

U n i v e r z i t e t u N i š u
E l e k t r o n s k i f a k u l t e t

Vladan Vučković, Teufik Tokić

PROGRAMIRANJE MIKROKONTROLERA



Edicija: Osnovni udžbenici

U n i v e r z i t e t u N i š u
E l e k t r o n s k i f a k u l t e t

Vladan Vučković, Teufik Tokić

PROGRAMIRANJE MIKROKONTROLERA

PROGRAMIRANJE MIKROKONTROLERA

Autori: Prof. dr Vladan Vučković, Prof. dr Teufik Tokić

Izdavač: Elektronski fakultet u Nišu
P. fah 73, 18000 Niš
<http://www.elfak.ni.ac.rs>

Recenzenti: prof. dr Ivan Milentijević, Elektronski fakultet u Nišu
doc. dr Vladimir Ćirić, Elektronski fakultet u Nišu

Glavni i odgovorni urednik: prof. dr Dragan Mančić

Odlukom Nastavno-naučnog veća Elektronskog fakulteta u Nišu, br. 07/05-

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

**Preštampavanje ili umnožavanje ove knjige nije dozvoljeno bez pismene
dozvole izdavača.**

Tiraž: 100 primeraka

Štampa: SVEN d.o.o., Niš

U n i v e r z i t e t u N i š u
E l e k t r o n s k i f a k u l t e t

Vladan Vučković, Teufik Tokić

PROGRAMIRANJE MIKROKONTROLERA



Edicija: Osnovni udžbenici

2016.

Predgovor

Pomoćni udžbenik **“Programiranje mikrokontrolera”** obezbeđuje neophodne informacije studentima za savladavanje gradiva iz skladu je sa planom i programom predmeta Mikroračunarski sistemi i Mikrokontroleri i programiranje na Osnovnim akaedmskim studijama.

Udžbenik je namenjen studentima elektronskih elektrotehničkih fakulteta, drugih tehničkih fakulteta, kao i ostalih srodnih fakulteta kao udžbenik u oblasti Računarske tehnike, a studentima računarstva i informatike obezbeđuje dobru osnovu i detaljan pregled osnovnih tema iz ove oblasti nauke i tehnologije. Studentima drugih studijskih programa elektronskih i elektrotehničkih fakulteta, kao i drugih tehničkih i prirodno-matematičkih fakulteta pruža uvid u ovu oblast računarstva i informatike i mevanje osnovnih metoda, tehnologija i principa ove naučne i inženjerske discipline koja veoma značajno utiče na sve aspekte savremenog ljudskog društva.

Autori se zahvaljuju recenzentima, prof. dr Ivanu Milentjeviću i doc. dr Vladimiru Ćiriću na angažovanju i trudu koji su uložili recenzirajući ovaj pomoćni udžbenik, kao i na zapažanjima, primedbama i korisnim sugestijama, koje su doprinele kvalitetu ovog udžbenika.

Takođe, autori se zahvaljuju dipl. ing. Nini Aritonović za tehničku podršku u realizaciji rukopisa.

Autori

Sadržaj

1	UVOD	1
2	HARDVERSKI OPIS PIC16F84	3
2.1	Primene	4
2.2	Portovi mikrokontrolera	6
2.3	Opis pinova	7
2.4	PIC reset I oscillator	8
2.4.1	Tipovi oscilatora	8
2.4.2	XT oscillator	8
2.4.4	Vrste reseta	9
2.5	Organizacija memorije	10
2.5.1	Organizacija programske memorije	10
2.6	Registri mikrokontrolera	11
2.6.1	Namena registara	11
2.6.1.1	STATUS registar	12
2.6.1.2	OPTION_REG registar	13
2.6.1.3	INTCON registar	14
2.7	Tajmer0 modul I TMR0 registar	15
2.8	Načini adresiranja podataka	16
2.9	EEPROM Memorija	19
2.10	Prekidi	22
2.11	Prilog	24
3	PREGLED MICROC PROGRAMSKOG OKRUŽENJA	31
3.1	Code editor (*Editor koda)	32
3.1.1	Code Assistant [CTRL+SPACE] (Pomoć za Kod)	33
3.1.2	Parameter Assistant [CTRL+SHIFT+SPACE] (*Pomoć za parametre)	33
3.1.3	Code Template [CTRL+J] (*Obrazac Koda)	33
3.1.4	Auto Correct (*Automatska Ispravka)	33
3.1.5	Comment/Uncomment (*Postavljanje/Uklanjanje oznaka za komentar)	34
3.1.6	Bookmarks (*Obeleživač)	34
3.1.7	Goto Line (*Idi na liniju)	34

3.2	Code explorer (*Preglednik koda)	34
3.3	Debugger (*Prečistač koda)	35
3.3.1	Primer	35
3.3.2	Run/Pause Debugger [F6]	36
3.3.3	Step Into [F7]	36
3.3.4	Step Out[CTRL+F8]	36
3.3.5	Run to Cursor[F4]	36
3.3.6	Toggle Breakpoint[F5]	36
3.4	Promenljive	36
3.4.1	Stopwatch Window(*Prozor Štoperice)	37
3.4.2	View RAM Window (*Prozor Pregleda RAM-a)	37
3.5	Error window (*Prozor greške)	38
3.6	Statistics (*Statistika)	39
3.6.1	Memory Usage Window (*Prozor Iskorištenja Memorije)	39
3.6.2	Procedure (Graph) Window (*Prozor Procedura (Graf))	39
3.6.3	Procedures (Locations) Window (*Prozor Procedura(Lokacije))	40
3.6.4	Procedures (Details) Window (*Prozor Procedura(Detalji))	40
3.6.5	RAM Window (*Prozor RAM-a)	40
3.6.6	ROM Window (*Prozor ROM-a)	41
3.7	Integrated tools (*Integrirani alati)	41
3.7.1	Usart Terminal	41
3.7.2	ASCII Chart (*ASCII Tabela)	42
3.7.3	7 Segment Display Decoder (7-mo Segmentni Displej Dekoder)	42
3.7.4	EEPROM Editor	43
3.7.5	mikroBootLoader	43
3.7.6	Integriranje Korisničkog i Boot koda	44
3.8	Keyboard shortcuts (*Prečice sa tastature)	44
3.8.1	IDE Prečice	44
3.8.2	Osnovne prečice editor	45
3.8.3	Napredne prečice editor	45
3.8.4	Prečice Debugger-a	45
3.1.5	Izrada aplikacija	46
3.9	Projects (*Projekti)	46
3.9.1	New Project (*Novi Projekt)	46
3.9.2	Editing Project(*Uređivanje projekta)	46
3.9.3	Add/Remove Files from Project (*Dodajte/Odstranite Fajlove iz Projekta)	46
3.9.4	Proširena funkcionalnost Project Files odeljka	46
3.10	Source files (*Izvorni fajlovi)	48
3.10.1	Putanje Traženja	48
3.10.2	Putanje za izvorne fajlove (.c)	48
3.10.3	Putanje za fajlove zaglavlja (.h)	48
3.10.4	Upravljanje Izvornim Fajlovima	48
3.10.5	Kreiranje novog izvornog fajla	48
3.10.6	Otvaranje postojećeg fajla	49
3.10.7	Štampanje otvorenog fajla	49

3.10.8	<i>Snimanje fajla</i>	49
3.10.9	<i>Zatvaranje fajla</i>	49
3.11	<i>Compilation (*Kompajliranje)</i>	50
3.11.1	<i>Intel HEX file (.hex)</i>	50
3.11.2	<i>Binary mikro Compiled Library (.mcl)</i>	50
3.11.3	<i>List File (.lst)</i>	50
3.11.4	<i>Assembler File (.asm)</i>	50
3.12	<i>Poruke o greškama</i>	50
3.12.1	<i>Compiler Warning Messages (*Kompajlerove Poruke Upozorenja)</i>	52
4	OSNOVNE KARAKTERISTIKE MIKROC JEZIKA	55
4.1	<i>Pic specifičnosti</i>	55
4.1.1	<i>Efikasnost tipova</i>	55
4.1.2	<i>Ograničenja Ugnježdenih Poziva</i>	55
4.2	<i>PIC16xxx Specifičnosti</i>	56
4.2.1	<i>Proboj Stranice</i>	56
4.2.2	<i>Ograničenja Indirektnog Pristupa Kroz FSR</i>	56
4.3	<i>mikroC specifičnosti</i>	56
4.3.1	<i>Pitanja ANSI Standarda</i>	56
4.3.2	<i>Odstupanje od ANSI C Standarda</i>	56
4.3.3	<i>Ponašanje definisano implementacijom</i>	56
4.3.4	<i>Predefinisane Globalne i Konstante</i>	56
4.3.5	<i>Pristupanje Pojedinačnim Bitovima</i>	56
4.3.6	<i>Prekidi</i>	58
4.3.7	<i>Funkcijski Pozivi iz Prekida</i>	58
4.3.8	<i>Direktive Povezivača</i>	58
4.3.9	<i>Direktiva absolute</i>	59
4.3.10	<i>Direktiva org</i>	59
4.3.11	<i>Optimizacija koda</i>	59
4.3.12	<i>Sklapanje konstanti</i>	59
4.3.13	<i>Propagacija konstanti</i>	59
4.3.14	<i>Propagacija kopije</i>	59
4.3.15	<i>Numeracija vrednosti</i>	59
4.3.16	<i>Eliminacija „Mrtvog Koda“</i>	59
4.3.17	<i>Alokacija stack</i>	59
4.3.18	<i>Optimizacija lokalnih promenljivih</i>	59
4.3.19	<i>Bolje generisanje koda i lokalna optimizacija</i>	59
4.3.20	<i>Indirektni Funkcijski Pozivi</i>	59
4.4	<i>Leksički elementi</i>	60
4.4.1	<i>Prazan proctor</i>	61
4.4.2	<i>Komentari</i>	61
4.4.3	<i>C komentari</i>	61
4.4.4	<i>C++ komentari</i>	61
4.5	<i>Mikro ICD (Debugger na nivou kola)</i>	62
4.5.1	<i>Opcije mikro ICD Debugera</i>	63

4.5.2	<i>Mikro ICD Debugger Primer</i>	64
4.6	Mikro ICD (Debugger na nivou kola)	68
4.6.1	<i>Watch Window (*Prozor Pregleda)</i>	68
4.6.2	<i>View EEPROM Window (*Prozor Pregleda EEPROM-a)</i>	69
4.6.3	<i>View Code Window (*Prozor Pregleda Koda)</i>	69
4.6.4	<i>View RAM Window (*Prozor Pregleda RAM-a)</i>	70
4.6.5	<i>Uobičajene Greške</i>	71
4.7	Tokeni	71
4.7.1	<i>Primer Ekstrakcije Tokena</i>	71
4.8	Konstante	72
4.8.1	<i>Celobrojne konstante</i>	72
4.8.2	<i>Sufiksi Long i Unsigned</i>	72
4.8.3	<i>Decimalne konstante</i>	73
4.8.4	<i>Heksadecimalne Konstante</i>	73
4.8.5	<i>Binarne Konstante</i>	73
4.8.6	<i>Oktalne Konstante</i>	73
4.8.7	<i>Konstante u Pokretnom Zarezu</i>	74
4.8.8	<i>Znakovne Konstante</i>	74
4.8.9	<i>String Konstante</i>	75
4.8.10	<i>Nastavljanje linije kosom crtom unazad</i>	76
4.8.11	<i>Konstante pobrojanja</i>	76
4.8.12	<i>Pokazivačke Konstante</i>	76
4.8.13	<i>Konstantni Izrazi</i>	76
4.9	Ključne reči	77
4.10	Indetifikatori	77
4.10.1	<i>Velika i Mala Slova</i>	77
4.10.2	<i>Jedinstvenost i Doseg</i>	78
4.11	Znaci interpunkcije	78
4.11.1	<i>Srednje Zgrade</i>	78
4.11.2	<i>Male Zgrade</i>	78
4.11.3	<i>Velike Zgrade</i>	78
4.11.4	<i>Zarez</i>	79
4.11.5	<i>Tačka-zarez</i>	79
4.11.6	<i>Dvostruka Tačka</i>	79
4.11.7	<i>Asterisk (Deklaracija Pokazivača)</i>	79
4.11.8	<i>Znak Jednakosti</i>	80
4.11.9	<i>Znak „Taraba“ (Pred-procesorska Direktiva)</i>	80
4.12	Objekti i L-vrednosti	80
4.12.1	<i>Objekti</i>	80
4.12.2	<i>Objekti i Deklaracije</i>	81
4.12.3	<i>L-vrednosti</i>	81
4.12.4	<i>R-vrednosti</i>	81
4.13	Doseg i vidljivost	82
4.13.1	<i>Doseg</i>	82
4.13.2	<i>Blokovski Doseg</i>	82

4.13.3	<i>Fajl Doseg</i>	82
4.13.4	<i>Funkcijski doseg</i>	82
4.13.5	<i>Doseg Funkcijskog Prototipa</i>	82
4.13.6	<i>Vidljivost</i>	82
4.14	<i>Prostori imena</i>	83
4.14.1	<i>Imena goto labela</i>	83
4.14.2	<i>Strukture, unije, i oznake pobrojanja</i>	83
4.14.3	<i>Imena članova struktura i unija</i>	83
4.14.4	<i>Promenljive, typedef imena, finkcije i članovi pobrojanja</i>	83
4.15	<i>Trajanje</i>	84
4.15.1	<i>Statičko Trajanje</i>	84
4.15.2	<i>Lokalno Trajanje</i>	84
4.16	<i>Tipovi</i>	85
4.16.1	<i>Kategorije Tipova</i>	86
4.17.2	<i>Celobrojni Tipovi</i>	86
4.17.3	<i>Tipovi Pokretnog Zareza</i>	86
4.17.4	<i>Pobrojanja</i>	87
4.17.5	<i>Deklaracija Pobrojanja</i>	88
4.17.6	<i>Anonimni Enum Tip</i>	88
4.17.7	<i>Tip Void (*Prazno)</i>	89
4.17.8	<i>Void funkcije</i>	89
4.17.9	<i>Generički Pokazivači</i>	89
4.18	<i>Izvedeni tipovi</i>	89
4.18.1	<i>Deklaracija Niza</i>	90
4.18.2	<i>Nizovi u Izrazima</i>	91
4.18.3	<i>Višedimenzionalni Nizovi</i>	91
4.18.4	<i>Pokazivači</i>	91
4.18.5	<i>Deklaracije Pokazivača</i>	92
4.18.6	<i>Null Pokazivači</i>	92
4.18.7	<i>Dodeljivanje adrese Funkcijskom Pokazivaču</i>	93
4.18.8	<i>Aritmetika Pokazivača</i>	94
4.18.9	<i>Nizovi i Pokazivači</i>	94
4.18.10	<i>Dodela i Poređenje</i>	95
4.18.11	<i>Dodavanje Pokazivaču</i>	95
4.18.12	<i>Strukture</i>	96
4.18.13	<i>Deklaracija i Inicijalizacija Struktura</i>	97
4.18.14	<i>Nepotpune Deklaracije</i>	97
4.18.15	<i>Neoznačene Strukture i Typedef</i>	98
4.18.16	<i>Strukturalne Dodele</i>	98
4.18.17	<i>Veličina Strukture</i>	98
4.18.18	<i>Strukture i Funkcije</i>	98
4.18.19	<i>Pristupanje Ugnježdenim Strukturama</i>	99
4.18.20	<i>Jedinstvenost Struktura</i>	100
4.18.21	<i>Unije</i>	100
4.18.22	<i>Deklaracija Unija</i>	100

4.18.23	<i>Veličina Unije</i>	101
4.18.24	<i>Pristup Članovima Unije</i>	101
4.18.25	<i>Bit Polja</i>	101
4.18.26	<i>Deklaracija Bit Polja</i>	102
4.19	<i>Konverzije tipova</i>	102
4.19.1	<i>Standardne Konverzije</i>	103
4.19.2	<i>Aritmetičke Konverzije</i>	103
4.19.3	<i>Konverzije Pokazivača</i>	104
4.20	<i>Deklaracije</i>	105
4.20.1	<i>Uvod u Deklaracije</i>	105
4.20.2	<i>Deklaracije i Definicije</i>	105
4.20.3	<i>Deklaracije i Deklaratori</i>	106
4.20.4	<i>Povezivanje</i>	106
4.20.5	<i>Pravila Povezivanja</i>	107
4.20.5	<i>Klase Skladištenja</i>	108
4.20.6	<i>Register</i>	108
4.20.7	<i>Static</i>	108
4.20.8	<i>Extern</i>	108
4.20.9	<i>Kvalifikatori Tipa</i>	109
4.20.10	<i>Kvalifikator const</i>	109
4.20.11	<i>Kvalifikator volatile</i>	109
4.20.12	<i>Typedef Specifikator</i>	109
4.20.13	<i>Asm Deklaracija</i>	110
4.20.14	<i>Migracija sa starijih verzija mikroC-a</i>	111
4.20.15	<i>Inicijalizacija</i>	111
4.20.16	<i>Automatska Inicijalizacija</i>	112
4.21	<i>Funkcije</i>	112
4.21.1	<i>Deklaracija Funkcija</i>	113
4.21.2	<i>Prototipovi Funkcija</i>	113
4.21.3	<i>Definicija Funkcije</i>	114
4.21.4	<i>Ponovna Ulažnost Funkcija</i>	114
4.21.5	<i>Pozivi Funkcija</i>	114
4.21.5	<i>Konverzije Argumenta</i>	115
4.21.5	<i>Operator Tri Tačke ('...')</i>	116
4.22	<i>Operatori</i>	116
4.22.1	<i>Prvenstvo i Asocijativnost Operatora</i>	117
4.22.2	<i>Aritmetički Operatori</i>	118
4.22.3	<i>Relacioni Operatori</i>	119
4.22.4	<i>Pregled Relacionih Operatora</i>	119
4.22.5	<i>Relacioni Operatori u Izrazima</i>	120
4.22.6	<i>Operatori na Nivou Bitova</i>	120
4.22.7	<i>Pregled Operatora na Nivou Bitova</i>	120
4.22.8	<i>Bitsko naspram Logičkog</i>	121
4.22.9	<i>Logički Operatori</i>	122
4.22.10	<i>Logički Izrazi i Bočni Efekti</i>	122

4.22.11	<i>Logičko naspram Bitskog</i>	122
4.22.12	<i>Pravila Uslovnog Operatora</i>	123
4.22.13	<i>Operatori Dodele</i>	123
4.22.14	<i>Obični Operator Dodele</i>	124
4.22.15	<i>Složeni Operatori Dodele</i>	124
4.22.16	<i>Sizeof Operator</i>	124
4.22.17	<i>Sizeof Primenjen na Izraz</i>	125
4.22.18	<i>Sizeof Primenjen na Tip</i>	125
4.23	<i>Izlazi</i>	125
4.23.1	<i>Izrazi sa Zarezom</i>	126
4.24	NAREDBE	126
4.24.1	<i>Naredbe sa Labelom</i>	127
4.24.2	<i>Naredbe Izraza</i>	127
4.24.3	<i>Naredbe Selekcije</i>	127
4.24.4	<i>If Naredba</i>	127
4.24.5	<i>Ugnježdene if naredbe</i>	128
4.24.6	<i>Switch Naredba</i>	128
4.24.7	<i>Naredbe Iteracije</i>	129
4.24.8	<i>While Naredba</i>	129
4.24.9	<i>Do Naredba</i>	129
4.24.10	<i>For Naredba</i>	130
4.24.11	<i>Naredbe Skoka</i>	130
4.24.12	<i>Break Naredba</i>	131
4.24.13	<i>Continue Naredba</i>	131
4.24.14	<i>Goto Naredba</i>	131
4.24.15	<i>Return Naredba</i>	132
4.24.16	<i>Složene Naredbe (Blokovi)</i>	132
4.25	Pred-procesor	132
4.25.1	<i>Pred-procesorske Direktive</i>	132
4.25.2	<i>Nastavak linije sa Kosom Crtom Unazad</i>	133
4.25.3	<i>Makroi</i>	133
4.25.4	<i>Definisanje Makroa i Makro Proširenja</i>	133
4.25.5	<i>Makroi sa Parametrima</i>	134
4.25.6	<i>Poništavanje Definicija Makroa</i>	135
4.25.7	<i>Uključivanje Fajla</i>	135
4.25.8	<i>Eksplcitna Putanja</i>	136
4.25.9	<i>Pred-procesorski Operatori</i>	136
4.25.10	<i>Operator #</i>	136
4.25.11	<i>Operator ##</i>	137
4.25.12	<i>Uslovno Kompajliranje</i>	137
4.25.13	<i>Direktive #if, #elif, #else, i #endif</i>	137
4.25.14	<i>Direktive #ifdef i #ifndef</i>	138
5	MPLAB-INTEGRISANO RAZVOJNO OKRUŽENJE ZA PIC MIKROPROCESORE BAZIRANO NA WINDOWS OPERATIVNOM SISTEMU	139

5.1	MPLAB – Razvojno okruženje (IDE)	140
5.2	MPLAB razvojni alati	141
5.2.1	<i>MPLAB Project Manager</i>	141
5.2.2	<i>MPLAB Editor</i>	141
5.2.3	<i>MPLAB-SIM Simulator</i>	141
5.2.4	<i>MPLAB-ICE Emulator</i>	141
5.2.5	<i>MPLAB-ICD Debager</i>	141
5.2.6	<i>MPASM univerzalni asembler/MPLINK relokatibilni linker/MPLIB</i> <i>menadžer biblioteka</i>	141
5.2.7	<i>MPLAB-C17/C18 C kompajleri</i>	141
5.2.8	<i>PRO MATE II i PICSTART Plus Programatori</i>	142
5.2.9	<i>PICMASTER I PICMASTER-CE Emulatori</i>	142
5.3	MPLAB pregled opcija menija	142
5.3.1	<i>Struktura opcija u sistemu menija MPLAB-a</i>	144
5.3.2	<i>Editing Project Information – Meni za editovanje projektnih informacija</i>	144
5.3.3	<i>Projektni meni</i>	146
5.3.4	<i>Meni za editovanje</i>	147
5.3.5	<i>Opcije editor</i>	147
5.3.6	<i>Meni debugiranja</i>	148
5.3.7	<i>Meni za kontrolu rada programatora:</i>	149
5.3.8	<i>Options Meni</i>	149
5.3.9	<i>Tools meni</i>	150
5.3.10	<i>Window Meni</i>	150
5.3.11	<i>Osnovna procedura razvoja programa u MPLAB razvojnom okruženju</i>	151
5.3.12	<i>Editovanje programa</i>	151
5.3.13	<i>Prevođenje programa</i>	152
5.3.14	<i>Debugiranje i emulacija rada programa</i>	155
5.3.15	<i>Prekidne tačke</i>	157
5.3.16	<i>Simulator</i>	157
5.3.17	<i>Programiranje mikroprocesora</i>	158
6	MAŠINSKI JEZIK MIKROKONTROLERA PIC16F84	163
6.1	Set instrukcija mikrokontrolera PIC16F84 (Alfabetски redosled naredbi)	165
6.2	Set instrukcija mikrokontrolera PIC16F84 (Funkcionalna tabela)	167
6.3	Mašinske instrukcije mikrokontrolera PIC16F84	170
7	ZAKLJUČAK	197
	LISTA SLIKA	
	LITERATURA	

1 UVOD

Pomoćni udžbenik **“Programiranje mikrokontrolera”** obezbeđuje neophodne informacije studentima za savladavanje gradiva iz i u skladu je sa planom i programom predmeta Mikroračunarski sistemi i Mikrokontroleri i programiranje na Osnovnim akademskim studijama, kao i na predmetu Embedded sistemi na Master studijama.

Sama problematika kojom se bavi udžbenik je veoma aktuelna, naročito u moderno doba u kome postoji izuzetno široka primena mikrokontrolera u milijardama već proizvedenih uređaja.

Ovaj udžbenik ne predstavlja teoretsko razmatranje i klasifikaciju ove naučne oblasti koja se veoma brzo razvija, već je proistekao iz praktičnog rada i iskustva koji su autori stekli držeći predavanja i računske vežbe iz istoimenog predmeta. Priručnik je namenjen pre svega savladavanju osnovnih hardverski i softverskih komponenti koje su neophodne za razvoj i kodiranje programa na mikrokontroleru *Microchip* PIC 16F84A. Posebna pažnja data je detaljnom pregledu mašinskih naredbi i praktičnih primera programiranja ovog mikrokontrolera.

U prvom poglavlju je dat hardverski prikaz mikro kontrolera PIC 16F84A. Najpre je navedeno nekoliko primera primene, opisani su portovi mikrokontrolera i način određivanja smerova signala (ulazni, izlazni), vrste oscilatora, organizacija memorije, registri mikrokontrolera i njihova namena, tajmeri, sistem i prekida, EEPROM memorija i neke tehnike programiranja. Svi delovi izlaganja su ilustrovani kraćim ili dužim primerima.

U sledećem poglavlju udžbenika dat je pregled *MicroC* programskog okruženja. Programiranje mikrokontrolera na programskom jeziku C je veoma razvijeno imajući u vidu da se ANSI kompatibilan prevodilac, koji je ovde korišćen, pokazao odličan u smislu blizine mašinskom jeziku, što je najbitnije imajući u vidu ograničene hardverske resurse kojima ovaj CPU raspolaže. U ovom delu udbenika dato je i detaljno uputstvo i procedura korićenja i ravoja programa u ovom programskom okruženju, kao i pregled komponenti i aplikacija koje se koriste (Code editor, Code explorer, Debugger, Integrirani alati). Najvažniji deo ovog poglavlja obuhvata detaljni prikaz neophodnih koraka u razvoju jedne aplikacije.

U nastavku prikazan je detaljni opis osnovne ANSI verzije programskog jezika C koji se koristi u ovom razvojnom okruženju. Dat je veliki broj kraćih programskih primera.

Sledeće poglavlje posvećeno je MPLAB-u – Integrisanom razvojnom okruženju za PIC mikroprocesore bazirano na *Windows* operativnom sistemu. Prikazan je opis osnovnih elemenata ovog popularnog razvojnog okruženja uključujući i detaljne primere razvoja aplikacije u njemu.

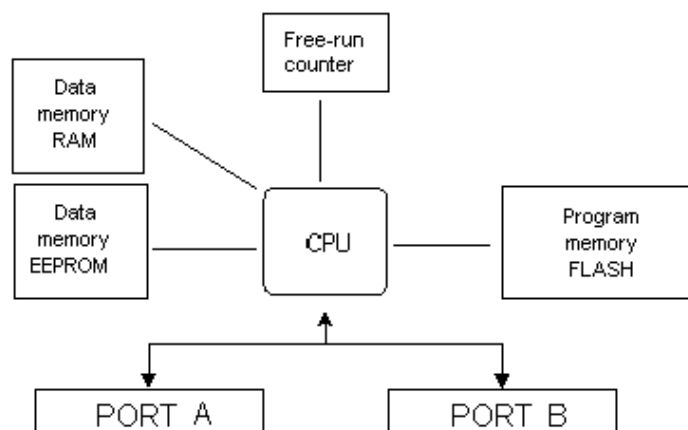
PROGRAMIRANJE MIKROKONTROLERA

U finalnom poglavlju dat je pregled mašinskog jezika mikrokontrolera PIC16F84 sa detaljnim opisom svih mašinskih naredbi koje su definisane skupom instrukcija ovog mikroprocesora.

Imajući u vidu da na našem jeziku ne postoji dovoljno adekvatne literature, pogotovo za ovaj predmet koji se nekoliko godina drži kao osnovni kurs prema stranoj literaturi, autori smatraju da je ovo izdanje konkretan doprinos nastavi i efikasnom usvajanju znanja iz ove oblasti kao i pripremama studenata za ispite ili kasniji rad u ovoj veoma dinamičnoj i interesantnoj oblasti.

2 HARDVERSKI OPIS PIC16F84

U ovom delu biće opisana hardverska struktura mikro kontrolera PIC16F84, bez namere da to bude prevod korisničkog uputstva, već da se na jedan postupan način predstave najvažnije osobine ove familije mikrokontrolera. Neki delovi, koji bi samo povećali obim ovog dela, su namerno izostavljeni, a za detaljnije upoznavanje se preporučuju originilni priručnici dati u spisku literature. PIC16F84 pripada klasi 8-bitnih mikrokontrolera sa RISC arhitekturom. Njegova struktura je prikazana na sledećoj slici.



PIC16F84 outline

Slika 2.1

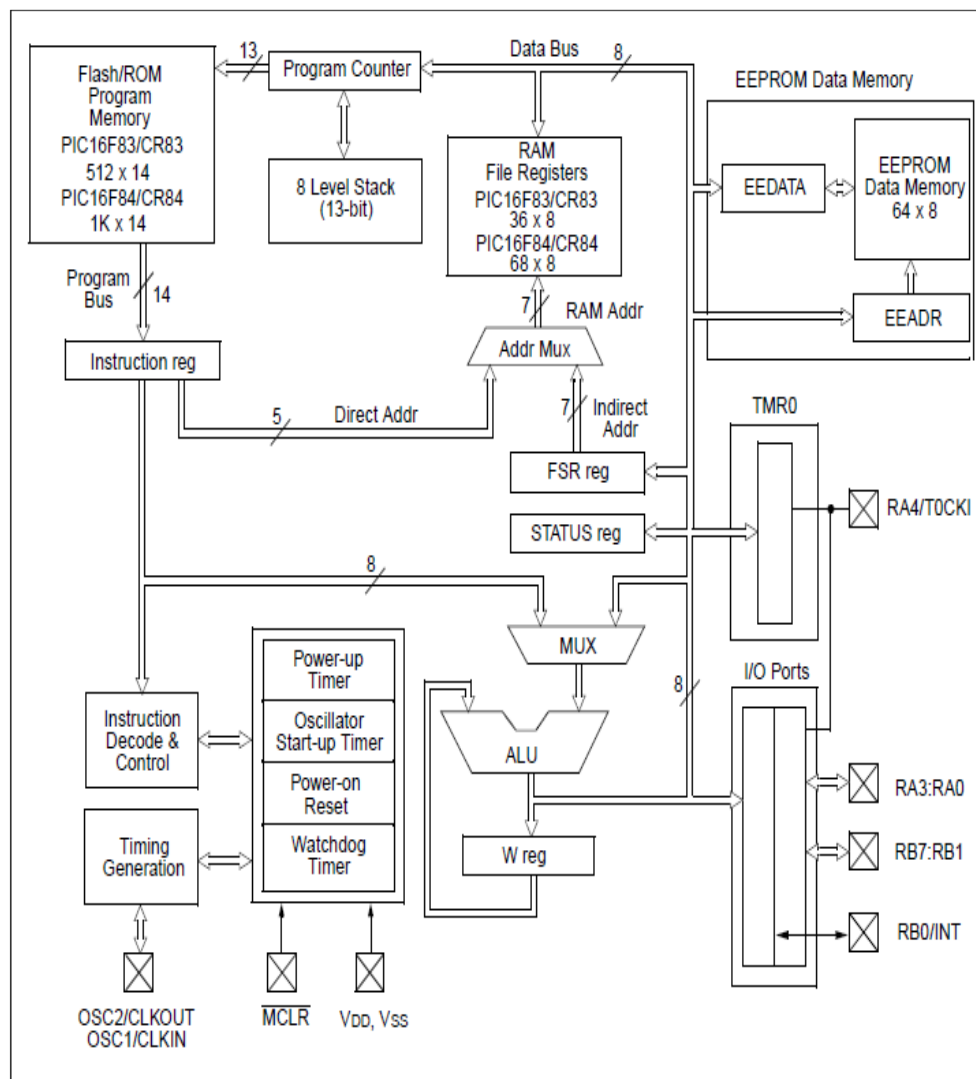
U programskoj memoriji se pamti upisani program. Ta memorija je u FLASH (fleš) tehnologiji tako da ju je moguće programirati više puta, što je pogodno u postupku razvoja uređaja. EEPROM memorija za podatke čuva svoj sadržaj i u odsustvu napajanja. Koristi se za pamćenje važnih podataka koji ne smeju da budu izgubljeni prilikom iznenadnog nestanka napajanja. (Na primer broj pređenih kilometara u vozilu, neka podešavanja sa kojima želimo da se nastavi kad ponovo dođe napajanje.

RAM memorija se koristi za vreme izvršenja programa. U RAM-u su zapamćeni svi međurezultati i privremeni podaci. PORTA i PORTB predstavljaju fizičku vezu između mikrokontrolera i spoljašnjeg sveta. Port A ima 5, a port B 8 priključaka.

PROGRAMIRANJE MIKROKONTROLERA

Unutar mikrokontrolera je i jedan 8-bitni tajmer, TMR0, koji radi nezavisno od programa. Svaki četvrti takti impuls povećava sadržaj tog brojača sve do 255 kada ponovo počinje od nule. Znajući učestanost oscilatora, moguće je, korišćenjem ovog brojača, meriti vremenske intervale. CPU kordinira rad svih blokova i izvršava korisnički program.

Arhitektura mikrokontrolera PIC16F84A prikazana je na sledećoj slici.



Slika 2.2

2.1 Primene

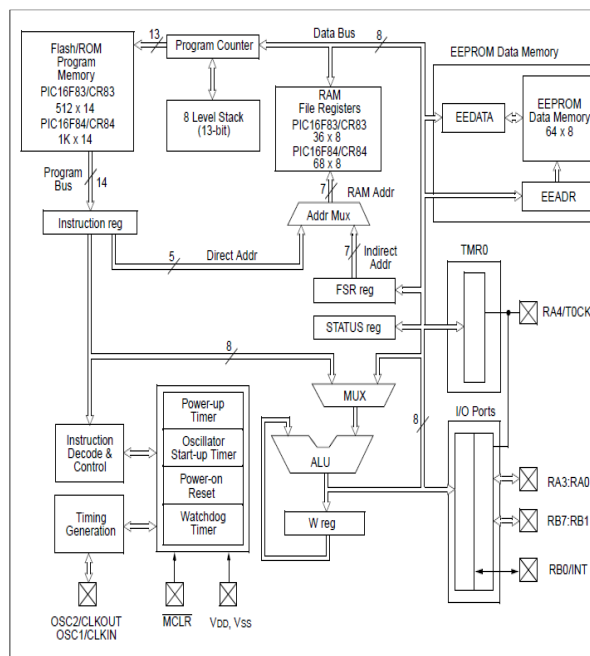
PIC16F84 je pogodan za mnoge primene, kao što su automobilska industrija, upravljanje kućnim aparatima i industrijskim instrumentima, udaljeni senzori, automatska vrata, alarmni sistemi itd. Pogodan je i za smart kartice, kao i zbog male potrošnje i za uređaje sa baterijskim napajanjem.

Na sledećim slikama prikazane su neke primene mikrokontrolera. Veoma često se umesto mikroprocesora u embeded sistimima koriste mikrokontroleri. Široku primenu pokazuju podaci da u jednom domaćinstvu ima od 30 do 49 takvih sistema, a u automobilu čak do 50. U automobilu je tu recimo upravljanje automatskim menjačem, ubrizgavanjem goriva, ABS, ESP, klima uređaj itd., a u domaćinstvu razni muzički i video uređaji, mašina za pranje veša ili sudova, grejne ploče, rerna, uređaji za grejanje ili hlađenje stana itd.



Slika 2.3 – a) Automatski menjač, b) Veš mašina, c) Digitalni termostat d) MP3 plejer

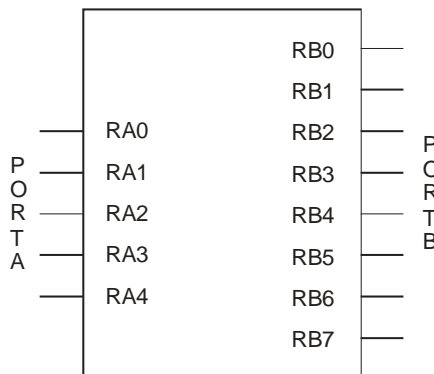
Upis programa u mikrokontroler se vrši samo preko dva priključka i moguće ga je vršiti i na ciljnoj ploči/uređaju, što daje veliku fleksibilnost proizvodima tako da se mogu menjati i nakon što su sastavljeni i testirani. Dalje, to daje mogućnost kalibracije nakon završnog testiranja ili čak promenu programa radi unapređenja nekih funkcija na gotovom uređaju.



Slika 2.4

2.2 Portovi mikrokontrolera

Veza mikrokontrolera sa spoljašnjom sredinom ostvarena je preko portova čija su imena PORTA i PORTB (za 16F84) i PORTC, PORTD, PORTE (za mikrokontrolere sa više portova). Oznake pinova su RA0, RA1, ..., RA4 za PORTA i RB0, RB1, ..., RB7 za PORTB (Slika 2.5).



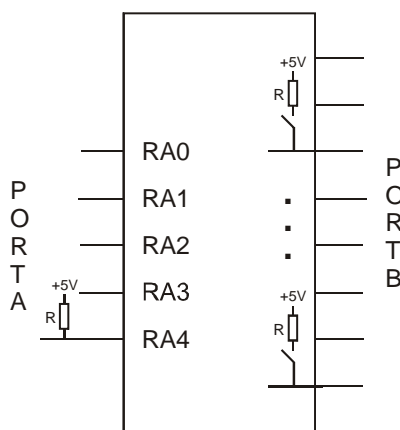
Slika 2.5 – Portovi mikrokontrolera PIC16F84

Svaki od pinova na portovima može biti ulazni (prima spoljašnji signal) ili izlazni (daje signal). Smerovi su određeni bitovima u TRIS registrima (TRISA i TRISB) na sledeći način:

- 0 (nula) pin je izlazni
- 1 (jedan) pin je ulazni

Radi lakšeg pamćenja možemo reći da 0 podesća na Output (izlaz), a 1 na Input (ulaz)

Primer 1. Ako je TRISA = XXX01001 (najviša 3 bita nemaju uticaja) i TRISB = 00111000 tada su pinovi RA4, RA2, RA1, RB7, RB6, RB2, RB1 i RB0 izlazni, a pinovi RA3, RA0, RB5, RB4 i RB3 su ulazni.

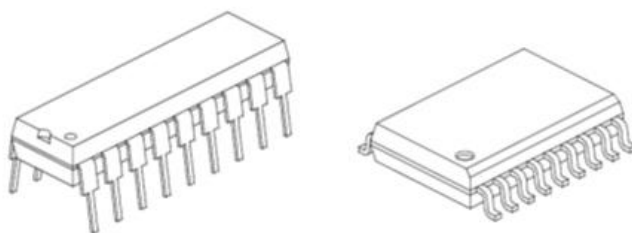


Slika 2.6

Pojedini priključci na portovima mogu imati i neke druge osobine, tako de je kod projektovanja potrebno voditi računa o nameni pinova kako bi se što bolje iskoristile te dodatne osobine. Evo nekih od tih specifičnosti. Svi pinovi na PORTB mogu imati takozvane „pull up“ otpornike, tj. otpornike vezane na napon napajanja unutar mikrokontrolera. Svi otpornici se uključuju ili isključuju brisanjem ili postavljanjem bit 7 u registru OPTION. Potrebno je obratiti pažnju i na pin RA4 koji je „open collector“ tipa, što znači da se na ovaj pin spolja mora staviti otpornik vezan na napon napajanja. Ovaj pin ima još jednu osobinu a to je da se može koristiti za taktovanje internog brojača TMR0. Pinovi RB0 i RB4 – RB7 na PORTB se mogu koristiti i kao linije za različite vrste prekida o čemu će biti reči kasnije u delu posvećenom prekidima. U zavisnosti od izabranog kontrolera pinovi mogu imati i druge dodatne funkcije, kao što su analogni ulazi, zatim pinovi za serijski ulaz/izlaz, izlaz za PWM (impulsno širinska modulacija) itd. Zato je veoma važno detaljno proučiti korisničko uputstvo za izabrani tip kontrolera, kako bi se optimalno iskoristile mogućnosti koje data familija pruža.

2.3 Opis pinova

PIC16F84 ima ukupno 18 pinova i najčešće se nalazi u DIP kućištu, ali se može naći i SMD kućištu.



Slika 2.7

Opis pinova mikrokontrolera PIC16F84

Pin Br.	Ime	Opis
Pin Br. 1	RA2	Drugi pin na portu A. Nema dodatnih funkcija
Pin Br. 2	RA3	Treći pin na portu A. Nema dodatnih funkcija
Pin Br. 3	RA4	Četvrti pin na portu A. TOCK1 se može koristiti kao takt (clock) za tajmer0.
Pin Br. 4	MCLR	Reset ulaz i Vpp napon programiranja.
Pin Br. 5	Vss	Masa napajanja.
Pin Br. 6	RB0	Nulti pin na portu B. Ulaz za prekid kao dodatna funkcija.
Pin Br. 7	RB1	Prvi pin na portu B. Nema dodatnih funkcija.
Pin Br. 8	RB2	Drugi pin na portu B. Nema dodatnih funkcija.
Pin Br. 9	RB3	Treći pin na portu B. Nema dodatnih funkcija.
Pin Br. 10	RB4	Četvrti pin na portu B. Ulaz za prekid na promenu.

Pin Br. 11	RB5	Peti pin na portu B. Ulaz za prekid na promenu.
Pin Br. 12	RB6	Šesti pin na portu B. Ulaz za prekid na promenu. 'Clock' linija za vreme programiranja.
Pin Br. 13	RB7	Sedmi pin na portu B. Ulaz za prekid na promenu. 'Data' linija za vreme programiranja.
Pin Br. 14	Vdd	Pozitivni napon napajanja.
Pin Br. 15	OSC2	Pin namenjen za priključenje oscilatora.
Pin Br. 16	OSC1	Pin namenjen za priključenje oscilatora.
Pin Br. 17	RA0	Nulti pin na portu A. Nema dodatnih funkcija
Pin Br. 18	RA1	Prvi pin na portu A. Nema dodatnih funkcija

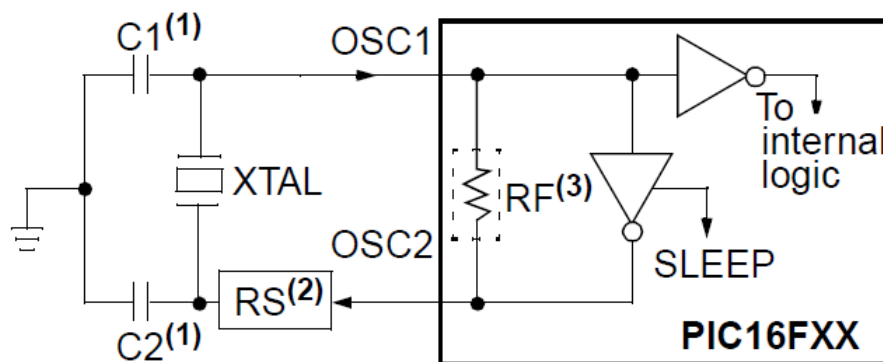
2.4 PIC reset i oscillator

2.4.1 Tipovi oscilatora

PIC16F84 može da radi sa različitim tipovima oscilatora. Najčešće korišćeni tipovi oscilatora su kristalni oscilator i oscilator sa otpornikom i kondenzatorom. Kristalni oscilator označavamo sa XT, HS ili LP, a sa otpornikom i kondenzatorom sa RC.

2.4.2 XT oscilator

Kristalni oscilator se nalazi u metalnom kućištu sa dva pina. Na kućištu je navedena učestanost oscilovanja. Potrebno je oba pina povezati kondenzatorima od 30pF na masu. Ponekad se oscilator i kondenzatori pakuju u jedno kućište sa 3 pina. Takva komponenta se naziva keramički rezonator. Središnji pin se tada vezuje na masu, dok se krajnji pinovi vezuju na OSC1 i OSC2 pinove mikrokontrolera. kod dizajniranja uređaja, pravilo je da se oscilator stavlja što bliže kontroleru da bi se izbegle smetnje koje se mogu javiti na linijama preko kojih mikrokontroler prima takt (klok). Šema XT oscilatora je prikazana na sledećoj slici.



Slika 2.8 – Eksterni RC oscilator

2.4.4 Vrste reseta

Mikrokontroler PIC16F8X ima više tipova reseta:

- Reset prilikom uključanja napajanja (Power-on Reset -POR)
- MCLR reset za vreme normalnog izvršenja operacija
- MCLR reset za vreme SLEEP moda
- WDT reset za vreme normalnog izvršenja operacija
- WDT buđenje za vreme SLEEP moda

Najvažnija su prva dva. Prvi se dešava kada god se dovede napajanje na mikrokontroler i služi za dovođenje svih registara u početno stanje. Drugi reset se dešava dovođenjem logičke nule na MCLR pin za vreme normalnog izvršenja operacija. Često se koristi tokom razvoja programa.

Prilikom reseta memorijske lokacije u RAM memorije se ne resetuju. One su neodređenog sadržaja nakon reseta prilikom uključanja napajanja, a ne menjaju se nakon reseta na MCLR pinu za vreme normalnog izvršenja operacija. Za razliku od njih SFR registri se dovode u početna stanja koja su definisana u Tabeli xx.

Tabela xx

adresa	naziv	vrednost nakon POR reseta	vrednost nakon ostalih reseta
banka 0			
00h	INDF	----	----
01h	TMR0	xxxx xxxx	uuuu uuuu
02h	PCL	0000 0000	0000 0000
03h	STATUS	0001 1xxx	000q quuu
04h	FSR	xxxx xxxx	uuuu uuuu
05h	PORTA	---x xxxx	---u uuuu
06h	PORTB	xxxx xxxx	uuuu uuuu
07h	nepostojeca lokacija, cita se kao "0"		
08h	EEDATA	xxxx xxxx	uuuu uuuu
09h	EEADR	xxxx xxxx	uuuu uuuu
0Ah	PCLATH	---0 0000	---0 0000
0Bh	INTCON	0000 000x	0000 000u
banka 1			
80h	INDF	----	----
81h	OPTION_REG	1111 1111	1111 1111

PROGRAMIRANJE MIKROKONTROLERA

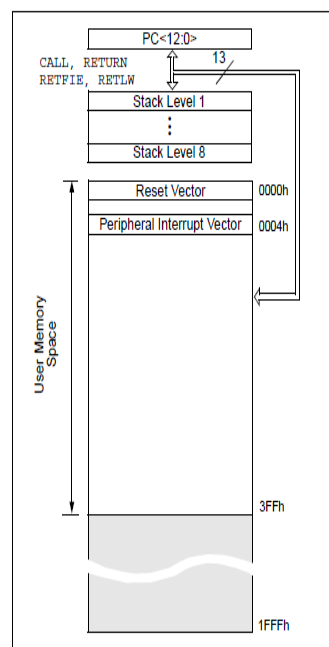
82h	PCL	0000 0000	0000 0000
83h	STATUS	0001 1xxx	000q quuu
84h	FSR	xxxx xxxx	uuuu uuuu
85h	TRISA	---1 1111	---1 1111
86h	TRISB	1111 1111	1111 1111
87h	nepostojeca lokacija, cita se kao "0"		
88h	EECON1	---0 x000	---0 q000
89h	EECON2	---- ----	---- ----
0Ah	PCLATH	---0 0000	---0 0000
0Bh	INTCON	0000 000x	0000 000u

legenda: x=nepoznato, u=nepromenjeno, q=zavisi od uslova

2.5 Organizacija memorije

PIC mikrokontroleri imaju „Harvard arhitekturu“, što znači da imaju fizički odvojene memorije i puteve signala za naredbe i podatke. To je omogućilo i različite dužine reči u ovim memorijama kao i različite širine puteva podataka i naredbi.

2.5.1 Organizacija programske memorije



Slika 2.9

2.6 Registri mikrokontrolera

Registri mikrokontrolera PIC16F84 navedeni su u sledećoj tabeli

00	INDF	INDF	80
01	TMR0	OPTION_REG	81
02	PCL	PCL	82
03	STATUS	STATUS	83
04	FSR	FSR	84
05	PORTA	TRISA	85
06	PORTB	TRISB	86
07			87
08	EEDATA	EECON1	88
09	EEADR	EECON2	89
0A	PCLATH	PDLATH	8A
0B	INTCON	INTCON	8B
0C			8C

Registri se nalaze u dve banke BANKA0 i BANKA1. Da bi se pristupilo nekomod registara potrebno je da ta banka bude selektovana. Namena svakog od registara će biti opisana kasnije, a sada su navedene samo njihove funkcije.

- INDF Nije fizički implementiran (ne postoji), koristi se kod indirektnog adresiranja
- TMR0 Tajmer0 8-bitni brojački registar
- PCL Niži deo brojača naredbi (program counter low)
- STATUS Registar stanja
- FSR Koristi se kod indirektnog adresiranja
- PORTA Ulazno izlazni port
- PORTB Ulazno izlazni port
- EEDATA Podatak koji se upisuje ili čita iz EEPROM-a
- EEADR Adresa podatka u EEPROM-u
- PCLATH Viši deo brojača naredbi (program counter high)
- INTCON Registar za upravljanje prekidima
- OPTION_REG Registar za izbor nekih dodatnih opcija
- TRISA Registar za određivanje smera signala na (ulazni ili izlazni)
- TRISB Registar za određivanje smera signala (ulazni ili izlazni)
- EECON1 Upravljački registar za upis u EEPROM
- EECON2 Upravljački registar za upis u EEPROM

2.6.1 Namena registara

Namena registara nije opisana po redosledu adresa tih registara u memoriji, već na način koji će obezbediti brže lakše upoznavanje sa mogućnostima mikrokontrolera PIC16F84.

2.6.1.1 STATUS registar

STATUS registar sadrži stanja aritmetičke jedinice, stanje nakon reseta i bitove za selekciju banke. Kao i u svaki drugi registar STATUS može biti određeno za bilo koju naredbu. Ako je STATUS registar određeno za naredbe koje utiču na Z, DC ili C bitove, tada je upisu te bitove onemogućen. Njihove vrednosti određuje logika aritmetičke jedinice. Nije dozvoljen upis u TO i PD bitove. Zato se rezultat u STATUS može razlikovati od očekivanog. Na primer, naredba CLRF STATUS će obrisati najviše 3 bita i postaviti Z bit. Nakon te naredbe STATUS registar će imati vrednost 000u uluu, (gde je u = nepromenjeno).

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	\overline{TO}	\overline{PD}	Z	DC	C
bit7							bit0

Slika 2.10

Legenda:

R = moguće čitanje (Readable)

W = moguć upis (Writable)

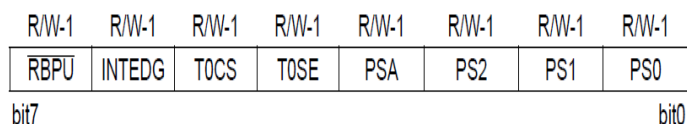
U = neimplementiran (Unimplemented) čita se kao '0'

-n = vrednost nakon POR reseta (nakon uključanja)

bit 7	Ne koristi se
bit 6-5	bitovi za selekciju banke kod direktnog adresiranja RP0 = 0 BANKA0 (00h - 7Fh) RP0 = 1 BANKA1 (80h - FFh) bit 6 (RP1) se ne koristi kod 16F8X
bit 4	TO: (time out) 1=nakon dovođenja napajanja, CLRWDT ili SLEEP naredbe 0=dogodio se time out reset od WDT (watch dog timer)
bit 3	PD: (power down) 1=nakon dovođenja napajanja ili CLRWDT naredbe 0=nakon izvršenja SLEEP naredbe
bit 2	Z:(zero) indikator (flag) za '0' 1=rezultat aritmetičke ili logičke operacije je 0 0= rezultat aritmetičke ili logičke operacije nije 0
bit 1	DC: Digital carry (decimalni prenos ili pozajmica kod BCD operacija) 1=biloje prenosa iz niže u višu tetradu 0= nije biloje prenosa iz niže u višu tetradu
bit 0	C: (carry/borrow) prenos ili pozajmica iz bit najviše pozicije 1=dogodio se prenos iz bita najviše pozicije 0=nije bilo prenosa iz bita najviše pozicije Napomena: kod oduzimanja je vrednost pozajmice komplemnirana, tj bit je 1 kad nije bilo pozajmice

2.6.1.2 OPTION_REG registar

OPTION_REG je registar se može čitati i u koji se može upisivati. Sadrži različite upravljačke bitove kojima se se može konfigurirati preskaler, spoljašnji prekid (INT), tajmer TMR0, kao i otpornici na pinovima PORTB.



