SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**
Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave

Upute za 2. laboratorijsku vježbu iz predmeta

# ARHITEKTURA RAČUNALA 2

Mikroprogramiranje

Zagreb, 2008.

# Uvod

Tema vježbe je *mirkoprogramiranje*. Cilj je upoznavanje s elementarnim operacijama od kojih se sastoje strojne instrukcije i mogućnostima oblikovanja vlastitih strojnih instrukcija oblikovanjem odgovarajucih mikroprograma.

Mikroprogramiranje je jedan on dvaju temeljnih načina realizacije upravljačke jedinice procesora (drugi način je sklopovska izvedba). Svaka strojna instrukcija – koja se u kontekstu mikroprogramiranja naziva *makroinstrukcijom* – predstavljena je jednim *mikroprogramom*, odnosno nizom *mikroinstrukcija*. Ove mikroinstrukcije (ili *mikroriječi*) smještene su u posebnoj, tzv. *mikroprogramskoj memoriji* ili *mikromemoriji* koja se nalazi unutar same upravljačke jedinice, dakle unutar procesora. Mikroinstrukcija je, pak, n-bitna riječ koja kodirano predstavlja jednu ili veći broj *mikrooperacija*. Mikrooperacije su elementarne i nedjeljive operacije koje su izravno sklopovski podržane. Koncept mikroprogramiranja detaljnije je opisan u predavanjima i u knjizi:
S. Ribarić: "Naprednije arhitekture mikroprocesora", Školska knjiga, 1990, (poglavlju 6, "Upravljačka jedinica").

U ovoj vježbi, studenti će, koristeći mikroprogramski simulator, projektirati vlastiti jednostavni procesor s vlastitim skupom od nekoliko instrukcija.

# Zadaci

Potrebno je proučiti upute za korištenje mikroprogramskog sustava te napisati mikroprograme za dolje navedene strojne instrukcije (ne treba rješavati zadatke iz 6. poglavlja ovih uputa). Prije rješavanja zadataka preporuča se da studenti isprobaju i analiziraju primjere mikroprograma koji su dani u 5. poglavlju ovih uputa.

**MOVE A,B** - kopiranje sadržaja registra A u registar B
**SUB A,B** - oduzimanje sadržaja dvaju registara, rezultat se pohranjuje u B
**AND A,B** - logičko I između registara A i B, rezultat se pokranjuje u B
**SHL A** - posmak sadržaja registra A u lijevo; bit koji "ispada" odlazi u zastavicu carry
**ASR** - aritmetički posmak (čuva se bit predznaka) sadržaja registra A u desno
**JPZ #** - uvjetni skok na zadanu adresu, ako je zastavica Z postavljena
**PUSH A** - pohranjivanje sadržaja registra A na stog
**POP A** - skidanje podatka sa stoga i njegovo spremanje u registar A
**CALL #** - poziv potprograma
**RET** - povratak iz potprograma

Kao rješenje potrebno je modelirati i testirati zadani skup instrukcija na simulatoru te ih kratko dokumentirati. Svaku ostvarenu (makro)instrukciju kratko opisati, navesti njen operacijski kod, te prikazati odgovarajući sadržaj mikroprogramske memorije (adresa, sljedeća adresa, upravljanje zastavicama, mnemonik funkcije, registar, pristup memoriji, opis mikroinstrukcije). U dokumentaciji priložiti ispitne makroprograme i primjere rezultata izvršavanja.

# Kazalo

# 1. Uvod

Tema ove laboratorijske vježbe je **mikroprogramiranje** mikroprogramskog sustava zasnovanog oko mikroprocesora na osnovi bit odreska. Korištenjem raspoloživog mikroprogramskog razvojnog sustava MP3000, dodatnog međusklopa i odgovarajuće programske podrške moguće je pisati mikro i makro programe, te izvoditi ih korak po korak uz praćenje stanja takvog mikroprocesora (njegovih registara, zastavica i sabirnica).

Važno je napomenuti da se ovdje pod pojmom *makronaredba* podrazumijeva uobičajena strojna, zbirna, odnosno asemblerska naredba. Time se ističe viša razina programiranja od mikroprogramiranja.

## 1.1. Značajke mikroprocesora na osnovi bit-odreska

Osnovna značajka mikroprocesora na osnovi bit odreska je u tome što su funkcije upravljanja i obrade podataka izdvojene na različitim integriranim sklopovima. Sam bit odrezak predstavlja jedan integrirani sklop koji obrađuje ili upravlja dijelom bitova računalne riječi takvog računala. Izrađen je tako da se više istih može spojiti u kaskadu čime se dobija računalo proizvoljne bitovne širine. Tako postoje bit odresci od 2, 4, 8, a u novije vrijeme i od 16 bitova. Sa njima se može načiniti računalo sa širinom riječi koja je višekratnik osnovne bitovne širine.

Druga osnovna značajka mikroprocesora je mikroprogramabilnost, što znači da korisnik - projektant mikroprocesora na osnovi bit-odreska može *mikroprogramirati*, odnosno oblikovati skup strojnih naredbi s obzirom na buduću primjenu. Osnovnu ideju o takvom računalu dao je 1951. M. V. Wilkes. Mnogi današnji mikroprocesori su CISC arhitekture čija je osnova mikroprogramirani set strojnih naredbi, ali korisnik nema pristupa mikroprogramu i ne dopušta mu se vlastito mikroprogramiranje. To dozvoljavaju mikroprogramirana računala čiji je mikroprogram smješten u radnoj memoriji. Takav je ovaj sustav na kojem se rade ove laboratorijske vježbe.

Općenito, mikroračunalo na osnovi bit-odreska ima dvije logičke cjeline: **upravljačku jedinicu** i **procesorsku jedinicu.**

## 1.2. Upravljačka jedinica

Upravljačka jedinica obuhvaća mikroprogramsku memoriju, mikroprogramsku upravljačku jedinicu i pomoćne sklopove.

Mikroprogramska radna memorija sadrži *upravljački program* za izvođenje pojedinih strojnih naredbi, *makronaredbi*. Veličinu mikroprogramske riječi definira projektant mikroprogramskog sustava. Za određen broj procesnih elemenata i veličinu mikroprogramske memorije postoji minimalan broj bitova koji nužno mora imati mikroprogramska riječ, ali se često dodaje još nekoliko korisničkih bitova čime se dodaju razne mogućnosti (npr. proširenje adresnog prostora mikroprogramske memorije i upravljanje dodatnim sklopovljem). Organizacija mikroprogramske riječi specifična je za određeni izgrađeni mikroprogramski sustav.

Osnovni dio mikroprogramske upravljačke jedinice je *mikroprogramski sljednik* koji računa adresu slijedeće mikronaredbe, i to na osnovu operacijskog koda slijedeće makronaredbe ili skokom (uvjetnim ili bezuvjetnim) definiranim u odgovarajućem polju tekuće mikronaredbe.

Upravljačka jedinica može biti izvedena na jednom integriranom sklopu što donekle ograničava veličinu mikroprogramske memorije (tada je treba organizirati po stranicama) ili kao bit-odrezak što omogućava projektiranje proizvoljno velike mikroprogramske memorije. Prvi način je karakterističan za mikroprogramski sustav Intel 3000 koji se nalazi u raspoloživom *MP3000*, a drugi koristi mikroprogramski sustav Am 2900.

### 1.3. Procesorska jedinica

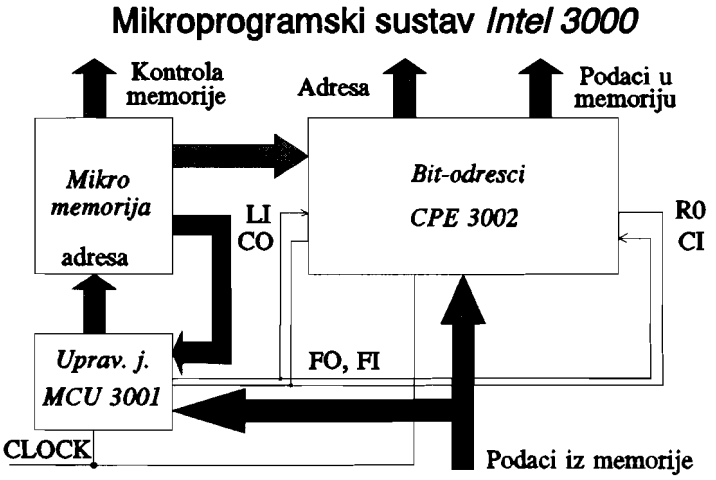
Procesorska jedinica (ili RALU) sadrži aritmetičko-logičku jedinicu, mikrofunkcijski dekoder i registre opće namjene. Izgrađuje se nizom procesnih elemenata spojenih paralelno na željenu veličinu riječi. Upravljačke linije vežu se paralelno na sve procesne elemente, jer svi procesni elementi istovremeno izvode istu funkciju nad određenim dijelom operanda. Linije za prijenos bitova kod aritmetičko-logičkih operacija se između procesnih elemenata spajaju serijski.

Procesni element veličine *n* bitova ima registre, aritmetičko-logičku jedinicu i sve sabirnice (podatkovne, adresne, ulazno/izlazne i dr.) širine *n* bitova, i sve operacije izvodi na svojih *n* bitova kojima pridonosi ukupnoj veličini makroriječi.

Detaljnije o mikroprogramiranju i sličnoj materiji može se naći u literaturi koja se koristi za predmet "*Arhitektura i organizacija digitalnih računala*".

## 2. Mikroprogramski sustav Intel 3000

Iako danas postoje bit-odresci bitno veće brzine rada i širine riječi, mikroprogramski sustav MP3000 izgrađen oko porodice Intel 3000 pruža zadovoljavajuću osnovu za upoznavanje sa ovakvom arhitekturom mikroračunala, jer je vrlo blizu općenitoj arhitekturi mikroprogramskog sustava na osnovi bit-odreska. Na slijedećoj slici prikazana je osnovna shema povezivanja porodice Intel 3000.




**Slika 1:** Blok shema sustava izgrađenog oko porodice Intel 3000

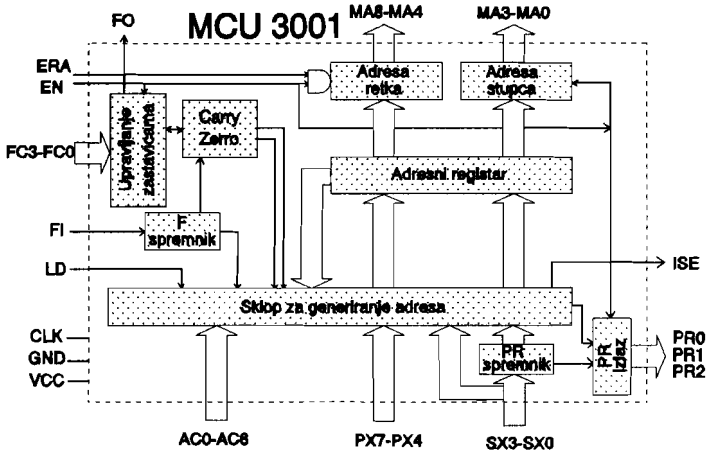
Porodica Intel 3000 sadrži više sklopova kojima se izgrađuje mikroprogramirljivi mikroprocesor/mikroračunalo na osnovi bit-odreska. Izrađeni su u TTL bipolarnoj tehnologiji. To su slijedeći sklopovi:

- mikroprogramska upravljačka jedinica (*Microprogram Control Unit*) MCU 3001;
- osnovni procesni element (*Central Processing Element*) CPE 3002;
- generator za predviđanje bita prijenosa (*Look-ahead Carry Generator*) LCG 3003;
- zaporni sklop opće namjene (*Multi-mode Latch Buffer*) MLB 3212;
- sklop za upravljanje prekidom (*Interrupt Control Unit*) ICU 3214;
- dvosmjerno paralelno sabirničko pojačalo (*Parallel Bidirectional Bus Driver*) PBBD 3216.

U mikroprogramskom sustavu MP3000 koriste se svi navedeni sklopovi osim LCG 3003. Taj je sklop

predviđen za 16-bitovnu konfiguraciju, a budući da je MP3000 izgrađen kao 8-bitna konfiguracija, umjesto njega koristi se 74182 Carry Look-Ahead Generator. Najvažniji sklopovi su MCU 3001 i CPE 3002, dok su ostali opće namjene i po funkciji nisu specifični za mikroprogramski sustav, te neće biti detaljnije obrađivani.

## 2.1. Mikroprogramska upravljačka jedinica MCU 3001




**Slika 2:** Mikroprogramska upravljačka jedinica MCU 3001

Slijedi kratki opis najvažnijih izvoda MCU 3001 (prikaz na slici 2):

**PX7-PX4** *Primarna sabirnica* - podatak na ovu sabirnicu dovodi se iz radne memorije. Uglavnom se koristi za dekodiranje makronaredbe i tada sadrži 4 značajnija bita operacijskog koda.

**SX3-SX0** *Sekundarna sabirnica* - podatak na ovu sabirnicu se također dovodi iz radne memorije, a prilikom dekodiranja sadrži 4 manje značajna bita operacijskog koda.

**FC3-FC0** *Upravljanje zastavicama* - ovi se bitovi dovode iz odgovarajućih bitova mikroprogramske riječi pročitane iz mikromemorije. Bitovi FC0 i FC1 određuju u koju će se zastavicu (C ili Z) upisati stanje ulaza FI, a FC2 i FC3 određuju koja će se zastavica pojaviti na izlazu FO.

**FI** *Ulaz zastavice* - vrijednost koja se upisuje u C ili Z zastavicu. Spaja se na izlaz prijenosa i pomaka procesnih elemenata.

**FO** *Izlaz zastavice* - vrijednosti iz C ili Z spremnika. Spaja se na ulaz prijenosa i pomaka procesnih elemenata.

**MA8-MA0** *Mikroprogramska adresa* - sabirnica na koju MCU postavlja adresu mikroriječi koju upravo treba pribaviti iz upravljačke memorije.

**AC6-AC0** *Upravljanje slijedećom mikroadresom* - na ove se bitove dovode bitovi mikroriječi koji određuju mikroadresu slijedeće mikroriječi koju treba pribaviti i izvršiti.

**LD** *Postavljanje mikroadrese* - ovaj se bit također dovodi iz mikroriječi, a kada je aktivan zabranjuje određivanje slijedeće mikroadrese preko linija AC6-AC0, već se slijedeća mikroadresa određuje na osnovu operacijskog koda slijedeće makronaredbe.

MCU 3001 određuje adresu slijedeće mikronaredbe koju treba dohvatiti iz mikroprogramske upravljačke memorije (zato se naziva i *mikroprogramskim sljednikom*). Dva su osnovna načina dobijanja te adrese:

- na osnovi operacijskog koda slijedeće makronaredbe, ili
- uvjetnim/bezuvjetnim skokom na određenu mikroprogramsku lokaciju.

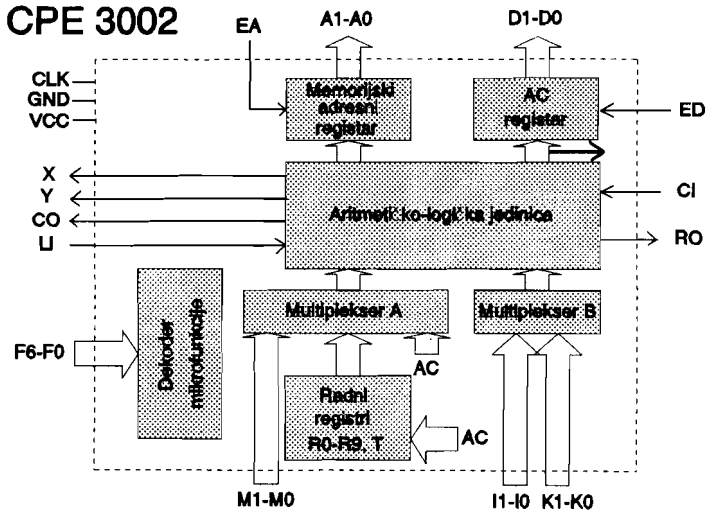
Koji će se način koristiti određuje posebni bit u mikronaredbi, odnosno linija LD na MCU 3001. Uz LD=1 operacijski kod makronaredbe preuzima se sa linija $SX_0$-$SX_3$ i $PX_4$-$PX_7$ (tada se $MA_8$ postavlja na 0), a uz LD=0 linije $AC_0$-$AC_6$ određuju se skok na slijedeću mikronaredbu. Postoji 11 vrsta skokova (4 bezuvjetna i 7 uvjetnih) na slijedeću mikroadresu, a njihov opis može se naći u prilogu ovih uputa.

U upravljačkoj jedinici nalaze se i *Carry* i *Zero* zastavice cijelog mikroprocesora. One se postavljaju i/ili preuzimaju kod aritmetičko logičkih operacija preko odgovarajućih izlaza i ulaza na upravljačkoj jedinici. Linije $FC_0$ i $FC_1$ na MCU 3001 upravljaju ulazom, a $FC_2$ i $FC_3$ izlazom zastavica. Na osnovi stanja tih zastavica računaju se i uvjetni skokovi.

Mikroprogramska memorija ima 512 lokacija (9 mikroadresnih linija) po 32 bita. Organizirana je u obliku 2D matrice od 32 retka i 16 stupaca, ali je fizički ona izgrađena linearno. Zbog takve organizacije važno je znati da linije $MA_0$-$MA_3$ na MCU 3001 adresiraju *stupac*, a $MA_4$-$MA_8$ *redak* mikroprogramske memorije. Ovakva organizacija dolazi do izražaja kod skokova na slijedeću mikronaredbu, jer funkcije skoka omogućavaju skok unutar istog stupca, retka, grupe susjednih redova ili stupaca i slično. Ne postoji mogućnost direktnog skoka na proizvoljnu slijedeću mikroprogramsku adresu, ali to nije niti potrebno jer se logički povezane mikronaredbe najčešće smještaju jedna do druge. Prilikom pohranjivanja mikroriječi u mikromemoriju o tome treba voditi računa.

## 2.2. Osnovni procesni element CPE 3002

Procesni element CPE 3002 (prikazan na slici 3), bit-odrezak ili režanj porodice Intel 3000, obavlja aritmetičko-logičke operacije na dva bita makroprocesne riječi. Na sve procesne elemente dovodi se kompletna funkcijska sabirnica (linije $F_0$-$F_6$) iz mikroriječi, dok su ostale sabirnice veličine dva bita.




**Slika 3:** Procesni element CPE 3002

Procesni element ima slijedeće izvode:

**D1-D0** *Izlazna podatkovna sabirnica* - izlazna sabirnica podataka iz procesnih elemenata u makroprogramsku memoriju.

**A1-A0** *Makroadresna sabirnica* - izlazna sabirnica za adresiranje makromemorije.

**K1-K0** *Sabirnica konstante* - ulazna sabirnica sa binarnim uzorkom koji može detaljnije određivati funkciju procesnog elementa (kada oba bita imaju istu vrijednost) ili može predstavljati *masku* kod aritmetičko-logičkih operacija.

**M1-M0** *Ulazna podatkovna sabirnica* - ulazna sabirnica podataka iz makromemorije u procesni element.

**I1-I0** *Sabirnica podataka sa vanjske jedinice* - ulazna sabirnica sa podacima iz periferne jedinice (u konfiguraciji MP3000 ona se ne koristi).

**F6-F0** *Mikrofunkcijska sabirnica* - na ove se linije dovode bitovi odgovarajućeg polja mikroriječi koji određuju funkciju koju procesni element mora obaviti, te operande nad kojima tu funkciju obavlja.

**LI** *Ulaz za pomak.*

**CI** *Ulaz prijenosa.*

**RO** *Izlaz pomaka.*

**CO** *Izlaz prijenosa.*

**X, Y** *Predviđanje prijenosa* - ovi se bitovi vode na generator za predviđanje bita prijenosa, kojim se ubrzava izvoeđenj aritmetičko logičkih operacija.

Važno je voditi računa da sve sabirnice rade u negativnoj logici, odnosno da im je aktivno stanje logička 0, međutim programska podrška je pisana tako da korisnik radi u pozitivnoj logici.

Aritmetičko logička jedinica može obaviti cijeli niz operacija, a popis funkcija i mogućih operanada nalazi se u prilogu.

Procesni elementi imaju 12 dvobitovnih registara: $R_0$-$R_9$, T i akumulator AC. R registri su opće namjene, dok su T i AC pomoćni kod obavljanja aritmetičko-logičkih operacija, premještanja podataka među registrima ili rad sa makromemorijom (upis ili čitanje u/iz makromemorije mora ići preko akumulatora). Važno je reći da su to svi registri, odnosno nema posebno izvedenog programskog brojača, pokazivača stoga ili indeksnih registara. Oni se svi moraju organizirati korištenjem osnovnih R registara, tako da, za današnje prilike, ovo i nije naročita raskoš u broju registara. Ako se želi simulirati neki noviji raskošniji mikroprocesor, trebalo bi koristiti dio makromemorije kao radne registre, što rezultira bitno manjom brzinom i utroškom makromemorije.

## 3. Mikroprogramski razvojni sustav MP3000

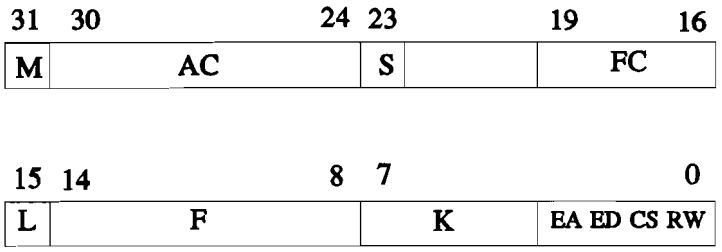
Razvojni sustav MP3000 izgrađen je oko porodice Intel 3000. Sadrži 4 procesna elementa CPE 3002, tako da tvori 8-bitno mikroračunalo sa 8-bitnom makroadresnom sabirnicom, te tako ima 256 okteta radne memorije.

Sam MP3000 nema mikro i makro memorije, pa je u istom kućištu i dodatni međusklop koji ima potrebnu memoriju, a sa PC računalom povezan je serijskom komunikacijom. O komunikaciji i upravljanju MP3000 brine se ugrađena programska podrška.

### 3.1. Opis mikroriječi sustava MP3000

Sustav MP3000 koristi 32-bitnu mikroriječ (vidi sliku 4). U nastavku su opisane funkcije pojedinih bitova.

## Mikroprogramska riječ




**Slika 4:** Mikroriječ sustava MP3000

**bit 0** - R/W*[1] bit (*Read/Write**). Kada je u logičkoj jedinici znači da ta mikronaredba treba pročitati sadržaj neke makroprogramske lokacije preko M sabirnice, analogno tome kada je u logičkoj nuli upisuje se vrijednost sa D sabirnice u makromemoriju; u oba slučaja se adresa lokacije nalazi na A sabirnici; ovaj je bit u funkciji je kada je aktivan bit 1 (CS bit) u mikroriječi.

**bit 1** - CS bit (*Chip Select*). Ako je u logičkoj jedinici mikronaredba radi operaciju čitanja ili pisanja u makromemoriju, a koja se operacija radi definira se u bitu 0.

**bit 2** - ED bit (*Enable Data*). Bit koji direktno upravlja ED linijom procesnih elemenata, a aktivno stanje označava da se podatak iz AC registra procesnih elemenata pojavi na njihovoj D sabirnici. Mora biti aktivan prilikom pisanja u makroprogramsku memoriju.

**bit 3** - EA bit (*Enable Address*). Bit koji direktno upravlja EA linijom svih procesnih elemenata, a aktivno stanje označava da se podatak iz MAR-a[2] pojavi na A sabirnici procesnih elemenata. Koristi se kada mikronaredba piše ili čita iz makromemorije.

**bitovi 4..7** - K bitovi (*mask*). Bitovi maske koji se vode na K sabirnicu procesnih elemenata. Prije je spomenuto da procesni elementi obavljaju jednu funkciju kada su svi K bitovi u log. 0, drugu kada su svi u log. 1, a treću kada su različiti. Budući da se ovi bitovi najčešće koriste kada su svi u istom stanju, a da bi se optimirala mikroprogramska riječ, sa ova se 4 bita upravlja sa svih 8 bitova K sabirnice na slijedeći način:

bit 4: stanje bita $K_0$;
bit 5: stanje bitova $K_1$, $K_2$ i $K_3$;
bit 6: stanje bitova $K_4$, $K_5$ i $K_6$; i
bit 7: stanje bita $K_7$.

**bitovi 8..14** - F bitovi $F_0$-$F_6$ (*Function*; $F_0$ je bit 8). Funkcijski bitovi koji se vode na odgovarajućih 7 izvoda $F_0$-$F_6$ svakog procesnog elementa. Pri tome bitovi $F_0$-$F_3$ određuju registar ili registarsku grupu koja sudjeluje u funkciji, a $F_4$-$F_7$ određuju funkciju. Popis svih funkcija koje procesni elementi mogu obavljati nalazi se u prilogu.

---

[1]Znakom * označava se negativna logika, tj. aktivno stanje je logičko 0.

[2]Engl. *Memory Address Register*; memorijski adresni registar, sadrži makroprogramsku adresu koju treba postaviti na A sabirnicu procesnih elemenata.

**bit 15** - LD bit (*Load*). Aktivno stanje (log. 1) aktivira LD liniju MCU 3001 kojom se određuje da se adresa slijedeće mikronaredbe ne računa pomoću AC bitova, već na osnovu operacijskog koda makronaredbe dovedenom na SX i PX linije MCU (koje su spojene na M sabirnicu procesnih elemenata) prema rasporedu u tablici 1.

| Bitovi mikroprogramske adrese | $MA_8$ | $MA_7$ | $MA_6$ | $MA_5$ | $MA_4$ | $MA_3$ | $MA_2$ | $MA_1$ | $MA_0$ |
|---|---|---|---|---|---|---|---|---|---|
| Bitovi SX i PX sabirnice | 0 | $SX_3$ | $SX_2$ | $SX_1$ | $SX_0$ | $PX_7$ | $PX_6$ | $PX_5$ | $PX_4$ |
| Bitovi M sabirnice | 0 | $M_3$ | $M_2$ | $M_1$ | $M_0$ | $M_7$ | $M_6$ | $M_5$ | $M_4$ |

**Tablica 1:** Raspored bitova operacijskog koda makronaredbe za računanje mikroadrese

Važno je uočiti da se mikroprogramska adresa tvori tako da se zamijene gornja i donja četiri bita operacijskog koda makronaredbe, pri čemu se $MA_8$ uvijek postavlja na 0. Npr. Makronaredba sa operacijskim kodom $25_{(16)}$ dovedenim na PX i SX sabirnice MCU, uz LD bit u stanju 1, uzrokovati će izvođenje mikroprograma od adrese $052_{(16)}$.

