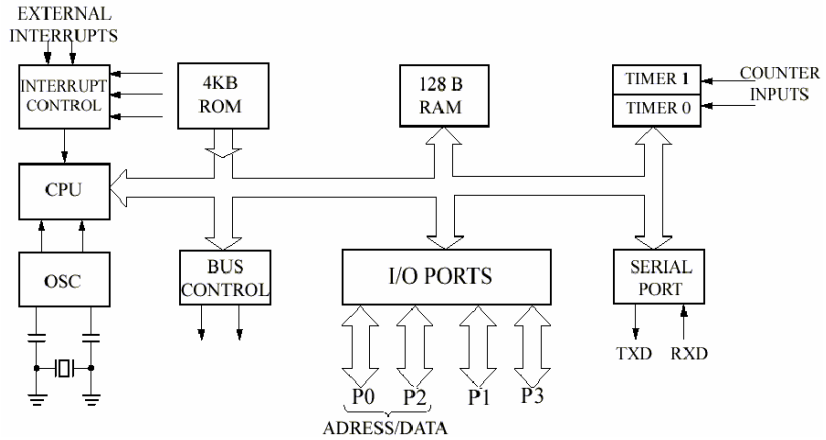
Vežba 1,2
Mikrokontroler Intel 8051

1. ARHITEKTURA MIKROKONTROLERA INTEL 8051

U familiji INTEL-ovih mikrokontrolera izvedenih u vidu jednog čipa mikrokontroler, INTEL 8051 je sintetizovan namenski da služi kao procesor u sistemima digitalnog upravljanja. Osnovna varijanta čipa INTEL 8051 ima strukturu prikazanu na sl.1.1. Unutar čipa se nalaze centralna procesorska jedinica (CPU), ROM memorija, RAM memorija, tajmeri, brojači, paralelni i serijski ulazi/izlazi. Sem osnovne verzije, na tržištu se može naći veliki broj varijanti ovog mikrokontrolera, koji se od osnovne verzije razlikuju u broju i vrsti periferijskih komponenti, veličini i tipu ugrađene memorije (RAM i ROM), kao i po mehaničkoj izvedbi (tip kućišta i broj nožica). Ovo uputstvo obuhvata osnovnu varijantu mikrokontrolera 8051, uz opis dodatnih elemenata za mikrokontroler 8052.

Osnovne karakteristike mikrokontrolera INTEL 8051 su:

- 8-bitni CPU;
- 4 KB (8 KB za 8052) interne ROM ili EPROM memorije za čuvanje programa, sa mogućnošću proširenja do 64 KB spoljašnje memorije;
- 128 (256 za 8052) bajtova RAM memorije namenjene za upisivanje i čitanje podataka. U ovoj memoriji se nalaze 4 registarske banke od po 8 registara, kao i stek memorija;
- 64 KB adresnog prostora memorije podataka;
- 64 KB adresnog prostora programske memorije;
- 8-bitni pokazivač stek memorije, koji pokriva internu memoriju
- Dva (tri za 8052) programabilna 16-bitna tajmera/brojača;
- Programabilni serijski ulaz/izlaz (puni dupleks).
- Četiri 8-bitna ulazno/izlazna priključka (portovi P0, P1, P2 i P3);
- Tajmerski i ulazno/izlazni prekidi sa dva nivoa prioriteta;
- 111 naredbi;
- Aritmetičko-logička jedinica (ALU) koja može da izvršava aritmetičke operacije sabiranja, oduzimanja, množenja i deljenja, kao i logičke operacije I, ILI, EXILI, komplement i negacija;
- 256 adresabilnih bita za rad sa Bulovom algebrom, od toga 128 bita u internoj memoriji i 128 bita pridruženih postojećim internim registrima.

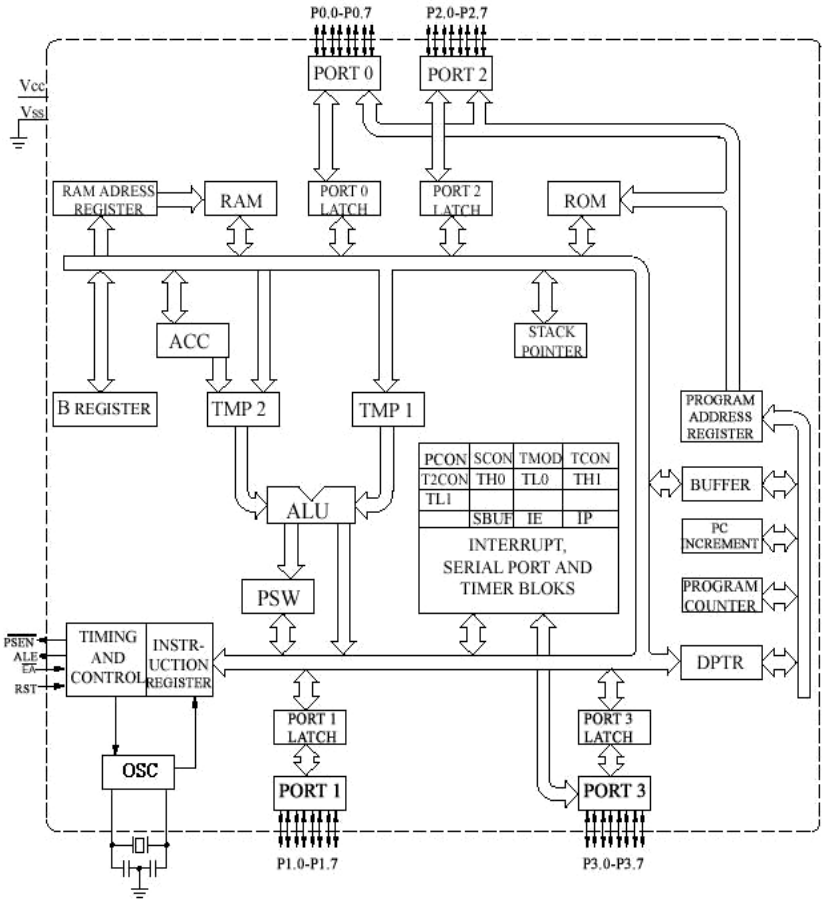


Slika 1.1 Struktura čipa mikrokontrolera INTEL 8051

Ovako sintetizovan mikrokontroler u mnogome olakšava posao projektanta hardvera, smanjuje ukupno hardversko okruženje i broj spoljašnjih veza i na taj način povećava pouzdanost sistema.

Na sl. 1.2 prikazana je detaljnija arhitektura mikrokontrolera INTEL 8051. Ovde se vide ranije pomenute komponente: aritmetičko-logička jedinica (**ALU**) sa parom registara (**TMP1**, **TMP2**) za

privremeno upisivanje podataka, akumulator (**ACC**) sa pomoćnim registrom **B**, statusni registar (**PSW**), registar naredbi sa dekoderom, programski brojač (**PC**) i registar za inkrementiranje (povećanje za 1) programskog brojača (**PC-incrementer**).



Slika 1.2 Detaljnija struktura mikrokontrolera INTEL 8051

Pomoćni registar **B**, koji se naziva multiplikativnim registrom, služi za smeštanje drugog operanda za aritmetičke operacije množenja i deljenja. Posle izvršene operacije množenja ili deljenja u njemu se nalazi viši bajt rezultata množenja ili ostatak deljenja, respektivno. Zajedno sa **ACC** pomoćni registar B čini registarski par.

Pokazivač steka (**SP**-Stack Pointer) služi za adresiranje vrha (najviše lokacije) stek memorije. Ovaj registar se inkrementira prilikom upisivanja podataka u stek, a dekrementira prilikom čitanja podataka iz ove memorije.

Mikrokontroler INTEL 8051 poseduje 4 prihvatna registra (**LATCH**-a) za čuvanje stanja izlaza na portovima P0, Pl, P2 i P3. Registar serijskog prenosa (**SBUF**) služi za upis podatka koji se šalje i čitanje podatka koji se prima preko serijske veze. Registarski parovi (**TH0, TL0**) i (**TH1, TL1**) čine dva 16-bitna tajmera ili brojača.

Za kontrolu i upisivanje statusa prilikom prekida (*INTERRUPT*-a) za tajmere, brojače i za serijski prenos podataka koriste se registri specijalne namene označeni na sl 1.2 sa **IP**, **IE**, **TMOD**, **TCON**, **SCON** i **PCON**.

IP služi za određivanje nivoa prioriteta prekida, **IE** za maskiranje (dozvolu ili zabranu) prekida, **TMOD** i **TCON** za određivanje načina rada tajmera i brojača, **SCON** za kontrolu serijskog prenosa i **PCON** za dodatnu kontrolu serijskog prenosa i režim rada mikrokontrolera.

Memorijski adresni prostor mikrokontrolera INTEL 8051 je podeljen u dva osnovna dela: adresni prostor rezervisan za programe (*Code Address Space*) i adresni prostor rezervisan za podatke (*Data Address Space*).

Mikrokontroler može da adresira 64KB programske memorije; interno u samom čipu ima 4KB, a ostatak od 60KB je predviđen kao spoljašnja memorija. *Programska memorija* je tipa ROM (u nekim varijantama i EPROM ili FLASH), a sem izvođenja programa moguće je i čitanje nekih konstantih podataka.

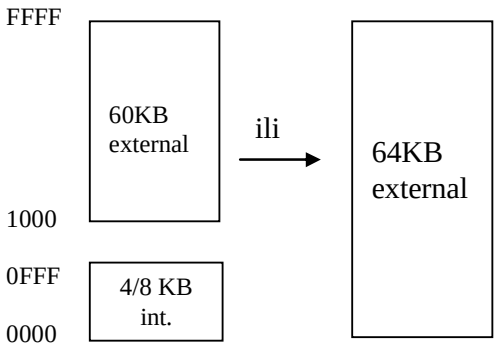
Memorija podataka je tipa RAM, a sastoji se iz interne (128/256 bajtova) i eksterne memorije (do 64KB). Pri tome, ako nema previše podataka, eksterna memorija ne mora da se koristi. Internu memoriju je moguće adresirati direktno registarski ili registarski indirektno. Od toga u prostoru od 16 bajtova moguće je adresirati svaki bit (*Bit Addressable Segment*). U okviru interne memorije podataka nalazi se i *stek*, koji je organizovan tako da se poslednji upisani podatak prvi čita (Last Input First Output ili *LIFO*-princip), a koristi se za privremeno čuvanje sadržaja brojača naredbi prilikom poziva na potprograme i za pamćenje adrese izvršenja programa u slučaju prekida. Napomenimo još da se eksternoj memoriji se uvek pristupa indirektno, preko odgovarajućih registara.

2. ORGANIZACIJA MEMORIJE

Kao što je već rečeno, memorijski adresni prostor mikrokontrolera INTEL 8051 je podeljen u dva osnovna dela: adresni prostor rezervisan za programe (*Code Address Space*) i adresni prostor rezervisan za podatke (*Data Address Space*).

2.1 Programska memorija

Mikrokontroler može da adresira 64KB programske ROM memorije, slika 2.1.