Slika 2.11

Legenda:

R = moguće čitanje (Readable)

W = moguć upis (Writable)

U = neimplementiran (Unimplemented) čita se kao '0'

-n = vrednost nakon POR reset (nakon uključanja)

bit 7	Not RBP_U : Otpornici vezani na Vcc uključeni/isključeni 0=otpornici na PORTB uključeni 1= otpornici na PORTB isključeni		
bit 6	INTEDG : ivica za spoljašnji prekid 1= prekid na rastućoj ivici signala na RB0/INT priključku 0= prekid na opadajućoj ivici signala na RB0/INT priključku		
bit 5	T0CS : izbor takta za TMR0 1= sa RA4/T0CKI priključka 0= interni takt kojim se izvršavaju naredbe (CLKOUT)		
bit 4	T0SE : izbor ivice takta za TMR0 1 = Uvećava se na prelazu sa visoko na nisko na RA4/T0CKI priključku 0 = Uvećava se na prelazu sa nisko na visoko na RA4/T0CKI priključku		
bit 3	PSA : dodela preskalera 1= preskaler je dodeljen WDT tajmeru 0= preskaler je dodeljen TMR0		
bit 2-0	PS2:PS0: bitovi za izbor odnosa deljenja preskalera		
	PS2:PS0	TMR0 učestanost	WDT učestanost
	000	1 : 2	1 : 1
	001	1 : 4	1 : 2
	010	1 : 8	1 : 4
	011	1 : 16	1 : 8
	100	1 : 32	1 : 16
	101	1 : 64	1 : 32
	110	1 : 128	1 : 64

PROGRAMIRANJE MIKROKONTROLERA

	111	1 : 256	1 : 128
--	-----	---------	---------

2.6.1.3 INTCON registar

INTCON je registar se može čitati i u koji se može upisivati. Sadrži bitove dozvole za različite vrste prekida kao i njihove indikatore.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
bit7							bit0

Slika 2.12

Legenda:

R = moguće čitanje (Readable)

W = moguć upis (Writable)

U = neimplementiran (Unimplemented) čita se kao '0'

-n = vrednost nakon POR resetu (nakon uključnja)

bit 7	GIE: bit globalne dozvole prekida 1=dozvoljeni su svi prekidi 0=zabranjeni su svi prekidi
bit 6	EEIE dozvola prekida na završetku upisa u data EEPROM 1=dozvoljen prekid na završetku upisa u data EEPROM 0=zabranjen prekid na završetku upisa u data EEPROM
bit 5	TOIE: dozvola prekida od tajmera (TMR0) 1=dozvoljen prekid od TMR0 0=zabranjen prekid od TMR0
bit 4	INTE: dozvola prekida na RB0/INT priključku 1=dozvoljen prekid na RB0/INT priključku 0=zabranjen prekid na RB0/INT priključku
bit 3	RBIE: dozvola prekida na promenu na priključcima RB4-RB7 1=dozvoljen prekid na promenu na priključcima RB4-RB7 0=zabranjen prekid na promenu na priključcima RB4-RB7
bit 2	TOIF: indikator prekida od TMR0 1=dogodio se prelaz TMR0 a 255 na 0 0=nije se dogodio prelaz TMR0 a 255 na 0
bit 1	INTF: indikator prekida na RB0/INT priključku 1=dogodio se prekid na RB0/INT priključku 0=nije se dogodio prekid na RB0/INT priključku
bit 0	RBIF: indikator prekida na promenu na priključcima RB4-RB7 1=dogodio se prekid na promenu na priključcima RB4-RB7 0=nije se dogodio prekid na promenu na priključcima RB4-RB7

Napomena: Indikatori prekida se postavljaju kada se steknu uslovi za prekid, bez obzira da li je taj prekid dozvoljen ili da li je bit globalne dozvole prekida postavljen.

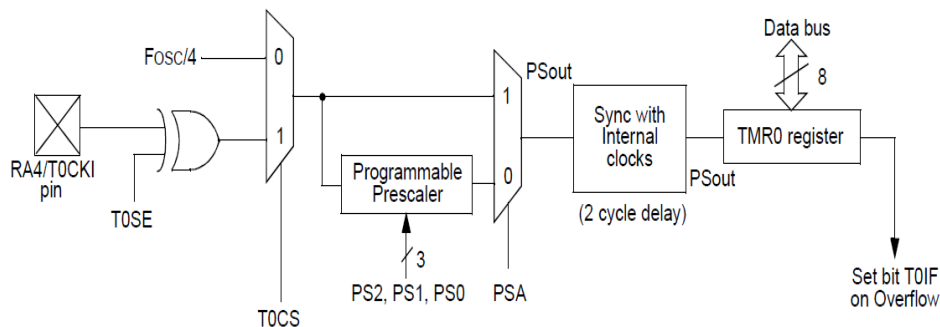
2.7 Tajmer0 modul I TMR0 registar

- Modul tajmer0 ima sledeće karakteristike:
- 8-bitni tajmer-brojač
- moguće je čitanje i upis
- 8-bitni programabilni preskaler
- prekid prilok prelaska sa FFh na 00h
- izbor ivice za spoljašnji takt

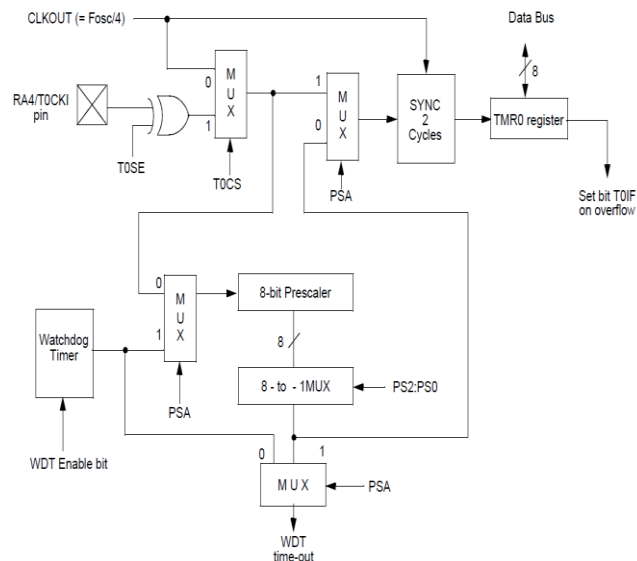
Kada je **T0CS=0** (**OPTION_REG<5>**), ovaj registar radi kao tajmer. TMR0 se uvećava za 1 u svakom ciklusu naredbe (bez preskalera). Nakon upisa u **TMR0**, uvećavanje je sprečeno u naredna dva ciklusa. Programer mora da void računa o ovome i da podesi željenu vrednost kod upisa.

Kada je **T0CS=1** (**OPTION_REG<5>**), ovaj registar radi kao brojač. U ovom modu **TMR0** se uvećava na rastućoj ili opadajućoj ivici signala na priključku **RA4/T0CKI**. Ivica na kojoj se vrši povećanje je određena vrednošću bita **T0SE** (**OPTION_REG<4>**). Ako je vrednost tog bita 0 izabrana je rastuća ivica.

Preskaler je deljiv između **TMR0** i **WDT**. Dodela se vrši bitom **PSA** (**OPTION_REG<3>**). Ako je ovaj bit 0 preskaler je dodeljen tajmeru0. Odnosi deljenja učestanosti dati su u opisu registra **OPTION_REG**. Šema preskalera je i celog tajmerskog- brojačkog su prikazane na sledećim slikama_



Slika 2.13 - Preskaler



Slika 2.14 - Tajmersko – brojački modul

Primer: Neka je radna učestanost mikrokontrolera 3.2768 MHz. Ako je potrebno je obezbediti prelasku **TMR0** sa FFh na 00h, vrednost koja se upisuje u **OPTION_REG** se određuje na sledeći način:

$$\frac{2^{15} \times 100 \text{ Hz}}{2^2 \times 2^8 \times 2^{n+1}} = 400 \text{ Hz}, \Rightarrow n = 2$$

OPTION_REG = 1XX0 0010 = 1000 0010; (nisu uljučeni otpornici na **PORTB**)

MOVLW 0x82

MOVWF OPTION_REG ; kraj primera

2.8 Načini adresiranja podataka

PIC mikrokontroleri imaju tri načina adresiranja podataka:

- neposredno (immediately)
- direktno, i
- indirektno

Kod neposrednog adresiranja podatak je sastavni deo naredbe. Ove naredbe u svom mnemoniku imaju "LW", (Na primer; MOVLW, ADDLW, ANDLW, ...).

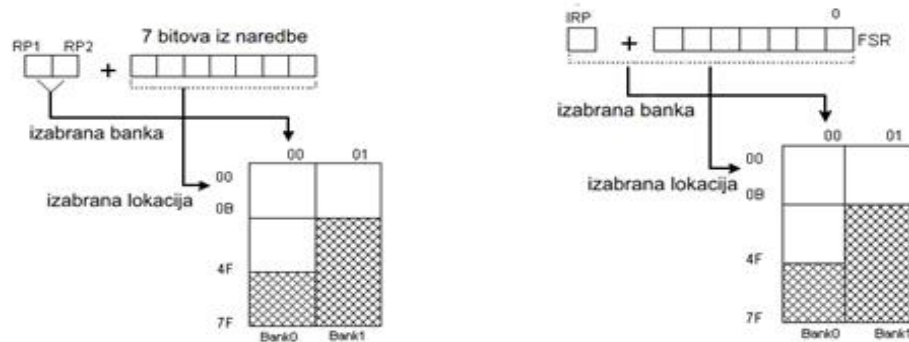
Direktno adresiranje se vrši 9-bitnom adresom, koja se sastoji od 7 bitova adrese dobijenih iz naredbe i dva bita (RP1, RP0) iz STATUS registra.

Primer: Postaviti viša 4 priključka na PORTB kao izlazne, i niža 4 priključka kao ulazne

BSF STATUS, RP0 ; Selkcija Banke 1

MOVLW 0x0F ; w=0x0F, neposredno adresiranje

MOVWF TRISB ; adresa registra TRISB je uzeta iz naredbe



Slika 2.15 – a) Direktno adresiranje, b) Indirektno adresiranje

Kod indirektnog adresiranja adresa operanda se ne dobija iz naredbe već iz IRP bita u STATUS i FSR registra. U takvim naredbama se u adresnom delu naredbe nalazi adresa registra INDF.

Primer: Neka se podatak FFh nalazi u promenljivoj data na adresi 22h.

Da bi indirektno pristupili ovom podatku potrebno je najpre njegovu adresu upisati u FSR:

```
data: .EQU 22;
```

```
...
```

```
MOVLW data; ili MOVLW 0x22
```

```
MOVF INDF,W; indirektno adresiranje
```

```
; Posledica je da radni registar W uzima vrenost FFh.
```

Primer: Neka se 8 podataka nalazi počev od adrese 0x20. Odrediti srednju vrednost tih podataka

```
CBLOCK 0x20
```

```
pod0
```

```
pod1
```

```
pod2
```

```
pod3
```

```
pod4
```

```
pod5
```

```
pod6
```

```
pod7
```

```
sr0
```

```
sr1
```

PROGRAMIRANJE MIKROKONTROLERA

```
ENDC

;-----
-

;This code executes when a reset occurs.
ORG 0x0000
GOTO ResetCode
ORG 0x0004 ;place code at interrupt vector
goto InterruptCode
ResetCode:
    CLRF PCLATH ;select program memory page 0
    GOTO Main ;go to beginning of program
Main:
    MOVLW pod7
    MOVWF FSR ;adresa pod7 je u FSR
    CLRF sr0
    CLRF srl
saber_i:
    MOVF sr0,W
    ADDWF INDF,W ;dodavanje brojeva pocv od lokacije pod7
                    ; do pod0

    BTFSC STATUS,C
    INCF srl,F
    MOVWF sr0
    DECF FSR,F
    MOVF FSR,W
    XORLW 0x1F
    BTFSS STATUS,Z ; provera da li su sabrani svi brojevi
    GOTO saber_i
; zbir svih osam brojeva je u dve promenljive srl i sr0
; srednja vrednost se dobija pomeranjem za 3 mesta udesno
srl&sr0
```

```
RRF sr1,F
RRF sr0,F
RRF sr1,F
RRF sr0,F
RRF sr1,F
RRF sr0,F;srednja vrednost je u SR0
opet:
GOTO opet
```

2.9 EEPROM Memorija

PIC16F84 ima EEPROM 64 lokacije za pamćenje podataka (od adrese 00h do 3Fh). U ovoj memoriji podaci ostaju zapamćeni i u odsustvu napajanja. Na primer, ako kontroler broji potrošnju vode, ili pređene kilometre, to će ostati zapamćeno i ako se prekine napajanje. Dalji rad će se normalno nastaviti dovođenjem napajanja. Korišćenjem EEPROM-a je moguće i vršiti i razna dekodiranja (recimo BCD na 7-segmenata) i preslikavanja promenljivih radi realizacije nekih funkcija.

Promenljivama u ovoj memoriji se pristupa na drugačiji način nego promenljivama koje snalaze u registarskom polju. Naime koristi se indirektno adresiranje Registrima specijalnih funkcija (SFR). Ti registri su:

- EECON1
- EECON2
- EEDATA, sadrži podatak
- EEADR, sadrži adresu podatka

Primer: Neka se na adresi "KONFIGURACIJA" nalazi podatak koji predstavlja neka inicijalna podešavanja (koja su ostala zapamćena u prethodnom radu). Deo koda koji to radi je:

```
BCF STATUS, RP0 ; Bank 0
MOVLW KONFIGURACIJA
MOVWF EEADR ; Adresa sa koje se čita
BSF STATUS, RP0 ; Bank 1
BSF EECON1, RD ; čitanje EEPROMA
BCF STATUS, RP0 ; Bank 0
MOVF EEDATA, W ; W = EEDATA, inicijalna
konfiguracija je
: u radnom registru W
```

Primer: Preslikavanje BCD koda cifre u njen izgled na 7-segmentnom displeju koji je priključen na PORTB (segment a na RB6, b na RB5, itd segment g na RB0) korišćenjem data EEPROM-a.

Binary Inputs					Decoder Outputs							7 Segment Display Outputs
D	C	B	A		a	b	c	d	e	f	g	
0	0	0	0		1	1	1	1	1	1	0	0
0	0	0	1		0	1	1	0	0	0	0	1
0	0	1	0		1	1	0	1	1	0	1	2
0	0	1	1		1	1	1	1	0	0	1	3
0	1	0	0		0	1	1	0	0	1	1	4
0	1	0	1		1	0	1	1	0	1	1	5
0	1	1	0		1	0	1	1	1	1	1	6
0	1	1	1		1	1	1	0	0	0	0	7
1	0	0	0		1	1	1	1	1	1	1	8
1	0	0	1		1	1	1	1	0	1	1	9

Slika 2.16 – Preslikavanje BCD u 7-segmenata

DATA Dump																
0x0A		FF		Data Memory Size: 64 Bytes												
0x00		7E	30	6D	79	33	5B	5F	70	7F	7B	FF	FF	FF	FF	FF
0x10		FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x20		FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x30		FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Slika 2.17 – Prikaz podataka u data EEPROM

Deo koda koji to radi je:

```
BCF STATUS, RP0 ; Bank 0
MOVF BCDCIFRA,W; NEKA BCD cifra
MOVWF EEADR ; Adresa sa koje se čita
BSF STATUS, RP0 ; Bank 1
BSF EECON1, RD ; čitanje EEPROMA
BCF STATUS, RP0 ; Bank 0
MOVF EEDATA, W ; W = EEDATA, raspored 7
                segmenata
MOVWF PORTB; prikaz na PORTB
; kraj primera
```

Preslikavanje BCD koda u izgled na 7-segmentnom displeju semože uraditi i korišćenjem programske memorije, kao što je to navedeno u sledećem primeru.

Primer: Preslikavanje BCD koda cifre u njen izgled na 7-segmentnom displeju koji je priključen na PORTB (segment a na RB6, b na RB5, itd segment g na RB0) korišćenjem programske memorije.

```
MOVF BCDCIFRA,W; NEKA BCD cifra
CALL DEKODIRANJE
                MOVWF PORTB; prikaz na PORTB
...
DEKODIRANJE:
```

```
ADDWFPCL,F;dodavanje BCDCIFRA na PCL
RETLW0x7e ;izgled cifre „0“
RETLW0x30 ;izgled cifre „1“
RETLW0x6D ;izgled cifre „2“
RETLW0x79 ;izgled cifre „3“
RETLW0x33 ;izgled cifre „4“
RETLW0x5B ;izgled cifre „5“
RETLW0x5F ;izgled cifre „6“
RETLW0x70 ;izgled cifre „7“
RETLW0x7F ;izgled cifre „8“
RETLW0x7B ;izgled cifre „9“
```

Upis u EEPROM je složeniji i zahteva se specijalna sekvenca koja je data u sledećem primeru.

Primer: Tekuća konfiguracija se čita sa PORTB i upisuje u EEPROM na adresu "KONFIGURACIJA"

```
BCF  INTCON, GIE ; ZABRANA SVIH PREKIDA
MOVF PORTB,W ;čitanje sadržaj PORTB
MOVWF EEDATA
MOVWL "KONFIGURACIJA"
MOVWF EEADR;    upis    u    eeprom    na    adresu
                  "KONFIGURACIJA"

BSF  STATUS, RP0
BSF  EECON1,WREN ;dozvola upisa
MOVLW $55
MOVWF EECON2 ;tako
MOVLW $AA ;treba
MOVWF EECON2
BSF  EECON1,WR;setovati fleg ``izvrši upis``

PROVER:
BTFSS EECON1,4 ;provera upisa
GOTO  PROVER
BCF  EECON1,WREN ;blokada upisa
BCF  EECON1,4 ;brisi bit 4 u EECON1
BCF  STATUS, RP0
; kraj primera
```

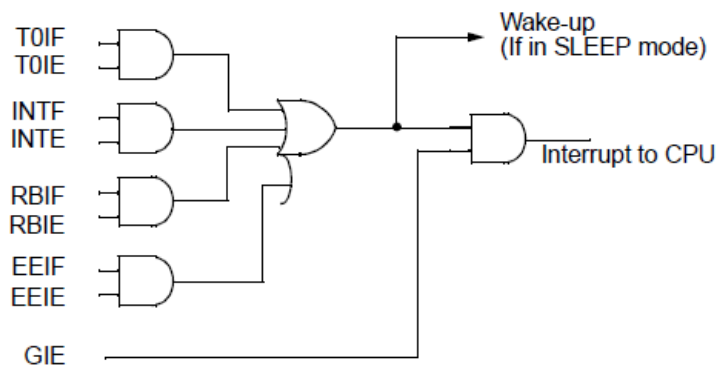
NAPOMENA: Preporučuje se da bit WREN bude isključen sve vreme izvršenja programa, osim kod upisa EEPROM. Na taj način je mogućnost neželjenog upisa EEPROM mala. Svi upisi u EEPROM će automatski obrisati sadržaje lokacija pre upisa novih sadržaja.

2.10 Prekidi

PIC16F84A ima 4 izvora prekida:

- spoljašnji prekid na RB0/INT priključku
- prekid od prekoračenja TMR0
- prekid na promenu na PORTB (priključci RB7:RB4)
- prekid kad se završi upis u data EEPROM

Prekidna logika je prikazana na sledećoj slici



Slika 2.18 – Prekidna logika

Registar INTCON sadrži globalnu dozvolu prekidu (GIE) i pojedinačne dozvole (EEIE, TOIE, INTE, RBIE), kao i indikatore za pojedine prekide (TOIF, INTF, RBIF), dok se indikator prekida od EEPROM (EEIF) nalazi u EECON1.

Kada se dogodi neki prekid GIE se briše (postaje 0), da bi se sprečili dalji prekidi, PC se pamti na steku (magacin) i PC se puni adresom 0004h. Na toj adresi se nalazi naredba kojom se prelazi na izvršenje prekidnog programa (`GOTO prekid`). U prekidnoj rutini se najpre vrši pamćenje neophodnih registara `W`, `STATUS` (save routine), zatim se vrši ispitivanje izvora prekida, usluga datog prekida i na kraju vraćanje vrednosti registara `W`, `STATUS` (resave routine).

Primer: deo programa u kome se koriste prekidi. U okviru ovog primera se registri `STATUS` i `W` pamte u `WREG_TEMP` i `STATUS_TEMP`

```
ORG 0x0000
    GOTO ResetCode
ORG 0x0004 ;mesto za prekidni program
goto Prekidni_program
```

ResetCode:

CLRF PCLATH ; izbor BANK0

GOTO Main; prelazak na glavni program

Main:

MOVLW 0xf8

MOVWF INTCON ; dozvoljeni svi prekidi

;-----

; ovaj deo se izvršava kad se dogodi prekid

Prekidni_program:

movwf WREG_TEMP ; save WREG

movwf WREG_TEMP ; save WREG

swapf STATUS, W ; store STATUS in WREG

clrf STATUS ; select file register bank0

movwf STATUS_TEMP ; save STATUS value

; test interrupt flags here

;-----

; resave routine

EndInt:

swapf STATUS_TEMP, W ; get saved STATUS value

movwf STATUS ; restore STATUS

swapf WREG_TEMP, F ; prepare WREG to be
restored

swapf WREG_TEMP, W ; restore WREG without
affecting STATUS

retfie ; return from interrupt

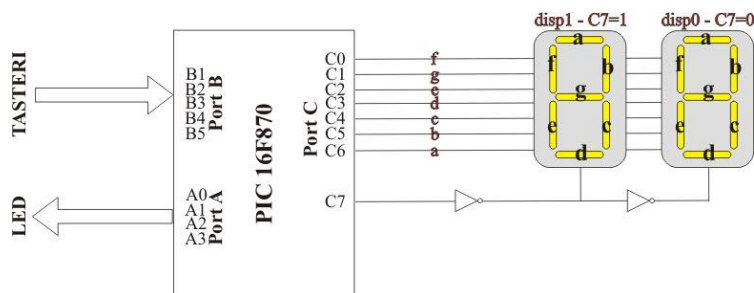
end

; kraj primera

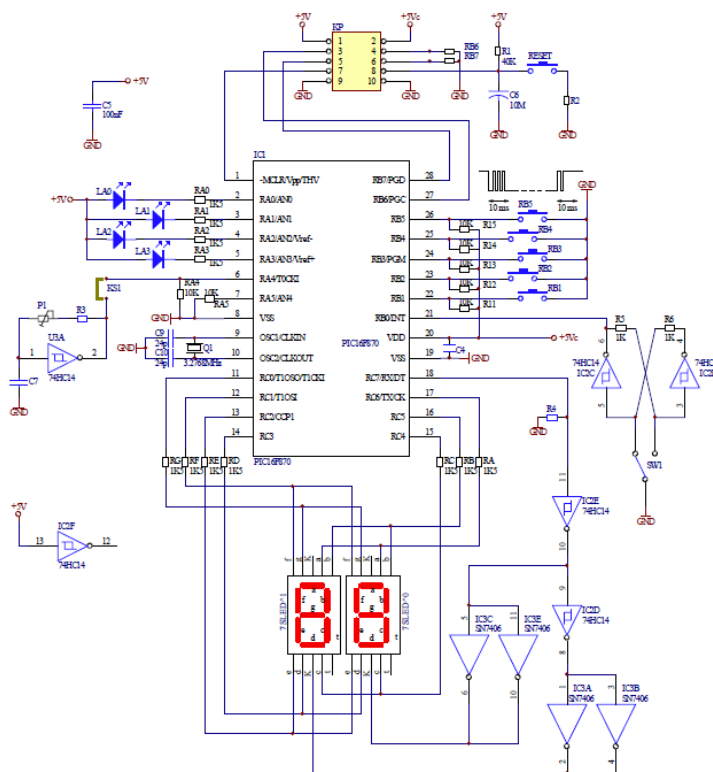
2.11 Prilog

Primer: Data je električna i blok šema sistema sa mikro kontrolerom PIC16F870, dva 7-segmentna displeja priključena na PORTC, pet tastera priključenih na RB1:RB5, prekidačem koji je izvor prekida na RB0/INT, 4 diode na RA0:RA3 i izorom spoljašnjeg takta na RA4.

Na RB1 je priključen taster povećaj, a na RB2 taster smanji. Početni sadržaj na displeju je 00. Pritiskom na taster „povećaj“ uvećati broj na displeju za 1. Pritiskom na taster „smanji“ smanjiti broj na displeju za 1. Osvežavanje displeja vršiti u prekidnom programu 100 puta u sekundi. Ispitivanje tastera vršiti 4 puta sporije (na 40 ms).



Slika 2.19



Slika 2.20


```
;=====
=====

;          Filename:  incdec.asm
list p=16f870;list directive to define processor
#include <p16f870.inc> ;processor          specific
definitions
errorlevel -302;suppress "not in bank 0" message
;#include <makro.asm>

__CONFIG __CP_OFF & __WDT_OFF & __BODEN_ON & __PWRTE_ON &
__HS_OSC & __WRT_ENABLE_ON & __LVP_OFF & __CPD_OFF

; '__CONFIG' direktiva se koristi da se konfiguracioni
bitovi ugrade ; u HEX fajl
; labele koje slede iza direktive se nalaze u
odgovarajucem .inc ; fajlu.

;-----
-----

;Promenljive

CBLOCK      0x20
WREG_TEMP   ;pamti W registar za vreme prekida
STATUS_TEMP ;pamti STATUS registar za vreme prekida
C0          ;desna cifra
C1          ;leva cifra
STANJE
broj
ENDC

;-----
-----

ORG          0x0004
goto        InterruptCode
```

PROGRAMIRANJE MIKROKONTROLERA

```
ResetCode:
    clrf      PCLATH    ;select program memory page 0
    goto     Main      ;go to beginning of program
Main:
    BANKSEL   TRISB      ;PRELAZAK U BANKU 1
    movlw    0xFF
    movwf    TRISB
    clrf     TRISA
    clrf     TRISC
    movlw    0x04        ;ucetanost      osvezavaanja
    ispleja je 100Hz
    movwf    OPTION_REG

    BANKSEL   PORTC      ;POVRATAK U BANKU 0
    clrf     PORTC
    clrf     PORTA
    clrf     C0
    clrf     C1
    clrf     broj
    movlw    1
    movwf    STANJE
    movlw    0xA0
    movwf    INTCON      ;dozvoljen prekid od TMR0
opet:
    goto     opet

DECOD:addwf   PCL,F
    ;        0abcdegf
    retlw    b'01111101' ;0
    retlw    b'00110000' ;1
    retlw    b'01101110' ;2
```

```
retlw      b'01111010' ;3
retlw      b'00110011' ;4
retlw      b'01011011' ;5
retlw      b'00011111' ;6
retlw      b'01110000' ;7
retlw      b'01111111' ;8
retlw      b'01110011' ;9
```

```
;-----
-----
```

;Ovaj kod se izvrsava kada se dogodi prekid

InterruptCode:

```
movwf      WREG_TEMP    ;save WREG
swapf      STATUS,W     ;store STATUS in WREG
clrf       STATUS       ;select file register bank0
movwf      STATUS_TEMP  ;save STATUS value
```

;ovde e ispiyuju indikatori prekida

```
bcf        INTCON,T0IF ;jer je samo taj prekid bio
dozvoljen
```

;osvezavamo displej

```
movf       C0,W
btfsc      broj,0
movf       C1,W;U W je C0 ili C1 u zavisnosti od
broj,0
call       DECOD
movwf      PORTC;dekodirana cifra je na PORTC i to
desno
btfsc      broj,0
bsf        PORTC,7;a sada je levo
incf       broj,F
```

PROGRAMIRANJE MIKROKONTROLERA

```
        movf      broj,W
        andlw     b'00000011'
        btfss     STATUS,Z
        goto      EndInt
;ispitujemo tastaturu svaki 4-ti put
        btfsc     PORTB,1;ispitivanje tastera uvecaj
        goto      nast0
        btfss     STANJE,0      ;TASTER JE PRITISNUT
        goto      EndInt
        clrf      STANJE;i to je prvo ispitivanje od kad
je pritisnut
        call      Uvecaj
        goto      EndInt

nast0:btfsc     PORTB,2;ispitivanje tastera smanji
        goto      nast1
        btfss     STANJE,0      ;TASTER JE PRITISNUT
        goto      EndInt
        clrf      STANJE;i to je prvo ispitivanje od kad
je pritisnut
        call      Smanji
        goto      EndInt

nast1:bsf       STANJE,0      ;nema pritisnutih tastera
        ;-----
        ;Kraj prekidne rutine vraćanje zapamćenih sadržaja
EndInt:
        swapf     STATUS_TEMP,W ;get saved STATUS value
        movwf     STATUS;restore STATUS
```

2.HARDVERSKI OPIS PIC16F84

```
        swapf      WREG_TEMP,F ;prepare   WREG    to    be
restored

        swapf      WREG_TEMP,W ;restore   WREG    without
affecting STATUS

        retfie     ;return from interrupt


;potprogram      Uvecaj
Uvecaj:incf       C0,F
        movf       C0,W
        xorlw      0x0A
        btfss      STATUS,Z      ;da li je 10
        return
        clrf       C0              ;jeste
        incf       C1,F
        movf       C1,W
        xorlw      0x0A
        btfsc      STATUS,Z      ;da li je 10
        clrf       C1              ;jeste
        return


;potprogram      Smanji
Smanji:movf       C0,W
        btfsc      STATUS,Z      ;da li je 0
        goto       ss1
        decf       C0,F
        return
ss1:
        movlw      9
        movwf      C0
        movf       C1,W
        btfsc      STATUS,Z      ;da li je 0
```

PROGRAMIRANJE MIKROKONTROLERA

```
    goto      ss2
    decf      C1,F
    return
ss2:movlw    9
    movwf    C1
    return
end
```

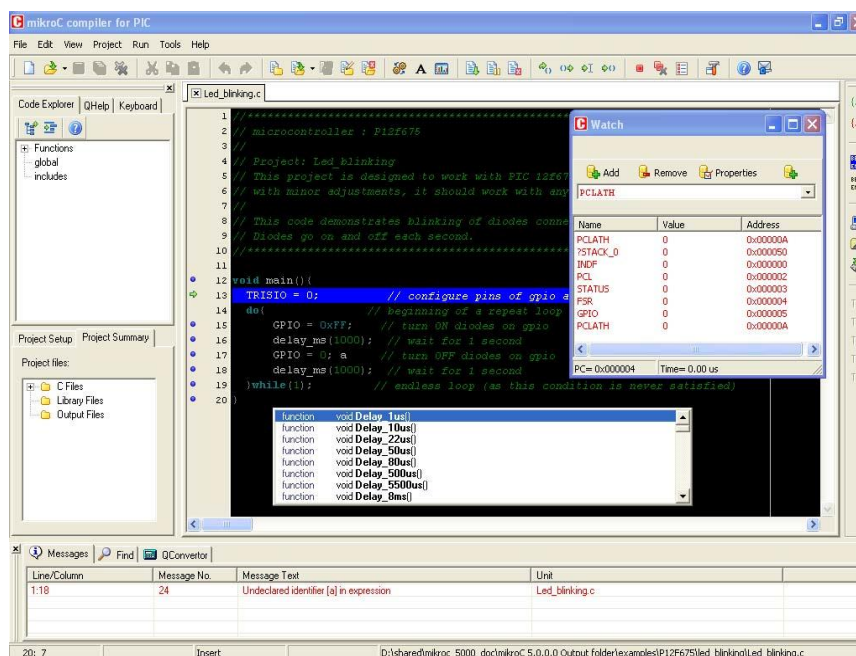
```
;-----
-----
```

```
; kraj primera
```

3 PREGLED MICROC PROGRAMSKOG OKRUŽENJA

MicroC je moćan razvojni alat za PIC kontrolere sa mnoštvom funkcija. Dizajniran je tako da obezbedi korisniku najlakši mogući način razvoja aplikacija za ugrađene sisteme bez kompromisa po pitanju performansi ili kontrole.

PIC i C se dobro uklapaju: PIC je najpopularniji 8-bitni čip na svetu i koristi se u širokom dijapazonu primena dok je C, cenjen zbog svoje efikasnosti, prirodan izbor za razvoj ugrađenih sistema. MicroC pruža taj uspešan spoj korištenjem visoko-naprednog IDE-a, ANSI -kompatibilnog kompajlera, širokog skupa hardverskih biblioteka, sveobuhvatnom dokumentacijom i mnoštvom primera spremnih za upotrebu.



Slika 3.1 - Razvojno okruženja MicroC

MicroC vam omogućava da brzo razvijete i primenite kompleksne aplikacije:

- Napišite vaš C izvorni kod korištenjem naprednog editora koda.
- Koristite ugrađene microC biblioteke da dramatično ubrzate razvoj: sakupljanje podataka, memorija, displeji, konverzije, komunikacije...
- Nadzirite strukturu vašeg programa, promenljive, i funkcije u Code Explorer-u. Generišite komentarisane asemblerski kod razumljiv programerima i standardne HEX kompatibilne fajlove.
- Istražite tok programa i očistite od grešaka izvršnu logiku programa integrisanim Debugger-om. Dobijte detaljne izveštaje i grafove o statistici koda, listingu asemblera, stablu poziva...
- Obezbedili smo mnoštvo primera koje možete proširivati, razvijati i koristiti kao osnovne gradivne elemente u vašim projektima.

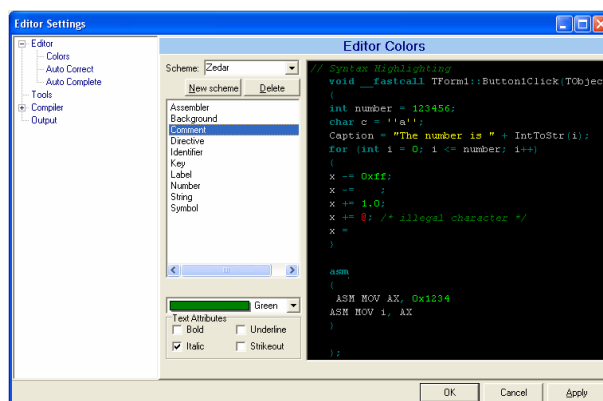
3.1 Code editor (*Editor koda)

Code Editor je napredni editor teksta uobličen tako da zadovolji profesionalne zahteve. Uopšteno editovanje koda je isto kao i u bilo kojem standardnim editoru teksta uključujući poznate Copy, Paste, i Undo komande, uobičajene za Windows okruženje.

Napredni editor takođe uključuje:

- Podesivo Naglašavanje Sintakse
- Pomoć za Kod
- Pomoć za Parametre
- Obrazac Koda (Automatsko Kompletiranje)
- Automatsku ispravku za uobičajene greške u kucanju
- Obeleživače i Goto Liniju

Ove opcije možete podesiti po sopstvenom nahođenju u dijalogu Editor Settings. Da pristupite podešavanjima odaberite Tools>Options u padajućem meniju ili kliknite na ikonicu Tools.



Slika 3.2 - Postavljanje opcija editora

3.1.1 Code Assistant [CTRL+SPACE] (Pomoć za Kod)

Ako otkucate prvih nekoliko slova reči i pritisnete CTRL+SPACE, svi validni identifikatori koji se poklapaju sa slovima koja ste otkucali, će se pojaviti u plutajućem panelu (pogledajte sliku). Sada možete nastaviti da kucate i time smanjivati izbor ili odabrati sa liste koristeći strelice na tastaturi i Enter.



Slika 3.3 - Code Assistant prozor

3.1.2 Parameter Assistant [CTRL+SHIFT+SPACE] (*Pomoć za parametre)

Parameter Assistant se automatski poziva kada otvorite malu zagradu ili pritisnete CTRL+SHIFT+SPACE. Ako ime validne funkcije prethodi zagradi, onda će očekivani parametri biti predstavljeni u plutajućem panelu. Dok kucate stvarni parametar, sledeći očekivani parametar postaje podebljan.



Slika 3.4

3.1.3 Code Template [CTRL+J] (*Obrazac Koda)

Možete da ubacite obrazac koda kucajući ime obrasca (npr. whileb), pa potom pritisnete CTRL+J, i Editor Koda će automatski generisati kod. Možete i kliknuti na dugme sa Code linije alata i odabrati obrazac sa liste.

Možete dodati i vaše sopstvene obrasce na listu. Dovoljno je da odaberete Tools>Options iz padajućeg menija ili da kliknete na Tools ikonicu sa Settings linije alata pa potom odaberete Auto Complete odeljak. Ovde možete uneti odgovarajuću ključnu reč, opis, i kod vašeg obrasca.

3.1.4 Auto Correct (*Automatska Ispravka)

Auto Correct služi za ispravljanje uobičajenih grešaka u kucanju. Da pristupite listi prepoznatljivih grešaka odaberite Tools >Options iz padajućeg menija ili kliknite na Tools ikonicu, i potom odaberite Auto Correct odeljak. Ovde možete staviti i vaše sopstvene česte greške na listu.



Slika 3.5

3.1.5 Comment/Uncomment (*Postavljanje/Uklanjanje oznaka za komentar)

Editor koda vam dopušta da odabrani deo koda proglasite za komentar ili ponovo za kod jednostavnim klikom miša, koristeći Comment/Uncomment ikonice sa Code linije alata.

3.1.6 Bookmarks (*Obeleživač)

Obeleživači olakšavaju navigaciju kroz velike kodove.

CTRL+<broj>: Idi na obeleživač

CTRL+SHIFT+<broj>: Postavi obeleživač

3.1.7 Goto Line (*Idi na liniju)

Opcija Goto line olakšava navigaciju kroz velike kodove. Odaberite Search>Goto line iz padajućeg menija, ili koristite prečicu CTRL+G.

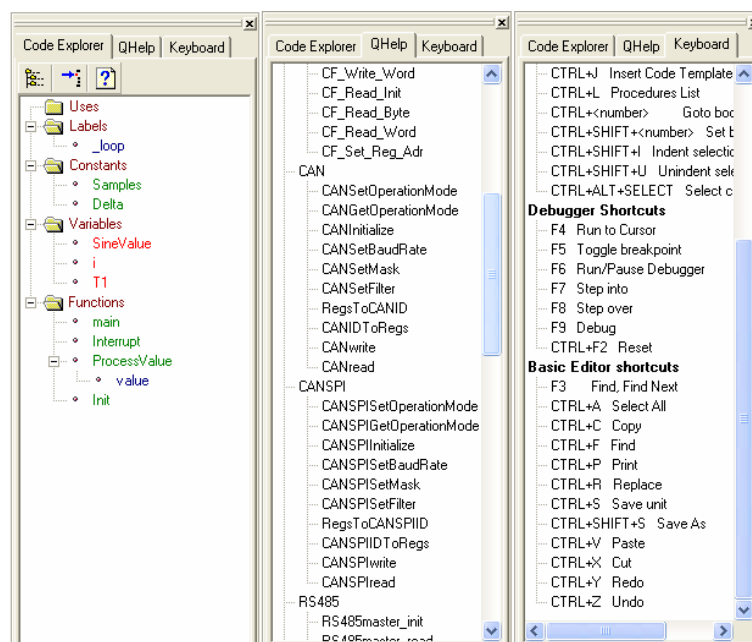
3.2 Code explorer (*Preglednik koda)

Code explorer je postavljen levo od glavnog prozora po podrazumevanim vrednostima i daje čist pregled svake deklarisanе stavke u izvornom kodu. Možete skočiti na deklaraciju bilo koje stavke klikanjem na nju ili klikanjem na Find Declaration (Nađi Deklaraciju) ikonicu. Da proširite ili suzite pregled po stablu u Code Explorer-u, koristite Collapse/Expand All ikonicu.



Slika 3.6

Takođe, u Code Explorer-u su dostupna još dva odeljka. QHelp Odeljak predstavlja listu svih prisutnih ugrađenih i bibliotečkih funkcija, za brzo objašnjenje. Dupli klik na neku rutinu u QHelp odeljku otvara relevantnu Help temu. Keyboard odeljak lista sve postojeće prečice tastature u mikroC-u.



Slika 3.7 - MicroC Debugger

3.3 Debugger (*Prečištač koda)



Slika 3.8

Debugger na izvornom nivou je integralna komponenta mikroC razvojnog okruženja. Dizajniran je da simulira rad Microchip Technology PICmicro-a i da pomogne korisnicima u otklanjanju grešaka u programima pisanim za ove uređaje.

Debugger simulira tok programa i izvršavanje linija instrukcija ali ne oponaša u potpunosti PIC uređaje – ne obnavlja tajmer-e, prekidne indikatore itd.

Pošto ste uspešno kompajlirali vaš projekat, možete pokrenuti Debugger odabirom Run>Debug iz padajućeg menija, ili klikom na Debug ikonicu. Sa pokretanjem debugger postaju dostupne nove opcije: Step into, Step Over, Run to Cursor, itd. Linija koja će se izvršiti je naglašena bojom.

3.3.1 Primer

Startuje Debugger



Slika 3.9

3.3.2 Run/Pause Debugger [F6]

Pokreće ili pauzira Debugger



Slika 3.10

3.3.3 Step Into [F7]

Izvršava trenutnu C instrukciju (jedno ili više-ciklusnu) i onda se zaustavlja. Ako je instrukcija poziv rutine, preskače je i zaustavlja se na prvoj instrukciji koja sledi poziv.



Slika 3.11

3.3.4 Step Out[CTRL+F8]

Izvršava trenutnu C instrukciju (jedno ili više-ciklusnu) i onda se zaustavlja. Ako je instrukcija u okviru rutine izvršava instrukciju i zaustavlja se na prvoj instrukciji koja sledi poziv.



Slika 3.12

3.3.5 Run to Cursor[F4]

Izvršava sve instrukcije između trenutne instrukcije i pozicije kurzora.



Slika 3.13

3.3.6 Toggle Breakpoint[F5]

Postavlja prekidnu tačku na trenutnoj poziciji kurzora. Da vidite sve prekidne tačke odaberite Run>View Breakpoints iz padajućeg menija. Dupli klik na stavku iz liste locira prekidnu tačku.

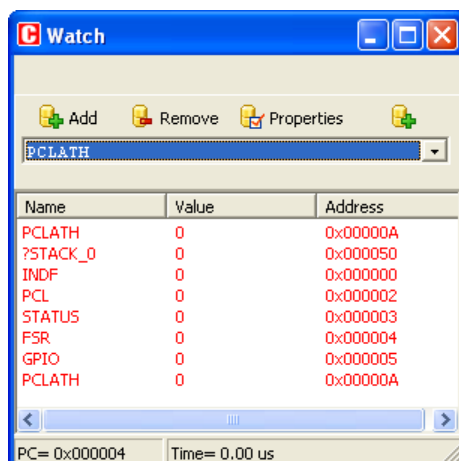


Slika 3.14

3.3.7 Watch Window (*Prozor Pregleda)

3.4 Promenljive

Watch Window vam dozvoljava da nadzirete programske stavke dok se vaš program izvršava. On prikazuje promenljive i registre specijalnih funkcija PIC MCU-a, njihove adrese i vrednosti. Vrednosti se usklađuju kako idete kroz simulaciju.



Slika 3.16 - Watch prozor

Dupli klik na neku od stavki otvara prozor u kojem možete dodeliti novu vrednost odabranoj promenljivoj ili registru i promeniti način formatiranja brojeva.

3.4.1 Stopwatch Window (*Prozor Štoperice)

Stopwatch Window prikazuje trenutni broj ciklusa/vremena od poslednje akcije Debugger-a. *Štoperica* meri vreme izvršenja (broj ciklusa) od momenta pokretanja Debugger-a i može biti resetovana u bilo kojem trenutku. Delta predstavlja broj ciklusa između prethodne linije instrukcija (linije gde je obavljena aktivnost debugger-a) i aktivne linije instrukcija (gde je aktivnost Debugger-a zastala).

Primedba: Možete promeniti klok u Stopwatch Window-u; ovaj postupak će dovesti do preračunavanja vrednosti za novo-specifikovanu frekvenciju. Promena klocka u Stopwatch Window-u ne utiče na stvarna podešavanja u projektu – ona samo pruža simulaciju.

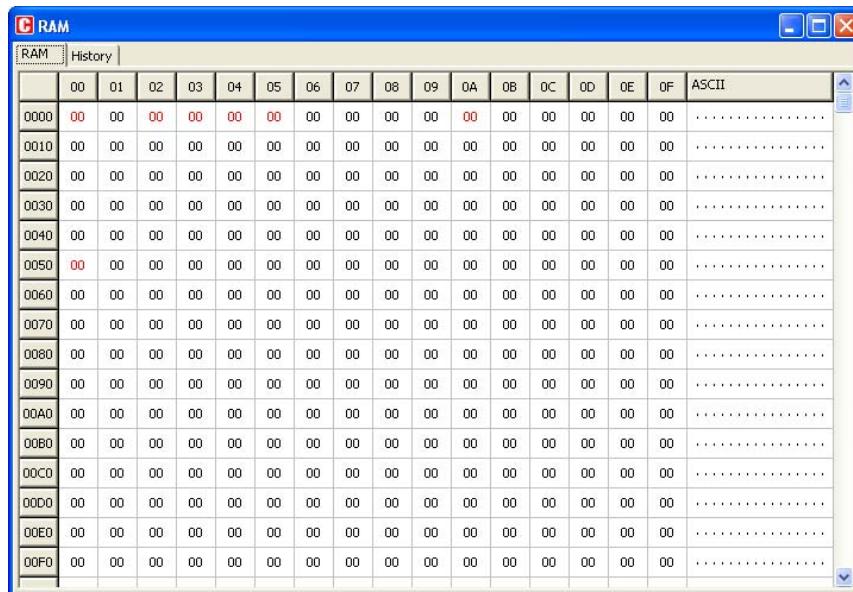


Slika 3.17 - Stopwatch prozor

3.4.2 View RAM Window (*Prozor Pregleda RAM-a)

Debugger-ovom View RAM Window-u se pristupa iz padajućeg menija **View>Debug Windows>View RAM**.

View RAM Window prikazuje mapu PIC-ovog RAM-a, gde su skorašnje promene predstavljene crvenom bojom. Možete promeniti vrednost bilo kojeg polja klikom na to polje.

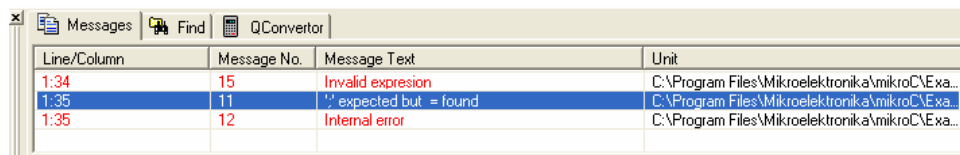


Slika 3.18 - RAM prozor

3.5 Error window (*Prozor greške)

U slučaju da se jave greške tokom kompajliranja, kompajler će ih prijaviti i neće generisati HEX fajl. Error Window će biti pozvan i prikazan na dnu glavnog prozora po podrazumevanim podešavanjima.

Error Window je postavljen ispod odeljka sa porukama i prikazuje lokaciju i tip grešaka sa kojima se kompajler susreo. Kompajler takođe prijavljuje upozorenja ali ona ne utiču na rezultat; jedino greške mogu uticati na generisanje HEX-a.



Slika 3.19 - Prozor greške

Dupli klik na liniju poruke u Error Window-u naglašava liniju gde se je nađena greška. Proverite Error Messages da dobijete više informacija o greškama koje kompajler prepoznaje.

3.6 Statistics (*Statistika)

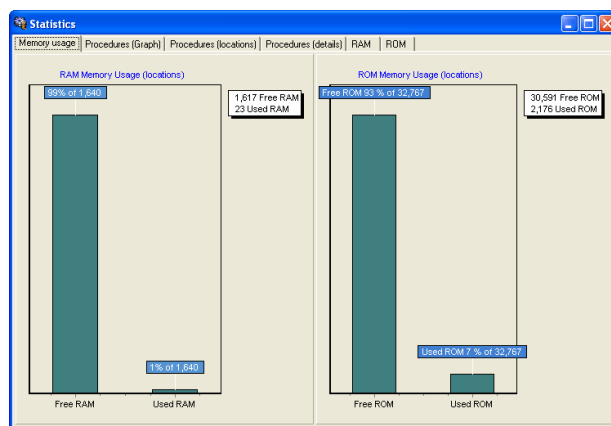


Slika 3.20

Nakon uspješne kompilacije možete pogledati statistiku vašeg koda. Odaberite Project>View Statistics iz padajućeg menija, ili kliknite ikonicu Statistics. Postoji šest prozora pod odeljcima.

3.6.1 Memory Usage Window (*Prozor Iskorištenja Memorije)

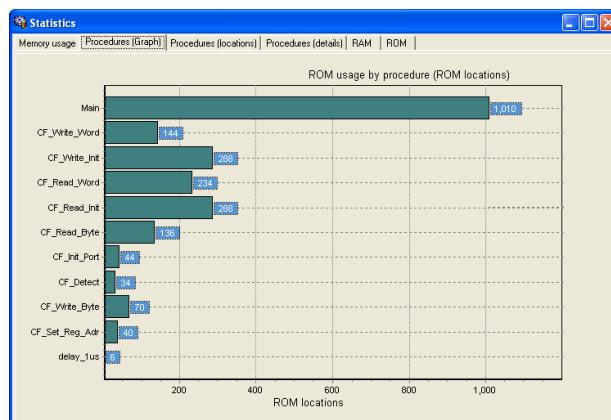
Pruža pregled iskorištenja RAM i ROM memorije u formi histograma.



Slika 3.21 - Pregled iskorištenja RAM i ROM memorije u formi histograma.

3.6.2 Procedure (Graph) Window (*Prozor Procedura (Graf))

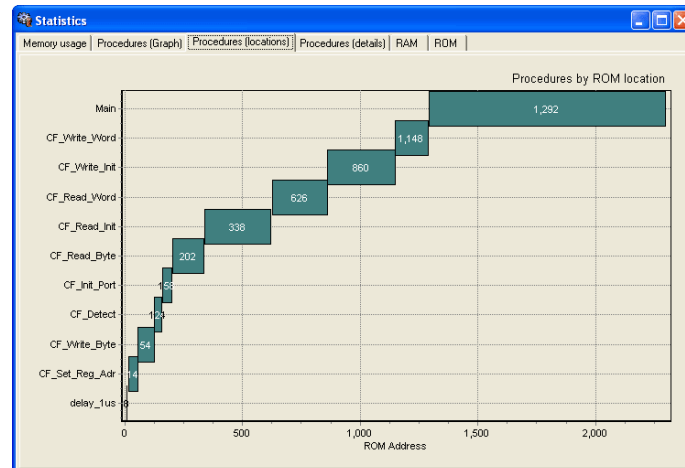
Prikazuje funkcije u formi histograma u skladu sa memorijom koja im je dodeljena.



Slika 3.22 - Funkcije u formi histograma u skladu sa memorijom koja im je dodeljena

3.6.3 Procedures (Locations) Window (*Prozor Procedura(Lokacije))

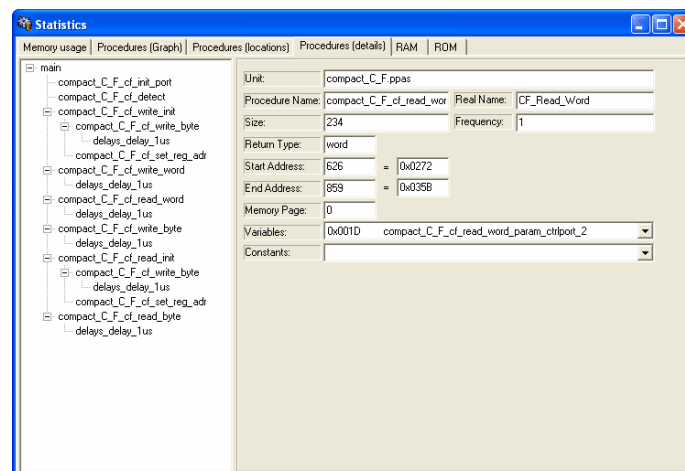
Prikazuje kako su funkcije raspodeljene u memoriji mikrokontrolera.



Slika 3.23 - Raspored funkcija u memoriji mikrokontrolera

3.6.4 Procedures (Details) Window (*Prozor Procedura(Detalji))

Prikazuje stablo poziva sa detaljima za svaku funkciju;



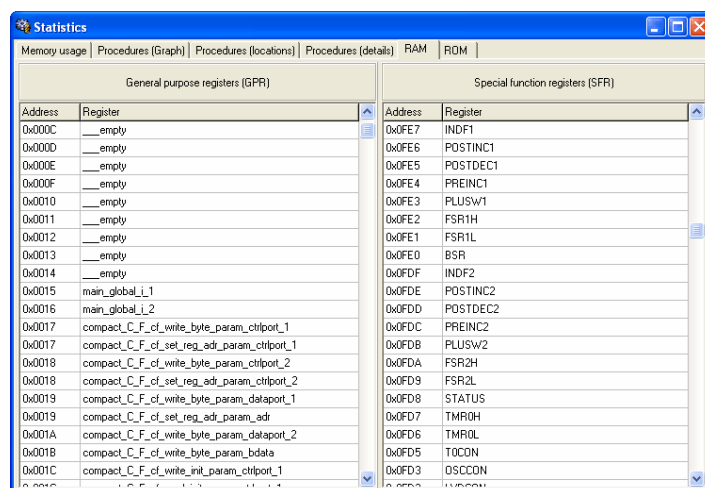
Slika 3.24 - Stablo poziva sa detaljima za svaku funkciju

Veličinu, početnu i krajnju adresu, frekvenciju poziva, tip vraćene vrednosti, itd.

3.6.5 RAM Window (*Prozor RAM-a)

Pregled svih GPR i SFR registara i njihovih adresa. Takođe prikazuje simbolička imena promenljivih i njihove adrese.

3. PREGLED MICROC PROGRAMSKOG OKRUŽENJA

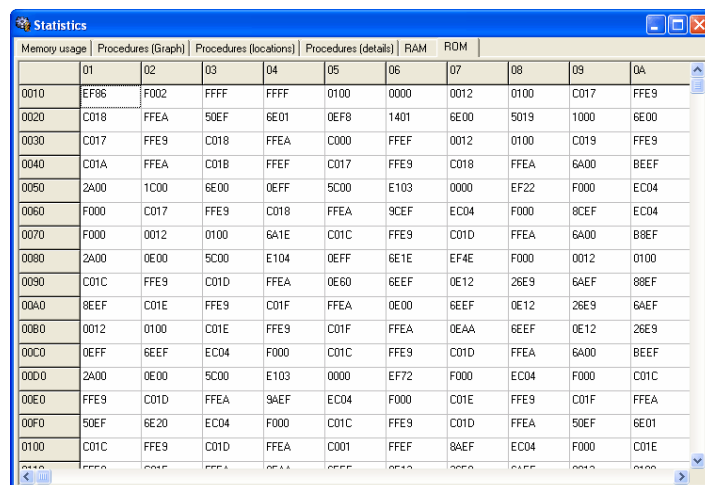


General purpose registers (GPR)		Special function registers (SFR)	
Address	Register	Address	Register
0x000C	__empty	0x0FE7	INDF1
0x000D	__empty	0x0FE6	POSTINC1
0x000E	__empty	0x0FE5	POSTDEC1
0x000F	__empty	0x0FE4	PREINC1
0x0010	__empty	0x0FE3	PLUSW1
0x0011	__empty	0x0FE2	FSR1H
0x0012	__empty	0x0FE1	FSR1L
0x0013	__empty	0x0FE0	BSR
0x0014	__empty	0x0DFD	INDF2
0x0015	main_global_i_1	0x0FDE	POSTINC2
0x0016	main_global_i_2	0x0FDD	POSTDEC2
0x0017	compact_C_F_cf_write_byte_param_ctlport_1	0x0FDC	PREINC2
0x0017	compact_C_F_cf_set_reg_adr_param_ctlport_1	0x0FDB	PLUSW2
0x0018	compact_C_F_cf_write_byte_param_ctlport_2	0x0FDA	FSR2H
0x0018	compact_C_F_cf_set_reg_adr_param_ctlport_2	0x0FD9	FSR2L
0x0019	compact_C_F_cf_write_byte_param_dataport_1	0x0FD8	STATUS
0x0019	compact_C_F_cf_set_reg_adr_param_adr	0x0FD7	TMR0H
0x001A	compact_C_F_cf_write_byte_param_dataport_2	0x0FD6	TMR0L
0x001B	compact_C_F_cf_write_byte_param_bdata	0x0FD5	TOCON
0x001C	compact_C_F_cf_write_init_param_ctlport_1	0x0FD3	OSCCON

Slika 3.25 - Pregled svih GPR i SFR registara i njihovih adresa.