**bitovi 16..19** - FC bitovi $FC_0$-$FC_3$ (*Flag Control*, $FC_0$ je bit 16). Ovi bitovi kontroliraju operacije sa zastavicama i vode se direktno na MCU 3001. Sa $FC_0$ i $FC_1$ odabire se zastavica (niti jedna, C, Z ili obje) za ulazni podatak sa linije FI (*Flag Input*), a sa $FC_2$ i $FC_3$ određuje se što će se pojaviti (0, C, Z ili 1) na liniji FO (*Flag Output*).

**bitovi 20..22** - trenutno neiskorišteni bitovi mikroprogramske riječi.

**bit 23** - S bit (*Set flag*). Kada je postavljen u log. 1 omogućeno je djelovanje funkcija za rad sa zastavicama na linijama $FC_0$-$FC_1$. Ovaj je bit uveden zbog specifične sklopovske izvedbe MP3000, a o njegovom stanju brine se programska podrška.

**bitovi 24..30** - AC bitovi $AC_0$-$AC_6$ (*Address Control*, $AC_0$ je bit 24). Ovim se bitovima određuje adresa slijedeće mikronaredbe u slučaju da je LD bit u logičkoj nuli. Postoji niz uvjetnih i bezuvjetnih skokova po mikroprogramskoj memoriji, a detaljni opis može se naći u prilogu.

**bit 31** - M bit. Označava upravljačkom programu koji kontrolira rad MP3000 da je to zadnja mikronaredba koju treba izvesti, odnosno da treba zaustaviti izvođenje simulacije. Ovaj bit nema uticaja na neku funkciju unutar MP3000.

### 3.2. Osnovni princip rada sustava MP3000

Prateći sliku 1 na kojoj je prikazan način povezivanja porodice Intel 3000, a koji se koristi i u MP3000, moguće je objasniti princip rada ovog sustava.

Pretpostavimo da je za određenu makronaredbu potrebno izvršiti niz mikronaredbi. Uz postavljen LD bit u mikroriječi, MCU će na osnovu operacijskog koda makronaredbe generirati mikroadresu prve mikronaredbe. Na sabirnici MA pojaviti će se ta adresa, a na izlazu iz upravljačke memorije pojavljuje se odgovarajuća mikroprogramska riječ. Bitovi dohvaćene mikroriječi upravljaju pojedinim aktivnostima u sustavu: bitovi F, EA, ED i K upravljaju sa CPE, bitovi AC, LD, FC upravljaju sa MCU, bitovi CS i RW koriste se za radnu makromemoriju, dok je M bit oznaka kraja rada sustava (koristi ga programska podrška). Ovisno o izvršenoj mikronaredbi, CPE postavljaju makroadresu na svoju A sabirnicu zbog rada sa makromemorijom. Podatak koji se upisuje u makromemoriju nalazi se na D sabirnici, a podatak koji se čita preuzima se preko M sabirnice. Nakon što se izvede i zadnja mikronaredba, preuzima se operacijski kod slijedeće makronaredbe.

Ostali detalji oko principa rada i programiranja mogu se naći u nastavku teksta u kojem se opisuje programska podrška.

# 4. Korištenje programa za razvoj mikroprograma

Za korištenje razvojnog sustava MP3000 postoji programska podrška koja omogućava pisanje mikroprograma, pisanje makroprograma, prevođenje makroprograma, te izvođenje mikro i makro programa uz praćenje stanja registara, zastavica, sabirnica i makromemorije. Programska je podrška pisana tako da se korisnika potpuno oslobodi brige oko prebacivanja napisanih programa u sustav MP3000 i specifičnih radnji vezanih na simulirano izvođenje programa. Svi su ti "kućanski poslovi" za korisnika potpuno nevidljivi. U nastavku je opisan razvojni program i njegovo korištenje, uz prikaz svih izbornika i poruka koje se javljaju u programu. Upute su pisane onim redom kojim se i radi sa programom prilikom razvoja mikro i makro programa.

## 4.1. Pokretanje programa

Najjednostavnije pokretanje programa je sa samo PCMP i <ENTER>. Međutim uz ime prilikom pokretanja iz komandne linije DOS-a može se navesti nekoliko opcija opisane u nastavku.

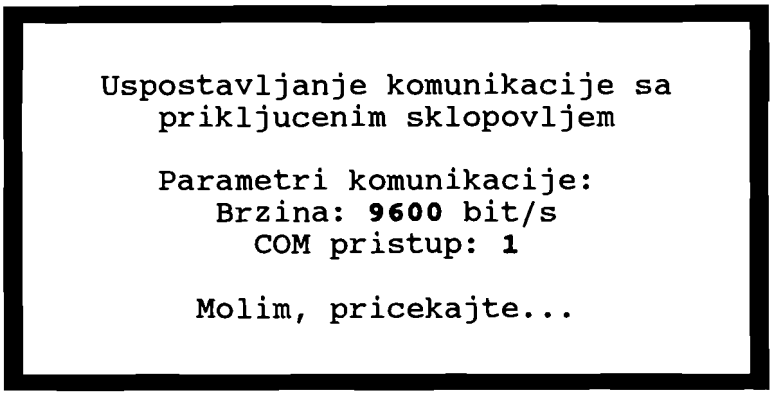
- **/br=n** - zadavanje brzine serijske komunikacije; ta se brzina mora podudarati sa onom podešenom sa mikroprekidačima na međusklopu (koristi se 9600 b/s);
- **/com=n** - zadavanje rednog broja serijskog pristupa na koji se priključio kabel mikroprogramskog simulatora; moguće vrijednosti su 1 ili 2 za COM1, odnosno COM2;
- **/nohw** - ovom se opcijom kazuje programu da se radi bez priključenog mikroprogramskog simulatora;
- /? - ispis kratkih uputa o svim ovim opcijama; te se upute ispisuju i prilikom svakog izlaska iz programa.

Primjer: sa pcmp `/br=4800 /com=2` program se pokreće tako da radi brzinom 4800 b/s preko serijskog pristupa COM2.

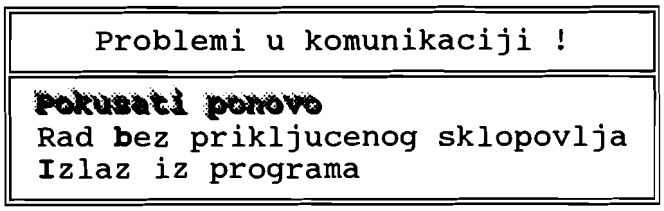
Niti jedna opcija nije obavezna. Ukoliko se ne zada brzina i/ili serijski pristup, program postavlja zadnje stanje koje pročita iz konfiguracijske datoteke. Ako ta datoteka ne postoji, postavlja vrijednosti 9600 b/s i COM1.

Stanje opcije `/nohw` ne pamti se u konfiguracijskoj datoteci i potrebno ju je navesti svaki put kad se stvarno radi bez sklopovlja. U tom slučaju program ne pokušava uspostaviti komunikaciju sa sklopovljem, odmah se ispisuje glavni izbornik (opisuje se u slijedećem poglavlju) i dozvoljava rad sa onim modulima programa koji ne zahtjevaju prisustvo sklopovlja. Ukoliko se nakon toga priključi sklopovlje, potrebno je izaći iz programa, i ponovo ga pokrenuti bez opcije `/nohw`.

Kada se program pokrene sa priključenim sklopovljem najprije se radi sinkronizacija serijske komunikacije. To se radi uvijek, te je potrebno međusklop resetirati ako se program na osobnom računalu pokrene ponovo, a međusklop nije isključivan (pa se nije sam resetirao ponovnim uključenjem). Za to vrijeme na zaslonu stoji prozor na kojem se vidi sa kojim parametrima program radi:

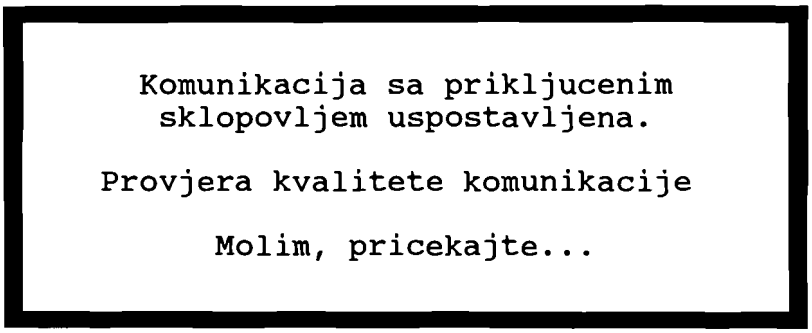



Ako se sinkronizacija uspostavi, program ide na testiranje kvalitete komunikacije. Ukoliko sinkronizacija ne uspije, ispisuje se slijedeći izbornik:

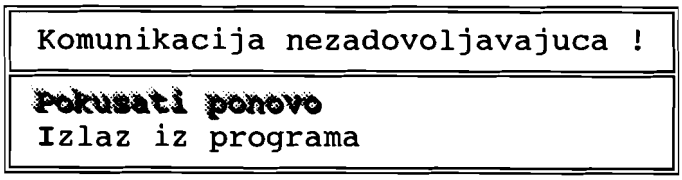



Nakon što se pokuša otkloniti problem može se prvom opcijom pokušati ponovo uspostaviti komunikacija. Ako se u stvari radi bez sklopovlja, ali se prilikom poziva programa zaboravilo navesti opciju `/nohw`, može se opcijom "`Rad bez priključenog sklopovlja`" programu reći da nastavi rad bez sklopovlja. Ako se radi o nekom većem problemu, opcijom "Izlaz iz programa" prekida se rad.

Iza sinkronizacije radi se ispitivanje kvalitete komunikacije. Tada se prikazuje slijedeći prozor:

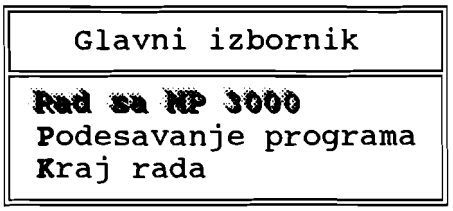



Ako je kvaliteta komunikacije zadovoljavajuća (način ispistivanja opisan je u prethodnom poglavlju), program nastavlja rad prikazujući glavni izbornik. U suprotnom slučaju nešto nije u redu, iako je sinkronizacija prije postignuta. U tom se slučaju pojavljuje slijedeći izbornik i od korisnika traži daljnja akcija:



## 4.2. Rad sa programom

Glavni se izbornik prikazuje nakon uspješne provjere serijske komunikacije ili odmah, ako je program pokrenut bez sklopovlja:



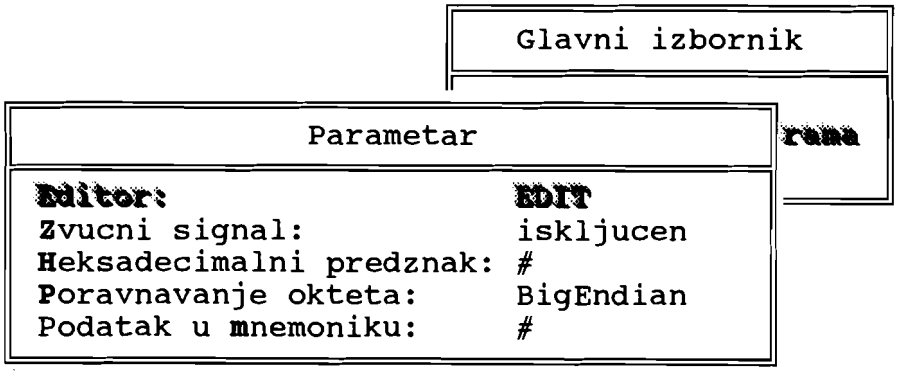

Ovdje će biti opisan princip prikaza i rada sa izbornicima, iako su oni već maloprije spomenuti kod provjere komunikacije.

Svi izbornici izledaju slično: naslov izbornika govori o kakvom se izboru radi, ispod naslova su sve ponuđene opcije. Određena se opcija pokreće tako da se tipkama za smjer željena opcija prikaže invertirano (zamjenjene boje pozadine i tinte), a tipkom ENTER se ona aktivira. Većina opcija u izbornicima ima jedno slovo istaknuto; to je tipka kojom se opcija direktno aktivira bez kretanja tipkama za smjer. Dok je izbornik aktivan, ali i u ostalim dijelovima programa u posljednjem retku zaslona nalazi se popis tipki rapoloživih u tom trenutku, tako da korisnik vidi kojim se tipkama može koristiti. Iz svakog se izbornika može izaći bez odabira ijedne opcije (i vratiti u prethodni izbornik, ako postoji) tipkom ESC.

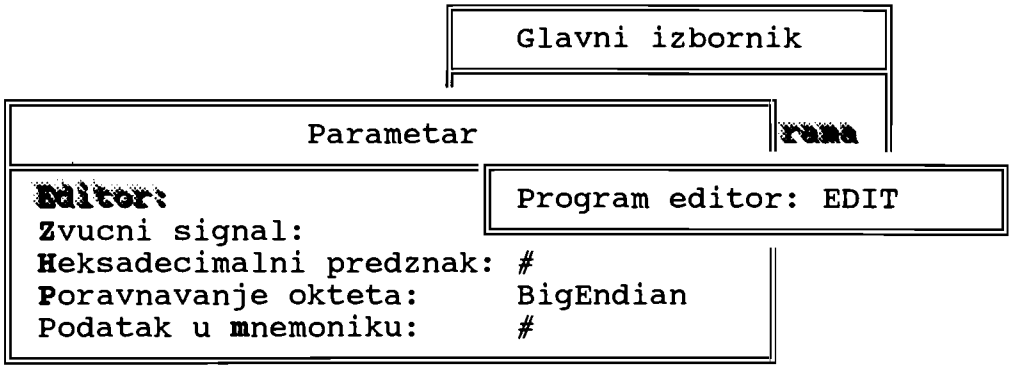
Prije samog opisa načina mikro i makro programiranja opisati će se opcije "Podešavanje programa" i "Kraj rada".

### *4.2.1. Podešavanje programa*

Opcija podešavanja program u glavnom izborniku služi za podešavanje raznih parametara na razini cijelog programa. Odabirom ove opcije prikazuje se izbornik sa parametrima koje je moguće podesiti i njihova trenutna stanja ili vrijednosti:

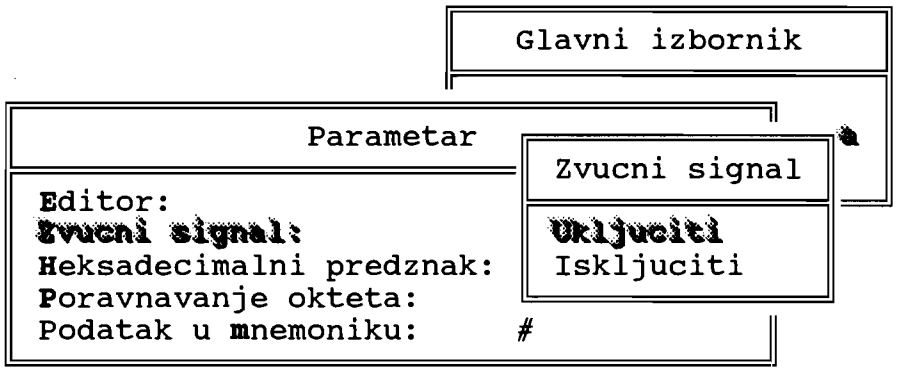



Opcijom "Editor" otvara se okvir za unos imena programa (ASCII tekst editora) koji se koristi prilikom makroprogramiranja. Taj se program automatski poziva iz ove programske okoline, tako da se ne mora izlaziti iz programa PCMP prilikom pisanja ili ispravljanja makroprograma. U okviru se nalazi ime trenutno podešenog programa, pa to ime treba prepraviti u željeno. Sa ESC može se odustati od unosa.

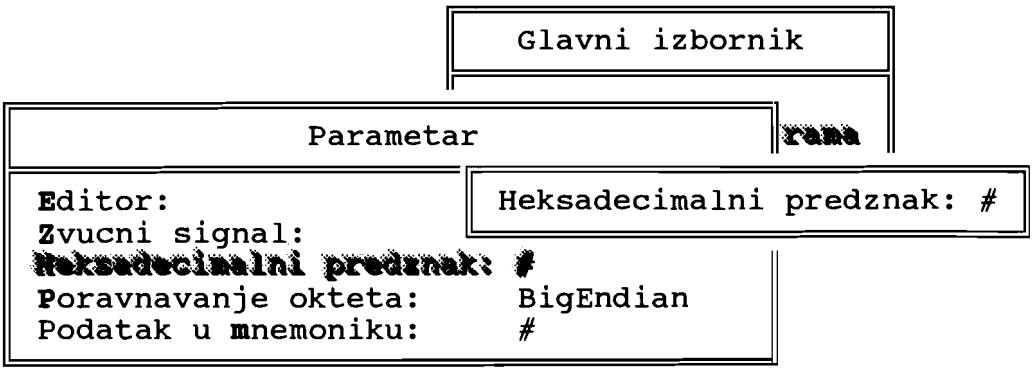



Editor može biti bilo koji, s tim da treba voditi računa o njegovoj veličini s obzirom na slobodnu memoriju, jer se tada u memoriji nalaze oba programa. Međutim, sam PCMP program je relativno mali, tako da se može koristiti praktično bilo koji editor. Provjereno je korištenje slijedećih editora: **NE** (Norton Editor), **EDIT** (iz DOS-a 5.0 ili 6.0), **Brief 3.0**, pa čak i **WordPerfect 5.1** (s time da se napisano mora snimiti kao "DOS text"). Potrebno je napomenuti da se kod ovog navođenja imena programa, navodi samo njegovo osnovno ime, a i ne put do njegovog direktorija. Zato put do njegovog direktorija mora biti podešen sa PATH naredbom iz DOS-a koja se obično podešava u AUTOEXEC.BAT datoteci. Početno se postavlja EDIT iz DOS-a, jer se on nalazi na svim osobnim računalima sa DOS-om 5.0 ili kasnijom inačicom.

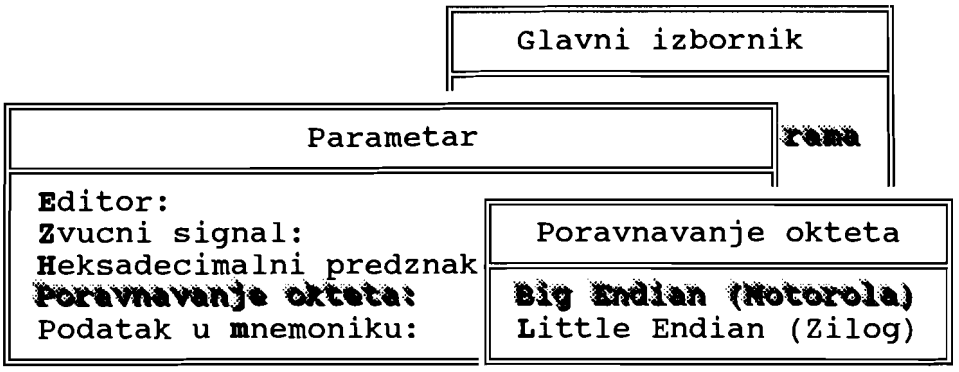
Slijedeća opcija je uključivanje/isključivanje zvučnog signala koji se koristi prilikom ispisa važnih upozorenja u prozoru, kako bi se korisniku skrenula pažnja na obavijest. Početno je zvučni signal isključen.




Predznak za heksadecimalne brojeve različit je od asemblera do asemblera, pa su programeri i korisnici naučeni na različite oznake (najčešće su to '#' ili '$'). Ovdje podešenu oznaku program koristi kod svih ispisa (neki ispisi su isključivo u heksadecimalnom prikazu pa se predznak izostavlja radi kraćeg ispisa), ali taj znak mora koristiti i programer u svojim makroprogramima, inače će biti prijavljena greška. Ne može se podesiti notacija npr. 03EH. Pretpostavljena oznaka je '#'.

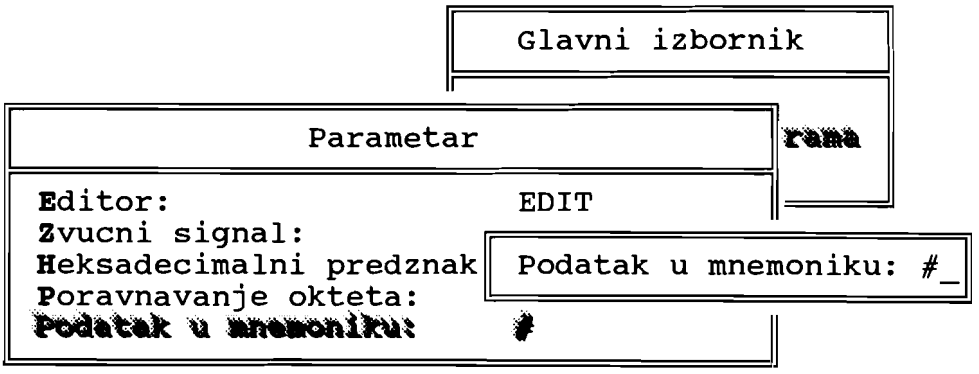



Poravnavanje okteta odnosi se na redoslijed pohranjivanja ili čitanja višeoktetnih podataka u 8-bitnoj makromemoriji koji si korisnik može sam podesiti prilikom rada u ovom razvojnom sustavu.




U izvođenju programa isključivo se koristi ovaj parametar prilikom disasembliranja, dok se u makroprogramima može asemblerskim pseudonaredbama to mijenjati, s time da se ovdje podešeno uzima kao pretpostavljeno. Moguće je koristiti tzv. *BigEndian* koji se koristi na npr. Motorolinim mikroprocesorima (najprije se pohranjuje oktet najveće težine, a iza toga prema nižim težinama; gledajući prema rastućim adresama), ili tzv. *LittleEndian* na npr. Zilogovim i Intelovim mikroprocesorima (obratno od Motorole). Pretpostavljeni redoslijed je BigEndian (Motorolin način).

Posljednji podesivi parametar je oznaka za podatkovni dio u mnemonicima koje korisnik definira prilikom mikroprogramiranja.



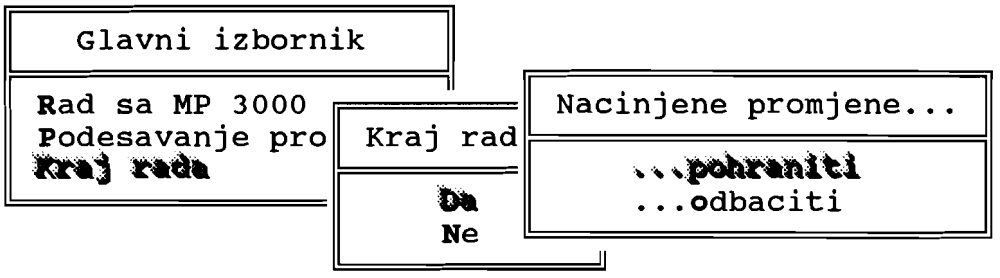

O tome više kod mikroprogramiranja, a ovdje je dovoljno reći da je pretpostavljena oznaka '#', i ne smeta ako je ista kao predznak za heksadecimalne brojeve, jer se ne koristi na istim mjestima u razvojnom sustavu. Ovo se treba promijeniti, ako se npr. rade mnemonici nalik onima na MC68000 u kojima se koristi izravno

adresiranje označeno upravo sa '#'.

### *4.2.2. Izlazak iz programa*

Opcijom "Kraj rada" u glavnom izborniku izlazi se iz programa, ali se prije samog izlaska traži potvrda da se stvarno želi izaći.

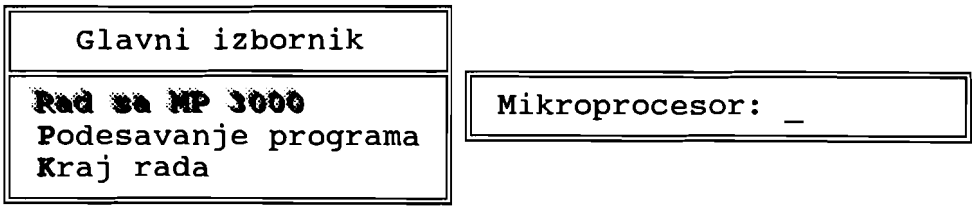

Nakon potvrdnog odgovora da se želi van iz programa, program provjerava da su u toku rada učinje kakve promjene na parametrima koji se pohranjuju u datoteku PCMP.CFG (to nisu samo parametri iz prije opisanog "Podešavanja programa"). Ako ih nema, tada slijedećim izbornikom:




...korisnik može odabrati da se načinjene promjene pohrane ili ne pohrane u spomenutu datoteku. Ako se promjene pohrane, program će to ponovo postaviti prilikom slijedećeg pokretanja.

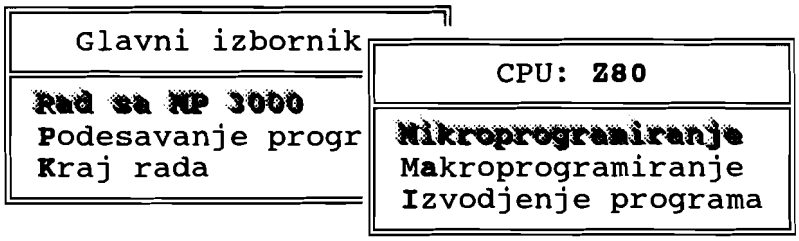
### *4.2.3. Priprema za mikroprogramiranje*

Rad na mikro i makro programiranju započinje opcijom "Rad sa MP3000" u glavnom izborniku. Odabirom te opcije pojavljuje se okvir za unos:




Tu je potrebno unijeti ime mikroprocesora (stvarnog ili zamišljenog) za koji se želi raditi simulacija u ovom razvojnom sustavu. Namjerno se kaže "ime mikroprocesora" budući da se modulu za mikroprogramiranje definiraju asemblerski mnemonici tog mikroprocesora, njihovi operacijski kódovi, mikroprogrami za te makronaredbe, pa čak i registri, a to sve sa programske strane čini mikroprocesor. Ime može sadržavati slova i brojeve do ukupne dužine od 8 znakova. Uneseno ime mikroprocesora program koristi za ime datoteke sa nastavkom *.CPU u koju pohranjuje sve što se definira za taj mikroprocesor.

Zadavanjem imena mikroprocesora pojavljuje se izbornik sa modulima razvojnog sustava u kojima se mikroprogramira, makroprogramira ili izvodi programa, a sve na osnovu odabranog mikroprocesora čije se ime nalazi u naslovu tog izbornika.




Za novi mikroprocesor logičan redoslijed je mikroprogramiranje, zatim makroprogramiranje definiranim mnemonicima, a na kraju izvođenje. Jednom definirani mnemonici, mikroprogrami i makroprogrami spremaju se u datoteke tako da se kasnije mogu koristiti bez ponovnog pisanja.

Međutim, prije rada na novom mikroprocesoru potrebno je najprije izvršiti pripremu, budući da je mikroprogramiranje vrlo niska razina razvoja, te se ne može odmah sjesti za računalo i pisati mikroprograme.