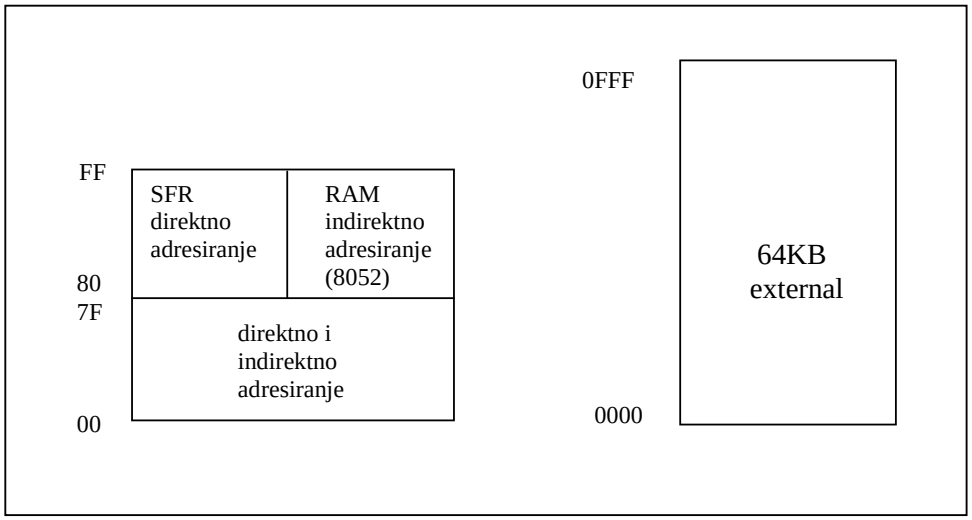


Slika 2.1 Programska memorija

Donjih 4/8 KB programske memorije može biti interna memorija (nalazi se unutar čipa) ako je $\overline{EA}$=Vcc, ili spoljašnja memorija ako je $\overline{EA}$=0. Signal $\overline{PSEN}$ se koristi za pristup spoljašnjoj programskoj memoriji, tako što se vodi na $\overline{OE}$ ulaz EPROM-a. Kada se koristi spoljašnja memorija, port **P2** služi za adresiranje višeg bajta a **P0** za adresiranje nižeg bajta te memorije.

2.2 Memorija za podatke

Memorija podataka je tipa RAM, slika 2.2, a sastavljena je od interne (128 bajtova) i eksterne (do 64KB) memorije. Sem toga, u zoni interne memorije (u opsegu adresa 128 - 255) se nalazi i blok specijalnih registara (*S*pecial *F*unction *R*egisters - *SFR*) za kontrolu periferijskih jedinica i rada mikrokontrolera.



Slika 2.2 Memorija za podatke

F8h									FFh
F0h	B								F7h
E8h									EFh
E0h	ACC								E7h
D8h	PSW								DFh
D0h									D7h
C8h	(T2CON)	(T2MOD)	(RCAP2L)	(RCAP2H)	(TL2)	(TH2)			CFh
C0h									C7h
B8h	IP								BFh
B0h	P3								B7h
A8h	IE								AFh
A0h	P2								A7h
98h	SCON	SBUF							9Fh
90h	P1								97h
88h	TCON	TMOD	TL0	TL1	TH0	TH1			8Fh
80h	P0	SP	DPL	DPH				PCON	87h

Memorijska mapa SFR zone

** - T2MOD registar ne pripada standardnom mikrokontroleru 8052, ali postoji kod Atmel AT89C52.*

Tabela 2.1 Raspored registara sa specijalnim funkcijama - SFR

30h-7Fh	Slobodna memorija	
20h-2Fh	Bit-adresibilna memorija	
18h-1Fh	**R0..R7**	**GRUPA 3**
10h-17h	**R0..R7**	**GRUPA 2**
08h-0Fh	**R0..R7**	**GRUPA 1**
00h-07h	**R0..R7**	**GRUPA 0**

Memorijska mapa internog RAM-a

Slika 2.3 Memorijska mapa internog RAM-a i SFR zone

Na slici 2.3, registri dati u zagradi postoje samo u mikrokontroleru 8052, a registri označeni sa dvostrukim okvirom su bit-adresabilni. Na slici 2.4. uočavamo registar **PSW** (Program Status Word), koji je veoma važan za rad mikrokontrolera. Kod njega nije zanimljiva binarna vrednost koju sadrži, nego isključivo stanje pojedinačnih bita.

7	6	5	4	3	2	1	0
C	AC	F0	RS1	RS0	OV	-	P

Slika 2.4 PSW registar

C (PSW.7) Carry Flag (Bit prenosa). Ovo je ekstenzija (deveti bit) za sve aritmetičke operacije i instrukcije pomeranja (šiftovanja), a takođe je i glavni registar za 1-bitne operacije.

AC (PSW.6) Auxiliary Carry Flag (pomoćni bit prenosa), samo za BCD operacije, a odnosi se na prenos između nižeg i višeg nibla (donja i gornja grupa od po 4 bita). Koristi ga uglavnom mikrokontroler preko naredbi za BCD konverziju.

F0 (PSW.5) Fleg 0 stoji na raspolaganju programeru kao bit za univerzalnu upotrebu.

RS1 i RS0 (PSW.4 i PSW.3) Register Select, služe za izbor dela internog RAM-a u kome je smeštena tekuća grupa registara (registarska banka) **R0**-**R7**, u skladu sa sledećom tabelom:

RS1	RS0	Mesto u RAM-u
0	0	Grupa 0 00h-07h
0	1	Grupa 1 08h-0Fh
1	0	Grupa 2 10h-17h
1	1	Grupa 3 18h-1Fh

Tabela 2.2 Odabir aktivne grupe registara pomoću RS1 i RS0

OV (PSW.2) Overflow (prekoračenje). Setuje se (postavlja na 1) ako je rezultat aritmetičke operacije sa predznakom takav da ne može da se prikaže u jednom bajtu, tj. dođe do prekoračenja opsega (na primer, sabiranjem dva pozitivna broja dobije se negativan broj ili obrnuto). Ako nema prekoračenja, ovaj bit će biti 0.

(PSW.1) Rezervisano od stane proizvođača za budući razvoj čipa.

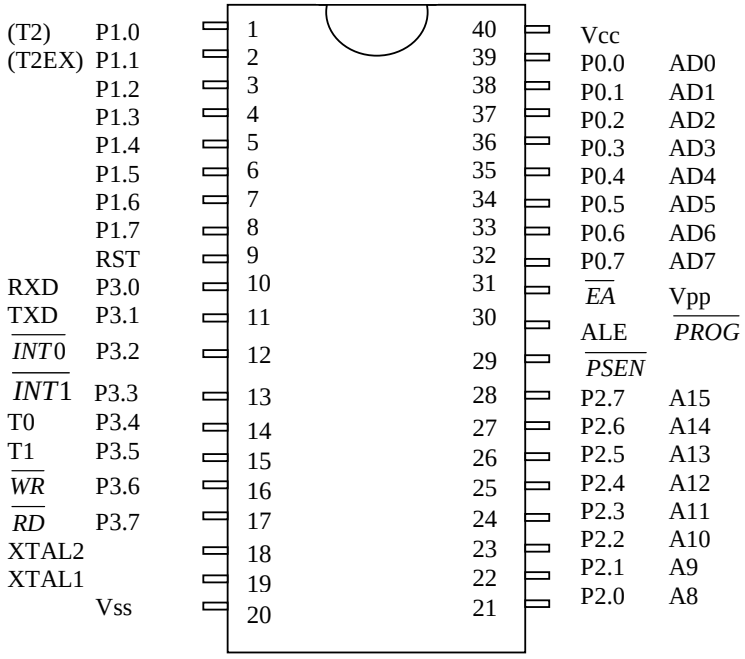
P (PSW.0) Parnost. Ako je broj setovanih bita u akumulatoru paran, ovaj bit će biti postavljen na 1, a ako je neparan, biće resetovan na 0. Uglavnom se koristi za generisanje bita parnosti kod slanja bajta na serijski port ili za testiranje parnosti posle serijskog prijema.

Data Pointer (DPTR) je 16-bitni registar, koji se sastoji od dva 8-bitna registra: **DPH** (*Data Pointer High* – viši bajt) i **DPL** (*Data Pointer Low* – niži bajt).

Ostali registri posebne namene će biti objašnjeni u poglavljima o tajmerima/brojačima i prekidima.

3. RASPORED I FUNKCIJE POJEDINIH NOŽICA

Na slici 3.1 prikazan je izgled mikrokontrolera 8051. Ako su signali nadvučeni to znači da je aktivan nizak nivo signala a pasivan visok.



Slika 3.1 Raspored nožica mikrokontrolera

3.1 Opis funkcija pojedinih nožica

Od 1 do 8 - Port 1:

Svaka od ovih nožica može da se koristi kao ulazni ili izlazni priključak, prema potrebi. Za 8052, **P1.0** (**T2**) je spoljašnji brojački ulaz za tajmer 2 a **P1.1** (**T2EX**) je spoljašnja kontrola (triger) za tajmer 2.

9 - Reset:

- Visok logički nivo na ovom ulazu resetuje sve interne registre (registre dovodi u stanje 00000000), sa sledećim izuzecima:
  - **P0**, **P1**, **P2** i **P3** (izlazni registri svih spoljnih portova) se dovode u stanje 11111111
  - **SBUF** se ne menja
  - **SP** se dovodi u stanje 00000111 (07h)
  - Neki biti u registrima **IP**, **IE** i **PCON** fizički ne postoje, pa tako ne mogu ni da se resetuju
  - Sadržina celog internog RAM-a se ne menja
- Najvažnija posledica aktiviranja **RESET** ulaza je da se **PC** (Program Counter) resetuje, tako da će započeti izvršavanje programa od adrese 0000h.

od 10 do 17 - Port 3:

- Ako se koristi kao univerzalni ulaz ili izlaz, po svemu je sličan portu 1, ali na svakoj nožici ima još po neku specijalnu funkciju:
- 10 (**P3.0**) **RXD** - Serijski ulaz za asinhronu komunikaciju (mod 1, 2 i 3) ili serijski izlaz za sinhronu komunikaciju (mod 0)
- 11 (**P3.1**) **TXD** - Serijski izlaz za asinhronu komunikaciju (mod 1, 2 i 3) ili taktni (clock) izlaz sa sinhronu komunikaciju (mod 0)
- 12 (**P3.2**) **INT0** - Ulaz za prekid (interapt) 0
- 13 (**P3.3**) **INT1** - Ulaz za prekid (interapt) 1
- 14 (**P3.4**) **T0** - Ulaz spoljnjeg takta za brojač 0
- 15 (**P3.5**) **Tl** - Ulaz spoljnjeg takta za brojač 1
- 16 (**P3.6**) **WR** - Signal za upis u spoljnu memoriju
- 17 (**P3.7**) **RD** - Signal za čitanje iz spoljne memorije

18 i 19 - X2 i X1:

Izlaz i ulaz internog oscilatora. Ako se koristi kvarc-kristal za stabilizaciju učestanosti oscilatora (što je najčešći slučaj), on se vezuje za ove dve nožice, s tim što na svaku nožicu (prema masi) treba dodati još po jedan kondenzator od 20-40pF. Ovoje potrebno da bi se sprečilo oscilovanje na nekom višem harmoniku. Opseg učestanosti je od 1 do 12 MHz, a izrađuju se i mikrokontroleri koji rade i na znatno višim frekvencijama.

20 - Masa

Od 21 do 28 - Port 2 ili adrese A8 do A15:

Ako se koristi mikrokontroler sa internim ROM-om i nema spoljnjeg ROM-a ili RAM-a, mogu se koristiti sve linije ovog porta kao univerzalni ulazi ili izlazi. Ako se koristi spoljna memorija, onda su ovo visoki adresni izlazi, od **A8** do **A15**. U tom slučaju, čak i ako se koriste samo neke adrese, preostale nožice ovog porta ne mogu da se koriste kao ulazi ili izlazi.