3.6.6 - ROM Window (*Prozor ROM-a)

Lista op-kodove i njihove adrese u formi HEX koda razumljivog programeru.



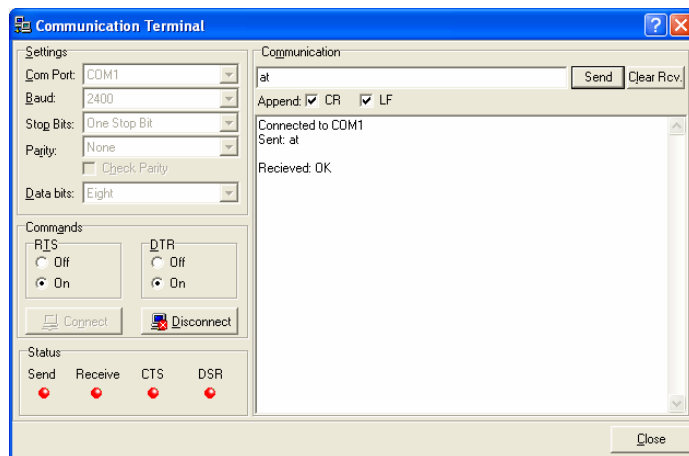
	01	02	03	04	05	06	07	08	09	0A
0010	EF86	F002	FFFF	FFFF	0100	0000	0012	0100	C017	FFE9
0020	C018	FFEA	50EF	6E01	0EF8	1401	6E00	5019	1000	6E00
0030	C017	FFE9	C018	FFEA	C000	FFEF	0012	0100	C019	FFE9
0040	C01A	FFEA	C01B	FFEF	C017	FFE9	C018	FFEA	6A00	BEEF
0050	2A00	1C00	6E00	0EFF	5C00	E103	0000	EF22	F000	EC04
0060	F000	C017	FFE9	C018	FFEA	9CEF	EC04	F000	8CEF	EC04
0070	F000	0012	0100	6A1E	C01C	FFE9	C01D	FFEA	6A00	B8EF
0080	2A00	0E00	5C00	E104	0EFF	6E1E	EF4E	F000	0012	0100
0090	C01C	FFE9	C01D	FFEA	0E60	6EEF	0E12	26E9	6AEF	88EF
00A0	8EEF	C01E	FFE9	C01F	FFEA	0E00	6EEF	0E12	26E9	6AEF
00B0	0012	0100	C01E	FFE9	C01F	FFEA	0EAA	6EEF	0E12	26E9
00C0	0EFF	6EEF	EC04	F000	C01C	FFE9	C01D	FFEA	6A00	BEEF
00D0	2A00	0E00	5C00	E103	0000	EF72	F000	EC04	F000	C01C
00E0	FFE9	C01D	FFEA	9AEF	EC04	F000	C01E	FFE9	C01F	FFEA
00F0	50EF	6E20	EC04	F000	C01C	FFE9	C01D	FFEA	50EF	6E01
0100	C01C	FFE9	C01D	FFEA	C001	FFEF	8AEF	EC04	F000	C01E

Slika 3.26 - Lista op-kodova i njihovih adresa u formi HEX koda razumljivog programeru

3.7 Integrated tools (*Integrirani alati)

3.7.1 Usart Terminal

MikroC sadrži USART(Universal Synchronous Asynchronous Receiver Transmitter (*Univerzalni Sinhrono Asinhroni Primopredajnik)) komunikacioni terminal za RS232 komunikaciju. Možete ga pokrenuti iz padajućeg menija: Tools>Terminal ili klikom na Terminal ikonicu.



Slika 3.27 - USART Terminal

3.7.2 ASCII Chart (*ASCII Tabela)

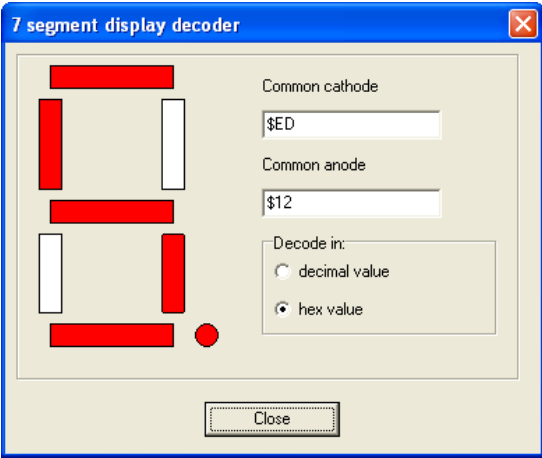
ASCII Chart je korisan alat pogotovo pri radu sa LCD displejima. Možete ga pokrenuti iz padajućeg menija Tools>ASCII chart.

CHAR	DEC	HEX	BIN
NUL	0	0x00	0000 0000
SOH	1	0x01	0000 0001
STX	2	0x02	0000 0010
ETX	3	0x03	0000 0011
EOT	4	0x04	0000 0100
ENQ	5	0x05	0000 0101
ACK	6	0x06	0000 0110
BEL	7	0x07	0000 0111
BS	8	0x08	0000 1000
HT	9	0x09	0000 1001
LF	10	0x0A	0000 1010
VT	11	0x0B	0000 1011
FF	12	0x0C	0000 1100
CR	13	0x0D	0000 1101

Slika 3.28 - ASCII karta

3.7.3 7 Segment Display Decoder (7-mo Segmentni Displej Dekoder)

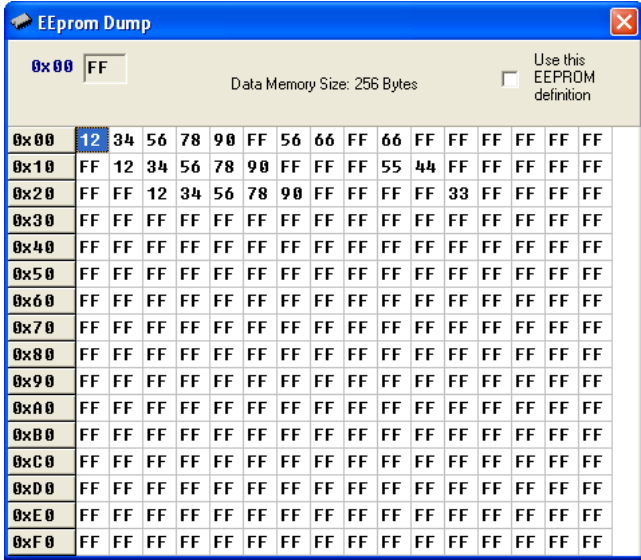
7 Segment Display Decoder je prikladan vizuelni panel koji vraća decimalnu/heksadecimalnu vrednost za bilo koju važeću kombinaciju koju bi želeli da prikazete na 7-mo segmentom displeju. Klikom na delove 7-mo segmentne slike dobijate željene vrednosti u edit poljima. Možete ga pokrenuti iz padajućeg menija Tools>7 Segment Display.



Slika 3.29 - Sedmosegmetni displej

3.7.4 EEPROM Editor

EEPROM Editor vam omogočava da lako upravljate EEPROM-om PIC mikrokontrolera.



Slika 3.30 - EEprom dump memorije

3.7.5 mikroBootLoader

mikroBootLoader se može koristiti samo sa PICmikroima koji podržavaju flash upis.

1. Unesite u PIC odgovarajući HEX fajl korištenjem konvencionalnih programskih tehnika (npr. za PIC16F877A koristite p16f877a.hex).
2. Pokrenite mikroBootLoader iz padajućeg menija Tools>Bootloader
3. Kliknite na Setup Port I odaberite COM port koji će biti korišten. Osigurajte da je BAUD postavljena na 9600 Kbps.

4. Kliknite na Open File i odaberite HEX fajl za upload.
5. Pošto bootcode u PIC-u daje kompjuteru samo 4-5 sekundi da se priključi, trebalo bi da resetujete PIC i da kliknete na Connect dugme u roku od 4-5 sekundi.
6. Poslednja linija u history prozoru bi onda trebalo da glasi "Connected".
7. Da počnete sa upload-om dovoljno je da kliknete na Start Bootloader dugme.
8. Vaš program će biti upisan u flaš PIC-a. Bootloader će prijaviti ako se jave ikakve greške.
9. Resetujte vaš PIC i pošnite sa izvršenjem.

Boot code daje kompjuteru 5 sekundi da se priključi na njega. Ako se to ne desi, on započinje sa izvršavanjem postojećeg korisničkog koda. Ako postoji novi korisnički kod za download, boot code prima i piše podatke u programsku memoriju.

Češće osobine koje bootloader može imati su pobrojane ispod:

- Kod na lokaciji reseta
- Kod drugde u maloj oblasti memorije
- Provera da vidi da li korisnik želi da se unese novi korisnički kod
- Započinje izvršenje korisničkog koda ako nema novog korisničkog koda za unos
- Prima novi korisnički kod preko komunikacionog kanala ako kod treba da se unese
- Programira novi korisnički kod u memoriju

3.7.6 Integrisanje Korisničkog i Boot koda

Boot kod skoro uvek koristi Reset lokaciju i nešto dodatne programske memorije. To je jedno jednostavno parče koda koje nema potrebe da koristi prekide; stoga korisnički kod može da koristi normalni vector prekida na 0x0004. Boot kod mora da izbegne korišćenje prekidnog vektora tako da bi trebalo da ima programsko grananje u adresnom opsegu 0x0000 do 0x0003. Boot kod mora biti programiran u memoriju korišćenjem konvencionalnih programskih tehnika i konfiguracioni bitovi moraju biti programirani u ovom trenutku. Boot kod nije u stanju da pristupi konfiguracionim bitovima pošto oni nisu označeni u prostoru programske memorije.

3.8 Keyboard shortcuts (*Prečice sa tastature)

Dole sledi kompletna lista prečica sa tastature dostupnih u mikroC IDE-u. Prečice sa tastature takođe možete videti u Code explorer-u, odeljak Keyboard.

3.8.1 IDE Prečice

F1	Help(*Pomoć)
CTRL+N	New Unit(*Nova Jedinica)
CTRL+O	Open(*Otvori)

CTRL+F9 Compile(*Kompajliraj)
CTRL+F11 Code Explorer on/off (*uključen/isključen)
CTRL+SHIFT+F5 View brekpoints(*Pogledaj obeleživače)

3.8.2 Osnovne prečice editora

F3 Find/Find Next (*Nađi, Nađi sledeći)
CTRL+A Select all(*Odaberi sve)
CTRL+C Copy
CTRL+F Find (*Nađi)
CTRL+P Print(*Štampaj)
CTRL+R Replace(*Zameni)
CTRL+S Save unit(*Snimi jedinicu)
CTRL+SHIFT+S Save as
CTRL+V Paste
CTRL+X Cut
CTRL+Y Redo
CTRL+Z Undo

3.8.3 Napredne prečice editora

CTRL+SPACE Code Assistant(*Pomoć za kod)
CTRL+SHIFT+SPACE Parameters Assistant(*Pomoć za parametre)
CTRL+D Find declaration(*Nađi deklaraciju)
CTRL+G Goto line (*idi na liniju)
CTRL+J Insert Code Template(*Ubaci obrazac koda)
CTRL+<number(*broj)> Goto bookmark(*Idi na obeleživač)
CTRL+SHIFT+<number(*broj)> Set bookmark(*Postavi obeleživač)
CTRL+SHIFT+I Indent selection(*Ubaci razmak za odabrano)
CTRL+SHIFT+U Unindent selection(*Izbaci razmak za odabrano)
CTRL+ALT+SELECT Select columns(*Odaberi kolone)

3.8.4 Prečice Debugger-a

F4 Run to cursor(*Pokreni do kurzora)
F5 Toggle breakpoint(*Uključi prekidnu tačku)
F6 Run/Pause Debugger(*Pokreni/Pauziraj Debugger)

F7	Step into(*Zakorači u)
F8	Step over (*Zakorači preko)
F9	Debug
CTRL+F2	Reset

3.1.5 Izrada aplikacija

Izrada aplikacija i mikroC-u je laka i intuitivna. Project Wizard (*Čarobnjak za Projekte) vam omogućava da postavite vaš projekat u samo nekoliko klikova: imenujte vašu aplikaciju, odaberite čip, postavite indikatore, i krenite.

MikroC vam dopušta da distribuirate vaše projekte u onoliko fajlova koliko vi nalazite za shodno, a potom možete deliti vaše mikro-kompajlirane biblioteke (.mcl) sa drugim programerima bez otkrivanja izvornog koda. Najbolje od svega je što možete koristiti .mcl pakete kreirane mikroPascal-om ili mikroBasic-om.

3.9 Projects (*Projekti)

MikroC organizuje aplikacije u projekte, koji se sastoje je od jednog projektnog fajla (ekstenzija .ppc) i jednog ili više izvornih fajlova(ekstenzija .c). Izvorne fajlove možete kompajlirati samo ako su deo projekta.

Projektni fajl nosi sledeće informacije:

- Ime projekta i opcionalni opis
- Ciljni uređaj
- Indikatori uređaja (konfiguraciona reč) i klok uređaja
- Listu izvornih fajlova projekta sa putanjama

3.9.1 New Project (*Novi Projekt)



Slika 3.31

Najlakši način za kreaciju projekta je uz pomoć New Project Wizard-a, iz padajućeg menija odaberite Project>New Project. Dovoljno je da popunite dijalog sa željenim vrednostima (ime projekta i opis, lokacija, uređaj, klok, konfiguraciona reč) i mikroC će kreirati odgovarajući projektni fajl. Takođe, po podrazumevanim postavkama će se kreirati jedan prazan izvorni fajl imenovan po projektu.

3.9.2 Editing Project(*Uređivanje projekta)



Slika 3.32

Kasnije možete promeniti postavke projekta iz padajućeg menija Project>Edit Project. Možete preimenovati projekt, modifikovati njegov opis, promeniti čip, klok, konfiguracionu reč, itd. Ako želite da obrišete projekt jednostavno obrišite folder gde je on snimljen.

3.9.3 Add/Remove Files from Project (*Dodajte/Odstranite Fajlove iz Projekta)



Slika 3.33



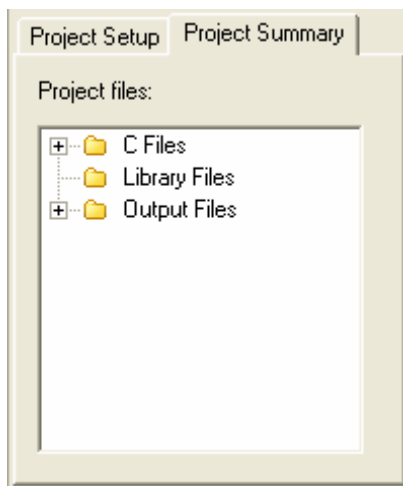
Slika 3.34

Projekt može sadržati proizvoljan broj izvornih fajlova (ekstenzija .c). Lista relevantnih izvornih fajlova je spremljena u projektnom fajlu (ekstenzija .ppc). Da dodate izvorni fajl vašem projektu, odaberite Project>Add to Project iz padajućeg menija. Svaki dodani izvorni fajl mora biti samodovoljan, npr. mora imati sve neophodne definicije posle pred-procesuiranja. Da odstranite fajl (fajlove) iz vašeg projekta odaberite Project>Remove Project iz padajućeg menija.

Primedba: Za uključenje fajlova zaglavlja, korsistite pred-procesorsku direktivu #include.

3.9.4 Proširena funkcionalnost Project Files odeljka

Korištenjem novih funkcija dodatih Project Files-u, možete dosegnuti sve izlazne fajlove (.lst, .asm) jednim klikom. Takođe možete uključiti u projekt bibliotečke fajlove (.mcl), koje su vaše sopstvene ili projektno-specifične koje se nalaze po podrazumevanim podešavanjima kompajlera.



Slika 3.35 - Project Files odeljak

Biblioteke (.mcl) sada imaju drugačiji, kompaktniji format u poređenju sa mikroC verzijom 2. Ovo međutim znači da bibliotečki formati sada nisu kompatibilni. Korisnici koji se

prebacuju sa verzije 2 na 5 moraju ponovo kompajlirati sve njihovo prethodno napisane biblioteke da bi mogli da ih koriste u novoj verziji. Sav izvorni kod napisan i testiran u prethodnim verzijama bi trebao da se ispravno kompajlira u verziji 5.0, sa izuzetkom `asm{ }` blokova koji su komentarisani u `asm` odeljku `help-a`.

3.10 Source files (*Izvorni fajlovi)

Izvorni fajlovi koji sadrže C kod bit trebalo da imaju `.c` ekstenziju. Lista izvornih fajlova od značaja za aplikaciju se drži u projektnom fajlu sa ekstenzijom `.ppc`, uz ostale informacije o projektu. Izvorne fajlove možete kompajlirati samo ako su deo projekta.

Koristite pred-procesorsku direktivu `#include` da uključite zaglavlja. Ne oslanjajte se na pred-procesor za uključenje drugih izvornih fajlova – pogledajte Projects za više informacija.

3.10.1 Putanje Traženja

3.10.2 Putanje za izvorne fajlove (.c)

Možete specifikovati svoje sopstvene putanje traženja odabirom Tools>Options a zatim odeljak Advanced.

U project settings (*projektna podešavanja) možete specifikovati apsolutnu ili relativnu putanju do izvornog fajla. Ako odredite relativnu putanju mikroC će tražiti fajl na sledećim lokacijama i po sledećem redu:

1. Projektni folder (folder koji sadrži projektni fajl `.ppc`)
2. Vaša sopstvena putanja traženja
3. mikroC instalacioni folder > “uses” folder.

3.10.3 Putanje za fajlove zaglavlja (.h)

Fajlovi zaglavlja se uključuju preko pred-procesorske direktive `#include`. Ako postavite eksplicitnu putanju do fajla zaglavlja u vašoj pred-procesorskoj direktivi, tražiće se samo na toj lokaciji.

Možete specifikovati vaše sopstvene putanje traženja: odaberite Tools>Options iz padajućeg menija a potom Search Path.

U projektnim podešavanjima možete specifikovati apsolutnu ili relativnu putanju do zaglavlja. Ako odredite relativnu putanju mikroC će tražiti fajl na sledećim lokacijama i po sledećem redu:

1. Projektni folder (folder koji sadrži projektni fajl `.ppc`)
2. mikroC instalacioni folder > “include” folder
3. Vaša sopstvena putanja traženja

3.10.4 Upravljanje Izvornim Fajlovima

3.10.5 Kreiranje novog izvornog fajla



Slika 3.36

Da kreirate novi izvorni fajl, uradite sledeće:

Odaberite File>New iz padajućeg menija, ili pritisnite CTRL+N, ili kliknite na New File ikonicu. Otvoriće se novi odeljak, imenovan "Untitled 1". Ovo je vaš novi izvorni fajl. Odaberite File>Save As iz padajućeg menija i imenujte ga po vašoj želji. Ako ste koristili New Project Wizard, prazan izvorni fajl, imenovan po projektu sa ekstenzijom c je kreiran automatski. MikroC ne zahteva da imate izvorni fajl sa istim imenom kao projekt-takvo imenovanje je urađeno u cilju pojednostavljenja.

3.10.6 Otvaranje postojećeg fajla



Slika 3.37

Odaberite File>Open iz padajućeg menija, ili pritisnite CTRL+O, ili kliknite na Open File ikonicu. Otvara se dijalog Select Input File. U dijalogu pronađite lokaciju fajla koji želite da otvorite i odaberite ga. Kliknite na Open dugme. Odabrani fajl je prikazan u svom sopstvenom odeljku. Ako je odabrani fajl već otvoren njegov trenutni editorski odeljak će postati aktivan.

3.10.7 Štampanje otvorenog fajla



Slika 3.38

Uverite se da je prozor koji sadrži fajl koji želite da štamplete aktivan. Odaberite File>print iz padajućeg menija ili pritisnite CTRL+P, ili kliknite na print ikonicu. U prozoru Print Preview odredite željeni oblik dokumenta i pritisnite OK dugme. Fajl će biti odštampan na odabranom printeru.

3.10.8 Snimanje fajla



Slika 3.39

Uverite se da je prozor koji sadrži fajl koji želite da snimate aktivan. Odaberite File>save As iz padajućeg menija, ili pritisnite SHIFT+CTRL+S. Prikazaće se dijalog New File Name. U ovom dijalog idite do foldera gde želite da snimate fajl. U polju File Name izmenite ime fajla koji želite da snimate. Kliknite na Save dugme.

3.10.9 Zatvaranje fajla



Slika 3.40

Uverite se da je prozor koji sadrži fajl koji želite da snimate aktivan. Odaberite File>Close iz padajućeg menija ili desnim klikom pritisnite odeljak fajla koji želite da zatvorite u Code Editor-u. Ako je fajl bio promenjen od momenta poslednjeg snimanja biće vam ponuđeno da snimate promene.

3.11 Compilation (*Kompajliranje)



Slika 3.41

Kada ste kreirali projekt i napisali izvorni kod, biće potrebno da ga kompajlirate. Odaberite Project>Build iz padajućeg menija ili kliknite na Build ikonicu, ili jednostavno pritisnite CTRL+F9.

Pojaviće se progresna linija koja vas obaveštava o statusu kompajliranja. Ako postoje greške, bićete obavešteni u Error Window-u. Ako se ne pronađu greške, mikroC će generisati izlazne fajlove.

Izlazni fajlovi

Nakon uspešne kompilacije, mikroC će generisati izlazne fajlove u projektnom folderu (folder koji sadrži projektni fajl .ppc). Izlazni fajlovi su navedeni dole:

3.11.1 Intel HEX file (.hex)

HEX zabeleške u Intel-ovom stilu. Koristite ovaj fajl za programiranje PIC MCU-a.

3.11.2 Binary mikro Compiled Library (.mcl)

Binarna distribucija aplikacija koje mogu biti uključene u drugim projektima.

3.11.3 List File (.lst)

Pregled alokacije PIC-ove memorije: adrese instrukcija, registri, rutine,...itd.

3.11.4 Assembler File (.asm)

Programeru čitljiv asemblerski fajl sa simboličkim imenima, izvedenim iz List Fajla.

Asemblerski Pregled



Slika 3.42

Nakon kompajliranja vašeg programa u mikroC-u, možete kliknuti na View Assembly ikonicu ili odabrati Project>View Assembly iz padajućeg menija da pregledate generisani asemblerski kod (.asm fajl) u novom prozoru. Assembler je čitljiv programeru, sa simboličkim imenima. Sve fizičke adrese i druge informacije mogu biti nađene u Statistics odeljku ili u List fajlu (.lst).

Ako program nije kompajliran i ako nema asemblerskog fajla, pokretanje ove opcije će kompajlirati vaš kod i onda prikazati assembler.

3.12 Poruke o greškama

Poruke o Greškama

- Specifier needed (*Potreban specifikator)
- Invalid declarator (*Nevažeći deklarator)
- Expected '(' or identifier (*Očekivano '(' ili identifikator)
- Integer const expected (*Očekivana integer konstanta)
- Array dimension must be greater then 0 (*Dimenzija niza mora biti veća od 0)
- Local objects cannot be extern (*Lokalni objekti ne mogu biti vanjski)
- Declarator error (*Greška deklaratora)
- Bad storage class (*Loša klasa skladištenja)
- Arguments cannot be of void type (*Argumenti ne mogu biti tipa void)
- Specifier/qualifier list expected (*Očekivana lista specifikatora/kvalifikatora)
- Address must be greater then 0 (*Adresa mora biti veća od 0)
- Identifier redefined (*Identifikator redefinisani)
- Case out of switch (*Case naredba izvan dosega switch naredbe)
- Default label out of switch (*Default labela izvan dosega switch naredbe)
- Switch exp. Must evaluate to integral type (*switch mora porediti integer tip)
- Continue outside of loop (*Continue izvan petlje)
- Break outside of loop or switch (*Break naredba izvan petlje ili switch naredbe)
- Void func cannot return values (*Funkcija tipa void ne vraća vrednosti)
- Unreachable code (*Nedohvatljiv kod)
- Illegal expression with void (*Nevažeći izraz sa void)
- Left operand must be pointer (*Levi operand mora biti pokazivač)
- Function required (*Zahteva se funkcija)
- Too many chars (*Previše karaktera)
- Undefined struct (*Nedefinisana struct)

- Nonexistent field (*Nepostojeće polje)
- Aggregate init error (*Init greška celine)
- Incompatible types (*nekompatibilni tipovi)
- Identifier redefined (*identifikator redefinisan)
- Function definition not found (*Nije pronađena definicija funkcije)
- Signature does not match (*Potpis se ne slaže)
- Cannot generate code for expression (*Ne može se generisati kod za izraz)
- Too many initializers of subaggregate (*Previše inicijalizatora pod-celine)
- Nonexistent subaggregate (*Nepostojeća pod-celina)
- Stack Overflow: func call in complex expression (*Prelivanje stek-a: poziv funkcije u kompleksnom izrazu)
- Syntax Error: expected %s but %s found (*Sintaksna greška: očekivano %s a nađeno %s)
- Array element cannot be function (*Element niza ne može biti funkcija)
- Function cannot return array (*Funkcija ne može vratiti niz)
- Inconsistent storage class (*Nekonzistentna klasa skladištenja)
- Inconsistent type (*Nekonzistentan tip)
- %s tag redefined (*Redefinisan tag)
- Illegal typecast (*Nevažeće dodeljivanje tipova)
- %s is not valid identifier (*%s nije važeći identifikator)
- Invalid statement (*Nevažeća naredba)
- Constant expression required (*Zahtevan konstantni izraz)
- Internal error %s (*Interna greška %s)
- Too many arguments (*Previše argumenata)
- Not enough parameters (*Nedovoljan broj parametara)
- Invalid expression (*Nevažeći izraz)
- Identifier expected but %s found (*Očekivan identifikator ali pronađeno %s)
- Operator [%s] not applicable to this operand [%s] (*Operator [%s] nije primenljiv na ovaj operand [%s])
- Assigning to non-lvalue [%s] (*Dodeljivanje ne-lvrednosti)
- Cannot cast [%s] to [%s] (*Ne može se [%s] pretvoriti po tipu u [%s])
- Cannot assign [%s] to [%s] (*Ne može se [%s] dodeliti [%s])
- Lvalue required (*Zahteva se lvrednost)

- Pointer required (*Zahteva se pokazivač)
- Argument is out of range (*Argument izvan opsega)
- Undeclared identifier [%s] in expression (*Nedeklarisani identifikator u izrazu)
- Too many initializers (*Previše inicijalizatora)
- Cannot establish this baud rate at %s MHz clock (*Ne može se uspostaviti ovaj baud odnos pri %s MHz kloku.

3.12.1 Compiler Warning Messages (*Kompajlerove Poruke Upozorenja)

- Highly inefficient code – func call in complex expression (*Visoko neefikasan kod – funkcijski poziv u kompleksnom izrazu)
- Inefficient code – func call in complex expression (*Neefikasan kod – funkcijski poziv u kompleksnom izrazu)

4 OSNOVNE KARAKTERISTIKE MIKROC JEZIKA

Jezik C nudi moć i fleksibilnost bez premca u programiranju mikrokontrolera. MikroC čak dodaje još snage nizom biblioteka specijalizovanih za PIC HW module i komunikacije. Ovo poglavlje bi trebalo da vam pomogne da naučite ili da se prisjetite C sintakse, zajedno sa specifičnostima programiranja PIC mikrokontrolera. Ako imate iskustva u C programiranju verovatno ćete prvo želeći da pogledate mikroC Specifičnosti.

4.1 Pic specifičnosti

Sa ciljem dobijanja maksimuma od vašeg mikroC kompajlera, trebali bi se upoznati sa određenim aspektima PIC MCU-a. Ovo znanje nije od esencijalne važnosti ali vam može pružiti bolje razumevanje mogućnosti i ograničenja PIC-a, kao i o njihovom uticaju na pisanje koda.

4.1.1 Efikasnost tipova

Pre svega, trebali bi znati da je PIC-ova ALU, koja izvodi aritmetičke operacije, optimizovana za rad sa bajtovima. Iako je mikroC sposoban za rad sa veoma kompleksnim tipovima podataka, oni mogu da predstavljaju problem za PIC – pogotovo ako radite na nekim od starijih modela. Ovo može dramatično da produži vreme protrebno za izvođenje čak i nejjednostavnijih operacija. Opšti savet je da koristite najmanji mogući tip u svakoj situaciji. Ovo važi u programiranju uopšteno ali posebno u radu sa mikrokontrolerima.

Svi PIC mikroi nisu jednakih performansi po pitanju izračunavanja. Na primer porodica PIC16 nema hardverske resurse za množenje dva bajta pa se to kompenzira softverskim algoritmom. Sa druge strane, porodica PIC18 im hardverski množač što za rezultat ima da se množenja odvijaju značajno brže.

4.1.2 Ograničenja Ugnježdenih Poziva

Ugnježdeni pozivi predstavljaju pozive funkcija unutar tela funkcije, ili samih sebe (rekurzivni pozivi) ili drugih funkcija. Rekurzivni funkcijski pozivi su podržani i mikroC-u ali sa ograničenjima. Rekurzivni funkcijski pozivi ne mogu sadržavati nikakve funkcijske parametre i lokalne promenljive zbog memorijskih i stek ograničenja kod PIC-a.

MikroC ograničava broj ne-rekurzivnih ugnježdenih poziva na:

- 8 poziva za PIC 12 familiju
- 8 Poziva za PIC 16 familiju
- 31 poziv za PIC 18 familiju

Broj dozvoljenih ugnježenih poziva se smanjuje za jedan ako koristite bilo koji od sledećih operatora u kodu: **/%*. Ovaj broj se dalje smanjuje ako koristite prekide u programu. Broj smanjivanja je određen brojem funkcija pozvanih iz prekida. Pogledajte **ponovnu ulaznost funkcija**. Ako je prekoračen dozvoljeni broj ugnježenih poziva, kompajler će generisati stack overflow (*prekoračenje kapaciteta steka) grešku.

4.2 PIC16xxx Specifičnosti

4.2.1 Proboj Stranice

U aplikacijama namenjenim za PIC16, nijedna pojedinačna rutina ne bi smela preći jednu stranicu (2000 instrukcija). Ako rutina ne staje u okvir jedne stranice, povezač će prijaviti grešku. Ako se suočite sa ovim problemom možda bi trebalo da ponovo razmislite o dizajnu vaše aplikacije – probajte da razbijete rutinu na nekoliko delova itd.

4.2.2 Ograničenja Indirektnog Pristupa Kroz FSR

Pokazivači kod PIC16 su “bliski”: oni nose samo donjih 8 bitova adrese. Kompajler će automatski očistiti deveti bit po pokretanju, tako da će se pokazivači odnositi na stranice fajl registara (memorijske banke) 0 i 1. Da bi se pristupilo objektima u stranicama fajl registara 3 i 4 preko pokazivača, korisnik mora manuelno podesiti IRP, i vratiti ga na nulu posle operacije. Gore navedena pravila važe za bilo koji indirektni pristup: nizove, strukture unijska dodeljivanja, itd.

Primedba: Veoma je važno ispravno se pobrinuti za IRP ako planirate ovakav pristup. Ako nađete da ovakav metod nije odgovarajući zbog previše promenljivih, trebalo bi da razmislite o prelasku na PIC18.

Primedba: ako imate previše promenljivih u kodu, probajte da ih preuredite pomoću povezačeve directive *absolute*. Promenljive kojima se pristupa samo direktno bi trebalo da se prebace u stranice fajl registara 3 i 4 u cilju povećanja efikasnosti.

4.3 mikroC specifičnosti

4.3.1 Pitanja ANSI Standarda

4.3.2 Odstupanje od ANSI C Standarda

MikroC odstupa od ANSI C Standarda u nekoliko oblasti. Neke od ovih modifikacija su poboljšanja uvedena sa namerom olakšavanja PIC programiranja, dok su druge rezultat hardverskih ograničenja PICmikro-a:

Rekurzivne funkcije su podržane sa ograničenjima zbog nepostojanja steka lakog za korištenje i ograničene memorije. Pogledajte PIC specifičnosti.

Pokazivači na promenljive i pokazivači na konstante nisu kompatibilni, npr: nikakvo dodeljivanje ili poređenje između njih nije moguće.

MikroC tretira identifikatore deklarisanе sa `const` kvalifikatorom kao “prave konstante” (u C++ stilu). Ovo dozvoljava korištenje `const` objekata na mestima gde bi ANSI C očekivao konstantni izraz. Ako vam je cilj portabilnost (*mogućnost izvršavanja koda na više različitih sistema), koristite tradicionalne pred-procesorski definisane konstante. Pogledajte Kvalifikatori Tipova i Konstante.

MikroC dozvoljava jednolinijske komentare u C++ stilu korištenjem dve kose crte (`//`).

U toku je izrada sledećih svojstava: anonimne strukture i unije.

4.3.3 Ponašanje definisano implementacijom

Pojedini delovi ANSI standarda imaju ponašanje definisano implementacijom. Ovo znači da tačno ponašanje nekog C koda može varirati od kompajlera do kompajlera. U help-u se nalaze poglavlja koja opisuju kako se mikroC kompajler ponaša u takvim situacijama. Najvažnije specifičnosti obuhvataju: Tipove sa Pokretnim zarezom, Klase Skladištenja, i Bit Polja.

4.3.4 Predefinisane Globalne i Konstante

U cilju omogućavanja PIC programiranja mikroC je implementirao određen broj predefinisanih globalnih i konstanti.

Svi PIC-ovi SFR su implicitno deklarirani kao globalne promenljive tipa *volatile unsigned short*. Ovi identifikatori imaju eksterno povezivanje i vidljivi su iz celog projekta. MikroC pri kreiranju projekta uključuje odgovarajući .def fajl koji sadrži deklaracije dostupnih SFR i konstanti (npr: TOIE, INTF, itd...). Svi identifikatori su napisani velikim slovima, identično nomenklaturi koja se koristi u Microchip-ovim dokumentacijama. Informacije o kompletnom skupu predefinisanih globalnih i konstanti možete naći u “Defs” u vašem mikroC instalacionom folderu, ili pokrenite Code Assistant za specifična slova (Ctrl + Space u Editoru).

4.3.5 Pristupanje Pojedinačnim Bitovima

MikroC vam dozvoljava da pristupite pojedinačnim bitovima 8-bitne promenljive, tipa *char* i *unsigned short*. Jednostavno koristite direktni selektor člana (.) sa promenljivom, praćen jednim od identifikatora F0, F1,...,F7. Npr:

`//Ako je RB0 postavljen, postavi RC0:`

```
if (PORTB.F0) PORTC.F0 = 1;
```

Nema potrebe za bilo kakvim specijalnim deklaracijama; ovaj način selektivnog pristupa je ugrađena osobina mikroC-a i može se koristiti bilo gde u kodu. Identifikatori F0-F7 se mogu pisati i malim i velikim slovima i imaju svoje sopstveno mesto za imena.

U slučaju da ste dobro upoznati sa nekim određenim čipom, možete pristupiti bitovima i po imenu:

```
INTCON.TMR0F = 0; // Obriši TMR0F
```

Pogledajte Predefinisane Globalne I Konstante za više informacija o imenima registara/bitova.

Primedba: Ako vam je bitna portabilnost izbegavajte ovakav način pristupa pojedinačnim bitovima I umesto toga koristite bit-polja.

4.3.6 Prekidi

Prekidima se može lako upravljati služeći se rezervisanom reči interrupt. MikroC implicitno deklarira funkciju interrupt (*prekid) koja ne može biti re-deklarirana. Njen prototip je:

void interrupt (**void**);

Napišite vašu sopstvenu definiciju (telo funkcije) za upravljanje prekidima u vašoj aplikaciji. MikroC snima sledeće SFR u stek kada ulazi u prekid i izbacuje ih nazad po povratku:

PIC12 i PIC16 porodice: W, STATUS, FSR, PCLATH

PIC18 porodica: FSR (brzi kontekst se koristi za snimanje WREG, STATUS, BSR)

Primedba: mikroC ne podržava prekide niskog prioriteta; za PIC18 porodicu prekidi moraju biti visokog prioriteta.

4.3.7 Funkcijski Pozivi iz Prekida

Pozivanje funkcija iz interrupt() rutine je sada moguće. Kompajler se brine o registrima koji se koriste, istovremeno i u „prekidnoj“ i u „glavnoj“ niti, i izvodi „pametno“ zamenjivanje konteksta između njih, snimajući samo registre koji se koriste u obe niti. Proverite **ponovnu ulaznost funkcija**.

Ovde dajemo jedan jednostavan primer upravljanja prekidima iz TMR0 (ako nikakvi drugi prekidi nisu dozvoljeni):

```
void interrupt() {  
    counter++;  
    TMR0=96;  
    INTCON=$20;  
}
```

U slučaju da su omogućeni višestruki prekidi, potrebno je da proverite koji od prekida se dogodio i onda da nastavite sa odgovarajućim kodom (upravljanjem prekidima).

4.3.8 Direktive Povezivača

MikroC koristi interni algoritam za raspoređivanje objekata u memoriji. Ako vam je potrebno da imate promenljivu ili rutinu na specifičnoj predefinisanoj adresi, koristite direktive *absolute* i *org*.

4.3.9 Direktiva absolute

Direktiva *absolute* određuje početnu adresu u RAM-u za promenljivu. Ako je promenljiva više-bajtovska, viši bajtovi se skladište na uzastopnim lokacijama. Direktiva *absolute* se dodaje deklaraciji promenljive:

```
int foo absolute 0x23;
```

```
//Promenljiva će zauzeti 2 bajta sa adresama 0x23 i 0x24;
```

Direktivu *absolute* koristite sa oprezom, pošto se može desiti da greškom dođe do preklapanja promenljivih. Na primer:

```
char i absolute 0x33;
```

```
//Promenljiva i će zauzeti jedan bajt sa adresom 0x33
```

```
long jiji absolute 0x30;
```

```
//Promenljiva će zauzeti 4 bajta sa adresama 0x30, 0x31, 0x32, 0x33,
```

```
//tako da će promena i istovremeno promeniti najviši bajt jiji istovremeno
```

4.3.10 Direktiva org

Direktiva *org* određuje početnu adresu rutine u ROM-u.

Direktiva *org* se dodaje definiciji funkcije. Direktive primenjene na ne-definišuće deklaracije će se ignorisati, s time da će poveziivač izdati odgovarajuće upozorenje. Direktiva *org* ne može biti primenjena na prekidnu rutinu.

Sledi jednostavan primer:

```
void func(char par) org 0x200 {
```

```
    //Funkcija će započeti na adresi 0x200
```

```
    nop;
```

```
}
```

4.3.11 Optimizacija koda

Optimizator je ugrđen sa ciljem proširivanja upotrebljivosti kompajlera, smanjivanja količine koda koji se generiše u ubrzanja njegovog izvođenja. Glavne odlike su:

4.3.12 Sklapanje konstanti

Svi izrazi koji mogu biti izračunati u vreme kompajliranja (npr. konstante) se zamenjuju njihovim rezultatom;

4.3.13 Propagacija konstanti

Kada se konstantna vrednost dodeli određenoj promenljivoj, kompajler to prepoznaje i zamenjuje korišćenje promenljive u kodu koji sledi konstantom, onoliko dugo dok vrednost promenljive ostane nepromenjena.

4.3.14 Propagacija kopije

Kompajler prepoznaje da dve promenljive imaju istu vrednost i eliminiše jednu od njih u kodu koji sledi.

4.3.15 Numeracija vrednosti

Kompajler prepoznaje da dva izraza daju isti rezultat i stoga može eliminisati celokupan izračun za jednog od njih.

4.3.16 Eliminacija „Mrtvog Koda“

Delovi koda koji se ne koriste drugde u programu ne utiču na konačni rezultat aplikacije. Oni se automatski odstranjuju.

4.3.17 Alokacija steka

Privremeni registri („Stekovi“) se koriste racionalnije, dozvoljavajući izračunavanje VEOMA kompleksnih izraza uz minimum potrošnje stek-a.

4.3.18 Optimizacija lokalnih promenljivih

Ne koriste se lokalne promenljive ako njihov rezultat ne utiče na neku od globalnih ili promenljivih tipa volatile.

4.3.19 Bolje generisanje koda i lokalna optimizacija

Generisanje koda je konzistentnije, a mnogo pažnje je posvećeno implementaciji specifičnih rešenja za „građevne elemente“ koda, sa ciljem daljnjeg smanjenja obima izlaznog koda.

4.3.20 Indirektni Funkcijski Pozivi

Ako se poveziivač susretne sa indirektnim funkcijskim pozivom (pokazivačem na funkciju), on pretpostavlja da bilo koja od funkcija, čije su adrese uzete bilo gde u programu, može biti pozvana u tom momentu. Koristite *#pragma funcall* direktivu da naložite poveziivaču koje funkcije mogu biti pozvane indirektno iz trenutne funkcije:

```
#pragma funcall <ime_funk><pozivana_funk>[,<pozivana_funk>,...]
```

Korespondirajuća pragma mora biti smeštena u izvorni modul gde je ime_funk implementirana. Ovaj modul takođe mora uključivati deklaracije svih funkcija pobrojanih u pozivana_funk listi.

Sve funkcije pobrojane u pozivana_funk listi će biti povezane ako se u kodu pozove funkcija ime_funk, bez obzira na to da li je bilo koja od njih pozvana ili ne.

Primedba: #pragma funcall direktiva može pomoći poveziivaču da optimizuje alokaciju funkcijskih okvira u kompajliranom steku.

4.4 Leksički elementi

Ove teme pružaju formalnu definiciju mikroC leksičkih elemenata. One opisuju različite kategorije jedinica sličnih rečima (tokena) koje jezik prepoznaje.

U fazi kompilacije gde se određuju tokeni, izvorni kod se analizira – razgrađuje na tokene i prazna polja. Tokeni u mikroC-u se izvode iz serije operacija koje kompajler i njegov ugrađeni pred-procesor izvode nad vašim programom. MikroC program započinje život kao niz ASCII karaktera koji predstavljaju izvorni kod, kreiran preko tastature, korištenjem prikladnog tekst editora (kao što je mikroC editor). Osnovna programska jedinica u mikroC-u je fajl. Ona obično odgovara imenovanom fajlu koji se nalazi u RAM-u ili na disku, sa ekstenzijom .c.

4.4.1 Prazan proctor

Prazan prostor je kolektivno ime koje se daje za razmak (blanko znak), horizontalne i vertikalne tabulatore, karaktere koji označavaju novu liniju, i komentare. Prazan prostor može služiti da naznači gde tokeni počinju i završavaju ali mimo ove funkcije, sav dodatni prazan prostor se odbacuje. Na primer, dve sekvence

```
int i; float f;
```

```
i
```

```
int i;
```

```
float f;
```

su leksički ekvivalentne i obrađuju se identično, davajući 6 tokena.

ASCII karakteri koji predstavljaju prazan prostor se mogu pojaviti u okviru znakovnih stringova, u kojem slučaju su zaštićeni od normalnog proces obrade (ostaju deo stringa).

4.4.2 Komentari

Komentari su delovi teksta koji služe za označavanje u programu i tehnički su jedna forma praznog prostora. Komentari služe samo programeru; oni se odbacuju od izvornog koda pre obrade. Postoje dva načina za označavanje komentara: C metod i C++ metod. Oba ova metoda su podržana od strane mikroC-a.

4.4.3 C komentari

C komentar je bilo koji niz znakova smešten iza para simbola `/*`. Komentar se završava prvim pojavljivanjem para `*/` koji prati početnih `/*`. Ceo niz, uključujući četiri simbola koji ograničavaju komentar, se zamenjuje jednim razmakom posle makro proširenja.

U mikroC-u,

```
int /* tip */ i /*identifikator*/;
```

se obrađuje kao:

```
int i;
```

Obratite pažnju da mikroC ne podržava ne-portabilnu strategiju dodavanja tokena korištenjem `/**/`. Za više informacija o dodavanju tokena pogledajte teme o Pred-procesoru.

4.4.4 C++ komentari

MikroC dozvoljava komentare u jednoj liniji korištenjem dve kose crte (`//`). Komentar može započeti na bilo kojem mestu i traje do sledećeg novog reda. Sledeći kod,

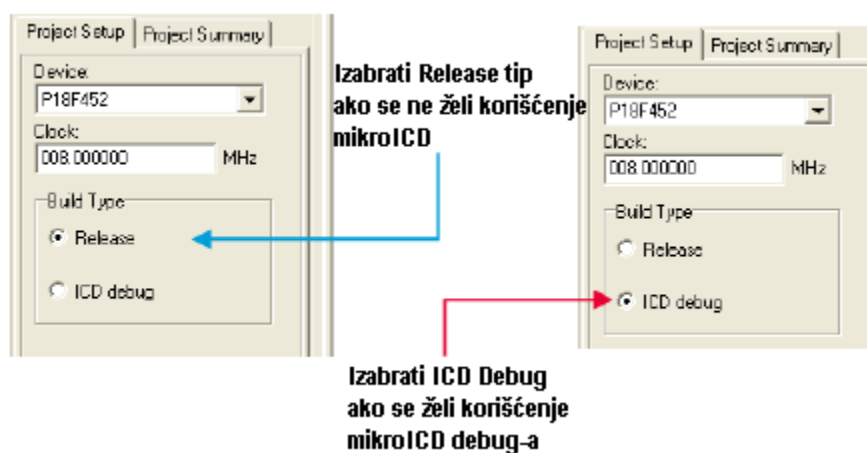
```
int i; //ovo je komentar  
int j;  
se obrađuje kao:  
int i;  
int j;
```

4.5 Mikro ICD (Debugger na nivou kola)

Mikro ICD je visoko efikasan alat za debugovanje u stvarnom vremenu na hardverskom nivou. ICD debugger vam omogućava da izvršite mikroC program na prijemnom PIC mikrokontroleru i vidite vrednosti promenljivih, Specijalnih Funkcijskih Registara (SFR), memorije i EEPROM-a dok je program u toku.

Ako imate odgovarajući hardver i softver za korišćenje mikro ICD-a, po završetku pisanja vašeg programa treba da se odlučite za **Release build type** (*način kompajliranja za gotovo izdanje) ili **ICD Debug build type**(*način kompajliranja ICD Debug).

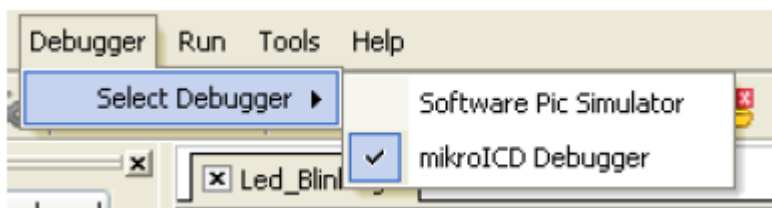
Korak br.1



Slika 4.1

Korak br.2

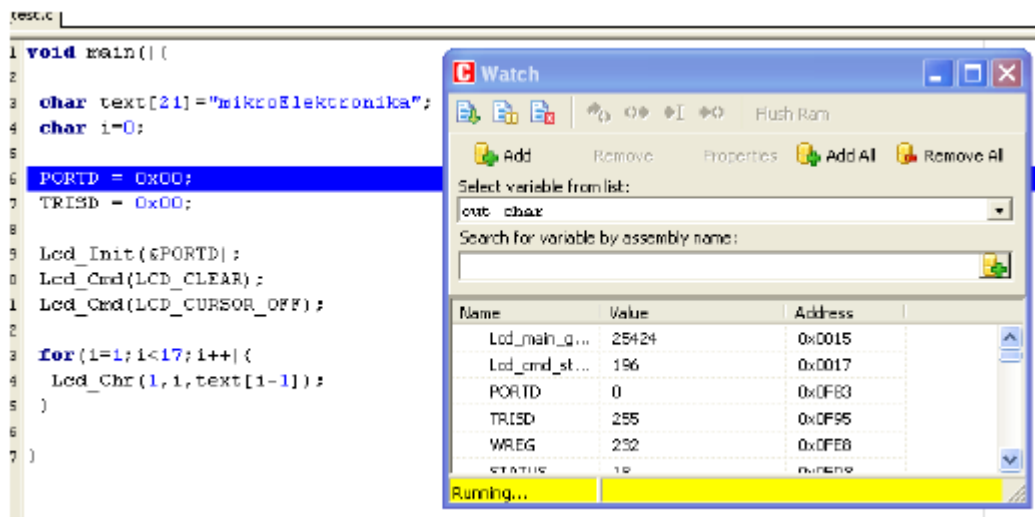
Pošto odaberete ICD Debug Build Type, vreme je da kompajlirate vaš projekat. Posle uspešne kompilacije morate programirati PIC korištenjem F11 prečice, a potom treba da odaberete mikro ICD preko **Debugger >Select Debugger >mikroICD Debugger** iz padajućeg menija.



Slika 4.2

Korak br.3

Možete pokrenuti mikro ICD odabirom **Run> Debug** iz padajućeg menija, ili klikom na Debug ikonicu. Pokretanjem Debuggera postaje dostupno više opcija: Step Into, Step Over, Run to Cursor, itd. Linija koja treba da se izvrši je naznačena bojom (plavom po podrazumevanim podešavanjima). Takođe je prisutno obaveštenje o izvršavanju programa i ono se može naći u Watch Window-u (žuta statusna linija). Obratite pažnju da nekim funkcijama treba vreme da se izvrše tako da je odvijanje programa naznačeno u Watch Window-u.



Slika 4.3 - Watch prozor

4.5.1 Opcije mikro ICD Debuggera

Naziv	Opis	Funkcijska tipka
Debug	Pokreće Debugger.	[F9]
Run/Pause Debugger	Izvršava ili pauzira Debugger.	[F6]
Toggle Breakpoints	Uključuje/isključuje tačku prekida na trenutnoj poziciji kurzora. Za pregled svih tačaka prekida, odaberite Run	[F5]

PROGRAMIRANJE MIKROKONTROLERA

	>View Breakpoints iz padajućeg menija. Dupli klik na stavku u prozoru liste locira tačku prekida.	
Run to cursor	Izvršava sve instrukcije između trenutne instrukcije i pozicije kurzora.	[F4]
Step Into	Izvršava trenutnu C instrukciju(jedno ili više-klokovsku), i potom se zaustavlja. Ako je instrukcija poziv rutine, ulazi u rutinu i zaustavlja se na prvoj funkciji koja sledi poziv.	[F7]
Step Over	Izvršava trenutnu C instrukciju(jedno ili više-klokovsku), i potom se zaustavlja. Ako je instrukcija poziv rutinepreskače je i zaustavlja se na prvoj funkciji koja sledi poziv.	[F8]
Flush RAM	Puni trenutni PIC RAM. Sve promenljive će biti promenjene u skladu sa vrednostima u Watch Window-u	Nema

4.5.2 Mikro ICD Debugger Primer

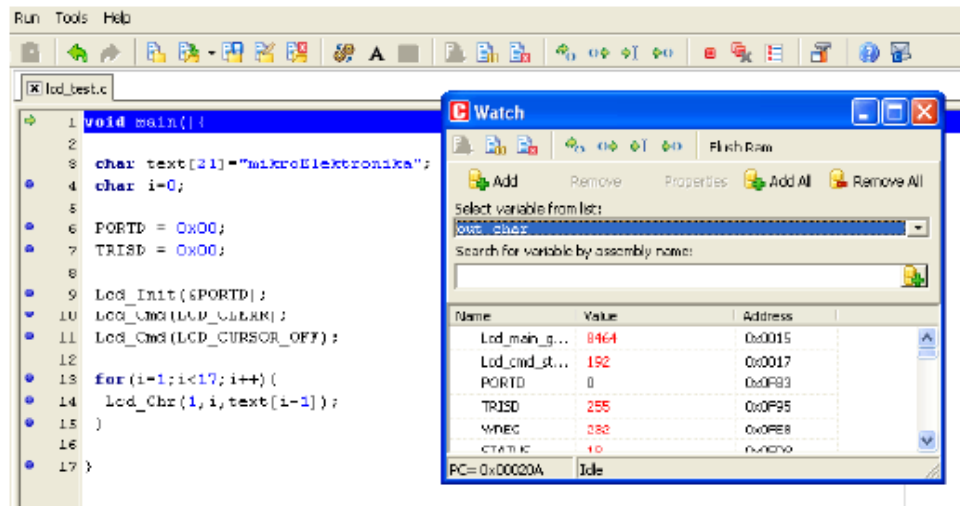
Korak br.1

Ovde dajemo primer korištenja ICD Debuggera, korak po korak. Prvo morate da napišete program. Pokazaćemo kako mikro ICD radi korištenjem sledećeg primera:

```
void main(){  
    char text[21]="mikroElektronika";  
    char i=0;  
    PORTD=0x00;  
    TRISD=0x00;  
    Lcd_init(&PORTD);  
    Lcd_Cmd(LCD_CLEAR);  
    Lcd_Cmd(LCD_CURSOR_OFF);  
    for (i=1;i<17;i++){  
        Lcd_Chr(1,i,text[i-1]);  
    }  
}
```


Korak br.2

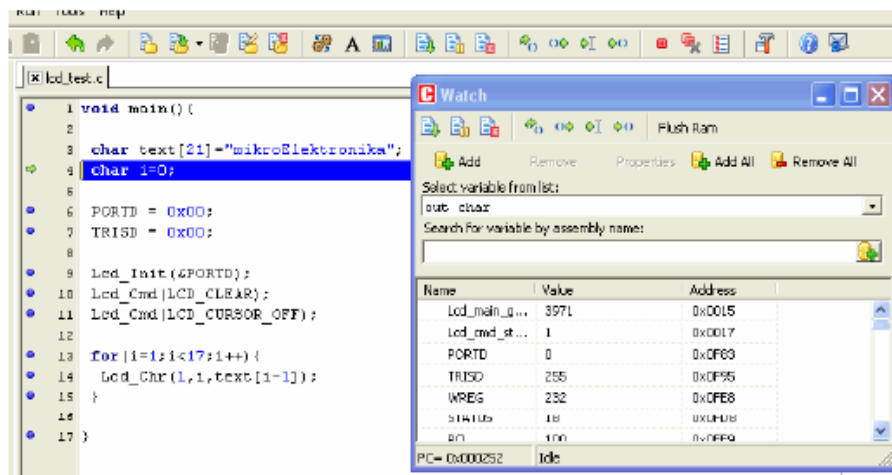
Po uspešnom kompajliranju i PIC programiranju pritisnite F9 da pokrenete mikro ICD. Nakon inicijalizacije mikro ICD-a bi trebalo da se pojavi plava aktivna linija.