☞ Prije svega, potrebno se upoznati sa bit-odrescima *Intel CPE 3002* (njihovim registrima, mikrooperacijama koje mogu obavljati, pristup makromemoriji), pripadajućom mikroprogramskom upravljačkom jedinicom *Intel MCU 3001* (organizacija mikromemorije, računanje mikroprogramskih adresa, skokovi na slijedeću mikroadresu, operacije sa zastavicama), načinom na koji se sa njima gradi mikroračunalo, kako je organizirana mikroprogramska riječ, te neki važni detalji vezani za samo konstrukciju mikroprogramskog simulatora MP3000 koliko je to nužno za samo mikroprogramiranje (npr. S bit i M bit u mikroprogramskoj riječi). Daljnji tekst podrazumijevati će poznavanje navedene materije.

Postoji i priprema koja se radi za svaki novi mikroprocesor. Najprije je potrebno definirati zadatak, za koji je potrebno razmotriti da li se postojeća arhitektura sustava MP3000 uopće može iskoristiti za realizaciju istog. To se naročito odnosi na broj i veličinu raspoloživih registara. MP3000 je izgrađen kao 8-bitni mikroprocesor sa 10 registara opće namjene. Ne postoje dodatni registri sa unaprijed definiranom namjenom kao što su npr. PC (programsko brojilo) ili SP (pokazivač stoga). Sve se to mora raditi sa spomenutih 10 registara. Korisnik sam određuje koji će od tih registara biti **Program Counter**, koji **Stack Pointer** i slično. Zato se preporuča raditi tipične makronaredbe, a ne mnogo sličnih koje rade istu operaciju samo sa različitim registrima.

Priprema za projektiranje mikroprocesora opće namjene uključuje slijedeće:

1) definirati broj radnih registara i njihovu ulogu (akumulator, registar opće namjene, programsko brojilo, pokazivač stoga, indeksni registri, registar stanja i dr.);
2) definirati naredbeni skup željenog mikroprocesora prema osnovnim grupama naredbi:
   - naredbe za punjenje registara (iz registra u registar, iz registra u memoiju, iz memorije u registar u različitim modovima adresiranja);
   - aritmetičko logičke naredbe;
   - naredbe skokova (uvjetni, bezuvjetni, relativni, apsolutni);
   - naredbe za rad sa stogom;
   - specijalne naredbe (pretraživanja memorije, izmjena registara i sl.);
3) provjeriti da li naredbeni skup sa svojim mikroprogramima može stati u 512 lokacija mikromemorije, te da li ima do 256 različitih makrokódova;

Daljnji rad podrazumijeva izradu sekvenci mikronaredbi koje će obavljati zadanu akciju makronaredbe, i to za svaku makronaredbu iz definiranog skupa. To je već faza mikroprogramiranja. Tokom toga treba razmišljati i o smještanju mikroprograma u mikromemoriju, imajući na umu moguće skokove na slijedeću mikrolokaciju. Ove radnje se u početku mogu najprije raditi na papiru, ali **programska je podrška pisana**

**upravo tako da se, nakon početnog upoznavanja sa načinom rada, sve ovo može raditi direktno u programu, bez korištenja papira i literature sa popisima i objašnjenjima mikroprogramskih operacija.**

### *4.2.4. Mikroprogramiranje*

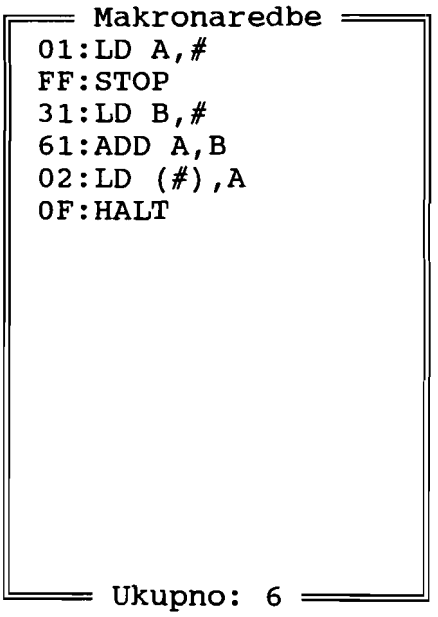
Pod pretpostavkom da korisnik ima gore navedeno predznanje i načinjenu pripremu, može se odabrati prva opcija "Mikroprogramiranje" u prethodnom izborniku. Tom se opcijom ulazi u modul za mikroprogramiranje. Na zaslonu se otvara ukupno 4 prozora:

1) prozor sa popisom definiranih makroasemblerskih mnemonika trenutnog mikroprocesora;
2) prozor sa prikazom svih lokacija mikroprogramske memorije;
3) prozor sa detaljim prikazom mikroprogramske riječi sa određene lokacije u mikromemoriji; uz njega se nalazi simbolički (mnemonički) prikaz vrijednosti pojedinih polja u mikroprogramskoj riječi;
4) prozor sa popisom sinonima registara.

U jednom trenutku može biti aktivan samo jedan prozor, i sve u tom trenutku raspoložive komande (popisane su u posljednjem retku zaslona) odnose se na taj prozor. Aktivni prozor prepoznaje se po dvostrukoj liniji okvira oko prozora, dok neaktivni prozori imaju jednostruku liniju. U nastavku su detaljno opisani svi prozori i rad sa njima. Na odsječcima zaslonskih prikaza vidjeti će se izrađeni primjer za mikroprocesor Z80.

#### *4.2.4.1. Prozor sa mnemonicima makronaredbi*

Po ulasku u modul za mikroprogramiranje, aktivan je prozor sa popisom mnemonika definiranih makronaredbi. Prozor izgleda ovako:

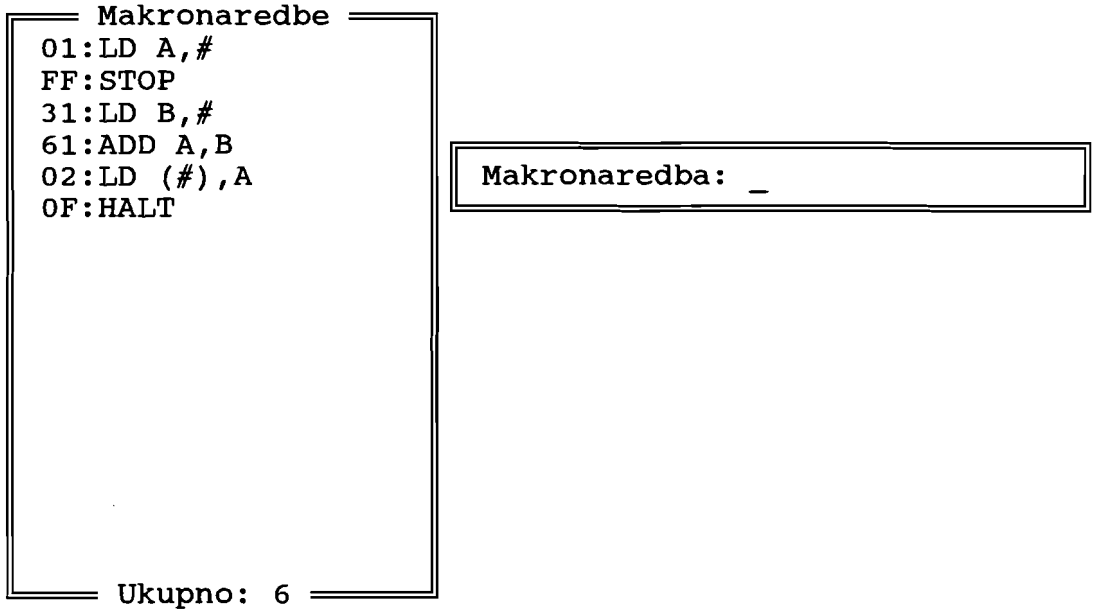



U prozoru se odjednom prikazuje do 17 definiranih makronaredbi, od popisa od najviše 256 makronaredbi. U dnu prozora piše koliko je trenutno definiranih makronaredbi. Tokom rada aktivna je jedna makronaredba na čijoj se mikroprogramskoj sekvenci radi, a prikazana je inverzno. Sa lijeve strane mnemonika piše operacijski kód te makronaredbe koji se definira prilikom dodavanja nove makronaredbe, a ispisan je uvijek u heksadecimalnom obliku.

Aktivnu makronaredbu bira se tipkama za smjer (kursorske tipke) ↓ i ↑. Kako se kreće po popisu

makronaredbi, tako se u neaktivnom mikromemorijskom prozoru mikrolokacije koje pripadaju trenutno aktivnoj makronaredbi prikazuju svijetlije, tako da korisnik ima pregledan prikaz. Po tome se vidi da program "zna" koje mikroprogramske lokacije pripadaju kojoj makronaredbi, iako se to po samoj mikroriječi ne može odrediti. Pored toga, u prozoru mikroriječi detaljno je prikazana prva mikroriječ te makronaredbe, a ona se nalazi na mikrolokaciji određenoj operacijskim kódom makronaredbe.

Nova se makronaredbe dodaje tipkom **INS** (ili **Insert**). Pritiskom na tu tipku otvara se okvir za unos:




U tom se okviru unosi makronaredba opisana na poseban način; takav da ga prilikom makroprogramiranja asembler prepozna i prevede u njegov operacijski kód zajedno sa podacima. Vrijedi slijedeće pravilo: makronaredba se unosi onako kako će se pisati u makroprogramima, s time da se na mjestu gdje se u naredbi očekuje numerički podatak stavi specijalni znak za **oznaku podatka**. Taj se specijalni znak određuje proizvoljno u opciji "**Podešavanje programa**" glavnog izbornika programa, a početno je namješten '#'. Taj specijalni znak koristi i asembler prilikom prepoznavanja makronaredbe u makroprogramu.
Slova se mogu kucati ili mala ili velika, program ih ionako pamti kao velika.

Tih podatkovnih oznaka može i biti i više uzastopno, pri čemu svaki vrijedi za očekivani 1 oktet podatka. Ukupno može biti do 4 uzastopne podatkovne oznake, čime se omogućava rad sa 8, 16, 24 i 32-bitnim podacima (to mogu biti i konstantne adrese). Nizova podatkovnih oznaka može biti i više u istoj definiciji. Razmak između mnemonika i operanda je moguć. Može se unijeti više uzastopnih razmaka, ali program pamti samo jedan, a koji u makroasemblerskom programu vrijedi za bilo kakav razmak (jedan ili više razmaka i/ili tabulatorske oznake). Evo nekoliko primjera:

| | | |
|---|---|---|
| ADD A,B | - | prevedenu makronaredbu čini samo operacijski kód bez podataka; |
| LD A,# | - | iza operacijskog kóda makronaredbe asembler će smjestiti 8-bitni podatak preveden iz makroasemblerskog programa; ako je operacijski kód ove makronaredbe npr. #01, a u programu se napiše **LD A,7**, asembler će to prevesti u niz 01 07; |
| LD HL,## | - | iza operacijskog kód smjestiti 16-bitni podatak; |
| LD (IX+#),# | - | iza operacijskog kóda smjestiti najprije prvi 8-bitni podatak, a zatim drugi; |
| LD (##),## | - | iza operacijskog kóda smjestiti dva 16-bitna podatka. |

Samo prepoznavanje tipa makronaredbe i prikupljanje podataka iza operacijskog kóda stvar je mikroprograma koji sve to radi i koji treba u tom smislu napisati.

Potrebno je navesti i nekoliko napomena oko definiranja makronaredbi.

*Ime registra* koje se navodi u definiciji makronaredbe nije povezano sa sinonimima. Sinonimi služe samo za jasniji prikaz u kasnijem izvođenju programa. Oni čine osnovno ime makronaredbe kao i sam mnemonik.

Ne postoji prepoznavanje *adresnih modova*. Ne može se niz sličnih naredbi npr. MOVE D0,D1; MOVE D0,D2; MOVE D0,D3 itd. opisati jednom definicijom. Njih treba navesti svaku pojedinačno, ali treba razmisliti ima li smisla definirati takve naredbe s obzirom na ograničenja mikroprogramskog simulatora (o tome je pisano u prethodnom poglavlju vezano na pripreme za mikroprogramiranje).

*Slične naredbe* nisu dozvoljene. Pod sličnim naredbama podrazumijevaju se naredbe koje se podudaraju u istom slovu, istoj poziciji jednog ili više uzastopnih razmaka, te istoj poziciji jedne ili više uzastopnih podatkovnih oznaka. Primjer: **LD A,#** i **LD A,##** ne mogu istovremeno postojati jer niti jedan asembler ne može znati kako će prevesti podatak, da li kao 8 ili 16 bitni. U tom i sličnim slučajevima ispisuje se slijedeće upozorenje:

```
Makronaredba "LD A,##" vec postoji
u istom ili vrlo slicnom obliku.

Odaberite drugaciji mnemonik.

Pritisnite bilo koju tipku
```

Tada je potrebno promijeniti samo ime mnemonika u npr. **LDW A,##**.

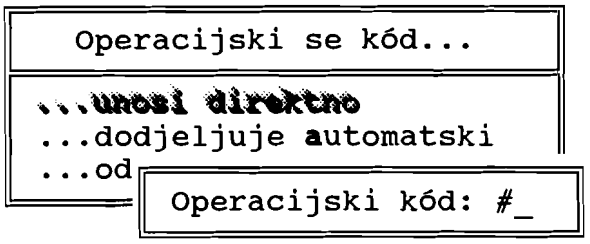
Nakon što se unese opis mnemonika makronaredbe, program trži da se definira njen operacijski kód. Operacijski kód je isključivo 8-bitni, i na osnovu njega MCU 3001 nalazi mikroadresu prve mikronaredbe sekvence koja izvršava tu makronaredbu. Mikroadresa je 9-bitna, a adresa je određena time što se zamijene bitovi polovica okteta, a najviši bit je 0. To je vidljivo iz slijedećeg prikaza:

| Bitovi mikroprogramske adrese | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| Bitovi operacijskog kóda | (uvijek 0) | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 |

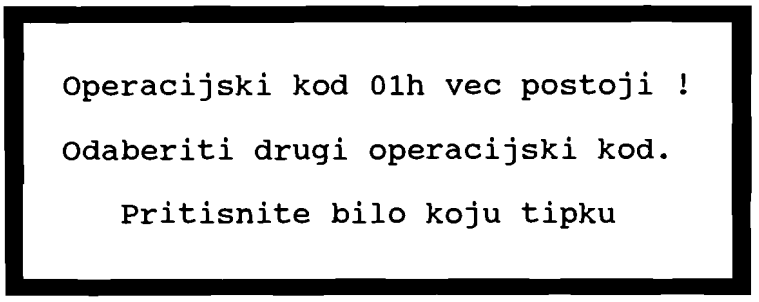
Operacijski se kód može unijeti na više načina koji se odabire iz slijedećeg izbornika:

```
Operacijski se kód...

...unosi direktno
...dodjeljuje automatski
...odabire u mikromemoriji
```

Direktan unos otvara dodatni okvir za unos u koji se u heksadecimalnom obliku (na to upućuje oznaka heksadecimalnog predznaka) ručno unosi operacijski kód makronaredbe:




Operacijski kód mora biti jedinstven, do sada ne dodijeljen, tj. mora prema gore opisanom načinu, pokazivati na slobodnu mikroprogramku lokaciju. Ako to nije zadovoljeno ispisuje se upozorenje:

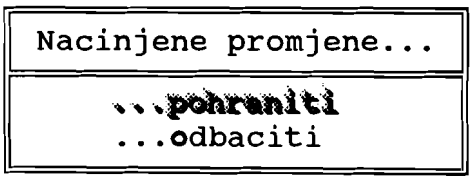



Drugi način je da program sam dodjeljuje operacijski kód, a to radi tako da traži prvu slobodnu lokaciju u mikroprogramskoj memoriji, i na osnovu njene adrese (obrnuto kako je prije gore opisano) dobija operacijski kód. Postoje neke preporuke u vezi korištenja ove opcije, a to se objašnjava u poglavlju sa dodatnim napomena i preporukama u nastavku.

Treći način dodjele operacijskog kóda je vjerojatno i najzgodniji, a to je odabir lokacije u mikroprogramskoj memoriji. Odabirom ovog načina, moguće je "šetati" tipkama za smjer po lokacijama po mikromemoriji i odabrati lokaciju prve mikronaredbe, a time i operacijski kód. Prilikom kretanja tekuća lokacija za odabir treperi, a odabir se potvrđuje sa ENTER. Moguće je odabrati samo slobodnu mikroprogramsku lokaciju. Simboli u prikazu mikroprogramske memorije su slijedeći (cijelo su vrijeme prikazani na zaslonu sa lijeve strane prozora mikroprogramske memorije): ⣿ označava zauzetu lokaciju, a ■ označava slobodnu lokaciju. Tipkom ESC može se odustati od takvog odabira.

Dok je aktivan prozor sa makronaredbama može se sa ESC završiti rad u cijelom ovom modulu i vratiti u izbornik mikroprogramiranja, makroprogramiranja ili izvođenja programa. Iz ostalih se prozora ne može direktno izaći iz cijelog modula već je potrebno vratiti se u ovaj prozor (također sa ESC).

Program pazi na to da li je u ovom modulu učinjena kakva izmjena, te prilikom izlaska iz modula nudi pohranjivanje ili odbacivanje tih promjena. Odbacivanje podrazumijeva da se promjene ne spremaju u datoteku *.CPU, ali one ostaju u memoriji dok se ponovo ne unese ime mikroprocesora.



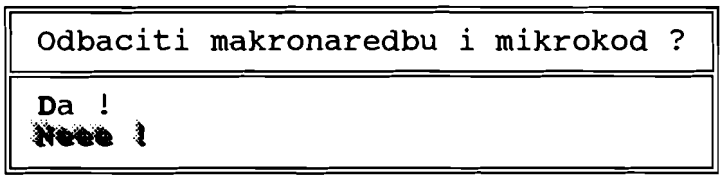

Dakle, sve promjene načinjene u popisu mnemonika ili u mikromemoriji snimaju se već prilikom izlaska iz modula mikroprogramiranja, a ne kao općeniti parametri razvojnog sustava prilikom izlaska iz cijelog programa.

Iz ovog se prozora može tipkom **F9** pozvati prozor sa definicijama sinonima registara koji se opisuje u

nastavku.

Pritiskom na tipku ENTER na određenoj makronaredbi, odlazi se prozor mikromemorije gdje se na mikroprogramskoj sekvenci odabrane makronaredbe. Mikromemorijski prozor je također opisan u nastavku.

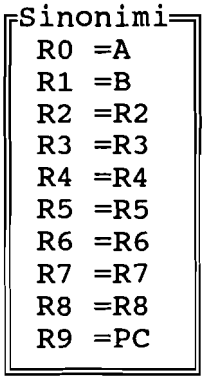
Pritiskom na tipku **DEL** na određenoj makronaredbi ona se briše iz popisa, a također i mikroprogramske riječi koje pripadaju toj makronaredbi koje se tada proglašavaju slobodnima. Prije te destruktivne operacije traži dodatna potvrda izbornikom:




Ovo brisanje mikroprogramskih riječi izbrisane makronaredbe posebno je važno ako je neka mikroprogramska sekvenca zajednička za više makronaredbi, a definirana je kao da pripada jednoj. Ta mogućnost postoji i opisana je u poglavlju sa dodatnim napomenama.

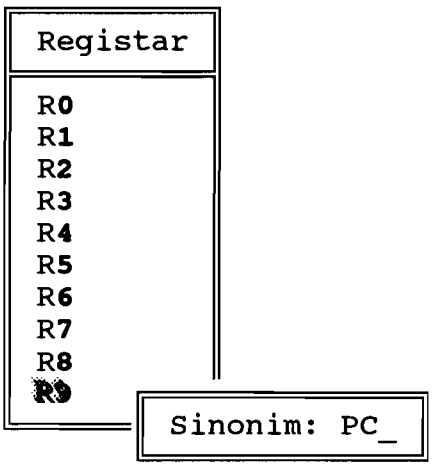
*4.2.4.2. Prozor sa sinonimima registara*

U prozor sa sinonimima registara dolazi se tipkom **F9** iz prozora sa makronaredbama.




Dodjela sinonima registrima ugrađena je ponajprije radi lakšeg rada prilikom izvođenja programa, kada na zaslonu imamo prikaz svih registara, a nije lako napamet znati namjenu svih deset registara samo po oznaci R0, R1, R2 itd., već je zgodnije imati imena prema namjeni koju smo im dodjelili u pripremi za taj mikroprocesor (A, PC, SP...). Kako je rečeno, sami sinonimi nemaju veze sa definiranim makromnemonicima. Tokom rada na mikroprogramiranju, u ovom se prozoru nalazi popis originalnih imena i sinonima, jer se u tom modulu ipak mora raditi sa osnovnim oznakama. Definirani mnemonici prikazani su svijetlije.

Dolaskom u taj prozor, odmah se otvara izbornik sa popisom registara, od kojih možemo odabrati onaj kojem želimo dodijeliti ime koje mu odgovara namjeni.

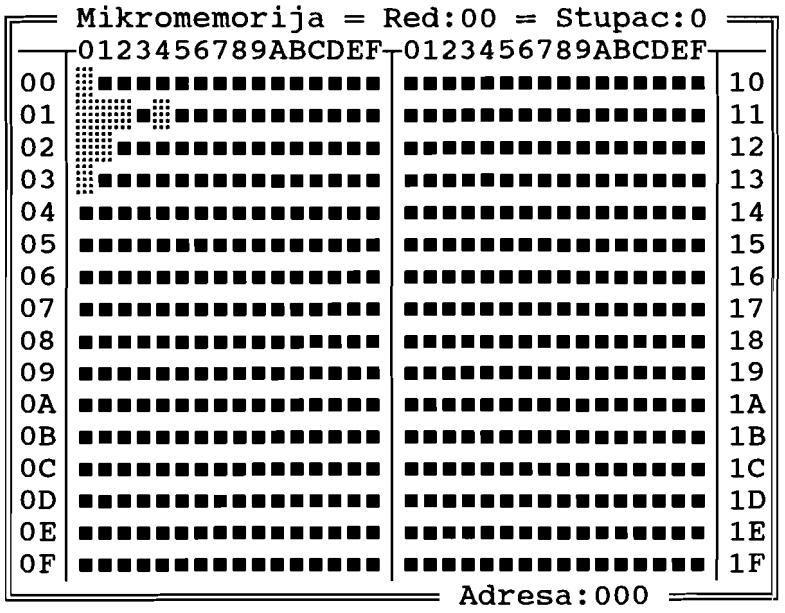



Odabirom određenog registra otvara se okvir za unos sinonima tog registra. Moguće je koristiti slova i brojeve do ukupno 3 znaka, što je u principu dovoljno za većinu uobičajenih imena registara.

*4.2.4.3. Prozor sa mikromemorijom*

U prozor sa mikromemorijom ulazi se pritiskom na tipku **ENTER** na odabranoj makronaredbi u makronaredbenom prozoru. Tada taj prozor postaje aktivan što se vidi po dvostrukoj liniji oko prozora, a makronaredbeni postaje neaktivan.

U tom prozoru prikazane su sve lokacije mikromemorije. Mikromemorija koja je organizirana kao matrica od 32 retka i 16 stupaca mikrolokacija prikazana je kao dvije polovice, jedna pored druge, a zbog nemogućnosti prikaza u jednom dijelu od 32 retka. Radi lakše orijentacije uz lijevi i desni rub prozora popisani su redni brojevi redaka, a pri vrhu prozora su redni brojevei stupaca; jedni i drugi brojevi prikazani se heksadecimalno zbog kraćeg (ali i jasnijeg nego decimalnog) zapisa. Osim toga, što god se radi po mikromemoriji u prvom retku prozora piše koja je trenutno aktivna mikrolokacija prema koordinatama retka i stupca, a u zadnjem retku prozora piše koja je to mikroadresa. Iako 9-bitna mikroadresa nastaje tako da se 5 bitova broja retka stavi na gornjih 5 bitova mikroadrese, a 4 bita radnog broja stupca stavi na najniža 4 bita mikroadrese (to se lako radi i napamet), pokazalo se korisnim imati na zaslonu takav pregledan prikaz.

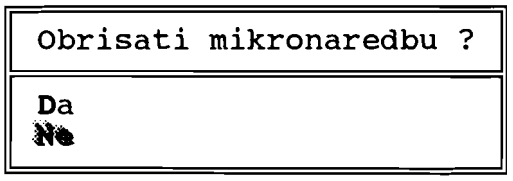



Svaka se lokacija mikromemorije prikazuje određenim simbolom po kojem se ima vizualna informacija o sadržaju. Sa desne strane prozora (izvan prozora) nalazi se i kratka "legenda" sa popisom i objašnjenjem pojedinih simbola. Znakom ■ označena je prazna lokacija, odnosno lokacija koja nije dodijeljena niti jednoj makronaredbi. Znakom ⣿ označena je puna lokacija, odnosno lokacija koja pripada nekoj makronaredbi. To su jedini simboli koji se koriste, ali oni mogu biti ispisani na različite načine. Ako je simbol pune lokacije svijetliji, to znači da ta mikrolokacija pripada trenutno aktivnoj makronaredbi u makronaredbenom prozoru odakle se i došlo u ovaj prozor. Ovi simboli mogu biti prikazani i inverzno, ali to se koristi kod definiranja skoka u mikroriječi pa će to biti objašnjeno u opisu prozora mikroriječi.

Dok je aktivan mikromemorijski prozor moguće se tipkama za smjer kretati po mikromemoriji. Trenutno aktivna mikromemorijska lokacija treperi tako da je dovoljno uočljiva, s obzirom na različite moguće ispise. Kretanjem po mikromemorijskim lokacijama u prozoru mikroriječi odmah se prikazuje detaljni prikaz te mikrolokacije u binarnom obliku, zajedno sa mnemoničkim oblikom sa desne strane tog prozora. O tome detaljnije kasnije.

Tipkom **ESC** u tom prozoru vraćamo se u prozor sa makronaredbama.

Tipkom **DEL** može se obrisati neka mikrolokacija, odnosno proglasiti je slobodnom, tj. da ne pripada niti jednoj makronaredbi. Budući da je to destruktivna operacija traži se potvrda te akcije izbornikom:




Osim toga, dozvoljeno je u tom trenutku brisati samo mikrolokacije koje pripadaju trenutno aktivnoj makronaredbi, ali da istovremeno to nije prva mikrolokacija mikroprogramske sekvence makronaredbe na koju se dolazi na osnovu vrijednosti operacijskog kóda makronaredbe. Tu se lokaciju može obrisati tako da se obriše ta makronaredba u makronaredbenom prozoru, što je i logično jer makronaredba postoji ako postoji mikrolokacija određena njenim operacijskim kódom. Pokušaj brisanje nedozvoljene lokacije otvara prozor sa upozorenjem:

```
Moguce je obrisati samo mikronaredbe tekuce
makronaredbe, osim one koju se naslovljuje
operacijskim kodom.

Pritisni bilo koju tipku
```

Pritiskom na tipku **ENTER** u tom prozoru na određenoj lokaciji odlazi se u prozor u kojem se može raditi na odabranoj mikroprogramskoj lokaciji, odnosno njenim bitovnim poljima. Dozvoljeno je raditi na mikrolokacijama koje pripadaju trenutno aktivnoj makronardbi u makronaredbenom prozoru ili na slobodnim lokacijama. Za vrijeme rada u prozoru mikroriječi, simbol odabrane lokacija u mikromemoriji treperi, tako da se na zaslonu odmah vidi sa kojom se makronaredbom i mikrolokacijom radi.