29 - PSEN: Program Select Enable (aktiviranje spoljašnjeg ROM-a):

Normalno se ovaj izlaz spaja sa **CS** ili **OE** ulazom na spoljnom EPROM-u, jer ga mikrokontroler aktivira (dovodi na nizak nivo) svaki put kad čita bajt iz programske memorije (za kontrolu spoljašnjeg RAM-a se koriste druge nožice).

30 - ALE: Address Latch Enable (Upis u adresni registar):

Da bi sve željene funkcije spakovao u standardno kućište od samo 40 nožica, Intel je morao da pribegne multipleksiranju nekih signala. Tako je port **P0** dobio dve funkcije: izlazne adrese **A0**-**A7** i ulaz/izlaz podataka **D0**-**D7**. Pre svakog očitavanja programa iz spoljne memorije ili prozivanja RAM-a mikrokontroler na **P0** prosleđuje niži bajt adresnog registra i aktivira izlaz **ALE**. Spoljni registar (najčešće **LATCH** registar tipa 373 ili 573 iz **TTL** 74xx familije) na visok nivo **ALE** memoriše stanje **P0**, a izlazi ovog registra se koriste kao **A0**-**A7**. U drugom delu mašinskog ciklusa mikrokontrolera **P0** se koristi kao magistrala podataka (*Data Bus*).

31 - EA: External Access (Pristup spoljašnjem ROM-u):

Ako je ovaj ulaz nizak, mikrokontroler će sve instrukcije čitati iz spoljnog ROM-a, bez obzira da li ima interni ROM, a ako je visok, prvih 4 KB (8051, 8751) ili 8 KB (8052, 8752) će čitati iz internog, a sve ostalo do kraja adresnog prostora iz eksternog ROM-a. Ako se koristi 8031 ili 8032, na ovaj ulaz treba uvek dovesti nizak nivo (najbolje je spojiti ga sa masom).

od 32 do 39 - Port 0, Adrese A0-A7 ili magistrala podataka:

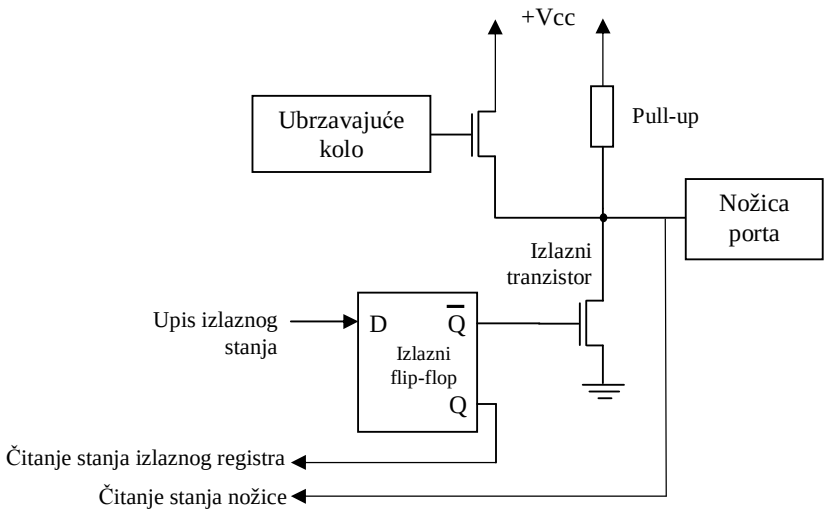
Slično portu **P2**, i port **P0** može da se koristi kao univerzalni ulaz i izlaz samo ako se ne koristi spoljna memorija. Ako se koristi, tada je **P0** adresni izlaz za **A0**-**A7** kad je **ALE** visok, a magistrala podataka (*Data Bus*) kada je **ALE** nizak.

40 - Napajanje +5V

3.2 Hardverska struktura portova

Kao što je prikazano, postoje četiri 8-bitna porta, pri čemu kod konfiguracija sa spoljnom memorijom portovi P0 i P2 služe za adresiranje i pristup podacima iz memorije, a P1 i P3 su univerzalni i koriste se onako kako to konkretan projekat zahteva. Svi ulazi i izlazi su kompatibilni sa TTL standardom, a to praktično znači da možemo direktno da ih spajamo sa TTL ili kompatibilnim kolima. Ipak, neke specifičnosti portova treba imati u vidu kada se projektuje okruženje ovih mikrokontrolera.

Svi portovi su realizovani sa otvorenim drejnom (*Open Drain*), uz dodatni MOSFET tranzistor u spoju aktivnog otpornika prema napajanju (pull-up, oko 50KΩ), kao na slici 3.2. Ovakvo rešenje obezbeđuje dobru logičku nulu (kada je izlazni tranzistor uključen). Kada je ovaj tranzistor isključen, tada samo aktivni otpornik određuje logičku jedinicu, pri čemu se odgovarajuća nožica ponaša i kao ulazni priključak, jer spoljašnje kolo može da obori napon na logičku nulu (rešenje je poznato i pod nazivom “ožičeno I”).



Slika 3.2 Pojednostavljeni prikaz nožica porta

U slučaju kada se na izlaz porta postavlja logička jedinica, uključuje se kratkotrajno dodatni tranzistor preko ubrzavajućeg kola, čime je obezbeđeno savlađivanje izlaznih kapacitivnosti.

Zavisno od tipa instrukcije, moguće je čitanje stanja izlazne nožice, ali i stanja izlaznog flip-flopa. Instrukcije koje samo čitaju port uzimaju informaciju o stanju direktno sa nožice (označeno kao "čitanje stanja nožice"), dok instrukcije koje modifikuju port, tj. pročitaju stanje, promene ga i zatim ponovo upišu na port, pri čitanju uzimaju stanje izlaznog flip-flopa, a ne izlazne nožice (označeno kao "čitanje stanja izlaznog registra"). Ovakve instrukcije su poznate pod nazivom "Read-Modify-Write" (čitanje-izmena-upis), a primeri su:

```
ORL   P1,#72h     ; Port se pročita, uradi se logička
                  ; operacija OR, a zatim se vrednost
                  ;piše u port
SETB  P1.2        ; Port se pročita, postavi se jedan
                  ; bit, a zatim se vrednost upiše u
                  ; port
```

Primeri instrukcija koje čitaju izlazno stanje su:

```
MOV   A,P2
ANL   C,P1.1
```

4. INSTRUKCIJE

Set instrukcija za 8051 obuhvata ukupno 111 instrukcija, od kojih je 49 jednobajtnih, 45 dvobajtnih i 17 trobajtnih. Ako se uzmu u obzir i varijante istih instrukcija (na primer ista instrukcija primenjena na različite registre iz registarske banke), ukupno postoji 255 instrukcija. Generalni format koji važi u svakom asembleru za ove instrukcije je sledeći (mada postoje i varijacije u ovom formatu):

mnemonik odredište, izvor

Mnemonik nosi kod instrukcije, a operandi ***odredište*** i ***izvor*** samo nose podatke ili adresu odakle se uzimaju podaci kojima se vrši operacija. Ako postoje dva operanda, onda se rezultat operacije uvek smešta u prvi koji je naveden. Recimo, ako instrukcija glasi ADD A,B to je isto kao da smo u nekom višem programskom jeziku napisali A=A+B.

4.1 Prenos podataka

MNEMONIC	OPERATION	ADRESSING MODES				EXECUTION TIME (μs)
		DIR	IND	REG	IMM	
MOV A, <src>	A= <src>	X	X	X	X	1
MOV <dest>, A	<dest>= A	X	X	X		1
MOV <dest>,<src>	<dest>=<src>	X	X	X	X	2
MOV DPTR,#data16	DPTR=16-bit constant				X	2
PUSH <src>	INC SP:MOV"@SP".<src>	X				2
POP <dest>	MOV <dest>,"@SP":DEC SP	X				2
XCH A, <byte>	ACC and <byte> exchange data	X	X	X		1
XCHD A, @Ri	ACC and @Ri exchange low nibbles		X			1

Slika 4.1 Prenos podataka sa instrukcijama koje pristupaju internoj memoriji

ADRESS WIDTH	MNEMONIC	OPERATION	**EXECUTION TIME (μs)**
8 bits	MOVX A,@Ri	Read external RAM @Ri	2
8 bits	MOVX @Ri,A	Write external RAM @Ri	2
16 bits	MOVX A,@DPTR	Read external RAM @DPTR	2
16 bits	MOVX @DPTR,A	Write external RAM @DPTR	2

Slika 4.2 Prenos podataka sa instrukcijama koje pristupaju spoljašnjoj memoriji

MNEMONIC	OPERATION	EXECUTION TIME (μs)
MOVC A,@A+DPTR	Read program memory at (A+DPTR)	2
MOVC A,@A+PC	Read program memory at (A+PC)	2

Slika 4.3 Instrukcije za čitanje lookup tabele

Kod instrukcija iz grupe prenosa podataka (MOV, MOVC i MOVX) zarez između operanda možemo da shvatimo kao strelicu nalevo, jer se uvek sadržina drugog operanda upisuje u prvi, pri čemu se drugi ne menja. Ovo ne važi za instrukcije XCH i XCHD, gde operandi međusobno izmenjuju vrednosti (kod XCHD samo četiri najniža bita).

Sve instrukcije iz ove grupe obavljaju prenos 8-bitnog podatka, sa dva izuzetka: XCHD vrši razmenu 4-bitnog podatka, a MOV DPTR, #data upisuje 16-bitnu vrednost u registre DPH i DPL.

Instrukcije čija svrha je upis bajta iz programske memorije ili komunikacija sa spoljnim RAM-om imaju 16-bitno adresno polje. Prvi slučaj (upis bajta iz programske memorije u akumulator) se uglavnom koristi za lookup tabele, gde se pozicija u tabeli izračunava iz bazne adrese početka tabele (obično se smešta u DPTR) i ofseta (koji je u akumulatoru), pa je zbog toga ova instrukcija unapred proširena i glasi MOVC A,@A+DPTR. Ona izvršava tačno ono što možemo i da pretpostavimo: najpre sabere 16-bitni DPTR sa 8-bitnim akumulatorom, pa rezultat iskoristi za adresiranje programske memorije, pročita bajt i upiše ga u akumulator. Ako nam treba samo 16-bitno adrsiranje bez ofseta, pre ove instrukcije treba upisati 0 u akumulator (CLR A).

Instrukcija MOVC A, @A+PC radi isto to, ali umesto DPTR uzima 16-bitnu adresu sledeće instrukcije u programu. Sledeći potprogram uzima vrednost člana pod rednim brojem koji je upisan u akumulator (u ovom slučaju od 0 do 4), iz tabele koja sledi odmah iza potprograma:

```
REL_PC:   INC   A
          MOVC A, @A+PC
          RET
          DB   27h
          DB   41h
          DB   00h
          DB   5Eh
          DB   7Fh
```

Instrukcija INC A je potrebna da bi se u kalkulaciji preskočila instrukcija RET, koja u programu zauzima jedan bajt. Ako ovde ima još koda, onda broj njegovih bajtova treba sabrati sa akumulatorom.
Instrukcije koje komuniciraju sa spoljnim RAM-om (MOVX) mogu svoje 16-bitno adresno polje da uzmu iz DPTR, ili kombinovanjem P2 (visoki bajt) i R0 ili Rl (niski bajt).
Nijedna od instrukcija iz ove grupe ne utiče na flegove, izuzev POP i MOV instrukcija kojima je prvi operand PSW, jer njihova funkcija i jeste izmena sadržaja registra koji sadrži flegove.