Slika 4.4 - Watch prozor

Korak br.3

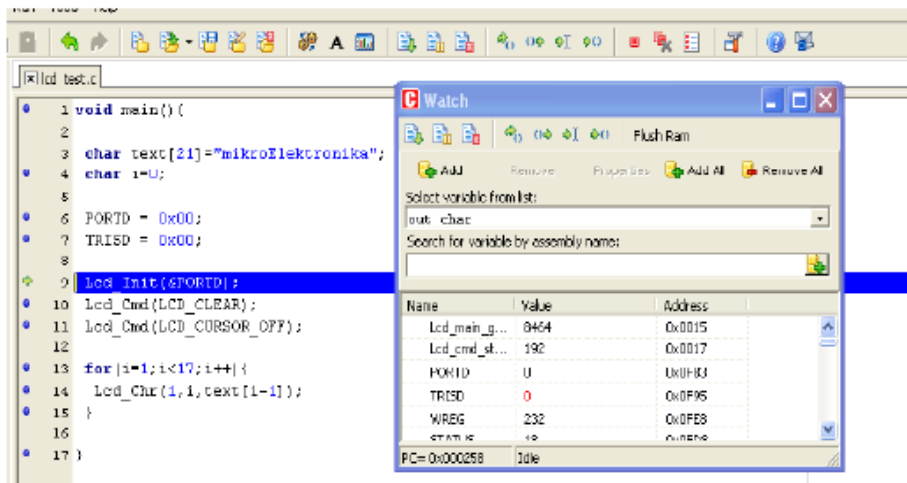
Izvršićemo inspekciju koda liniju po liniju. Pritiskanjem F8 izvršavamo kod liniju po liniju. Ne preporučuje se korištenje Step Into [F7] i Step Over [F8] preko Delays(*Odlaganje) rutina i rutina koja sadrže odlaganja. Umesto njih koristite Run to cursor [F4] i Breakpoints funkcije.



Slika 4.5 - Inspekciju koda liniju po liniju.

PROGRAMIRANJE MIKROKONTROLERA

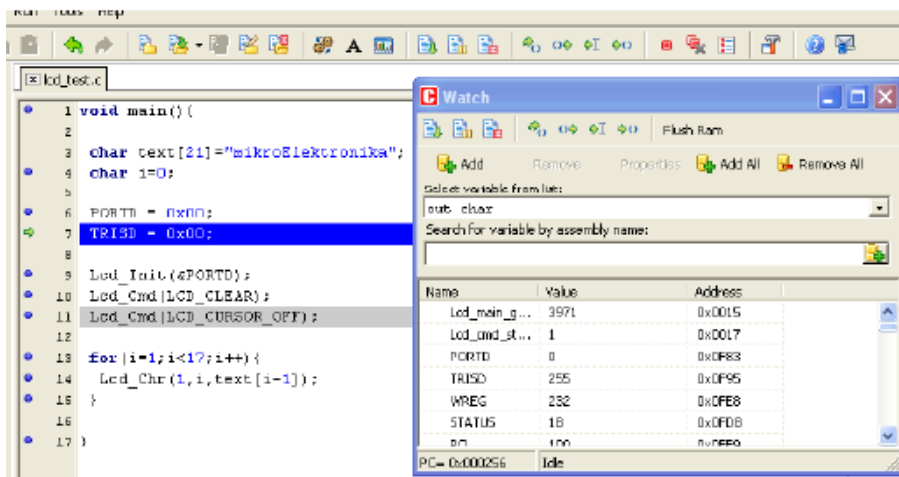
Sve promene se čitaju iz PIC-a i unose u Watch Window. Primetite da je **TRISD** promenio vrednost sa 255 na 0.



Slika 4.6 - Status posle promene TRISD promenljive

Korak br.4

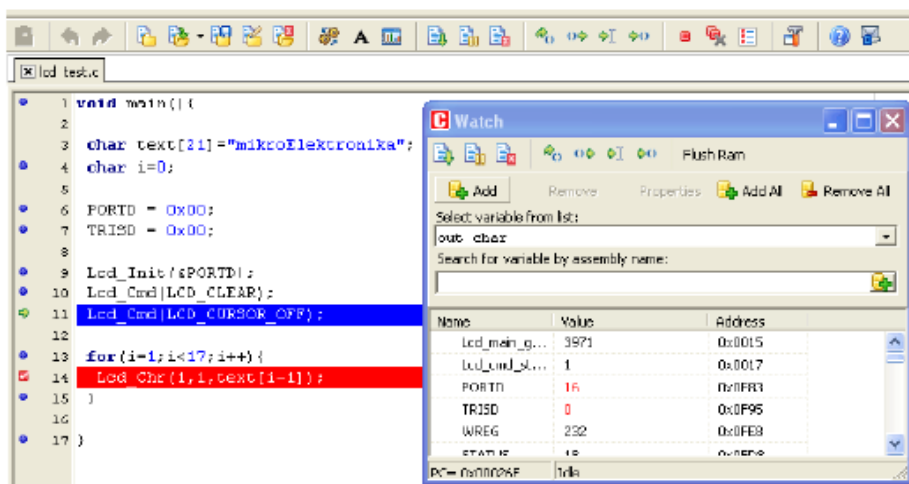
Step Into [F7] i Step Over [F8] su funkcije mikro ICD debuggera koje se koriste u „stepping“ modu (*rad po koracima). Mikro ICD takođe podržava Real-Time mod (*rad u stvarnom vremenu). Funkcije koje se koriste pri radu u stvarnom vremenu su Run/Pause Debugger [F6] i Run to cursor [F4]. Pritiskom na F4 se ide na liniju koju je odabrao korisnik. Sve što korisnik treba da uradi je da odabere liniju kurzorom pa da pritisne F4, što će dovesti do izvršavanja koda dok se ne dođe do odabrane linije.



Slika 4.7 - Izvršavanje koda do određene tačke

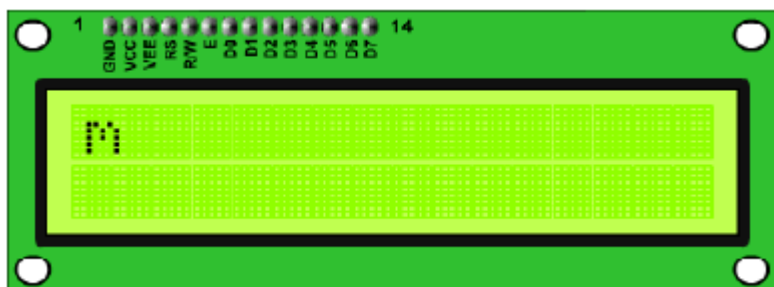
Korak br.5

Run(Pause) Debugger [F6] i Toggle Breakpoints [F5] su funkcije mikro ICD debugger-a koje se koriste u modu stvarnog vremena. Pritiskanje **F5** označava liniju koju je odabrao korisnik kao prekidnu tačku. **F6** izvršava kod do nailaska na prekidnu tačku. Debugger se zaustavlja po nailasku na prekidnu tačku. U našem primeru ćemo koristiti prekidne tačke za pisanje „mikroElektronika“ na LCD-u, znak po znak. Prekidna tačka se postavlja na LCD_Chr i program će zastati svaki put kada stigne do ove funkcije. Po dostizanju prekidne tačke moramo opet pritisnuti **F6** da bi nastavili sa izvršavanjem programa.



Slika 4.8 - Upotreba prekidnih tačaka

Prekidne tačke su podeljene u dve grupe. Postoje hardverske i softverske prekidne tačke. Hardverske prekidne tačke su smeštene u PIC i one omogućavaju najbrže debugovanje. Broj hardverskih prekidnih tačaka je ograničen (1 za PIC16 i 3 za PIC18). Ako se iskoriste sve hardverske prekidne tačke, sledeća prekidna tačka koja se koristi će biti softverska prekidna tačka. Ove prekidne tačke se pamte unutar mikro ICD-a i one simuliraju hardverske prekidne tačke. Softverske prekidne tačke su značajno sporije u poređenju sa hardverskim. Ove razlike u hardveru i softveru nisu vidljive u mikro ICD softveru ali njihova različita vremena su prilično izražena, tako da je važno znati da postoje dva tipa prekidnih tačaka.



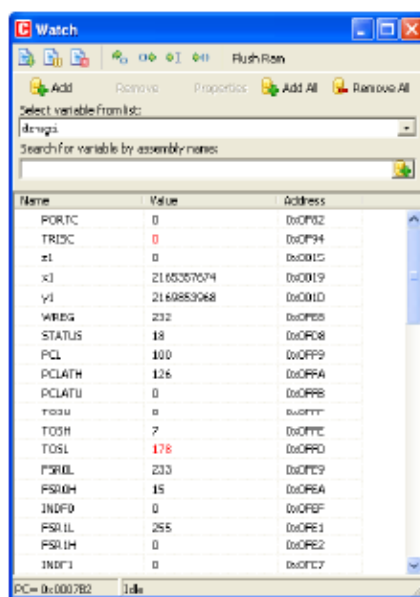
Slika 4.9 - Izgled displeja

4.6 Mikro ICD (Debugger na nivou kola)

4.6.1 Watch Window (*Prozor Pregleda)

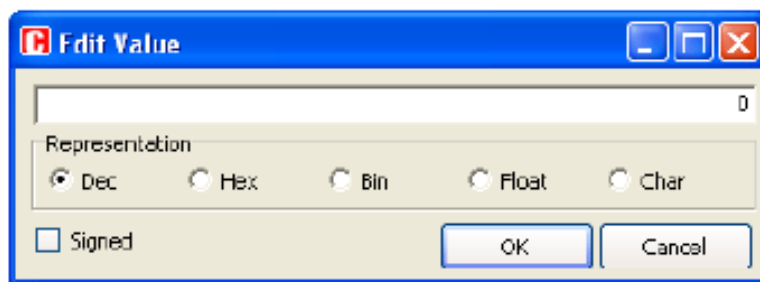
Debuggerov Watch Window je glavni prozor debugera koji vam omogućava da nadzirete programske stavke tokom izvršavanja programa. Da prikazete Watch Window, odaberite **View>Debug Windows>Watch Window** iz padajućeg menija.

Watch Window prikazuje promenljive i registre PIC-a, sa njihovim adresama i vrednostima. Vrednosti se osvežavaju kako idete kroz simulaciju. Koristite padajući meni da dodate i odstranite stavke koje želite da nadzirete. Stavke koje su se nedavno promenile su obojene crvenom bojom.



Slika 4.10 - Watch prozor

Dupli klik na stavku otvara prozor Edit Value (*uredite vrednost) u kojem možete dodeliti novu vrednost odabranoj promenljivoj ili registru. Takođe možete promeniti način prezentacije na binarni, hex, char, ili decimalni, za odabranu stavku.

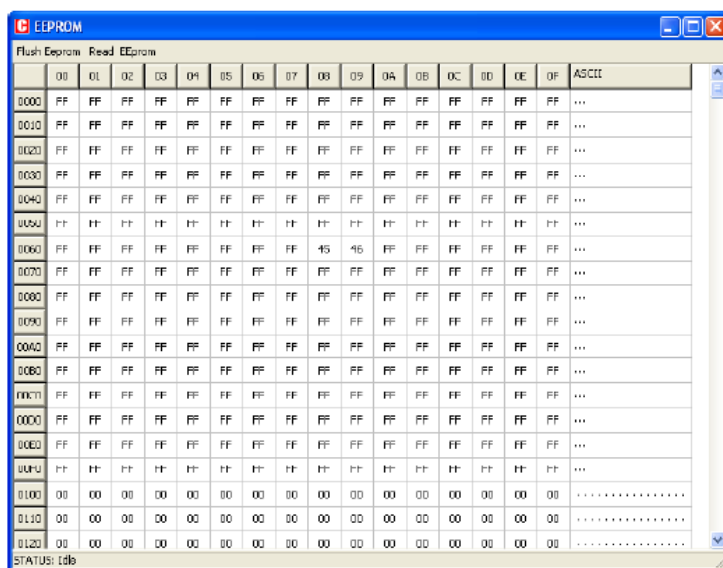


Slika 4.11 - Prozor za učitavanje vrednosti

4.6.2 View EEPROM Window (*Prozor Pregleda EEPROM-a)

Mikro ICD EEPROM prozoru se pristupa iz padajućeg menija odabirom **View>Debug Windows>View EEPROM**.

EEPROM prozor prikazuje trenutne vrednosti snimljene u PIC-ovoj unutrašnjoj EEPROM memoriji. Postoje dva operaciona dugmeta koja se tiču EEPROM Watch Window-a – **Flush EEPROM** (*Napuni EEPROM) i **Read EEPROM** (*Čitaj EEPROM). **Flush EEPROM** upisuje podatke iz EEPROM prozora u PIC-ovu unutrašnju EEPROM memoriju. **Read EEPROM** učitava podatke iz PIC-ove unutrašnje memorije u EEPROM prozor.



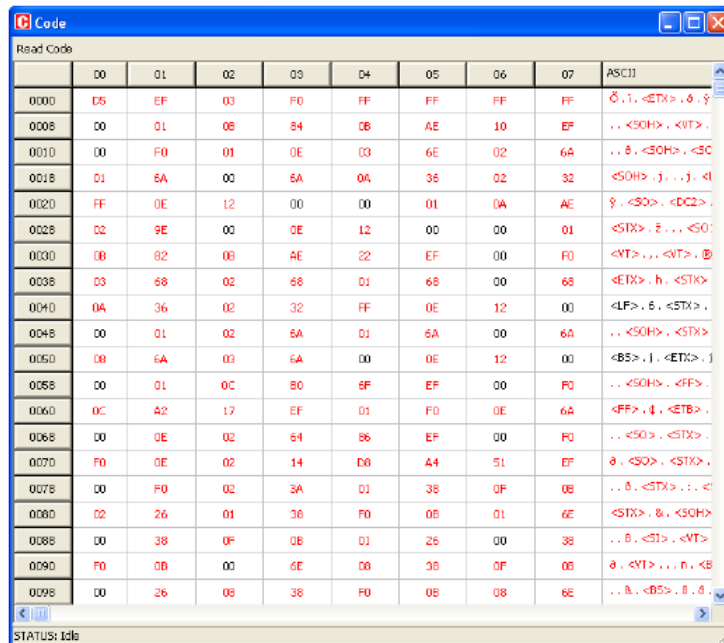
Slika 4.12 - Pregled sadržaja EEPROM memorije

4.6.3 View Code Window (*Prozor Pregleda Koda)

Mikro ICD View Code Window-u se pristupa iz padajućeg menija, odabirom **View>Debug Windows>View Code**.

View Code Window prikazuje kod (hex kod) koji je upisan u PIC. U vezi sa View Code Window-om stoji operaciono dugme **Read Code** (*Čitaj Kod). **Read Code** učitava kod iz PIC-a u View Code Window.

PROGRAMIRANJE MIKROKONTROLERA

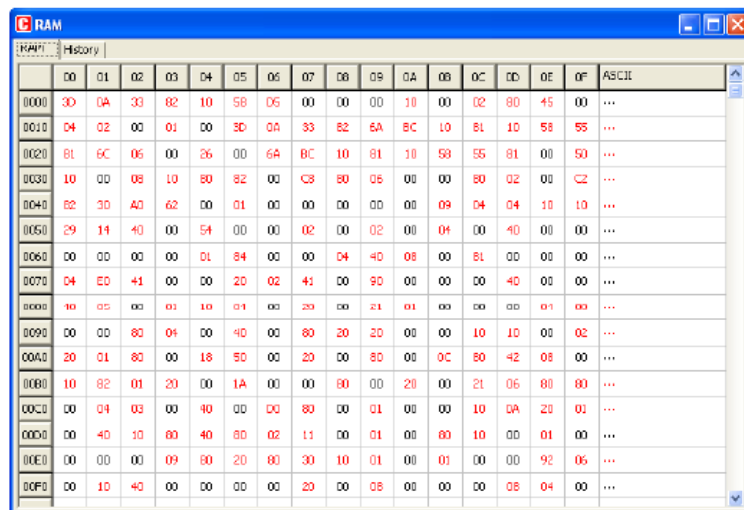


Slika 4.13 - Pregled koda

4.6.4 View RAM Window (*Prozor Pregleda RAM-a)

Debuggerovom View RAM Window-u se pristupa iz padajućeg menija odabirom **View>Debug Windows>View RAM**.

View RAM Window prikazuje mapu PIC-ovog RAM-a, gde su skorije promene naznačene crvenom bojom.



Slika 4.14 - Sadržaj RAM memorije

4.6.5 Uobičajene Greške

- Pokušaj programiranja PIC-a dok je mikro ICD aktivan
- Pokušaj debugovanja **Release** build Type verzije programa.
- Pokušaj debugovanja promenjenog programskog koda koji nije kompajliran i programiran u PIC.
- Pokušaj odabira prazne linije za Run to cursor [F4] i Toggle Breakpoints [F5] funkcije.

4.7 Tokeni

Token je najmanji element nekog C programa koji ima smisla za kompajler. Sintaksni analizator razdvaja tokene iz ulaznog toka kreiranjem najdužeg mogućeg tokena od ulaznih znakova, sa leve strane prema desnoj.

MikroC prepoznaje sledeće vrste tokena:

- Ključne reči
- Identifikatori
- Konstante
- Operatori
- Znaci Interpunkcije (takođe poznati kao separatori)

4.7.1 Primer Ekstrakcije Tokena

Ovde dajemo primer ekstrakcije tokena. Neka imamo sledeću kodnu sekvencu:

```
inter = a+++b;
```

Prvo treba da primetimo da će *inter* biti procesuirano kao jedani identifikator a ne ključna reč *int* praćena identifikatorom *er*.

Programer koji je napisao kod je možda nameravao da napiše

```
inter = a+ (++b);
```

ali time neće da postigne očekivani rezultat. Kompajler će to procesuirati kao niz od sledećih sedam tokena:

```
inter    //identifikator
=        //operator dodele
a        //identifikator
++       //operator post-inkrementiranja
+        //operator sabiranja
b        //identifikator
;        //separator tačka-zarez
```

Primetite da se +++ prepoznaje kao ++ (najduži mogući token) praćeno sa +.

4.8 Konstante

Konstante ili literali su tokeni koji predstavljaju fiksne numeričke ili znakovne vrednosti.

MikroC podržava:

- Celobrojne konstante (*integer),
- Konstante u pokretnom zarezu (*floating point),

- Znakovne konstante (*character),
- Konstante niza znakova (*string)(znakovni literali),
- Konstante pobrojavanja (*enumeration),

Kompajler utvrđuje pripadnost tipu podataka neke konstante korištenjem naznaka kao što su numerička vrednost i format korišten u izvornom kodu.

4.8.1 Celobrojne konstante

Celobrojne konstante mogu biti decimalne (osnove 10), heksdecimalne (osnove 16), binarne (osnove 2), ili oktalne (osnove 8). U odsustvu ikakvih predefinišućih sufiksa, tip podatka celobrojne konstante se određuje na osnovu njene vrednosti.

4.8.2 Sufiksi Long i Unsigned

Sufiks L (ili l) pridodan bilo kojoj konstanti čini da ona bude tretirana kao long (*dugačka). Slično tome, sufiks U (ili u) čini da konstanta bude unsigned (*neoznačena). Moguće je korištenje istovremeno oba sufiksa na istoj konstanti u bilo kojem redu: ul, Lu, UL, itd.

U nedostatku bilo kakvog sufiksa (U, u, L, ili l), konstanti se dodeljuje „najmanji“ od sledećih tipova koji mogu smestiti njenu vrednost: *short*, *unsigned short*, *int*, *unsigned int*, *long int*, *unsigned long int*.

U suprotnom slučaju.

Ako konstanta ima U ili u sufiks, njen tip podatka će biti prvi od sledećih koji mogu smestiti njenu vrednost: *unsigned short*, *unsigned int*, *unsigned long int*.

Ako konstanta ima L ili l sufiks, njen tip podatka će biti prvi od sledećih koji mogu smestiti njenu vrednost: *long int*, *unsigned long int*.

Ako konstanta ima istovremeno i U i L sufikse, (ul, lu, UL, lU, uL, Lu, LU, ili UL) njen tip podatka će biti *unsigned long int*.

4.8.3 Decimalne konstante

Dozvoljene su decimalne konstante od -2147483648 do 4294967295. Konstante koje premašuju ova ograničenja će proizvesti „Out of range“ (*Van opsega) grešku. Decimalne konstante ne smeju koristiti početnu nulu. Celobrojna konstanta koja ima početnu nulu se interpretira kao oktalna konstanta.

U odsustvu bilo kakvih predefinišućih sufiksa, tip podatka decimalne konstante se izvodi iz njene vrednosti, kao što je prikazano ispod:

```
< -2147483648      Error: Out of range!(*Greška: Van opsega!)
-2147483648 .. -32769      long
-32768 .. -129      int
-128 .. 127      short
128 .. 255      unsigned short
256 .. 32767      int
32768 .. 65535      unsigned int
65536 .. 2147483647      long
2147483648 .. 4294967295  unsigned long
```


> 4294967295 Error: Out of range!

4.8.4 Heksadecimalne Konstante

Sve konstante koje počinju sa 0x (ili 0X) se uzimaju kao heksadecimalne. U odsustvu bilo kakvih predefinišućih sufiksa, tip podatka heksadecimalne konstante se izvodi iz njene vrednosti, u skladu sa gore predstavljenim pravilima. Na primer, 0xC367 će biti tretirano kao *unsigned int*.

4.8.5 Binarne Konstante

Sve konstante koje počinju sa 0b (ili 0B) se uzimaju kao binarne. U odsustvu bilo kakvih predefinišućih sufiksa, tip podatka binarne konstante se izvodi iz njene vrednosti, u skladu sa gore predstavljenim pravilima. Na primer, 0b11101 će biti tretirano kao *short*.

4.8.6 Oktalne Konstante

Sve konstante sa početnom nulom se uzimaju kao oktalne. Ako oktalna konstanta sadrži nedozvoljene cifre 8 ili 9, prijavljuje se greška. U odsustvu bilo kakvih predefinišućih sufiksa, tip podatka oktalne konstante se izvodi iz njene vrednosti, u skladu sa gore predstavljenim pravilima. Na primer, 0777 će biti tretirano kao *int*.

4.8.7 Konstante u Pokretnom Zarezu

Konstanta u pokretnom zarezu se sastoji od:

- Decimalnog celobrojnog broja
- Decimalne tačke
- Decimalnog ostatka
- e ili E i označenog celobrojnog eksponenta (opcionarno)
- Sufiksa tipa: f ili F ili l ili L (opcionarno)

Možete izostaviti decimalni celobrojni broj ili decimalni ostatak, ali ne oboje. Možete izostaviti decimalnu tačku ili slovo e (ili E) i označeni celobrojni eksponent, ali ne oboje. Ova pravila omogućavaju i konvencionalni e naučni (eksponencijalni) način obeležavanja.

Negativne konstante u pokretnom zarezu se uzimaju kao pozitivne sa prefiksiranim unarnim operatorom minus (-).

MikroC ograničava opseg konstanti u pokretnom zarezu na:

$\pm 1.17549435082E38$.. $\pm 6.80564774407E38$.

Konstante u pokretnom zarezu u MikroC-u su tipa *double*. Obratite pažnju da mikroC-ova implementacija ANSI Standarda posmatra *float* i *double* (sa *long double* varijantom) kao da pripadaju istom tipu.

4.8.8 Znakovne Konstante

Znakovna konstanta je jedan ili više znakova ograđenih jednostrukim navodnicima, kao što su 'A', '+', ili '\n'. Konstante od jednog znaka u C-u su tipa *int*. Konstante sastavljene od više karaktera se nazivaju stringovi ili string literali. Za više informacija pogledajte String Konstante.

Sekvence Izlaza

PROGRAMIRANJE MIKROKONTROLERA

Znak kose crte unazad (\) se koristi da označi izlaznu sekvencu, koja dozvoljava vizuelnu prezentaciju određenih ne-grafičkih znakova. Jedna od najuobičajenijih izlaznih konstanti je znak za novi red (\n).

Kosa crta unazad se koristi sa oktalnim ili heksadecimalnim brojevima za predstavljanje ASCII simbola ili kontrolnog koda koji odgovara toj vrednosti; na primer '\x3F' za znak pitanja. Možete koristiti bilo koji string veličine do tri oktalna znaka ili bilo kojeg broja heksadecimalnih znakova, ako je ispunjen uslov da je vrednost u okviru dozvoljenog opsega za char tip podataka (0 do 0xFF za mikroC). Veći brojevi će generisati grešku kompajlera „numeric constant too large“ (*prevelika numerička konstanta).

Na primer, oktalni broj \777 je veći od maksimalne dozvoljene vrednosti (\377) i proizvešće grešku. Prvi ne-oktalni ili ne-heksadecimalni znak na koji se naiđe u jednoj oktalnoj ili heksadecimalnoj izlaznoj sekvenci označava kraj sekvence.

Primerba: Za predstavljanje ASCII kose crte unazad se koristi \\ kao u putanjama operativnih sistema.

Sledeća tabela prikazuje postojeće izlazne sekvence u mikroC-u:

Sekvenca	Vrednost	Char	Efekat
\a	0x07	BEL	Zvučni signal
\b	0x08	BS	Brisanje unazad
\f	0x0C	FF	Nova stranica
\n	0x0A	LF	Nova linija
\r	0x0D	CR	Skok u novi red
\t	0x09	HT	Horizontalni tabulator
\v	0x0B	VT	Vertikalni tabulator
\\	0x5C	\	Kosa crta unazad
\'	0x27	'	Navodni znak (jednostruki)
\"	0x22	„	Navodni znak (dvostruki)
\?	0x3F	?	Upitnik
\O		Bilo koji	O = oktalni string do tri oktalne cifre
\xH		Bilo koji	H = string heksadecimalnih cifara
\XH		Bilo koji	H = string heksadecimalnih cifara

4.8.9 String Konstante

String konstante, takođe poznate pod nazivom string literali, su specijalni tip konstanti koje čuvaju fiksne nizove znakova. Jedan string literal je niz proizvoljnih broja znakova uokvirenih dvostrukim navodnim znacima.

“Ovo je jedan string.”

Null string, ili prazan string se piše kao `""`. Jedan literalni string se interno pamti kao dati niz znakova plus jedan null znak na kraju. Null string se pamti kao jedan null znak.

Znakovi unutar dvostrukih navodnika mogu sadržavati izlazne sekvence npr:

`"\t" "Ime" "\t Adresa\n\n"`

Susedni string literali, odvojeni samo praznim znakovima, se sabiraju tokom obrade. Npr:

“Ovo je “ *samo*”

“jedan primer”

Je ekvivalentno sa

“Ovo je samo jedan primer”

4.8.10 Nastavljanje linije kosom crtom unazad

Kosu crtu unazad možete koristiti kao znak za nastavak da bi produžili string preko granica linije:

“Ovo je zapravo \

Jedno-linijski string”

4.8.11 Konstante pobrojanja

Konstante pobrojanja su identifikatori definisani u deklaracijama tipa *enum*. Ovi identifikatori su obično izabrani mnemonički u cilju povećanja čitljivosti. Konstante pobrojanja su *int* tipa. One se mogu koristiti u bilo kojem izrazu u kojem su celobrojne konstante dozvoljene.

Na primer:

enum dani { NED=0, PON, UTO, SRE, ČET, PET, SUB};

Identifikatori (numeratori) moraju biti korišteni unutar dosega *enum* deklaracije. Negativni inicijalizatori nisu dozvoljeni. Pogledajte Pobrojanja za detalje o *enum* deklaracijama.

4.8.12 Pokazivačke Konstante

Pokazivač ili objekt na koji se pokazuje može biti deklarisan sa *const* modifikatorom. Bilo šta što je deklarisan sa *const* ne može menjati svoju vrednost. Takođe nije dozvoljeno stvaranje pokazivača koji bi mogao narušiti nemogućnost dodele konstantnog objekta.

4.8.13 Konstantni Izrazi

Konstantni izraz je izraz koji je jednak konstanti i sadrži se samo od konstanti (literala) ili simboličkih konstanti. On se izračunava tokom kompajliranja i rezultat mora biti konstanta koja se nalazi u opsegu prikazivih vrednosti za svoj tip. Konstantni izrazi se izračunavaju na isti način kao i obični.

Konstantni izrazi se mogu sastojati samo od sledećih elemenata: literala, konstanti pobrojavanja, prostih konstanti (bez konstantnih nizova ili struktura), i sizeof operatora.

Konstantni izrazi ne smeju sadržavati sledeće operatore ukoliko se ti operatori ne nalaze kao operandi u okviru sizeof operatora: dodela, zarez, dekrementacija, poziv funkcije, inkrementacija.

Konstantni izraz možete koristiti na bilo kojem mestu gde je dozvoljena upotreba konstanti.

4.9 Ključne reči

Ključne reči su reči rezervisane za specijalne namene i ne smeju biti korištene kao normalna imena identifikatora.

Pored standardnih C ključnih reči, svi relevantni SFR su definisani kao globalne promenljive i predstavljaju rezervisane reči koje ne mogu biti redefinisane (na primer: TMR0, PCL, itd). Možete otkucati željena slova u Code Assistant-u (Ctrl+Space u Editoru) ili pogledati Unapred definisane Globalne i Konstante.

Ovo je abecedni spisak ključnih reči u C-u:

asm	enum	signed
auto	extern	sizeof
break	float	static
case	for	struct
char	goto	switch
const	if	typedef
continue	int	union
default	long	unsigned
do	register	void
double	return	volatile
else	short	while

Takođe, mikroC sadrži određeni broj unapred definisanih identifikatora u bibliotekama. Njih možete zameniti svojim sopstvenim definicijama, ako planirate razvoj svojih sopstvenih biblioteka. Za više informacija pogledajte mikroC Biblioteke.

4.10 Indetifikatori

Identifikatori su proizvoljna imena bilo koje dužine, data funkcijama, promenljivim, simboličkim konstantama, korisnički-definisanim tipovima podataka, i labelama. Sve ove programske elemente ćemo zvati objekti kroz help (i ovo ne treba mešati sa značenjem objekta u objektno-orijentisanom programiranju).

Identifikatori mogu biti sastavljeni od slova *a* do *z* i od *A* do *Z*, donjom linijom “_”, i cifara od 0 do 9. Jedino ograničenje je da prvi znak mora biti slovo ili donja linija.

4.10.1 Velika i Mala Slova

Kod mikroC identifikatora trenutno nije važno da li koristimo mala ili velika slova tako da *Sum*, *sum*, i *suM* predstavljaju iste identifikatore. Ipak, buduće verzije mikroC-a će nuditi opciju da se aktivira ili suspenduje razlika između malih i velikih slova. Trenutno jedini izuzetci su rezervisane reči *main* i *interrupt*, koje moraju biti napisane malim slovima.

4.10.2 Jedinstvenost i Doseg

Iako su imena identifikatora proizvoljna (u okviru gore pomenutih pravila), greške se događaju ako se isto ime koristi za više od jednog identifikatora u okviru istog desega koji dele isti prostor imena. Dupliranje imena je dozvoljeno za različite prostoje imena, bez obzira na pravila koja se tiču dosega. Za više informacija o dosegima pogledajte Doseg i Vidljivost.

4.11 Znaci interpunkcije

MikroC znaci interpunkcije (poznati i pod nazivom separatori), uključuje male, srednje i velike zagrade, zarez, tačku, tačku-zarez, dvostruku tačku, znak puta, znak jednako i znak „taraba“. Većina ovih znakova interpunkcije takođe služe i kao operatori.

4.11.1 Srednje Zagrade

Srednje zagrade [] označavaju indekse jedno i više-dimenzionih nizova:

```
char ch, str[] = "mikro";
int mat[3][4]; /* 3 x 4 matrica */
ch = str[3]; /* 4-ti element */
```

4.11.2 Male Zagrade

Male zagrade () se koriste za grupisanje izraza, izolaciju uslovnih izraza, i da ukažu na pozive funkcija i parametre funkcija:

```
d = c * (a + b); /* premoštavanje normalnog prvenstva operatora */
if (d == z) ++x; /* ključno kod uslovnih izraza */
func(); /* poziv funkcije bez argumenata */
void func2(int n); /* deklaracija funkcije sa parametrima */
```

Male zagrade se preporučuju u definicijama makroa da bi se izbegli potencijalni problemi sa prvenstvom tokom ekspanzije:

```
#define CUBE(x) ((x)*(x)*(x))
```

Za više informacija, pogledajte Izrazi i Prvenstvo Operatora.

4.11.3 Velike Zagrade

Velike zagrade { } označavaju početak i kraj složenih izraza:

```
if (d == z) {
    ++x;
    func();
}
```

}

Zagrada koja zatvara služi kao označavač kraja složenog izraza, tako da tačka-zarez nije neophodna posle } , osim u deklaraciji struktura. Često je tačka-zarez nedozvoljena, kao u:

if (statement)

{ ... }; /* nedozvoljena tačka-zarez! */

else

{ ... };

Za više informacija pogledajte Složene Izraze.

4.11.4 Zarez

Zarez (,) odvaja elemente funkcijske liste argumenata:

void func(int n, float f, char ch);

Zarez se takođe koristi kao operator u izrazima sa zarezom. Mešanje ova dva načina upotrebe zareza nije dozvoljeno i moraju se koristiti male zagrade da bi se razlikovali. Primetite da (exp1, exp2) procenjuje oboje ali je jednak drugom:

/* pozivanje funkcije sa dva argumenta */

func(i, j);

/* takođe pozivanje funkcije sa dva argumenta! */

func((exp1, exp2), (exp3, exp4, exp5));

4.11.5 Tačka-zarez

Tačka-zarez (;) je znak koji označava kraj naredbe. Bilo koji dozvoljeni C izraz (uključujući prazan izraz) praćen tačka-zarezom se uzima kao naredba. Izraz se izračunava a njegova vrednost odbacuje. Ako naredba nema bočnih efekata, mikroC bi mogao da je ignoriše.

a + b; /* izračunava a + b ali odbacuje rezultat */

++a; /* bočni efekat na a ali odbacuje vrednost ++a */

; /* prazan izraz ili null naredba */

Tačka-zarez se ponekad koristi za stvaranje prazne naredbe:

for (i = 0; i < n; i++) ;

Za više informacija pogledajte Naredbe.

4.11.6 Dvostruka Tačka

Koristite dvostruku tačku (:) za označavanje naredbi sa labelom. Na primer:

start: x = 0;

...

goto start;

Labele se razmatraju u Naredbama sa Labelom

4.11.7 Asterisk (Deklaracija Pokazivača)

Asterisk (*) u deklaraciji označava stvaranje pokazivača na određeni tip:

```
char *char_ptr; /*deklarisan je pokazivač na char*/
```

Za više informacija pogledajte Pokazivače.

4.11.8 Znak Jednakosti

Znak jednakosti (=) razdvaja deklaracije promenljivih od listi inicijalizacije:

```
int test[5] = { 1, 2, 3, 4, 5};
```

```
int x = 5;
```

Znak jednakosti se takođe koristi kao operator dodele u izrazima:

```
int a, b, c;
```

```
a = b + c;
```

Za više informacija pogledajte Operatore Dodele.

4.11.9 Znak „Taraba“ (Pred-procesorska Direktiva)

Znak „taraba“ (#) označava pred-procesorsku direktivu kada se pojavi kao prvi znak u liniji koji nije prazan prostor. On označava radnju kompajlera koja nije neophodno vezana sa generisanjem koda. Pogledajte Pred-procesorske Direktive za više informacija.

i ## se takođe koriste kao operatori za zamenu i spajanje tokena tokom pred-procesorske faze skeniranja. Pogledajte Pred-procesorske operatore.

4.12 Objekti i L-vrednosti

4.12.1 Objekti

Objekat je određeni region u memoriji koji može da sadrži fiksnu ili promenljivu vrednost (ili skup vrednosti). Da bi se izbegla zabuna, ovakva upotreba reči objekat je različita od uopštenijeg termina koji se koristi u objektno-orijentisanim jezicima. Naša definicija te reči obuhvata funkcije, promenljive, simboličke konstante, tipove podataka definisane od strane korisnika, i labelle.

Svaka vrednost ima pridruženo ime i tip (takođe poznat i kao tip podatka). Ime se koristi za pristupanje objektu. Ovo ime može biti jednostavni identifikator ili kompleksni izraz koji jedinstveno označava objekat.

4.12.2 Objekti i Deklaracije

Deklaracije uspostavljaju neophodno mapiranje između identifikatora i objekata. Svaka deklaracija pridružuje tip podatka identifikatoru.

Povezivanje identifikatora sa objektima zahteva da svaki identifikator ima barem dva atributa: klasu skladištenja i tip (koji se ponekad naziva tip podatka). MikroC kompajler izvlači ove attribute iz implicitnih i eksplicitnih deklaracija u izvornom kodu. Uobičajeno, jedino je tip eksplicitno specifikovan dok specifikator klase skladištenja predpostavlja automatsku vrednost-auto.

Uopšteno govoreći, identifikator ne može biti na važeći način korišten u programu pre tačke njegove deklaracije u izvornom kodu. Važeći izuzetak od ovog pravila (poznat kao pred-referenciranje) su labele, pozivi nedeklarisanih funkcija, kao i elementi struktura i unija.

Objekti koji mogu biti deklarirani uključuju:

promenljive, funkcije, tipove, nizove drugih tipova, strukture, unije, elemente pobrojavanja, članove struktura i unija, konstante pobrojavanja, labele naredbi, pred-procesorske makroe.

Rekurzivna priroda sintakse deklaratora omogućava kompleksne deklaratore. Verovatno ćete poželeti da koristite typedef u cilju unapređenja preciznosti pri konstrukciji kompleksnih objekata.

4.12.3 L-vrednosti

L-vrednost je lokator objekta: izraz koji određuje objekat. Primer jednog l-vrednost izraza je `*P`, gde je `P` bilo koji izraz koji nije pokazivač na ništa. Izmenjiva l-vrednost je identifikator ili izraz koji ukazuje na objekat kojem se može pristupiti i koji može na dozvoljen način biti promenjen u memoriji. Konstantni pokazivač na konstantu, na primer, nije izmenjiva l-vrednost. Pokazivač na konstantu može biti promenjen (ali vrednost na koju ukazuje ne može).

Istorijski, `l` označava „left“ (levo), što znači da l-vrednost može na dozvoljen način stajati na levoj (prijemnoj) strani naredbe dodele. Sada jedino izmenjiva l-vrednost može regularno stajati na levoj strani operatora dodele. Na primer, ako su `a` i `b` ne-konstantni celobrojni identifikatori sa ispravno dodeljenim memorijskim prostorom, onda su oba izmenjive l-vrednosti, i dodele kao što su `a = 1`, i `b = a + b`, su dozvoljene.

4.12.4 R-vrednosti

Izraz `a + b` nije l-vrednost: `a + b = a` nije validno zato što izraz na levoj strani nije povezan sa objektom. Izrazi ove vrste se ponekad nazivaju r-vrednosti (skraćeno od right (*desne) vrednosti).

4.13 Doseg i vidljivost

4.13.1 Doseg

Doseg identifikatora je deo programa u kojem identifikator može biti korišten da bi pristupio svom objektu. Postoje različite vrste dosega: blokovski (ili lokalni), funkcije, prototipa funkcije, i fajl, zavisno od toga kako i gde su deklarirani identifikatori.

4.13.2 Blokovski Doseg

Doseg identifikatora sa blokovskim (ili lokalnim) dosegom počinje u tački deklaracije i završava se sa krajem bloka koji sadrži tu deklaraciju (takav jedan blok je poznat pod nazivom ograđujući blok). Deklaracije parametara u okviru definicije funkcije takođe imaju blokovski doseg, ograničen na doseg tela funkcije.

4.13.3 Fajl Doseg

Identifikatori sa fajl dosegom, takođe poznati kao globalne, su deklarirani izvan svih blokova; njihov doseg počinje od tačke deklaracije i traje do kraja izvornog fajla.

4.13.4 Funkcijski doseg

Jedini identifikatori koji imaju funkcijski doseg su labele naredbe. Imena labela se mogu koristiti sa goto naredbama bilo gde u funkciji u kojoj je labela deklarirana. Labela se deklarirše implicitno - pisanjem *ime_labela*: praćeno naredbom. Labele moraju biti jedinstvene u okviru funkcije.

4.13.5 Doseg Funkcijskog Prototipa

Identifikatori deklarirani u okviru liste deklaracija parametara u prototipu neke funkcije (nije deo definicije funkcije) imaju doseg funkcijskog prototipa. Ovaj doseg se završava sa krajem prototipa funkcije.

4.13.6 Vidljivost

Vidljivost nekog identifikatora je oblast izvornog koda programa iz koje je moguće na važeći način pristupiti objektu na koji identifikator ukazuje.

Doseg i vidljivost se obično poklapaju, iako postoje okolnosti pod kojima neki objekat postaje privremeno sakriven pojavom duplikata identifikatora: objekat i dalje postoji ali prvobitni identifikator ne može biti korišten da bi se njemu pristupilo do kraja dosega duplikata identifikatora.

Tehnički, vidljivost ne može premašiti doseg ali doseg može premašiti vidljivost. Pogledajte sledeći primer:

```
void f (int i) {  
  int j;      // auto po default-u  
  j = 3;      // int i i j su u dosegu I vidljivi  
  {          // ugnježdjeni blok  
    double j; // j je lokalno ime u ugnježdenom bloku  
    j = 0.1;  // i je double i vidljivo;  
    // int j = 3 je u dosegu ali sakriveno  
  }  
  // double j izvan dosega  
  j += 1;    // int j vidljivo = 4  
}  
// i i j su izvan dosega
```

4.14 Prostori imena

Prostor imena je opseg u kojem neki identifikator mora biti jedinstven. C koristi četiri različite kategorije identifikatora:

4.14.1 Imena goto labela

Ona moraju jedinstvena u okviru funkcija u kojima su deklarirana.

4.14.2 Strukture, unije, i oznake pobrojanja

Moraju biti jedinstvene u okviru bloka u kojem su definisane. Oznake deklarirane izvan bilo koje funkcije moraju biti jedinstvene.

4.14.3 Imena članova struktura i unija

Moraju biti jedinstvena u okviru strukture ili unije u kojoj su definisani. Ne postoje ograničenja tipa ili pomaka članova sa istim članskim imenom u različitim strukturama.

4.14.4 Promenljive, typedef imena, funkcije i članovi pobrojanja

Moraju biti jedinstveni u okviru dosega u kojem su definisani. Eksterno deklarirani identifikatori moraju biti jedinstveni među eksterno definisanim promenljivim. Dupliranje imena je važeće za različite prostore imena bez obzira na pravila o dosegu.

Na primer:

```
int plavo = 73;
{ // otvori blok
enum boje { crno, crveno, zeleno, plavo, belo } c;
/* enumerator plavo sakriva vanjsku deklaraciju int plavo */
struct boje { int i, j; };
// POGRESNO: dupla oznaka boje
double crveno = 2;
// POGRESNO: redefinisane crveno
}
plavo = 37; // nazad u doseg int plavo
```

4.15 Trajanje

Trajanje, koje je usko vezano za klasu skladištenja, definiše period tokom kojeg deklarirani identifikator ima stvarni, fizički objekat dodeljen u memoriji. Takođe, pravimo razliku između objekata koji postoje tokom kompajliranja i tokom izvršenja. Promenljive na primer, za razliku od tipova i typedef definicija, imaju stvarnu memoriju dodeljenu tokom vremena izvršavanja. Postoje dve vrste trajanja: *statičko* i *lokalno*.

4.15.1 Statičko Trajanje

Memorija se dodeljuje objektima sa statičkim trajanjem čim započne izvršavanje; ovakva dodela traje do okončanja programa. Objekti sa statičkim trajanjem su obično smešteni u fiksne segmente podataka dodeljene u skladu sa memorijskim modelom koji je na snazi. Sve globalne imaju statičko trajanje. Sve funkcije, gde god da su definisane, su objekti sa statičkim trajanjem. Druge promenljive mogu dobiti statičko trajanje korištenjem eksplicitnih specifikatora klase skladištenja *static* ili *extern*.

U mikroC-u, objekti sa statičkim trajanjem nisu inicijalizovani nulom(null) u odsustvu ikakvog eksplicitnog inicijalizatora.

Objekat može imati statičko trajanje i lokalni doseg – pogledajte primer na sledećoj stranici.

4.15.2 Lokalno Trajanje

Objekti sa lokalnim trajanjem su takođe poznati i kao automatski objekti. Oni se kreiraju u steku (ili u registru) kada se uđe u ograđujući blok ili funkciju. Kada program izađe iz bloka ili funkcije dolazi do uklanjanje njihove dodele. Objekti sa lokalnim trajanjem moraju biti eksplicitno inicijalizovani; u suprotnom slučaju je njihov sadržaj nepredvidiv.

Specifikator klase skladištenja *auto* se može koristiti za deklaraciju promenljivih sa lokalnim trajanjem ali je obično suvišan zato što je *auto* podrazumevana vrednost za promenljive definisane u okviru bloka.

Objekat sa lokalnim trajanjem takođe ima lokalni doseg zato što ne postoji izvan ograđujućeg bloka. Obrnuta tvrdnja nije istinita: objekat sa lokalnim dosegom *može* imati statičko trajanje.

Ovde dajemo primer dva objekta sa lokalnim dosegom ali različitim trajanjima:

```
void f() {  
    /* promenljiva sa lokalnim trajanjem a koja se inicijalizuje svakim pozivom f */  
    int a = 1;  
    /* promenljiva sa statičkim trajanjem b koja se inicijalizuje samo prvim pozivom f */  
    static int b = 1;  
    /* tačka provere! */  
    a++;  
    b++;  
}  
void main() {  
    /* Na tački provere imaćemo: */  
    f(); // a=1, b=1, posle prvog poziva,  
    f(); // a=1, b=2, posle drugog poziva,  
    f(); // a=1, b=3, posle trećeg poziva,  
    // itd.  
}
```

4.16 Tipovi

C je strogo tipiziran jezik, što znači da svaki objekat, funkcija i izraz moraju imati strogo definisan tip koji je poznat u vreme kompajliranja. Primetite da C radi isključivo sa numeričkim tipovima.

Tipovi služe:

- Da se odredi ispravna dodela memorije koja je inicijalno potrebna
- Da se protumače bit obrasci nađeni u objektu kod naknadnih pristupa
- Da se osigura prepoznavanje nevažećih dodela u mnogim situacijama provere tipa

MikroC podržava mnoge standardne (unapred definisane) tipove podataka kao i one definisane od strane korisnika, uključujući označene i neoznačene celobrojne brojeve raznih veličina, brojeve u pokretnom zarezu različitih preciznosti, nizove, strukture, i unije. Dodatno, mogu se uspostaviti pokazivači na većinu ovih objekata i sa njima se može upravljati u memoriji.

Tip određuje koliko memorije se dodeljuje objektu i kako će program protumačiti bit obrasce nađene u lokaciji gde je objekat uskladišten. Dati tip podataka može biti posmatran kao skup vrednosti (često zavisao od implementacije) koji identifikatori tog tipa mogu uzeti, sa skupom operacija koje su dozvoljene nad tim vrednostima. Operator koji se koristi u vreme kompajliranja *sizeof* vam omogućava da odredite veličinu u bajtovima bilo kojeg standardnog ili korisnički definisanog tipa.

MikroC standardne biblioteke i vaši sopstveni programski i fajlovi zaglavlja, moraju pružiti jasne identifikatore (ili izraze dobijene iz njih) i tipove tako da mikroC može konzistentno da pristupa, tumači, i (verovatno) menja bit obrasce u memoriji u skladu sa svakim aktivnim objektom u vašem programu.

4.16.1 Kategorije Tipova

Osnovni tipovi predstavljaju tipove koji ne mogu biti razdvojeni u manje delove. Oni se nekada nazivaju i ne-strukturalni tipovi. Osnovni tipovi su *void*, *char*, *int*, *float*, i *double*, uključujući *short*, *long*, *signed*, i *unsigned* varijante nekih od njih.

Izvedeni tipovi su takođe poznati i kao strukturalni tipovi. Izvedeni tipovi uključuju pokazivače na druge tipove, nizove drugih tipova, funkcijske tipove, strukture, i unije.

4.17 Osnovni tipovi

4.17.1 Aritmetički Tipovi

Specifikatori aritmetičkih tipova se grade od sledećih ključnih reči: *void*, *char*, *int*, *float*, i *double*, skupa sa prefiksima *short*, *long*, *signed*, i *unsigned*. Od ovih ključnih reči možete graditi celobrojne i tipove pokretnog zareza. Pregled tipova dajemo na sledećoj strani.

4.17.2 Celobrojni Tipovi

Tipovi *char* i *int*, skupa sa njihovim varijantama, se smatraju celobrojnim tipovima podataka. Varijante se kreiraju korištenjem nekog od prefiksnih modifikatora *short*, *long*, *signed*, i *unsigned*.

Dole navedena tabela je pregled celobrojnih tipova – ključne reči u zagradama mogu biti (a često i jesu) izostavljene.

Modifikatori *signed* i *unsigned* se mogu primeniti i na *char* i na *int*. U odsustvu *unsigned* (*neoznačeni) prefiksa, za celobrojne tipove se automatski podrazumeva *signed* (*označeni). Jedini izuzetak je *char*, koji je *unsigned* po podrazumevanim vrednostima. Ključne reči *signed* i *unsigned*, kada se koriste samostalno, znače *signed int* i *unsigned int*, tim redom.

Modifikatori *short* i *long* se primenjuju samo na *int*. Ključne reči *short* i *long*, kada se koriste samostalno, znače *short int*, i *long int*, tim redom.

4.17.3 Tipovi Pokretnog Zareza

Tipovi *float* i *double*, skupa sa *long double* varijantom, se smatraju tipovima pokretnog zareza. MikroC-ova implementacija ANSI Standarda posmatra sva tri kao isti tip.

Pokretni zarez u mikroC-u je primenjen korištenjem Microchip AN575 32-bitnog formata (u skladu sa IEEE 754).

Sledeća tabela predstavlja pregled aritmetičkih tipova:

Tip	Veličina	Opseg
(unsigned) char	8 bita	0 .. 255
signed char	8 bita	- 128 .. 127
(signed) short (int)	16 bita	- 128 .. 127
unsigned short (int)	16 bita	0 .. 255
(signed) int	32 bita	-32768 .. 32767
unsigned (int)	32 bita	0 .. 65535
(signed) long (int)	64 bita	-2147483648 .. 2147483647
unsigned long (int)	64 bita	0 .. 4294967295
float	32 bita	±1.17549435082E-38 .. ±6.80564774407E38
double	64 bita	±1.17549435082E-38 ..

		$\pm 6.80564774407E38$
long double	32 bita	$\pm 1.17549435082E-38$.. $\pm 6.80564774407E38$

4.17.4 Pobrojavanja

Pobrojavački tip podataka se koristi za prikazivanje apstraktnih, diskretnih skupova vrednosti sa odgovarajućim simboličkim imenima.

4.17.5 Deklaracija Pobrojavanja

Pobrojavanja se deklariraju na sledeći način:

enum *oznaka* {*lista-pobrojavanja*};

Ovde je *oznaka* opcionalno ime pobrojavanja: *lista-pobrojavanja* je jedna lista diskretnih vrednosti, numeratora. Numeratori pobrojani unutar zagrade su takođe poznati i pode imenom konstante pobrojavanja. Svakom je dodeljena fiksna celobrojna vrednost. U odsustvu eksplicitnih inicijalizatora, prvi numerator se postavlja na nulu, a svaki sledeći raste za jedan u odnosu na svog prethodnika.