### *4.2.4.4. Prozor sa mikroprogramskom riječi*

U prozor mikroprogramske riječi dolazi se odabirom određene mikrolokacije. U tom je prozoru prikazana kompletna 32-bitna mikroriječ u binarnom obliku, a sa desne strane prozora prikazana je mikroriječ u mnemoničkom (disasembliranom) obliku koji je puno jasniji od niza bitova.

```
31═══════ Mikroprogramska rijec: 320F190B ═══════0
M|  AC6-AC0  |S|    FC3-0  |L|   F6-F0   |   K   |EaEdCsRw
1 0 9 8 7 6 5 4|3 2 1 0 9 8 7 6|5 4 3 2 1 0 9 8|7 6 5 4 3 2 1 0
0 0 1 1 0 0 1 0|0 0 0 0 1 1 1 1|0 0 0 1 1 0 0 1|0 0 0 0 1 0 1 1
```

Bit najveće težine je krajnji lijevi bit, a bit najniže težine je krajnji desni, što i piše u prvom retku prozora. U istom retku ispisan je i heksadecimalni oblik 32-bitne riječi. U trećem retku su navedeni redni brojevi bitova u skraćenom obliku. U drugom retku popisane su skraćene oznake pojedinih polja u mikroriječi, koje odgovaraju notaciji korištenoj u literaturi:

- M bit za oznaku kraja izvođenja mikroprograma;
- polje AC od 7 bitova za određivanje mikroadrese slijedeće mikronaredbe;
- S bit za dozvolu mijenjanja zastavica C i Z u MCU 3001;
- L bit za oznaku određivanja mikroadrese slijedeće mikronaredbe na osnovu operacijskog kóda upravo dobavljenje makronaredbe;
- F polje od 7 bitova za ALU operaciju koju izvode bit-odresci CPE 3002;
- bitovi K sabirnice;
- bitovi za kontrolu pristupa makromemoriji:

  EA = *Enable Address* - na adresnu sabirnicu stavlja sadržaj Memorijskog Adresnog Registra iz CPE 3002;

  ED = *Enable Data* - podatak iz akumulatora AC stavlja na izlaznu podatkovnu sabirnicu;

  CS = *Chip Select* - oznaka da ova mikroriječ nešto radi sa makromemorijom;

  R/W* = *Read/Write* - naznaka da li se radi Read ili Write u makromemoriju.

Definiranje vrijednosti pojedinih polja na binarnoj razini vrlo je mukotrpno i lako se griješi, budući da se radi gotovo svih 32 bita u više mikroprogramskih lokacija. Stalno traženje mnemonika mikrooperacija i skokova po tablicama, te prepisivanje njihovih binarnih oblika jako otežava i usporava mikroprogramiranje. Ovaj dio jedan je od nekoliko dijelova ove programske podrške koji su značajno olakšali rad na ovoj programskoj razini, i predstavljaju glavnu prednost i razlog primjene ovog programa.

U mikroriječi postoji više prije navedenih bitovnih polja, svako sa svojom namjenom. Međutim, korištenje nekih polja isključuje korištenje nekih drugih (npr. ako se mikroadresa slijedeće mikronaredbe određuje

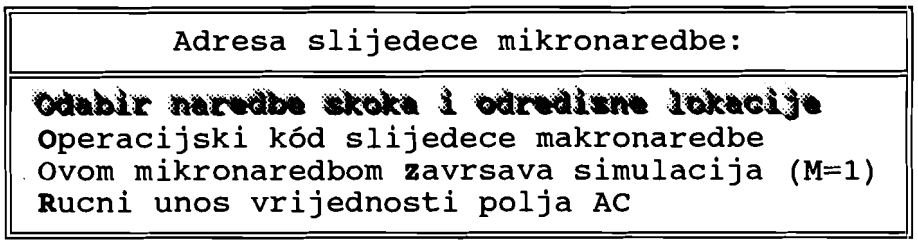
operacijskim kódom slijedeće makronaredbe, tada se L bit postavlja na 1, a polje AC se ne koristi). Takvih slučajeva ima još. To je također uzeto u obzir prilikom definiranja rada na mikroriječi.

Kroz mikroriječ se po pojedinim poljima kreće tipkama za smjer ← i →. Aktivno polje na kojem se nešto može mijenjati prikazuje se invezno. Ako se želi mijenjati sadržaj aktivnog polja potrebno je pritisnuti **ENTER** na tom polju. Tada se otvara izbornik ovisno o odabranom polju preko kojeg se definira sadržaj polja. Potrebno je uočiti da nema kretanja po svim poljima, već po poljima koje ima smisla mijenjati i koja su uopće iskorištena u mikroriječi (ima i neiskorištenih bitova). Osim toga redoslijed kretanja po poljima odgovara redoslijedu po kojem se definira cijela mikroriječ. Neka polja se, općenito, ne mijenjaju direktno, ali su određena izborom u nekom drugom polju mikroriječi (npr. K bitovi).

Iako se sva polja mogu definirati odabirom mnemonika ili značenja iz izbornika mogućih vrijednosti, gotovo se sva polja mogu unijeti i ručno u binarnom obliku. Time je ostavljen slobodan prostor za sve moguće vrijednosti svih bitovnih polja u mikroriječi.

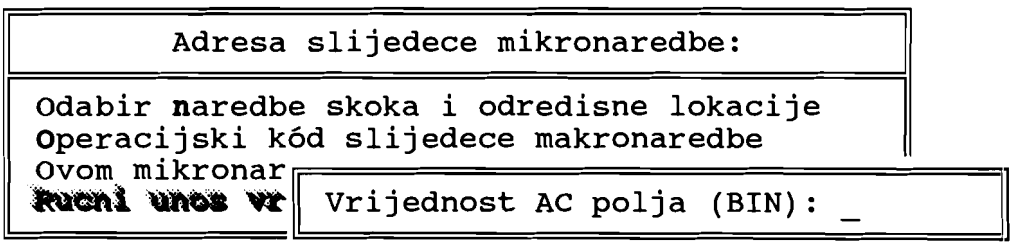
U nastavku će biti opisan rad na bitovnim poljima mikroriječi. Iako se može mijenjati bilo koje polje, najbolje je ići redom po poljima kako to vodi sam program kada se kreće tipkom u desno.

Prvo polje koje se definira, odnosno grupa polja, je određivanje adrese slijedeće mikroinstrukcije koju treba izvesti. To je moguće odrediti na nekoliko načina o čemu ovisi koja će se polja koristiti.




Ova akcija definira vrijednosti bitova M, L i AC, ali im sam program dodjeljuje vrijednosti.

Jedan od mogućih načina je da korisnik unese bitovnu vrijednost polja za skok AC koristeći okvir za unos te opcije:




Taj će se način sigurno najmanje koristiti, jer je bolje je koristiti ostale opcije. Ukoliko želimo da nam kasnije izvođenje programa stane nakon izvršenja ove mikronaredbe, u tom slučaju nema definiranja adrese slijedeće mikronaredbe. U tom slučaju odabire se opcija "Ovom mikronaredbom zavrsava simulacija (M=1)", a program postavlja M bit na 1, a bitove L i AC ostavlja na 0 jer ionako nemaju uticaja.

Ako se slijedeća mikroadresa želi odrediti operacijskim kódom nove makronaredbe (npr. na kraju mikroprogramske sekvence pribavljanja operacijskog kóda slijedeće makronaredbe) odabire se opcija "Operacijski kód slijedece makronaredbe". Tada program postavlja L bit na 1, a M i AC bitovi su na 0.

Slijedeća će se mikrolokacija ipak najčešće određivati opcijom "Odabir naredbe skoka i odredisne lokacije". Tom opcijom otvara se izbornik sa popisom naredbi skoka koje može izvršiti MCU 3001 (ne postoji apsolutni skok, već samo relativno na tekuću lokaciju):

| Naredbe skoka |
|---|
| JCC: Jump in Current Column |
| JZR: Jump in Zero Row |
| JCR: Jump in Current Row |
| JCE: Jump in Column & Enable |
| JFL: Jump & test F Latch |
| JCF: Jump & test C Flag |
| JZF: Jump & test Z Flag |
| JPR: Jump & test PR latch |
| JLL: Jump & test Left Latch |
| JRL: Jump & test Right Latch |
| JPX: Jump & test PX bus |

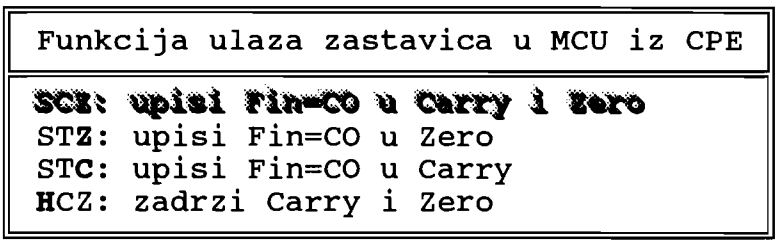
Tu su popisane sve moguće vrste skokova, iako se od toga redovito koristi tek nekoliko. Budući da korisnik cijelo vrijeme ima na zaslonu prikaz mikromemorije, on unaprijed otprilike zna gdje će mu biti slijedeća mikronaredba (npr. u istom retku, stupcu i slično). Zato se odmah i nudi takav izbornik. Međutim, potrebno je odrediti i lokaciju na koju se skače tim skokom. Nakon odabira naredbe skoka aktivira se mikromemorija tako da se dostupne lokacije s obzirom na tekuću lokaciju i odabranu naredbu skoka prikažu inverzno. Tako se vide sve lokacije na koje se može skočiti. Ako nam to ne odgovara, sa **ESC** se vraćamo u izbornik i biramo na neki drugi način slijedeću adresu.

Tipkama za smjer možemo se kretati po mikromemoriji i odabrati željenu lokaciju sa **ENTER**. Trenutna lokacija *ili lokacije* na kojoj se nalazimo treperi (ili trepere). Naime, uvjetne naredbe skoka mogu skočiti na nekoliko lokacija ovisno o uvjetu tokom izvršavanja programa. Moguće lokacije su određene fiksno i uvijek se radi o susjednih 2, 4 ili 16 lokacija. Zato se kod takvih naredbi odmah odabire više slijedećih lokacija, pa prilikom kretanja po mikromemoriji treperi 2, 4 ili 16 ciljnih lokacija. Na koju će se od njih skočiti ovisi o stanju uvjeta tokom izvršenja programa. Potrebno je napomenuti da se sve treperujuće lokacije moraju nalaziti na inverznim, tj. sve ciljne lokacije moraju spadati u dostupne lokacije, inače program ne dozvoljava odabir.

Time je određena adresa slijedeće mikronaredbe ili kraj simulacije. Slijedeće polje koje se definira je polje funkcija izlaza stanja zastavica iz spremnika u MCU na ALU operaciju u CPE. U izborniku se nalaze 4 moguće funkcije:

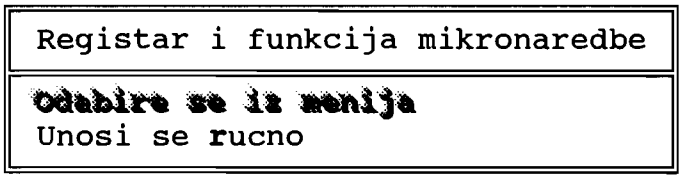
| Funkcija izlaza zastavica iz MCU u CPE |
|---|
| FF0: postavi 0 na CI=Fout |
| FFC: postavi Carry na CI=Fout |
| FFZ: postavi Zero na CI=Fout |
| FF1: postavi 1 na CI=Fout |

Ovisno o odabranoj opciji, program postavlja bitove FC3 i FC2. Slijedeće polje je polje funkcije ulaza zastavica u spremnik u MCU nakon operacije u CPE. Moguće opcije se također nude u izborniku:

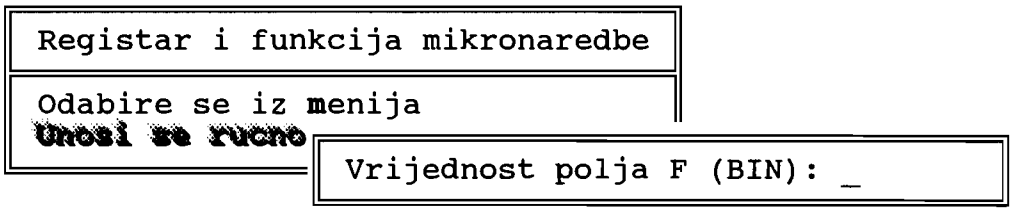



Zbog konstrukcije mikroprogramskog simulatora MP3000, potrebno je postaviti i S bit mikroriječi kada se nešto upisuje u registre zastavica u MCU, ali o tome brine sam program.

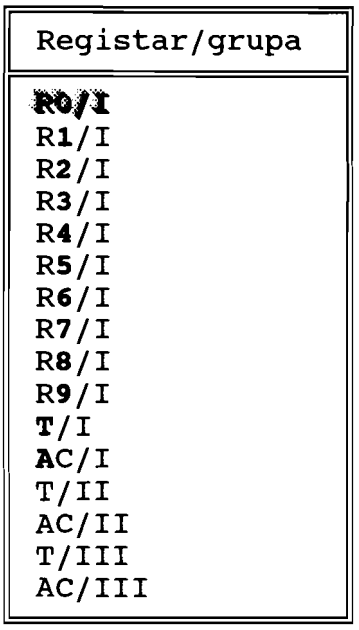
Slijedeće polje je polje operacije i registra nad kojim se vrši operacija tom makronaredbom. Mogućih mikronaredbi ima vrlo mnogo, te ih je nemoguće ovdje sve navesti i opisati. To se polje također može odrediti ručnim unosom binarne vrijednosti ili izborom iz izbornika:



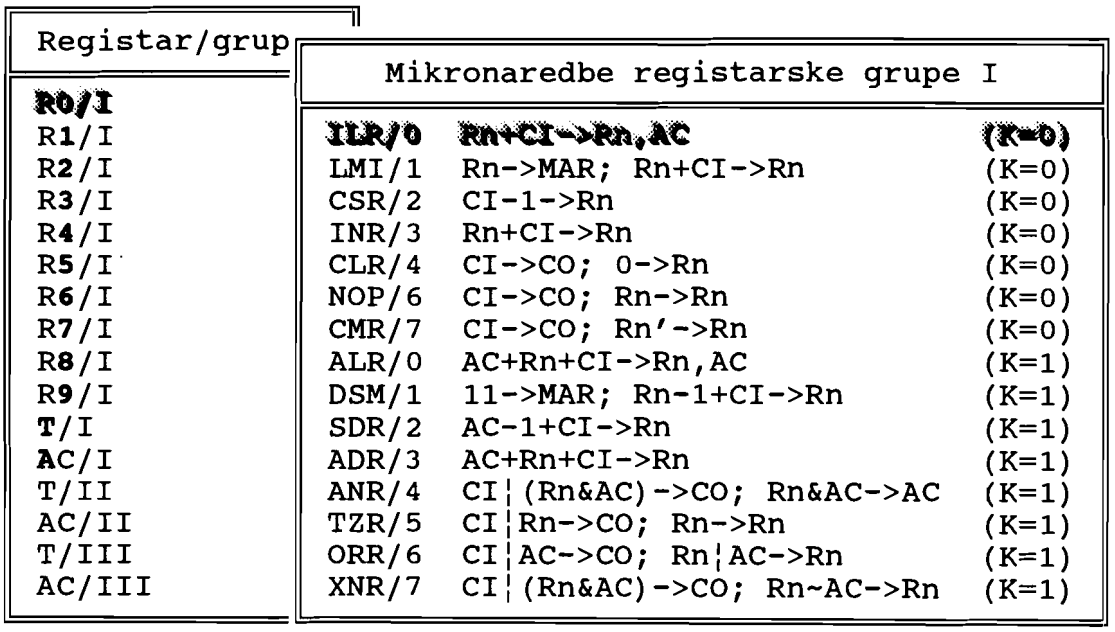

Za ručni unos otvara se okvir za unos, te se unosi željenih 7 bitova:



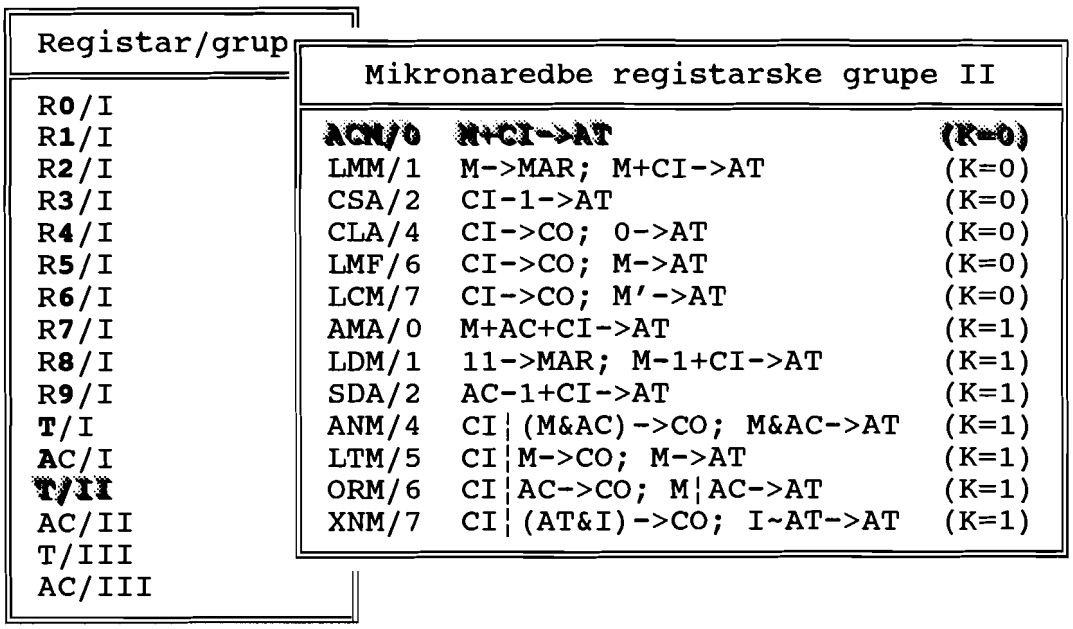

Samu mikronaredbu određuje i grupa registara nad kojim se želi izvršiti operacija, ali i stanje K sabirnice. Budući da se K sabirnica na MP3000 ne može koristiti u cijelosti, najčešće se koriste posebni slučajevi kada su svi bitovi K sabirnice 0 ili su svi 1. U programu se radi tako da se najprije odabere registar nad kojim se radi operacija (čime se određuje i registarska grupa):



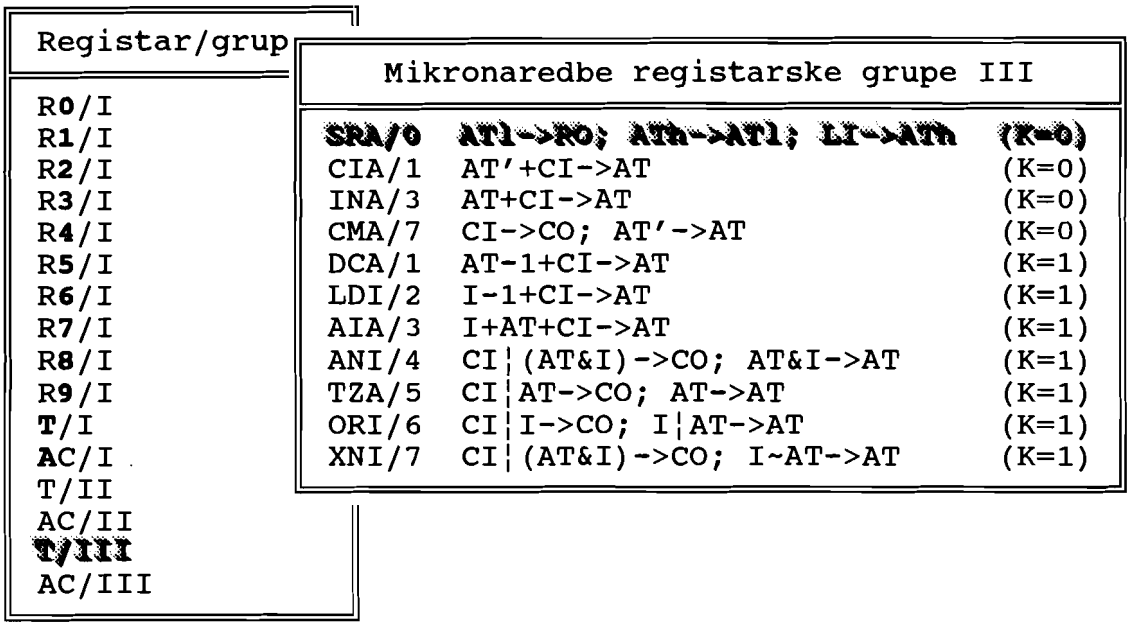

Nakon odabira registra i registarske grupe otvara se izbornik sa onim makronaredbama koje su na raspolaganju za tu registarsku grupu, i to za oba moguća stanja K sabirnice. Prikazuje se mnemonik mikronaredbe i funkcijska grupa kojoj pripada (to je uobičajena notacija), logički opis mikrofunkcije koju obavlja CPE, te vrijednost K sabirnice za tu operaciju. Objašnjenje svih operacija i notacija logičkog opisa funkcije može se potražiti u odgovarajućoj literaturi. Za prvu registarsku grupu moguć je slijedeći izbor:




Druga registarska grupa dozvoljava slijedeće operacije:

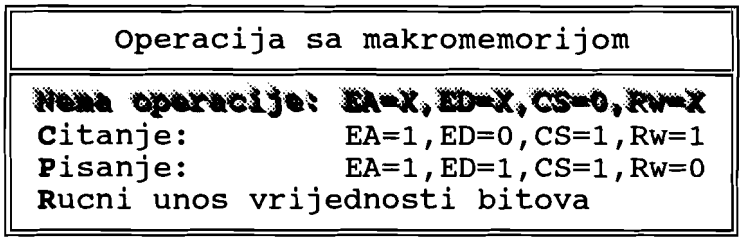



I operacije treće registarske grupe:

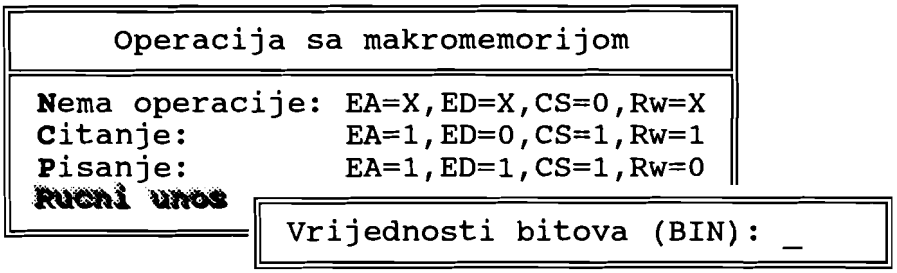



Odabirom određenog registra i makronaredbe program upisuje odgovarajući binarni kód u polja F i K.

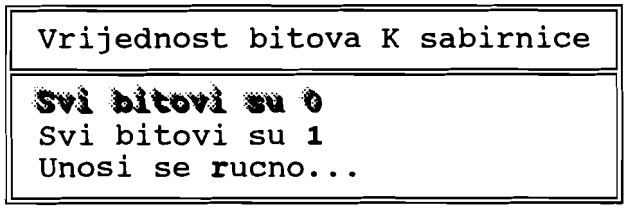
Posljednje polje koje se podešava je grupa bitova za rad sa makromemorijom. Tu se koriste čak 4 bita za samo tri moguća stanja: čitanje iz makromemorije, pisanje u nju ili nikakva operacija sa mikromemorijom. Za ovo se moglo potrošiti samo 2 bita, ali to je stvar konstrukcije MP3000 i ne može se mijenjati. Zato postoji izbornik sa navedenim mogućim kombinacijama, te eventualnim ručnim unosom:



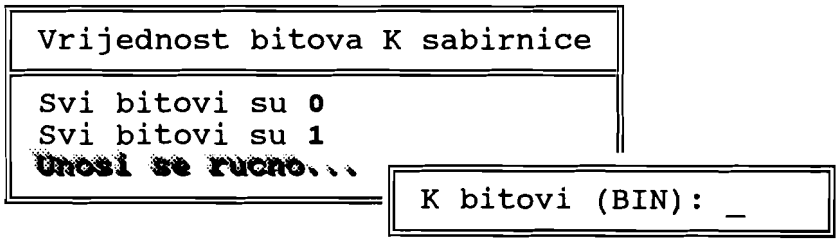

Sa ručnim unosom treba znati što se radi jer je to direktno vezano na konstrukciju MP3000:



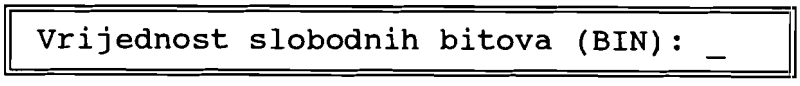

To su bila sva polja koja treba podešavati. Daljnjim kretanjem u desno skače se na polje K sabirnice (koje je u biti lijevo od posljednje aktivnog polja !), čija se vrijednost može posebno birati ako nismo zadovoljni kako je to napravio program nakon odabira makronaredbe:




Vrijednost K sabirnice može se unijeti i ručno, ali treba voditi računa o specifično povezanim bitovima u konstrukciji MP3000, što bitno ograničava određeniju upotrebu:




Preostalo je još jedno polje od 3 bita koje nema namjenu, ali se jednim okvirom omogućilo da se postave i vrijednosti i tih bitova, radi možda buduće prepravke i predefinicije mikroriječi:



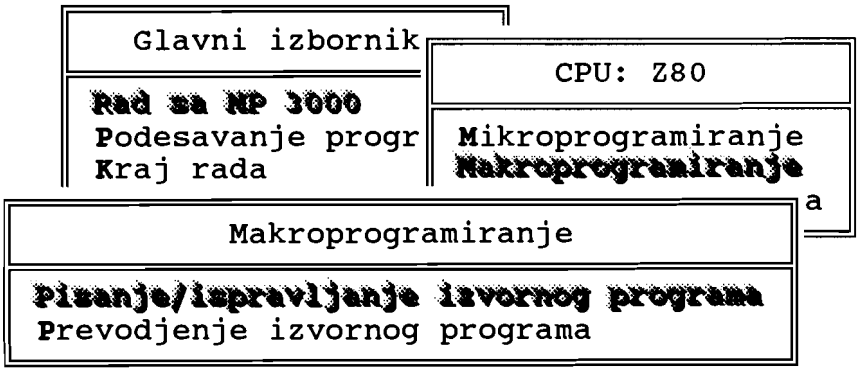

To su bila sva moguća podešavanja bitova u mikroriječi odabrane mikrolokacije. To je ono što je u cijeloj ovoj materiji i razvojnom sustavu najsloženije i najdetaljnije, i što zahtjeva najviše vremena. Vjerojatno će u početku treba malo vremena u snalaženju po poljima mikroriječi, ali nakon nekoliko definiranih, to postaje stvar

rutine.