4.2 Aritmetičke instrukcije

MNEMONIC	OPERATION	ADRESSING MODES				EXECUTION TIME (μs)
		DIR	IND	REG	IMM	
ADD A,<byte>	A=A+<byte>	X	X	X	X	1
ADDC A,<byte>	A=A+<byte>+C	X	X	X	X	1
SUBB A,<byte>	A=A-<byte>-C	X	X	X	X	1
INC A	A=A+1					1
INC <byte>	<byte>=<byte>+1	X	X	X		1
INC DPTR	DPTR =DPTR+1					2
DEC A	A=A-1					1
DEC <byte>	<byte>=<byte>-1	X	X	X		1
MUL AB	B:A = B*A					4
DIV AB	A=Int[A/B] B=Mod[A/B]					4
DA A	Decimal Adjust					1

Mikrokontroler 8051 podržava sve četiri osnovne aritmetičke operacije. Samo osmobitna aritmetika je podržana (sa izuzetkom INC DPTR, koji uvećava stanje 16-bitnog registra), i to sa pozitivnim brojevima, mada overflow fleg omogućava sabiranje i oduzimanje sa predznakom. Sabiranje može da se obavlja i sa binarno kodiranim decimalnim (BCD) brojevima, kod kojih osmobitni registar nosi dve odvojene decimalne cifre, svaku u po 4 bita.

Sabiranje

INC (increment) uvećava operand za 1.
ADD sabira akumulator sa izvornim operandom i smešta rezultat u A. Bit C je deveti bit rezultata, zapravo bit prenosa za sabiranje višebajtnih brojeva.
ADDC (ADD with Carry) sabira akumulator sa izvornim operandom i sa bitom C, a rezultat smešta u akumulator.

Primer sabiranja višebajtnih binarnih brojeva:
Ako treba DPTR sabrati sa 16-bitnim binarnim brojem čiji je viši bajt u R4, a niži u R5 i rezultat smestiti u DPTR, program može da izgleda ovako:

```
MOV   A,DPL      ; uzmi niži bajt
ADD   A,R5       ; saberi
MOV   DPL,A      ; smesti rezultat nižeg bajta
MOV   A,DPH      ; uzmi viši bajt
ADDC  A,R4       ; saberi i dodaj prenos
MOV   DPH,A      ; smesti rezultat višeg bajta
```

DA A (Decimal Add Adjust for Addition) koriguje rezultat posle binarnog sabiranja, tako da odgovara za BCD brojeve. Na primer, ako akumulator i registar B sadrže BCD brojeve (00-99) onda će sledeći program sabrati ova dva BCD broja i ispravan BCD rezultat smestiti u akumulator:

```
ADD  A,B        ; saberi binarno
DA   A          ; koriguj rezultat
```

Ako je rezultat posle instrukcije DA A veći od 99, tada je bit C (prenos) setovan.

Oduzimanje

DEC (decrement) umanjuje operand za 1.

SUBB (subtract with borrow) oduzima drugi operand od prvog (koji je uvek akumulator). Pošto instrukcija SUB (oduzimanje na koje bit C ne utiče) ne postoji u setu za 8051, možemo da ga sintetišemo tako što pre SUBB izvedemo jedno CLR C (C=0).

Oduzimanje višebajtnih binarnih brojeva se vrši slično kao kod sabiranja. Od DPTR treba oduzeti 16-bitni binarni broj čiji je viši bajt u R4, a niži u R5 i rezultat smestiti u DPTR:

```
MOV   A,DPL     ; uzmi niži bajt
CLR   C         ; da bi SUBB radio kao SUB
SUBB  A,R5      ; oduzmi R5
MOV   DPL,A     ; smesti rezultat nižeg bajta
MOV   A,DPH     ; uzmi viši bajt
SUBB  A,R4      ; oduzmi R4 i prenos
MOV   DPH,A     ; smesti rezultat višeg bajta
```

Množenje

MUL AB množi dva 8-bitna broja bez predznaka, od kojih je jedan smešten u akumulator, a drugi u B registar, i 16-bitni rezultat smešta u akumulator (niži bajt) i registar B (viši bajt). Pošto je ovde rezultat 16-bitni, prekoračenje nije moguće pa je bit C uvek resetovan, a bit OV je setovan ako je rezultat veći od 255 (odnosno ako je registar B veći od 0), a resetovan ako je B=0.

Deljenje

DIV AB deli akumulator sa registrom B i celobrojni rezultat smešta u A, a ostatak deljenja u B. Deljenje sa nulom (ako je B=0) rezultira sa nepredvidivim sadržajem registara, i u tom slučaju bit OV će biti setovan. Inače, u normalnom slučaju (kad je B>0) bitovi OV i C su resetovani.

Logičke instrukcije

MNEMONIC	OPERATION	ADRESSING MODES				EXECUTION TIME (μs)
		DIR	IND	REG	IMM	
ANL A,<byte>	A=A AND <byte>	X	X	X	X	1
ANL <byte>,A	<byte>=<byte> AND A	X				1
ANL <byte>,#data	<byte>=<byte> AND #data	X				2
ORL A, <byte>	A=A OR <byte>	X	X	X	X	1
ORL <byte>,A	<byte>=<byte> OR A	X				1
ORL <byte>,#data	<byte>=<byte> OR #data	X				2
XRL A,<byte>	A=A XOR <byte>	X	X	X	X	1
XRL <byte>,A	<byte>=<byte> XOR A	X				1
XRL <byte>,#data	<byte>=<byte> XOR #data	X				2
CLR A	A=00H					1
CPL A	A= not A					1
RL A	Rotate A left 1 bit					1
RLC A	Rotate left through Carry					1
RR A	Rotate A right through Carry					1
RRC A	Rotate right through Carry					1
SWAP A	Swap nibbles in A					1

MNEMONIC	OPERATION	EXECUTION TIME (μs)
ANL C,bit	C = C AND bit	2
ANL C,/bit	C = C AND NOT bit	2
ORL C,bit	C = C or bit	2
ORL C,/bit	C = C or NOT bit	2
MOV C,bit	C = bit	1
MOV bit,C	bit = C	2
CLR C	C = 0	1
CLR bit	Bit = 0	1
SETB C	C = 1	1
SETB bit	bit = 1	1
CPL C	C = NOT C	1
CPL bit	Bit = NOT bit	1
JC rel	Jump if C = 1	2
JNC rel	Jump if C = 0	2
JB bit,rel	Jump if bit = 1	2
JNB bit,rel	Jump if bit = 0	2
JBC bit,rel	Jump if bit = 1;CLR bit	2

CLR resetuje navedeni bit, a CLR A resetuje sve bitove u akumulatoru.
CPL komplementira navedeni bit (menja mu stanje), a CPL A komplementira sve bitove u akumulatoru.
SETB setuje navedeni bit.
RL (RR) rotira nalevo (nadesno) akumulator. Bit 7 (bit 0) dolazi na mesto bita 0 (bita 7). Bit koji izlazi sa krajnje pozicije i ulazi sa druge strane.
RLC (RRC) rotira nalevo (nadesno) akumulator kroz bit C (9-bitna rotacija). Tako će bit C da se preseli u bit 0 (bit 7), a bit 7 (bit 0) u bit C.
SWAP rotira akumulator 4 puta nalevo (odnosno na desno), tj. vrši zamenu nižeg i višeg nibla akumulatora.
ANL izvodi logičku operaciju AND između prvog i drugog operanda, a rezultat smešta u prvi operand. Ova operacija je moguća sa operandima koji zauzimaju jedan bajt ili jedan bit.
ORL izvodi logičku operaciju OR između prvog i drugog operanda, a rezultat smešta u prvi operand. Ova operacija je moguća sa operandima koji zauzimaju jedan bajt ili jedan bit.

XRL izvodi logičku operaciju XOR između prvog i drugog operanda (oba su 8-bitna), a rezultat smešta u prvi operand.

4.3 Potprogrami

ACALL (Absolute Call) i LCALL (Long Call) pozivaju potprograme na sledeći način: najpre 16-bitnu adresu sledeće instrukcije u programu stave na stek i uvećaju SP za dva, a onda u PC upišu pridruženu adresu. Ta adresa u slučaju poziva ACALL može da se nađe samo u istom bloku od 2K u kome je i prva instrukcija posle poziva (ona koja se stavlja na stek), a kod LCALL nema ograničenja, jer je adresa 16-bitna. Razlog za ograničenje kod ACALL je taj što ova instrukcija zauzima samo 2 bajta (prema 3 bajta koje zauzima LCALL), i 5 bitova zauzima kod instrukcije, pa tako za adresno polje ostaje samo 11 bitova.
RET izaziva akciju koja je inverzna instrukciji ACALL odnosno LCALL: sa steka se uzimaju dva bajta i smeštaju u programski brojač. SP se umanjuje za dva. To je povratak iz potprograma, jer će posle ovoga program nastaviti da se izvršava od instrukcije koja je sledila iza poziva.
RETI radi isto što i RET, samo što se uz to omogućava i prekid tekućeg nivoa prioriteta. Ovo je standardna instrukcija za povratak iz prekidnog potprograma.

4.4 Bezuslovni skokovi

MNEMONIC	OPERATION	EXECUTION TIME (μs)
JMP addr (*)	Jump to addr	2
JMP @A+DPTR	Jump to A+DPTR	2
CALL addr	Call subroutine at addr	2
RET	Return from subroutine	2
RETI	Return from interrupt	2
NOP	No operation	1

(*) Pod naredbom JMP se podrazumeva jedna od tri naredbe skoka, AJMP, LJMP i SJMP.

AJMP (Absolute jump) i LJMP (Long Jump) prenose pridruženu adresu u programski brojač, tako da će program nastaviti da se izvršava nadalje od te adrese. AJMP i LJMP su po broju bitova adrese analogni pozivima potprograma ACALL i LCALL.

SJMP (Short jump) je dvobajtna instrukcija koja omogućava skokove u opsegu -128 do +127 u odnosu na sledeću instrukciju u programu. Sem dužine skoka, razlika između SJMP i AJMP je u tome što SJMP može da preskoči blok od 2KB, dok AJMP uvek skače unutar ovakvog bloka.
U tabeli, kod instrukcije “JMP addr”, reč JMP se odnosi na sve tri instrukcije skoka, AJMP, LJMP i SJMP.

JMP @A+DPTR skače na adresu koja se izračunava tako što se sabere 16-bitni registar DPTR sa ofsetom koji se nalazi u akumulatoru. Ovo je od koristi kod složenih grananja u potprogramu, kad od stanja akumulatora zavisi na koju adresu treba izvesti skok (tabelarni skokovi). Sledeći program će izvesti skok na jednu od lokacija iz tabele JP_TAB, zavisno od stanja akumulatora na ulazu:

```
                MOV  B,3            ;  priprema za  množenje
                MUL  AB             ; A = A*3, jer je dužina LJMP instrukcije 3 bajta
                MOV DPTR,#JP_TAB
                JMP  @A+DPTR        ; skok na željeno mesto
JP_TAB:         LJMP SERVICE_A
                LJMP SERVICE_B
                LJMP SERVICE_C
                LJMP SERVICE_D
```

Svaka instrukcija u tabeli mora da zauzme po 3 bajta, a posebnu pažnju treba obratiti da akumulator na ulazu nema vrednost veću od broja članova tabele umanjenog za 1 (u ovom primeru A mora da bude u opsegu od 0 do 3, jer ima 4 instrukcije skoka).