Promenljive enum tipa se deklariraju na isti način kao i promenljive bilo kojeg drugog tipa. Na primer, sledeća deklaracija:

enum *boje* {crna, crvena, zelena, plava, ljubicasta, bela} *c*;

uspostavlja jedinstven celobrojni tip, *boje*, jednu promenljivu *c* ovog tipa, i skup numeratora sa konstantnim celobrojnim vrednostima (crno = 0, crveno = 1,...). U C-u, promenljivoj pobrojavačkog tipa može biti dodeljena bilo koja vrednost tipa int, i iza toga se ne nameće nikakvo proveravanje tipa, - znači:

c = crveno; // u redu

c = 1; // takođe u redu, značenje je isto

Sa eksplicitnim celobrojnim inicijalizatorima možete odrediti da jedan ili više numeratora ima specifičnu vrednost. Takav inicijalizator može biti bilo koji izraz koji daje pozitivnu ili negativnu celobrojnju vrednost (posle moguće celobrojne pretvorbe). Sva sledeća imena bez inicijalizacije će se povećati za jedan. Ove vrednosti su obično jedinstvene iako je dupliranje dozvoljeno.

Red konstanti može biti eksplicitno preuređen. Na primer:

```
enum boje {  crna,      //vrednost 0
             crvena,    //vrednost 1
             zelena,    //vrednost 2
             plava = 6,  //vrednost 6
             ljubicasta, //vrednost 7
             bela = 4 }; //vrednost 4
```

Inicijalizatorski izraz može uključivati prethodno deklarisanе numеratore. Na primer, u sledećoj deklaraciji:

```
enum velicine_memorije { bit = 1, nibl = 4 * bit,
bajt = 2 * nibl, kilobajt = 1024 * bajt };
```

nibl bi dobio vrednost 4, *bajt* vrednost 8 a *kilobajt* vrednost 8192.

4.17.6 Anonimni Enum Tip

U našoj prethodnoj deklaraciji, identifikator *boje* je opcionalna oznaka pobrojavanja koja se može koristiti u naknadnim deklaracijama promenljivih pobrojavanja tipa *boje*:

```
enum boje bg, granicna_linija; //deklariše promenljive bg i granicna_linija
```

Kao i u slučaju deklaracija struktura i unija, možete izostaviti oznaku ako nema potrebe za naknadnim promenljivim ovog tipa pobrajanja:

```
/* Anonimni enum tip*/
```

```
enum {crna, crvena, zelena, plava, ljubicasta, bela} boja;
```

Doseg pobrojavanja

Oznake pobrojavanja dele isti prostor imena kao oznake struktura i unija. Numeratori dele isti prostor imena kao i obični identifikatori promenljivih. Za više informacija pogledajte Prostore Imena.

4.17.7 Tip Void (*Prazno)

void je specijalni tip koji označava odsustvo ikakve vrednosti. Ne postoje *void* objekti, umesto toga se *void* koristi za izvođenje kompleksnijih tipova.

4.17.8 Void funkcije

Koristite *void* kao povratni tip funkcije ako funkcija ne vraća nikakvu vrednost. Na primer:

```
void print_temp(char temp) {  
    Lcd_Out_Cp("Temperatura:");  
    Lcd_Out_Cp(temp);  
    Lcd_Chrcp(223); // znak stepena  
    Lcd_Chrcp('C');  
}
```

Koristite *void* u zaglavlju funkcije ako funkcija ne uzima nikakve parametre. Alternativno možete pisati samo prazne zagrade:

```
main (void) { // isto što i main()  
    ...  
}
```

4.17.9 Generički Pokazivači

Pokazivači mogu biti deklarirani sa *void*, što znači da mogu pokazivati na bilo koji tip. Ovi pokazivači se ponekad nazivaju generički.

4.18 Izvedeni tipovi

Izvedeni tipovi su takođe poznati i kao strukturalni tipovi. Ovi tipovi se koriste kao elementi u kreiranju kompleksnijih korisnički-definisanih tipova.

Nizovi

Nizovi su najjednostavniji i najčešće korišteni strukturalni tip. Promenljiva tipa niza je zapravo niz objekata istog tipa. Ovi objekti predstavljaju elemente jednog niza i određuje ih njihova pozicija u nizu. Niz se sastoji od kontinualne oblasti skladišnog prostora koja je velika tačno onoliko koliko treba da bi sadržala sve elemente niza.

4.18.1 Deklaracija Niza

Deklaracija niza je slična deklaraciji promenljive, sa srednjim zagradama dodanim nakon identifikatora:

tip ime_niza [konstantni-izraz]

Ovo je deklaracija niza imenovanog sa *ime_niza*, sastavljenog od elemenata tipa *tip*. Tip može biti skalarni tip (osim void), korisnički-definisani tip, pokazivač, pobrojavanje, ili drugi niz. Rezultat *konstantnog-izraza* u okviru srednjih zagrada određuje broj elemenata niza. Ako je izraz dat u deklaratoru niza, njegova vrednost nakon izračunavanja mora biti pozitivni konstantni celobrojni broj. Ova vrednost je broj elemenata u nizu.

Elementi niza su obeleženi brojevima od nula do broja elemenata niza minus jedan. Ako je broj elemenata jednak *n*, elementima niza se može pristupiti kao promenljivim *ime_niza[0]* .. *ime_niza[n-1]* tipa *tip*.

Sledi nekoliko primera deklaracije niza:

```
#define MAX = 50
```

```
int vektor_jedan[10];      /*niz od 10 celobrojnih brojeva*/
```

```
float vektor_dva[MAX];     /*niz od 50 brojeva tipa float*/
```

```
float vektor_tri[ MAX = 20]; /*niz od 30 brojeva tipa float */
```

Inicijalizacija Niza

Niz može biti inicijalizovan u deklaraciji dodelom sekvence vrednosti odvojenih zarezom, ograđene velikim zagradama. Prilikom inicijalizacije niza u deklaraciji možete izostaviti broj elemenata – on će biti automatski određen u skladu sa brojem dodeljenih elemenata. Na primer:

```
/* Niz koji sadrži broj dana po mesecima: */
```

```
int dani[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

```
/* Ova deklaracija je identična prethodnoj */
```

```
int dani[] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

Ako specifikujete i dužinu i početne vrednosti, broj početnih vrednosti ne sme premašiti dužinu. Obrnuto je moguće, i onda će se „višku“ elemenata dodeliti neke vrednosti zatečene tokom izvršavanja u memoriji.

U slučaju niza elemenata tipa char, možete koristiti kraću notaciju string literala. Na primer:

```
/*sledeće dve deklaracije su identične: */
```

```
const char msg1[] = {'T', 'e', 's', 't', '\0'};
```

```
const char msg2[] = "Test";
```


Za više informacija o string literalima pogledajte String Konstante.

4.18.2 Nizovi u Izrazima

Kada se ime niza pojavi u izračunavanju izraza (osim u slučaju operatora *sizeof* i *&*), implicitno se pretvara u pokazivač koji pokazuje na prvi elemenat niza. Pogledajte Nizovi i Pokazivači za više informacija.

4.18.3 Višedimenzionalni Nizovi

Neki niz je jednodimenzionalni ako je skalarnog tipa. Jednodimenzionalni nizovi se ponekad nazivaju i *vektori*.

Višedimenzionalni nizovi se konstruišu deklarisanjem nizova sastavljenih od elemenata tipa niz. Ovi nizovi se skladište u memoriji tako da se desni indeksi menjaju najbrže, što znači da se skladište u „redvima“. Ovde dajemo primer jednog jednostavnog dvodimenzionalnog niza:

```
float m [50] [20] ; /*dvodimenzionalni niz veličine 50x20*/
```

Promenljiva *m* je niz od 50 elemenata, koji su zapravo nizovi od po 20 brojeva tipa float svaki. Stoga, imamo matricu od 50x20 elemenata: prvi elemenat je *m* [0][0], a zadnji *m* [49] [19]. Prvi elemenat petog reda bi bio *m* [0][5].

Ako ne inicijalizujete niz u deklaraciji, možete izostaviti prvu dimenziju višedimenzionalnog niza. U tom slučaju niz se nalazi negde drugo, npr. u drugom fajlu. Ovo je uobičajena tehnika kada se prosleđuju nizovi kao parametri funkcije.

```
int a[3][2][4]; /* 3-dimenzionalni niz veličine 3x2x4 */
```

```
void func(int n[][2][4]) { /*možemo izostaviti prvu dimenziju */
```

```
//...
```

```
n[2][1][3]++; /* inkrement zadnjeg elementa*/
```

```
}//~
```

```
void main() {
```

```
//...
```

```
func(a);
```

```
}//~!
```

Možete inicijalizovati višedimenzionalni niz odgovarajućim skupom vrednosti u velikim zagradama. Na primer:

```
int a[3][2] = {{1,2}, {2,6}, {3,7}};
```

4.18.4 Pokazivači

Pokazivači su specijalni objekti za držanje (ili „pokazivanje na“) memorijskih adresa. U C-u se adresa nekog objekta u memoriji može dobiti korištenjem unarnog operatora *&*. Da bi pristupili objektu na koji se pokazuje, koristimo operator indirekcije (***) na pokazivaču.

Pokazivač tipa „pokazivač na objekt nekog tipa“ drži adresu (zapravo pokazuje na nju) nekog objekta nekog tipa. Pošto su pokazivači objekti, možete imati pokazivač koji pokazuje na pokazivač itd. Drugi objekti na koje se obično pokazuje su nizovi, strukture i unije.

Pokazivač na funkciju je najbolje doživljavati kao jednu adresu, obično u segmentu koda, gde je uskladišten izvršni kod funkcije, što bi bila adresa kojoj se prepušta kontrola po pozivu funkcije.

Iako pokazivači sadrže brojeve sa većinom osobina neoznačenih celobrojnih brojeva, oni imaju sopstvena pravila i ograničenja za deklaracije, dodele, konverzije, i aritmetiku. Primeri u sledećih nekoliko odeljaka ilustruju ova pravila i ograničenja.

Primedba: MikroC trenutno ne podržava pokazivače na funkcije, ali ova osobina će biti implementirana u sledećim verzijama.

4.18.5 Deklaracije Pokazivača

Pokazivači se deklariraju isto kao i bilo koja druga promenljiva, ali sa znakom `*` ispred identifikatora. Tip na početku deklaracije specifikuje tip objekta na koji se pokazuje. Pokazivač mora biti deklarisan tako da pokazuje na neki određen tip čak i kada je taj tip *void*, što zapravo znači pokazivač na bilo šta. Pokazivači na *void* se obično nazivaju *generički* pokazivači, i u mikroC-u se prema njima ophodi kao pokazivačima na *char*.

Ako je tip bilo koji unapred ili korisnički definisan tip, uključujući *void*, deklaracija

```
tip *p; /*ne-inicijalizovani pokazivač*/
```

deklariše da je *p* tipa „pokazivač na *tip*“. Sva pravila o doseg, trajanju, i vidljivosti važe za upravo deklarisan objekat *p*. Možete posmatrati deklaraciju na ovaj način: ako je `*p` objekat tipa *tip*, onda *p* mora da bude pokazivač na takve objekte.

Primedba: Pokazivači moraju biti inicijalizovani pre nego što ih koristite! Naš prethodno deklarisan pokazivač `*p` nije inicijalizovan (nije mu dodeljena vrednost), tako da još ne može biti korišten.

Primedba: U slučaju višestruke deklaracije pokazivača, svaki identifikator zahteva indirektni operator. Na primer:

```
int *pa, *pb, *pc;
```

```
/*je isto što i: */
```

```
int *pa;
```

```
int *pb;
```

```
int *pc;
```

Jednom deklarisan, pokazivač obično može biti preraspoređen tako da pokazuje na objekat drugog tipa. mikroC vam dozvoljava da preraspoređujete pokazivače bez korištenja funkcija konverzije tipa ali će vam kompajler izdati upozorenje ukoliko pokazivač nije prvobitno bio deklarisan da pokazuje na *void*. Možete dodeliti *void* pokazivač pokazivaču koji nije *void* – pogledajte Void Tip za detalje.

4.18.6 Null Pokazivači

Vrednost null pokazivača je adresa za koju je sigurno da će biti različita od bilo kojeg važećeg pokazivača u programu. Dodelom celobrojne konstante 0 pokazivaču dodeljujemo mu vrednost null pokazivača. Umesto nule, mnemoničko *NULL* (definisano u fajlovima zaglavlja standardnih biblioteka, kao što su *stdio.h*) se koristi zbog čitljivosti. Svi pokazivači se mogu uspešno proveriti da li su jednaki ili nisu *NULL*.

Na primer:

```
int *pn = 0; /* Ovo je jedan null pokazivač */
```

```
int *pn = NULL; /* Ovo je ekvivalentna deklaracija */
```

```
/*Ovako možemo proveriti pokazivač: */
```

```
if ( pn == 0 ) { ... }
```

```
/* .. ili ovako: */
```

```
if ( pn == NULL ) { ... }
```

Funkcijski Pokazivači

Funkcijski pokazivači su pokazivači, t.j. promenljive, koje pokazuju na adresu funkcije.

//Definicija funkcijskog pokazivača

```
int (*pnaFunkciju) (float, char, char);
```

Primerba: Stoga funkcije i pokazivači na funkcije sa različitim konvencijama poziva (različit poredak argumenata, tipovi argumenata ili povratni tip) nisu međusobno kompatibilni. Pogledajte Indirektne Funkcijske Pozive.

4.18.7 Dodeljivanje adrese Funkcijskom Pokazivaču

Prilično je lagano dodeliti adresu funkcije funkcijskom pokazivaču. Jednostavno uzmete ime prikladne i poznate funkcije ili funkcije članice. Opcionalno možete koristiti adresni operator & ispred imena funkcije.

//Dodela adrese funkcijskom pokazivaču

```
int Uradi(float a, char b, char c) { return a + b + c ;}
```

```
pnaFunkciju = &Uradi; //dodela
```

Primer:

```
int addC(char x,char y){
```

```
    return x+y;
```

```
}
```

```
int subC(char x,char y){
```

```
    return x-y;
```

```
}
```

```
int mulC(char x,char y){
```

```
    return x*y;
```

```
}
```

```
int divC(char x,char y){
```

```
    return x/y;
```

```
}
```

```
int modC(char x,char y){
```

```
    return x%y;
```

```
}
```

//niz pokazivača na funkcije koji primaju dva chars i vraćaju int

```
int (*arrpf[])(char,char) = { addC ,subC,mulC,divC,modC};
```

```
int res;
```

```
char i;
```

```
void main() {
```

```
    for (i=0;i<5;i++){
```

```
res = arrpf[i](10,20);  
}  
}//!
```

4.18.8 Aritmetika Pokazivača

Aritmetika Pokazivača u C-u je limitirana na:

- dodeljivanje jednog pokazivača drugom
- poređenje dva pokazivača
- poređenje pokazivača sa nulom (NULL)
- dodavanja/oduzimanja celobrojne vrednosti pokazivaču
- oduzimanje dva pokazivača

Unutrašnja aritmetika nad pokazivačima zavisi od memorijskog modela koji je na snazi i prisustva predefinišućih modifikatora pokazivača. Kada se izvodi aritmetika nad pokazivačima predpostavlja se da pokazivač pokazuje na neki niz objekata.

4.18.9 Nizovi i Pokazivači

Nizovi i pokazivači nisu potpuno nezavisni tipovi u C-u. Kada se ime niza pojavi u izračunavanju izraza (osim kod operatora `&` i `sizeof`), vrši se implicitna konverzija u pokazivač koji pokazuje na prvi elemenat niza. Zbog ove činjenice nizovi nisu izmenjive l-vrednosti.

Srednje zagrade `[]` označavaju indekse niza. Izraz

`id [exp]`

je definisan kao

`*((id) + (exp))`

gde je ili:

id pokazivač a *exp* celobrojna vrednost, ili

id celobrojna vrednost a *exp* pokazivač.

Sledeće je tačno:

`&a[i] = a + i`

`a[i] = *(a + i)`

U skladu sa ovim, možemo pisati:

`pa = &a[4]; // pa pokazuje na a[4]`

`x = *(pa + 3); // x = a[7]`

`y = *pa + 3; // y = a[4] + 3`

Takođe morate biti oprezni sa prvenstvom operatora:

`*pa++; //je jednako *(pa++), inkrementacija pokazivača!`

`(*pa)++; //inkrementira pokazivani objekat!`

Sledeći primeri su takođe ispravni ali ih je bolje izbegavati ovakvu sintaksu jer može učiniti kod *stvarno* nečitljivim:

```
(a + 1)[i] = 3;  
// isto kao: *((a + 1) + i) = 3, i.e. a[i + 1] = 3  
(i + 2)[a] = 0;  
// isto kao: *((i + 2) + a) = 0, i.e. a[i + 2] = 0
```

4.18.10 Dodela i Poređenje

Možete da koristite jednostavni operator dodele (=) da dodelite vrednost jednog pokazivača drugom ako su istog tipa. Ako su različitih tipova, morate koristiti operator konverzije tipa. Eksplicitna konverzija tipa nije neophodna ako je jedan od pokazivača generički (tipa *void*). Dodeljivanje celobrojne konstante 0 pokazivaču, mu dodeljuje vrednost null pokazivača. Mnemoničko *NULL* (definisano u standardnim bibliotekama zaglavlja, kao što je *stdio.h*), se može koristiti zbog čitljivosti.

Dva pokazivača koja pokazuju na isti niz se mogu porediti korištenjem relacionih operadora ==, !=, <, <=, >, i >=. Rezultati ovih operacija su isti kao da su korišteni nad indeksnim vrednostima elementa niza koji je u pitanju:

```
int *pa = &a[4], *pb = &a[2];  
if (pa > pb) { ...  
// ovo će biti izvršeno kao 4 je veće od 2  
}
```

Primedba: Poređenje pokazivača koji pokazuju na različite objekte/nizove se izvode na odgovornost programera – potreban je precizan pregled fizičkog skladištenja podataka.

4.18.11 Dodavanje Pokazivaču

Možete koristiti operatore +, ++, i +=, da dodate celobrojnu vrednost pokazivaču. Rezultat dodavanja je definisan samo ako pokazivač pokazuje na neki elemenat jednog niza i ako je rezultat pokazivač koji pokazuje na isti niz (ili jedan elemenat iza njega).

Ako je pokazivač deklarisan da pokazuje na *tip*, dodavanje celobrojne vrednosti pokazivaču povećava pokazivač za taj broj objekata tipa *tip*. Neformalno, možete shvatati P+n kao povećavanje pokazivača P za ($n * sizeof(tip)$) bajtova, onoliko dugo dok pokazivač ostaje u okviru svog važećeg opsega (prvi elemenat do onog iza zadnjeg elementa). Ako *tip* ima veličinu od 10 bajtova, onda dodavanje 5 pokazivaču na *tip* povećava pokazivač za 50 bajtova u memoriji. U slučaju tipa *void*, veličina koraka je jedan bajt.

Na primer:

```
int a[10]; // niz koji sadrži 10 int elemenata  
int *pa = &a[0]; // pa je pokazivač na int, koji pokazuje na a[0]  
*(pa + 3) = 6; // pa+3 je pokazivač koji pokazuje na a[3],  
// a[3] je sada jednako 6  
pa++; // pa sada pokazuje na sledeći elemenat niza, a[1]
```

Naravno da ne postoji takav elemenat kao što je „jedan iza zadnjeg elementa“, ali pokazivaču je dozvoljeno da uzme takvu vrednost. C „garantuje“ da će rezultat dodavanja biti definisan čak i kada pokazuje na jedan elemenat preko veličine niza. Ako P pokazuje na poslednji elemenat niza, P+1 je važeće ali je P+2 nedefinisano.

Ovo vam omogućava da napišete petlje koje sekventno pristupaju elementima niza inkrementacijom pokazivača – u poslednjoj iteraciji pokazivač će pokazivati na element za jedan preko veličine niza, što je dozvoljeno. Kakogod, primena indirektnog operatora (*) na „pokazivač za jedan iza poslednjeg elementa“ vodi nedefinisanim rezultatu.

Na primer:

```
void f (neki_tip a[], int n) {  
    /* funkcija f barata elementima niza a; */  
    /* niz a ima n elemenata tipa neki_tip */  
    int i;  
    neki_tip *p = &a[0];  
    for (i = 0; i < n; i++) {  
        /* .. ovde izvodimo neke operacije sa *p .. */  
        p++; /* .. I sa poslednjom iteracijom p premašuje poslednji element niza a */  
    }  
    /* u ovom momentu *p je nedefinisano! */  
}
```

Oduzimanje od Pokazivača

Slično dodavanju, možete koristiti operatore -, --, i -=, da oduzmete celobrojnu vrednost od pokazivača.

Takođe možete oduzeti dva pokazivača. Razlika će biti jednaka rastojanju između dve adrese na koje se pokazuje, u bajtovima.

Na primer:

```
int a[10];  
int *pi1 = &a[0], *pi2 = &a[4];  
i = pi2 - pi1; // i jednako 8  
pi2 -= (i >> 1); // pi2 = pi2 - 4: pi2 sada pokazuje na a[0]
```

4.18.12 Strukture

Struktura je izvedeni tip koji obično predstavlja korisnički-definisane kolekcije imenovanih članova (ili komponenti). Članovi mogu biti bilo kojeg tipa, ili osnovnog ili izvedenog, (sa nekim restrikcijama koje će biti pomenute kasnije), u bilo kojem poretku. Dodatno, član strukture može biti tip bit-polja koji nije dozvoljen drugde.

Za razliku od nizova, strukture se smatraju za pojedinačne objekte. MikroC-ov tip strukture važno dozvoljava da upravljate kompleksnim strukturama podataka skoro onoliko jednostavno kao pojedinačnim promenljivim.

Primer: mikroC ne podržava anonimne strukture (Odstupanje od ANSI standarda)

4.18.13 Deklaracija i Inicijalizacija Struktura

Strukture se deklariraju korištenjem ključne reči *struct*:

```
struct oznaka {lista_deklaratora_članova};
```

Ovde je *oznaka* ime strukture; *lista_deklaratora_članova* je lista članova strukture, zapravo lista deklaracija promenljivih. Promenljive strukturalnog tipa se deklariraju na isti način kao i promenljive bilo kojeg drugog tipa.

Članski tip ne sme biti isti kao strukturalni tip koji se upravo deklariraju. Kakogod, član može biti pokazivač na strukturu koja se deklariraju kao u sledećem primeru:

```
struct mojastrukt { mojastrukt s; }; /* neispravno! */
```

```
struct mojastrukt { mojastrukt *ps; }; /* OK */
```

Takođe, struktura može sadržavati prethodno definisane strukturalne tipove kada se deklariraju instanca jedne deklarirane strukture. Sledi primer:

```
/* struktura koja definiše Tačku: */
```

```
struct Tacka { float x, y; };
```

```
/* struktura koja definiše Krug: */
```

```
struct Krug {
```

```
  double r;
```

```
  struct Tacka centar;
```

```
} o1, o2; /* deklariraju promenljive o1 i o2 tipa Krug */
```

Primitite da možete izostaviti oznaku strukture ali onda ne možete deklarirati dodatne objekte ovog tipa drugde u programu. Za više informacija pogledajte „Neoznačene Strukture“ ispod.

Struktura se inicijalizuje dodelom sekvence vrednosti razdvojenih zarezom, ogradenih velikim zagradama, slično nizovima. U slučaju deklaracija iz prethodnog primera:

```
/* Deklariraju i inicijalizuju tačke p i q: */
```

```
struct Tacka p = { 1., 1. }, q = { 3.7, -0.5 };
```

```
/* Inicijalizuju već deklarirane krugove o1 i o2: */
```

```
o1 = { 1, { 0, 0 } }; // r je 1, centar je u (0, 0)
```

```
o2 = { 4, { 1.2, -3 } }; // r je 4, centar je u (1.2, -3)
```

4.18.14 Nepotpune Deklaracije

Nepotpune Deklaracije su takođe poznate i kao deklaracije unapred. Neki pokazivač na strukturu tipa A se može ispravno pojaviti u deklaraciji druge strukture B pre nego što je A deklarirano:

```
struct A;          //nepotpuno
```

```
struct B    { struct A *pa; } ;
```

```
struct A    { struct B *pb; } ;
```

Prvo pojavljivanje A se naziva nepotpunim zato što za A nema definicije u tom momentu. Nepotpuna deklaracija je ovde dozvoljena zato što definiciji B nije potrebna veličina A.

4.18.15 Neoznačene Strukture i Typedef

Izostavljanjem oznake strukture dobija se neoznačena struktura. Možete koristiti neoznačene strukture da deklarirate identifikatore u *struct_id_lista* razdvajanjem zareza tako da im je dat strukturalni tip (ili izveden od njega), ali ne možete deklarirati druge objekte ovog tipa na nekom drugom mestu.

Moguće je kreirati typedef tokom kreiranja strukture, sa ili bez oznake:

```
typedef struct { ... } Mojastruct;  
Mojastruct s, *ps, arrs[10];
```

4.18.16 Strukturalne Dodele

Promenljive istog strukturalnog tipa mogu biti dodeljene jedna drugoj korištenjem jednostavnog operatora dodele (=). Ova akcija će kopirati čitav sadržaj promenljive na određite, bez obzira na unutrašnju kompleksnost date strukture.

Ovde vredi primetiti da su dve promenljive istog strukturalnog tipa samo ako su obe definisane istom instrukcijom ili korištenjem istog identifikatora tipa. Na primer:

```
/* a i b su istog tipa: */  
struct {int m1, m2;} a, b;  
/* Ali c i d nisu iako su njihovi opisi struktura identični: */  
struct {int m1, m2;} c;  
struct {int m1, m2;} d;
```

4.18.17 Veličina Strukture

Veličinu strukture u memoriji možete dobiti korištenjem operatora *sizeof*. Veličina strukture ne mora neophodno da bude jednaka sumi veličina njenih članova. Ona je često veća zbog određenih ograničenja memorijskog skladištenja.

4.18.18 Strukture i Funkcije

Funkcija može vratiti strukturalni tip ili pokazivač na strukturalni tip:

```
mojastrukt funk1();      //funk1() vraća strukturu  
mojastrukt *funk2()     //funk2() vraća pokazivač na strukturu
```

Struktura može prosleđena funkciji kao argument na sledeće načine:

```
void funk1(mojastrukt s);    //direktno  
void funk2(mojastrukt *spok) //preko pokazivača
```

Pristup Članovima Strukture

Članovima struktura i unija se pristupa korištenjem dva operatora odabira:

. (tačka)

->(strelica nadesno)

Operator . se naziva direktnim odabiračem člana i koristi se za direktan pristup nekom od članova strukture. Pretpostavimo da je objekat *s* struct tipa *S*, zatim, ako je *m* član identifikator tipa *M*, deklarisanog u *s*, izraz

```
s.m    //direktan pristup članu m
```

je tipa *M*, i predstavlja članski objekat *m* u *s*.

Operator -> se naziva indirektnim (ili pokazivačkim) odabiračem člana. Pretpostavimo da je *ps* pokazivač na *s*, zatim da je *m* identifikator člana tipa *M* deklarisanog u *s*, izraz

```
ps->m    //indirektni pristup članu m;  
        //identičan (*ps).m
```


je tipa M , i predstavlja članski objekat m u s . Izraz $ps \rightarrow m$ je prigodna skraćenica za $(*ps).m$. Na primer:

```
struct mojastrukt {  
    int i; char str[10]; double d;  
} s, *sptr = &s;
```

.
.
.

$s.i = 3$; // dodela i članu *mojastrukt s*

$sptr \rightarrow d = 1.23$; // dodela d članu *mojastrukt s*

Izraz $s.m$ je l-vrednost, ako je ispunjen uslov da je s l-vrednost i da m nije tipa niza. Izraz $sptr \rightarrow m$ je l-vrednost ukoliko m nije tipa niza.

4.18.19 Pristupanje Ugnježdenim Strukturama

Ako struktura B sadrži polje čiji tip je struktura A, članovima A se može pristupiti preko dve primene odabirača člana:

```
struct A {  
    int j; double x;  
};  
struct B {  
    int i; struct A a; double d;  
} s, *sptr;  
//...  
 $s.i = 3$ ; // dodeli 3 i članu B  
 $s.a.j = 2$ ; // dodeli 2 j članu A  
 $sptr \rightarrow d = 1.23$ ; // dodeli 1,23 d članu B  
 $sptr \rightarrow a.x = 3.14$ ; // dodeli 3.14 x članu A
```

4.18.20 Jedinstvenost Struktura

Svaka deklaracija strukture unosi jedinstveni strukturalni tip, tako da u

```
struct A {  
    int i,j; double d;  
} aa, aaa;  
struct B {  
    int i,j; double d;  
} bb;
```

objekti *aa* i *aaa* su oba tipa struct A ali su objekti *aa* i *bb* različitog strukturalnog tipa. Strukture se mogu dodeljivati samo ako su izvor i odredište istog tipa:

$aa = aaa$; /* OK: isti tip, dodela član po član */

$aa = bb$; /* NEDOZVOLJENO: različiti tipovi */

```
/* ali je moguće vršiti dodelu po članovima: */
```

```
aa.i = bb.i;
```

```
aa.j = bb.j;
```

```
aa.d = bb.d;
```

4.18.21 Unije

Unijski tipovi su izvedeni tipovi koji dele mnogo sintaksnih i funkcionalnih osobina strukturalnih tipova. Ključna razlika je da unija dozvoljava samo jednom od svojih članova da bude „aktivan“ u datom vremenskom periodu – član koji je zadnji doživeo promene.

Primerba: mikroC ne podržava anonimne unije (Odstupanje od ANSI standarda).

4.18.22 Deklaracija Unija

Unije se deklariraju isto kao i strukture, sa ključnom reči *union* koja se koristi umesto *struct*:

```
union oznaka {lista_deklaratora_članova} ;
```

Za razliku od članova strukture, vrednost samo jednog od članova unije može biti snimljena u nekom momentu. Podledajmo jednostavan primer:

```
union mojaunija { // oznaka unije je 'mojaunija'
```

```
int i;
```

```
double d;
```

```
char ch;
```

```
} mu, *pm = &mu;
```

Identifikator *mu*, tipa *union mojaunija*, se može koristiti za čuvanje 2-bajtnog *int*, 4-bajtnog *double*, ili 1-bajtnog *char*, ali samo jednog od njih u datom momentu.

4.18.23 Veličina Unije

Veličina Unije je veličina njenog najvećeg člana. U našem prethodnom primeru, oboje – *sizeof(union mojaunija)* i *sizeof(mu)* vraća 4, ostavljajući 2 bajta neiskorištena kada *mu* drži *int* objekat i 3 kada *mu* drži *char*.

4.18.24 Pristup Članovima Unije

Članovima unije se može pristupiti odabiračima člana strukture (*. i ->*) ali je potrebna pažnja. Pogledajte sledeći primer.

Imajući u vidu deklaracije iz prethodnog primera:

```
mu.d = 4.016;
```

```
Lcd_Out_Cp(FloatToStr(mu.d)); // OK: prikazuje mu.d = 4.016
```

```
Lcd_Out_Cp(IntToStr(mu.i)); // neodređen rezultat
```

```
pm->i = 3;
```

```
Lcd_Out_Cp(IntToStr(mu.i)); // OK: prikazuje mu.i = 3
```

Drugi *Lcd_Out_Cp* se može pisati, pošto je *mu.i* celobrojni tip. Kakogod, bit obrazac u *mu.i* odgovara delu prethodno dodeljenog *double*. Kao takav verovatni neće pružiti korisnu celobrojnu interpretaciju.

Kada je ispravno pretvoren, okazivač na uniju pokazuje na svaki od njenih članova i obrnuto.

4.18.25 Bit Polja

Bit polja su određeni brojevi bita koji mogu ali i ne moraju imati pridruženi identifikator. Bit polja nude način podele struktura u imenovane delove veličina koje definiše korisnik.

Strukture i unije mogu sadržavati bit polja. Bit polja mogu biti do 16 bita veličine.

Nije moguće uzeti adresu bit polja.

Primer: Ako vam treba kontrola nad određenim bitovima 8-bitne promenljive (char i unsigned short), ili registara, nije neophodno da deklarirate bit polja. Daleko elegantnije rešenje je da koristite mikroC-ovu ugrađenu mogućnost za pojedinačni pristup – pogledajte Pristupanje Pojedinačnim Bitovima za više informacija.

4.18.26 Deklaracija Bit Polja

Bit polja se mogu deklarirati jedino u strukturama. Deklarirate neku strukturu na uobičajeni način i dodelite pojedinačna polja kao što sledi (polja trebaju da budu neoznačena):

```
struct oznaka { unsigned lista_deklaratora_bit_polja; }
```

Ovde je oznaka opcionalno ime strukture, a lista_deklaratora_bit_polja je lista bit polja. Svaki identifikator komponente zahteva dve tačke i širinu u bitovima da bi bio eksplicitno određen. Ukupna širina svih komponenti ne sme preći dva bajta (16 bitova).

Kao objekat, struktura bit polja zauzima dva bajta. Pojedinačna polja se smeštaju u okviru dva bajta, sa leva na desno. U listi_deklaratora_bit_polja možete uzostaviti identifikator(e) da bi stvorili veštačku „popunu“, i tako preskočili irelevantne bitova.

Na primer, ako nam treba kontrola samo nad bitovima 2-4 registra, u bloku, kreiramo strukturu:

```
struct {  
    unsigned : 2; // Preskoči bitove 0 i 1, ovde nema identifikatora  
    mojibitovi : 3; // Relevantni bitovi 2, 3, i 4  
    // Bitovi 5, 6, i 7 su implicitno izostavljeni  
} myreg;
```

Sledeći primer:

```
typedef struct {  
    prescaler : 2; timeronoff : 1; postscaler : 4; } mojeBitPolje;  
deklariše strukturalni tip MojeBitPolje koji sadrži sledeće tri komponente:  
prescaler (bitovi 0 i 1), timeronoff(bit 2), i postscaler( bitovi 3, 4, 5, i 6).
```

Pristup Bit Poljima

Bit poljima se može pristupiti na isti način kao i članovima strukture, korišćenjem direktnog i indirektnog odabirača člana (. i ->). Na primer, mogli bismo raditi sa našim prethodno deklarisanim *MojeBitPolje*:

```
// Deklaracija TimerControl bit polja:
```

```
MojeBitPolje TimerControl;
```

```
void main() {
```

```
    TimerControl.prescaler = 0;
```

```
TimerControl.timeronoff = 1;  
TimerControl.postscaler = 3;  
T2CON = TimerControl;  
}
```

4.19 Konverzije tipova

C je strogo tipiziran jezik gde svaki operator, naredba, ili funkcija zahteva operande/argumente odgovarajućeg tipa. Ipak, često nam je potrebno da u izrazima koristimo objekte čiji tipovi se ne „poklapaju“. U tom slučaju nam je potrebna konverzija tipa. Konverzija objekta jednog tipa je njegova promena u isti objekat drugog tipa (što znači primena drugog tipa datom objektu). C definiše jedan skup standardnih konverzija za ugrađene tipove, koje pruža kompajler kada je to neophodno.

Konverzija je potrebna u sledećim slučajevima:

- Ako naredba zahteva izraz određenog tipa (u skladu sa definicijom jezika) a mi koristimo izraz drugog tipa,
- Ako operator zahteva operand određenog tipa a mi koristimo operand drugog tipa,
- Ako funkcija zahteva formalni parametar određenog tipa a mi joj prosleđujemo objekat drugog tipa,
- Ako izraz koji prati ključnu reč `return` ne odgovara deklarisanom povratnom argumentu funkcije
- Ako inicijalizujemo objekat (u deklaraciji) sa objektom različitog tipa.

U ovim situacijama će kompajler pružiti automatsku implicitnu konverziju tipova bez ikakvog uplitanja korisnika. Takođe, korisnik može zahtevati eksplicitnu konverziju pomoću operatora konverzije tipova. Za više informacija pogledajte Eksplicitna Konverzija Tipova.

4.19.1 Standardne Konverzije

Standardne konverzije su ugrađene u C. Ove konverzije se izvode automatski, kada god je to potrebno u programu. One se takođe mogu eksplicitno zahtevati korištenjem operatora konverzije tipa (pogledajte Eksplicitna Konverzija Tipova).

Osnovno pravilo automatske (implicitne) konverzije je da se operand jednostavnijeg tipa konvertuju (unapređeni) u tip kompleksnijeg operanda, i potom je tip rezultata tip kompleksnijeg operanda.

4.19.2 Aritmetičke Konverzije

Kada koristite aritmetičke konverzije, kao što su $a + b$, gde su a i b različitih aritmetičkih tipova, mikroC izvršava implicitnu konverziju tipa pre nego što se izraz izračuna. Ove standardne konverzije uključuju unapređenje „nižih“ u „više“ tipove u interesu preciznosti i konzistentnosti.

Dodeljivanje označenog znakovnog objekta (kao što je promenljiva) celobrojnom objektu rezultuje automatskom proširenju znaka. Objekti tipa *signed char* uvek koriste proširenje znaka; objekti tipa *unsigned char* uvek postavljaju visoki bit na nulu kada se konvertuju u *int*.

Konvertovanje dužeg celobrojnog tipa u kraći tip odbacuje bitove višeg reda dok bitove nižeg reda ostavlja nepromenjenim. Konverzija kraćeg celobrojnog tipa u duži ili proširuje znak ili popunjava nulama dodatne bitove nove vrednosti, ovisno o tome da li je kraći tip označeni ili neoznačeni, tim redom.

Primedba: Konverzija brojeva u pokretnom zarezu u celobrojne vrednosti (u dodelama ili preko eksplicitnih konverzija tipa) produkuje tačne rezultate samo ako *float* vrednost ne premašuje opseg određiškog celobrojnog tipa.

Svi mali celobrojni tipovi se konvertuju u skladu sa sledećim pravilima:

1. *char* se konvertuje u *int*
2. *signed char* se konvertuje u *int* sa istom vrednošću
3. *short* se konvertuje u *int*, sa istom vrednošću, sa proširenjem znaka
4. *unsigned short* se konvertuje u *unsigned int*, sa istom vrednošću, sa popunjavanjem nulama
5. *enum* se konvertuje u *int*, sa istom vrednošću

Nakon ovoga, bilo koje dve vrednosti u vezi sa operatorom su ili *int* (uključujući *long* and *unsigned* modifikatore), ili *float* (sto je ekvivalentno sa *double* i *long double* u mikroC-u).

1. Ako je bilo koji operand *float*, drugi operand se konvertuje u *float*
2. U drugom slučaju, ako je bilo koji operator *unsigned long*, drugi operand se konvertuje u *unsigned long*
3. U drugom slučaju, ako je bilo koji operand *long*, drugi operand se konvertuje u *long*
4. U drugom slučaju, ako je bilo koji operand *unsigned*, drugi operand se konvertuje u *unsigned*
5. U drugom slučaju, oba operanda su *int*

Rezultat izraza je istog tipa kao i tip dva operanda.

Ovde dajemo nekoliko primera implicitne konverzije:

$2+3.1 // = 2. + 3.1 = 5.1$

$5/4*3. // = (5/4)*3. = 1*3. = 1.*3. = 3.0$

$3.*5/4 // = (3.*5)/4 = (3.*5.)/4 = 15./4 = 15./4. = 3.75$

4.19.3 Konverzije Pokazivača

Pokazivački tipovi se mogu pretvoriti u druge pokazivačke tipove korištenjem mehanizma konverzije tipova:

char *str;

int *ip;

str = (**char** *)ip;

Uopštenije, „kalup“ (*tip**) će konvertovati pokazivač na tip u „pokazivač na *tip*“.

Eksplicitne Konverzije Tipova

Kompajler će u većini situacija pružiti automatsku implicitnu konverziju tipova tamo gde je to potrebno, bez ikakvog uplitanja od strane korisnika, ali takođe možete i eksplicitno pretvoriti operand jednog tipa u drugi korištenjem prefiksnog unarnog operatora:

(*tip*) objekat

Na primer:

```
char a, b;  
/* Sledeća linija će primorati a da bude unsigned int: */  
(unsigned int) a;  
/* Sledeća linija će: primorati a da bude double,  
potom primorati b da bude double automatski,  
rezultujući vrednošću tipa double: */  
(double) a + b; ekvivalentno sa ((double) a) + b;
```

4.20 Deklaracije

4.20.1 Uvod u Deklaracije

Deklaracija uvodi jedno ili više imena u program – ona obaveštava kompajler šta ta imena predstavljaju, kojeg su tipa, šta su dozvoljene operacije sa njima, itd. Ovaj odeljak razmatra koncepte u vezi sa deklaracijama: deklaracije, definicije, specifikatore deklaracija, i inicijalizaciju.

Opseg objekata koji mogu biti deklarirani uključuje:

- Promenljive
- Konstante
- Funkcije
- Tipove
- Strukture, unije, i oznake pobrojanja
- Članove struktura
- Članove unija
- Nizove drugih tipova
- Labele naredbi
- Pred-procesorske makroe

4.20.2 Deklaracije i Definicije

Definišuće deklaracije, takođe poznate i kao definicije, pored toga što uvode ime objekta, takođe određuju kreaciju (gde i kada) objekta; što znači dodelu fizičke memorije i njenu verovatnu inicijalizaciju. Deklaracije upućivanja, ili samo deklaracije, jednostavno čine njihove identifikatore i tipove znanima kompajleru.

Ovde dajemo pregled. Deklaracija je takođe i definicija osim ako:

- deklarise funkciju bez specifikovanja tela funkcije
- ima eksterni specifikator, i nema inicijalizator ili telo(u slučaju funkcija)
- je typedef deklaracija

Može postojati mnogo deklaracija upućivanja za isti identifikator, pogotovo u programu koji se sastoji od više fajlova, ali samo jedna definišuća deklaracija za taj identifikator je dozvoljena.

Pogledajmo primer:

```
/* Ovde imamo ne-definišuću deklaraciju funkcije max; */  
/* it ona samo obaveštava kompajler da je max funkcija*/  
int max();  
/* Ovde je definicija funkcije max: */  
int max(int x, int y) {  
return (x>=y) ? x : y;  
}  
int i; /* definicija promenljive i */  
int i; /*Greška: i je već definisano! */
```

4.20.3 Deklaracije i Deklaratori

Deklaracija je lista imena. Ova imena se ponekad nazivaju deklaratorima ili identifikatorima. Deklaracija počinje sa opcionalnim specifikatorima klase skladištenja, specifikatorima tipa, i drugim modifikatorima. Identifikatori se razdvajaju zarezima a lista se okončava znakom tačka-zarez.

Deklaracije identifikatora promenljivih imaju sledeći obrazac:

```
klasa _skladištenja [kvalifikator_tipa] tip prom1 [=inic1], prom2 [=inic2],...;
```

gde su *prom1*, *prom2*,...bilo koja sekvenca različitih identifikatora sa opcionalnim inicijalizatorima. Svaka od promenljivih je deklarirana da bude tipa *tip*; ako se izostavi podrazumevana vrednost za *tip* je *int*. Specifikator *klasa _skladištenja* može uzeti vrednosti *extern*, *static*, *register*, ili *auto* po podrazumevanoj vrednosti. Opcionalno, *kvalifikator_tipa* može uzeti vrednosti *const* ili *volatile*. Za više detalja pogledajte Klase Skladištenja i Kvalifikatori Tipa.

Ovde dajemo primer deklaracije promenljivih:

```
/*Stvara 3 celobrojne promenljive zvane x, y, i z, i inicijalizuje x i y vrednostima 1 i 2, tim redom*/
```

```
int x = 1, y = 2, z; //z ostaje neinicijalizovano
```

Sve ove deklaracije su definišuće; skladištenje je dodeljeno a bilo kakvi opcionalni inicijalizatori primenjeni.

4.20.4 Povezivanje

Izvršni program se obično stvara kompajliranjem nekoliko nezavisnih jedinica prevođenja a onda povezivanjem rezultirajućih objektnih fajlova sa bibliotekama koje postoje od ranije. Termin jedinica prevođenja se odnosi na fajl izvornog koda skupa sa uključenim fajlovima, ali bez izvornih linija koje su izostavljene od uslovnih pred-procesorskih direktiva. Problem nastaje kada je isti identifikator deklarisan u različitim dosegima (na primer u različitim fajlovima), ili deklarisan više od jedanput u istom doseg.

Povezivanje je proces koji dozvoljava svakoj instanci identifikatora da bude ispravno pridružena sa jednim određenim objektom ili funkcijom. Svi identifikatori imaju jedan od dva atributa povezivanja koji su u bliskoj vezi sa njihovim dosegom: eksterno povezivanje ili interno povezivanje. Ovi atributi se određuju položajem i formatom vaših deklaracija, skupa sa eksplicitnim (ili implicitnim po podrazumevanim vrednostima) korištenjem specifikatora klase skladištenja *static* ili *extern*.

Svaka instanca nekog određenog identifikatora sa eksternim povezivanjem predstavlja isti objekat ili funkciju kroz ceo skup fajlova i biblioteka koji čine program. Svaka instanca određenog identifikatora sa internim povezivanjem predstavlja isti objekat ili funkciju samo u okviru jednog fajla.

4.20.5 Pravila Povezivanja

Lokalna imena imaju interno povezivanje; isti identifikator se može koristiti u različitim fajlovima da označava različite objekte. Globalna imena imaju eksterno povezivanje; identifikator označava isti objekat kroz sve programske fajlove.

Ako se isti identifikator pojavljuje i sa internim i sa eksternim povezivanjem u okviru istog fajla, identifikator će imati eksterno povezivanje.

Pravila Internog Povezivanja:

1. Imena koja imaju fajl doseg, eksplicitno deklarirana kao *static*, imaju interno povezivanje
2. Imena koja imaju fajl doseg, eksplicitno deklarirana kao *const* i ako nisu eksplicitno deklarirana kao *extern*, imaju interno povezivanje
3. *typedef* imena imaju interno povezivanje
4. Konstante pobrojanja imaju interno povezivanje

Pravilo Eksternog Povezivanja

1. Imena koja imaju fajl doseg, a nisu u skladu sa bilo kojim od prethodno navedenih pravila internog povezivanja, imaju eksterno povezivanje

Specifikatori klase skladištenja *auto* i *register* se ne mogu pojaviti u eksternoj deklaraciji. Za svaki identifikator u nekoj jedinici prevođenja koji je deklarisan sa internim povezivanjem, ne može biti data više od jedne eksterne definicije. Eksterna definicija je jedna eksterna deklaracija koja takođe definiše neki objekat ili funkciju; što znači da ona takođe dodeljuje skladišni prostor. Ako se identifikator deklarisan sa eksternim povezivanjem koristi u izrazu (osim kao deo operanda u *sizeof*), onda tačno jedna eksterna definicija tog identifikatora može postojati u celom programu.

MikroC dozvoljava kasniju deklaraciju eksternih imena, kao što su nizovi, strukture, i unije, da bi se dodala informacija ranijim deklaracijama. Ovo je primer:

```
int a[]; // Nema veličinu
struct mojastrukt; // Oznaka jedino, nema deklaratora članova
.
.
.
int a[3] = {1, 2, 3}; // obezbeđivanje veličine i inicijalizacija
struct mojastrukt {
int i, j;
}; // dodavanje deklaratora članova
```


4.20.5 Klase Skladištenja

Pridruživanje identifikatora objektima zahteva da svaki identifikator ima barem dva atributa: klasu skladištenja i tip (koji se ponekad naziva i kao tip podataka). MikroC-ov kompajler izvlači zaključke o ovim atributima iz implicitnih ili eksplicitnih deklaracija u izvornom kodu.

Klasa skladištenja diktira lokaciju (segment podataka, registar, heap, ili stek memorije) objekta i njegovo trajanje odnosno životni vek (tokom celog trajanja rada programa, ili tokom izvršavanja nekih blokova koda). Klasa skladištenja može biti određena sintaksom deklaracije, njenim pisanjem u izvršnom kodu, ili pomoću oba ova faktora:

klasa skladištenja tip identifikator

Specifikatori klase skladištenja u mikroC-u su:

auto

register

static

extern

Auto

Koristite *auto* modifikator da definišete da lokalna promenljiva ima lokalno trajanje. Ovo je podrazumevana vrednost za lokalne promenljive pa retko ima potrebe za korištenjem ovog modifikatora. Ne možete koristiti *auto* sa globalnim. Pogledajte Funkcije.

4.20.6 Register

Po podrazumevanim vrednostima, mikroC sprema promenljive u unutrašnju memoriju mikrokontrolera. Stoga, modifikator *register* tehnički nema posebno značenje. MikroC kompajler jednostavno ignoriše zahteve za dodelu registara.

4.20.7 Static

Globalno ime deklarirano sa *static* specifikatorom ima interno povezivanje, što znači da je ono lokalno za dati fajl. Pogledajte Povezivanje za više informacija.

Lokalno ime deklarirano sa *static* specifikatorom ima statičko trajanje. Koristite *static* sa lokalnom promenljivom da bi sačuvali zadnju vrednost između sukcesivnih poziva neke funkcije. Pogledajte Trajanje za više informacija.

4.20.8 Extern

Ime deklarirano sa *extern* specifikatorom ima eksterno povezivanje, osim ako je prethodno deklarirano tako da ima interno povezivanje. Deklaracija nije definicija ako ima *extern* specifikator i nije inicijalizovana. Ključna reč *extern* je opcionalna za prototip funkcije.

Koristite *extern* modifikator da naznačite da je zapravo stvarno skladištenje i inicijalna vrednost promenljive, ili tela funkcije, definisano u odvojenom modulu izvornog koda. Funkcije deklarirane sa *extern* su vidljive kroz sve izvorne fajlove u programu osim ako predefinišete funkciju kao *static*.

Pogledajte Povezivanje za više informacija.

4.20.9 Kvalifikatori Tipa

Kvalifikatori tipa - *const* i *volatile* su opcionalni u deklaracijama i zapravo ne utiču na tip deklarisanog objekta.

4.20.10 Kvalifikator *const*

Kvalifikator *const* nagoveštava da deklarirani objekat neće promeniti svoju vrednost tokom trajanja programa. U deklaracijama sa *const* kvalifikatorom potrebno je inicijalizovati sve objekte.

MikroC efektivno tretira objekte deklarirane sa *const* kvalifikatorom isto kao literale ili pred-procesorske konstante. Kompajler će generisati grešku u slučaju pokušaja promene objekta deklariranog sa *const* kvalifikatorom.

Na primer:

```
const double PI = 3.14159;
```

4.20.11 Kvalifikator *volatile*

Kvalifikator *volatile* nagoveštava da promenljiva može promeniti svoju vrednost tokom trajanja programa nezavisno od njega. Koristite *volatile* modifikator da naznačite da promenljiva može biti promenjena nekom rutinom koja se odvija u pozadini, prekidnom rutinom, ili U/I portom. Deklarisanje objekta kao *volatile* upozorava kompajler da ne pravi pretpostavke koje se tiču vrednosti objekta kada izračunava izraze u kojima se taj objekat pojavljuje zato što se njegova vrednost može promeniti u bilo kojem trenutku.

4.20.12 Typedef Specifikator

Specifikator *typedef* uvodi sinonim za neki određen tip. Možete koristiti *typedef* deklaracije da stvorite kraća ili smislenija imena za tipove koji su već određeni samim jezikom, ili za tipove koje ste vi deklarirali. Ne možete koristiti *typedef* specifikator unutar definicije funkcije.

Specifikator *typedef* stoji kao prvi u deklaraciji:

```
typedef <definicija_tipa> sinonim;
```

Ključna reč *typedef* dodeljuje sinonim *definiciji_tipa*. *Sinonim* treba da bude validan identifikator.

Deklaracije koje počinju sa *typedef* specifikatorom ne uvode neki novi objekat ili funkciju datog tipa, već novo ime za neki dati tip. Imajući to u vidu, *typedef* deklaracija je identična „normalnoj“ deklaraciji ali umesto objekata, ona deklarise tipove. Uobičajena praksa je da se identifikatori posebnih korisničkih tipova obeležavaju početnim velikim slovom – ali ovo C ne zahteva.

Na primer:

```
//Deklarišimo jedan sinonim za „unsigned long int“:
```

```
typedef unsigned long int Daljina;
```

```
//Sada sinonim Daljina može biti korišten kao identifikator tipa:
```

```
Daljina i; //Deklaracija promenljive i tipa unsigned long int
```

U *typedef* deklaracijama, kao i u bilo kojim drugim, možete deklarirati nekoliko tipova odjednom.

Na primer:

```
typedef int *Pti, Niz[10];
```

Ovde, *Pti* je sinonim za tip „pokazivač na *int*“, a *Niz* je sinonim za „niz od 10 *int* elemenata“.

4.20.13 Asm Deklaracija

C vam omogućava ugrađivanje asemblerskog koda u izvorni kod korištenjem asm deklaracije. Deklaracije `_asm` i `__asm` su takođe dozvoljene u mikroC-u, i imaju isto značenje. Primetite da ne možete koristiti brojne oznake kao apsolutne adrese za SFR ili GPR promenljive u asemblerskim instrukcijama. Umesto toga možete koristiti simbolička imena (listing će prikazati ta imena kao i adrese).

Možete grupisati asemblerske instrukcije pomoću ključne reči *asm* (ili `_asm`, ili `__asm`):

```
asm {  
    blok asemblerskih instrukcija  
}
```

C komentari (i jedno i više-linijski) su dozvoljeni u ugrađenom asemblerskom kodu. Komentar u asemblerskom stilu - koji počinju znakom tačka-zarez, nisu dozvoljeni.

Ako planirate da koristite određene C promenljive samo u ugrađenom asemblerskom kodu, osigurajte da su barem inicijalizovane u C kodu; u suprotnom će poveziavač prijaviti grešku. Ovo se ne odnosi na unapred definisane globalne kao što je PORTB.

Na primer, sledeći kod neće biti kompajliran pošto poveziavač neće biti u stanju da prepozna promenljivu *mojaprom*:

```
unsigned mojaprom;  
void main() {  
    asm {  
        MOVLW 10 // samo test  
        MOVLW test_main_global_mojaprom_1  
    }  
}
```

Dodavanje sledeće linije (ili neke slične) gore navedenom *asm* bloku će obavestiti poveziavač da se koristi ta promenljiva:

```
mojaprom := 0;
```

Primedba: mikroC neće proveravati da li su stranice fajl registara prikladno postavljene za vašu promenljivu. Stranice fajl registara morate postaviti manuelno u asemblerskom kodu.