Iz prozora mikroriječi izlazi se uobičajenom tipkom ESC i vraća se u prozor mikromemorije, gdje se odabire neka druga lokacija tekuće makronaredbe, i tako za svaku mikronaredbu, te makronaredbu.

### *4.2.5. Makroprogramiranje*

Makroprogramiranje kao viša programska razina slijedi nakon definiranja mnemonika makronaredbi i njihovih mikroprogramskih sekvenci u modulu mikroprogramiranja. Makroprogramiranje se odabire u izborniku mikroprocesora, a time se otvara novi izbornik:




Jednom se opcijom poziva program za pisanje izvornog kóda makroprograma (ASCII editorom prijavljenim u opciji "Podešavanje programa" glavnog izbornika), a drugom se napisani program prevodi u binarni oblik.

#### *4.2.5.1. Pisanje i ispravljanje izvornog programa*

Ova opcija prije poziva editora traži ime izvornog programa:


Ime programa predstavlja osnovno ime datoteke kojem program dodaje nastavak *.ASS i sa tim imenom poziva editor. Dozvoljeno je koristiti slova i brojeve u imenu koje može biti dugo do 8 znakova. Da se kod višestrukih poziva editora i asemblera ne mora uvijek unositi ime datoteke, program svaki put ponudi ime koje je korišteno prilikom zadnjeg poziva opcije, te je dovoljno pritisnuti samo **ENTER**.

Pravila pisanja asemblerskih makroprograma su uobičajena, a opisana su u nastavku. Svaki se redak makroasemblerskog programa sastoji od nekoliko polja. Ako se na početku retka nalazi znak ';' cijeli se redak smatra komentarom. Na početku retka može se navesti labela koja će poprimiti vrijednost adrese makronaredbe

prije koje se nalazi ili vrijednosti iza pseudonaredbe EQU. Ako nema labele u retku, na početku retka mora biti najmanje jedan razmak ili tabulator. Slijedeće polje je polje mnemonika makronaredbe ili asemblerske pseudonaredbe. Prije tog polja mora biti najmanje jedan razmak ili tabulator od početka retka ili nakon labele, ako ona postoji na početku tog retka. Koje pseudonaredbe asembler podržava opisati će se kasnije, a ovdje treba reći da prilikom pisanja makronaredbi definiranih prilikom mikroprogramiranja treba paziti da se one pišu onako kako je to opisano, pazeći na pozicije razmaka i podatkovnih oznaka.

Asembler podržava slijedeće pseudonaredbe (komande asembleru koje se ne prevode u binarni kód):

EQU - definiranje vrijednosti labeli sa početka retka; na početku tog retka nužna je labela, a iza EQU numerička vrijednost ili neka do tada već poznata labela; sve se labele pamte na 32 bita, a u makromemoriju pohranjuju se ovisno o kontekstu (broj podatkovnih oznaka u makronaredbi, pseudonaredbe DEFB, DEFW ili DEFL);

ORG - određivanje nove početne adrese za smještaj binarnog kóda u makromemoriju; iza ORG potreban je numerički podatak na 8 bita koji predstavlja novu adresu, s tim da ne smije biti manja od trenutne vrijednosti adrese na koju se smješta kód programa;

DEFB - pohranivanje jednog 8-bitnog podatka navedenog iza te pseudonaredbe na tekuću makromemorijsku lokaciju; ako vrijednost veća od 8 bitova, pohranjuje se najnižih 8;
DEFW - pohranjivaje jednog 16-bitnog podatka; ako je vrijednost veća, pohranjuje se najnižih 16 bitova;
DEFL - pohranjivanje jednog 32-bitnog podatka;
BEND - od tog retka pa nadalje višeoktetne podatke pohranjivati po BigEndian formatu;
LEND - od tog retka pa nadalje višeoktetne podatke pohranjivati po LittleEndian formatu;

Makronaredbe i pseudonaredbe mogu se pisati bilo malim, bilo velikim slovima. Numerički podatak kao argument nekih pseudonaredbi ili podatak u makronaredbi na poziciji podatkovne oznake može biti:

- decimalan broj (npr. 155, 10256 i sl. do 32 bita);
- heksadecimalni broj sa definiranim heksadecimalnim predznakom (npr. #5e, #a5ed);
- definirana labela kao simboličko ime adrese ili dodjeljena vrijednost sa EQU;

Sve ove vrste podataka mogu imati negativna predznak '-', kad se računa drugi komplement (na 32 bita !). Trenutno asembler ne podržava složenije izraze u kojima se koriste različiti aritmetičko-logički operatori.

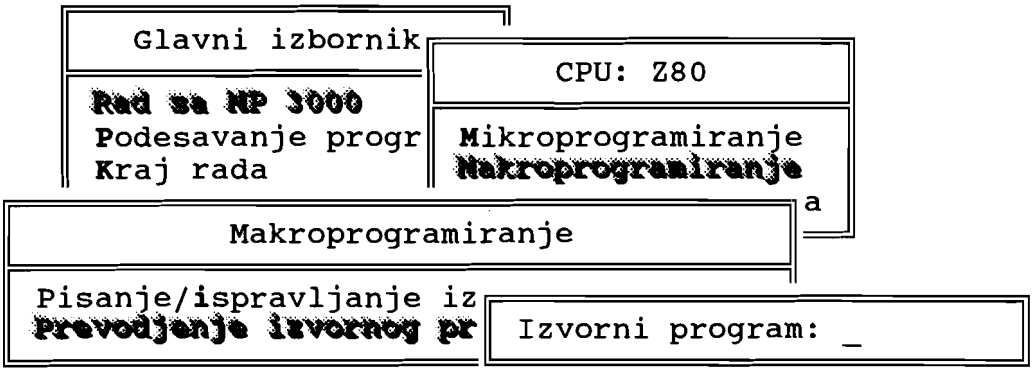
Evo nekoliko primjera ispravnih asemblerskih redaka (koriste se makronarebe primjera):

```
        ORG 10
; ovo je komentar
NUM     EQU  -#A4
BEGIN   LD A,NUM
        LD (DATA),A
        LEND
DATA    DEFB -10
        DEFW #FFFF
```

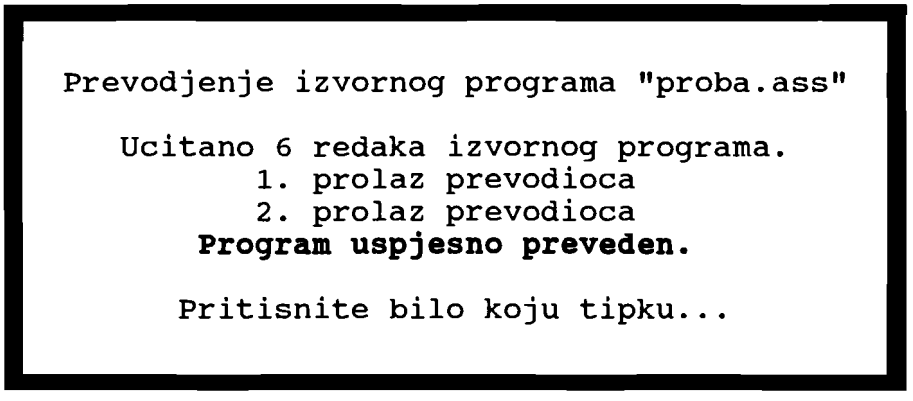
Postoje i određena ograničenja. Prilikom pisanja programa treba imati na umu da se program prevodi u 8-bitnu makromemoriju sa 256 lokacija, pa treba u skladu s tim imati i odgovarajuće makronaredbe. Labela može biti duga do 16 znakova, mogu je sačinjavati slova i brojevi, ali prvi znak mora biti slovo. I labele se mogu pisati malim ili velikim slovima, i to se ne razlikuje. Cijeli makroasemblerski program može biti dug do najviše 512 redaka, s time da svaki redak može biti dug najviše 128 znakova.

*4.2.5.2. Prevođenje makroasemblerskog programa*

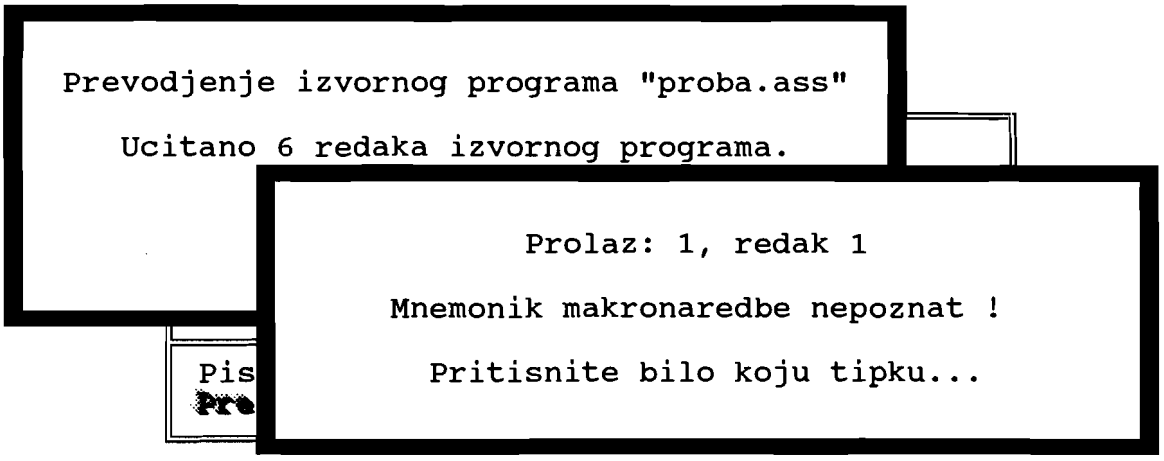
Prevođenje programa radi se drugom opcijom u izborniku makroprogramiranja. Odabirom te opcije potreno je zadati ime izvornog programa koji se želi prevesti u binarni oblik (asemblirati), a koji je napisan u editoru korištenjem prethodne opcije. Budući da se to može uzastopno pozivati više puta, u okviru za unos pamti se zadnje uneseno ime programa, pa je idući put dovoljno pritisnuti samo **ENTER** ako se radi o istom programu.




Prevođenje programa radi se u dva prolaza. U prvom se provjerava ispravnost sintakse makronaredbi i stvara popis definiranih labela i njihovih vrijednosti, a u drugom se stvara binarni kód programa. U prozoru se za to vrijeme ispisuje što program trenutno radi: učitan izvorni program, početak prvog prolaza asemblera, početak drugog prolaza asemblera, te kraj prevođenja. Ako se cijeli program prevede bez greške njegov se prevedeni kód sprema u datoteku istog osnovnog imena kao izvorni program, ali sa nastavkom *.OBJ, a popis labela i njihovih vrijednosti u datoteku *.LAB. Obje datoteke su u ASCII obliku i mogu se pogledati ASCII editorom. Uspješno prevedeni program može se izvoditi, što se opisuje u slijedećem poglavlju.




Ukoliko tokom prevođenja izvornog programa asembler naiđe na neku grešku, otvara se novi prozor u kojem se prolazu i retku izvornog programa naišlo na grešku, te o kojoj se greški radi. Npr:




U slučaju greške potrebno je vratiti se u editor i izvršiti potrebne ispravke, te ponovo pokrenuti prevođenje programa. U nastavku su navedene sve greške koje asembler može navesti sa kratki objašnjenjem problema.

"Nema datoteke sa izvornim programom !" - datoteka sa zadanim osnovnim imenom, a nastavkom *.ASS nije nađena; vjerojatno greška u navođenju imena izvornog programa.

"Nije definiran niti jedan mnemonik !" - pokušaj prevođenja izvornog programa, a da se u modulu mikroprogramiranja nije još definirao niti jedan mnemonik makronaredbi odabranog mikroprocesora; ako nije greška u zadavanju imena mikroprocesora, treba stvarno nešto prije definirati u mikroprogramskom modulu, a tek onda pisati makroprograme.

"Predugi redak u izvornom programu !" - redak izvornog programa duži je od najviše dozvoljenih 128 znakova; ako se namjerno nije otkucao tako dugi redak, vjerojatno je učitana datoteka koja nije izvorni program za ovaj asembler.

"Previse redaka u izvornom programu !" - izvorni program može imati najviše 512 redaka.

"Datoteka nije ASCII !" - datoteka *.ASS koja se pokušava učitati uopće nije ASCII, već binarna iz nepoznatih razloga.

"Predugo ime za labelu !" - ime labele smije biti najviše 16 znakova.

"Nedozvoljeni znak u labeli !" - ime labele može činiti slovo, broj ili '_', s time da mora započinjati slovom.

"Labela mora zapoceti sa slovom !" - vjerojatno je neki mnemonik krivo otkucan, pa se dio naredbe protumačio kao pogreška u labeli.

"Ocekuje se vrijednost ili izraz !" - iza nekih pseudonaredbi asemblera potrebno je navesti određenu numeričku vrijednost koja je izostala, odnosno na mjestu gdje se u opisu mnemonika stavila oznaka podatka nije se naišlo na podatak.

"Novi ORG je manji od tekuceg PC !" - nova vrijednost adrese za pohranjivanje kóda programa mora biti jednaka ili veća od tekuće adrese pohranjivanja, inače bi se novi kód prepisao preko postojećeg.

"Najveca dozvoljena adresa je 255 !" - dozvoljene vrijednosti nove adrese za pohranjivanje daljnjeg kóda mora biti u granicama $0 \leq adr \leq 255$ jer je makromemorija 8-bitna.

"EQU redak mora zapocinjati labelom !" - u retku sa pseudonaredbom EQU mora biti navedena labela kojoj se dodjeljuje vrijednost iza EQU.

"Labela je vec definirana !" - na početku retka, dakle za definiranje vrijednosti, se neka labela smije pojaviti samo jednom.

"Nedefinirana labela !" - u drugom prolazu pokušava se naći numerička vrijednost labele koja nije definirana.

"Mnemonik makronaredbe nepoznat !" - napisani mnemonik nije prema pravilima opisa uspješno uspoređen ni sa jednim definiranim mnemonikom u mikroprogramskom modulu.

"Greska u aritmeticko-logickom izrazu !" - na mjestu gdje se očekuje numerički podatak nije prepoznat ni decimalan, ni heksadecimalan broj niti labela. Moguća je greška u opisu ili u korištenju makronaredbe.

"Labela mora biti definirana !" - pseudonaredbe ORG i EQU mogu kao argumente imati i labele, ali one do retka sa ORG ili EQU u kojem se koriste moraju biti već definirane. Za ostale pseudonaredbe i podatke u makronaredbama to nije nužno.

### *4.2.6. Izvođenje programa*

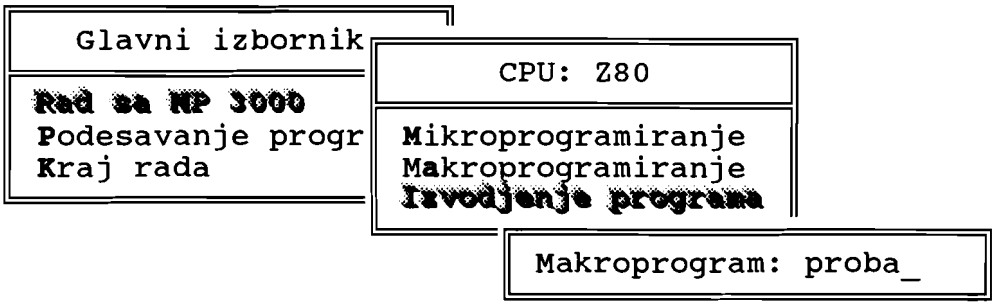
Izvođenje programa predstavlja vjerojatno najzanimljiviji dio cijelog programa jer se tu tek vidi što i kako rade napisani mikro i makro programi. Programe se može izvoditi korak po korak ili izvesti odmah cijeli program. Korak po korak može se izvoditi i mikronaredbu po mikronaredbu i makronaredbu po makronaredbu, uz dinamičko praćenje promjena u registrima i na sabirnicama.

Za izvođenje programa nužno je imati priključeno sklopovlje. Ukoliko je program pokrenut tako da radi bez sklopovlja, a pokuša se pokrenuti ova opcija ispisati će se upozorenje:

```
Nema prikljucenog sklopovlja !

Za izvodjenje programa nuzno je imati prikljucen
    mikroprogramski simulator MP3000.

Prikljucite simulator i pokrenite program
        bez opcije /nohw.

Pritisni bilo koju tipku
```

U tom je slučaju potrebno postupiti prema obavijesti u prozoru: izaći iz cijelog programa i ponovo ga pokrenuti bez opcije **/nohw**. Ako je sklopovlje stvarno priključeno, program će uspostaviti komunikaciju i dozvoliti pokretanje ove opcije.

U izvođenje programa se kreće nakon uspješno prevedenog makroprograma napisanog korištenjem makronaredbi definiranih prilikom mikroprogramiranja opcijom "Izvodjenje programa" u izborniku za odabrani mikroprocesor. Odabirom te opcije traži se da se unese osnovno ime izvornog programa (ono koje je korišteno prilikom pisanja i prevođenja izvornog programa):




Program nakon unosa imena pokušava naći datoteku sa tim osnovnim imenom i nastavkom *.OBJ, a koju je stvorio asembler. Ukoliko je ne nađe ispisuje obavijest u prozoru:

```
Nema prevedenog programa "PROBA.OBJ" !

Pritisni bilo koju tipku
```

Može biti problem u tome da se krivo navelo ime makroprograma ili pak napisani program još nije preveden asemblerom.

Kad program nađe potrebnu datoteku sa binarnim kódom prevedenog programa otvara se radni zaslon izvođenja programa, odnosno okoline u kojoj se izvodi i prati izvođenje mikro i makro programa. Program najprije međusklopu koji upravlja mikroprogramskim simulatorom pošalje sadržaj makroprogramske i mikroprogramske memorije. Budući da se radi o slanju 9 paketa podataka po 256 okteta, to traje nekoliko sekundi (ovisno o podešenoj brzini serijske komunikacije), a obavijest o tome vidi se u prozoru:

```
Prebacivanje mikro i makro
programa u MP3000

Pricekajte
```

Već ovdje, a i bilo kada tokom rada u ovom modulu može doći do problema u komunikaciji, budući da se gotovo sve komande izvršavaju na priključenom sklopovlju, tako da je komunikacija prilično česta, iako ne dugotrajna. Ukoliko dođe do nekih problema, program ispisuje obavijest:

```
Problemi u komunikaciji sa MP3000 !

Otklonite problem, resetirajte MP3000, te
ponovo pokrenite Izvodjenje programa.

Pritisni bilo koju tipku
```

Nakon toga program prekida rad sa ovim modulom i vraća se u izbornik mikroprocesora. Korisnik može pokušati otkloniti problem u komunikaciji, resetirati priključeno sklopovlje RESET tipkom, te ponovo pokrenuti opciju "Izvođenje programa". Program u tom slučaju "zna" da li je kod zadnjeg korištenja te opcije bilo problema, te još jednom podsjeća što treba učiniti:

```
Bilo je problema u komunikaciji sa MP3000 !

Otklonite problem i resetirajte MP3000.

Pritisni bilo koju tipku
```

Nakon te obavijesti program ponovo pokušava uspostaviti sinkonizaciju sa sklopovljem koja je analogna kao kod pokretanja cijelog programa. Ako sinkronizacija uspije, program nastavlja rad na izvođenju programa.

U okolini za izvođenje programa postoje tri prozora: prozor makromemorije u kojem se vidi njen sadržaj, prozor mikromemorije u kojoj se vide slobodne i zauzete lokacije, te lokaciju mikronaredbe koja se upravo treba izvesti i njen disasemblirani prikaz sa desne strane tog prozora, te treći prozor koji sadrži prikaz stanja svih registara, zastavica i sabirnica.

Potrebno je uočiti da su svi prozori istovremeno aktivni, za razliku od hijerarhijskog načina rada po prozorima prilikom mikroprogramiranja. To je takav način rada: postoji niz komandi koje se sve mogu pozivati direktno i u bilo kojem trenutku. U nastavku će biti podrobnije opisan svaki prozor i komande koje se odnose na njegovu namjenu. Budući da je na raspolaganju čak 17 funkcijskih tipki, vrlo je teško pamtiti koja čemu služi, a posljednji redak na zaslonu premali je za cijeli popis. Zato je cijelo vrijeme na raspolaganju tipka F1 koja u velikom prozoru ispisuje sve raspoložive tipke i njihovu namjenu:

```
                Raspolozive tipke u debugger-u

Makromemorija                  Mikronaredbe
F2-nacin ispisa                F5-izvrsi jednu
Shift-F2-pocetna adresa        Shift-F5-izvrsi n
Ctrl-F2-citanje iz MP3000      Alt-F5-zadavanje n
Alt-F2-mijenjanje sadrzaja     Ctrl-F5-mikroadresa

Registri                       Izvrsavanje makronaredbi
F9-mijenjanje sadrzaja         F6-izvrsi jednu
Ctrl-F9-citanje iz MP3000      Shift-F6-izvrsi n
Shift-F9-zastavice             Alt-F6-zadavanje n
Alt-F9-AC & T                  Ctrl-F6-zadavanje PRIBAVI
                               F7-pokretanje programa

                Pritisni bilo koju tipku
```

Tipkom **ESC** može se u bilo kojem trenutku izaći iz modula izvođenja programa u izbornik mikroprocesora.

*4.2.6.1. Prozor sa sadržajem makromemorije*

U prozoru sa sadržajem makromemorije prikazuje se binarni kód programa na različite načine. Radi kraćeg i jednostavnijeg ispisa, svi se brojevi prikazuju u heksadecimalnom obliku (na to se upozorava u prvom retku prozora), bez ispisivanja predznaka heksadecimalnih brojeva.

Sadržaj makromemorije moguće je prikazivati u **disasembliranom obliku**, odnosno mnemoničkom, pri čemu program na osnovu operacijskog kóda makronaredbe nalazi opis mnemonika i prikazuje ga u tekstualnom obliku umećući eventualni podatak uz tu naredbu na odgovarajuće mjesto u ispisu makronaredbe. Primjerice, to može izledati ovako:

```
Makromemorija (HEX) - (PC )
00:010A       LD A,0A
02:3114       LD B,14
04:61         ADD A,B
05:0210       LD (10),A
07:FF         STOP
08:00         (nepoznato)
09:00         (nepoznato)
0A:00         (nepoznato)
0B:00         (nepoznato)
0C:00         (nepoznato)
0D:00         (nepoznato)
0E:00         (nepoznato)
0F:00         (nepoznato)
10:00         (nepoznato)
11:00         (nepoznato)
12:00         (nepoznato)
13:00         (nepoznato)
```

Iz ovog primjera vidljiv je način prikaza: na početku svakog retka u prozoru je adresa na kojoj se nalazi operacijski kód makronaredbe, zatim slijedi binarni kód makronaredbe, a desno je mnemonički oblik makronaredbe ako je prepoznat operacijski kód na toj adresi. Budući da cijela makronaredba zajedno sa podatkovni dijelom može zauzimati i više okteta, a zbog ograničene širine ispisa, ispisuje se do 4 okteta u heksadecimalnom obliku, a ako ih ima više od 4, iza zadnjeg okteta dopisuje se znak '»' kako bi se korisniku skrenula pažnja da ima još okteta koji pripadaju makronaredbi, a nisu prikazani. Analogno vrijedi i za mnemonički oblik na desnoj strani prozora gdje se prikazuje do 13 znakova prevedenog oblika, ako ih ima još dopisuje se također znak '»'. Primjer takvog retka izgleda ovakao:

```
00:01112233» LD A,(1122334»
```

...dok bi cijeli redak izgledao ovako:

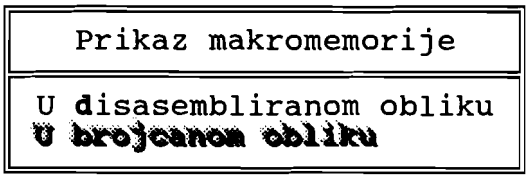
```
00:0111223344 LD A,(11223344)
```

Ovakve će se široke makronaredbe vjerojatko rijetko javljati, a njih će se lako prepoznati iako se ne vidi cijeli oblik.

Adresa na početku slijedećeg retka je adresa iz prethodnog retka uvećana da dužinu binarnog kóda prethodne makronaredbe. Iako se mogu prikazati u cijelosti, duge makronaredbe pravilno se analiziraju i nalazi se operacijski kód slijedeće makronaredbe.

Ukoliko operacijski kód nije prepoznat, program ispisuje tekst "(nepoznato)" i preskače samo jedan oktet.

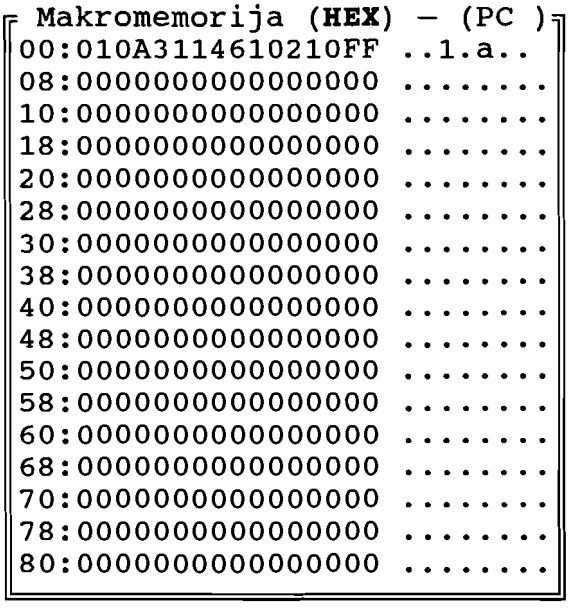
Drugi način prikaza makromemorije je **brojčano/znakovni**. Između ovog i prethodnog načina prikaza može se birati pritiskom na tipku **F2** čime se otvara izbornik:



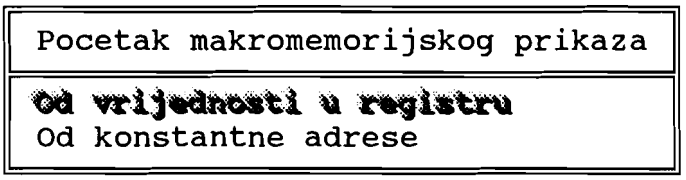

Brojčano znakovni prikaz vidi se na slijedećoj slici.

Na početku svakog retka je adresa grupe od 8 okteta, zatim slijedi heksadecimalni prikaz tih 8 uzastopnih okteta, a iza toga je znakovni prikaz tih 8 okteta prema ASCII standardu. Zbog problema sa ispisom na zaslonu, svi se kódovi sa brojem manjim od 32 prikazuju sa '.'.