4.5 Uslovni skokovi

MNEMONIC	OPERATION	ADRESSING MODES				EXECUTION TIME (μs)
		DIR	IND	REG	IMM	
JZ rel	Jump if A = 0					2
JNZ rel	Jump if A ≠ 0					2
DJNZ <byte>,rel	Decrement and jump if not zero	X		X		2
CJNE A, <byte>,rel	Jump if A ≠ <byte>	X			X	2
CJNE <byte>,#data,rel	Jump if <byte> ≠ #data		X	X		2

Svi uslovni skokovi su mogući samo u opsegu -128 do +127 u odnosu na sledeću instrukciju u programu, kao što je slučaj kod SJMP. Ako je potrebno skočiti na neku dalju tačku, treba blizu uslovnog skoka postaviti repetitor, koji se sastoji od instrukcije AJMP ili LJMP, i onda izvesti kratki uslovni skok na repetitor. Druga mogućnost je da se komplementira uslov i preskoči repetitor, kao u ovom primeru, koji zamenjuje skok JZ FAR_LAB:

instrukcija

```
        JZ      FAR_LAB
```

se zamenjuje sekvencom

```
        JNZ     LAB1
        LJMP    FAR_LAB
LAB1:
```

Bitovi koji mogu da se iskoriste za uslovne skokove:

JZ (Jump on Zero) izvršava skok ako je akumulator jednak nuli
JNZ (Jump on Not Zero) izvršava skok ako je akumulator različit od nule
JC (Jump on Carry) izvršava skok ako je flag C setovan
JNC (Jump on Not Carry) izvršava skok ako je fleg C resetovan
JB (Jump on Bit) izvršava skok ako je navedeni bit setovan
JNB (jump on Not Bit) izvršava skok ako je navedeni bit resetovan
JBC (Jump on Bit and Clear bit) izvršava skok ako je navedeni bit setovan i istovremeno resetuje navedeni bit
CJNE (Compare and jump if Not Equal) poredi prvi operand sa drugim i skače ako su različiti. Bit C će biti setovan ako je prvi operand manji od drugog, u suprotnom biće resetovan.
DJNZ (Decrement and jump if Not Zero) umanjuje operand za 1, a ako je posle umanjenja operand različit od nule, skače na navedenu adresu, dok u suprotnom ignoriše skok i nastavlja izvršenje programa od sledeće instrukcije (slično instrukciji LOOP kod mikroprocesora 8086).

4.6 Uticaj instrukcija na flegove

U toku izrade programa od velike je važnosti znati na koji način instrukcije utiču na flegove u PSW registru. Neke instrukcije uopšte ne utiču na flegove C (carry), OV (overflow) i AC (auxiliary carry), pa te instrukcije nisu pomenute u tabeli.

	C	OV	AC
ADD	X	X	X
ADDC	X	X	X
SUBB	X	X	X
MUL	0	X	-
DIV	0	X	-
DA	X	-	-
RRC	X	-	-
RLC	X	-	-
SETB C	1	-	-
CLR C	0	-	-
CPL C	X	-	-
ANL C,bit	X	-	-
ANL C,/bit	X	-	-
ORL C,bit	X	-	-
ORL C,/bit	X	-	-
MOV C,bit	X	-	-
CJNE	X	-	-

Legenda:

- instrukcija ne utiče na fleg
0 instrukcija resetuje fleg
1 instrukcija setuje fleg
X fleg se menja zavisno od rezultata operacije

Napomena: instrukcije koje direktno menjaju sadržaj registra PSW će uticati na flegove, bez obzira što ovde nisu navedene.

Preostala dva flega: P (parnost) i Z (zero) nisu fizički implementirani u PSW registru kao flip-flopovi, nego su zapravo grupe logičkih kola koje testiraju stanje akumulatora. Tako će izlaz iz logike za fleg P biti visok ako je broj setovanih bita u akumulatoru paran, a za fleg Z ako su svi biti u akumulatoru jednaki nuli. Prema tome, nema nikakvog smisla navoditi kako instrukcije utiču na ove flegove, jer to zavisi samo od trenutnog stanja akumulatora.

5. LISTA INSTRUKCIJA (SUMARNO)

Na sledećim stranama dat je pregled svih instrukcija za mikrokontroler 8051 sa mnemonikom, kratkim opisom operacije, brojem bajtova koje instrukcija zauzima u programskoj memoriji i vremenom koje je potrebno za izvršenje instrukcije, datom u broju taktova oscilatora.

Skraćenice korišćene u setu instrukcija:

- Rn - Registar R0-R7
- direct - 8-bitna adresa lokacije na kojoj se nalazi podatak. Ovo može da bude u internom RAM-u (0-127) ili neki od specijalnih registara (na primer neki port, upravljački registar, statusni registar, itd)
- @Ri - lokacija u internom RAM-u (0-127) adresirana 8-bitnim registrom R0 ili Rl
- #data - 8-bitna konstanta neposredno pridružena u instrukciji
- #data16 - 16-bitna konstanta neposredno pridružena u instrukciji

- addrl6 - 16-bitna adresa, koja sledi iza instrukcija LJMP ili LCALL
- addr11 - 11-bitna adresa, koja sledi iza instrukcija AJMP ili ACALL. Skok koji izvode ove instrukcije je ograničen samo na zaokruženi blok od 2KB u kome se nalazi prvi bajt sledeće instrukcije posle AJMP ili ACALL.
- rel - 8-bitni ofset bajt za adresiranje lokacije koja se nalazi u opsegu -128 do +127 u odnosu na adresu prvog bajta sledeće instrukcije.
- bit - direktno adresirani bit u internom RAM-u (u opsegu 0 do 127, tj na adresama interne memorije u opsegu 20h-2Fh) ili u specijalnom registru (koji dozvoljava adresiranje pojedinačnih bita).

5.1 Aritmetičke operacije

Mnemonik		Opis operacije	Br. bajtova	Br. taktova
ADD	A,Rn	Saberi akumulator sa registrom	1	12
ADD	A,direct	Saberi akumulator sa direktnim RAM-om	2	12
ADD	A,@Ri	Saberi akumulator sa indirektnim RAM-om	1	12
ADD	A,#data	Saberi akumulator sa priloženom vrednošću	2	12
ADDC	A,Rn	Saberi akumulator sa registrom i C flegom	1	12
ADDC	A,direct	Saberi akumulator sa direktnim RAM-om i C flegom	2	12
ADDC	A,@Ri	Saberi akumulator sa ind. RAM-om i C flegom	1	12
ADDC	A,#data	Saberi akumulator sa datom vrednošću i C flegom	2	12
SUBB	A,Rn	Oduzmi registar i C fleg od akumulatora	1	12
SUBB	A,direct	Oduzmi direktan RAM i C fleg od akumulatora	2	12
SUBB	A,@Ri	Oduzmi indirektan RAM i C fleg od akumulatora	1	12
SUBB	A,#data	Oduzmi priloženu vrednost i C fleg od akumulatora	2	12
INC	A	Uvećaj akumulator za 1	1	12
INC	Rn	Uvećaj registar za 1	1	12
INC	direct	Uvećaj direktan RAM za 1	2	12
INC	@Ri	Uvećaj indirektan RAM za 1	1	12
DEC	A	Umanji akumulator za 1	1	12
DEC	Rn	Umanji registar za 1	1	12
DEC	direct	Umanji direktan RAM za 1	2	12
DEC	@Ri	Umanji indirektan RAM za 1	1	12
INC	DPTR	Uvećaj Data Pointer za 1	1	24
MUL	AB	Pomnoži A i B	1	48
DIV	AB	Podeli A sa B	1	48
DA	A	Podesi decimalno akumulator	1	12

5.2 Logičke operacije

Mnemonik		Opis operacije	Br. bajtova	Br. taktova
ANL	A,Rn	AND akumulator sa registrom	1	12
ANL	A,direct	AND akumulator sa direktnim RAM-om	2	12
ANL	A,@Ri	AND akumulator sa indirektnim RAM-om	1	12
ANL	A, #data	AND akumulator sa priloženom vrednšću	2	12
ANL	direct,A	AND akumulator sa direktnim RAM-om	2	12
ANL	direct,#data	AND direktni RAM sa priloženom vrednšću	3	24
ORL	A,Rn	OR akumulator sa registrom	1	12
ORL	A,direct	OR akumulator sa direktnim RAM-om	2	12
ORL	A,@Ri	OR akumulator sa indirektnim RAM-om	1	12
ORL	A,#data	OR akumulator sa priloženom vrednošću	2	12
ORL	direct,A	OR direktni RAM sa akumulatorom	2	12
ORL	direct,#data	OR direktni RAM sa priloženom vrednošću	3	24
XRL	A,Rn	EX-OR akumulator sa registrom	1	12
XRL	A,direct	EX-OR akumulator sa direktnim RAM-om akumulatorom	2	12

XRL	A,@Ri	EX-OR akumulator sa indirektnim RAM-om	1	12
XRL	A,#data	EX-OR akumulator sa priloženom vrednošću	2	12
XRL	direct,A	EX-OR direktni RAM sa akumulatorom	2	12
XRL	direct,#data	EX-OR direktni RAM sa priloženom vrednošću	3	24
CLR	A	Poništi stanje akumulatora	1	12
CPL	A	Komplementiraj sve bitove u akumulatoru	1	12
RL	A	Rotiraj nalevo akumulator	1	12
RLC	A	Rotiraj nalevo akumulator kroz C fleg	1	12
RR	A	Rotiraj nadesno akumulator	1	12
RRC	A	Rotiraj nadesno akumulator kroz C fleg	1	12
SWAP	A	Zameni gornja i donja 4 bita akumulatora	1	12

5.3 Prenos podataka

Mnemonik		Opis operacije	Br. bajtova	Br. taktova
MOV	A,Rn	Upiši u akumulator registar	1	12
MOV	A,direct	Upiši u akumulator direktan RAM	2	12
MOV	A,@Ri	Upiši u akumulator indirektan RAM	1	12
MOV	A,#data	Upiši u akumulator priloženu vrednost	2	12
MOV	Rn,A	Upiši u registar akumulator	1	12
MOV	Rn,direct	Upiši u registar direktan RAM	2	24
MOV	Rn,#data	Upiši u registar priloženu vrednost	2	12
MOV	direct,A	Upiši u direktan RAM akumulator	2	12
MOV	direct,Rn	Upiši u direktan RAM registar	2	24
MOV	direct,direct	Upiši direktan RAM u direktan RAM	3	24
MOV	direct,@Ri	Upiši u direktan RAM indirektan RAM	2	24
MOV	direct,#data	Upiši u direktan RAM priloženu vrednost	3	24
MOV	@Ri,A	Upiši u indirektan RAM akumulator	1	12
MOV	@Ri,direct	Upiši u indirekni RAM direktni RAM	2	24
MOV	@Ri,#data	Upiši u indirektni RAM priloženu vrednost	2	12
MOV	DPTR,#data16	Upiši u Data Pointer priloženu 16-bitnu vrednost	3	24
MOVC	A,@A+DPTR	Upiši u akumulator Indirektni Code Byte @A+DPTR	1	24
MOVC	A,@A+ PC	Upiši u akumulator indirektni Code Byte @A+PC	1	24
MOVX	A,@Ri	Upiši u akumulator indirektni Ext RAM @Ri	1	24
MOVX	A,@DPTR	Upiši u akumulator indirektni Ext RAM @DPTR	1	24
MOVX	@Ri, A	Upiši u indirektni Ext RAM @Ri akumulator	1	24
MOVX	@DPTR,A	Upiši u indirektni Ext RAM @DPTR akumulator	1	24
PUSH	direct	Stavi direktni RAM na stek	2	24
POP	direct	Uzmi direktni RAM sa steka	2	24
XCH	A,Rn	Izmeni vrednosti registra i akumulatora	1	12
XCH	A,direct	Izmeni vrednosti direktnog RAM-a i akumulatora	2	12
XCH	A,@Ri	Izmeni vrednosti indirektnog RAM-a i akumulatora	1	12
XCHD	A,@Ri	Izmeni vrednosti donje cifre ind. RAM-a i	1	12