4.20.14 Migracija sa starijih verzija mikroC-a

Sintaksa koja se koristi u asm blokovima je nešto drugačija nego što je bila u verziji 2. Ove razlike su:

Deformacija promenljivih je promenjena i sada više nalik C-stilu. Na primer, za promenljivu imenovanu:

- `_myVar`, ako je globalna.

- `FARG_+XX`, ako je lokalna (ovo je stvarna pozicija `myVar` u okviru lokalne funkcij.

- `_myVar_L0(+XX)`, ako je lokalna statička promenljiva (+XX da bi se pristupilo sledećim pojedinačnim bajtovima).

Jedini tipovi čije ime ostaje isto u asm-u kao u C-u su konstante, npr. `INTCON`, `PORTB`, `WREG`, `GIE`, itd.

Pristupanje pojedinačnim bajtovima je takođe različito. Na primer, ako imate globalnu promenljivu „`g_var`“, koja je tipa `long` (znači 4 bajta), pristupate joj na sledeći način:

`MOVF g_var+0, 0` ;stavlja bajt najmanje važnosti `g_var` -a u `W` registar

`MOVF g_var+1, 0` ;drugi bajt `g_var`; odgovara `Hi(*visokom)(g_var)`

`MOVF g_var+2, 0` ;`Higher(*Viši)(g_var)`

`MOVF g_var+3, 0` ;`Highest(*Najviši)(g_var)`

... itd.

Sintaksa za dobijanje adrese nekog objekta je drugačija. Za objekte koji se nalaze u flash ROM-u:

`MOVLW # g_var` ;prvi bajt adrese

`MOVLW @# g_var` ; drug bajt adrese

`MOVLW @@# g_var` ; treći bajt adrese

... itd.

Za objekte koji se nalaze u RAM-u:

`MOVLW CONST1` ; prvi bajt adrese

`MOVLW @CONST1` ; drugi bajt adrese

... itd.

4.20.15 Inicijalizacija

U trenutku deklarisanja možete postaviti inicijalnu vrednost deklarisanog objekta, odnosno inicijalizovati ga. Deo deklaracije koji određuje inicijalizaciju se naziva inicijalizator.

Inicijalizatori za globalne i statičke objekte moraju biti konstante ili konstantni izrazi. Inicijalizator za automatski objekat može biti bilo koji ispravan izraz koji se izračunava na vrednost koj je kompatibilna po dodeli za tip promenljive koja je u pitanju.

Skalarni tipovi se inicijalizuju jednim izrazom, koji je opcionalno zaokružen velikim zagradama.

Na primer:

int i = 1;

char *s = "zdravo";

struct kompleks c = {0.1, -0.2};

// gde je 'kompleks' struktura (float, float)

Za strukture i unije sa automatskim trajanjem skladištenja, inicijalizator mora biti jedan od sledećeg:

- Lista inicijalizatora
- Pojedinačni izraz sa kompatibilnim unijskim ili strukturalnim tipom. U ovom slučaju, inicijalna vrednost objekta je vrednost izraza

Za više informacija pogledajte Strukture i Unije

Takođe možete inicijalizovati nizove znakovnog tipa string literalom, opcionalno ograđenog velikim zagradama. Svaki znak u stringu, uključujući null ograničavač, inicijalizuje sukcesivne elemente niza. Za više informacije pogledajte Nizove.

4.20.16 Automatska Inicijalizacija

MikroC ne pruža automatsku inicijalizaciju objekata. Neinicijalizovane globalne i objekti sa statičkim trajanjem će uzeti slučajne vrednosti iz memorije.

4.21 Funkcije

Funkcije zauzimaju centralno mesto u C programiranju. Funkcije se obično definišu kao potprogrami koji vraćaju neku vrednost u zavisnosti od određenog broja ulaznih parametara. Povratna vrednost funkcije se može koristiti u izrazima – tehnički se funkcijski poziv smatra operatorom koji je kao i svi drugi.

C dozvoljava funkciji da stvara dodatne rezultate pored povratne vrednosti, koji se nazivaju *bočnim efektima*. Često se povratna vrednost čak uopšte i ne koristi, u zavisnosti od bočnih efekata. Ovakve funkcije su ekvivalentne procedurama u drugim programskim jezicima, kao što je Paskal. C ne pravi razliku između procedure i funkcije, funkcije ispunjavaju obe te uloge.

Svaki program mora imati jedinstvenu eksternu funkciju zvanu *main*, koja označava ulaznu tačku programa. Funkcije se obično deklariraju kao prototipovi u standardnim ili korisničkim fajlovima zaglavlja, ili u okviru fajlova programa. Funkcije imaju eksterno povezivanje po podrazumevanim vrednostima i obično im je moguće pristupiti iz bilo kojeg fajla u programu. Ovo se može ograničiti korištenjem specifikatora klase skladištenja *static* u deklaraciji funkcije (pogledajte Klase Skladištenja i Povezivanje).

Primer: Proverite PIC specifičnosti za više informacija o ograničenjima funkcija kod PIC mikrokontrolera.

4.21.1 Deklaracija Funkcija

Funkcije se deklariraju u vašim izvornim fajlovima ili se stavljaju na uslugu pevezivanjem prethodno kompajliranih biblioteka. Sintaksa deklaracije funkcije je:

tip ime_funkcije(lista_deklaratora_parametara);

Ovde treba imati u vidu da *ime_funkcije* mora biti ispravan identifikator. Ovo ime se koristi za pozivanje funkcije; pogledajte Funkcijske Pozive za više informacija. Tip rezultata funkcije je predstavljen sa *tip* i može biti bilo koji standardni ili korisnički definisan tip. Za funkcije koje ne vraćaju rezultat koristite *void* tip. Ako se izostavi *tip* u deklaraciji globalne funkcije, funkcija će automatski pretpostaviti *int* tip po podrazumevanim vrednostima.

Funkcijski *tip* takođe može biti i pokazivač. Na primer *float** znači da je rezultat funkcije pokazivač na *float*. Generički pokazivač *void** je takođe dozvoljen. Funkcija **ne može** vratiti niz ili drugu funkciju.

U okviru malih zagrada, *lista_deklaratora_parametara*, je lista formalnih argumenata koje funkcija uzima. Ovi deklaratori određuju tip svakog parametra funkcije. Kompajler koristi ovu informaciju za proveru ispravnosti poziva funkcije. Ako je lista prazna, funkcija ne

uzima nikakve argumente. Takođe, ako je lista *void*, funkcija ne uzima nikakve argumente; primetite da je ovo jedini način da se *void* koristi kao tip argumenta.

Za razliku od deklaracije promenljivih, svaki argument u listi treba da ima svoj sopstveni specifikator tipa i mogući kvalifikator *const* ili *volatile*.

4.21.2 Prototipovi Funkcija

Data funkcija može biti definisana sam jednom u programu, ali može biti deklarirana više puta, uz uslov da su deklaracije kompatibilne. Ako pišete ne-definišuću deklaraciju funkcije, odnosno bez tela funkcije, ne morate da navedete formalne argumente. Ova vrsta deklaracije, uobičajeno poznata pod nazivom prototip funkcije, omogućava bolju kontrolu nad brojem argumenata i proverom tipa, kao i konverzijom tipa.

Ime parametra u prototipu funkcije ima doseg ograničen na prototip. Ovo omogućava različita imena parametara u različitim deklaracijama iste funkcije:

```
/* Ovde imamo dva prototipa iste funkcije: */
```

```
int test(const char*) // deklarise funkciju test
```

```
int test(const char*p) // deklarise istu funkciju test
```

Prototipovi funkcije uveliko pomažu dokumentovanju koda. Na primer, funkcija *Cf_Init* uzima dva parametra: Kontrolni Port(*Control Port) i Port Podataka(*Data Port). Pitanje je koji je koji? Prototip funkcije

```
void Cf_Init(char *ctrlport, char *dataport);
```

razrešava ovu dilemu. Ako neki fajl zaglavlja sadrži prototipove funkcija, možete pogledati taj fajl da dobijete informacije koje vam trebaju za pisanje programa koji pozivaju te funkcije. Ako uključite identifikator u parametru prototipa, on se koristi samo u slučaju postojanja kasnijih poruka o greškama koje se tiču tog parametra – on nema nikakvog drugog efekta.

4.21.3 Definicija Funkcije

Definicija Funkcije se sastoji od njene deklaracije i tela funkcije. Telo funkcije je tehnički gledano jedna blok – sekvenca lokalnih definicija i naredbi, uokruženih velikim zagradama { }. Sve promenljive deklarirane unutar tela funkcije su lokalne za funkciju, odnosno imaju funkcijski doseg.

Funkcija sama može biti definisana samo u okviru dosega fajla. Ovo znači da deklaracije funkcija ne mogu biti ugnježdene.

Da vratite rezultat funkcije koristite *return* naredbu. Naredba *return* u funkcijama *void* tipa ne može imati parametar – zapravo možete sasvim izostaviti *return* naredbu ako je ona zadnja naredba u telu funkcije.

Ovde dajemo jedan primer definicije funkcije:

```
/* funkcija max vraća veći od svoja dva argumenta: */
```

```
int max(int x, int y) {
```

```
  return (x>=y) ? x : y;
```

```
}
```

Ovde dajemo primer funkcije koja zavisi od bočnih efekata a ne od povratne vrednosti:

```
/* funkcija pretvara Dekartove koordinate (x,y)
```

```
u polarne koordinate (r,fi): */
```

```
#include <math.h>
void polar(double x, double y, double *r, double *fi) {
    *r = sqrt(x * x + y * y);
    *fi = (x == 0 && y == 0) ? 0 : atan2(y, x);
    return; /* ova linija može biti izostavljena */
}
```

4.21.4 Ponovna Ulaznost Funkcija

Ograničena ponovna ulaznost za funkcije je dozvoljena. Funkcije koje nemaju svoj sopstveni okvir funkcije (nemaju argumente i lokalne promenljive) mogu biti pozvane i iz prekida i iz „glavne“ niti. Funkcije koje imaju ulazne argumente i/ili lokalne promenljive mogu biti pozvane iz samo od jedne od gore pomenutih programskih niti.

Proverite *Indirektne Pozive Funkcija*.

4.21.5 Pozivi Funkcija

Funkcija se poziva sa stvarnim argumentima koji su postavljeni u istom redosledu kao formalni parametri koji im odgovaraju. Koristite operator poziva funkcije ():

ime_funkcije(izraz_1, ... , izraz_n)

Svaki izraz u pozivu funkcije je zapravo argument. Broj i tip stvarnih argumenata treba da se poklapa sa brojem i tipom formalnih parametara funkcije. Ako se tipovi ne poklapaju primenjuju se pravila implicitne konverzije tipa. Stvarni argumenti mogu biti bilo kojeg stepena kompleksnosti ali ne bi trebali da se oslanjate na redosled njihovog izvršavanja, zato što on nije određen.

Po pozivu funkcije, svi formalni parametri se kreiraju kao lokalni objekti inicijalizovani vrednostima stvarnih argumenata. Po povratku iz funkcije, privremeni objekat se kreira na mestu poziva, i on je inicijalizovan izrazom *return* naredbe. Ovo znači da se poziv funkcije kao operand u kompleksnom izrazu tretira kao rezultat funkcije.

Ako je funkcija bez rezultata (tip void) ili vam ne treba rezultat, možete pisati poziv funkcije kao samosvojan izraz.

U C-u, skalarni parametri se uvek prosleđuju funkciji po vrednosti. Funkcija može modifikovati vrednosti svojih formalnih parametara ali ovo nema uticaja na stvarne argumente u pozivnoj rutini. Možete proslediti skalarni objekat po adresi deklarisanjem da je formalni parametar pokazivač, a potom možete koristiti operator * da pristupite pokazivanom objektu.

4.21.5 Konverzije Argumenta

Kada nije prethodno deklarisan prototip funkcije, mikroC konvertuje celobrojne argumente u poziv funkcije i skladu sa pravilima celobrojnog proširenja (ekspanzije) kao što je to opisano u Standardnim Konverzijama. Kada je prototip funkcije u dosegu, mikroC konvertuje dati argument u tip deklarisanog parametra kao u dodelama.

Ako je prisutan prototip, broj argumenata se mora poklapati. Tipovi moraju biti kompatibilni samo do te mere da ih dodela može na važeći način konvertovati. Uvek možete koristiti eksplicitnu konverziju tipa da konvertujete argument u tip koji je prihvatljiv prototipu funkcije.

Primerba: Ako se vaš prototip funkcije ne poklapa sa stvarnom definicijom funkcije, mikroC će to primetiti ako i samo ako je definicija u istoj jedinici kompajliranja kao i prototip. Ako kreirate biblioteku rutina sa korespondentnim fajlom zaglavlja prototipova, razmislite o uključivanju tog fajla zaglavlja kada kompajlirate biblioteku, tako da bilo kakva neslaganja između prototipova i stvarnih definicija budu otkrivena.

Kompajler je takođe u stanju da primora argumente da budu odgovarajućeg tipa. Pretpostavimo da imamo sledeći kod:

```
int limit = 32;
char ch = 'A';
long res;
extern long func(long par1, long par2); // prototip
main() {
//...
res = func(limit, ch); // poziv funkcije
}
```

Pošto ima prototip funkcije za *func*, ovaj program konvertuje *limit* i *ch* u *long*, korištenjem standardnih pravila dodele, pre nego što ih smesti u stek za poziv *func*.

Bez prototipa funkcije, *limit* i *ch* bi bili smešteni u stek kao celobrojna i znak, respektivno, i u tom slučaju stek prosleđen funkciji *func* se ne bi slagao u veličini i sadržaju onome što *func* očekuje, što bi vodilo problematičnom rezultatu.

4.21.5 Operator Tri Tačke ('...')

Operator tri tačke ('...') se sastoji od tri uzastopne tačke bez praznog prostora između njih. Tri tačke možete koristiti u listama formalnih argumenata prototipa funkcija da naznačite promenljiv broj argumenata ili argumente sa promenljivim tipovima. Na primer:

```
void func(int n, char ch, ...);
```

Ova deklaracija naznačava da će funkcija biti definisana na takav način da će pozivi morati da imaju barem dva argumenta, jedan *int* i jedan *char*, ali će takođe moći da imaju bilo koji broj dodatnih argumenata.

Primer:

```
#include <stdarg.h>
int addvararg(char a1,...){
va_list ap;
char temp;
va_start(ap,a1);
while( temp = va_arg(ap,char))
a1 += temp;
return a1;
}
int res;
void main() {
res = addvararg(1,2,3,4,5,0);
```



```
res = addvararg(1,2,3,4,5,6,7,8,9,10,0);
} //~!
```

4.22 Operatori

Operatori su tokeni koji započinju neko izračunavanje kada se primene na promenljive i druge objekte u nekom izrazu.

MikroC prepoznaje sledeće operatore:

- Aritmetički Operatori
- Operatori Dodele
- Operatori na nivou Bitova
- Logički Operatori
- Indirektni Operatori (Operatori Upućivanja)(Pogledajte Aritmetiku Pokazivača)
- Relacioni Operatori
- Odabirači Člana Strukture (Pogledajte Pristup Članu Strukture)
- Zarez Operator , (Pogledajte Izraze sa Zarezom)
- Uslovni Operator ? :
- Operator Indeksiranja Niza [] (Pogledajte Nizove)
- Operator Poziva Funkcije () (Pogledajte Pozive Funkcija)
- Sizeof Operator
- Pred-procesorski Operatori # i ## (Pogledajte pred-procesorske Operatore)

4.22.1 Prvenstvo i Asocijativnost Operatora

Postoji 15 kategorija prvenstva, od koji neke sadrže samo jedan operator. Operatori iz iste kategorije imaju jednako prvenstvo međusobno.

Tabela na sledećoj stranici sumira sve mikroC operatore.

Na mestima u tabeli gde se pojavljuju duplikati operatora, prvo pojavljivanje je unarno a drugo binarno. Svaka kategorija ima asocijativno pravilo: sa leva na desno, ili sa desna na levo. U odsustvu zagrada, ova pravila razrešavaju grupisanje izraza sa operatorima jednakog prvenstva.

Prvenstvo	Operandi	Operatori	Asocijativnost
15	2	() [] . - >	Sa leva na desno
14	1	! ~ ++ -- + - * & (type) sizeof	Sa desna na levo
13	2	* / %	Sa leva na desno
12	2	+ -	Sa leva na desno

PROGRAMIRANJE MIKROKONTROLERA

11	2	<< >>	Sa leva na desno
10	2	< <= > >=	Sa leva na desno
9	2	== !=	Sa leva na desno
8	2	&	Sa leva na desno
7	2	^	Sa leva na desno
6	2		Sa leva na desno
5	2	&&	Sa leva na desno
4	2		Sa leva na desno
3	3	? :	Sa leva na desno
2	2	= *= /= %= += -= &= ^= = <<= >>=	Sa desna na levo
1	2	,	Sa leva na desno

4.22.2 Aritmetički Operatori

Aritmetički Operatori se koriste za izvođenje matematičkih izračunavanja. Oni imaju numeričke operande i vraćaju numeričke rezultate. Tip char tehnički gledano predstavlja male celobrojne brojeve tako da char promenljive mogu biti korištene kao operandi u aritmetičkim operacijama.

Svi aritmetički operatori imaju asocijativnost sa leva na desno.

Operator	Operacija	Prvenstvo
+	Sabiranje	12
-	Oduzimanje	12

*	Množenje	13
/	Deljenje	13
%	Vraća ostatak celobrojnog deljenja (ne može se koristiti sa brojevima u pokretnom zarezu)	13
+ (unarni)	Unarni plus (ne utiče na operand)	14
-(unarni)	Unarni minus (ne utiče na operand)	14
++	Inkrement – dodaje jedan vrednosti operanda	14
--	Dekrement – oduzima jedan od vrednosti operanda	14

Primerba: Operator * zavisi od konteksta korištenja i može takođe predstavljati operator pokazivanja. Pogledajte Pokazivače za više informacija.

Binarni Aritmetički Operatori

Deljenje dva celobrojna broja vraća celobrojni broj gde se ostatak jednostavno odbacuje:

/* na primer: */

7 / 4; // jednako 1

7 * 3 / 4; // jednako 5

/* ali: */

7. * 3. / 4.; // jednako 5.25 kao da radimo sa float

Operator ostatka % radi samo sa celobrojnim brojevima; znak rezultata je jednak znaku prvog operanda:

/* na primer: */

9 % 3; // jednako 0

7 % 3; // jednako 1

-7 % 3; // jednako -1

Možemo koristiti aritmetičke operatore za manipulaciju znakovima:

'A' + 32; // jednako 'a' (samo ASCII)

'G' - 'A' + 'a'; // jednako 'g' (ASCII i EBCDIC)

Unarni Aritmetički Operatori

Unarni operatori ++ i -- su jedini operatori u C-u koji mogu biti i prefiksni (npr. ++k, --k) ili sufiksni (npr. k++, k--).

Kada se koriste kao prefiks, operatori ++ i -- (pre-inkrement i pre-dekrement) dodaju ili oduzimaju jedan od vrednosti operanda **pre** izračunavanja. Kada se koriste kao sufiks, operatori ++ i -- dodaju ili oduzimaju jedan od vrednosti operanda **posle** izračunavanja.

Na primer:

```
int j = 5; j = ++k;
```

/ k = k + 1, j = k, što nam daje j = 6, k = 6 */*

```
int j = 5; j = k++;
```

/ j = k, k = k + 1, što nam daje j = 5, k = 6 */*

4.22.3 Relacioni Operatori

Koristite relacione operatore da proverite jednakost ili nejednakost izraza. Ako neki izraz tačan, on vraća 1, u suprotnom slučaju je jednak 0.

Svi relacioni operatori su asocijativni sa leva na desno.

4.22.4 Pregled Relacionih Operatora

Operator	Operacija	10. Prvenstvo
==	Jednako	9
!=	Nije jednako	9
>	Veće od	10
<	Manje od	10
>=	Veće ili jednako	10
<=	Manje ili jednako	10

4.22.5 Relacioni Operatori u Izrazima

Prvenstvo aritmetičkih i relacionih operatorima je određeno na takav način da dozvoli da kompleksni izrazi bez zagrada imaju očekivano značenje:

$a + 5 \geq c - 1.0 / e$ // npr. $(a + 5) \geq (c - (1.0 / e))$

Uvek imajte u vidu da relacioni operatori vraćaju ili 0 ili 1. Razmotrimo sledeće promere:

$8 == 13 > 5$ // vraća 0: $8 == (13 > 5)$, $8 == 1$, 0

$14 > 5 < 3$ // vraća 1: $(14 > 5) < 3$, 1 < 3, 1

$a < b < 5$ // vraća 1: $(a < b) < 5$, (0 or 1) < 5, 1

4.22.6 Operatori na Nivou Bitova

Koristite operatore na nivou bitova da modifikujete pojedinačne bitove numeričkih operanda.

Operatori na nivou bitova su asocijativni sa leva na desno. Jedini izuzetak je bitski komplement operator `~` koji je asocijativan sa desna na levo.

4.22.7 Pregled Operatora na Nivou Bitova

Operator	Operacija	Prvenstvo
<code>&</code>	Bitsko I; vraća 1 ako su oba bita 1, u suprotnom vraća 0	9
<code> </code>	Bitsko ILI; vraća 1 ako je bilo koji ili oba bita 1, u suprotnom vraća 0	9
<code>^</code>	Bitsko ekskluzivno ILI (XOR); vraća 1 ako su bitovi komplementarni, u suprotnom vraća 0	10
<code>~</code>	Bitski komplement (unarni); invertuje svaki bit	10
<code>>></code>	Bitski pomeraj ulevo; pomera bitove na levo	10
<code><<</code>	Bitski pomeraj udesno; pomera bitove na desno	10

Primedba: Operator `&` takođe može biti pokazivački operator upućivanja. Pogledajte Pokazivače za više informacija. Bitski operatori `&`, `|`, i `^` izvode logičke operacije nad odgovarajućim parovima bitova njihovih operanada. Na primer:

```
0x1234 & 0x5678; /* jednako 0x1230 */
```

```
/* zato što ..
```

```
0x1234 : 0001 0010 0011 0100
```

```
0x5678 : 0101 0110 0111 1000
```

```
-----
```

```
& : 0001 0010 0011 0000
```

```
.. that is, 0x1230 */
```

```
/* Slično: */
```

```
0x1234 | 0x5678; /* jednako 0x567C */
```

```
0x1234 ^ 0x5678; /* jednako 0x444C */
```

```
~ 0x1234; /* jednako 0xEDCB */
```

Bitski Operatori Pomeraja

Binarni operatori `<< i >>` pomeraju bitove levog operanda za broj pozicija određen desnim operandom, na levo ili desno, respektivno. Desni operand mora biti pozitivan.

Sa pomerajem ulevo (<<), bitovi koji su najviše sa leve strane se odbacuju, a „novim“ bitovima sa desne strane su dodeljene nule. Stoga, pomerao neoznačenog operanda na levo za n pozicija je ekvivalentno njegovom množenju sa 2^n ako su svi odbačeni bitovi bili nule. Ovo važi i za označene operande ako su svi odbačeni bitovi jednaki bitu znaka.

```
000001 << 5; /* jednako 000040 */
```

```
0x3801 << 4; /* jednako 0x8010, premašenje! */
```

Sa desnim pomerajem (>>) se odbacuju bitovi koji su najviše sa desne strane, a „oslobođenim“ bitovima sa leve strane se dodeljuju nule (u slučaju neoznačenog operanda) ili vrednost bita znaka (u slučaju označenog operanda). Pomeranje operanda na desno za n pozicija je ekvivalentno njegovoj podeli sa 2^n .

```
0xFF56 >> 4; /* jednako 0xFFFF5 */
```

```
0xFF56u >> 4; /* jednako 0xFFFF5 */
```

4.22.8 Bitsko naspram Logičkog

Budite svesni principijelne razlike između načina rada operatora na nivou bitova i logičkih operatora. Na primer:

```
0222222 & 0555555; /* jednako 000000 */
```

```
0222222 && 0555555; /* jednako 1 */
```

```
~ 0x1234; /* jednako 0xEDCB */
```

```
! 0x1234; /* jednako 0 */
```

4.22.9 Logički Operatori

Operandi logičkih operacija se smatraju istinitim ili neistinitim, odnosno različitim od nule ili nulom. Logički operatori uvek vraćaju 1 ili 0. Operandi u logičkom izrazu moraju biti skalarnog tipa.

Logički operatori && i || su asocijativni sa leva na desno. Logički operator negacije ! je asocijativan sa desna na levo.

Operator	Operacija	Prvenstvo
&&	Logičko ILI	5
	Logičko I	4
!	Logička negacija	14

Prvenstvo logičkih, relacionih, i aritmetičkih operatora je odabrano na takav način da omogućava očekivano značenje kompleksnim izrazima bez korišćenja zagrada:

```
c >= '0' && c <= '9'; // se čita kao: (c>='0') && (c<='9')
```

```
a + 1 == b || ! f(x); // se čita kao: ((a+1)== b) || (!f(x))
```

Logičko I (&&) vraća 1 samo ako se oba izraza ne svode na vrednost različitu od nule, inače vraća 0. Ako se prvi izraz svodi na neistinito, drugi izraz se ne izračunava. Na primer:

`a > b && c < d;` // se čita kao: `(a > b) && (c < d)`

// ako je `(a > b)` neistinito (0), `(c < d)` se neće izračunavati

Logičko ILI (||) vraća 1 ukoliko se bilo koji od izraza ne svodi na vrednost različitu od nule, inače vraća 0. Ako se prvi izraz svodi na istinito, drugi izraz se ne izračunava. Na primer:

`a && b || c && d;` // se čita kao: `(a && b) || (c && d)`

// ako je `(a && b)` istinito (1), `(c && d)` se neće izračunavati

4.22.10 Logički Izrazi i Bočni Efekti

Opšte pravilo u radu sa kompleksnim logičkim izrazima je da izračunavanje uzastopnih logičkih operanda staje istog trenutka kad je poznat konačni rezultat. Na primer, ako imamo izraz:

`a && b && c`

gde je *a* neistinito (0), to znači da se operandi *b* i *c* neće izračunavati. Ovo je veoma važno ako su *b* i *c* izrazi pošto njihovi mogući bočni efekti neće nastupiti.

4.22.11 Logičko naspram Bitskog

Imajte u vidu principijelnu razliku u načinu kako operatori na nivou bitova i logički operatori funkcionišu. Na primer:

`0222222 & 0555555` /* jednako 000000 */

`0222222 && 0555555` /* jednako 1 */

`~ 0x1234` /* jednako 0xEDCB */

`! 0x1234` /* jednako 0 */

Uslovni Operator ? :

Uslovni operator ? : je jedini ternarni operator u C-u. Sintaksa uslovnog operatora je:

izraz1 ? *izraz2* : *izraz3*

Izraz1 se izračunava prvi. Ako je njegova vrednost istinita, onda se izračunava *izraz2* dok se *izraz3* ignoriše. Ako se *izraz1* svodi na neistinito, onda se izračunava *izraz3* dok se *izraz2* ignoriše. Rezultat će biti vrednost ili *izraza2* ili *izraza3*. Činjenica da se izračunava samo jedan od ova dva izraza je veoma važna ako očekujete od njih da daju bočne efekte!

Uslovni operator je asocijativan sa desna na levo.

Ovde dajem par praktičnih primera:

/* Nađi max(a, b): */

`max = (a > b) ? a : b;`

/* Pretvori mala slova u velika: */

/* (zagrade zapravo nisu neophodne) */

`c = (c >= 'a' && c <= 'z') ? (c - 32) : c;`

4.22.12 Pravila Uslovnog Operatora

Izraz1 mora biti skalarni izraz; *izraz2* i *izraz3* se moraju povinovati jednom od sledećih pravila:

1. Oba aritmetičkih tipova: *izraz2* i *izraz3* su podložni uobičajenim aritmetičkim konverzijama koje određuju rezultujući tip.
2. Oba kompatibilnih *struct* ili *union* tipova. Rezultujući tip je tip strukture ili unije *izraza2* i *izraza3*.
3. Oba *void* tipa. Rezultujući tip je *void*
4. Oba tipa pokazivača na kvalifikovane ili nekvalifikovane verzije kompatibilnih tipova. Rezultujući tip je pokazivač na tip kvalifikovan svim kvalifikatorima tipa tipova na koje pokazuju oba operanda.
5. Jedan operand je pokazivač a drugi *null*-pokazivač konstanta. Rezultujući tip je pokazivač na tip kvalifikovan svim kvalifikatorima tipa tipova na koje pokazuju oba operanda.
6. Jedan operand je pokazivač na objekat nekompletnog tipa, a drugi je pokazivač na kvalifikovanu ili nekvalifikovanu verziju *void*-a. Rezultujući tip je tip operanda koji ne pokazuje na *void*.

4.22.13 Operatori Dodele

Za razliku od mnogih programskih jezika, C tretira dodelu vrednosti kao operaciju (predstavljenu operatorom) pre nego instrukciju.

4.22.14 Obični Operator Dodele

Za uobičajenu dodelu vrednosti, koristimo obični operator dodele (=):

izraz1 = *izraz2*

Izraz1 je objekat (memorijska lokacija) kojoj dodeljujemo vrednost *izraza2*. Operand *izraz1* mora biti l-vrednost dok *izraz2* može biti bilo koji izraz. Izraz dodele sam po sebi nije l-vrednost.

Ako su *izraz1* i *izraz2* različitih tipova, rezultat *izraza2* će biti konvertovan u tip *izraza1* ako je to neophodno. Pogledajte Konverziju Tipova za više informacija.

4.22.15 Složeni Operatori Dodele

C dozvoljava kompleksnije dodele služeći se složenim operatorima dodele. Sintaksa složenog operatora dodele je:

izraz1 op= *izraz2*

gde op može biti jedan od binarnih operatora +, -, *, /, %, &, |, ^, <<, ili >>.

Stoga, imamo 10 različitih složenih operatora dodele: +=, -=, *=, /=, %=, &=, |=, ^=, <<=, i >>=. Svi oni su asocijativni sa leva na desno. Prazan prostor unutar složenih operatora (npr. + =) će stvoriti grešku.

Složena dodela ima isti rezultat kao:

izraz1 = *izraz1* op *izraz2*

osim se l-vrednost *izraza1* izračunava samo jednom. Na primer,

izraz1 += *izraz2*

je isto što i

izraz1 = *izraz1* + *izraz2*

Pravila Dodele

Operandi *izraz1* i *izraz2* moraju poštovati jedno od sledećih parvila i u slučaju običnog kao i u slučaju složenih dodela:

1. *izraz1* je kvalifikovani ili nekvalifikovani aritmetički tip a *izraz2* je aritmetički tip.
2. *izraz1* ima kvalifikovanu ili nekvalifikovanu verziju tipa strukture ili unije, kompatibilan sa tipom *izraz2*.
3. *izraz1* i *izraz2* su pokazivači na kvalifikovanu ili nekvalifikovanu verziju kompatibilnih tipova, i tip na koji pokazuje levi ima sve kvalifikatore tipa na koji pokazuje desni.
4. *izraz1* ili *izraz2* je pokazivač na objekat nekompletnog tipa dok je drugi pokazivač na kvalifikovanu ili nekvalifikovanu verziju *void*-a. Tip na koji pokazuje levi ima sve kvalifikatore tipa na koji pokazuje desni.
5. *izraz1* je pokazivač a *izraz2* null-pokazivač konstanta.

4.22.16 Sizeof Operator

Prefiksni unarni operator *sizeof* vraća celobrojnu konstantu koja predstavlja veličinu memorijskog prostora u bajtovima koju koristi operand (određenu njegovim tipom, uz neke izuzetke).

Operator *sizeof* može kao operand uzeti ili identifikator tipa ili unarni izraz. Ne možete koristiti *sizeof* sa izrazima funkcijskog tipa, nekompletnim tipovima, imenima u zagradama takvih tipova, ili sa l-vrednošću koja označava objekat bit polja.

4.22.17 Sizeof Primenjen na Izraz

Ako se primeni na izraz, veličina operanda se određuje bez izračunavanja izraz (i stoga bez bočnih efekata). Rezultat operacije će biti veličina tipa rezultata izraza.

4.22.18 Sizeof Primenjen na Tip

Ako se primeni na identifikator tipa, *sizeof* vraća veličinu određenog tipa. Jedinica za veličinu tipa je **sizeof(char)** što je ekvivalentno jednom bajtu. Operacija **sizeof(char)** daje rezultat 1, bilo da je char označeni ili neoznačeni.

sizeof(char) /* vraća 1 */

sizeof(int) /* vraća 2 */

sizeof(unsigned long) /* vraća 4 */

Kada je operand ne – parametar tipa niza, rezultat je ukupni broj bajtova u nizu (drugim rečima, ime niza nije konvertovano u pokazivački tip):

int i, j, a[10];

//...

j = **sizeof**(a[1]); /* j = sizeof(int) = 2 */

i = **sizeof**(a); /* i = 10*sizeof(int) = 20 */

Ako je operand parametar deklarisan tako da je tipa niza ili funkcije, *sizeof* daje veličinu pokazivača. Kada se primenjuje na strukture i unije, *sizeof* daje ukupan broj bajtova uključujući popune. Operator *sizeof* se ne može primeniti na funkcije.

4.23 Izlazi

Izraz je sekvenca operatora, operanada, i znakova interpunkcije, koji određuju izračunavanje. Formalno, izraz se definiše rekurzivno: pod-izrazi mogu biti ugnježdjeni bez formalnog ograničenja. Kakogod, kompajler će prijaviti grešku prekoračenja memorije ako ne može da kompajlira izraz koji je previše kompleksan.

U ANSI C-u, primarni izrazi su: konstantna (takođe zvana i literal), identifikator, i (*izraz*), definisani rekurtivno.

Izrazi se izračunavaju u skladu sa određenom konverzijom, grupisanjem, asocijativnošću, i pravilima prvenstva koja zavise od operatora koji se koristi, prisutstva zagrada, i tipa podataka operanada. Prvenstvo i asocijativnost operatora su izloženi istoimenom poglavlju. Način na koji su operandi i pod - izrazi grupisani ne određuje neophodno stvarni red kojim ih izračunava mikroC.

Izrazi mogu stvoriti l-vrednost, ili r-vrednost, ili nikakvu vrednost. Izrazi mogu prouzrokovati bočne efekte bilo da stvaraju vrednost ili ne.

4.23.1 Izrazi sa Zarezom

Jedna od specifičnosti C-a je da vam dozvoljava korištenje zareza kao operatora sekvence da bi se formirali tzv. izrazi sa zarezom ili sekvence. Izraz sa zarezom je lista izraza razdvojenih zarezom – i formalno se tretira kao pojedinačni izraz tako da se može koristiti na mestima gde je predviđen izraz. Sledeća sekvenca:

```
izraz_1, izraz_2;
```

rezultira izračunavanjem sa leva na desno oba izraza, sa tima da vrednost i tip *izraza_2* daju rezultat celog izraza. Rezultat *izraza_1* se odbacuje.

Binarni operator zarez (,) ima najniže prvenstvo i asocijativnost sa leva na desno, tako da je *a, b, c* isto što i (*a, b*), *c*. Ovo nam omogućava da pišemo sekvence sa bilo kojim brojem izraza:

```
izraz_1, izraz_2, ... izraz_n;
```

što rezultira izračunavanjem sa leva na desno svih izraza, sa tim da vrednost i tip *izraza_n* daju rezultat celom izrazu. Rezultati ostalih izraza se odbacuju ali njihovi (mogući) bočni efekti uzimaju mesto.

Na primer:

```
rezultat = (a = 5, b /= 2, c++);
```

```
/* vraća pred-inkrementovanu vrednost c, ali takođe inicijalizuje a, deli b sa 2, i inkrementira c */
```

```
rezultat = (x = 10, y = x + 3, x--, z -= x * 3 - --y);
```

```
/* vraća izračunatu vrednost promenljive z i takođe izračunava x i y */
```

Primerba

Pazite da ne pomešate operator zarez (operator sekvence) sa znakom interpunkcije zarez koji razdvaja elemente u listi argumenata funkcije i listi inicijalizatora. Mešanje te dve upotrebe zareza je dozvoljeno, ali morate koristiti male zagrade da naznačite razliku između njih.

Da bi ste izbegli mogućnost dvojakog tumačenja zareza u listama argumenata funkcija i inicijalizator, koristite male zagrade. Na primer:

funk (i, (j = 1, j + 4), k);

poziva funkciju funk sa tri argumenta (i, 5, k), a ne četiri.

4.24 NAREDBE

Naredbe određuju tok kontrole kako se neki program izvršava. U odsustvu specifičnih naredbi skoka i selekcije, naredbe se izvršavaju sekvencijalno redom kojim se pojavljuju izvornom kodu.

Naredbe se grubo mogu podeliti na:

- Naredbe sa labelom
- Naredbe Izraza
- Naredbe Selekcije
- Naredbe Iteracije(Petlje)
- Naredbe Skoka
- Složene Naredbe (Blokovi)

4.24.1 Naredbe sa Labelom

Svaka naredba u programu može biti označena labelom. Labela je identifikator koji se dodaje pre naredbe na sledeći način:

identifikator_label : naredba;

Ne postoji neka posebna deklaracija labele, ona samo „označava“ datu naredbu. *Identifikator_label* ima funkcijski doseg i labela ne može biti predefinisana u okviru iste funkcije.

Labele imaju svoj sopstveni prostor imena: identifikator labele se može poklapati sa bilo kojim drugim identifikatorom u programu.

Naredba može biti označena labelom iz dva razloga:

1. Identifikator labele služi kao cilj безусловne *goto* naredbe,
2. Identifikator labele služi kao cilj *switch* naredbe. U ovu svrhu koriste se samo naredbe *case* i *default* označene labelom:

case konstantni_izraz: naredba

default: naredba

4.24.2 Naredbe Izraza

Bilo koji izraz praćen tačka-zarezom formira naredbu izraza:

izraz;

MikroC izvršava naredbu izraza izračunavanjem *izraza*. Svi bočni efekti ovog izračunavanja se kompletiraju pre nego što se izvrši sledeća naredba. Većina naredbi izraza su naredbe dodele i pozivi funkcija.

Null (**prazna*) *naredba* je poseban slučaj. Sastoji se od samog znaka tačka-zarez (;). *Null* naredba ne radi ništa i stoga je korisna u situacijama kada sintaksa mikroC-a očekuje naredbu ali vašem programu ona ne treba. Na primer *null* naredba se obično koristi u „praznoj“ petlji:

```
for (; *q++ = *p++ );  
/* telo ove petlje je null naredba */
```

4.24.3 Naredbe Selekcije

Naredbe selekcije, ili kontrole toka, odabiru neki od alternativnih pravaca delovanja ispitivanjem određenih vrednosti. U C-u postoje dve naredbe selekcije: *if* i *switch*.

4.24.4 If Naredba

Koristite *if* naredbu da primenite uslovni iskaz. Sintaksa *if* naredbe je sledeća:

```
if (izraz) naredba1 [else naredba2]
```

Kada je *izraz* istinit izvršava se *naredba1*. Ako je *izraz* neistinit, izvršava se *naredba2*. *Izraz* se mora svoditi na celobrojnu vrednost, u drugom slučaju je uslov loše formulisan. Male zagrade oko *izraza* su obavezne.

Ključna reč *else* je opcionalna, ali nijedna naredba ne sme stajati između *if* i *else*.

4.24.5 Ugnježdene if naredbe

Ugnježdene *if* naredbe zahtevaju posebnu pažnju. Opšte pravilo je da se ugnježdene uslovne naredbe obrađuju počevši od uslovne koja je najdublje unutra, sa svakom *else* vezanom za najbliži *if* sa svoje leve strane.

```
if (izraz1) naredba1
```

```
else if (izraz2)
```

```
if (izraz3) naredba2
```

```
else naredba3/* ovo pripada: if (izraz3) */
```

```
else naredba4/* ovo pripada: if (izraz2) */
```

Primer: *#if* i *#else* pred-procesorske naredbe (direktive) izgledaju slično *if* i *else* naredbama ali imaju veoma različit efekat. One kontrolišu koje linije izvornog fajla se kompajliraju a koje ignorišu. Pogledajte Pred-procesor za više informacija.

4.24.6 Switch Naredba

Koristite *switch* naredbu da prepustite kontrolu određenoj programskoj grani, bazirano na određenom uslovu. Sintaksa *switch* naredbe je:

```
switch (izraz) {  
case konstantni_izraz_1 : naredba_1;  
.  
.  
.  
case konstantni_izraz_n: naredba_n;  
[default : naredba;]  
}
```

Prvo se izračunava *izraz* (uslov). *Switch* naredba potom poredi rezultat sa svim dostupnim *konstantnim_izrazima* koji prate ključne reči *case*. Ako se nađe poklapanje, *switch* predaje kontrolu odgovarajućem *case* te se u toj tačku izvršava *naredba* koja prati poklapanje.

Primitite da se *konstantni_izraz* mora svoditi na integer. Ne sme biti dva *konstantna_izraza* koji se svode na istu vrednost.

Male zagrade oko *izraza* su obavezne.

Po nalasku poklapanja, programski tok se nastavlja normalno: sledeće instrukcije će biti izvršene po prirodnom redosledu nezavisno od moguće *case* labele. Ako nijedan *case* ne zadovoljava uslov, izračunava se *default* (ako je navedena *default* labela).

Na primer, ako promenljiva *i* ima vrednost između 1 i 3 sledeća *switch* će je uvek vratiti kao 4:

```
switch (i) {  
case 1: i++;  
case 2: i++;  
case 3: i++;  
}
```

Da izbegnete izračunavanje bilo kojeg drugog *case* i povratite kontrolu od *switch* naredbe, završite svaki *case* sa *break*.

Uslovne *switch* naredbe mogu biti ugnježdene – labele *case* i *default* se onda dodeljuju najdubljoj okružujućoj *switch* naredbi.

Ovde dajemo jednostavan primer sa *switch* naredbom. Pretpostavimo da imamo promenljivu sa samo 3 različita stanja (0, 1, i 2) i korespodentnu funkciju (dogadjaj) za svako od ovih stanja. Na sledeći način bi trebalo da prebacimo kod na odgovarajuću rutinu:

```
switch (stanje) {  
case 0: Nisko(); break;  
case 1: Srednje(); break;  
case 2: Visoko(); break;  
default: Poruka("Nevažeće Stanje!");  
}
```

4.24.7 Naredbe Iteracije

Naredbe Iteracije vam omogućavaju da neki skup naredbi izvršavate u petlji. Postoje tri forme naredbi iteracije u C-u: *while*, *do*, i *for*.

4.24.8 While Naredba

Koristite ključnu reč *while* za uslovnu iteraciju neke naredbe. Sintaksa *while* naredbe je sledeća:

while (*izraz*) *naredba*

Izvršenje *naredbe* se ponavlja sve dok vrednost *izraza* ne bude neistina. Provera se vrši pre izvršenja *naredbe*. Stoga, ako je *izraz* neistinit pri prvom prolazu, petlja se ne izvršava.

Zagrade oko *izraza* su obavezne.

Ovde dajemo primer izračunavanja skalarnog proizvoda dva vektora korištenjem *while* naredbe:

```
int s = 0, i = 0;  
while (i < n) {
```

```
s += a[i] * b[i];  
i++;  
}
```

Primetite da telo petlje može biti prazna naredba. Na primer:

```
while (*q++ = *p++);
```

4.24.9 Do Naredba

Do Naredba se izvršava dok uslov ne postane neistinit. Sintaksa *do* naredbe je:

```
do naredba while (izraz);
```

Izvršenje *naredbe* se ponavlja onoliko dugo dok je vrednost *izraza* različita od nule. *Izraz* se izračunava nakon svake iteracije tako da će petlja izvršiti *naredbu* barem jednom.

Male zagrade oko *izraza* su obavezne.

Primetite da je *do* jedina kontrolna struktura u C-u koja se eksplicitno završava tačka-zarezom. Druge kontrolne strukture se završavaju naredbom, što znači da one implicitno uključuju tačku-zarez ili okružujuće zagrade.

Ovde dajemo primer izračunavanja skalarnog proizvoda dva vektora, korištenjem *do* naredbe:

```
s = 0; i = 0;  
do {  
s += a[i] * b[i];  
i++;  
} while (i < n);
```

4.24.10 For Naredba

Naredba *for* primenjuje iterativnu petlju. Sintaksa *for* naredbe je:

```
for ([init-izr]; [uslov-izr]; [inkrement-izr]) naredba
```

Pre prve iteracije petlje, izraz *init-izr* postavlja početne promenljive za petlju. Ne možete proslediti deklaracije u *init-izr*.

Izraz *uslov-izr* se proverava pre prvog ulaska u blok; izvršenje *naredbe* se ponavlja dok vrednost *uslov-izr* ne postane neistinito. Nakon svake iteracije petlje *inkrement-izr* povećava brojač petlje. Konsekventno, *i++* je funkcionalno isto kao *++i*.

Svi izrazi su opcionalni. Ukoliko je izostavljen *uslov-izr*, uzima se da je uvek istinit. Stoga se „prazna“ *for* naredba obično koristi za stvaranje beskonačne petlje u C-u:

```
for ( ; ; ) { ... }
```

Jedini način da se izađe iz ove petlje je pomoću *break* naredbe.

Ovde dajemo primer izračunavanja skalarnog proizvoda dva vektora korištenjem *for* naredbe:

```
for (s = 0, i = 0; i < n; i++) s += a[i] * b[i];  
Ovo možete uraditi i na sledeći način:  
/* dozvoljeno, ali ružno */  
for (s = 0, i = 0; i < n; s += a[i] * b[i], i++);
```

ali se ovo smatra lošim stilom programiranja. Iako dozvoljeno, izračunavanje sume nebi trebalo biti deo inkrementalnog izraza zato što nije u službi rutine petlje. Primetite da smo koristili null naredbu (;) za telo petlje.

4.24.11 Naredbe Skoka

Naredba skoka, kada se izvrši, prebacuje kontrolu bezuslovno. Postoje četiri takve naredbe u mikroC-u: *break*, *continue*, *goto*, i *return*.

4.24.12 Break Naredba

Ponekad vam može zatrebati da zaustavite petlju iz njenog tela. Koristite *break* naredbu u okviru petlje da predate kontrolu prvoj naredbi koja prati najdublji *switch*, *for*, *while*, ili *do* blok.

Break se obično koristi u *switch* naredbama da spreči daljnje izvršavanje posle prvog poklapanja. Na primer:

```
switch (stanje) {  
  case 0: Nisko(); break;  
  case 1: Srednje(); break;  
  case 2: Visoko(); break;  
  default: Poruka("Nevažeće Stanje!");  
}
```

4.24.13 Continue Naredba

Naredbu *continue* možete koristiti u okviru petlji (*while*, *do*, *for*) da „preskočite krug“. Ona prosleđuje kontrolu na kraj najdublje ograđujuće zagrade koja pripada konstrukciji petlje. U ovoj tački se uslov nastavljanja petlje ponovo preispituje. Ovo znači da *continue* zahteva sledeću iteraciju ako je uslov nastavka petlje ispunjen.

4.24.14 Goto Naredba

Koristite *goto* naredbu za bezuslovni skok na lokalnu labelu – za više informacija o labelama pogledajte Naredbe sa Labelama. Sintaksa *goto* naredbe je:

```
goto identifikator_label;
```

Ova naredba će proslediti kontrolu lokaciji lokalne labele određene sa *identifikator_label*. *Identifikator_label* mora biti ime labele u okviru iste funkcije gde je i *goto* naredba. *Goto* linija može doći i pre i posle labele.

Možete koristiti *goto* da izađete iz bilo kojeg nivoa ugnježdene kontrolne strukture, ali *goto* ne može biti korišteno za skokove u blok a da se preskoče inicijalizacije tog bloka – na primer skakanje u telo petlje itd.

Korištenje *goto* naredbe se generalno ne ohrabruje pošto praktično svaki algoritam može biti realizovan i bez nje, što rezultira čitljivijim strukturalnim programima. Jedna moguća primena *goto* naredbe je izlaz iz duboko ugnježđenih kontrolnih struktura:

```
for (...) {  
for (...) {  
...  

```

if (katastrofa) **goto** Greska;

```
...  
}  
}  
.  
.  
.
```

Greska: */* Kod za obradu greske */*

4.24.15 Return Naredba

Koristite *return* naredbu da izađete iz trenutne funkcije nazad u pozivnu rutinu, opcionalno vraćajući neku vrednost. Sintaksa je:

return [*izraz*];

Ovo će izračunati *izraz* i vratiti rezultat. Vraćena vrednost će biti automatski konvertovana u funkcijin očekivani tip, ako je to potrebno. *Izraz* je opcionalan, ako se izostavi funkcija će vratiti slučajnu vrednost iz memorije.

Primer: *return* naredba u funkcijama void tipa ne može imati *izraz*, u stvari možete potpuno izostaviti *return* naredbu ako je ona poslednja naredba u telu funkcije.

4.24.16 Složene Naredbe (Blokovi)

Složena naredba, ili blok, je lista (moguće i prazna) naredbi, ogradenih velikim zagradama { }. Sintaksno, blok se može posmatrati kao pojedinačna naredba, ali on takođe igra ulogu u određivanju dosega identifikatora. Identifikator deklarisan u okviru bloka ima doseg koji počinje u tački deklaracije i završava se sa zatvorenom velikom zagradom. Blokovi mogu biti ugnježdjeni do proizvoljne dubine, do memorijskog ograničenja.

Na primer, *for* petlja očekuje jednu naredbu u svom telu tako da joj možemo proslediti složenu naredbu:

```
for (i = 0; i < n; i++) {  
  int temp = a[i];  
  a[i] = b[i];  
  b[i] = temp;  
}
```

Primetite da, za razliku od drugih naredbi, složene naredbe ne završavaju tačkom-zarez(;), odnosno nema tačke-zarez iza desne velike zagrade.

4.25 Pred-processor

Pred-processor je integrisani tekst procesor koji priprema izvorni kod za kompajliranje. Pred-processor omogućava:

- Umetanje teksta iz specifikovanog fajla u određenu tačku u kodu
- Zamenu određenih leksičkih simbola drugim simbolima

- Uslovno kompajliranje, koje uslovno uključuje i izostavlja delove koda.

Primitite da pred-procesor analizira tekst an nivou tokena a ne pojedinačnih znakova. Pred-procesor se kontroliše pomoću pred-procesorskih direktiva i pred-procesorskih operatora.

4.25.1 Pred-procesorske Direktive

Bilo koja linija koda koja počinje sa # se uzima kao pred-procesorska direktiva (ili kontrolna linija), osim ako # nije u okviru string literala, znakovne konstante, ili ugrađeno u komentar. Početnom # može prethoditi prazan prostor (osim novih linija).

Null (*prazna)direktiva se sastoji od jedne linije koja sadrži samo znak #. Ova linija se uvek ignoriše.

Pred-procesorske direktive se obično smeštaju na početku izvornog koda, ali ih je ispravno pisati i u bilo kojoj drugoj tački programa. MikroC pred-procesor detektuje pred-procesorske direktive i obrađuje tokene koji su u njima ugrađeni. Direktiva je na snazi od njene deklaracije do kraja programskog fajla.

MikroC podržava standardne pred-procesorske direktive:

# (null directive)	#if
#define	#ifndef
#elif	#ifndef
#else	#include
#endif	#line
#error	#undef

Primedba: *#pragma* direktiva je u fazi izrade.

4.25.2 Nastavak linije sa Kosom Crtom Unazad

Ako treba da razlomite direktivu u više linija, možete to da uradite okončanjem linije kosom crtom unazad (\):

```
#define MACRO Ova direktiva se nastavlja\
```

U sledeću liniju.

4.25.3 Makroi

Makroi pružaju mehanizam za zamenu tokena, pre kompajliranja, sa ili bez skupa formalnih parametara sličnih funkcijskim.

4.25.4 Definisanje Makroa i Makro Proširenja

#define direktiva definiše makro:

```
#define identifikator_makroa <sekvenca_tokena>
```

Svako pojavljivanje *identifikatora_makroa* u izvornom kodu koji dolazi posle ove kontrolne linije će biti zamenjeno, na prvobitnoj poziciji sa moguće i praznom *sekvencom_tokena* (postoje neki izuzetci koji će biti pomenuti kasnije). Takve zamene su takođe poznate kao makro proširenja. *Sekvenca_tokena* se ponekad naziva i telom makroa. Prazna sekvenca tokena rezultira uklanjanjem svakog odnosnog makro identifikatora iz izvornog koda.

Tačka-zarez nije potrebna za okončanje pred-procesorske direktive. Bilo koji znak koji se nađe u sekvenci tokena, uključujući tačku-zarez, će se pojaviti u makro proširenju.

Sekvenca_tokena se završava u prvoj novoj liniji na koju se naiđe (ako nije korištena kosa crta unazad). Bilo koja sekvenca praznih znakova, uključujući komentare, u *sekvenci_tokena* se zamenjuje jednim znakom razmaka.

Nakon svakog pojedinačnog makro proširenja, pravi se još jedan pregled novo-proširenog teksta. Ovo omogućava prepoznavanje ugnježđenih makroa: Prošireni tekst može sadržavati makro identifikatore koji su podložni zameni. Kakogod, ako se makro proširi u nešto što će izgledati kao pred-procesorska direktiva, pred-procesor neće prepoznati takvu direktivu. Makro identifikatori koji se nađu unutar literalnih stringova, znakovnih konstanti, ili komentar u izvornom kodu, se ne proširuju.

Makro neće biti proširen tokom svog sopstvenog proširenja (tako da se `#define MACRO MACRO` neće proširivati u nedogled).