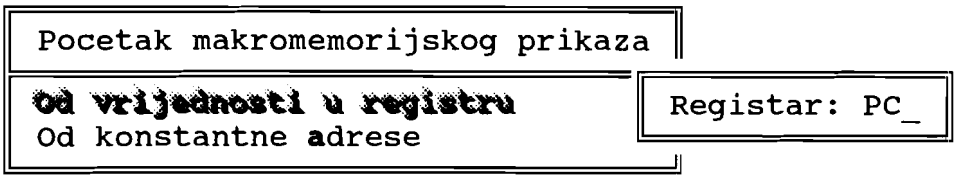
Daljnje podešavanje odnosi se na to od koje se adrese se prikazuje sadržaj makromemorije. To može biti i vrijednost u nekom registru. U prethodnih prikazima vidi se da u prvom retku prozora stoji (**PC** ) što upućuje na to da prikaz makromemorije počinje od adrese u registru sa sinonimom PC, a to je u ovom slučaju R9. Registar može biti bilo koji, a ono što je posebno značajno je da se prikaz automatski mijenja sa promjenom vrijednosti u zadanom registru. To je posebno zgodno za prikaz od tekuće vrijenosti u registru kojem smo namijenili ulogu programskog brojila (kao u prikazanom primjeru).



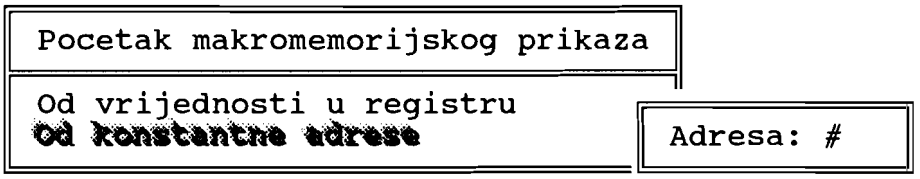

Adresa od koje se prikazuje makromemorija određuje se izbornikom koji se prikazuje pritiskom na **Shift-F2**:




Za odabranu opciju prikazivanja od vrijednosti u registru, otvara se okvir za unos imena registra od čije se vrijednosti prikazuje sadržaj makromemorije:




Drugom opcijom otvara se okvir za unos konstantne adrese od koje se vrši prikaz:




Takav nam je prikaz prikladniji u slučaju da želimo trajno nadgledati isti dio makromemorije tokom izvođenja programa. Za to vrijeme u prvom retku prozora će umjesto registra biti ispisana ta adresa.

Potrebno je napomenuti da svi ovi prikazi makromemorije od određene adrese nemaju veze sa makro ili mikro naredbom koja se upravo treba izvesti. Prikaz makromemorije potpuno je neovisan o izvođenju programa.

Pritiskom na **Alt-F2** moguće je mijenjati sadržaj određene lokacije makromemorije. Otvara se okvir za unos adrese u heksadecimalnom obliku:


Nakon unosa adrese otvara se okvir za unos 8-bitne vrijednosti koju treba staviti na tu lokaciju (ponuđena vrijednost je trenutna vrijednost na toj adresi):


Načinjena promjena odmah se prenosi i u makromemoriju na međusklopu mikroprogramskog simulatora, a obnavlja se i prikaz na zaslonu.

U prikazu makromemorije na raspolaganju nam je još i **Ctrl-F2** čime se programu kazuje da ponovo pročita sadržaj makromemorije sa samog međusklopa uz mikroprogramski simulator u memoriju programa na osobnom računalu. Ovo je načinjeno više radi formalnosti, jer sadržaj makromemorije na mikroprogramskom simulatoru općenito odgovara sadržaju makromemorije na osobnom računalu. Program na međusklopu koji izvodi simulaciju pazi na to da li mikro i makro program nešto upisuju u makromemoriju, te ako se to desi, nakon izvršene programske sekvence o se toj promjeni obavještava korisnika i nudi ponovno čitanje sadržaja, što se ne mora prihvatiti. O tome kod izvršavanja naredbi.

Svi se ovi načini prikaza pamte u konfiguracijskoj datoteci programa, pa se automatski postavljaju prilikom slijedećih izvođenjá programa.

#### *4.2.6.2. Prozor sa mikromemorijom*

Prozor sa mikromemorijom nema neku posebnu funkciju, već se u njemu samo vidi koje su lokacije slobodne, a koje zauzete (isti su simboli kao kod mikroprogramiranja). Lokacija sa koje se će se izvršiti

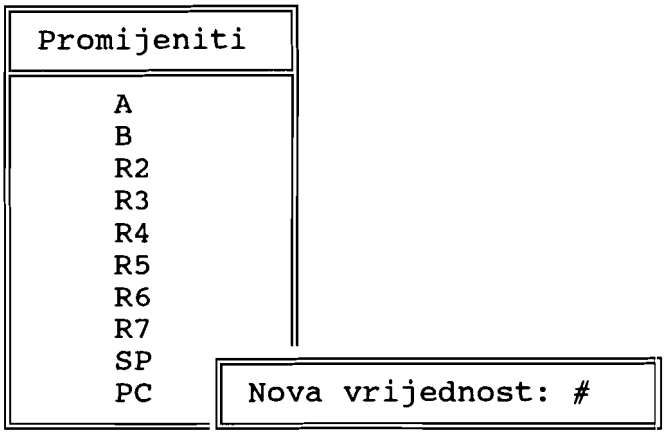
slijedeća mikronaredba je osvijetljena, a sa desne strane prozora dan je kompletan disasemblirani prikaz te tekuće mikronaredbe. Tekuća mikronaredba je ona koja je adresirana vrijednošću na mikroadresnoj sabirnici MCU 3001, a ta se vrijednost vidi u prozoru sa registrima, zastavicama i sabirnicama (označeno sa **MiA**).

*4.2.6.3. Prozor sa registrima, zastavicama i sabirnicama*

U ovom se prozoru vide sadržaji svih registara, zastavica i stanja na sabirnicama. Sve su vrijednosti prikazane u decimalnom i heksadecimalnom obliku. Ako postoji sinonim registra, on je prikazan na zaslonu umjesto originalnog imena R?.

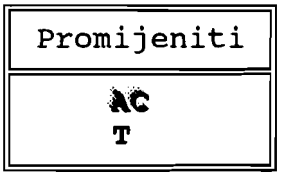
| Registri | | | | | Sabirnice |
|---|---|---|---|---|---|
| A =000 #00 | R3=000 #00 | R6=000 #00 | PC=000 #00 | | MiA=000 #000 |
| B =000 #00 | R4=000 #00 | R7=000 #00 | AC=000 #00 | C=0 | DB =000 #00 |
| R2=000 #00 | R5=000 #00 | SP=000 #00 | T =000 #00 | Z=0 | MaA=000 #00 |

Tpkom **F9** otvara se izbornik sa popisom registara (ako postoji, ispisuje se sinonim) čija se vrijednost može promijeniti. Odabirom određenog registra otvara se okvir za unos nove vrijednosti:

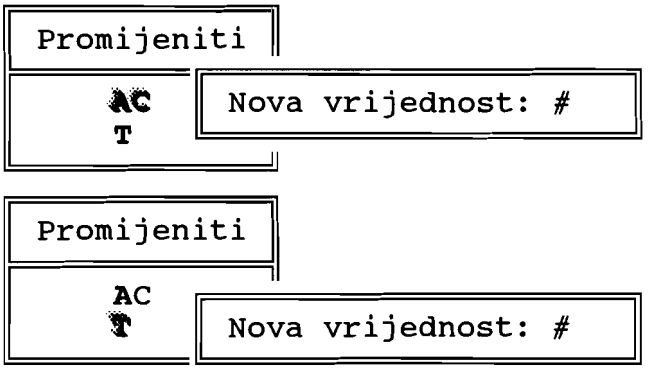



Nova se vrijednost šalje mikroprogramskom simulatoru, a na zaslonu se ispisuje novo stanje koje se uvijek čita iz MP3000.

Pritiskom na **Alt-F9** pojavljuje se izbornik sa registrima AC i T:

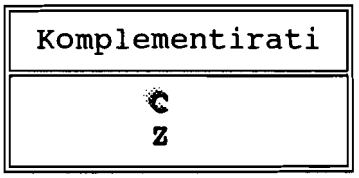



Odabirom jednog od registara pojavljuje se okvir za unos nove vrijednosti:




Unosi se heksadecimalna vrijednost (naglašen je odgovarajući predznak), a unesena vrijednost se odmah šalje na međusklop u postavlja u registar CPE. Novo stanje na zaslonu se postavlja nakon pročitanog novog stanja iz registara.

Nova vrijednost zastavica postavlja se sa **Shift-F9** čime se otvara izbornik:




Treba uočiti da se vrijednost obarane zastavice ne postavlja, već samo komplementira. Time je malo skraćen rad (nema unosa i onda tipka ENTER), jer ako se želi postaviti neka druga vrijednost zastavice od one koja je trenutno, to je upravo komplement.

Pritiskom na **Ctrl-F9** mogu se ponovo pročitati stanja svih registara iz MP3000.

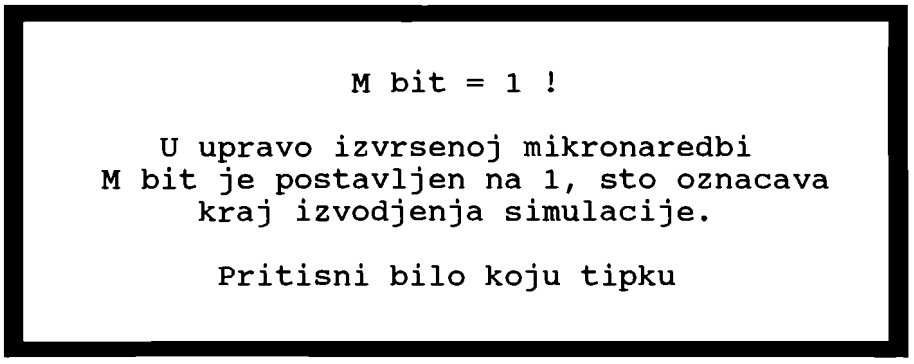
Na desnoj strani prozora prikazana su stanja na mikroadresnoj sabirnici MCU 3001 (MiA), izlaznoj podatkovnoj sabirnici CPE 3002 (DB) makroadresnoj sabirnici CPE 3002 (MaA). Njihovo se stanje čita čitanjem stanja registara, a to je nakon promjene sadržaja, nakon izvršavanja mikro ili makro naredbe ili direktnim čitanjem sa Ctrl-F9. Zato i ne postoji posebna komanda za čitanje stanja na sabirnicama.

#### *4.2.6.4. Izvršavanje mikro i makro naredbi*

Izvršavanje mikro i makro naredbi osnovna je funkcija ovog modula. To se može raditi na nekoliko načina, čime se pokušalo što više olakšati rad na tako niskoj programskoj razini.

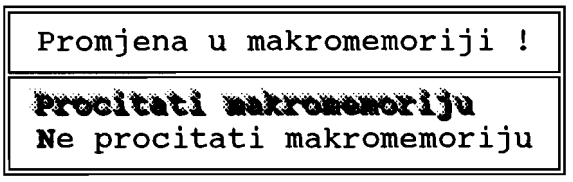
Najmanje što se može u jednom koraku izvršiti je jedna mikronaredba, a to se radi tipkom **F5** bez ikakvih upita. Zato je potrebno prije samog izvršenja provjeriti da li je sve postavljeno tako da se izvede upravo željena mikronaredba. Izvršiti će se ona mikronaredba koja je adresirana sa MCU, odnosno adresom na njenoj mikroadresnoj sabirnici, a ta se vrijednost vidi na zaslonu (**MiA** adresa). Ta je lokacija u mikromemorijskom prozoru osvijetljena, a sa desne strane istog prozora nalazi se njen prikaz u mnemoničkom obliku.

Zajedničko sa sve komande ove grupe je da se nakon izvršenja vraćaju obavijesti. Jedna takva obavijest je da ukoliko je upravo izvršena mikronaredba (može to biti i u sklopu izvršene makronaredbe ili programa) imala postavljeni M bit kao oznaku kraja simulacije ispisuje se slijedeće:




To je važno znati, budući da mikroadresa slijedeće mikronaredbe koja se treba izvršiti više nije pravilna, pa ju potrebno podesiti.

Ukoliko je izvršena mikronaredba nešto upisala u makromemoriju (to može biti i jedna od niza izvršenih mikronaredbi) ispisuje se izbornik:



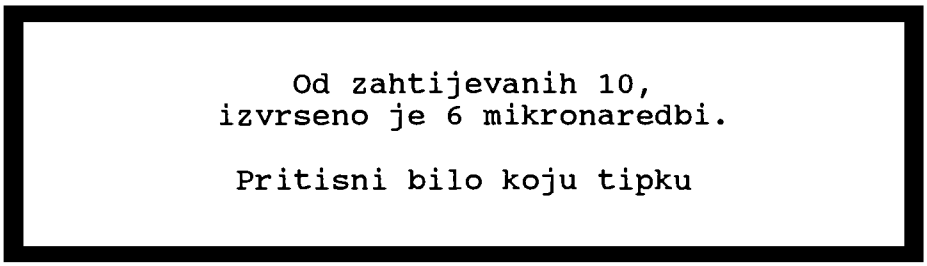

Korisnik tada može, ali ne mora, pročitati makromemoriju mikroprogramskog simulatora u memoriju osobnog računala i prikazati novo stanje u makromemorijskom prozu (to program radi automatski nakon pročitane makromemorije). To se preporuča uvijek raditi nakon ove obavijesti, inače sadržaj makromemorije na međusklopu mikroprogramskog simulatora neće odgovarati onome prikazanome na zaslonu.

Nova se mikroadresa postavlja sa **Ctrl-F5**. Adresa se unosi u heksadecimalnom obliku i smije biti u opsegu 0..#1FF (odnosno 0..511):


U okviru za unos pojavljuje se i trenutno stanje na mikroadresnoj sabirnici koje treba prepraviti u željenu vrijednost. Podešena vrijednost odmah se šalje u mikroprogramski simulator i postavlja u MCU, a promjena se vidi i na zaslonu.

Sa **Shift-F5** izvršava se više mikronaredbi uzastopno, a promjena registara, zastavica, sabirnice i makromemorije vide se tek nakon svih izvršenih mikronaredbi. Koliko će se mikronaredbi izvršiti zadaje se pritiskom na **Alt-F5** čime se otvara odgovarajući okvir za unos:


Može se zadati vrijednost do najviše 99. Potrebno je napomenuti da će se izvršavanje prekinuti i ranije ako se naiđe na mikronaredbu čiji je M bit postavljen na 1, kada će se ispisati prije spomenuta obavijest. Dodatno se u tom slučaju ispisuje obavijest o tome koliko je mikronaredbi izvršeno, npr.:




Analogne su komande za izvršavanje makronaredbi. Ovdje treba naprije reći nešto o tome gdje počinje i gdje završava makronaredba. Naime, po mikroprogramskoj riječi ne može se reći čijoj makronaredbi pripada. Mikroadresa na koju se dolazi na osnovu operacijskog kóda makronaredbe je prva za mikroprogramsku sekvencu specifičnu za tu makronaredbu, ali to je trenutak kada je operacijski kód pribavljen. To je u stvari kraj faze PRIBAVI, kao jedne od dvije faze u kojima se mikroprocesor može naći (druga je IZVRŠI). Pravilni početak makronaredbe je početak mikroprogramske sekvence PRIBAVI kojom se pribavlja operacijski kód slijedeće makronaredbe.

Ta mikroadresa nije konstantna i unaprijed određena, ali budući da je to rutina koja je u biti ista i zajednička za sve makronaredbe, obično se na ovom mikroračunalu oko bit-odrezaka Intel 3000 stavlja u nulti redak mikromemorije. Postoji i jak razlog za to: u nulti se redak može skočiti sa bilo koje druge mikroadrese, a taj je skok uobičajen na kraju sekvence svake izvršene makronaredbe. Radi jednostavnosti, ta se mikroprogramska sekvenca operacije PRIBAVI stavlja upravo od nulte adrese. Potrebno je napomenuti da to nije obavezno i nije pravilo (teoretski to može biti bilo gdje u mikromemoriji), ali je najpovoljnije prilikom mikroprogramiranja.

Zato se sa **Ctrl-F6** mora najprije zadati ta adresa u okviru za unos:

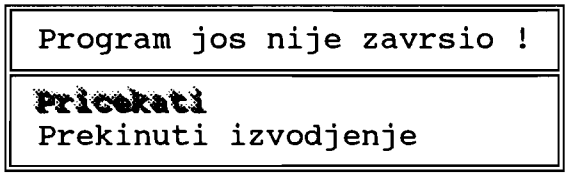

Ponuđena adresa je ona koja je zadnja podešena. Ta je adresa relevantna kód izvršavanja jedne ili više makronaredbi ili cijelog programa.

Važno je napomenuti da ta mikroprogramska sekvenca pripada onoj makronaredbi kod koje je definirana, tako da ako programer pokuša obrisati tu makronaredbu, obrisati će i tu sekvencu. Na to treba paziti.

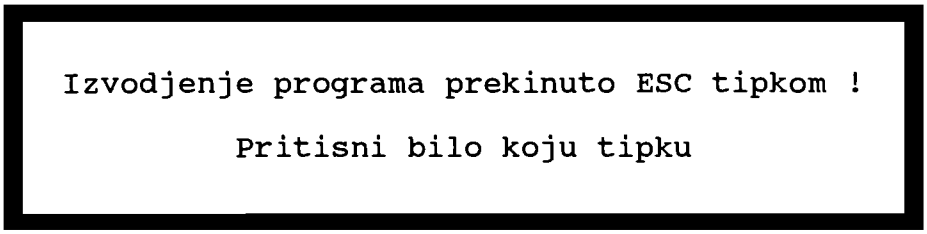
Jedna se makronaredba izvršava sa tipkom **F6**. Pri tome se uzima namještena adresa rutine PRIBAVI i izvodi mikroprogram od te mikroadrese do ponovnog dolaska na tu istu mikroadresu, što se smatra trenutkom kada završava prethodna makronaredba i treba početi slijedeća.

Iz toga proizilazi nekoliko problema koji se mogu javiti, a na koje treba paziti. Programer mora paziti da na kraju mikroprogramske sekvence jedne makronaredbe obavezno izvrši skok na tu adresu rutine PRIBAVI.

Previd tog pravila, ali i greška u mikroprogramiranju, može uzrokovati da izvršavanje mikronaredbi upadne u "mrtvu petlju" gdje se nikada ne dolazi do rutine PRIBAVI, ili do mikronaredbe sa M bitom na 1. Program tada ne može sam završiti, pa ga je potrbno prekinuti. Ako sa mikroprogramskog simulatora nema odgovora o kraju izvršavanja, ispisuje se izbornik:




Korisnik tada može pustiti da se program dalje izvodi ako je u pitanju neka stvarno duga programska petlja, ili prekinuti izvođenje. Ako se prekine izvođenje tada se uz ostale obavijesti javlja i slijedeća:




Analogno izvršavanju više mikronaredbi, sa **Shift-F6** može se izvršiti više makronaredbi. Također se tada uzima zadana adresa rutine PRIBAVI. Sa **Alt-F6** zadaje se broj koliko će se uzastopnih makronaredbi izvršiti:


Posljednji način izvršavanja je izvršavanje cijelog programa pritiskom na tipku **F7**. Program se tada izvršava dok se ne naiđe na mikronaredbu sa M bitom na 1 ili se prekine izvođenje.

Preporučljivo je ipak programe završavati "legalno". To je najbolje učiniti tako da se odmah definira jedna makronaredba tipa HALT ili STOP koja ne radi nikakvu mikronaredbu (npr. NOP/6), ali ima M bit na 1, pa će završiti izvođenje.

### 4.3. Datoteke koje stvara i koristi program

Sve datoteke koje stvara program su ASCII tako da se mogu vidjeti u editoru, ali njih je bolje ne mijenjati ukoliko se ne zna što se radi.

Za svaki mikroprocesor program stvara datoteku *ime_mikroprocesora*.CPU u koju upisuje sve sinonime registara, definirane makronaredbe i napisani mikroprogram.

Izvorni makroprogrami imaju nastavak *.ASS. Na osnovu tog imena program kreira datoteke istog osnovnog imena ali sa nastavcima *.OBJ (prevedeni binarni oblik program) i *.LAB (popis definiranih labela i njihove vrijednosti).

### 4.4. Dodatne napomene i preporuke

Na svim slobodnim mikromemorijskim lokacijama postavljen je M bit na 1, tako da ako program programerskom greškom skoči na krivu adresu simulacija prekida rad.

Automatsko generiranje operacijskog kóda prilikom definiranja makronaredbe preporuča se koristiti tek nakon napisane mikroprogramske rutine PRIBAVI na nultoj adresi, inače će program tu nultu lokaciju uzeti kao slobodnu i dodijeliti je makronaredbi sa operacijskim kódom #00.

Brisanje makronaredbe kod koje je načinjena mikroprogramska rutina za fazu PRIBAVI, briše i te mikronaredbe, pa ih je potrebno napisati ponovo u okviru neke druge makronaredbe.

# 5. Primjeri mikroprogramiranja

U ovom poglavlju biti će prikazani primjeri izrade mikroprograma za nekoliko tipičnih makronaredbi. **Preporuča se da se najprije upišu ovi primjeri, analiziraju mikroprogrami korak po korak, pišu makroprogrami sa načinjenim makronaredbama, te za vježbu modificiraju mikroopercije prije izrade samih zadataka.** Ovdje će biti samo navedene mikrooperacije sa kraćim opisom, dok je sam način upisa mikronaredbi u mikromemoriju opisan u prethodnom poglavlju. Oblik pisanja mnemonika mikrofunkcija i mikronaredbi skokova odgovara onom u originalnoj literaturi i programu koji se koristi.

Većina prikazanih makronaredbi slična je onima koje postoje na mikroprocesoru Zilog Z80 radi lakšeg razumijevanja, ali mnemonik makronaredbe može biti bilo kakav. Zato se preporuča da se za ove mnemonike odabrani mikroprocesor u programskoj podršci nazove upravo "Z80".

## 5.1. Priprema

Prije izrade mikroprograma pojedinih makronaredbi definirati ćemo nekoliko stvari. Kao prvo, naš mikroprocesor zvati će se "Z80", i imati će registre **A** i **B**, te programski brojač **PC**. Registar R0 bit će nam registar **A**, registar R1 **B**, a registar R9 programski brojač **PC**.

## 5.2. Mikrorutina PRIBAVI

Definirati ćemo zajedničku PRIBAVI rutinu koju čine dvije mikronaredbe na nultoj i prvoj mikroadresi. Ove mikronaredbe potrebno je upisati u okviru neke makronaredbe (svejedno koje, npr. LD A,#). Svi ostali mikroprogrami makronaredbi moraju i bit će pisani tako da na kraju skaču na nultu adresu (stupac 0 u retku 0).

| Adresa | Slijedeća | Zastavice | Funkcija | Registar | Memorija | Opis |
|---|---|---|---|---|---|---|
| 000 | JCR 1 | HCZ, FF1 | LMI/I | R9/I | čitanje | PC->MAR;(MAR)->M;PC+1->PC |
| 001 | LD=1 | HCZ, FF0 | NOP/6 | R0/I | ništa | A->A; CI->CO |

## 5.3. Makronaredba STOP - zaustavljanje izvođenja programa

Kao što je u prethodnom tekstu rečeno, dobro je imati naredbu tipa STOP kojom se zaustavlja rad mikroprogramskog sustava prilikom pokretanja programa bez izvođenja korak po korak. Njenoperacijski kod neka bude $FF_{(HEX)}$. Ona ne radi ništa, samo ima M bit na jedinici:

| Adresa | Slijedeća | Zastavice | Funkcija | Registar | Memorija | Opis |
|---|---|---|---|---|---|---|
| 0FF | M=1 | HCZ, FF0 | NOP/6 | R0/I | ništa | zaustavlja simulaciju |

## 5.4. Makronaredba LD A,# - punjenje registra konstantom

Ova makronaredba puni registar A (nama je to registar R0) konstantom koja se nalazi na lokaciji iza operacijskog koda makronaredbe. Njen operacijski kod neka bude $01_{(16)}$. Mikroprogram je slijedeći:

| Adresa | Slijedeća | Zastavice | Funkcija | Registar | Memorija | Opis |
|---|---|---|---|---|---|---|
| 010 | JCR 1 | HCZ, FF1 | LMI/1 | R9/I | čitanje | PC->MAR;(MAR)->M;PC+1->PC |

| | | | | | | |
|---|---|---|---|---|---|---|
| 011 | JCR 2 | HCZ, FF0 | ACM/0 | AC/II | ništa | M->AC |
| 012 | JZR 0 | HCZ, FF1 | SDR/2 | R0/I | ništa | AC->A |

### 5.5. Makronaredba ADD A,B - zbrajanje registara

Makronaredba ADD A,B zbraja registre A i B, postavlja C zastavicu ako je došlo do preljeva, te postavlja zastavicu Z ako je rezultat zbrajanja 0. Njen operacijski kod neka bude $61_{(16)}$. Mikroprogram je slijedeći:

| Adresa | Slijedeća | Zastavice | Funkcija | Registar | Memorija | Opis |
|---|---|---|---|---|---|---|
| 016 | JCR 7 | HCZ, FF0 | ILR/0 | R1/I | ništa | B->AC |
| 017 | JCC 2 | STC, FF0 | ADR/3 | R0/I | ništa | AC+A->A |
| 027 | JCR 6 | STZ, FF0 | TZR/5 | R0/I | ništa | zerro test |
| 026 | JCR 5 | HCZ, FFZ | CSR/2 | T/I | ništa | komplement zerro |
| 025 | JZR 0 | STZ, FF0 | TZR/5 | T/I | ništa | zerro test |

### 5.6. Makronaredba LD (#),A - pohranjivanje sadržaja registra u makromemoriju

Ovom makronaredbom pohranjuje se sadržaj registra A na zadanu lokaciju makromemorije. Adresa lokacije na koju se pohranjuje smješta se kao jedan oktet iza operacijskog koda. Operacijski kod neka bude $02_{(16)}$, a mikroprogram je slijedeći:

| Adresa | Slijedeća | Zastavice | Funkcija | Registar | Memorija | Opis |
|---|---|---|---|---|---|---|
| 020 | JCR 1 | HCZ, FF1 | LMI/1 | R9/I | čitanje | PC->MAR;(MAR)->M;PC+1->PC |
| 021 | JCR 2 | HCZ, FF0 | ACM/0 | AC/II | ništa | M->AC |
| 022 | JCR 3 | HCZ, FF0 | LMI/1 | AC/I | ništa | AC->MAR |
| 023 | JZR 0 | HCZ, FF0 | ILR/0 | R0/I | pisanje | A->AC->D-sabirnica |

### 5.7. Makronaredba INC A - povećavanje vrijednosti u registru za 1

Makronaredba INC A povećava vrijednost u registru A za 1 i postavlja Z zastavicu na 1, ako je nova vrijednost jednaka nuli. Operacijski kod naredbe neka bude $81_{(16)}$, a mikroprogram je slijedeći:

| Adresa | Slijedeća | Zastavice | Funkcija | Registar | Memorija | Opis |
|---|---|---|---|---|---|---|
| 018 | JCR 9 | HCZ, FF1 | ILR/0 | R0/I | ništa | A+1->A |
| 019 | JCR A | STZ, FF0 | TZR/5 | R0/I | ništa | zerro test |
| 01A | JCR B | HCZ, FFZ | CSR/2 | T/I | ništa | komplement zerro |
| 01B | JZR 0 | STZ, FF0 | TZR/5 | T/I | ništa | zerro test |

## 6. Zadaci

U nastavku su navedene tipične makronaredbe za koje je potrebno napisati mikroprograme. Preporuča se da se ove makronaredbe dodaju na one navedene u primjerima, kako bi se dobio mikroprocesor sa što više makronaredbi. Dan je samo osnovni opis, dok se detalji mogu samostalno odrediti prema vlastitom nahođenju i sličnosti sa proizvoljnim mikroprocesorom. Sve makronaredbe koje se izrađuju potrebno je rasčlaniti na mikrooperacije, napisati mikroprograme, izvesti ih mikronaredbu po mikronaredbu, te pisati makroprograme tako da se makronaredbe ispituju u specifičnim situacijama. Mikroprograme pisati tako da se upotrijebi što manje mikrooperacija. Radi proširivanja mogućnosti simuliranog mikroprocesora preporuča se dodati još registara, npr. C i D.

