ARHITEKTURA RAČUNARA
(pregled principa i evolucije)

Miroslav Hajduković
Žarko Živanov

NOVI SAD, 2019.

PREDGOVOR

Cilj ove knjige je da stvori funkcionalno zaokruženu sliku o radu računara. Zbog toga je akcenat stavljen na pregled celine, a ne na razradu detalja. Zato se knjiga zaustavlja na logičkom niovu i ne upušta se u razmatranje digitalnih sklopova, koji su prisutni ispod tog nivoa. Znači, računar se posmatra sa stanovišta njegovog korišćenja (iz ugla programera), a ne sa stanovišta njegovog pravljenja (iz ugla projektanta).

Za izlaganje suštinskih principa funkcionisanja računara koristi se hipotetski računar KONCEPT. Njegov razvoj je vođen idejom da se na što jednostavniji način podrži izvršavanje programa, koji su izraženi procedurnim programskim jezicima. Izlaganje kreće od korisničkog opisa procesora, nastavlja se primerima njegovog korišćenja na asemblerskom nivou i završava se izlaganjem osnova funkcionisanja svih delova hipotetskog računara KONCEPT.

Da čitalac ove knjige ne bi ostao samo na poznavanju principa funkcionisanja hipotetskog računara, izložena je i evolucija elektronskih računara. Posebna pažnja je posvećena karakteristikama računara u pojedinim evolucionim fazama i faktorima koji su usmeravali ovu evoluciju. Ukazano je na računare koji su izvršili značajan uticaj na tržište i čija upotreba nije bila ograničena samo na pojedina područja primene.

Ovo izdanje knjige je nastalo ispravljanjem grešaka koje su uočene u tekstu njenog prethodnog izdanja, kao i pojašnjavanjem i proširenjem delova teksta koji su bili nedovoljno razumljivi studentima. U tekst knjige su uključene glave 14. i 16., čiji autor je Žarko Živanov. On je obavio i tehničku obradu knjige.

SADRŽAJ

1. UVOD 1
1.1. POJAM ARHITEKTURE RAČUNARA 1
1.2. MODEL RAČUNARA 1
PRIMER UPOTREBE PROGRAMSKOG JEZIKA C 1
OSOBINE PROCEDURNIH PROGRAMSKIH JEZIKA 2
MEMORIJA 2
PROCESOR 3
ADRESIRANJE 3
SKUP NAREDBI 4
MAŠINSKE I ASEMBLERSKE NAREDBE 4
1.3. FIZIČKA OSNOVA MODELA RAČUNARA 5
1.4. PITANJA 8

2. BROJNI SISTEMI I PREDSTAVE BROJEVA 9
2.1. ARITMETIKA OGRANIČENOG BROJA CIFARA 9
KOMPLEMENT 10 PREDSTAVA OZNAČENIH CELIH BROJEVA 9
IZLAZAK VAN OPSEGA 11
POREĐENJE CELIH BROJEVA 12
VIŠESTRUKA PRECIZNOST 13
ZAKONI ARITMETIKE 13
2.2. PREDSTAVLJANJE REALNIH BROJEVA 14
ARITMETIKA MAŠINSKE NORMALIZOVANE FORME 15
2.3. BINARNI BROJNI SISTEM 16
POSTUPAK KONVERZIJE BROJEVA 16
2.4. ARITMETIKA OGRANIČENOG BROJA CIFARA U BINARNOM BROJNOM SISTEMU 19
KOMPLEMENT 2 PREDSTAVA OZNAČENIH CELIH BROJEVA 19
IZLAZAK VAN OPSEGA 20
POREĐENJE CELIH BROJEVA 22
VIŠESTRUKA PRECIZNOST 24
ZAKONI ARITMETIKE 24
2.5. PREDSTAVLJANJE VREDNOSTI REALNOG TIPA U BINARNOM BROJNOM SISTEMU 24
VIŠEZNAČNA INTERPRETACIJA CELIH BROJEVA 25
2.6. PITANJA 26