5.4 Bulove operacije

Mnemonik		Opis operacije	Br. bajtova	Br. taktova
CLR	C	Resetuj C fleg	1	12
CLR	bit	Resetuj direktan bit	2	12
SETB	C	Setuj C fleg	1	12
SETB	bit	Setuj direktan bit	2	12
CPL	C	Komplementiraj C fleg	1	12
CPL	bit	Komplementiraj direktan bit	2	12
ANL	C,bit	AND C fleg sa direktnim bitom	2	24
ANL	C,/bit	AND C fleg sa komplementiranim direktnim bitom	2	24
OR	C,bit	OR C fleg sa direktnim bitom	2	24
ORL	C,/bit	OR C fleg sa komplementaranim direktnim bitom	2	24
MOV	C,bit	Upiši u C fleg direktan bit	2	12
MOV	bit,C	Upiši u direktan bit C fleg	2	24
JC	rel	Skoči ako je C fleg setovan	2	24

JNC	rel	Skoči ako C fleg nije setovan	2	24
JB	bit,rel	Skoči ako je direktan bit setovan	3	24
JNB	bit,rel	Skoči ako direktan bit nije setovan	3	24
JBC	bit,rel	Skoči ako je direktan bit setovan i resetuj bit	3	24

5.5 Potprogrami i skokovi

Mnemonik		Opis operacije	Br. bajtova	Br. taktova
ACALL	addr 11	Poziv potprograma na apsolutnoj (11-bitnoj) adresi unutar tekućeg bloka od 2KB	2	24
LCALL	addr 16	Poziv potprograma na apsolutnoj 16-bitnoj adresi	3	24
RET		Povratak iz potprograma	1	24
RETI		Povratak iz prekidne rutine	1	24
AJMP	addr 11	Skok na apsolutnu (11-bitnu) adresu unutar tekućeg bloka od 2KB	2	24
LJMP	addr 16	Skok na apsolutnu 16-bitnu adresu	3	24
SJMP	rel	Kratki skok (8-bitna relativna adresa)	2	24
JMP	@A+DPTR	Indirektni skok na adresu iz Code Mem @A+DPTR	1	24
JZ	rel	Skoči ako je akumulator jednak nuli	2	24
JNZ	rel	Skoči ako je akumulator različit od nule	2	24
CJNE	A,direct,rel	Uporedi A sa direktnim RAM-om i skoči ako su različiti	3	24
CJNE	A,#data,rel	Uporedi A sa pridr. podatkom i skoči ako su različiti	3	24
CJNE	Rn,#data,rel	Uporedi Rn sa pridr. podatkom i skoči ako su različiti	3	24
CJNE	@Ri,#data,rel	Uporedi ind.RAM sa podatkom i skoči ako su različiti	3	24
DJNZ	Rn,rel	Umanji registar i skoči ako nije jednak nuli	3	24
DJNZ	direct,rel	Umanji direktni RAM i skoči ako nije jednak nuli	3	24
NOP		No operation - bez operacije	1	12

6. ASEMBLER A51 I UPOZNAVANJE SA SOFTVERSKIM PAKETOM µVision2

Direktive koje će ovde biti date odnose se na Franklin asembler A51, za druge asemblere oznake mogu da se razlikuju, pa o tome treba voditi računa.

6.1 Direktive

EQU dodeljuje numeričku vrednost navedenom simbolu.
Primer:

```
temperatura  EQU  20
```

Numerička vrednost koja je ovom direktivom dodeljena simbolu temperatura ne može se redefinisati u programu. Ovo se koristi u programu da bi kod bio čitljiviji, jer kada se u programu naiđe na simbole *temperatura* ili *brzina*, zna se da taj simbol označava vrednost temperature ili brzine. Ako bi bile napisane samo brojčane vrednosti, program bi bio znatno nerazumljiviji.

SET radi isto kao i EQU, ali se može redefinisati neograničen broj puta tokom izvršavanja programa.
Primer:

```
Brzina  SET  25
…
Brzina  SET  100
…
```

```
Brzina  SET  Brzina+10
```

BIT dodeljuje adresu bita navedenom simbolu.
Primer:

```
RELE        BIT  P1.0
LED         BIT  P3.4
RAMWR       BIT  WR
IZLAZ       BIT  P2.0
```

ORG definiše tekuću vrednost brojača lokacija.
Primer:

```
ORG  2000
```

Sada će adresa sledeće instrukcije ili komande DB (ili DW) imati vrednost 2000.

DS rezerviše prostor u memoriji izražen u bajtovima.
Primer:

```
        ORG  100        ; počni od adrese 100
        DS  7           ; rezerviši 7 bajtova
NovaLok:                ; nastavi program
```

Ovo znači da će labela NovaLok imati vrednost 107, i na nju se može skočiti ako želimo da nastavimo rad na adresi 107.

DB upisuje navedenu vrednost bajta u programsku memoriju. Ako se navodi više vrednosti, one se odvajaju zarezima. Ako se navodi ASCII niz, stavlja se pod jednostruke navodnike.
Primer:

```
DB  22,33,'Alarm',0
```

DW je isto kao DB, ali se upisuje 16-bitna (dvo-bajtna) vrednost. Za razliku od nekih drugih procesora (na primer 8086), prvo se upisuje visoki, pa niski bajt.

CSEG označava da se naredni segment odnosi na programsku memoriju.

XSEG označava da se naredni segment odnosi na spoljni RAM.

DSEG označava da se naredni segment odnosi na interni RAM.

ISEG označava da se naredni segment odnosi na gornji deo internog RAM-a, koji se može adresirati isključivo indirektno pomoću registara R0 i R1 (lokacije 80h-FFh, interna memorija koja nije implementirana u mikrokontroleru 8051).

6.2 Keil-ov asembler A51 sa integrisanim okruženjem µVision2

Asembler koji će ovde biti razmatran je Keil-ov A51 sa integrisanim okruženjem **µVision2** koje sadrži: editor izvornog koda, sistem za podešavanje parametara i poziv asemblera, kao i sistem za testiranje i simulaciju prevedenog programa. **µVision2** se koristi za kreiranje izvornog fajla (koda) i njegovo povezivanje unutar projekta, koji definiše željenu aplikaciju. **µVision2** automatski asemblira i povezuje sve programe u projektu. On podržava: C51 compiler, A51 macro assembler, BL51 linker/locator, LIB51 library manager, OH51 object-hex converter i RTX-51 real-time operating system.

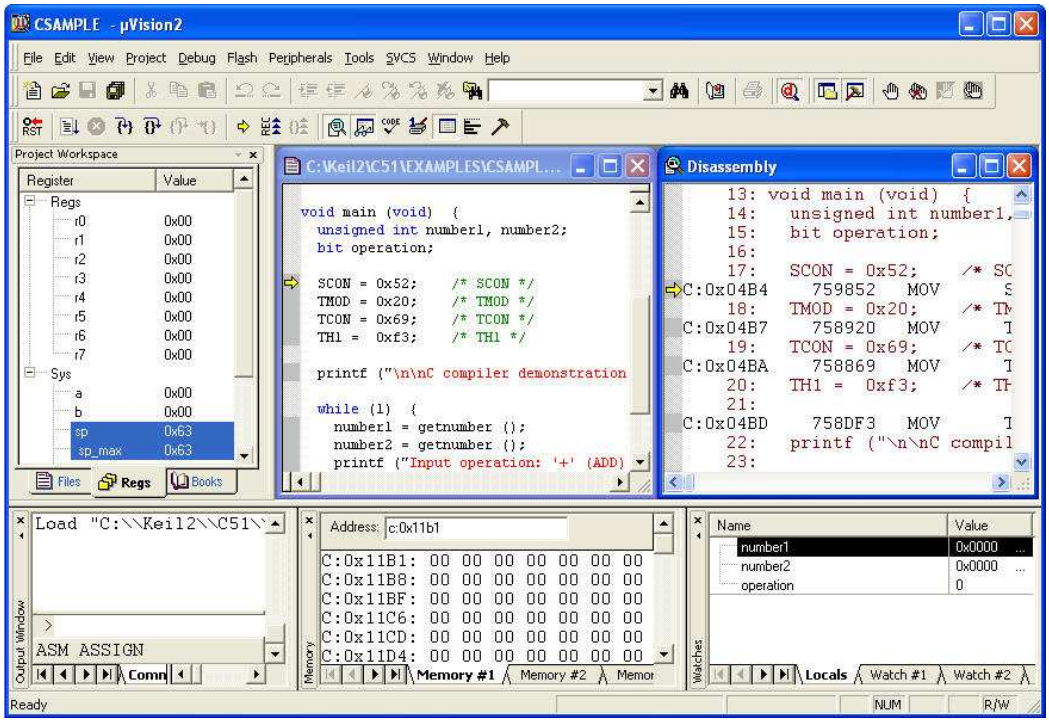
Postupak kreiranja softverske aplikacije:

- Startuje se **µVision2,** definiše se projekat i odabere se mikrokontroler sa ponuđene liste.
- Napiše se **izvorni (source) fajl** u editoru i doda se u projekat.
- Izvrši se podešavanje (unos) parametara hardverske strukture izabranog mikrokontrolera.
- **Izvorni fajl** prolazi **A51 macro assembler.** Proces asembliranja stvara objektni fajl.

- **Objektni fajlovi** se korišćenjem LIB51 library managera formiraju u objektne biblioteke.
- **Objektni fajlovi i objektne biblioteke** se povezuju sa BL51 Linker/Locator-om u apsolutni objektni modul. Ceo kod u apsolutnom objektnom fajlu zauzima fiksne memorijske lokacije.
- Zatim se izvrši komanda **build project,** pri čemu se formira HEX fajl ako nema grešaka u programu.
- Ako postoje **greške**, one se se prikazuju u donjem delu ekrana, uz kratak komentar o kakvoj je greški reč.
- Kada su ispravljene sve formalne greške, može se koristiti **µVision2 debugger** za simulaciju rada 8051 sistema. On prikazuje: memorijsku mapu promenljivih, lokalne promenljive i periferije.
- Na kraju se može programirati dati čip, direktnim korišćenjem **HEX** fajla, a po potrebi se **HEX** fajl pre programiranja prvo konvertuje u binarni (**BIN)** fajl.