### 6.1. Punjenje registar - registar

Potrebno je definirati nekoliko makronaredbi za kopiranje sadržaja jednog registra u drugi, npr. LD A,B; LD B,A i slično. Zastavice moraju ostati nepromijenjene.

### 6.2. Zbrajanje

Gore definiranu makronaredbu za zbrajanje iskoristiti kao osnovu za novu makronaredbu za zbrajanje dva registra i preljeva (carry zastavice).

Definirati makronaredbu oblika ADD A,B,C koja zbraja dva registra i rezultat sprema u treći registar (to je tzv. **troadresna** naredba).

### 6.3. Oduzimanje

Potrebno je napisati mikroprograme na nekoliko vrsta oduzimanja, kao što su SUB (obično oduzimanje uz odgovarajuće postavljanje zastavica), SBC (oduzimanje s preljevom) i DEC (smanjivanje vrijednosti registra za 1. Operacije izvoditi između registara, a po želji i sa memorijskim lokacijama.

### 6.4. Logičke operacije

Napisati mikroprograme za razne logičke operacije poput slijedećih:

AND; OR; NOT; NEG - logičke operacije I, ILI, NE, suprotan predznak
SLR; SRR - pomak registra u lijevo ili desno, pri čemu bit koji ispada odlazi u C zastavicu
RLC; RRC - kružni pomak registra, pri čemu bit koji ispada dolazi na suprotnu stranu registra
ASL; ASR - aritmetički pomak, pri čemu se bit predznaka zadržava

### 6.5. Čitanje iz memorije

Napisati mikroprogram makronaredbe oblika LD A,(#) kojom se određeni registar puni sadržajem određene makromemorijske lokacije.

### 6.6. Skokovi

Načiniti nekoliko makronaredbi za skokove unutar makroprograma:

JP adresa - skok na određenu makroadresu
JPZ adresa - skok na određenu makroadresu ako je zastavica Z=1

### 6.7. Rad sa stogom

Napisati mikroprograme za makronaredbe koje rade operacije sa stogom. Osnovne operacije bile bi PUSH i POP koje stavljaju sadržaj nekog registra na stog, odnosno uzimaju vrijednost u neki registar. U tu svrhu potrebno je definirati registar pokazivač stoga (npr. **SP**), te još neke nužne operacije kao što je makronaredba za postavljenje vrijednosti pokazivača stoga poput LD SP,#.

Daljne mogućnosti su da se načine makronaredbe stogovnog računala, tj. aritmetičko - logičke operacije koje nemaju eksplicitno zadane operande, već njih nalaze na vrhu stoga, gdje nakon izvršenja ostavljaju i

rezultat. Npr. ADDS zbraja dva operanda sa vrha stoga, i ostavlja umjesto njih rezultat na vrhu stoga.

### 6.8. Potprogrami

Kao nastavak na prethodni zadatak u kojem su se načinile osnovne operacije za rad sa stogom, mogu se načiniti makronaredbe za poziv makropotprograma (npr. bezuvjetno CALL adresa; uvjetno CALL C adresa) i za povratak iz potprograma (npr. RET; RET Z; RET C).

## 7. Prilozi

### 7.1. Mikronaredbe za skok na slijedeću mikroadresu (MCU 3001)

### 7.2. Mikrooperacije procesnih elemenata (CPE 3002)

## FUNCTIONAL DESCRIPTION

The following is a description of each of the eleven address control functions. The symbols shown below are used to specify row and column addresses.

| SYMBOL | MEANING |
|---|---|
| $row_n$ | 5-bit next row address where n is the decimal row address. |
| $col_n$ | 4-bit next column address where n is the decimal column address. |

## UNCONDITIONAL ADDRESS CONTROL (JUMP) FUNCTIONS

The jump functions use the current microprogram address (i.e., the contents of the microprogram address register prior to the rising edge of the clock) and several bits from the address control inputs to generate the next microprogram address.

| MNEMONIC | FUNCTION DESCRIPTION |
|---|---|
| JCC | Jump in current column. $AC_0$–$AC_4$ are used to select 1 of 32 row addresses in the current column, specified by $MA_0$–$MA_3$, as the next address. |
| JZR | Jump to zero row. $AC_0$–$AC_3$ are used to select 1 of 16 column addresses in $row_0$, as the next address. |
| JCR | Jump in current row. $AC_0$–$AC_3$ are used to select 1 of 16 addresses in the current row, specified by $MA_4$–$MA_8$, as the next address. |
| JCE | Jump in current column/row group and enable PR-latch outputs, $AC_0$–$AC_2$ are used to select 1 of 8 row addresses in the current row group, specified by $MA_7$–$MA_8$, as the next row address. The current column is specified by $MA_0$–$MA_3$. The PR-latch outputs are asynchronously enabled. |

## FLAG CONDITIONAL ADDRESS CONTROL (JUMP/TEST) FUNCTIONS

The jump/test flag functions use the current microprogram address, the contents of the selected flag or latch, and several bits from the address control function to generate the next microprogram address.

| MNEMONIC | FUNCTION DESCRIPTION |
|---|---|
| JFL | Jump/test F-latch. $AC_0$–$AC_3$ are used to select 1 of 16 row addresses in the current row group, specified by $MA_8$, as the next row address. If the current column group, specified by $MA_3$, is $col_0$–$col_7$, the F-latch is used to select $col_2$ or $col_3$ as the next column address. If $MA_3$ specifies column group $col_8$–$col_{15}$, the F-latch is used to select $col_{10}$ or $col_{11}$ as the next column address. |
| JCF | Jump/test C-flag. $AC_0$–$AC_2$ are used to select 1 of 8 row addresses in the current row group, specified by $MA_7$ and $MA_8$, as the next row address. If the current column group specified by $MA_8$ is $col_0$–$col_7$, the C-flag is used to select $col_2$ or $col_3$ as the next column address. If $MA_3$ specifies column group $col_8$–$col_{15}$, the C-flag is used to select $col_{10}$ or $col_{11}$ as the next column address. |
| JZF | Jump/test Z-flag. Identical to the JCF function described above, except that the Z-flag, rather than the C-flag, is used to select the next column address. |

## PX-BUS AND PR-LATCH CONDITIONAL ADDRESS CONTROL (JUMP/TEST) FUNCTIONS

The PX-bus jump/test function uses the data on the primary instruction bus ($PX_4$–$PX_7$), the current microprogram address, and several selection bits from the address control function to generate the next microprogram address. The PR-latch jump/test functions use the data held in the PR-latch, the current microprogram address, and several selection bits from the address control function to generate the next microprogram address.

| MNEMONIC | FUNCTION DESCRIPTION |
|---|---|
| JPR | Jump/test PR-latch. $AC_0$–$AC_2$ are used to select 1 of 8 row addresses in the current row group, specified by $MA_7$ and $MA_8$, as the next row address. The four PR-latch bits are used to select 1 of 16 possible column addresses as the next column address. |
| JLL | Jump/test leftmost PR-latch bits. $AC_0$–$AC_2$ are used to select 1 of 8 row addresses in the current row group, specified by $MA_7$ and $MA_8$, as the next row address. $PR_2$ and $PR_3$ are used to column addresses in $col_4$ through $col_7$ as the next column address. |
| JRL | Jump/test rightmost PR-latch bits. $AC_0$ and $AC_1$ are used to select 1 of 4 high-order row addresses in the current row group, specified by $MA_7$ and $MA_8$, as the next row address. $PR_0$ and $PR_1$ are used to select 1 of 4 possible column addresses in $col_{12}$ through $col_{15}$ as the next column address. |
| JPX | Jump/test PX-bus and load PR-latch. $AC_0$ and $AC_1$ are used to select 1 of 4 row addresses in the current row group, specified by $MA_6$–$MA_8$, as the next row address. $PX_4$–$PX_7$ are used to select 1 of 16 possible column addresses as the next column address. $SX_0$–$SX_3$ data is locked in the PR-latch at the rising edge of the clock. |

## PX-BUS AND PR-LATCH CONDITIONAL ADDRESS CONTROL (JUMP/TEST) FUNCTIONS (Continued)

The flag control functions of the MCU are selected by the four input lines designated $FC_0$–$FC_3$. Function code formats are given in "Flag Control Function summary".

The following is a detailed description of each of the eight flag control functions.

## FLAG INPUT CONTROL FUNCTIONS

The flag input control functions select which flag or flags will be set to the current value of the flag input (FI) line. Data on FI is stored in the F-latch when the clock is low. The content of the F-latch is loaded into the C and/or Z flag on the rising edge of the clock.

| MNEMONIC | FUNCTION DESCRIPTION |
|---|---|
| SCZ | Set C-flag and Z-flag to FI. The C-flag and the Z-flag are both set to the value of FI. |
| STZ | Set Z-flag to FI. The Z-flag is set to the value of FI. The C-flag is unaffected. |
| STC | Set C-flag to FI. The C-flag is set to the value of FI. The Z-flag is unaffected. |
| HCZ | Hold C-flag and Z-flag. The values in the C-flag and Z-flag are unaffected. |

## FLAG OUTPUT CONTROL FUNCTIONS

The flag output control functions select the value to which the flag output (FO) line will be forced.

| MNEMONIC | FUNCTION DESCRIPTION |
|---|---|
| FFO | Force FO to O. FO is forced to the value of logical O. |
| FFC | Force FO to C. FO is forced to the value of the C-flag. |
| FFZ | Force FO to Z. FO is forced to the value of the Z-flag. |
| FF1 | Force FO to 1. FO is forced to the value of logical 1. |

## STROBE FUNCTIONS

The load function of the MCU is controlled by the input line designated LD. If the LD line is active HIGH at the rising edge of the clock, the date on the primary and secondary instruction busses, $PX_4$–$PX_7$ and $SX_0$–$SX_3$, is loaded into the microprogram address register. $PX_4$–$PX_7$ are loaded into $MA_0$–$MA_3$ and $SX_0$–$SX_3$ are loaded into $MA_4$–$MA_7$. The high-order bit of the microprogram address register $MA_8$ is set to a logical 0. The bits from the primary instruction bus select 1 of 16 possible column addresses. Likewise, the bits from the secondary instruction bus select 1 of the first 16 row addresses.

The MCU generates an interrupt strobe enable on the output line designated ISE. The line is placed in the active high state whenever a JZR to $col_{15}$ is selected as the address control function. Generally, the start of a macroinstruction fetch sequence is situated at $row_0$ and $col_{15}$ so the interrupt control may be enabled at the beginning of

## ADDRESS CONTROL FUNCTION SUMMARY

| MNEMONIC | DESCRIPTION | FUNCTION | | | | | | | NEXT ROW | | | | | NEXT COL | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $AC_6$ | 5 | 4 | 3 | 2 | 1 | 0 | $MA_8$ | 7 | 6 | 5 | 4 | $MA_3$ | 2 | 1 | 0 |
| JCC | Jump in current column | 0 | 0 | $d_4$ | $d_3$ | $d_2$ | $d_1$ | $d_0$ | $d_4$ | $d_3$ | $d_2$ | $d_1$ | $d_0$ | $m_3$ | $m_2$ | $m_1$ | $m_0$ |
| JZR | Jump to zero row | 0 | 1 | 0 | $d_3$ | $d_2$ | $d_1$ | $d_0$ | 0 | 0 | 0 | 0 | 0 | $d_3$ | $d_2$ | $d_1$ | $d_0$ |
| JCR | Jump in current row | 0 | 1 | 1 | $d_3$ | $d_2$ | $d_1$ | $d_0$ | $m_8$ | $m_7$ | $m_6$ | $m_5$ | $m_4$ | $d_3$ | $d_2$ | $d_1$ | $d_0$ |
| JCE | Jump in column/enable | 1 | 1 | 1 | 0 | $d_2$ | $d_1$ | $d_0$ | $m_8$ | $m_7$ | $d_2$ | $d_1$ | $d_0$ | $m_3$ | $m_2$ | $m_1$ | $m_0$ |
| JFL | Jump/test F-latch | 1 | 0 | 0 | $d_3$ | $d_2$ | $d_1$ | $d_0$ | $m_8$ | $d_3$ | $d_2$ | $d_1$ | $d_0$ | $m_3$ | 0 | 1 | f |
| JCF | Jump/test C-flag | 1 | 0 | 1 | 0 | $d_2$ | $d_1$ | $d_0$ | $m_8$ | $m_7$ | $d_2$ | $d_1$ | $d_0$ | $m_3$ | 0 | 1 | c |
| JZF | Jump/test Z-flag | 1 | 0 | 1 | 1 | $d_2$ | $d_1$ | $d_0$ | $m_8$ | $m_7$ | $d_2$ | $d_1$ | $d_0$ | $m_3$ | 0 | 1 | z |
| JPR | Jump/test PR-latch | 1 | 1 | 0 | 0 | $d_2$ | $d_1$ | $d_0$ | $m_8$ | $m_7$ | $d_2$ | $d_1$ | $d_0$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |
| JLL | Jump/test left PR bits | 1 | 1 | 0 | 1 | $d_2$ | $d_1$ | $d_0$ | $m_8$ | $m_7$ | $d_2$ | $d_1$ | $d_0$ | 0 | 1 | $p_3$ | $p_2$ |
| JRL | Jump/test right PR bits | 1 | 1 | 1 | 1 | 1 | $d_1$ | $d_0$ | $m_8$ | $m_7$ | 1 | $d_1$ | $d_0$ | 1 | 1 | $p_1$ | $p_0$ |
| JPX | Jump/test PX-bus | 1 | 1 | 1 | 1 | 0 | $d_1$ | $d_0$ | $m_8$ | $m_7$ | $m_6$ | $d_1$ | $d_0$ | $x_7$ | $x_6$ | $x_5$ | $x_4$ |

NOTE:
$d_n$ = Data pm address control line n
$m_n$ = Data in microprogram address register bit n
$p_n$ = Data in PR-latch bit n
$x_n$ = Data on PX-bus line n (active LOW)
f, c, z = Contents of F-latch, C-flag, or Z-flag, respectively

## STROBE FUNCTIONS Cont'd.

the fetch/execute cycle. The interrupt control responds to the interrupt by pulling the enable row address (ERA) input line low to override the selected next row address from the MCU. Then by gating an alternative next row address on to the row address lines of the microprogram memory, the microprogram may be forced to enter an interrupt handling routine. The alternative row address placed on the microprogram memory address lines does not alter the contents of the microprogram address register. Therefore, subsequent jump functions will utilize the row address in the register, and not the alternative row address, to determine the next microprogram address.

Note, the load function always overrides the address control function on $AC_0$–$AC_6$. It does not, however, override the latch enable or load sub-functions of the JCE or JPX instruction, respectively. In addition, it does not inhibit the interrupt strobe enable or any of the flag control functions.

## FLAG CONTROL FUNCTION SUMMARY

| TYPE | MNEMONIC | DESCRIPTION | $FC_1$ | 0 |
|---|---|---|---|---|
| Flag Input | SCZ | Set C-flag and Z-flag to f | 0 | 0 |
| | STZ | Set Z-flag to f | 0 | 1 |
| | STC | Set C-flag to f | 1 | 0 |
| | HCZ | Hold C-flag and Z-flag | 1 | 1 |

| TYPE | MNEMONIC | DESCRIPTION | $FC_3$ | 2 |
|---|---|---|---|---|
| Flag Output | FF0 | Force FO to 0 | 0 | 0 |
| | FFC | Force FO to C-flag | 0 | 1 |
| | FFZ | Force FO to Z-flag | 1 | 0 |
| | FF1 | Force FO to 1 | 1 | 1 |

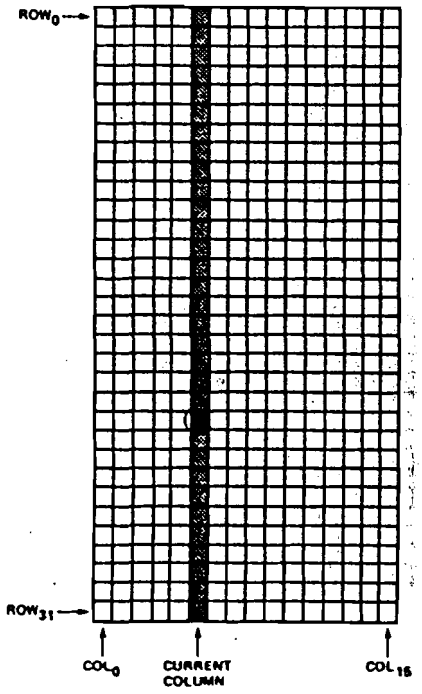
| LOAD FUNCTION | NEXT ROW | | | | | NEXT COL | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| LD | $MA_8$ | 7 | 6 | 5 | 4 | $MA_3$ | 2 | 1 | 0 | |
| 0 | See Appendix A | | | | | See Appendix A | | | | |
| 1 | 0 | $x_3$ | $x_2$ | $x_1$ | $x_0$ | $x_7$ | $x_6$ | $x_5$ | $x_4$ | |

NOTE:
f = Contents of the F-latch
$x_n$ = Data on PX- or SX-bus line n (active LOW)

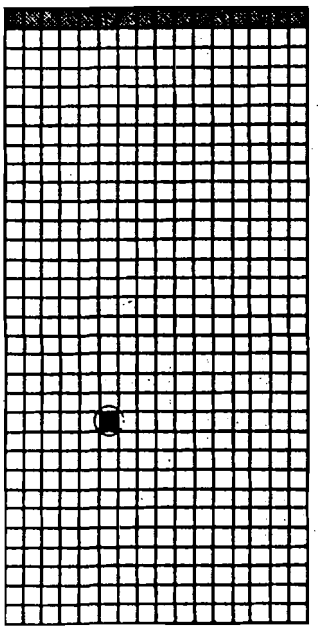
## JUMP SET DIAGRAMS

The following ten diagrams illustrate the jump set for each of the eleven jump and jump/test functions of the MCU. Location 341 indicated by the circled square, represents one current row ($row_{21}$) and current column ($col_5$) address. The dark boxes indicate the microprogram locations that may be selected by the particular function as the next address.

JCC
JUMP IN CURRENT COLUMN

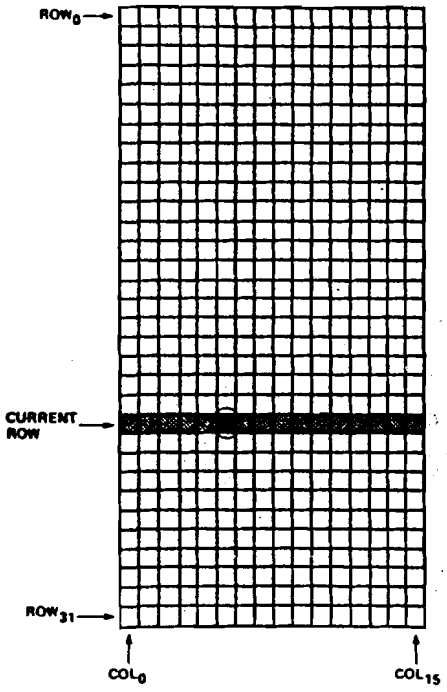



JZR
JUMP TO ZERO ROW

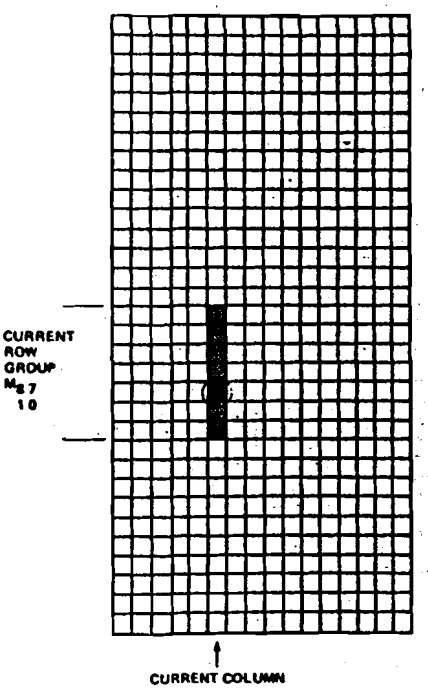


## JUMP SET DIAGRAMS Cont'd.

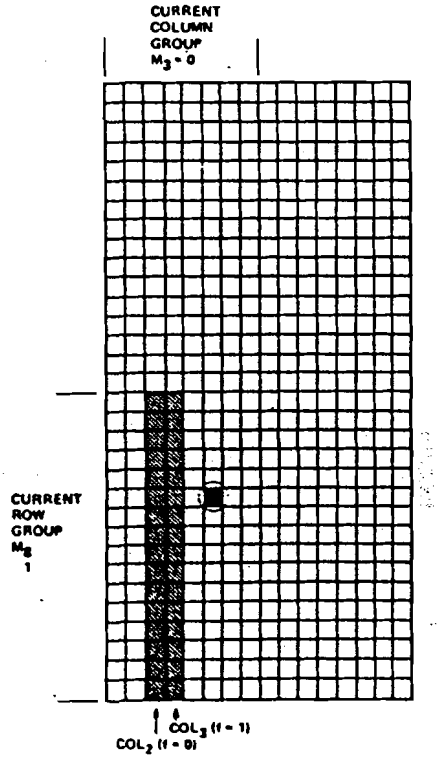
### JCR
### JUMP IN CURRENT ROW



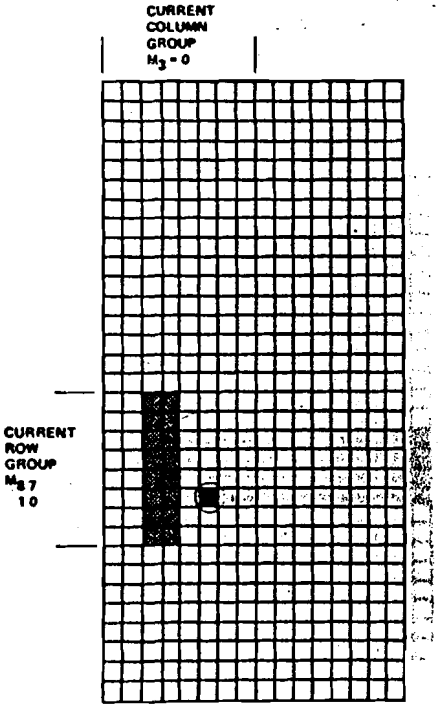

### JCE
### JUMP COLUMN/ENABLE




### JFL
### JUMP/TEST F-LATCH

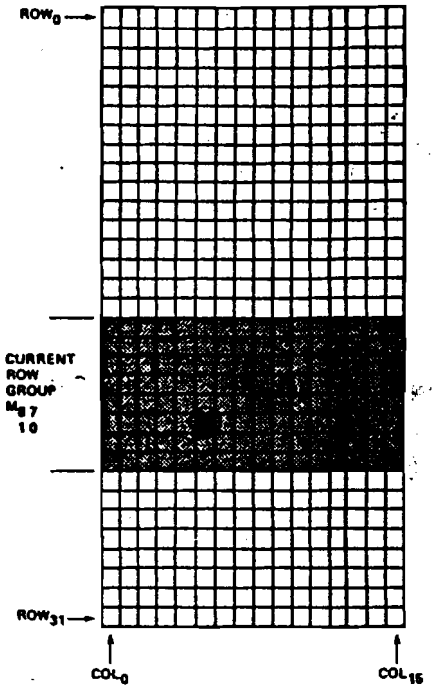



### JCF, JZF
### JUMP/TEST C-FLAG
### JUMP/TEST Z-FLAG

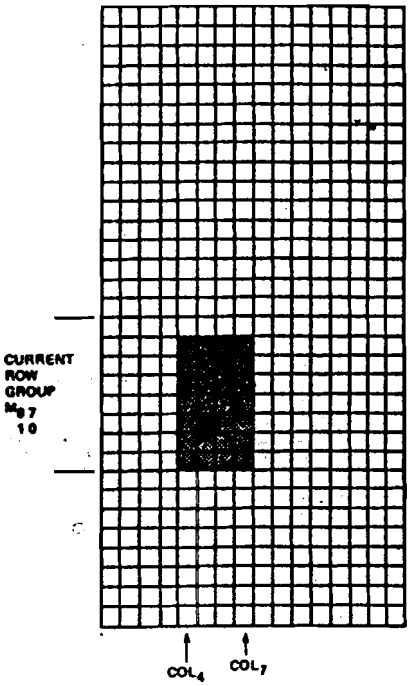


## JUMP SET DIAGRAMS Cont'd.

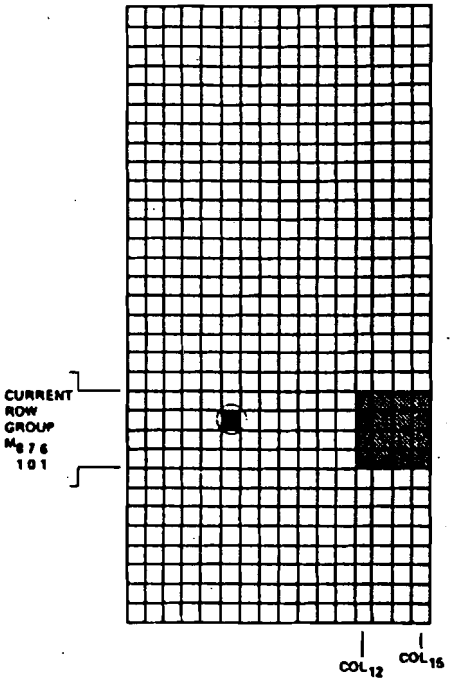
### JPR
### JUMP/TEST PR-LATCH



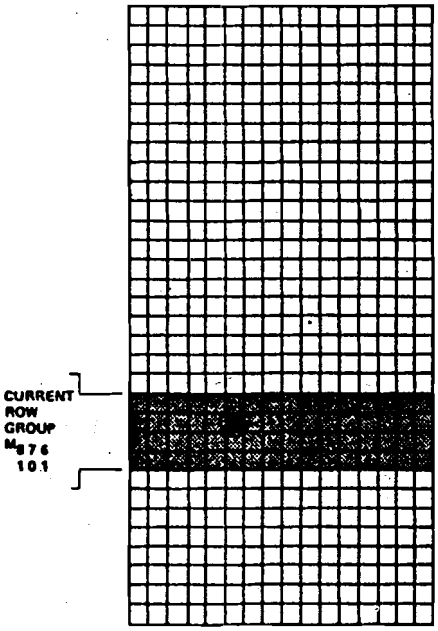

### JLL
### JUMP/TEST LEFT LATCH




### JRL
### JUMP/TEST RIGHT LATCH




### JPX
### JUMP/TEST PX-BUS



## FUNCTION DESCRIPTION

| F GROUP | R GROUP | K BUS | NAME | EQUATION | DESCRIPTION |
|---|---|---|---|---|---|
| 0 | I | XX | – | $R_n + (AC \wedge K) + CI \rightarrow R_n, AC$ | Logically AND AC with the K-bus. Add the result to $R_n$ and carry input (CI). Deposit the sum in AC and $R_n$. |
| | | OO | ILR | $R_n + CI \rightarrow R_n, AC$ | Conditionally increment $R_n$ and load the result in AC. Used to load AC from $R_n$ or to increment $R_n$ and load a copy of the result in AC. |
| | | II | ALR | $AC + R_n + CI \rightarrow R_n, AC$ | Add AC and CI to $R_n$ and load the result in AC. Used to add AC to a register. If $R_n$ is AC, then AC is shifted left one bit position. |
| 0 | II | XX | – | $M + (AC \wedge K) + CI \rightarrow AT$ | Logically AND AC with the K-bus. Add the result to CI and the M-bus. Deposit the sum in AC or T. |
| | | OO | ACM | $M + CI \rightarrow AT$ | Add CI to M-bus. Load the result in AC or T, as specified. Used to load memory data in the specified register, or to load incremented memory data in the specified register. |
| | | II | AMA | $M + AC + CI \rightarrow AT$ | Add the M-bus to AC and CI, and load the result in AC or T, as specified. Used to add memory data or incremented memory data to AC and store the sum in the specified register. |
| 0 | III | XX | – | $AT_L \wedge \overline{(I_L \wedge K_L)} \rightarrow RO$<br>$LI \vee [(I_H \wedge K_H) \wedge AT_H] \rightarrow AT_H$<br>$[AT_L \wedge (I_L \wedge K_L)] \vee [AT_H \vee (I_H \wedge K_H)] \rightarrow AT_L$ | None |
| | | OO | SRA | $AT_L \rightarrow RO \quad AT_H \rightarrow AT_L \quad LI \rightarrow AT_H$ | Shift AC or T, as specified, right one bit position. Place the previous low order bit value on RO and fill the high order bit from the data on LI. Used to shift or rotate AC or T right one bit. |
| 1 | I | XX | – | $K \vee R_n \rightarrow MAR$<br>$R_n + K + CI \rightarrow R_n$ | Logically OR $R_n$ with the K-bus. Deposit the result in MAR. Add the K-bus to $R_n$ and CI. Deposit the result in $R_n$. |
| | | OO | LMI | $R_n \rightarrow MAR \quad R_n + CI \rightarrow R_n$ | Load MAR from $R_n$. Conditionally increment $R_n$. Used to maintain a macro-instruction program counter. |
| | | II | DSM | $11 \rightarrow MAR \quad R_n - 1 + CI \rightarrow R_n$ | Set MAR to all one's. Conditionally decrement $R_n$ by one. Used to force MAR to its highest address and to decrement $R_n$. |