3. ASEMBLERSKO PROGRAMIRANJE 27
3.1. NIVOI PROGRAMIRANJA 27
3.2. ARHITEKTURA NAREDBI PROCESORA KONCEPT 28
NAREDBE ZA CELOBROJNU ARITMETIKU 28
NAREDBE ZA RUKOVANJE BITIMA 31
NAREDBE PREBACIVANJA 33
UPRAVLJAČKE NAREDBE 33
3.3. ASEMBLERSKI JEZIK KONCEPT 35
3.4. PRIMERI ASEMBLERSKIH PROGRAMA 39

RAČUNANJE NAJVEĆEG ZAJEDNIČKOG DELIOCA 39
IZLAZAK VAN OPSEGA KOD NEOZNAČENIH CELIH BROJEVA 40
IZLAZAK VAN OPSEGA KOD OZNAČENIH CELIH BROJEVA 40
RUKOVANJE MAŠINSKOM NORMALIZOVANOM FORMOM 41
RUKOVANJE LOGIČKIM VREDNOSTIMA 42
RAČUNANJE VREDNOSTI CELOBROJNOG IZRAZA 43
RUKOVANJE NIZOVIMA 43
RUKOVANJE SLOGOVIMA 44
3.5. POTPROGRAM 45
ASEMBLERSKI OBLIK POTPROGRAMA 46
3.6. MAKRO 49
OPIS MAKRO DEFINICIJA I MAKRO POZIVA 51
PRIMERI MAKRO DEFINICIJA I MAKRO POZIVA 52
MAKRO DEFINICIJA SA MAKRO DEFINICIJOM 53
USLOVNO ASEMBLIRANJE 54
3.7. STEK 55
FREJM 57
3.8. PITANJA 60
4. MEMORIJA I PROCESOR RAČUNARA KONCEPT 61
4.1. ORGANIZACIJA MEMORIJE RAČUNARA KONCEPT 61
DEKODIRANJE ADRESA 62
PRINCIPIJELNI IZGLED MEMORIJE SA 4 LOKACIJE 63
4.2. KODIRANJE I MAŠINSKI FORMATI NAREDBI PROCESORA KONCEPT 64
KODOVI 1. TIPA NAREDBI 66
KODOVI 2. TIPA NAREDBI 67
KODOVI 3. TIPA NAREDBI 67
KODOVI 4. TIPA NAREDBI 67
KODOVI 5. TIPA NAREDBI 68
KODOVI 6. TIPA NAREDBI 68
KODOVI 7. TIPA NAREDBI 69
KODOVI 8. TIPA NAREDBI 69
KODOVI 9. TIPA NAREDBI 69
KODOVI 10. TIPA NAREDBI 70
KODOVI 11. TIPA NAREDBI 70
KODOVI 12. TIPA NAREDBI 71
KODOVI 13. TIPA NAREDBI 71
KODOVI 14. TIPA NAREDBI 72
KODOVI 15 TIPA NAREDBI 72
4.3. ORGANIZACIJA PROCESORA KONCEPT 72
4.4. UPRAVLJANJE PROCESOROM KONCEPT 76
MIKRO-PROGRAMI OBAVLJANJA 77
INICIJALNI MIKRO-PROGRAM 83
4.5. UPRAVLJAČKA JEDINICA PROCESORA KONCEPT 84
4.6. UPRAVLJANJE PREKIDAČIMA IZ UPRAVLJAČKE JEDINICE PROCESORA KONCEPT 88

4.7. UPRAVLJANJE PREKIDAČIMA IZVAN UPRAVLJAČKE JEDINICE PROCESORA KONCEPT 89
4.8. MAŠINSKI OBLICI MIKRO-PROGRAMA 93
4.9. RAZMATRANJE RADA PROCESORA KONCEPT 101
4.10. PITANJA 102