Pogledajmo primer:

```
/*Ovde su neki primeri makroa: */
#define ERR_MSG "Van opsega!"
#define EVERLOOP for( ; ; )
/* što bi mogli da koristimo na sledeći način: */
main() {
    EVERLOOP {
    ...
    if (error) {Lcd_Out_Cp(ERR_MSG); break;}
    ...
    }
}
```

Pokušaj da se predefiniše već definisan makro identifikator će rezultirati upozorenjem ukoliko nova definicija nije tačno ista, token po token kao postojeća. Najbolja strategija u slučaju da bi definicije mogle postojati u drugim fajlovima zaglavlja, je sledeća:

```
#ifndef BLOCK_VELOCINA
#define BLOCK_VELOCINA 512
#endif
```

Srednja linija se obilazi u slučaju da je `BLOCK_VELOCINA` trenutno definisana. Ako `BLOCK_VELOCINA` nije trenutno definisana, poziva se srednja linija da je definiše.

4.25.5 Makroi sa Parametrima

Sledeća sintaksa se koristi za definisanje makroa sa parametrima:

```
#define makro_identifikator(<arg_lista>) sekvenca_tokena
```

Primetite da ne sme biti praznog prostora između *makro_identifikatora* i „(„. Opcionalna *arg_lista* je sekvenca identifikatora razdvojena zarezima, ne mnogo različita od liste argumenata C funkcije. Svaki identifikator razdvojen zarezom igra ulogu formalnog argumenta ili kalupa.

Ovakvi makroi se pozivaju pisanjem

```
makro_identifikator(<lista_stvarnih_arg>)
```

u izvornom kodu koji sledi. Sintaksa je identična sintaksi poziva funkcije; i zaista, mnoge „funkcije“ standardne C biblioteke su implementirane kao makroi. Kakogod, postoje neke važne semantičke razlike.

Opcionalna *lista_stvarnih_arg* mora sadržavati isti broj token sekvenci razdvojenih zarezom, znanih kao stvarni argumenti, kao što je nađeno u formalnoj *arg_listi #define* linije – mora postojati jedan stvarni argument za svaki formalni argument. Ako je broj argumenata u dve liste različit, biće prijavljena greška.

Rezultat poziva makroa su dva skupa zamena. Prvo se makro identifikator i argumenti unutar zagrada zamenjuju sekvencom tokena. U sledećem koraku, bilo kakvi formalni argumenti koji se pojavljuju u sekvenci tokena se zamenjuju odgovarajućim stvarnim argumentima koji se nalaze u *lista_stvarnih_arg*. Kao i sa običnim makro definicijama, ponovo se prolazi kroz tekst da bi se otkrili bilo kakvi ugrađeni makro identifikatori podložni proširenju.

Sledi jednostavan primer:

```
// jedan jednostavni makro koji vraća veći od svoja 2 argumenta:
```

```
#define _MAX(A, B) ((A) > (B)) ? (A) : (B)
```

```
// Pozovimo ga:
```

```
x = _MAX(a + b, c + d);
```

```
/* Pred-procesor će transformisati prethodnu liniju u:
```

```
x = ((a + b) > (c + d)) ? (a + b) : (c + d) */
```

Preporučuje se stavljanje malih zagrada oko svakog argumenta u telu makroa – ovo omogućava izbegavanje mogućih problema sa prvenstvom operatora.

4.25.6 Poništavanje Definicija Makroa

Možete poništiti definiciju makroa korištenjem *#undef* direktive.

#undef makro_identifikator

Direktiva *#undef* razdvaja bilo kakvu prethodnu sekvencu tokena od *makro_identifikatora*, definicija makro biva zaboravljena, a *makro_identifikator* je nedefinisan. Nikakvo proširenje makroa se ne dešava u okviru *#define* linija.

Stanje bivanja definisan i nedefinisan je važno obeležje jednog identifikatora bez obzira na stvarnu definiciju. *#ifdef* i *#ifndef* uslovne direktive, koje se koriste za proveravanje da li je neki identifikator trenutno definisan ili nije, nude fleksibilan mehanizam za kontrolu mnogih aspekata kompajliranja. Pošto je poništena definicija nekog makroa, on može biti predefinisan sa *#define*, korištenjem iste ili različite sekvence tokena.

4.25.7 Uključivanje Fajla

Pred-procesorska direktiva *#include* uvlači fajlove zaglavlja (ekstenzija *.h*) u izvorni kod. Ne oslanjajte se na pred-procesor za uključivanje izvornih fajlova (ekstenzija *.c*) – pogledajte Projekte za više informacija.

Sintaksa *#include* direktive ima dva formata:

```
#include <ime_zaglavlja>
```

```
#include "ime_zaglavlja"
```

Pred-procesor odstranjuje `#include` liniju i menja je celokupnim tekstom fajla zaglavlja u toj tački izvornog koda. Postavljanje `#include` stoga može uticati na doseg i trajanje bilo kojih identifikatora u uključenom fajlu.

Razlika između dva formata leži u algoritmu traženja koji se koristi u pokušaju lociranja fajla koji se uključuje.

Ako se `#include` direktiva koristi sa `<ime_zaglavlja>` verzijom, pretraga se vrši sukcesivno na svakoj od sledećih lokacija, ovim redom:

1. mikroC instalacioni folder > “include” folder,
2. vaše posebno definisane putanje traženja

Verzija “`ime_zaglavlja`” određuje fajl za uključivanje koji je obezbedio korisnik; mikroC će tražiti fajl zaglavlja na sledećim lokacijama, ovim redom:

1. projektni folder (folder koji sadrži fajl projekta .ppc)
2. mikroC instalacioni folder > “include” folder
3. vaše posebno definisane putanje traženja

4.25.8 Eksplicitna Putanja

Ako stavite eksplicitnu putanju u `ime_zaglavlja`, samo taj direktorijum će biti pretraživan. Na primer:

```
#include "C:\my_files\test.h"
```

Primerba: Postoji takođe i treća, retko korištena verzija `#include` direktive, koja predpostavlja da se ni `<ni>` ne pojavljuju kao prvi ne-prazan znak koji prati `#include`:

```
#include makro_identifikator
```

Ona pretpostavlja da postoji makro definicija koja će proširiti makro identifikator u validno označeno ime zaglavlja u ili `<ime_zaglavlja>` ili “`ime_zaglavlja`” formatu.

4.25.9 Pred-procesorski Operatori

`#` (znak „taraba“) je pred-procesorska direktiva onda kada se pojavi kao prvi ne-prazan znak u nekoj liniji. Takođe, `#` i `##` izvode zamenu i spajanje operatora tokom pred-procesorove faze skeniranja.

4.25.10 Operator

Kod C pred-procesora, sekvenca znakova uokvirena navodnim znacima se smatra tokenom i njen sadržaj se ne analizira. Ovo znači da se makro imena unutar navodnika ne proširuju.

Ako vam treba stvarni argument (tačna sekvenca znakova unutar navodnika) kao rezultat pred – procesuiranja, možete koristiti `#` operator u telu makroa. On može biti postavljen ispred formalnog argumenta makroa u definiciji, sa ciljem konvertovanja stvarnog argumenta u string nakon zamene.

Na primer, napravimo makro `LCD_PRINT` za štampanje imena i vrednosti promenljive na LCD-u.

```
#define LCD_PRINT(val) Lcd_Out_Cp(#val ": "); \
Lcd_Out_Cp(IntToStr(val));
```

(obratite pažnju na kosu crtu unazad kao simbola za nastavak linije)

Sada će sledeći kod,
LCD_PRINT(temp)
biti pred-procesuiran na ovo:
Lcd_Out_Cp("temp" ": "); Lcd_Out_Cp(IntToStr(temp));

4.25.11 Operator

Operator **##** se koristi za spajanje tokena. Dva tokena možete spojiti smeštanjem **##** između njih (plus opcionalni prazan znak na obe strane). Pred-procesor odstranjuje prazan znak i **##**, kombinujući odvojene tokene u jedan novi token. Ovo se obično koristi za konstrukciju identifikatora.

Na primer, možemo definisati makro SPLICE za spajanje dva tokena u jedan identifikator:

```
#define SPLICE(x,y) x ## _ ## y
```

Sada sepoziv SPLICE(cnt, 2) proširuje na identifikator cnt_2.

Primer: mikroC ne podržava stariji ne-portabilni metod spajanja tokena korištenjem (1/**/r).

4.25.12 Uslovno Kompajliranje

Direktive uslovnog kompajliranja se tipično koriste da naprave izvorni program lakšim za izmene i kompajliranje u različitim okruženjima izvršavanja. MikroC podržava uslovno kompajliranje zamenog odgovarajućih linija izvornog koda praznim linijama.

Sve direktive uslovnog kompajliranja moraju biti završene u izvornom ili uključenom fajlu u kojem su počele.

4.25.13 Direktive #if, #elif, #else, i #endif

Uslovne *direktive #if, #elif, #else, i #endif* funkcionišu veoma slično običnim C uslovnim naredbama. Ako izraz koji pišete iza **#if** ima vrednost različitu od nule, grupa linija koja neposredno prati **#if** direktivu je zadržana u prevodilačkoj jedinici.

Sintaksa je:

```
#if konstantni_izraz_1  
<odeljak_1>  
[#elif konstantni_izraz_2  
< odeljak _2>]  
...  
[#elif konstantni_izraz _n  
< odeljak _n>]  
[#else  
<finalni_ odeljak >]  
#endif
```

Svaka **#if** direktiva u izvornom fajlu mora biti uparena sa okružujućom **#endif** direktivom. Proizvoljan broj **#elif** direktiva se može pojaviti između **#if** i **#endif** direktiva ali je dozvoljena najviše jedna **#else** direktiva. **#else** direktiva, ako je prisutna, mora biti poslednja direktiva pre **#endif**.

Odeljci mogu biti bilo koji programski tekst koji ima nekog smisla kompajleru pred-procesora. Pred-procesor odabira pojedinačni odeljak izračunavanjem *konstantnog_izraza* koji prati svaku *#if* ili *#elif* direktivu, dok ne nađe istinit (različit od nule) konstantni izraz. *Konstantni_izraz* je podložan makro proširenju.

Ako su sva pojavljivanja konstantnog izraza neistinita, ili ako se ne pojavi nijedna *#elif* direktiva, pred-procesor odabira tekst blok iza *#else* naredbe. Ako je *#else* naredba izostavljena i sve instance *konstantnog_izraza* u *#if* bloku neistinite, nijedan odeljak se ne odabire za daljnje procesuiranje.

Bilo koji procesuirani *odeljak* može sadržavati daljnje uslovne naredbe, ugnježdene po dubini. Svaka ugnježdena *#else*, *#elif*, ili *#endif* direktiva, pripada najbližoj prethodnoj *#if* direktivi.

Zbirni rezultat prethodnog scenarija je da samo jedan *odeljak* koda (moguće i prazan) biva kompajliran.

4.25.14 Direktive *#ifdef* i *#ifndef*

Možete koristiti *#ifdef* i *#ifndef* direktive bilo gde gde se može koristiti *#if*. *#ifdef* i *#ifndef* uslovne direktive vam omogućavaju da proverite da li je neki identifikator trenutno definisan ili ne. Linija

#ifdef identifikator

ima isti efekat kao i *#if 1* ako je *identifikator* trenutno definisan, i isti efekat kao *#if 0* ako je *identifikator* trenutno nedefinisan. Druga direktiva, *#ifndef*, proverava istinitost „nedefinisanog“ uslova, proizvodeći suprotan rezultat.

Sintaksa prati sintaksu *#if*, *#elif*, *#else*, i *#endif*.

Identifikator definisan kao NULL se smatra definisanim.

135

Organizacija MPLAB razvojnih alata i funkcija je u vidu padajućih menija. Takođe, moguće je definisati i menjati rasporede tastera za pokretanje pojedinih opcija. MPLAB dozvoljava korisniku da:

- Asemblira, prevodi i linkuje izvorni kod,
- Debugira izvršni kod posmatrajući tok programa sa simulatorom ili koristeći MPLAB-ICE emulator u realnom vremenu,
- Realizuje merenje vremenskih intervala,
- Posmatra promenjive u *Watch* prozoru,
- Realizuje industrijske verzije aplikacije (*firmware*) sa PICSTART Plus ili PRO MATE II
- Pronalazi brze odgovore na pitanja iz MPLAB *on-line Help*-a

5.1 MPLAB – Razvojno okruženje (IDE)

MPLAB predstavlja integrisano razvojno okruženje (IDE) koje je lako za učenje. Ovo okruženje omogućava razvojnim inženjerima fleksibilnost u razvoju i debugiranju *firmware* proizvođača za *Microchip*-ovu *PICmicro* porodicu procesora. MPLAB IDE radi već na *Windows 3.1x* i podržava sve moderne verzije *Windows* operativnih sistema. Ne funkcionišu sve hardverske komponente pod MPLAB IDE (kao na primer emulatori i programatori uređaja) pod svim operativnim sistemima. Detalji o izuzecima se mogu naći u korisničkom uputstvu za specifičan hardverski uređaj.

MPLAB omogućava funkcije koje dozvoljavaju:

- Kreiranje i editovanje izvornog fajla,
- Grupisanje fajlova u projekte,
- Debugiranje izvornog koda,
- Debugiranje izvršnog koda koristeći simulator i emulator.

MPLAB IDE omogućava kreiranje izvornog koda uz korišćenje svih opcija i mogućnosti koje pruža ugrađeni ekranski editor. Dalje, korisnik ima mogućnost da debugira izvorni kod i da u prozoru *Bulid results* prati greške koje je generisao prevodilac, assembler ili linker.

Projekat Manager omogućava korisniku da grupiše izvorne datoteke, prekompajlirane objektna fajlove, biblioteke i *script* datoteke koje generiše linker u formatu projekata. MPLAB IDE takođe omogućava simulator sa bogatim izborom opcija i okruženja emulatora za debugiranje izvršnog koda.

Veliki broj različitih varijanti prozora omogućavaju pogled na sadržaj podataka i memorijskih lokacija. Izvorni kod, programska memorija i listing programa dozvoljava pogled na izvorni kod simultano sa njegovim ekvivalentom na asemblerskom nivou.

Standardne opcije debugiranja koje su podržane u MPLAB IDE su: postavljanje prekidnih tačaka, *Trace* opcija, standardne i prekidačke prekidne tačke.

5.2 MPLAB razvojni alati

MPLAB IDE integriše nekoliko alata koji omogućavaju realizaciju kompetnog razvojnog okruženja:

5.2.1 MPLAB Project Manager

Koristi se *Project Manager* za kreiranje projekta i rad sa specifičnim datotekama vezanim za projekat. Korišćenjem projekta, izvorni kod je izgrađen i učitao u simulator i emulator jednim pritiskom tastera.

5.2.2 MPLAB Editor

Koristi se MPLAB Editor za kreiranje i editovanje tekst datoteka kao izvornih datoteka, koda i *skrip* datoteka linkera.

5.2.3 MPLAB-SIM Simulator

Softverski simulator modeluje izvršavanje instrukcija i U/I *PICmicro* mikrokontrolera (MCUs).

5.2.4 MPLAB-ICE Emulator

MPLAB-ICE emulator koristi hardver za emuliranje PIC mikroprocesora u realnom vremenu sa ili bez ciljnog sistema.

5.2.5 MPLAB-ICD Debager

MPLAB-ICD je programer za PIC16F87X porodicu procesora a takođe predstavlja *in-circuit* debager. On prebacuje *hex* datoteke na PIC16F87X i nudi osnovne osobine kao što je izvršavanje koda u realnom vremenu, izvršavanje korak po korak i definisanje prekidnih tačaka.

5.2.6 MPASM univerzalni assembler/MPLINK relokabilni linker/MPLIB menadžer biblioteka

MPASM assembler omogućava da se izvorni kod asembliira bez napuštanja integrisanog okruženja. MPLINK kreira izvršni kod aplikacije povezujući relokabilne module iz MPASM i MPLAB-C17/C18 biblioteke. MPLIB upravlja i korisnički definisanim bibliotekama čime postiže maksimalnu fleksibilnost.

5.2.7 MPLAB-C17/C18 C kompajleri

MPLAB-C17/C18 C kompajleri omogućavaju ANSI-bazirana programska rešenja, za programske jezike visokog nivoa. U osnovnoj verziji podržane su PIC17CXXX i PIC18CXXX serije procesora. Kompleksni projekti mogu koristiti kombinaciju programskog jezika C i assemblera u istoj izvornoj datoteci za maksimalne efekte kako u pogledu efikasnosti tako i u pogledu lakoće pisanja programa i fleksibilnosti.

5.2.8 PRO MATE II i PICSTART Plus Programatori

U toku razvoja programa koriste se simulator i emulator, assembler ili kompajler i neki od alata za programiranje uređaja. Ovo može biti izvedeno korišćenjem mogućnosti samog MPLAB-a. Iako PRO MATE II ne zahteva MPLAB kao bi funkcionisao, programiranje fizičkih procesora je ipak lakše korišćenjem MPLAB-a.

5.2.9 PICMASTER I PICMASTER-CE Emulatori

PICMASTER emulator koristi hardver za emuliranje PIC mikroprocesora u realnom vremenu sa i bez ciljnog sistema. CE kompatibilna verzija procesora je namenjena zemljama Evropske Unije. Takođe, dostupna je i najnovija verzija emulatora: MPLAB-ICE.

5.2.10 Alati treće kategorije

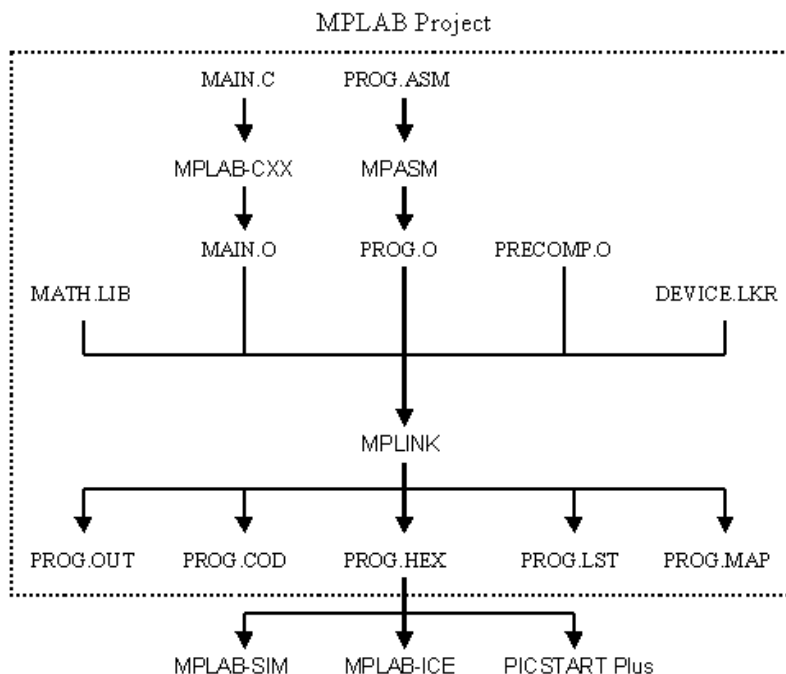
Mnoge od kompanija su razvile alate za *Microchip* proizvode koji rade sa MPLAB okruženjem. Detalji se mogu naći u dokumentu *Microchip Third Party Guide* (DS00104).

5.3 MPLAB pregled opcija menija

Projekat u MPLAB-u je grupa datoteka potrebnih za izgradnju aplikacije, zajedno sa njihovim međusobnim vezama sa različitim alatima. U nastavku su prikazani nivoi u fazi prevođenja projekta. Boldiranim slovima su prikazani programski delovi sistema MPLAB.

- Izvorne datoteke,
- **Assembler-kompajler,**
- Objektne datoteke,
- Biblioteke i linkerski *script* fajlovi,
- **Linker,**
- Izlazne datoteke,
- **Simulator,**
- **Emulator,**
- **Programator.**

Na sledećoj slici je prikazana ogovarajuća hijerarhijska struktura generičkog MPLAB projekta:



Slika 5.1 - Generički MPLAB projekat – Datoteke koje su deo projekta.

U ovom MPLAB projektu, C izvorni kod (*main.c*) je asociran sa MPLAB-CXX prevodiocem. Asemblerski izvorni program (*prog.asm*) je na slici prikazan u vezi sa assemblerom MPASM. MPLAB koristi ove informacije da generiše objektnu datoteku (*main.o*) kao ulaz u linker (MPLINK).

Prekompajlirani objektni fajl (*precomp.o*) može biti uključen u projekat. Tipovi pre-kompajliranih objektnih datoteka koje generalno zahteva projekat su:

- Startni kod,
- Inicijalni kod,
- Interrapt servis rutine,
- Definicije funkcija specijalnih registara.

Pre-kompajlirane objektno datoteke su često zavisne od ciljnog uređaja i u funkciji od primenjenog memorijskog modela. Neke od bibliotečkih datoteka, na primer *math.lib* su raspoložive zajedno sa kompajlerom. Ostale se moraju izgraditi korišćenjem sistemskog programa MPLIB koji služi za generisanje biblioteka. Objektno datoteke, zajedno sa script fajlovima linkera (*device.lkr*) su korišćene za generisanje izlaznih datoteka linkera (MPLINK).

Glavna izlazna datoteka generisana od strane MPLINK-a je *hex* datoteka (*prog.hex*) koji koristi simulator (MPLAB-SIM), emulator (MPLAB-ICE i PICMASTER) i programatori (PRO MATE II i PICSTART Plus). Ostale izlazne datoteke su:

- COFF datoteka (*.out*). U pitanju je među-datoteka koju koristi MPLINK za generisanje *code* datoteke, *hex* datoteke i *listing* datoteke.
- Datoteka koda (*.cod*). *Debug* datoteka se koristi u MPLAB.
- Listing datoteka (*.lst*). Originalni izvorni kod zajedno sa izlaznim binarnim kodom.
- *Map* datoteka (*.map*). Pokazuje na strukturu memorije posle završetka procesa linkovanja. Vršiti indikaciju neiskorišćenih područja memorije.

Prikazani alati predstavljaju sve komponente koje dolaze zajedno sa *Microchip*-ovim razvojnim sistemom. Uz ove alate su dostupni i dodatni razvojni alati nezavisnih proizvođača (DS00104).

5.3.1 Struktura opcija u sistemu menija MPLAB-a

U nastavku ćemo prikazati strukturu opcija koje sadrži MPLAB. Uz originalni naziv koji je prisutan u programu ukratko ćemo dati objašnjenje funkcije odgovarajuće opcije.

5.3.2 Editing Project Information – Meni za editovanje projektnih informacija

Korišćenjem dijaloga *Edit Project* postavljamo opcije tekućeg projekta.

- *Opening an existing project* – otvaranje tekućeg projekta,
- *Entering the target filename* – ubacivanje imena ciljne datoteke,
- *Setting the include path, library path, or linker script path* – postavljanje direktorijuma za uključivanje datoteka, putokaza do biblioteka, linker *script* putokaza,
- *Setting the development mode* - postavljanje razvojnog moda,
- *Selecting the language tool suite* - postavljanje izvornog jezika,
- *About project files* - informacije o projektnim datotekama,
- *Adding a node to a project* - dodavanje čvora projektu,
- *Creating a new node based on an existing node* - kreiranje novog čvora baziranog na postojećem čvoru,
- *Deleting a node* - brisanje čvora,
- *Building a node* - izgradnja čvora,
- *Setting a node's properties* - postavljanje osobina čvora.
- *Creating and Building an MPLAB Project* - kreiranje i izgradnja MPLAB projekta.

Da bi se kreirao projekat potrebno je slediti korake koji su dati u nastavku:

1. *Select the language tool* - biranje izvornog jezika,
2. *Set the development mode* - postavljanje razvojnog moda,
3. *Start a new project* - pokretanje novog projekta,
4. *Check development mode and tool settings* - provera razvojnog moda i lista izabranih alata.

Informacije u vezi projekta:

- *Set the include path* - postavljanje putokaza na *include* datoteke,
- *Set node properties for the target node* - postavljanje osobina čvora za ciljni čvor,
- *Set the library path* - postavljanje putokaza ka bibliotekama,
- *Set the linker script path* - postavljanje putokaza na linkerove *script* datoteke.

Dodavanje datoteka projektu:

- *Add the source file(s)* - Dodavanje izvorne datoteke,
- *Set node properties for the source file(s)* - postavljanje osobina čvora za izvornu datoteku,
- *Add object files* - dodavanje objektnih datoteka,
- *Add library files* - dodavanje biblioteka,
- *Add the linker script file* - dodavanje linker *script* datoteka,
- *Save the project* - snimanje projekta.

Izgradnja projekta:

- *Build the project* - izgradnja projekta,
- *Troubleshoot the build process* - problemi (greške) u fazi izgradnje procesa.

1. Pokretanje i debugiranje MPLAB projekta

Da bi se pokrenuo i debugirao MPLAB projekat potrebno je slediti sledeće procedure:

- Pokrenuti izvršavanje programa korak po korak,
- Debugirati projekat korišćenjem *Watch* prozora, prekidnih tačaka i *trace* prekidnih tačaka.

2. MPLAB meni i reference funkcija

MPLAB meniji dozvoljavaju pristup MPLAB funkcijama:

- *File* – datoteka,
- *Project* – projekat,
- *Edit* - editovanje teksta,
- *Debug* – debugiranje.

(Meni za programere)

- *Options* – opcije,
- *Tools* – alati,
- *Window* – prozori,
- *Help* – pomoć.

(Ključne tačke)

- *MPLAB-SIM* - Okruženje simulatora,
- *MPLAB Desktop* - MPLAB radni prozor,
- *File extensions used by MPLAB* - ekstenzije datoteka koje se koriste u MPLAB-u.

Meni za datoteke sadrži sledeće opcije:

- *New* - nova datoteka,
- *Open* - otvaranje datoteke,
- *View* – pogled,
- *Save* – snimanje,
- *Save As* - snimanje sa promenjenim imenom,
- *Save All* – snimanje svih podataka,
- *Close* – zatvaranje,
- *Close All* - zatvaranje svih podataka,
- *Import* – izvoz,
- *Export* – uvoz,
- *Print* – štampanje,
- *Print Setup* - podešavanje štampača,
- *Exit* – izlaz,
- *Most-Recently-Used Files* - najviše korišćene datoteke.

5.3.3 Projektni meni

Sledeće opcije su dostupne na projektnom meniju:

- *New Project...* – novi projekat; kreiranje novog projekta,
- *Open Project..* - otvaranje postojećeg MPLAB projekta,
- *Close Project* - zatvaranja postojećeg MPLAB projekta,
- *Save Project* - snimanje MPLAB projekta,
- *Edit Project* - dozvoljavanje podešavanja mnogih karakteristika projekta,
- *Make Project* - omogućava prevođenje datoteka u jedinstvenu *hex* datoteku,
- *Build All* - izgradnja svih čvorova,
- *Build Node* - izgradnja izabranog čvora,
- *Install Language Tool* - dozvoljavanje da konfiguraciona procedura MPLAB-a prepozna alat za razvoj jezika,
- *Most Recently Used Projects* - prikaz najviše korišćenih projekata.

5.3.4 Meni za editovanje

Meni za editovanje sadrži opcije koje dozvoljavaju lako kreiranje i menjanje teksta. Ako nijedna datoteka nije otvorena, opcije menija se prikazuju u sivoj pozadini.

5.3.5 Opcije editora

U nastavku su navedene neke od osnovnih opcija ekranskog editora koji je ugrađen

- *Undo* - **Ctrl+Z** (Vraćanje; poništavanje promene),
- *Cut* - **Ctrl+X** (Brisanje),
- *Copy* - **Ctrl+C** (Kopiranje),
- *Paste* - **Ctrl+V** (Ubacivanje teksta),
- *Select All* - (Izabiranje celokupnog teksta),
- *Select Word* - (Izabiranje jedne reči; dupli klik na levi taster miša),
- *Delete Line* - **Ctrl+Shift+K** (Brisanje linije),
- *Delete EOL* - **Ctrl+K** (Brisanje specijalnog karaktera koji predstavlja kraj linije),
- *Goto Line* - **Ctrl+G** (Skok na liniju),
- *Find* - **F3** (Pronalaženje reči),
- *Replace* - **F4** (Zamena reči),
- *Match Brace* - **Ctrl+B** (Poklapanje).
 - *Template Options* – opcije *template* forme,
 - *Text Options* - opcije teksta.
 - *Transpose* - **Ctrl+T** (premesti),
 - *Uppercase* - (velika slova),
 - *Lowercase* - (mala slova),
 - *Indent* - (uvlačenje teksta, tabulacija),
 - *Unindent* - (vraćanje tabulatora nazad).

5.3.6 Meni debugiranja

Posle postavljanja i kompajliranja projekta u MPLAB-u, potrebno je pokrenuti izvršni program. Ako uz sistem postoji i programator onda je moguće programirati mikrokontroler direktno na ciljnom uređaju ili ga posle programiranja prebaciti na aplikaciju i proveriti kako program funkcioniše u realnim uslovima. Najverovatnije aplikacija neće odmah da radi kako treba, pa je zato potrebno proveriti njeno funkcionisanje uz pomoć debagera.

Moguće je koristiti MPLAB-SIM za simuliranje koda ili koristiti MPLAB-ICE emulator za pokretanje konačne *firmware* aplikacije. Najčešće će biti potrebno koristiti *break* ili *trace* prekidne tačke u fazi analize koda. Stalno je potrebno pratiti vrednosti registara u *Registar* prozoru ili specijalni funkcionalni registar prozor za pamćenje stanja procesora u toku izvršavanja programa u kontinuitetu ili u režimu korak po korak.

MPLAB-ICE emulator pokreće kod u realnom vremenu na ciljnom hardveru zaustavljajući se jedino u prekidnim tačkama. MPLAB-SIM simulira izvršavanje PIC mikroprocesora kao i U/I uslove na brzinama koje zavise od PC-ja na kome radi emulator. Sledeće *debug* funkcije rade u isto vreme sa simulatorom i emulatorom.

Osnovne funkcije emulatora su:

- *Emulation Memory (Program Memory Window)* - emuliranje memorije,
- *Break and trace points* – postavljanje prekidnih tačaka,
- *Single-stepping* - izvršavanje programa korak po korak,

- *Register monitoring* – resetovanje monitoringa programa.

Sve ove funkcije koriste informacije iz MPLAB projekta. Brojevi linija u izvornom kodu, simboličke lokacije u memoriji, imena funkcija iz koda koje mogu biti korišćene za postavljanje prekidnih tačaka ili ispitivanje i modifikovanje registara.

Debug meni sadrži sve opcije koje se koriste u fazi debugiranja koda za vreme simulacije i emulacije:

- *Run Menu* - meni za pokretanje programa,
- *Execute* - izvršavanje programa,
- *Simulator Stimulus* - stimuliranje ulaza za simulator,
- *Center Debug Location* - centralna *debug* memorijska lokacija,
- *Breakpoint Settings* - postavljanje prekidnih tačaka,
- *Trace Point Settings* - postavljanje *trace* kontrole,
- *Trigger In/Out Settings* - uključivanje-isključivanje podešavanja,
- *Trigger Out Point Settings -II-*
- *Clear All Points* - čišćenje svih prekidnih tačaka,
- *Complex Trigger Settings* - postavljanje složenih varijanti kontrolnih prekidača,
- *Enable Code Coverage* - omogućavanje praćenja koda,
- *Clear Program Memory (Ctrl+Shift+F2)* - brisanje programske memorije,
- *System Reset (Ctrl+Shift+F3)* - sistemski reset,
- *Power-On-Reset (Ctrl+Shift+F5)* - *power-on* reset.

5.3.7 Meni za kontrolu rada programatora:

Da bi se selektovao programator, izabere se opcija *Options>Programmer Options>Select Programmer*.

Posle izabiranja programatora, MPLAB će prikazati *warning* poruku i automatski se isključiti. Zbog toga se mora restartovati MPLAB pre nego što opcije programatora postanu dostupne.

Kada se restartuje MPLAB, meni koje je povezan sa specifičnim programatorom će se pojaviti na listi opcija u glavnom meniju.

U sistemu postoje još sledeće opcije:

- **PICSTART Plus Meni**
- **PRO MATE Meni**

Obe opcije su orijentisane na podršku sistemskim programima.

5.3.8 Options Meni

Meni *Options* sadrži različite opcije u vezi konfiguracije sistema:

- *Development mode* - razvojni mod,
- *Window setup* - *windows* podešavanje,
- *Current editor settings* - tekuće podešavanje editora,

- *Reset editor modes* - resetovanje modova editora,
- *Environment setup (Ctrl+F7)* - podešavanje okoline,
- *Programmer options* - opcije programatora.

5.3.9 Tools meni

Opcije iz Tool menija dozvoljavaju pokretanje DOS programa, konfiguraciju emulatora i verifikaciju pravilnog rada emulatora .

- *DOS Command to Window (F11)* - pokretanje DOS prompta,
- *Repeat DOS Command to Window (Ctrl+F11)* - ponavljanje DOS komande u prozoru,
- *Verify PICMASTER* - verifikacija PICMASTER-a,
- *Verify MPLAB-ICE* - verifikacija MPLAB-ICE.

5.3.10 Window Meni

MPLAB omogućava praćenje programske memorije, steka i sadržaja registara. Sve *Windows* opcije su podržane i u modu simulatora i u modu emulatora. U *Editor Only* modu *Absolute Listing* i *Show Symbol List* su podržane.

Opcije koje su na raspolaganju u ovom delu su:

- *Program Memory* – programska memorija,
- *Trace Memory* - trace memorija,
- *EEPROM Memory* - EEPROM memorija,
- *Calibration Data* - kalibracija podataka,
- *Absolute Listing* - apsolutni listing,
- *Map File* - map fajl,
- *Stack* – kontrola steka,
- *File Registers* - sadržaj fajl registara,
- *Special Function Registers* - specijalni funkcijski registri,
- *Show Symbol List (Ctrl+F8)* - pokaži listu simbola,
- *Stopwatch* - zaustavljanje praćenja sadržaja registara,
- *Project Window* - projektni prozor,
- *Watch Windows* - watch prozor,
- *Modify* – modifikacija,
- *Tile Horizontal* – horizontalni pomeraj,
- *Tile Vertical* – vertikalni pomeraj,
- *Cascade* – kaskadno generisanje prozora,
- *Iconize All* - povezivanje sa ikonama,
- *Arrange Icons* - kreiranje ikona,
- *Open Windows* - otvaranje prozora.

3. Help Meni

U okviru ove opcije omogućeno je korisniku da na veoma lak i efikasan način pristupi informacijama:

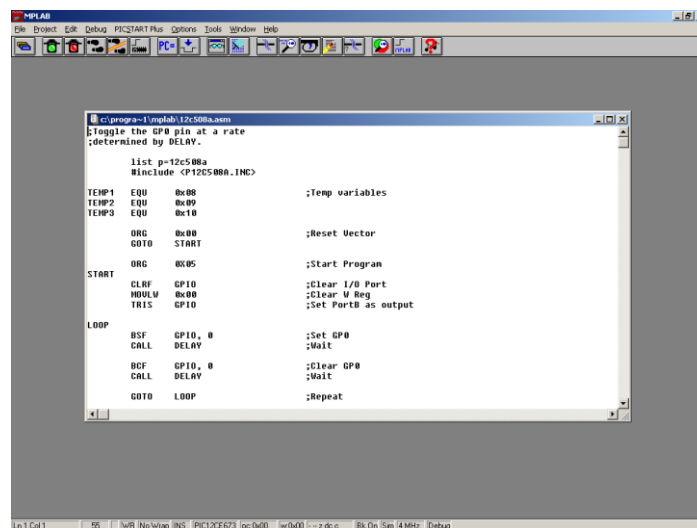
- *Release Notes (Shift+F1)* - beleške o detaljima realizacije programa,
- *Tool Release Notes* - realizacija alata,
- *MPLAB Help* - MPLAB pomoć,
- *Editor Help* - pomoć oko editora,
- *Error Help* - pomoć u pogledu grešaka,
- *MPASM Help* - pomoć u vezi sa assemblerom MPASM,
- *MPLINK Help* - pomoć u vezi sa MPLINK-om,
- *Tool Help* - pomoć u vezi sa alatima,
- *About* - podaci o proizvođaču.

5.3.11 Osnovna procedura razvoja programa u MPLAB razvojnom okruženju

Osnovni ciklus razvoja assemblyskih programa u MPLAB razvojnom okruženju obuhvata proces editovanja, prevođenja, debugiranja i programiranja mikrokontrolera odnosno njegove emulacije. Ovu proceduru razvoja programa prikazaćemo na sledećem primeru:

5.3.12 Editovanje programa

Editovanje programa je procedura unošenja ili menjanja izvornog koda programa u ekranskom editoru, uz korišćenje svih opcija za manipulaciju tekstualnim podacima. U editoru, program je na nivou izvornog (ASCII) koda. Integrisano razvojno okruženje MPLAB ima mogućnost otvaranja više prozora istovremeno. Na sledećoj slici prikazan je jedan prozor sa izvornim kodom programa:



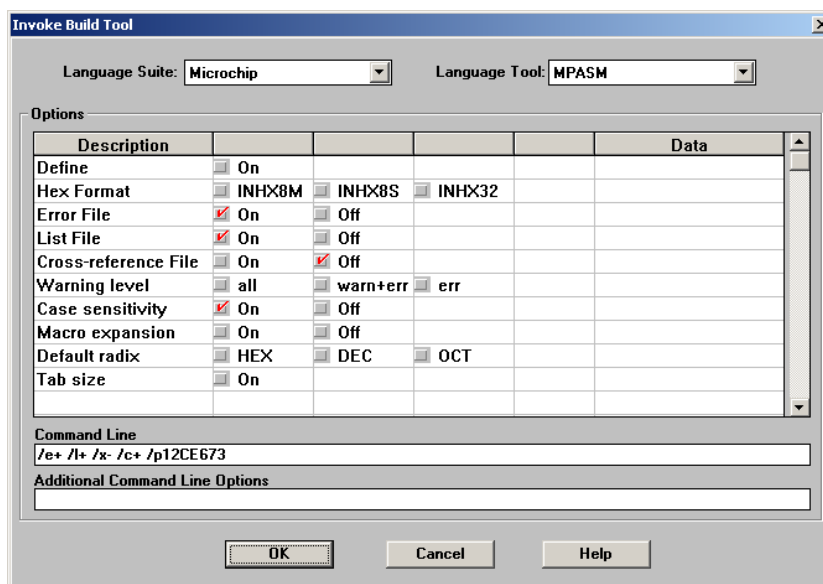
Slika 5.2 - Izvorni kod programa u MPLAB integrisanom okruženju

U fazi razvoja programa moguće je koristiti pogodnosti ugrađenog makroasemblera ili odgovarajućeg prevodioca nekog višeg programskog jezika, na primer programskog jezika C.

5.3.13 Prevođenje programa

Osnovna opcija za prevođenje programa je *Build Node* u *Project* meniju. Prevođenje programa se u svakom trenutku može pokrenuti kombinacijom tastera **Alt+F10**. Imajući u vidu da je čvor (nod) osnovni element, koji može biti hijerarhijski povezan sa drugim strukturama izvornih kodova, proces prevođenja predstavlja u stvari proces izgradnje programa (*Build*), poštujući tu hijerarhijsku organizaciju. Inače, ovaj tip organizacije je dosta korišćen kod viših programskih jezika.

Pokretanjem opcije *BuildNone*, pojavljuje se prozor čiji je izgled prikazan na sledećoj slici:



Slika 5.3 - Izgled i opcije u prozoru posle pokretanja *BuildNode* naredbe (**Alt+F10**)

Podešavanjem mnogobrojnih opcija u ovom prozoru, veoma je jednostavno prilagoditi prevodilac zahtevanim uslovima i okruženju. U skladu sa izabranim opcijama, sistem sam generiše komandnu liniju koja pokreće prevodilac. Pritiskom na opciju OK, pokrenut je postupak automatskog prevođenja programa.

Posle završetka faze prevođenja, sistem generiše prozor izveštaja u kome su izlistane karakteristike grešaka koje su se pojavile u programu. Da bi se otklonile greške koje je prevodilac uočio, potrebno je vratiti se korak unazad u fazu editovanja. Ovaj ciklus: editovanje/prevođenje se ponavlja sve dok više nema kritičnih grešaka (tipa *error*) u ovoj fazi.

Faza prevođenja programa praćena je generisanjem izveštaja u odgovarajućim prozorima, što je pokazano na sledećim slikama:

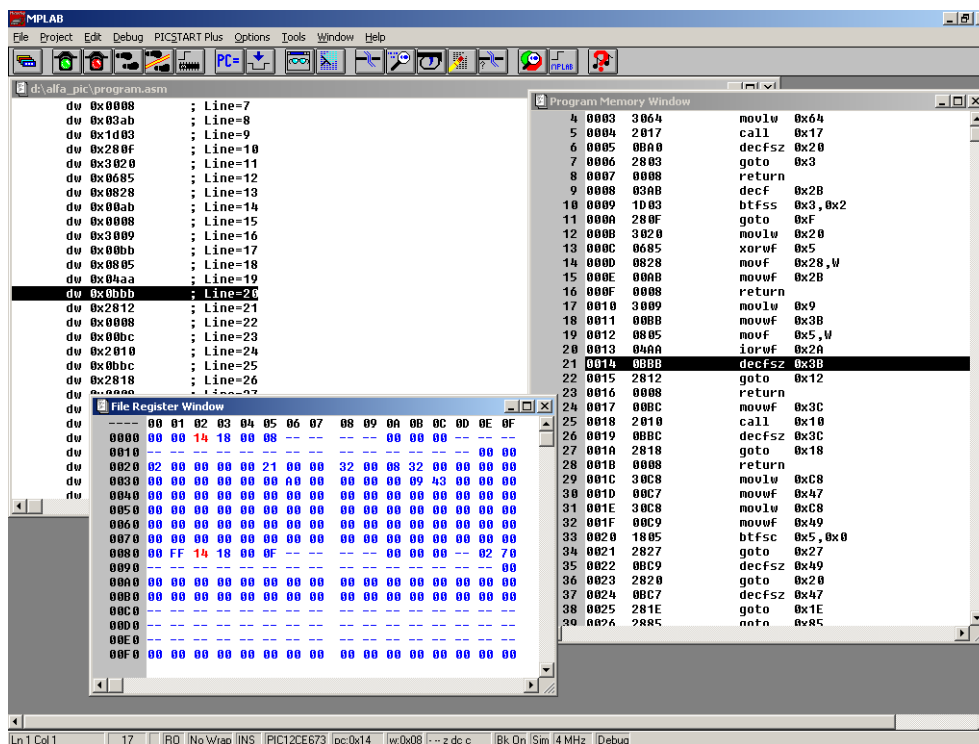
5.3.14 Debugiranje i emulacija rada programa

Debugiranje i emulacija rada programa su procesi koji se obavljaju na programskom nivou u samom razvojnom okruženju sa idejom da se već na tom nivou eliminišu logičke greške kako bi program, kada se upotrebom programatora prebaci na ciljani procesor, bio u potpunosti funkcionalan.

U svrhu debugiranja, u MPLAB razvojnom okruženju predviđeni su su različiti alati za praćenje statusa koda odnosno pojedinih memorijskih lokacija, fajl registara kao i raznih drugih stanja procesora.

Osnovni metod analize oslanja se na upotrebi integrisanog debagera koji sadrži sve standardne opcije koje sistemski alati ovog tipa poseduju. Opcije koje podržavaju rad sa debagerom se nalaze u podmeniju *Debug*.

Osnovni metod analize je takozvani *step* metod koji u osnovi ima mehanizam izvršavanja naredbi korak po korak kroz praćenje stanja procesora u različitim prozorima. Na sledećoj slici je prikazana tipična situacija u faze debugiranja:



Slika 5.6 - Debugiranje programa, izgled osnovnih prozora

Sistem dozvoljava otvaranje različitih prozora u kojima se prate stanja skoro svih komponenti procesora kao što su programska, radna i EEPROM memorija kao i drugi delovi sistema. Na predhodnoj slici prikazana je analiza rada programa korišćenjem podataka koji se nalaze u prozoru sa programskim kodom, prozoru sa disasembliranim naredbama kao i prozoru u kome su prikazana stanja registara.

Korišćenjem opcije *Step* u podmeniju *Debug>Run>Step*, pritiskom na jedan taster, programski brojač se kreće korak po korak kroz izvorni kod, prilagođavajući istovremeno sadržaj svih prozora trenutnom stanju procesora. Na taj način, veoma preglednom analizom se može uočiti niz grešaka u programu i otkloniti ih na licu mesta. Ovo je naročito značajno kod procedura na mašinskom jeziku koje imaju komplikovaniju funkciju, kao na primer procedure za množenje i deljenje celobrojnih numeričkih vrednosti. Kod procedura ovoga tipa, debugiranje programa metodom korak po korak je nezamenljivo.

Ostale opcije iz ove grupe koje se odnose na debugiranje programa su *Run*, *Animate*, *Reset* i *Halt*. Opcija *Run* pokreće emulaciju izvršavanja programa a opcije *Reset* odnosno *Halt* zaustavljaju izvršavanje programa. Opcija *Animate* je naročito interesantna jer pokreće usporeno izvršavanje programa. Ona predstavlja varijantu naredbe *step debug* sa time što je izvršavanje automatizovano – nije potrebno stalno pritiskati taster F7. Umesto stalnog pritiska na taster sistem generiše odgovarajuće međupauze. Ova opcija je korisna za uočavanje petlji i ciklusa u mašinskom programu.

5.3.15 Prekidne tačke

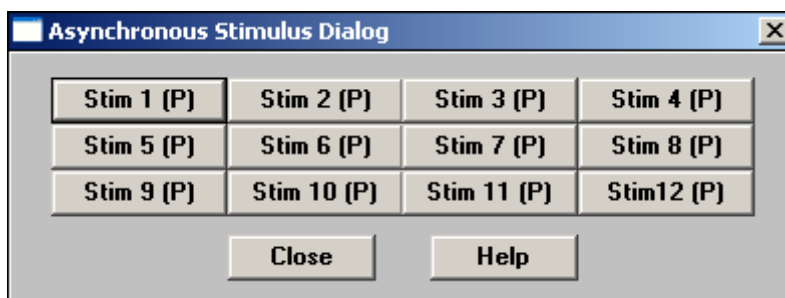
Koncept prekidnih tačaka zasniva se na postavljanju istih u sam kod. Izvršavanje programa odvija se automatski do nailaska na prekidnu tačku, gde se program zaustavlja. U tom momentu se prikazuje tekuće stanje memorije i registara. Postavljanje prekidnih tačaka obavlja se u podmeniju *Debug>Break Settings*.

5.3.16 Simulator

Simuliranje toka izvršavanja programa zasniva se na koncepciji emulacije mašinskog koda mikroprocesora PIC familije upotrebom mašinskog koda procesora x86. Standardna frekvencija rada PIC mikroprocesora iznosi 4Mhz tako da za mašine čiji je radni takt preko 1Ghz nije nikakav problem emulirati slabiji procesor u realnom vremenu, pa čak i mnogo brže.

Da bi se obezbedilo da procesor može da obrađuje ulazne signale u realnom vremenu i oni se emuliraju na neki način korišćenjem operacije *Simulator Stimulus* (opcija *Debug>Simulator Stimulus*). U ovom delu postoje četiri opcije koje regulišu pokretanje stimulacije procesora:

- *Asynchronous stimulus* – asinhrona stimulacija korišćenjem pritiska na *button-e* (sledeća slika),
- *Pin stimulus* – stimulacija pinova,
- *Clock stimulus* – simulacija časovnika,
- *Register stimulus* – simulacija registara.



Slika 5.7 - Asinhrona stimulacija ulaza

Osim prve opcije (*Asynchronous stimulus*), ostale opcije koriste datoteku sa ekstenzijom *.sti* u kojoj se nalaze snimljenje stimulacije procesora.

Ovakav pristup omogućava da se virtualno, u PC okruženju, pokrene emulacija PIC mikroprocesora u realnom vremenu kao i da se u potpunosti izvrši program čiji su ulazi unapred snimljeni u *Stimulus* fajl. Takođe, moguće je da se zatim pogleda izlazno stanje koje je generisano u toku izvršavanja emulacije. U velikom broju slučajeva ovaj pristup omogućava da se na nivou emulacije izvrše sve potrebne analize i podešavanja programa.

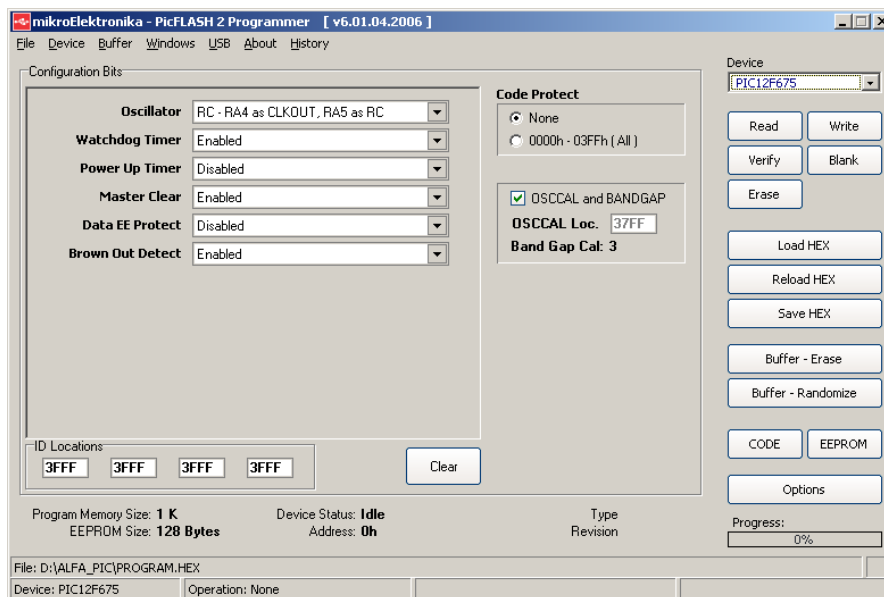
5.3.17 Programiranje mikroprocesora

Posle završetka faze editovanja, prevođenja i debugovanja, odnosno emulacije, generisan je pročišćeni izvršni kod koji je zapamćen u vidu *.hex* fajla. Na primer, detalj iz jednog *.hex* fajla je dat na sledećem listingu:

```
:1000000085280530A00064301720A00B03280800C5
:10001000AB03031D0F28203085062808AB0008001D
:10002000930BB000508AA04BB0B12280800BC005D
.....
:10024000FB2834202A1DFB2836089B3A031D1D2954
:100250003230A4003420A40B2A29B60334202A1DEE
:10026000FB2834202A1DFB2836085A3A031D2D2965
:100270004B30A4003420A40B3A29182900000000B8
:00000001FF
```

Ovaj format izvršnog koda je standard za programatore korporacije *Microchip*. Prikazana datoteka predstavlja ulaz u sledeću programsku fazu – programiranje mikrokontrolera. Za te namene, u programskom sistemu MPLAB, predviđena je opcija *PICSTART Plus* koja u stvari pokreće sam programator koji je USB kablom povezan za računar. U tom slučaju programator čini funkcionalnu celinu sa razvojnim sistem MPLAB ali nije neophodno koristiti baš taj tip programatora.

U nastavku je dat primer korišćenja programatora PICflash2 koji je na sličan način povezan sa matičnim PC računarom. Da napomenemo da je ovaj programator, sa svojom softverskom i hardverskom komponentom u potpunosti samostalan proizvod koji kao ulaz zahteva jedino odgovarajući *.hex* fajl.



Slika 5.8 - Aplikacija programatora za *Microchip* familiju procesora

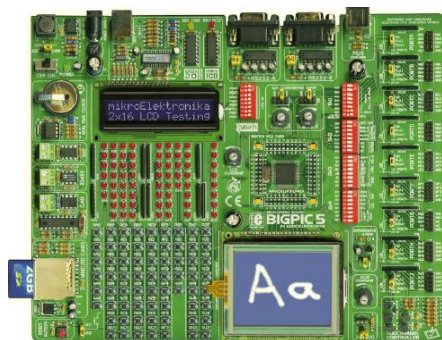
Komunikacija između MPLAB razvojnog sistema i programatora obavlja se preko USB dražvera koji se automatski aktivira kada se programator priključi na napajanje. Softverska komponenta programatora omogućava da se programiraju karakteristike velikog broja mikroprocesora PIC familije koji su ugrađeni u bazi podataka.

Programator realizuje osnovne i niz dodatnih i naprednih funkcija. Podešavanjem opcija iz prozora programatora, moguće je definisati tip oscilatora, uključiti ili isključiti *Watchdog* ili *PowerUp* tajmer, podesiti *Master Clear*, *Brown Out detect*, odnosno *Data EE* zaštitu. Sve navedene komponente programiranja mikroprocesora predstavljaju u stvari bitove konfiguracione reči koja se programira.

Takođe, programator omogućava da se osim slanja programa i podataka, ciljni procesor zaštiti od čitanja kao i da se izvrši verifikacija upisanih podataka.

Sam proces programiranja je veoma brz i oslanja se na korišćenju USB komunikacionog kanala kao i brze procesorske i programatorske jedinice na samom programatoru.

Kao primer komercijalnih programatora koji se dostupni na našem tržištu dati su proizvodi kompanije *mikroElektronika*. Na sledećoj slici je dat *bigPIC5* programator, koji uz osnovne funkcije programatora sadrži čitav niz dodatnih periferijskih komponenti kao što su nizovi tastera i LED dioda za signalizaciju stanja na portovima, tasteri za definisanje ulaza, displeji, COM i USB portovi, A/D odnosno D/A konvertori i sl.



Slika 5.9 - bigPIC5 programator – hardverska komponenta razvojnog sistema

Kao jeftinija varijanta prikazan je *PICflash2* programator iste kompanije. On se takođe sa PC računarom povezuje upotrebom USB porta, ali osim programiranja mikroprocesora nema druge funkcije niti ulazno izlanih komponenti koje smo imali prilike da uočimo kod predhodnog programatora.