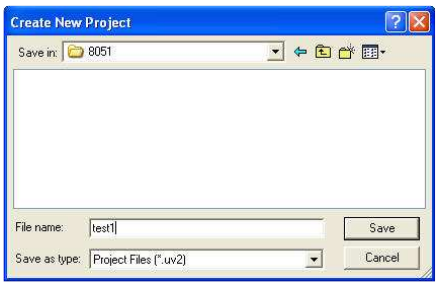
6.3 Startovanje µVision2 programa i kreiranje projekta

Program se startuje pozivom µVision2. Izgled menija i pojedinih ‘prozora' prikazan je na sledećoj slici:



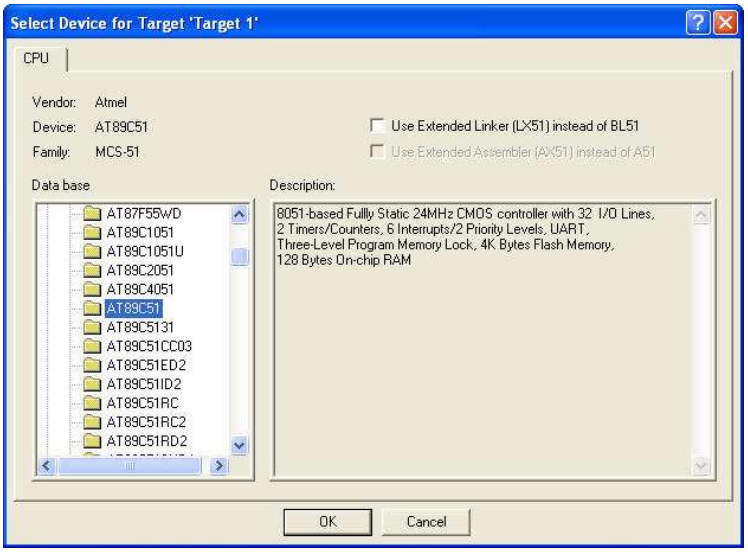
Slika 6.1 Izgled programa µVision2

Kreiranje novog projekta se vrši izborom opcije **Project-New Project,** posle čega se odabere mesto na hard disku gde će biti smešten projekat sa ekstenzijom .uv2 (preporučuje se prethodno kreiranje odgovarajućeg foldera na hard disku). Na primer, ako se projekat zove PROJEKAT1.UV2:



Slika 6.2 Prozor za kreiranje novog projekta

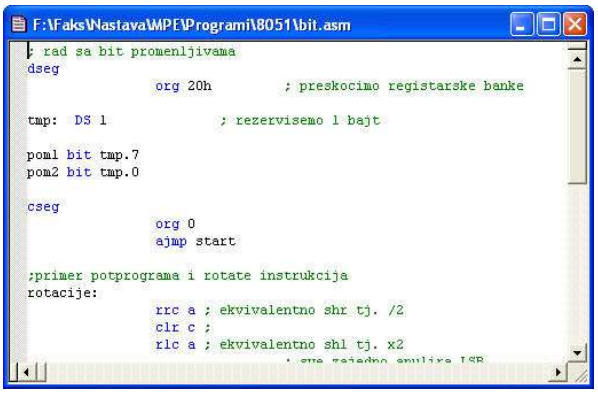
Odmah posle toga program zahteva izbor mikrokontrolera (CPU), kao što je prikazano na sledećoj slici gde je odabran Atmelov 89C51:



Slika 6.3 Odabiranje mikrokontrolera sa kojim će se raditi

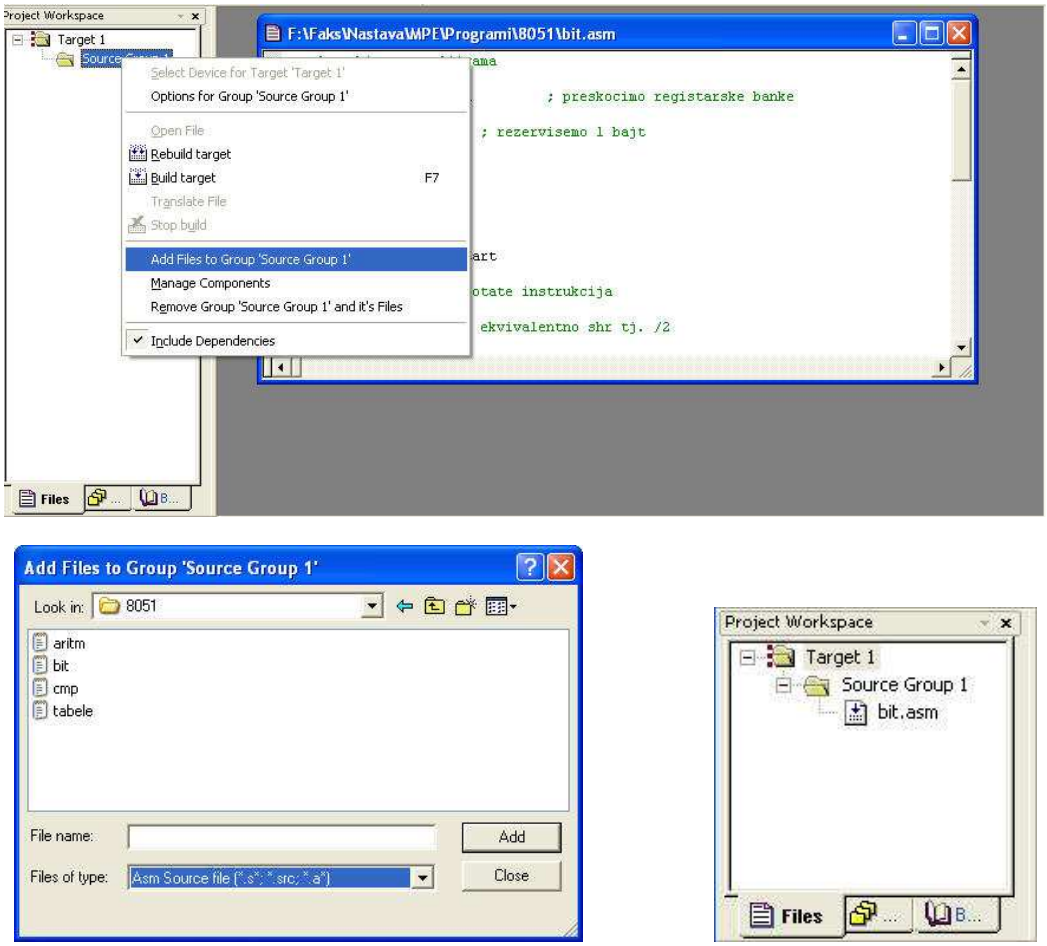
6.4 Kreiranje izvornog (source) fajla i dodavanje fajla u projekat

Kreiranje novog izvornog fajla se vrši izborom opcije **File-New,** pri čemu se otvara prazan editorski prozor u koji se unosi izvorni kod u **asembleru**. Zatim se izabere opcija **File-Save As** i zada se ime fajla sa ekstenzijom **.asm**, npr. bit.asm:



Slika 6.4 Kreiranje fajla sa ekstenzijom .asm

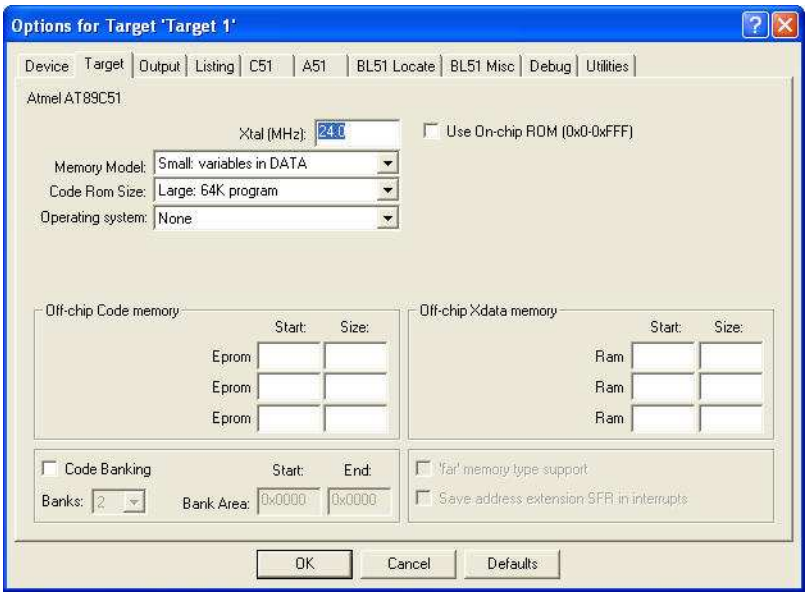
Sada treba taj fajl dodati u projekat. To se može uraditi tako što se prvo otvori **Project Window** (**View -Project Window**), ako već nije otvoren. Zatim se klikne desnim tasterom miša na **Source Group1,** a potom izabere **Add Files to Group** '**Source Group1'**:



Slika 6.5 Dodavanje izvornog fajla u projekat

6.5 Unos parametara hardverske strukture mikrokontrolera

Unos parametara strukture mikrokontrolera se vrši izborom opcije **Project-Options for target 'Target1':**

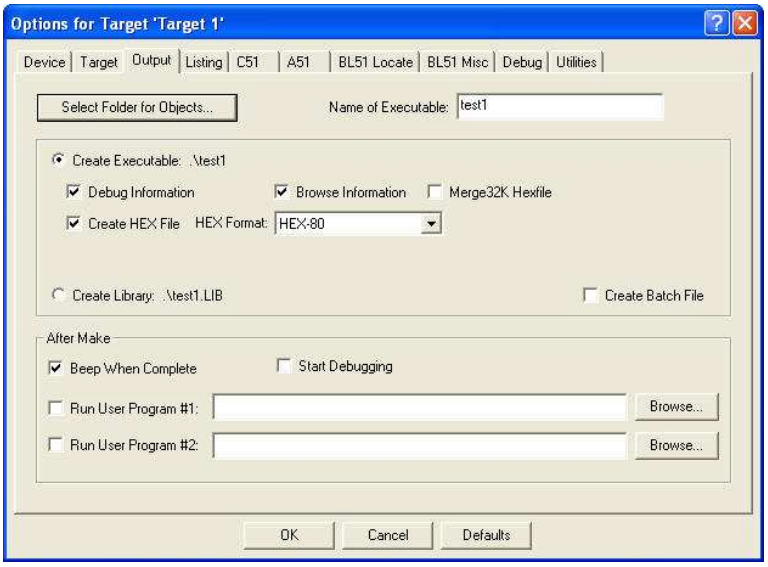


Slika 6.6 Prozor za unos parametara strukture mikrokontrolera

Xtal(MHz) označava frekvenciju rada mikrokontrolera (bitno samo za simulator).
Memorijski model određuje koliko će memorije biti korišćeno za promenljive i podatke. O ovome će biti više reči kada budemo govorili o Keil-ovom C kompajleru C51.