| F GROUP | R GROUP | K BUS | NAME | EQUATION | |
|---|---|---|---|---|---|
| 1 | II | XX | – | $K \vee M \rightarrow MAR$<br>$M + K + CI \rightarrow AT$ | Logically OR the M-bus with the K-bus. Deposit the result in MAR. Add the K-bus to the M-bus and CI. Deposit the sum in AC or T. |
| | | OO | LMM | $M \rightarrow MAR \quad M + CI \rightarrow AT$ | Load MAR from the M-bus. Add CI to the M-bus. Deposit the result in AC or T. Used to load the address register with memory data for macro-instructions using indirect addressing. |
| | | II | LDM | $11 \rightarrow MAR$<br>$M - 1 + CI \rightarrow AT$ | Set MAR to all ones. Subtract one from the M-bus. Add CI to the difference and deposit the result in AC or T, as specified. Used to load decremented memory data in AC or T. |
| 1 | III | XX | – | $(\overline{AT} \vee K) + (AT \wedge K) + CI \rightarrow AT$ | Logically OR the K-bus with the complement of AC or T, as specified. Add the result to the logical AND of specified register with the K-bus. Add the sum to CI. Deposit the result in the specified register. |
| | | OO | CIA | $\overline{AT} + CI \rightarrow AT$ | Add CI to the complement of AC or T, as specified. Deposit the result in the specified register. Used to form the 1's or 2's complement of AC or T. |
| | | II | DCA | $AT - 1 + CI \rightarrow AT$ | Subtract one from AC or T, as specified. Add CI to the difference and deposit the sum in the specified register. Used to decrement AC or T. |
| 2 | I | XX | – | $(AC \wedge K) - 1 + CI \rightarrow R_n$<br>(See Note 1) | Logically AND the K-bus with AC. Subtract one from the result and add the difference to CI. Deposit the sum in $R_n$. |
| | | OO | CSR | $CI - 1 \rightarrow R_n$<br>(See Note 1) | Subtract one from CI and deposit the difference in $R_n$. Used to conditionally clear or set $R_n$ to all 0's or 1's, respectively. |
| | | II | SDR | $AC - 1 + CI \rightarrow R_n$<br>(See Note 1) | Subtract one from AC and add the difference to CI. Deposit the sum in $R_n$. Used to store AC in $R_n$ or to store the decremented value of AC in $R_n$. |
| 2 | II | XX | – | $(AC \wedge K) - 1 + CI \rightarrow AT$<br>(See Note 1) | Logically AND the K-bus with AC. Subtract one from the result and add the difference to CI. Deposit the sum in AC or T, as specified. |
| | | OO | CSA | $CI - 1 \rightarrow AT$<br>(See Note 1) | Subtract one from CI and deposit the difference in AC or T. Used to conditionally clear or set AC or T. |
| | | II | SDA | $AC - 1 + CI \rightarrow AT$<br>(See Note 1) | Subtract one from AC and add the difference to CI. Deposit the sum in AC or T. Used to store AC in T, or decrement AC, or store the decremented value of AC in T. |

## FUNCTION DESCRIPTION (CONT'D)

| F GROUP | R GROUP | K BUS | NAME | EQUATION | DESCRIPTION |
|---|---|---|---|---|---|
| 2 | III | XX | – | $(I \wedge K) - 1 + CI \rightarrow AT$ (See Note 1) | Logically AND the data of the K-bus with the data on the I-bus. Subtract one from the result and add the difference to CI. Deposit the sum in AC or T, as specified. |
| | | OO | CAS | $CI - 1 \rightarrow AT$ | Subtract one from CI and deposit the difference in AC or T. Used to conditionally clear or set AC or T. |
| | | II | LDI | $I - 1 + CI \rightarrow AT$ | Subtract one from the data on the I-bus and add the difference to CI. Deposit the sum in AC or T, as specified. Used to load input bus data or decremented input bus data in the specified register. |
| 3 | I | XX | – | $R_n + (AC \wedge K) + CI \rightarrow R_n$ | Logically AND AC with the K-bus. Add $R_n$ and CI to the result. Deposit the sum in $R_n$. |
| | | OO | INR | $R_n + CI \rightarrow R_n$ | Add CI to $R_n$ and deposit the sum in $R_n$. Used to increment $R_n$. |
| | | II | ADR | $AC + R_n + CI \rightarrow R_n$ | Add AC to $R_n$. Add the result to CI and deposit the sum in $R_n$. Used to add the accumulator to a register or to add the incremented value of the accumulator to a register. |
| 3 | II | XX | – | $M + (AC \wedge K) + CI \rightarrow AT$ | Logically AND AC with the K-bus. Add the result to CI and the M-bus Deposit the sum in AC or T. |
| | | OO | ACM | $M + CI \rightarrow AT$ | Add CI to M-bus. Load the result in AC or T, as specified. Used to load memory data in the specified register, or to load incremented memory data in the specified register. |
| | | II | AMA | $M + AC + CI \rightarrow AT$ | Add the M-bus to AC and CI, and load the result in AC or T, as specified. Used to add memory data or incremented memory data to AC and store the sum in the specified register. |
| 3 | III | XX | – | $AT + (I \wedge K) + CI \rightarrow AT$ | Logically AND the K-bus with the I-bus. Add CI and the contents of AC or T, as specified, to the result. Deposit the sum in the specified register. |
| | | OO | INA | $AT + CI \rightarrow AT$ | Conditionally increment AC or T. Used to increment AC or T. |
| | | II | AIA | $I + AT + CI \rightarrow AT$ | Add the I-bus to AC or T. Add CI to the result and deposit the sum in the specified register. Used to add input data or incremented input data to the specified register. |

## FUNCTION TRUTH TABLE

| FUNCTION GROUP | $F_6$ | $F_5$ | $F_4$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 |
| 4 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 |

| REGISTER GROUP | REGISTER | $F_3$ | $F_2$ | $F_1$ | $F_0$ |
|---|---|---|---|---|---|
| I | $R_0$ | 0 | 0 | 0 | 0 |
| | $R_1$ | 0 | 0 | 0 | 1 |
| | $R_2$ | 0 | 0 | 1 | 0 |
| | $R_3$ | 0 | 0 | 1 | 1 |
| | $R_4$ | 0 | 1 | 0 | 0 |
| | $R_5$ | 0 | 1 | 0 | 1 |
| | $R_6$ | 0 | 1 | 1 | 0 |
| | $R_7$ | 0 | 1 | 1 | 1 |
| | $R_8$ | 1 | 0 | 0 | 0 |
| | $R_9$ | 1 | 0 | 0 | 1 |
| | T | 1 | 1 | 0 | 0 |
| | AC | 1 | 1 | 0 | 1 |
| II | T | 1 | 0 | 1 | 0 |
| | AC | 1 | 0 | 1 | 1 |
| III | T | 1 | 1 | 1 | 0 |
| | AC | 1 | 1 | 1 | 1 |

| SYMBOL | MEANING |
|---|---|
| I, K, M | Data on the I, K, and M busses, respectively |
| CI, LI | Data on the carry input and left input, respectively |
| CO, RO | Data on the carry output and right output, respectively |
| $R_n$ | Contents of register n including T and AC (R-Group I) |
| AC | Contents of the accumulator |
| AT | Contents of AC or T, as specified |
| MAR | Contents of the memory address register |
| L, H | As subscripts, designate low and high order bit, respectively |
| + | 2's complement addition |
| – | 2's complement subtraction |
| $\wedge$ | Logical AND |
| $\vee$ | Logical OR |
| $\overline{\vee}$ | Exclusive-NOR |
| $\rightarrow$ | Deposit into |

NOTE:
1. 2's complement arithmetic adds 111 . . . 11 to perform subtraction of 000 . . . 01.

## FUNCTION DESCRIPTION (CONT'D)

| F GROUP | R GROUP | K BUS | NAME | EQUATION | DESCRIPTION |
|---|---|---|---|---|---|
| 4 | I | XX | – | $CI \vee (R_n \wedge AC \wedge K) \rightarrow CO$<br>$R_n \wedge (AC \wedge K) \rightarrow R_n$ | Logically AND the K-bus with AC. Logically AND the result with the contents of $R_n$. Deposit the final result in $R_n$. Logically OR the value of CI with the word-wise OR of the bits of the final result. Place the value of the carry OR on the carry output (CO) line. |
| | | OO | CLR | $CI \rightarrow CO$ $0 \rightarrow R_n$ | Clear $R_n$ to all 0's. Force CO to CI. Used to clear a register and force CO to CI. |
| | | II | ANR | $CI \vee (R_n \wedge AC) \rightarrow CO$<br>$R_n \wedge AC \rightarrow R_n$ | Logically AND AC with $R_n$. Deposit the result in $R_n$. Force CO to one if the result is non-zero. Used to AND the accumulator with a register and test for a zero result. |
| 4 | II | XX | – | $CI \vee (M \wedge AC \wedge K) \rightarrow CO$<br>$M \wedge (AC \wedge K) \rightarrow AT$ | Logically AND the K-bus with AC. Logically AND the result with the M-bus. Deposit the final result in AC or T. Logically OR the value of CI with the word-wise OR of the bits of the final result. Place the value of the carry OR on CO. |
| | | OO | CLA | $CI \rightarrow CO$ $0 \rightarrow AT$ | Clear AC or T, as specified, to all 0's. Force CO to CI. Used to clear the specified register and force CO to CI. |
| | | II | ANM | $CI \vee (M \wedge AC) \rightarrow CO$<br>$M \wedge AC \rightarrow AT$ | Logically AND the M-bus with AC. Deposit the result in AC or T. Force CO to one if the result is non-zero. Used to AND M-bus data to the accumulator and test for a zero result. |
| 4 | III | XX | – | $CI \vee (AT \wedge I \wedge K) \rightarrow CO$<br>$AT \wedge (I \wedge K) \rightarrow AT$ | Logically AND the I-bus with the K-bus. Logically AND the result with AC or T. Deposit the final result in the specified register. Logically OR CI with the word-wise OR of the final result. Place the value of the carry OR on CO. |
| | | OO | CLA | $CI \rightarrow CO$ $0 \rightarrow AT$ | Clear AC or T, as specified, to all 0's. Force CO to CI. Used to clear the specified register and force CO to CI. |
| | | II | ANI | $CI \vee (AT \wedge I) \rightarrow CO$<br>$AT \wedge I \rightarrow AT$ | Logically AND the I-bus with AC or T, as specified. Deposit the result in the specified register. Force CO to one if the result is non-zero. Used to AND the I-bus to the accumulator and test for a zero result. |
| 5 | I | XX | – | $CI \vee (R_n \wedge K) \rightarrow CO$<br>$K \wedge R_n \rightarrow R_n$ | Logically AND the K-bus with $R_n$. Deposit the result in $R_n$. Logically OR CI with the word-wise OR of the result. Place the value of the carry OR on CO. |
| | | OO | CLR | $CI \rightarrow CO$ $0 \rightarrow R_n$ | Clear $R_n$ to all 0's. Force CO to CI. Used to clear a register and force CO to CI. |
| | | II | TZR | $CI \vee R_n \rightarrow CO$<br>$R_n \rightarrow R_n$ | Force CO to one if $R_n$ is non-zero. Used to test a register for zero. Also used to AND K-bus data with a register for masking and, optionally, testing for a zero result. |
| 5 | II | XX | – | $CI \vee (M \wedge K) \rightarrow CO$<br>$K \wedge M \rightarrow AT$ | Logically AND the K-bus with the M-bus. Deposit the result in AC or T, as specified. Logically OR CI with the word-wise OR of the result. Place the value of the carry OR on CO. |
| | | OO | CLA | $CI \rightarrow CO$ $0 \rightarrow AT$ | Clear AC or T, as specified, to all 0's. Force CO to CI. Used to clear the specified register and force CO to CI. |
| | | II | LTM | $CI \vee M \rightarrow CO$<br>$M \rightarrow AT$ | Load AC or T, as specified, from the M-bus. Force CO to one if the result is non-zero. Used to load the specified register from memory and test for a zero result. Also used to AND the K-bus with the M-bus for masking and, optionally, testing for a zero result. |
| 5 | III | XX | – | $CI \vee (AT \wedge K) \rightarrow CO$<br>$K \wedge AT \rightarrow AT$ | Logically AND the K-bus with AC or T, as specified. Deposit the result in the specified register. Logically OR CI with the word-wise OR of the result. Place the value of the carry OR on CO. |
| | | OO | CLA | $CI \rightarrow CO$ $0 \rightarrow AT$ | Clear AC or T, as specified, to all 0's. Force CO to CI. Used to clear the specified register and force CO to CI. |
| | | II | TZA | $CI \vee AT \rightarrow CO$<br>$AT \rightarrow AT$ | Force CO to one if AC or T, as specified, is non-zero. Used to test the specified register for zero. Also used to AND the K-bus to the specified register for masking and, optionally, testing for a zero result. |

## FUNCTION DESCRIPTION (CONT'D)

| F GROUP | R GROUP | K BUS | NAME | EQUATION | DESCRIPTION |
|---|---|---|---|---|---|
| 6 | I | XX | – | $CI \vee (AC \wedge K) \rightarrow CO$<br>$R_n \vee (AC \wedge K) \rightarrow R_n$ | Logically OR CI with the word-wise OR of the logical AND of AC and the K-bus. Place the result of the carry OR on CO. Logically OR $R_n$ with the logical AND of AC and the K-bus. Deposit the result in $R_n$. |
| | | OO | NOP | $CI \rightarrow CO$ $R_n \rightarrow R_n$ | Force CO to CI. Used as a null operation or to force CO to CI. |
| | | II | ORR | $CI \vee AC \rightarrow CO$<br>$R_n \vee AC \rightarrow R_n$ | Force CO to one if AC is non-zero. Logically OR AC with $R_n$. Deposit the result in $R_n$. Used to OR the accumulator to a register and, optionally, test the previous accumulator value for zero. |
| 6 | II | XX | – | $CI \vee (AC \wedge K) \rightarrow CO$<br>$M \vee (AC \wedge K) \rightarrow AT$ | Logically OR CI with the word-wise OR of the logical AND of AC and the K-bus. Place the carry OR on CO. Logically OR the M-bus, with the logical AND of AC and the K-bus. Deposit the final result in AC or T. |
| | | OO | LMF | $CI \rightarrow CO$ $M \rightarrow AT$ | Load AC or T, as specified, from the M-bus. Force CO to CI. Used to load the specified register with memory data and force CO to CI. |
| | | II | ORM | $CI \vee AC \rightarrow CO$<br>$M \vee AC \rightarrow AT$ | Force CO to one if AC is non-zero. Logically OR the M-bus with AC. Deposit the result in AC or T, as specified. Used to OR M-bus with the AC and, optionally, test the previous value of AC for zero. |
| 6 | III | XX | – | $CI \vee (I \wedge K) \rightarrow CO$<br>$AT \vee (I \wedge K) \rightarrow AT$ | Logically OR CI with the word-wise OR of the logical AND of the I-bus and the K-bus. Place the carry OR on CO. Logically AND the K-bus with the I-bus. Logically OR the result with AC or T, as specified. Deposit the final result in the specified register. |
| | | OO | NOP | $CI \rightarrow CO$ $R_n \rightarrow R_n$ | Force CO to CI. Used as a null operation or to force CO to CI. |
| | | II | ORI | $CI \vee I \rightarrow CO$<br>$I \vee AT \rightarrow AT$ | Force CO to one if the data on the I-bus is non-zero. Logically OR the I-bus to AC or T, as specified. Deposit the result in the specified register. Used to OR I-bus data with the specified register and, optionally, test the I-bus data for zero. |

## FUNCTION TRUTH TABLE

| FUNCTION GROUP | $F_6$ | $F_5$ | $F_4$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 |
| 4 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 |

| REGISTER GROUP | REGISTER | $F_3$ | $F_2$ | $F_1$ | $F_0$ |
|---|---|---|---|---|---|
| I | $R_0$ | 0 | 0 | 0 | 0 |
| | $R_1$ | 0 | 0 | 0 | 1 |
| | $R_2$ | 0 | 0 | 1 | 0 |
| | $R_3$ | 0 | 0 | 1 | 1 |
| | $R_4$ | 0 | 1 | 0 | 0 |
| | $R_5$ | 0 | 1 | 0 | 1 |
| | $R_6$ | 0 | 1 | 1 | 0 |
| | $R_7$ | 0 | 1 | 1 | 1 |
| | $R_8$ | 1 | 0 | 0 | 0 |
| | $R_9$ | 1 | 0 | 0 | 1 |
| | T | 1 | 1 | 0 | 0 |
| | AC | 1 | 1 | 0 | 1 |
| II | T | 1 | 0 | 1 | 0 |
| | AC | 1 | 0 | 1 | 1 |
| III | T | 1 | 1 | 1 | 0 |
| | AC | 1 | 1 | 1 | 1 |

| SYMBOL | MEANING |
|---|---|
| I, K, M | Data on the I, K, and M busses, respectively |
| CI, LI | Data on the carry input and left input, respectively |
| CO, RO | Data on the carry output and right output, respectively |
| $R_n$ | Contents of register n including T and AC (R-Group I) |
| AC | Contents of the accumulator |
| AT | Contents of AC or T, as specified |
| MAR | Contents of the memory address register |
| L, H | As subscripts, designate low and high order bit, respectively |
| + | 2's complement addition |
| – | 2's complement subtraction |
| $\wedge$ | Logical AND |
| $\vee$ | Logical OR |
| $\overline{\oplus}$ | Exclusive-NOR |
| $\rightarrow$ | Deposit into |

NOTE:
1. 2's complement arithmetic adds 111 . . . 11 to perform subtraction of 000 . . . 01.

## FUNCTION DESCRIPTION (CONT'D)

| F GROUP | R GROUP | K BUS | NAME | EQUATION | DESCRIPTION |
|---|---|---|---|---|---|
| 7 | I | XX | — | $CI \vee (R_n \wedge AC \wedge K) \rightarrow CO$<br>$R_n \bar{\circ} (AC \wedge K) \rightarrow R_n$ | Logically OR CI with the word-wise OR of the logical AND of $R_n$ and AC and the K-bus. Place the carry OR on CO. Logically AND the K-bus with AC. Exclusive-NOR the result with $R_n$. Deposit the final result in $R_n$. |
| | | OO | CMR | $CI \rightarrow CO$ $\quad \overline{R_n} \rightarrow R_n$ | Complement the contents of $R_n$. Force CO to CI. |
| | | II | XNR | $CI \vee (R_n \wedge AC) \rightarrow CO$<br>$R_n \bar{\circ} AC \rightarrow R_n$ | Force CO to one if the logical AND of AC and $R_n$ is non-zero. Exclusive-NOR AC with $R_n$. Deposit the result in $R_n$. Used to exclusive-NOR the accumulator with a register. |
| 7 | II | XX | — | $CI \vee (M \wedge AC \wedge K) \rightarrow CO$<br>$M \bar{\circ} (AC \wedge K) \rightarrow AT$ | Logically OR CI with the word-wise OR of the logical AND of AC and the K-bus and M-bus. Place the carry OR on CO. Logically AND the K-bus with AC. Exclusive NOR the result with the M-bus. Deposit the final result in AC or T. |
| | | OO | LCM | $CI \rightarrow CO$ $\quad \overline{M} \rightarrow AT$ | Load the complement of the M-bus into AC or T, as specified. Force CO to CI. |
| | | II | XNM | $CI \vee (M \wedge AC) \rightarrow CO$<br>$M \bar{\circ} AC \rightarrow AT$ | Force CO to one if the logical AND of AC and the M-bus is non-zero. Exclusive-NOR AC with the M-bus. Deposit the result in AC or T, as specified. Used to exclusive-NOR memory data with the accumulator. |
| 7 | III | XX | — | $CI \vee (AT \wedge I \wedge K) \rightarrow CO$<br>$AT \bar{\circ} (I \wedge K) \rightarrow AT$ | Logically OR CI with the word-wise OR of the logical AND of the specified register and the I-bus and K-bus. Place the carry OR on CO. Logically AND the K-bus with the I-bus. Exclusive-NOR the result with AC or T, as specified. Deposit the final result in the specified register. |
| | | OO | CMA | $CI \rightarrow CO$ $\quad \overline{AT} \rightarrow AT$ | Complement AC or T, as specified. Force CO to CI. |
| | | II | XNI | $CI \vee (AT \wedge I) \rightarrow CO$<br>$I \bar{\circ} AT \rightarrow AT$ | Force CO to one if the logical AND of the specified register and the I-bus is non-zero. Exclusive-NOR AC with the I-bus. Deposit the result in AC or T, as specified. Used to exclusive-NOR input data with the accumulator. |

## FUNCTION TRUTH TABLE

| FUNCTION GROUP | $F_6$ | $F_5$ | $F_4$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 |
| 4 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 |

| REGISTER GROUP | REGISTER | $F_3$ | $F_2$ | $F_1$ | $F_0$ |
|---|---|---|---|---|---|
| I | $R_0$ | 0 | 0 | 0 | 0 |
| | $R_1$ | 0 | 0 | 0 | 1 |
| | $R_2$ | 0 | 0 | 1 | 0 |
| | $R_3$ | 0 | 0 | 1 | 1 |
| | $R_4$ | 0 | 1 | 0 | 0 |
| | $R_5$ | 0 | 1 | 0 | 1 |
| | $R_6$ | 0 | 1 | 1 | 0 |
| | $R_7$ | 0 | 1 | 1 | 1 |
| | $R_8$ | 1 | 0 | 0 | 0 |
| | $R_9$ | 1 | 0 | 0 | 1 |
| | T | 1 | 1 | 0 | 0 |
| | AC | 1 | 1 | 0 | 1 |
| II | T | 1 | 0 | 1 | 0 |
| | AC | 1 | 0 | 1 | 1 |
| III | T | 1 | 1 | 1 | 0 |
| | AC | 1 | 1 | 1 | 1 |

| SYMBOL | MEANING |
|---|---|
| I, K, M | Data on the I, K, and M busses, respectively |
| CI, LI | Data on the carry input and left input, respectively |
| CO, RO | Data on the carry output and right output, respectively |
| $R_n$ | Contents of register n including T and AC (R-Group I) |
| AC | Contents of the accumulator |
| AT | Contents of AC or T, as specified |
| MAR | Contents of the memory address register |
| L, H | As subscripts, designate low and high order bit, respectively |
| + | 2's complement addition |
| − | 2's complement subtraction |
| $\wedge$ | Logical AND |
| $\vee$ | Logical OR |
| $\bar{\circ}$ | Exclusive-NOR |
| $\rightarrow$ | Deposit into |

NOTE:
1. 2's complement arithmetic adds 111 . . . 11 to perform subtraction of 000 . . . 01.

## ABSOLUTE MAXIMUM RATINGS*

| | |
|---|---|
| Temperature Under Bias | 0°C to +70°C |
| Storage Temperature | −65°C to +160°C |
| All Output and Supply Voltages | −0.5V to +7V |
| All Input Voltages | −1.0V to +5.5V |
| Output Currents | 100mA |

*COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum ratings for extended periods may effect device reliability.