Slika 5.10



Slika 5.11 - PICFlash2 programator

Kod ovog tipa programatora, potrebno je realizovati ciljnu počicu sa električnim kolom za programiranje procesora direktno na ciljnoj pločici (*onboard*), kako bi se funkcije programa direktno testirale na ciljnom hardveru.

Obe varijante programatora su interesantne i veoma korišćene u praksi.

6 MAŠINSKI JEZIK MIKROKONTROLERA PIC16F84

Mikrokontroler *Microchip PIC16F84* poseduje RISC jezgro i ima veoma redukovan skup mašinskih instrukcija.

Sve instrukcije imaju identičnu dužinu koda koja iznosi 14-bita. Zavisno od tipa instrukcije, u datom formatu su smeštene kombinacije operacionih kodova, destinacioni bit, adresa memorijskog (fajl) registra, 8-bitni podatak (*literal*) odnosno 11-bitna apsolutna adresa skoka ili poziva potprograma.

U nastavku će biti izložen pregled mašinskih instrukcija koje su svrstane u tri grupe:

- Bajt-orijentisane mašinske instrukcije,
- Bit-orijentisane mašinske instrukcije,
- Naredbe za rad sa literalima odnosno kontrolne instrukcije.

Osnovni simboli koji će se koristiti imaju sledeća značenja:

- **f** predstavlja *memorijski (fajl) registar*. Imajući u vidu da je u kodu odvojeno 7 bita za predstavljanje memorijskog registra, može se adresirati maksimalno 127 registara.
- **d** označava *destinaciju*. Destinacija predstavlja lokaciju upisa rezultata posle mašinske operacije i ona se kodira jednim bitom. Ako je taj bit 0 destinacija je akumulator (W) a ako je 1 destinacija je memorijski registar.
- **b** predstavlja kod odgovarajućeg bita u memorijskom registru. Za definisanje pozicije bita, odvojena su 3 bita u kodu. Bit najmanje težine se kodira sa 000b a bit najveće težine sa 111b.
- **k** predstavlja konstantu odnosno *literal*. Konstanta može da bude osmobarbitna za operacije za rad sa memorijskim registrima ili jedanaestobarbitna za pamćenje adrese bezuslovnih skokova.

Sve instrukcije se izvršavaju u jednom taktu procesora izuzev instrukcija uslovnih i bezuslovnih skokova kao poziva potprograma i povratka iz njih koji se izvršavaju u dva takta. Razlog za dodatni takt koji mora biti potrošen kod ovog tipa instrukcija leži u dvostepenoj *pipe-line* arhitekturi samog procesora.

PROGRAMIRANJE MIKROKONTROLERA

Svaki procesorski takt predstavlja jednu četvrtinu spoljašnjeg takta oscilatora. Na primer, ako na mikrokontroler priključimo kvarc kristal frekvencije 4 Mhz, procesor će raditi brzinom 1 MIPS-a (milion mašinskih instrukcija u sekundi). Trajanje svake mašinske instrukcije će u tom slučaju iznositi jednu mikrosekundu osim instrukcije skokova koje će na svoje izvršavanje potrošiti 2 mikrosekunde.

Svi mnemonici mašinskih instrukcija u sledećoj tabeli se mogu prevesti na mašinski kod koristeći standardni MPASM™ assembler.

Simboli koji se javljaju u opisu mašinskih naredbi mogu se prikazati sledećom tabelom:

Simbol (polje u operacionom kodu)	4. Opis
f	Adresa memorijskoga (0x00 do 0x7f)
W	Radni registar - akumulator
b	Adresa bita unutar 8-bitnog memorijskog registra
k	Literal, konstanta ili labela
x	Podatak nije važan za dekodiranje mašinske operacije. Asembleri oznaku x uvek prevode sa 0 što je dobro u pogledu kompatibilnosti
d	Destinacija: ako je 0 rezultat operacije odlazi u W a ako je 1 u memorijski registar f
PC	Programski brojač
TO	<i>Time-out</i> bit
PD	<i>Power-down</i> bit (on se setuje kada se registruje pad napona na ulazu napajanja mikrokontrolera)
0xhh	Definicija heksadecimalne konstante HH.

Kao što je već naglašeno svi kodovi mašinskih naredbi imaju 14 bita, a njihovo značenje je različito u funkciji prirode same mašinske naredbe. U sledećim tabelama prikazani su opšti formati mašinskih instrukcija.

Bajt orijentisane naredbe za rad sa memorijskim registrima:

13 8	7	6 0
Operacioni kod	Destinacija (d)	Memorijski registar (f)

d=0 za W, d=1 za f. f = sedmobitna adresa memorijskog registra

Bit orijentisane naredbe za rad sa memorijskim registrima:

13	10	9	7	6	0
Operacioni kod		Bit (b)		Memorijski registar (f)	

b = trobitna adresa bita,

f = sedmobitna adresa memorijskog registra.

Kontrolne operacije i operacije za rad sa literalima:

13	8	7	0
Operacioni kod		Literal (k)	

k = 8 bitna konstanta.

CALL i GOTO instrukcije:

13	11	10	0
Operacioni kod		Literal (k)	

k = 11-bitna konstanta koja predstavlja apsolutnu adresu skoka.

Napomena: Procesor prepoznaje još dve mašinske naredbe: *OPTION* i *TRIS* ali se njihovo korišćenje ne preporučuje iz razloga vertikalne kompatibilnosti procesora.

5.4 Set instrukcija mikrokontrolera PIC16F84 (Alfabetски redosled naredbi)

Mnemonik – mašinska naredba	Oper andi	Opis mašinske naredbe	Br oj taktov a	14-bitni kodovi naredbe	Stanje flegova
				MSb LSb	
ADDLW	k	W=W+L	1	11 111x kkkk kkkk	C,DC,Z
ADDWF	f,d	Sabira W i f	1	00 0111 dfff ffff	C,DC,Z
ANDLW	k	W=W.and.L	1	11 1001 kkkk kkkk	Z
ANDWF	f,d	W.and.f	1	00 0101 dfff ffff	Z

PROGRAMIRANJE MIKROKONTROLERA

BCF	f,b	Obriši bit b u f	1	01 00bb bfff ffff	
BSF	f,b	Postavi na 1 bit b u f	1	01 01bb bfff ffff	
BTFSC	f,b	Testiraj bit b u f, preskok ako je 0	1 (2)	01 10bb bfff ffff	
BTFSS	f,b	Testiraj bit b u f, preskok ako je 1	1 (2)	01 11bb bfff ffff	
CALL	k	Poziv potprograma	2	10 0kkk kkkk kkkk	
CLRF		Briše f	1	00 0001 1fff ffff	Z
CLRW		Briše W	1	00 0001 0xxx xxxx	Z
CLRWDT	-	Obriši Watchdog tajmer	1	00 0000 0110 0100	TO,PD
COMF	f,d	Logički komplement f	1	00 1001 dfff ffff	Z
DECF	f,d	Umanji f za 1	1	00 0011 dfff ffff	Z
DECFSZ	f,d	Umanji f za 1, preskok ako je 0	1 (2)	00 1011 dfff ffff	
GOTO	k	Bezuslovni skok na adresu	2	10 1kkk kkkk kkkk	
INCF	f,d	Uvećaj f za 1	1	00 1010 dfff ffff	Z
INCFSZ	f,d	Uvećaj f za 1, preskok ako je 0	1 (2)	00 1111 dfff ffff	
IORLW	k	W=W.or. L	1	11 1000 kkkk kkkk	Z
IORWF	f,d	W.or.f	1	00 0100 dfff ffff	Z
MOVF	f,d (W)	Prebaci f u W	1	00 1000 dfff ffff	Z
MOVLW	k	Prebaci konstantu L u W	1	11 00xx kkkk kkkk	

6. MAŠINSKI JEZIK MIKROKONTROLERA PIC16F84

MOVWF	f	Prebaci W u f	1	00 0000 1fff ffff	
NOP	-	Nema operacije	1	00 0000 0xx0 0000	
RETFIE	-	Povratak iz interapt procedure	2	00 0000 0000 1001	
RETLW	k	Povratak iz interapt procedure, uz dodelu W=L	2	11 01xx kkkk kkkk	
RETURN	-	Povratak iz potprograma	2	00 0000 0000 1000	
RLF	f,d	Rotacija ulevo f kroz carry fleg	1	00 1101 dfff ffff	C
RRF	f,d	Rotacija udesno f kroz carry fleg	1	00 1100 dfff ffff	C
SLEEP	-	Prelazak u standby mod	1	00 0000 0110 0011	TO,PD
SUBLW	k	W=L-W	1	11 110x kkkk kkkk	C,DC, Z
SUBWF	f,d	f-W	1	00 0010 dfff ffff	C,DC, Z
SWAPF	f,d	Zameni viši i niži nibbl (4 bita) u f	1	00 1110 dfff ffff	
XORLW	k	W=W.xor .L	1	11 1010 kkkk kkkk	Z
XORWF	f,d	W.xor.f	1	00 0110 dfff ffff	Z

5.5 Set instrukcija mikrokontrolera PIC16F84 (Funkcionalna tabela)

Mnemonik – mašinska naredba	Operand i	Opis mašinske naredbe	Broj taktova	14-bitni kodovi naredbe	Stanje flegova
				5. <u>MS</u> b <u>LS</u> b	

PROGRAMIRANJE MIKROKONTROLERA

6. AD DW F	f,d	Sabira W i f	1	00 0111 dfff ffff	C,DC,Z
ANDWF	f,d	W.and.f	1	00 0101 dfff ffff	Z
CLRF		Briše f	1	00 0001 1fff ffff	Z
CLRWF		Briše W	1	00 0001 0xxx xxxx	Z
COMF	f,d	Logički komplement f	1	00 1001 dfff ffff	Z
DECF	f,d	Umanji f za 1	1	00 0011 dfff ffff	Z
DECFSZ	f,d	Umanji f za 1, preskok ako je 0	1 (2)	00 1011 dfff ffff	
INCF	f,d	Uvećaj f za 1	1	00 1010 dfff ffff	Z
INCFSZ	f,d	Uvećaj f za 1, preskok ako je 0	1 (2)	00 1111 dfff ffff	
IORWF	f,d	W.or.f	1	00 0100 dfff ffff	Z
MOVF	f,d (W)	Prebaci f u W	1	00 1000 dfff ffff	Z
MOVWF	f	Prebaci W u f	1	00 0000 1fff ffff	
NOP	-	Nema operacije	1	00 0000 0xx0 0000	
RLF	f,d	Rotacija ulevo f kroz carry fleg	1	00 1101 dfff ffff	C
RRF	f,d	Rotacija udesno f kroz carry fleg	1	00 1100 dfff ffff	C
SUBWF	f,d	f-W	1	00 0010 dfff ffff	C,DC,Z
SWAPF	f,d	Zameni viši i niži nibbl (4 bita) u f	1	00 1110 dfff ffff	
XORWF	f,d	W.xor.f	1	00 0110 dfff ffff	Z
BCF	f,b	Obriši bit b u f	1	01 00bb bfff ffff	
BSF	f,b	Postavi na 1 bit b u f	1	01 01bb bfff ffff	

BTFSC	f,b	Testiraj bit b u f, preskok ako je 0	1 (2)	01 10bb bfff ffff	
BTFSS	f,b	Testiraj bit b u f, preskok ako je 1	1 (2)	01 11bb bfff ffff	
7. AD DL W	k	W=W+L	1	11 111x kkkk kkkk	C,DC,Z
ANDLW	k	W=W.and.L	1	11 1001 kkkk kkkk	Z
CALL	k	Poziv potprograma	2	10 0kkk kkkk kkkk	
CLRWDT	-	Obriši <i>Watchdog</i> tajmer	1	00 0000 0110 0100	TO,PD
GOTO	k	Bezuslovni skok na adresu	2	10 1kkk kkkk kkkk	
IORLW	k	W=W.or.L	1	11 1000 kkkk kkkk	Z
MOVLW	k	Prebaci konstantu L u W	1	11 00xx kkkk kkkk	
RETFIE	-	Povratak iz interapt procedure	2	00 0000 0000 1001	
RETLW	k	Povratak iz interapt procedure, uz dodelu W=L	2	11 01xx kkkk kkkk	
RETURN	-	Povratak iz potprograma	2	00 0000 0000 1000	
SLEEP	-	Prelazak u <i>standby</i> mod	1	00 0000 0110 0011	TO,PD
SUBLW	k	W=L-W	1	11 110x kkkk kkkk	C,DC,Z
XORLW	k	W=W.xor.L	1	11 1010 kkkk kkkk	Z

5.6 Mašinske instrukcije mikrokontrolera PIC16F84

U nastavku su prikazane detaljne tabele mašinskih naredbi u standardnom formatu. Ove tabele definišu sve potrebne komponente kodiranja i izvršavanja svih mašinskih naredbi mikrokontrolera PIC16F84.

ADDLW	Sabiranje akumulatora W i konstante k			
Sintaksa naredbe:	[labela] ADDLW k			
Opseg definisanosti operanda:	0<=k<=255			
Operacija:	W=W+k			
Promene flegova posle izvršenja naredbe:	Z,C,DC			
Opis:	Sadržaj akumulatora se sabira sa osmobićnom konstantom k i rezultat se upisuje nazad u W.			
Kod:	11	111x	kkkk	kkkk
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1 Dekodiranje	Q2 Ćitanje litala K	Q3 Obrada	Q4 Upis u W
Primer korišćenja naredbe:	<p>8. ADDLW 0x01</p> <p><u>Pre instrukcije:</u></p> <p>W=0x20</p> <p><u>Posle instrukcije:</u></p> <p>W=0x21</p>			

ANDLW	Logićki AND akumulatora W i konstante k			
Sintaksa naredbe:	[labela] ANDLW k			
Opseg definisanosti operanda:	0<=k<=255			
Operacija:	W=W.and.k			
Promene flegova posle izvršenja naredbe:	Z			
Opis:	Obavlja se logićka AND operacija nad sadržajem akumulatora i konstantom k. Rezultat se upisuje nazad u W.			
Kod:	11	1001	kkkk	kkkk
Broj taktova za izvršenje:	1			

Q ciklusi:	Q1	Q2	Q3	Q4
	Dekodiranje	Čitanje literals K	Obrada	Upis u W
Primer korišćenja naredbe:	ANDLW 0x01 <u>Pre instrukcije:</u> W=0x21 <u>Posle instrukcije:</u> W=0x01			

ADDWF	Sabiranje akumulatora W i memorijskog registra f			
Sintaksa naredbe:	[labela] ADDWF f,d			
Opseg definisanosti operanda:	0<=f<=127 ; d=0,1			
Operacija:	W=W+(f)			
Promene flegova posle izvršenja naredbe:	Z,C,DC			
Opis:	Sadržaj akumulatora se sabira sa sadržajem memorijskog registra f i rezultat se upisuje u destinaciju.			
Kod:	00	0111	dfff	ffff
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1	Q2	Q3	Q4
	Dekodiranje	Čitanje memorijskog registra f	Obrada	Upis u destinaciju
Primer korišćenja naredbe:	ADDWF FSR,1 <u>Pre instrukcije:</u> W=0x10 FSR=0x02			

PROGRAMIRANJE MIKROKONTROLERA

	<u>Posle instrukcije:</u> W=0x10 FSR=0x12
--	---

ANDWF	Logicki AND akumulatora W i memorijskog registra f			
Sintaksa naredbe:	[labela] ADDWF f,d			
Opseg definisanosti operanda:	0<=f<=127 ; d=0,1			
Operacija:	W=W.and.(f)			
Promene flegova posle izvršenja naredbe:	Z			
Opis:	Sadržaj akumulatora se sabira sa osmobičnom konstantom k i rezultat se upisuje nazad u W.			
Kod:	00	0101	dfff	ffff
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1 Dekodiranje	Q2 Čitanje memorijskog registra f	Q3 Obrada	Q4 Upis u destinaciju
Primer korišćenja naredbe:	ANDWF FSR,0 <u>Pre instrukcije:</u> W=0x21 FSR=0x01 <u>Posle instrukcije:</u> W=0x01 FSR=0x01			

BCF	Postavljanje bita u memorijski registar f na 0
-----	--

Sintaksa naredbe:	[labela] BCF f,b			
Opseg definisanosti operanda:	$0 \leq f \leq 127$; $0 \leq b \leq 7$			
Operacija:	$f[b]=0$			
Promene flegova posle izvršenja naredbe:	-			
Opis:	Bit b memorijskog registra f dobija vrednost 0.			
Kod:	01	00bb	bfff	ffff
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1	Q2	Q3	Q4
	Dekodiranje	Čitanje memorijskog registra f	Obrada	Upis u registar f
Primer korišćenja naredbe:	BCF FSR,7 <u>Pre instrukcije:</u> FSR=0x80 <u>Posle instrukcije:</u> FSR=0x00			

BSF	Postavljanje bita b u memorijskom registru f na 1			
Sintaksa naredbe:	[labela] BSF f,b			
Opseg definisanosti operanda:	$0 \leq f \leq 127$; $0 \leq b \leq 7$			
Operacija:	$f[b]=1$			
Promene flegova posle izvršenja naredbe:	-			
Opis:	Bit b memorijskog registra f dobija vrednost 1.			
Kod:	01	01bb	bfff	ffff
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1	Q2	Q3	Q4

PROGRAMIRANJE MIKROKONTROLERA

	Dekodiranje	Čitanje memorijskog registra f	Obrada	Upis u registar f
Primer korišćenja naredbe:	BSF FSR,7 <u>Pre instrukcije:</u> FSR=0x7f <u>Posle instrukcije:</u> FSR=0xff			

BTFSC	Testiranje bita b u memorijskom registru f, ako je 0 preskok sledeće mašinske naredbe			
Sintaksa naredbe:	[labela] BTFSC f,b			
Opseg definisanosti operanda:	0<=f<=127 ; 0<=b<=7			
Operacija:	Ako je f[b]=0, preskok sledeće naredbe			
Promene flegova posle izvršenja naredbe:	-			
Opis:	Testira se bit b memorijskog registra f, ako je 0 preskače se sledeća mašinska naredba.			
Kod:	01	10bb	bfff	ffff
Broj taktova za izvršenje:	1 (2 ako je uslov za preskok ispunjen)			
Q ciklusi (kada uslov za skok nije ispunjen):	Q1	Q2	Q3	Q4
	Dekodiranje	Čitanje memorijskog registra f	Obrada	Nema operacije
Q ciklusi (kada je uslov za skok ispunjen, ubacuje se dodatni takt):	Q1	Q2	Q3	Q4
	Nema operacije	Nema operacije	Nema operacije	Nema operacije
Primer korišćenja naredbe:	BTFSC FSR,0 L1: NOP L2: ADDLW 1			

	<p><u>Pre instrukcije:</u></p> <p>FSR=0x00</p> <p>Pošto je bit 0 memorijskog registra FSR nula, preskače se NOP naredba i skače se na labelu L2. Da je odgovarajući bit 1, izvršila bi se naredba na labeli L1.</p>
--	---

BTFSS	Testiranje bita b u memorijskom registru f, ako je 1 preskok sledeće mašinske naredbe			
Sintaksa naredbe:	[labela] BTFSS f,b			
Opseg definisanosti operanda:	0<=f<=127 ; 0<=b<=7			
Operacija:	Ako je f[b]=1, preskok sledeće naredbe			
Promene flegova posle izvršenja naredbe:	-			
Opis:	Testira se bit b u memorijskom registru f, ako je 1 preskače se sledeća mašinska naredba.			
Kod:	01	11bb	bfff	ffff
Broj taktova za izvršenje:	1 (2 ako je uslov za preskok ispunjen)			
Q ciklusi (kada uslov za skok nije ispunjen):	Q1	Q2	Q3	Q4
	Dekodiranje	Čitanje memorijskog registra f	Obrada	Nema operacije
Q ciklusi (kada je uslov za skok ispunjen, ubacuje se dodatni takt):	Q1	Q2	Q3	Q4
	Nema operacije	Nema operacije	Nema operacije	Nema operacije
Primer korišćenja naredbe:	<p>BTFSS FSR,0</p> <p>L1: NOP</p> <p>L2: SUBLW 2</p> <p><u>Pre instrukcije:</u></p> <p>FSR=0x01</p> <p>Pošto je bit 0 memorijskog registra FSR jedan, preskače se NOP naredba i skače se na labelu L2. Da</p>			

PROGRAMIRANJE MIKROKONTROLERA

	je odgovarajući bit 0, izvršila bi se naredba na labeli L1
--	--

CALL	Bezuslovni poziv potprograma			
Sintaksa naredbe:	[labela] CALL k			
Opseg definisanosti operanda:	$0 \leq k \leq 2047$			
Operacija:	TOS=PC+1 PC[10:0]=k PC[12:11]=PCLATH[4:3]			
Promene flegova posle izvršenja naredbe:	-			
Opis:	Na stek se ostavlja povratna adresa (PC+1). U PC se učitavaju nižih 11 bita adrese. Viši bitovi PC se učitavaju iz PCLATH. Poziva se potprogram PP.			
Kod:	10	0kkk	kkkk	kkkk
Broj taktova za izvršenje:	2			
Q ciklusi (prvi takt):	Q1	Q2	Q3	Q4
	Dekodiranje	Čitanje literala k, snimanje PC na stek	Obrada	Upis u PC
Q ciklusi (drugi takt):	Q1	Q2	Q3	Q4
	Nema operacije	Nema operacije	Nema operacije	Nema operacije
Primer korišćenja naredbe:	<p>main: CALL PP</p> <p><u>Pre instrukcije:</u></p> <p>PC=main</p> <p><u>Posle instrukcije:</u></p> <p>PC=PP</p> <p>TOS=main+1</p>			

CLRF	Brisanje sadržaja memorijskog registra f			
Sintaksa naredbe:	[labela] CLRF f			
Opseg definisanosti operanda:	0<=f<=127			
Operacija:	(f)=0 Z=1			
Promene flegova posle izvršenja naredbe:	Z			
Opis:	Sadržaj memorijskog registra f se briše.			
Kod:	00	0001	1fff	ffff
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1 Dekodiranje	Q2 Čitanje memorijskog registra f	Q3 Obrada	Q4 Upis u registar f
Primer korišćenja naredbe:	CLRF FSR <u>Pre instrukcije:</u> FSR=0x12 <u>Posle instrukcije:</u> FSR=0x00			

CLRW	Brisanje sadržaja akumulatora W			
Sintaksa naredbe:	[labela] CLRW			
Opseg definisanosti operanda:	-			
Operacija:	W=0 Z=1			
Promene flegova posle izvršenja naredbe:	Z			
Opis:	Sadržaj akumulatora se briše.			
Kod:	00	0001	0xxx	xxxx

PROGRAMIRANJE MIKROKONTROLERA

Broj taktova za izvršenje:	1			
Q ciklusi:	Q1	Q2	Q3	Q4
	Dekodiranje	Bez operacije	Obrada	Upis u W
Primer korišćenja naredbe:	CLR W <u>Pre instrukcije:</u> W=0xff <u>Posle instrukcije:</u> W=0x00			

CLR WDT	Brisanje WATCHDOG timera			
Sintaksa naredbe:	[labela] CLR WDT			
Opseg definisanosti operanda:	-			
Operacija:	WDT=0 WDT preskaler = 0 TO = 1 PD = 1			
Promene flegova posle izvršenja naredbe:	TO,PD			
Opis:	Briše se <i>Watchdog</i> tajmer. Briše se WDT preskaler. Flegovi TO i PD se postavljaju na 1.			
Kod:	00	0000	0110	0100
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1	Q2	Q3	Q4
	Dekodiranje	Bez operacije	Obrada	Brisanje WDT brojača
Primer korišćenja naredbe:	CLR WDT <u>Pre instrukcije:</u>			

	<p>WDT brojač = 0x34</p> <p><u>Posle instrukcije:</u></p> <p>WDT brojač = 0 WDT preskaler = 0 TO = 1 , PD = 1</p>
--	---

COMF	Komplementiranje sadržaja memorijskog registra f				
Sintaksa naredbe:	[labela] COMF f,d				
Opseg definisanosti operanda:	0 ≤ f ≤ 127 ; d = 0,1				
Operacija:	(f) = .not.(f)				
Promene flegova posle izvršenja naredbe:	Z				
Opis:	Sadržaj memorijskog registra f se komplementira (jedinični, logički komplement).				
Kod:	00	1001	dfff	ffff	
Broj taktova za izvršenje:	1				
Q ciklusi:	Q1	Q2	Q3	Q4	
	Dekodiranje	Čitanje memorijskog registra f	Obrada	Upis u destinaciju	
Primer korišćenja naredbe:	<p>COMF FSR,1</p> <p><u>Pre instrukcije:</u></p> <p>FSR=0x88</p> <p><u>Posle instrukcije:</u></p> <p>FSR=0x77</p>				

DECF	Umanjivanje sadržaja memorijskog registra f za 1
Sintaksa naredbe:	[labela] DECF f,d

PROGRAMIRANJE MIKROKONTROLERA

Opseg definisanosti operanda:	$0 \leq f \leq 127$; d=0,1			
Operacija:	$(f) = (f) - 1$			
Promene flegova posle izvršenja naredbe:	Z			
Opis:	Umanjuje se sadržaj memorijskog registra f za 1.			
Kod:	00	0011	dfff	ffff
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1 Dekodiranje	Q2 Čitanje memorijskog registra f	Q3 Obrada	Q4 Upis u destinaciju
Primer korišćenja naredbe:	<p>DECF FSR,0</p> <p><u>Pre instrukcije:</u></p> <p>W=0x12 FSR=0x80</p> <p><u>Posle instrukcije:</u></p> <p>W=0x7F FSR=0x80</p>			

DECFSZ	Umanjivanje sadržaja memorijskog registra f za 1, skok ako je 0
Sintaksa naredbe:	[labela] DECFSZ f,d
Opseg definisanosti operanda:	$0 \leq f \leq 127$; d=0,1
Operacija:	$(f) = (f) - 1$ Skok ako je rezultat 0.
Promene flegova posle izvršenja naredbe:	-
Opis:	Umanjuje se sadržaj memorijskog registra za 1. U zavisnosti od destinacije rezultat se šalje nazad u f ili ide u W. Ako je posle umanjivanja rezultat 0, preskače se sledeća naredba, a ako nije ona se izvršava.

Kod:	00	1011	dfff	ffff	
Broj taktova za izvršenje:	1 (2 ako je uslov za preskok ispunjen)				
Q ciklusi (kada uslov za skok nije ispunjen):	Q1	Q2	Q3	Q4	
	Dekodiranje	Čitanje memorijskog registra f	Obrada	Upis u destinaciju	
Q ciklusi (kada je uslov za skok ispunjen, ubacuje se dodatni takt):	Q1	Q2	Q3	Q4	
	Nema operacije	Nema operacije	Nema operacije	Nema operacije	
Primer korišćenja naredbe:	<p>loop: DECFSZ FSR,1 L1: GOTO loop L2: NOP</p> <p><u>Pre instrukcije:</u></p> <p>FSR=0x02</p> <p>Umanjuje se sadržaj memorijskog registra FSR za 1. Za slučaj da rezultat nije 0 izvršava se naredba na labeli L1 (skok na početak petlje). Ako je rezultat 0, skače se na labelu L2. Ova naredba je veoma pogodna za realizaciju brojačkih petlji.</p> <p><u>Posle instrukcije:</u></p> <p>FSR=0x01</p>				

GOTO	Bezuslovni skok
Sintaksa naredbe:	[labela] GOTO k
Opseg definisanosti operanda:	0<=k<=2047
Operacija:	TOS=PC+1 PC[10:0]=k PC[12:11]=PCLATH[4:3]
Promene flegova posle izvršenja naredbe:	-

PROGRAMIRANJE MIKROKONTROLERA

Opis:	Obavlja se bezuslovni skok na naredbu čija je lokacija specificirana. U PC se učitavaju nižih 11 bita adrese. Viši bitovi PC se učitavaju iz PCLATH.				
Kod:	10	1kkk	kkkk	kkkk	
Broj taktova za izvršenje:	2				
Q ciklusi (prvi takt):	Q1 Dekodiranje	Q2 Čitanje literals k	Q3 Obrada	Q4 Upis u PC	
Q ciklusi (drugi takt):	Q1 Nema operacije	Q2 Nema operacije	Q3 Nema operacije	Q4 Nema operacije	
Primer korišćenja naredbe:	9. GOTO labela				

INCF	Uvećavanje sadržaja memorijskog registra f za 1				
Sintaksa naredbe:	[labela] INCF f				
Opseg definisanosti operanda:	0<=f<=127				
Operacija:	(f)=(f)+1				
Promene flegova posle izvršenja naredbe:	Z				
Opis:	Sadržaj memorijskog registra f se uvećava za 1.				
Kod:	00	1010	dfff	ffff	
Broj taktova za izvršenje:	1				
Q ciklusi:	Q1 Dekodiranje	Q2 Čitanje memorijskog registra f	Q3 Obrada	Q4 Upis u destinaciju	
Primer korišćenja naredbe:	INCF FSR,1 <u>Pre instrukcije:</u> W=0x12 FSR=0x80 <u>Posle instrukcije:</u> W=0x12				

	FSR=0x81
--	----------

INCFSZ	Uvećavanje sadržaja memorijskog registra f za 1, skok ako je 0			
Sintaksa naredbe:	[labela] INCFSZ f,d			
Opseg definisanosti operanda:	0<=f<=127 ; d=0,1			
Operacija:	(f)=(f)+1 Skok ako je rezultat 0			
Promene flegova posle izvršenja naredbe:	-			
Opis:	Uvećava se sadržaj memorijskog registra za 1. U zavisnosti od destinacije rezultat se šalje nazad u f ili u W. Ako je posle umanjivanja rezultat 0 preskače se sledeća naredba, a ako nije ona se izvršava.			
Kod:	00	1111	dfff	ffff
Broj taktova za izvršenje:	1 (2 ako je uslov za preskok ispunjen)			
Q ciklusi (kada uslov za skok nije ispunjen):	Q1	Q2	Q3	Q4
	Dekodiranje	Čitanje memorijskog registra f	Obrada	Upis u destinaciju
Q ciklusi (kada je uslov za skok ispunjen, ubacuje se dodatni takt):	Q1	Q2	Q3	Q4
	Nema operacije	Nema operacije	Nema operacije	Nema operacije
Primer korišćenja naredbe:	loop: INCFSZ FSR,1 L1: GOTO loop L2: NOP <u>Pre instrukcije:</u> FSR=0x02 Uvećava se sadržaj memorijskog registra FSR za 1. Za slučaj da rezultat nije 0 izvršava se naredba na labeli L1 (skok na početak petlje). Ako je rezultat 0, skače se na labelu L2. Ova naredba je veoma pogodna za realizaciju brojačkih petlji.			

PROGRAMIRANJE MIKROKONTROLERA

	<u>Posle instrukcije:</u> FSR=0x03
--	---

IORLW	Logički OR akumulatora W i konstante k			
Sintaksa naredbe:	[labela] IORLW k			
Opseg definisanosti operanda:	0<=k<=255			
Operacija:	W=W.or.k			
Promene flegova posle izvršenja naredbe:	Z			
Opis:	Obavlja se logička OR operacija između sadržaja akumulatora i osmobiitne konstante k i rezultat se upisuje nazad u W.			
Kod:	11	1000	kkkk	kkkk
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1	Q2	Q3	Q4
	Dekodiranje	Čitanje literala K	Obrada	Upis u W
Primer korišćenja naredbe:	IORLW 0x01			
	<u>Pre instrukcije:</u>			
	W=0xFE			
	<u>Posle instrukcije:</u>			
	W=0xFF			

IORWF	Logički OR akumulatora W i memorijskog registra f
Sintaksa naredbe:	[labela] IORWF f,d
Opseg definisanosti operanda:	$0 \leq f \leq 127$; d=0,1

Operacija:	W=W.or.(f)				
Promene flegova posle izvršenja naredbe:	Z				
Opis:	Obavlja se logička OR operacija između sadržaja akumulatora i osmobične konstante k i rezultat se upisuje u destinaciju.				
Kod:	00	0100	dfff	ffff	
Broj taktova za izvršenje:	1				
Q ciklusi:	Q1	Q2	Q3	Q4	
	Dekodiranje	Čitanje memorijskog registra f	Obrada	Upis u destinaciju	
Primer korišćenja naredbe:	IORWF FSR,0 <u>Pre instrukcije:</u> W=0x00 FSR=0x11 <u>Posle instrukcije:</u> W=0x11 FSR=0x11				

MOVLW	Dodela vrednosti konstante k akumulatoru W				
Sintaksa naredbe:	[labela] MOVLW k				
Opseg definisanosti operanda:	0<=k<=255				
Operacija:	W=k				
Promene flegova posle izvršenja naredbe:	-				
Opis:	Akumulatoru se dodeljuje sadržaj osmobične konstante k.				
Kod:	11	00xx	kkkk	kkkk	
Broj taktova za izvršenje:	1				
Q ciklusi:	Q1	Q2	Q3	Q4	

PROGRAMIRANJE MIKROKONTROLERA

	Dekodiranje	Čitanje literals K	Obrada	Upis u W	
Primer korišćenja naredbe:	MOVLW 0x12 <u>Pre instrukcije:</u> W=0x00 <u>Posle instrukcije:</u> W=0x12				

MOVF	Dodela vrednosti memorijskog registra f				
Sintaksa naredbe:	[labela] MOVF f,d				
Opseg definisanosti operanda:	0<=f<=127 ; d=0,1				
Operacija:	W=(f)				
Promene flegova posle izvršenja naredbe:	-				
Opis:	Sadržaj akumulatora se upisuje u destinaciju d. Ima smisla samo naredba oblika MOVF f,w .				
Kod:	00	1000	dfff	ffff	
Broj taktova za izvršenje:	1				
Q ciklusi:	Q1	Q2	Q3	Q4	
	Dekodiranje	Čitanje memorijskog registra f	Obrada	Upis u destinaciju	
Primer korišćenja naredbe:	MOVF FSR,0 <u>Pre instrukcije:</u> FSR=0x34 W=0x00 <u>Posle instrukcije:</u>				

	W=0x34
--	--------

MOVWF	Dodela vrednosti akumulatora W memorijskom registru f				
Sintaksa naredbe:	[labela] MOVWF f				
Opseg definisanosti operanda:	0<=f<=127				
Operacija:	(f)=W				
Promene flegova posle izvršenja naredbe:	-				
Opis:	Sadržaj akumulatora se upisuje u destinaciju (u memorijski registar f).				
Kod:	00	0000	1fff	ffff	
Broj taktova za izvršenje:	1				
Q ciklusi:	Q1	Q2	Q3	Q4	
	Dekodiranje	Čitanje memorijskog registra f	Obrada	Upis u memorijski registar f	
Primer korišćenja naredbe:	MOVWF FSR <u>Pre instrukcije:</u> FSR=0x00 W=0xFF <u>Posle instrukcije:</u> W=0xFF FSR=0xFF				

NOP	Nema operacije
Sintaksa naredbe:	[labela] NOP
Opseg definisanosti operanda:	-

PROGRAMIRANJE MIKROKONTROLERA

Operacija:	-			
Promene flegova posle izvršenja naredbe:	-			
Opis:	Nema operacije			
Kod:	00	0000	0xx0	0000
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1	Q2	Q3	Q4
	Dekodiranje	Nema operacije	Nema operacije	Nema operacije
Primer korišćenja naredbe:	NOP			

RETFIE	Povratak iz interapta			
Sintaksa naredbe:	[labela] RETFIE			
Opseg definisanosti operanda:	-			
Operacija:	PC=TOS GIE=1			
Promene flegova posle izvršenja naredbe:	-			
Opis:	Povratak iz interapta. Sa vrha steka se uzima povratna adresa koja se učitava u PC. Dozvoljavaju se interapti upisom vrednosti 1 u GIE.			
Kod:	00	0000	0000	1001
Broj taktova za izvršenje:	2			
Q ciklusi (prvi takt):	Q1	Q2	Q3	Q4
	Dekodiranje	Nema operacije.	Postavljanje GIE bita na vrednost 1	Učitavanje sa steka
Q ciklusi (drugi takt):	Q1	Q2	Q3	Q4
	Nema operacije	Nema operacije	Nema operacije	Nema operacije
Primer korišćenja naredbe:	RETFIE <u>Posle interapta:</u> PC = TOS GIE = 1			

--	--

OPTION	Učitaj OPTION registar
Sintaksa naredbe:	[labela] OPTION
Opseg definisanosti operanda:	-
Operacija:	OPTION=W
Promene flegova posle izvršenja naredbe:	-
Opis:	Ova instrukcija učitava vredost OPTION registra u akumulator. Instrukcija je uvedena u cilju kompatibilnosti sa starijom PIC16C5x serijom.
Kod:	00 0000 0110 0010
Broj taktova za izvršenje:	1
Napomena:	Da bi se očuvala vertikalna kompatibilnost sa budućim PIC16Fxx procesorima NE KORISTITI OVU INSTRUKCIJU.

RETLW	Povratak iz potprograma i učitavanje konstante u akumulator			
Sintaksa naredbe:	[labela] RETLW k			
Opseg definisanosti operanda:	0<=k<=255			
Operacija:	W=k PC=TOS			
Promene flegova posle izvršenja naredbe:	-			
Opis:	Osmobitna konstanta k se učitava u akumulator W. Sa vrha steka se uzima povratna adresa koja se učitava u PC.			
Kod:	11	01xx	kkkk	kkkk
Broj taktova za izvršenje:	2			
Q ciklusi (prvi takt):	Q1	Q2	Q3	Q4
	Dekodiranje	Čitanje literala k	Nema operacije	Upis konstante k u W.

PROGRAMIRANJE MIKROKONTROLERA

				Upis u PC
Q ciklusi (drugi takt):	Q1	Q2	Q3	Q4
	Nema operacije	Nema operacije	Nema operacije	Nema operacije
Primer korišćenja naredbe:	<p>Realizacija tabele preskoka:</p> <p>MOVLW 2 CALL TAB ...</p> <p>TAB:</p> <p>ADDWF PCL,W ; Dodavanje vrednosti akumulatora W programskom brojaču</p> <p>RETLW c0 RETLW c1 RETLW c2 ; Izlazna tačka, akumulator dobija vrednost c2</p> <p>RETLW c3 ...</p>			

RETURN	Povratak iz potprograma			
Sintaksa naredbe:	[labela] RETURN			
Opseg definisanosti operanda:	-			
Operacija:	PC=TOS			
Promene flegova posle izvršenja naredbe:	-			
Opis:	Povratak iz potprograma. Sa vrha steka se povratna adresa učitava u PC.			
Kod:	00	0000	0000	1000
Broj taktova za izvršenje:	2			
	Q1	Q2	Q3	Q4

Q ciklusi (prvi takt):	Dekodiranje	Nema operacije	Nema operacije	Upis u PC
Q ciklusi (drugi takt):	Q1 Nema operacije	Q2 Nema operacije	Q3 Nema operacije	Q4 Nema operacije
Primer korišćenja naredbe:	RETURN <u>Posle izvršenja:</u> PC=TOS			

RLF	Rotiranje memorijskog registra ulevo kroz <i>carry fleg</i>			
Sintaksa naredbe:	[labela] RLF f,d			
Opseg definisanosti operanda:	0<=f<=127 ; d=0,1			
Operacija:	(f):=(f).rol.1			
Promene flegova posle izvršenja naredbe:	C			
Opis:	Sadržaj fajl registra se rotira ulevo kroz <i>carry fleg</i> za jednu poziciju. Na taj način se ostvaruje rotacija za 8+1 bita. Rezultat se šalje u destinaciju.			
Kod:	00	1101	dfff	ffff
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1 Dekodiranje	Q2 Čitanje memorijskog registra f	Q3 Obrada	Q4 Upis u destinaciju
Primer korišćenja naredbe:	RLF FSR,1 <u>Pre instrukcije:</u> Carry=0 FSR=0x81 <u>Posle instrukcije:</u>			

PROGRAMIRANJE MIKROKONTROLERA

	<i>Carry</i> =1 <i>FSR</i> =0x02
--	-------------------------------------

RRF	Rotiranje memorijskog registra udesno kroz <i>carry fleg</i>			
Sintaksa naredbe:	[labela] RRF f,d			
Opseg definisanosti operanda:	0<=f<=127 ; d=0,1			
Operacija:	(f)=(f).ror.1			
Promene flegova posle izvršenja naredbe:	C			
Opis:	Sadržaj fajl registra se rotira udesno kroz <i>carry fleg</i> za jednu poziciju. Na taj način se ostvaruje rotacija za 8+1 bita. Rezultat se šalje u destinaciju.			
Kod:	00	1100	dfff	ffff
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1 Dekodiranje	Q2 Čitanje memorijskog registra f	Q3 Obrada	Q4 Upis u destinaciju
Primer korišćenja naredbe:	RRF FSR,0 <u>Pre instrukcije:</u> <i>Carry</i> =0 <i>FSR</i> =0x80 <u>Posle instrukcije:</u> <i>Carry</i> =0 <i>FSR</i> =0x80 <i>W</i> =0x40			

SLEEP	Prelazak u <i>stand-by</i> režim
--------------	---

Sintaksa naredbe:	[labela] SLEEP			
Opseg definisanosti operanda:	-			
Operacija:	WDT=0 WDT prescaler = 0 TO=1 PD=0			
Promene flegova posle izvršenja naredbe:	TO, PD			
Opis:	<i>Power-down</i> status bit PD se postavlja na 0. <i>Time-out</i> status bit TO se postavlja na 1. Tajmer i prescaler se brišu. Procesor odlazi u SLEEP mod i oscilator se zaustavlja.			
Kod:	00	0000	0110	0011
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1 Dekodiranje	Q2 Nema operacije	Q3 Nema operacije	Q4 Odlazak u SLEEP mod
Napomena:	SLEEP			

SUBLW	Oduzimanje akumulatora W od konstante k			
Sintaksa naredbe:	[labela] SUBLW k			
Opseg definisanosti operanda:	$0 \leq k \leq 255$			
Operacija:	$W = k - W$			
Promene flegova posle izvršenja naredbe:	Z,C,DC			
Opis:	Sadržaj akumulatora se oduzima od osmobične konstante k i rezultat se upisuje nazad u W. Ako je rezultat nula, <i>zero</i> fleg se postavlja na 1. Ako je rezultat negativan <i>carry</i> fleg se postavlja na 1.			
Kod:	11	110x	kkkk	kkkk
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1	Q2	Q3	Q4

PROGRAMIRANJE MIKROKONTROLERA

	Dekodiranje	Čitanje literals K	Obrada	Upis u W	
Primer korišćenja naredbe:	SUBLW 0xA0 <u>Pre instrukcije:</u> W=0x02 <u>Posle instrukcije:</u> W=0x08				

SUBWF	Oduzimanje akumulatora W od memorijskog registra f				
Sintaksa naredbe:	[labela] SUBWF f,d				
Opseg definisanosti operanda:	0<=f<=127 ; d=0,1				
Operacija:	Destinacija = (f)-W				
Promene flegova posle izvršenja naredbe:	Z,C,DC				
Opis:	Sadržaj akumulatora se oduzima od memorijskog registra f i rezultat se upisuje destinaciju. Ako je rezultat nula, zero fleg se postavlja na 1. Ako je rezultat negativan carry fleg se postavlja na 1.				
Kod:	00	0010	dfff	ffff	
Broj taktova za izvršenje:	1				
Q ciklusi:	Q1	Q2	Q3	Q4	
	Dekodiranje	Čitanje memorijskog registra f	Obrada	Upis u destinaciju	
Primer korišćenja naredbe:	SUBWF FSR,1 <u>Pre instrukcije:</u> W=0x07 FSR=0x10				

	<u>Posle instrukcije:</u> W=0x07 FSR=0x09
--	---

SWAPF	Zamena <i>niblova</i> (viši i niži 4-bitni delovi) memorijskog registra f			
Sintaksa naredbe:	[labela] SWAPF f,d			
Opseg definisanosti operanda:	0<=f<=127 ; d=0,1			
Operacija:	F[0..3] \leftrightarrow F[4..7]			
Promene flegova posle izvršenja naredbe:	-			
Opis:	Zamenjuju se <i>niblovi</i> (viši i niži 4-bitni polubajtovi) memorijskog registra f. Rezultat se upisuje u destinaciju.			
Kod:	00	1110	dfff	ffff
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1 Dekodiranje	Q2 Čitanje memorijskog registra f	Q3 Obrada	Q4 Upis u destinaciju
Primer korišćenja naredbe:	SWAPF FSR,0 <u>Pre instrukcije:</u> W=0x00 FSR=0x24 <u>Posle instrukcije:</u> W=0x42 FSR=0x24			

TRIS	Učitaj TRIS registar
-------------	-----------------------------

PROGRAMIRANJE MIKROKONTROLERA

Sintaksa naredbe:	[labela] TRIS f
Opseg definisanosti operanda:	5<=f<=7
Operacija:	TRIS=W
Promene flegova posle izvršenja naredbe:	-
Opis:	Ova instrukcija učitava vredost TRIS registra u akumulator. Instrukcija je uvedena u cilju kompatibilnosti sa starijom PIC16C5x serijom.
Kod:	00 0000 0110 0fff
Broj taktova za izvršenje:	1
Napomena:	Da bi se očuvala vertikalna kompatibilnost sa budućim PIC16Fxx procesorima NE KORISTITI OVU INSTRUKCIJU.

XORLW	Logički XOR akumulatora W i konstante k			
Sintaksa naredbe:	[labela] XORLW			
Opseg definisanosti operanda:	0<=k<=255			
Operacija:	W=W.xor.k			
Promene flegova posle izvršenja naredbe:	Z			
Opis:	Obavlja se logička XOR operacija između sadržaja akumulatora i osmobične konstante k i rezultat se upisuje nazad u W.			
Kod:	11	1010	kkkk	kkkk
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1 Dekodiranje	Q2 Čitanje literals K	Q3 Obrada	Q4 Upis u W
Primer korišćenja naredbe:	XORLW 0xFF <u>Pre instrukcije:</u> W=0xC1			

	<u>Posle instrukcije:</u> W=0x3E
--	---

XORWF	Logički XOR akumulatora W i memorijskog registra f			
Sintaksa naredbe:	[labela] XORWF f,d			
Opseg definisanosti operanda:	0<=f<=127 ; d=0,1			
Operacija:	W=W.xor.(f)			
Promene flegova posle izvršenja naredbe:	Z			
Opis:	Obavlja se logička XOR operacija između sadržaja akumulatora i memorijskog registra f i rezultat se upisuje u destinaciju.			
Kod:	00	0110	dfff	ffff
Broj taktova za izvršenje:	1			
Q ciklusi:	Q1 Dekodiranje	Q2 Čitanje memorijskog registra f	Q3 Obrada	Q4 Upis u destinaciju
Primer korišćenja naredbe:	XORWF FSR,1 <u>Pre instrukcije:</u> W=0x1F FSR=0x01 <u>Posle instrukcije:</u> W=0x0F FSR=0x1E			

7 ZAKLJUČAK

Ovaj pomoćni udžbenik je proistekao iz praktičnog rada i iskustva koji su autori stekli držeći teoretska predavanja i računske vežbe iz predmeta Mikrokontroleri i programiranje i Mikrorračunarski sistemi Osnovnim studijama na Elektronskom fakultetu u Nišu. Autori su u prvom redu namenili ovaj rukopis studentima koji su polaznici ovih kurseva.

U toku izrade rukopisa autori su se oslanjali na obiman materijal iz literature, koja je navedena u rukopisu. Udžbenik sadrži veliki broj praktičnih primera u hardveru, programskom okruženju i jeziku MicroC, primer razvojne procedure u MPLAB okruženju i detaljni pregled mašinskih instrukcija mikrokontrolera PIC 16F84A koji ilustruju teoretske navode, što predstavlja verovatno i njegovu najveću praktičnu vrednost. U skladu sa ovim, autori smatraju da se udžbenik može koristiti kao pomoćna literatura, kako u teoretskom tako i u praktičnom delu iz pomenutih predmeta.

Takođe, materijal koji je prikazan u udžbeniku može poslužiti i razvojnim inženjerima i stručnjacima u ovoj oblasti kao odlična literatura i priručnik za praktični rad.

Imajući u vidu da na našem jeziku ne postoji dovoljno adekvatne literature, pogotovo za navedene predmete, autori smatraju da je ovo izdanje konkretan doprinos nastavi i efikasnom usvajanju znanja iz ove oblasti kao i pripremama studenata za ispite ili kasniji rad u ovoj veoma dinamičnoj i interesantnoj oblasti. Sa druge strane, način na koji je izložena materija omogućava i znatno širu primenu i podršku inženjerskoj praksi u raznim oblicima i vidovima Embedded sistema, koji su danas veoma aktuelni.

LISTA SLIKA

Slika 2.1	3
Slika 2.2	4
Slika 2.3 – a) Automatski menjač, b) Veš mašina, c) Digitalni termostat d) MP3 plejer	5
Slika 2.4	5
Slika 2.5 – Portovi mikrokontrolera PIC16F84	6
Slika 2.6	6
Slika 2.7	7
Slika 2.8 – Eksterni RC oscillator	8
Slika 2.9	10
Slika 2.10	12
Slika 2.11	13
Slika 2.12	14
Slika 2.13 – Preskaler	15
Slika 2.14 - Tajmersko – brojački modul	16
Slika 2.15 – a) Direktno adresiranje, b) Indirektno adresiranje	17
Slika 2.16 – Preslikavanje BCD u 7-segmenata	20
Slika 2.17 – Prikaz podataka u data EEPROM	20
Slika 2.18 – Prekidna logika	22
Slika 2.19	24
Slika 2.20	24
Slika 3.1 - Razvojno okruženja MicroC	31
Slika 3.2 - Postavljanje opcija editor	32
Slika 3.3 - Code Assistant prozor	33
Slika 3.4	33
Slika 3.5	33
Slika 3.6	34
Slika 3.7 - MicroC Debugger	35
Slika 3.8	35
Slika 3.9	35
Slika 3.10	36
Slika 3.11	36
Slika 3.12	36
Slika 3.13	36
Slika 3.14	36
Slika 3.16 - Watch prozor	37
Slika 3.17 - Stopwatch prozor	37
Slika 3.18 - RAM prozor	38
Slika 3.19 - Prozor greške	38
Slika 3.20	39
Slika 3.21 - Pregled iskorištenja RAM i ROM memorije u formi histograma	39
Slika 3.22 - Funkcije u formi histograma u skladu sa memorijom koja im je dodeljena	39
Slika 3.23 - Raspored funkcija u memoriji mikrokontrolera	40
Slika 3.24 - Stablo poziva sa detaljima za svaku funkciju	40
Slika 3.25 - Pregled svih GPR i SFR registara i njihovih adresa.	41

Slika 3.26 - Lista op-kodova i njihovih adresa u formi HEX koda razumljivog programeru	41
Slika 3.27 - USART Terminal	42
Slika 3.28 - ASCII karta	42
Slika 3.29 - Sedmosegmetni displej	43
Slika 3.30 - EEprom dump memorije	43
Slika 3.31	46
Slika 3.32	46
Slika 3.33	47
Slika 3.34	47
Slika 3.35 - Project Files odeljak	47
Slika 3.36	49
Slika 3.37	49
Slika 3.38	49
Slika 3.39	49
Slika 3.40	50
Slika 3.41	50
Slika 3.42	50
Slika 4.1	62
Slika 4.2	63
Slika 4.3 - Watch prozor	63
Slika 4.4 - Watch prozor	65
Slika 4.5 - Inspekciju koda liniju po liniju.	65
Slika 4.6 - Status posle promene TRISD promenjive	66
Slika 4.7 - Izvršavanje koda do određene tačke	66
Slika 4.8 - Upotreba prekidnih tačaka	67
Slika 4.9 - Izgled displeja	67
Slika 4.10 - Watch prozor	68
Slika 4.11 - Prozor za učitavanje vrednosti	68
Slika 4.12 - Pregled sadržaja EEPROM memorije	69
Slika 4.13 - Pregled koda	70
Slika 4.14 - Sadržaj RAM memorije	70
Slika 5.1 MPLAB – Integrirano razvojno okruženje bazirano na Windows operativnom sistemu	135
Slika 5.1 - Generički MPLAB projekat – Datoteke koje su deo projekta	139
Slika 5.2 - Izvorni kod programa u MPLAB integrisanom okruženju	146
Slika 5.3 - Izgled i opcije u prozoru posle pokretanja BuildNode naredbe (Alt+F10)	147
Slika 5.4 - Prozor Build Results prikazuje listu grešaka koje su uočene u fazi prevođenja programa	148
Slika 5.5 - Prevođenje programa je uspešno završeno	148
Slika 5.6 - Debugiranje programa, izgled osnovnih prozora	149
Slika 5.7 - Asinhrona stimulacija ulaza	151
Slika 5.8 - Aplikacija programatora za Michochip familiju procesora	152
Slika 5.9 - bigPIC5 programator – hardverska komponenta razvojnog Sistema	153
Slika 5.10	153
Slika 5.11 - PICFlash2 programator	153

Literatura

- [1] J. B. Peatmann.: 'Design with PIC Microcontrollers', Prentice-Hall, 1998. ISBN-13: 978-0137592593
- [2] PIC16F84A Data Sheet (02/05/2013), Microchip,
<http://ww1.microchip.com/downloads/en/DeviceDoc/35007C.pdf>
- [3] PIC16F87X Data Sheet (29/11/2012), Microchip,
<http://ww1.microchip.com/downloads/en/DeviceDoc/30292D.pdf>
- [4] [PIC Microcontrollers tutorial: Index - Rickey's World of Microcontrollers ...](https://www.8051projects.net/pic_tutorial/introduction.php)
https://www.8051projects.net/pic_tutorial/introduction.php