led dioda
                         na priključku
RB3 = 1;               //RB5 sveti, na priključku RB3 pa ne
}
else                   //**** ko tipka ni pritisnjena, je
                         logično stanje enako 1
{
RB3 = 0;               //če tipka ni pritisnjena, led dioda
                         na priključku
RB5 = 1;               //RB3 sveti, na priključku RB5 pa ne
}
}
}
```

Naloga 3: Spremenite program tako, da bo led dioda utripala hitreje (na primer 10 Hz), ali počasneje (na primer 2 Hz).

```
// INICIALIZACIJA
int frq=frekvenca_kHz;
void main (void)       //IME GLAVNE FUNKCIJE
{
TRISA = 0B11111111;    //PORTA definiramo kot vhod
ADCON1 = 0x07;         //PORTA definiramo kot digitalna
vhodno-izhodna vrata
```

```
TRISB = 0B11111110;    //**** RB0 določimo kot izhod, vsi
                         ostali priključki so vhodi
TRISC = 0B11111111;    //PORTC definiramo kot vhod
TRISD = 0B11111111;    //PORTD definiramo kot vhod
TRISE = 0B00000111     //PORTE definiramo kot vhod

// GLAVNI PROGRAM
while (1>0)
{
RB0 = 0;               //na izhodu RB0 se pojavi napetost 0V
                         (logična 0) - dioda se prižge
wait_mili(50);         //**** počakamo pol periode
                       //**** pri 10 Hz znaša parameter 50,
                       //**** pri 2 Hz znaša parameter 250
RB0 = 1;               //na izhodu RB0 se pojavi napetost +5V
                       (logična 1) - dioda ugasne
wait_mili(50);         //**** počakamo pol periode
                       //**** pri 10 Hz znaša parameter 50,
                       //**** pri 2 Hz znaša parameter 250
}                      // vse skupaj ponavljamo (while
                       zanka), da prične led dioda utripati
}
```

Naloga 4: Napišite program, da bodo utripale vse led diode na vratih PORTB.

```
// INICIALIZACIJA
int frq=frekvenca_kHz;
void main (void)       //IME GLAVNE FUNKCIJE
{
TRISA = 0B11111111;    //PORTA definiramo kot vhod
ADCON1 = 0x07;         //PORTA definiramo kot digitalna
                         vhodno-izhodna vrata
TRISB = 0B00000000;    //**** vse priključke vrat PORTB
                         določimo kot izhode
TRISC = 0B11111111;    //PORTC definiramo kot vhod
TRISD = 0B11111111;    //PORTD definiramo kot vhod
TRISE = 0B00000111     //PORTE definiramo kot vhod

// GLAVNI PROGRAM
while (1>0)
{
PORTB = 0B00000000;    //**** prižgemo vse led diode na
                         vratih PORTB
wait_mili(100);        //počakamo pol periode
PORTB = 0B11111111;    //**** ugasnemo vse led diode na
                         vratih PORTB
wait_mili(100);        //še enkrat počakamo pol periode
}                      //vse skupaj ponavljamo (while zanka),
                         da prične led dioda utripati
}
```

Spremenljivke v programskem jeziku C

Osnovni podatkovni objekti, nad katerimi se izvajajo operacije, so spremenljivke in konstante. Bistvena razlika med spremenljivko in konstanto je, kot že njuni imeni povesta, da se vrednost spremenljivke med tekom programa lahko ves čas spreminja, konstanta pa v celotnem programu zavzema isto vrednost in je ni mogoče spreminjati.

V začetku pisanja programa je potrebno vse spremenljivke deklarirati. Konstante, ki v programu pogosto nastopajo, pa je smiselno nadomestiti s prireditvenimi imeni.

V programskem jeziku C ločimo štiri tipe osnovnih spremenljivk. To so:

- bitne spremenljivke (1-bitne spremenljivke),
- znaki (8-bitna števila),
- nepredznačena cela števila (8-bitna, 16-bitna, 32-bitna števila),
- predznačena cela števila (8-bitna, 16-bitna, 32-bitna števila),
- števila s plavajočo vejico (32-bitna, 64-bitna, 80-bitna števila).

Natančen obseg posameznih tipov prikazuje sledeča tabela:

tip spremenljivke	dolžina [bitov]	obseg vrednosti
bit	1	0 ali 1
char	8	-128 do 127
unsigned char	8	0 do 255
int	16	-32,768 do +32,767
unsigned int	16	0 do 65535
long	32	-2,147,483,648 do +2,147,483,647
unsigned long	32	0 do 4,294,967,295
float	32	$3.4*10^{-38}$ do $3.4*10^{+38}$
double	64	$1.7*10^{-308}$ do $1.7*10^{+308}$
long double	80	$3.4*10^{-4932}$ do $3.4*10^{+4932}$

Tabela 1: Tipi spremenljivk in njihov obseg

Deklaracija spremenljivk

Pred uporabo je potrebno vsako spremenljivko posebej deklarirati. S tem smo določili njen značaj in namen uporabe. Včasih z deklaracijo določimo tudi začetno vrednost spremenljivke, ki pa se v toku izvajanja programa spreminja.

Primeri deklaracij spremenljivk:

```
char znak_0='a';       //spremenljivka znak_0 dobi začetno
                         vrednost a
int stevilo_0, stevilo_1=764;
                       //16-bitna spremenljivka stevilo_0
                         nima začetne vrednosti, 16-bitna
                         spremenljivka stevilo_1 ima začetno
                         vrednost 764
char stevilo_2= 0B01010011, stevilo_3=0x3A;
char stevilo_4 @0x50;  //16-bitna spremenljivka stevilo_4 ima
                       določen statični naslov (fiksni) 0x50
```

Vse spremenljivke razen stevilo_0 imajo z deklaracijo določene tudi začetne vrednosti. Opazimo, da so spremenljivke znak_0, stevilo_2, stevilo_3 in stevilo_4 tipa char, čeprav je le znak_0 zares znakovnega tipa. Prevajalnik namreč ne loči posebnih razlik med spremenljivko znakovnega tipa in 8-bitno spremenljivko. Prevajalnik bo vsem spremenljivkam spominske lokacije dodelil samodejno (poiskal bo prvo prosto mesto). Le spremenljivka stevilo_4 ima statično spominsko lokacijo (0x50); naslov spremenljivke smo sami določili.

Zapisovanje konstant

V programskem jeziku C ločimo več načinov zapisovanja števil in znakov:

764
desetiški zapis števila (število se ne sme začeti z 0),
0b01010011
dvojiški zapis števila (število se prične z znakoma 0b),
0x34
šestnajstiški zapis števila (število se prične z znakoma 0x),
0256
osmiški zapis števila (število se prične s črko 0),
1.8e-3
zapis števila s plavajočo vejico,
'a'
zapis konstante znakovnega tipa.

Polja

Več spremenljivk, ki ustrezajo skupnemu opisu, lahko združimo v skupino spremenljivk, ki jo imenujemo polje. Vsaka spremenljivka v polju zavzema svoje mesto in jo tako tudi obravnavamo. Pri uporabi spremenljivk v polju moramo nujno poleg imena spremenljivke navesti tudi pripadajoči indeks.

Primer deklaracije polja:

```
char senzor[3];          //polje s tremi elementi, elementi so
                           8-bitna števila
char senzor[3]= {0b11100010, 0b11001100, 0b11011100};
                         //polje s tremi elementi z definirani
                           mi začetnimi vrednostmi, elementi so
                           8-bitna števila
```

Gornja zapisa se razlikujeta v tem, da v drugem primeru istočasno z deklariranjem polja določamo tudi začetne vrednosti elementov, medtem ko v prvem primeru začetne vrednosti elementov niso določene. Zapis prikazuje primer polja senzor, ki vsebuje podatke o stanjih senzorjev preteklih treh merjenj. S klicanjem polja senzor imamo dostop do kateregakoli podatka zadnjih treh merjenj.

Primer uporabe:

```
senzor[2] = senzor[1]; //stare meritve se shranijo na višjo
                         lokacijo ter pripravimo
senzor[1] = senzor[0]; //na lokaciji senzor[0] prostor za
                         novo meritev
senzor[0] = PORTA;     // stanje na senzorjih, ki so priklju
                         čeni na vrata PORTA, se prenese v
                         polje senzor na lokacijo 0; to so
                         sveži podatki
```

Strukture

Strukture so v osnovi zelo podobne poljem, vendar jih za razliko od slednjih lahko sestavljajo elementi različnih tipov. Elementi struktur so samostojni elementi, različnih tipov in dolžin in skupaj tvorijo celoto.

Primer deklaracije strukture:

```
struct{
unsigned STOP:1;       //v strukturi control definiramo
                         spremenljivko STOP, ki zavzema 1 bit
unsigned NAPREJ:1;     //v strukturi control definiramo
                         spremenljivko NAPREJ,ki zavzema 1 bit
unsigned LEVO:1;       //v strukturi control definiramo
                         spremenljivko LEVO, ki zavzema 1 bit
unsigned DESNO:1;      //v strukturi control definiramo
                         spremenljivko DESNO,ki zavzema 1 bit
unsigned PODATEK:4;    //v strukturi control definiramo
                         spremenljivko PODATEK,ki zavzema 4 bite
}control;              //na koncu napišemo ime strukture
```

Primer uporabe:

```
if (control.STOP==1)   //če je v registru control postavljen
                         bit stop,
PORTC&= 0b11000110;    //potem naj se motorja ustavita
                         priključena sta preko vrat PORTC)
```

Kazalci

Kazalec je spremenljivka, ki vsebuje naslov neke druge spremenljivke. S kazalcem, katerega vsebina je naslov spremenljivke, izvajamo operacije nad spremenljivkami na posredni način. V prenekaterih primerih je uporaba kazalcev nepogrešljivo orodje. Vendar je za začetnika tovrstni način programiranja kar hud zalogaj, zato se bomo v začetnih korakih kazalcem izognili.

Primer deklaracije kazalca:

```
char *kazalec;         //kazalec na 8-bitno število (kazalec
                         prepoznamo po prefiksu *)
```

Pred praktičnim primerom bi omenili še dva znaka, ki spremljata delo s kazalci:

* prefiks pomeni vrednost spremenljivke na naslovu, na katerega kaže kazalec,
& prefiks pomeni naslov spremenljivke, na katerega kaže kazalec.

Primer uporabe kazalcev pri prepisu vsebine vrat PORTB na PORTC:

```
kazalec = &PORTB;      //v kazalec se shrani naslov vrat
                         PORTB (0x0006)
temp = *kazalec;       //v temp se shrani vrednost spremen
                         ljivke, katere naslov (0x0006,
                         naslov vrat PORTB) hrani kazalec
kazalec = kazalec+1;   //vrednost kazalca se poveča za 1
                         (sedaj znaša 0x0007)
*kazalec= temp;        //na naslov, kamor kaže kazalec
                         (0x0007, naslov vrat PORTC), se
                         prepiše vrednost iz spremenljivke
                         temp
```

Operatorji v programskem jeziku C

Poglejmo si, katere operacije uporabljamo pri programiranju s programskim jezikom C.

operacija:	primer:	opis:	pomen:
=	a = b		priredi spremenljivki a vrednost spremenljivke b
[ ]	[i]	a[2]	izbor elementa v polju
.	a.b	register.bit	izbor elementa v strukturi
*	*p	a= *p	vrednost spremenljivke na naslovu
&	&a	p= &a	vrednost naslova spremenljivke
,	char a,b;	char a; char b;	naštevanje
;	a = b;		znak ; označuje konec stavka

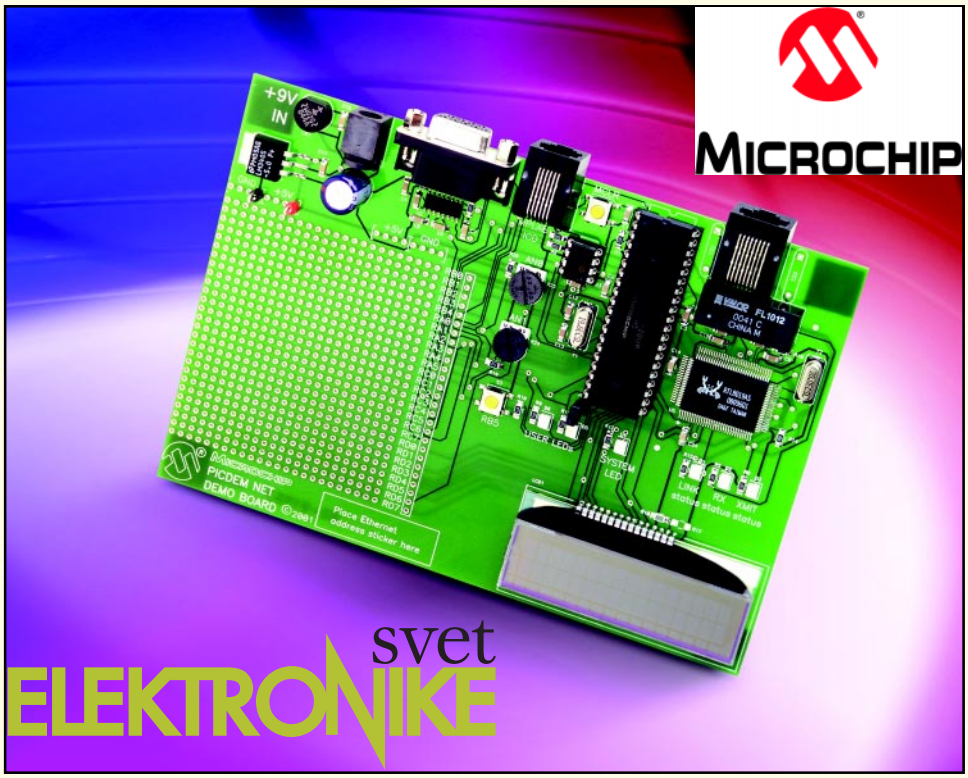
Ostali operatorji

Sodelujte v nagradnem žrebanju Microchipove PICDEM.net™ Lite Internet/Ethernet demo plošče

www.microchip-comp.com/se-picdem

Svet elektronike svojim bralcem nudi možnost osvojitve Microchip PICDEM.net Lite demo plošče in MPLAB ICD 2 razhroščevalnika.

PICDEM.net Lite demo plošča je pravzaprav Internet/Ethernet razvojna plošča z vgrajenim PIC18F452 mikrokontrolerjem in tovarniško vgrajenim TCP/IP skladom. Plošča podpira vsa 40-pinska DIP vezja s standardnim razporedom pinov, kakršnega imata PIC16F877 ali PIC18F452.



PICDEM.net plošča je namenjena eksperimentiranju z različnimi Microchipovimi TCP/IP rešitvami. Uporabniku je po inicialni nastavitvi IP naslova omogočen takojšnji dostop omrežju. Flash mikrokontroler omogoča modifikacije v demo programu oziroma dodajanje aplikacijskega programa.

Na demo plošči se nahaja tudi stabiliziranih 5 V za napajanje dodatnih senzorjev ali testnih uporabniških vezij. Z namenom razvoja lastne aplikacije lahko v čip naložimo tudi druge standardne ali uporabniške strežniške programe.

Na plošči je uporabljen brezplačen Microchipov TCP/IP sklad, ki je opisan v referenčni aplikaciji AN833 (DS00833). V omenjenem dokumentu so na voljo tudi primeri kode.

Microchip TCP/IP sklad je garnitura programov, ki omogoča uporabo tako standardnih TCP/IP aplikacij (HTTP Server, Mail Client, itn.), kot tudi uporabniških TCP/IP aplikacij. Da bi lahko uporabili TCP/IP sklad, se uporabnikom ni potrebno učiti vseh njegovih zapletenih specifikacij in svoj trud lahko osredotočijo le na HTTP strežniško aplikacijo, ki pa ne zahteva poznavanje TCP/IP protokola.

TCP/IP sklad je implemetiran na modularen način, z vsemi svojimi zmožnostmi kreiranja visoko abstraktnih plasti, ki jim lahko dostopamo iz katerekoli plasti neposredno pod njimi. Sklad je napisan v programskem jeziku C, namenjen je Microchipovim prevajalnikom C18 in HI-TECH PICC 18 in je predviden le za delovanje v Microchipovi PIC 18 družini mikrokontrolerjev.