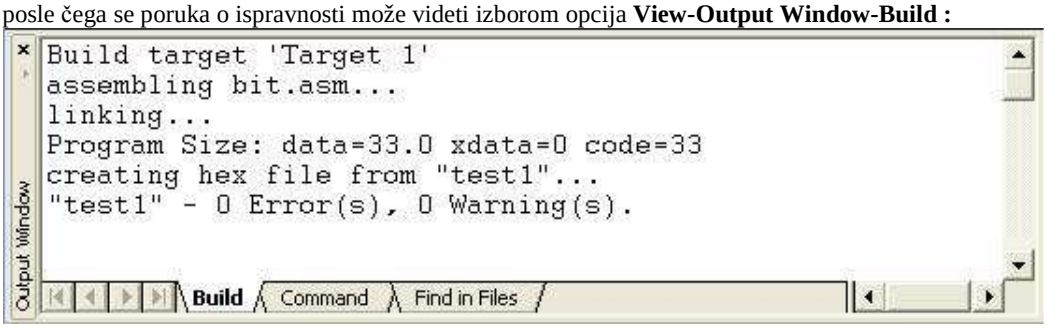
6.6 Testiranje projekta i kreiranje hex fajla

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



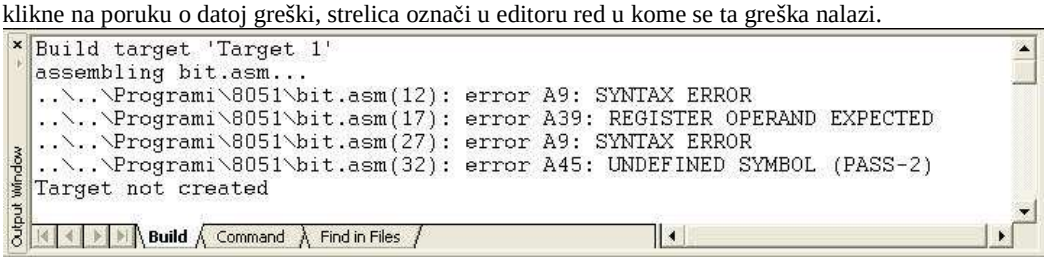
Slika 6.6 Kreiranje odgovarajućeg HEX fajla

Zatim se vrši provera ispravnosti unešenog programa u projektu izborom opcije **Project–Build Target,** posle čega se poruka o ispravnosti može videti izborom opcija **View-Output Window-Build :**



Slika 6.7 Izgled prozora "Output window" kada program nema grešaka

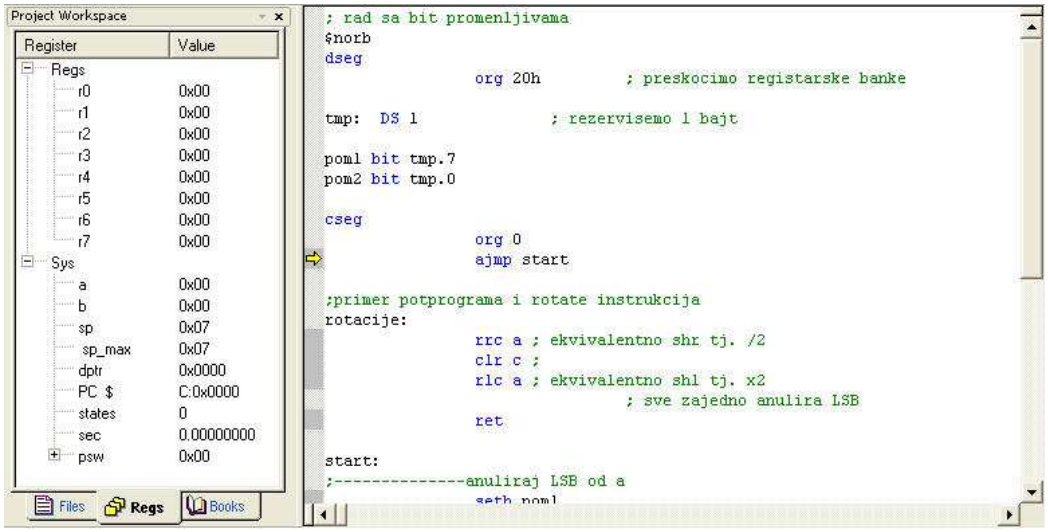
Na gornjoj slici je prikazan program bez grešaka, dok na donjoj postoje 4 greške. Kada se dva puta klikne na poruku o datoj greški, strelica označi u editoru red u kome se ta greška nalazi.



Slika 6.8 Izgled prozora "Output window" kada program ima greške

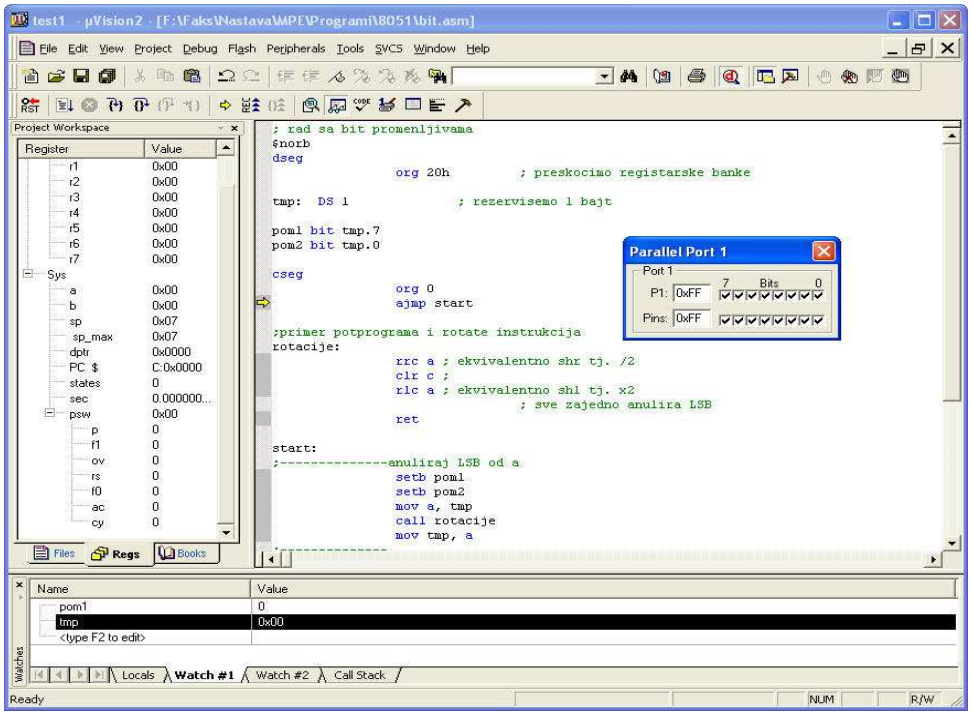
6.7 Simulacija rada mikrokontrolera 8051

Tek kada su ispravljene sve greške u programu može se pristupiti simulaciji rada mikrokontrolera. Postoje mnogi programi za simulaciju, a ovde će u kratkim crtama biti opisane mogućnosti **µVision2 debugger**-a**.** On simulira memorijsku mapu promenljivih, lokalne promenljive i periferije (serijski port, spoljašnje I/O nožice i tajmere). Startovanje se vrši izborom opcije **Debug-Start/Stop Debug Session**:



Slika 6.9 Izgled debugger-a

Sa leve strane se vide registri u koje su smeštene promenljive, a takođe se može videti i sadržaj **PSW** registra. Pritiskom na taster **F11** izvršavaju se instrukcije jedna po jedna, pri čemu žuta strelica prati redosled izvršavanja programa. Istovremeno, sadržaj registara se menja zavisno od izvršene instrukcije. Mogu se posmatrati i lokalne promenljive (ovde je to npr. pom1 ili tmp) uključenjem opcije **View-Watch&Call Stack Window**, kao i stanje na portovima izborom opcije **Peripherals-I/0-Ports** (ovde se posmatra samo port P1):



Slika 6.10 Izgled debugger-a sa uključenim "Watch" i "Port 1" opcijama

Prekidanje simulacije se vrši izborom opcije **Debug-Stop Running (Debug-Start/Stop Debug Session).** Ovaj vid simulacije je pogodan za proveru redosleda izvršavanja programa, matematičkih izračunavanja i osnovnog prikaza periferija.

6.8 Jednostavniji primeri programa napisanih u asembleru

1) Program u kome je prikazano korišćenje osnovnih aritmetičkih operacija

```
; osnovne aritmeticke instrukcije
dseg
       org 20h      ; preskocimo registarske banke

rez:  DS 1          ; rezervisemo 1 bajt

jedinica equ 1h ;definisemo konstantu

cseg
       org 0
       ajmp start

start:
       mov a, #0feh
       add a, #jedinica
       add a, #1    ; a <- 0 ; c <-1
       addc   a, 0   ; a <- 1
       setb c       ; c <- 1
       subb a, #1   ; a <- ff ; c <-1
       inc a
       mov r7, #5
       mov a, r7
       mov b, #3
       mul ab       ; a <- 5*3 = f
       mov b, #2
       div ab       ; a <- div(f/2) = 7 ; b <- mod(f/2) = 1
       mov rez, a
       sjmp start   ; kod mikrokontrolera uvek imamo glavnu petlju
end
```

2) Program u kome je prikazano korišćenje instrukcija za rad sa bitima

```
; rad sa bit promenljivama
dseg
       org 20h      ; preskocimo registarske banke

tmp:  DS 1          ; rezervisemo 1 bajt

pom1 bit tmp.7
pom2 bit tmp.0

cseg
       org 0
       ajmp start

;primer potprograma i rotate instrukcija
rotacije:
       rrc a ; ekvivalentno shr tj. /2
```

```
        clr c ;
        rlc a ; ekvivalentno shl tj. x2
                        ; sve zajedno anulira LSB
        ret

start:
;---------------------------------- anuliraj LSB od a
        setb pom1
        setb pom2
        mov a, tmp
        call rotacije
        mov tmp, a
;-------------------------------------------------------------------------
        mov a, #81h   ; ova i sledece takodje anuliraju LSB
        anl A,#0FEh
;-------------------------------------------------------------------------
        mov c, pom1 ; kako koristiti jnc/jc/jnb/jb "bit jumpove"
        jnc one
        clr c
one:  setb pom1
;-------------------------------------------------------------------------
        ; kako su se prethodne cetiri mogle krace napisati?
        jnb pom1, one
;-------------------------------------------------------------------------

        jmp start     ; kod mikrokontrolera uvek imamo glavnu petlju
end
```

3) Program u kome su prikazana razna poređenja i poređenja sa skokovima

```
;primeri poredjenja i poredjenja sa skokovima
dseg
        org 20h       ; preskocimo registarske banke

cx:  DS 1             ; rezervisemo 1 bajt

prvi equ 20   ;prvi broj

cseg
        org 0
        ajmp start

start:
;--------------primer djnz instrukcije
        mov cx, #10
petlja:
        ; ovde nesto radi
        djnz cx, petlja ; kao loop kod x86 ili (if !(--cx)) u c-u

;--------------primer cjne instrukcije

        mov a, #19
petlja2:
        cjne A,#prvi, nisu_jednaki
        ;jednaki su pa zavrsi
        sjmp dalje
nisu_jednaki:
        inc a
        sjmp petlja2
;--------------ostala poredjenja
dalje:
        ; idu preko oduzimanja
        ; pa se gleda carry i/ili zero jz/jnz/jc/jnc
        ; npr. a > r1 ?

        mov a, #9
        mov r1,#11
```

```
              clr c ;jer nema sub instrukcije pa se mora obrisati c !
              subb a,r1
              jc a_je_manji
              jz jednaki_su
              ;r1 je veci od a
       a_je_manji:
              ;...
              sjmp dalje
       jednaki_su:
              ;...
              sjmp dalje

       ;-----------------------------------------------------------------

              jmp start    ; kod mikrokontrolera uvek imamo glavnu petlju
       end
```

4) Program u kome je prikazan rad sa tabelama

```
       ;ilustrovanje rada sa tabelama
       DSEG
              org 20h
              tmp: ds 1
              count:ds 1

       CSEG
              org 0
              sjmp start

       neka_fja:
              inc a                      ;da bi preskocili bajt koji je od
ret
              movc a,@a+pc
              ret
              db 27h,41h,0,5eh,7fh

       start:
       ;--------------------------------------------------------------------

              mov a, #0 ;
              mov dptr, #M
              movc A,@A+dptr      ; uzmi 'M1'
              mov R0, a

              ;...

       ;--------------------------------------------------------------------

              mov a, #3
              call neka_fja ;posle ovoga ce u a biti 5eh
       ;--------------------------------------------------------------------

       sjmp start

       ; tabele se uvek stavljaju na kraju koda
       ; ili van 'aktivnog' koda da od nje ne bi
       ; postale instrukcije

       M:     db 11101110b, 11100010b, 10101000b, 00001100b

       END
```