Čeprav je ta implementacija namenjena le za delovanje na PICDEM.net Internet/Ethernet demo plošči, jo lahko enostavno prilagodimo tudi za delovanje v katerikoli napravi, ki je opremljena s PIC18 mikrokontrolerjem. PICDEM.net vsebuje Ethernet in RS-232 komunikacijski vmesnik. HTML spletne strani, ki jih generira PICmicro® mikrokontroler, si lahko ogledujemo s katerimkoli standardnim spletnim brskalnikom (npr. Microsoft® Explorer). Začetna konfiguracija plošče (nastavitev IP naslova) se izvaja preko RS-232 vrat s pomočjo standardnega terminalskega programa. Demo plošča je opremljena tudi s 6-pinskim modularnim konektorjem za direktno povezavo z MPLAB® ICD 2 razhroščevalnikom. S pomočjo MPLAB® ICD 2 lahko uporabniki spreminjajo oz. preprogramirajo flash PICmicro mikrokontroler in tako prilagajajo delovanje sistema svojim potrebam. Na plošči je tudi veliko prostora za dodajanje raznih dodatnih komponent potrebnih za izvedbo lastnih aplikacij. Površina zadostuje za namestitev vgradnega modema, ki bo zagotavljal klicno zmogljivost. Na voljo je tudi več statusnih indikatorjev in uporabniških vmesnikov, vključno s 16 x 2 LCD prikazovalnikom in LEDikami.

Da bi sodelovali v nagradnem žrebanju tega razvojnega kita, je dovolj, da se prijavite na www.microchip-comp.com/se-picdem.

operacija:	primer:	opis:	pomen:
+	a+b		seštevanje dveh števil
-	a-b		odštevanje dveh števil
*	a*b		množenje dveh števil
/	a/b		deljenje dveh števil
%	a%b		ostanek po celoštevilskem deljenju (modulo)
( )	(a+b)*c		matematični oklepaj
++	++a	a= a+1	poveča vrednost spremenljivke za 1 in jo nato uporabi (predinkrement)
++	a++	a= a+1	uporabi spremenljivko in jo nato poveča za 1 (postinkrement)
--	--a	a= a-1	zmanjša vrednost spremenljivke za 1 in jo nato uporabi (preddekrement)
--	a--	a= a-1	uporabi spremenljivko in jo nato zmanjša za 1 (postdekrement)
-	-a	a= -a	negativna vrednost (predznak minus)
+=	a+=b	a= a+b	seštevanje dveh števil in prirejanje
-=	a-=b	a= a-b	odštevanje dveh števil in prirejanje
*=	a*=b	a= a*b	množenje dveh števil in prirejanje
/=	a/=b	a= a/b	deljenje dveh števil in prirejanje
%=	a%=b	a= a%b	modulo dveh števil in prirejanje

Aritmetični operatorji:

operacija:	primer:	opis:	pomen:
&	a&b		bitna in operacija
|	a|b		bitna ali operacija
^	a^b		bitna izključna ali operacija
~	~a		bitna ne operacija nad spremenljivko (eniški komplement)
!	!RB0		bitna ne operacija nad posameznim bitom
<<	a<<b		logični pomik v levo (pomakne vrednost spremenljivke a za b bitov v levo)
>>	a>>b		logični pomik v desno (pomakne vrednost spremenljivke a za b bitov v desno)
&=	a&=b	a= a&b	bitni in in prirejanje
|=	a|=b	a= a|b	bitni ali in prirejanje
^=	a^=b	a= a^b	bitni izključni ali in prirejanje
<<=	a<<=b	a= a<<b	pomik levo in prirejanje
>>=	a>>=b	a= a>>b	pomik desno in prirejanje

Logični operatorji:

Kontrola poteka programa

Stavki za kontrolo toka programa v programskem jeziku določajo vrstni red izvajanja programa. V programskem jeziku C ločimo dve vrsti stavkov:

operacija:	primer:	opis:	pomen:
<	a<b		manjše kot
>	a>b		večje kot
?	( )?r1:r2	c= (a>b)?r1:r2	pogojni operator (če je izraz v oklepaju resničen, je rezultat operacije enak spremenljivki r1, sicer pa je enak spremenljivki r2)
==	a==b		enako kot
!=	a!=b		različno od
>=	a>=b		večje ali enako kot
<=	a<=b		manjše ali enako kot
&&	( )&&( )	(a>0)&&(b<3)	logični test in
||	( )||( )	(a>0)||(b<3)	logični test ali

Relacijski operatorji:

- logične odločitve in
- programske zanke.

Logične odločitve

K logičnim odločitvam štejemo tiste stavke, ki poskrbijo za vejitev programa. V to skupino spadajo:

- stavek if-else,
- stavek switch-case in
- programski skok goto.

Stavek if-else ter programski skok goto sta bila opisana že v pretekli številki, zato ju ne bi ponovno opisovali.

Stavek switch-case

Poglejmo si uporabo switch-case stavka. Stavek switch-case uporabljamo v primerih, ko se program lahko nadaljuje po več različnih poteh. Na nek način je stavek switch-case nadgradnja stavka if-else, pri slednjem se program lahko nadaljuje le po dveh različnih poteh. Pot, po kateri se bo program nadaljeval, je odvisen od izpolnjevanja pogojev (case).

Sintaksa:

```
switch(a)
{
case 0: akcija_0;       //če je a enak 0, se izvede del
                          programa akcija_0
break;                  //izhod iz zanke
case 1: akcija_1;       //če je a enak 1, se izvede del
                          programa akcija_1
break;                  //izhod iz zanke
case 2: akcija_2;       //če je a enak 2, se izvede del
                          programa akcija_2
break;                  //izhod iz zanke
default: akcija_drugo; //v ostalih primerih se izvede del
                          programa akcija_drugo
break;                  //izhod iz zanke
}
```

Primer uporabe:

```
switch (senzor_stat)   // testiranje pogojev
{
case 0B00000000:
mot2a=1;               // del programa, ki se izvede v
```

```
primeru,
mot2b=0;              // ko je vrednost
mot1a=0;              // spremenljivke senzor_stat
mot1b=1;              // enaka 0b00000000
break;

case 0B00000001:
mot2a=1;              // del programa, ki se izvede v
primeru,
mot2b=0;              // ko je vrednost
mot1a=1;              // spremenljivke senzor_stat
mot1b=0;              // enaka 0b00000001
break;

case 0B00000100:
mot2a=0;              // del programa, ki se izvede v
primeru,
mot2b=1;              // ko je vrednost
mot1a=1;              // spremenljivke senzor_stat
mot1b=0;              // enaka 0b00000100
break;

default:
mot2a=1;              // del programa,
mot2b=0;              // ki se izvede
mot1a=0;              // v ostalih
mot1b=1;              // primerih
break;
}
```

Programske zanke

Programske zanke so deli programa, ki se večkrat ponovijo. Število ponovitev je odvisno od izpolnjevanja pogojev, ki se pred vsako ponovitvijo zanke preverjajo. V to družino stavkov spadajo:

- programska zanka while,
- programska zanka do-while,
- programska zanka for ter
- stavka break in continue.

Zanki while ter do-while sta nam že znani iz pretekle številke, ogledali si bomo še preostali dve.

Programska zanka for

Programska zanka for je nadgradnja zanke while. Za razliko od zanke while, ki je od parametrov potrebovala le pogoj testiranja, potrebuje zanka for tri začetne parametre, ki so med seboj ločeni s podpičji, in sicer:

- začetno vrednost,
- pogoj testiranja in
- korak zanke (reinicializacija).

Sintaksa:

```
for (začetna_vrednost, pogoj_testiranja, korak_zanke)
{
akcija_1;
}
```

Pred pričetkom izvajanja zanke se nastavijo začetne vrednosti spremenljivk, nato se izvede zanka. Ob zaključku zanke se izvede še operacija korak zanke. Zanka se izvaja tako dolgo, dokler je pogoj izpolnjen. Preverjanje se vrši vedno v začetku zanke, šele nato se izvede akcija_1.

Primer uporabe:

```
for (st_ponovitev=0;st_ponovitev<3;st_ponovitev=st_ponovi-
tev+1)                //začetna vrednost spremenljivke
                      //st_ponovitev je enaka 0
{                     //tu se izvede preverjanje, če je
st_ponovitev<3
servo2_out(51);       //izvaja
                      //se
wait_sec(5);          //sledeči
servo2_out(77);       //del
                      //programa
wait_sec(5);          //vse
servo2_out(102);      //do
                      //tukaj
wait_sec(5);          //oziroma do tukaj
}                     //ob zaključku zanke se spremenljivka
                      //st_ponovitev poveča za 1
```

Z uporabo programske zanke while bi isti program izgledal takole:

```
st_ponovitev=0;       //pred pričetkom izvajanja zanke
                      //nastavimo začetno vrednost
while(st_ponovitev<3) //testiranje pogojev
{
servo2_out(51);
                      wait_sec(5);
                      servo2_out(77);
                      wait_sec(5);
                      servo2_out(102);
                      wait_sec(5);
st_ponovitev=st_ponovitev+1
                      //korak zanke
}
```

Stavka break in continue

Stavka break in continue uporabljamo za prekinitev ali nadaljevanje programske zanke. S stavkom break prekinemo izvajanje katerekoli zanke, kljub temu, da so pogoji izvajanja zanke izpolnjeni. S stavkom continue pa dosežemo ravno nasprotno: programska zanka se bo vedno ponovila, ne glede na to, ali so pogoji izvajanja zanke izpolnjeni ali ne.

Sintaksi:

```
break;                // prekine izvajanje zanke
continue;             // nadaljuje izvajanje zanke
```

Primer uporabe:

```
for (stevec=0;stevec<5;stevec++)
{
if(stevec==2)
{
continue;             //v primeru, da je spremenljivka
                        stevec enaka 2
                      //(četudi je vrednost spremenljivke
                        stevec manjša kot 5),
                      //se izvajanje zanke ustavi in
}
                      //vrne na začetek for zanke
PORTB<<=1;
```

```
if(stevec==4)
{
break;                  //kljub temu, da je pogoj izpolnjen
}
                        //(vrednost stevec manjša kot 5),
}
                        //se izvajanje zanke predčasno prekine
                        //in nadaljuje izven for zanke
```

Zbirniški ukazi

Pa še to. Včasih se pojavi zahteva, da moramo del programa napisati v zbirnem jeziku. To je nižji programski jezik, ki ga uporabljamo pri programiranju mikrokontrolerjev. Po naravi je primitivnejši, to pomeni, da je potrebno opravila, ki jih v višjih programskih jezikih zapišemo v eni vrstici (primer matematične operacije množenja), v zbirniku zapišejo v več korakih. Prednost zbirnika pred višjimi programskimi jeziki je predvsem v krajši programski kodi in posledično hitrejšem izvajanju programa.

Sintaksa 1:

```
#asm                    //označuje pričetek zbirniške kode
programska koda
#endasm                 //označuje konec zbirniške kode
```

Sintaksa 2:

```
asm ("ukaz");           //vnos posamičnega ukaza v zbirniškem
                          jeziku
```

Primer uporabe:

```
#asm
nop                     //del programa
movlw 0x4a              //v
movwf portb             //zbirniškem jeziku
#endasm
asm ("movlw 0x3e");     //ukaz v zbirniškem jeziku
asm ("movwf portc");    //ukaz v zbirniškem jeziku
```

Do prihodnje številke

S teorijo smo danes zaključili. Upam, da ni bilo preveč dolgočasno. Prihodnjič pa zopet nadaljujemo z zanimivejšimi stvarmi – pričeli bomo s krmiljenjem enosmernih motorčkov. Pripravite se. Pa lep pozdrav! ●

Literatura:

[1] HI-TECH, PICC Lite C Manual, HI-TECH Software, 2002, http://www.htsoft.com/downloads/manuals.php;
[2] B.W. Kernighan, D.M. Ritchie, Programski jezik »C«, Ljubljana: Fakulteta za elektrotehniko, 1991.