5. RAČUNAR KONCEPT 104
5.1. ORGANIZACIJA RAČUNARA KONCEPT 104
5.2. ULAZNI I IZLAZNI UREĐAJI RAČUNARA KONCEPT 106
KOMANDNI JEZIK 106
TASTATURA 106
EKRAN 108
RAČUNAR KONCEPT SA ZNAKOVNIM ULAZOM I IZLAZOM 110
STANDARDNI ZNAKOVNI KODOVI 111
ZNAKOVNA I INTERNA PREDSTAVA CELIH BROJEVA 112
ZNAKOVNA INTERAKCIJA KORISNIKA I RAČUNARA 113
BIOS 115
5.3. VRSTE MEMORIJE 116
5.4. KODOVI ZA OTKRIVANJE I POPRAVKU GREŠAKA 121
5.5. OPERATIVNI SISTEM 123
STRUKTURA OPERATIVNOG SISTEMA 124
INTERPRETER KOMANDi OPERATIVNOG SISTEMA 125
SISTEMSKI PROGRAMI 126
ODNOS *BIOS*-A I OPERATIVNOG SISTEMA 127
5.6. PREKLJUČIVANJE 127
5.7. PREKID 130
MEHANIZAM PREKIDA 131
IZVEDBA PREKIDA 132
ODNOS OBRADE PREKIDA I PREKLJUČIVANJA 136
ORGANIZACIJA DRAJVERA TERMINALA 137
ORGANIZACIJA DRAJVERA DISKA 138
USKLAĐIVANJE RADA KONTROLERA I UREĐAJA 138
5.8. SABIRNICA 139
5.9. VIŠEKORISNIČKI RAD 142
LOGIČKI I FIZIČKI ADRESNI PROSTORI 142
PRETVARANJE LOGIČKE ADRESE U FIZIČKU 143
IZUZETAK 145
PRIVILEGOVANI I NEPRIVILEGOVANI REŽIM RADA PROCESORA 146
IZVEDBA SISTEMSKIH POZIVA 147
5.10. PITANJA 147

6. SISTEMSKI PROGRAMI 150
6.1. EDITOR 150
6.2. ASEMBLER 151
6.3. MAKRO PRETPROCESOR 155
6.4. LINKER 157
PROBLEM RELOKACIJE 157

RELATIVNO ADRESIRANJE ... 159
PROBLEM SPOLJAŠNJIH REFERENCI ... 160
OBRAZOVANJE IZVRŠNE SEKVENCE ... 162
6.5. LOUDER ... 165
6.6. DIBAGER ... 165
6.7. PITANJA ... 166
7. EVOLUCIJA ARHITEKTURE RAČUNARA ... 168
7.1. PRECIZIRANJE POJMA ARHITEKTURE RAČUNARA ... 168
7.2. POKRETAČI RAZVOJA ARHITEKTURE RAČUNARA ... 169
7.3. PERIODI EVOLUCIJE ... 170
7.4. PITANJA ... 170
8. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 1950. GODINE ... 171
8.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 1950. GODINE ... 171
ARHITEKTURA RAČUNARA PRVE GENERACIJE ... 172
MANE RAČUNARA PRVE GENERACIJE ... 173
8.2. PITANJA ... 173
9. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 1960. GODINE ... 174
9.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 1960. GODINE ... 174
DIGITALNA KOLA ... 174
RADNA MEMORIJA ... 175
KONTROLERI ... 176
ARHITEKTURA NAREDBI ... 177
PROGRAMSKI JEZICI VISOKOG NIVOA ... 178
MEMORIJSKA HIJERARHIJA ... 179
OPERATIVNI SISTEMI ... 179
MANE RAČUNARA DRUGE GENERACIJE ... 180
9.2. PITANJA ... 180
10. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 1970. GODINE ... 181
10.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 1970. GODINE ... 181
INTEGRISANA KOLA ... 181
MAGNETNI DISKOVI ... 181
VELIKI I MINI RAČUNARI ... 182
FAMILIJE RAČUNARA ... 182
OTVORENA ARHITEKTURA ... 182
PRODUKTIVNOST PROGRAMERA ... 183
ARHITEKTURA NAREDBI ... 184
MIKRO-PROGRAMIRANJE ... 184

PROMENLJIVI FORMATI MAŠINSKIH NAREDBI 186
ADRESIRANJA 186
ORGANIZACIJA RADNE MEMORIJE 186
MULTIPROGRAMIRANJE 187
IDEJA VIRTUELNE MEMORIJE 188
IDEJA SKRIVENE MEMORIJE 194
ODNOS VIRTUELNE I SKRIVENE MEMORIJE 196
OPERATIVNI SISTEM I VIRTUELNA MAŠINA 197
MANA RAČUNARA TREĆE GENERACIJE 197
10.2. ISHODI EVOLUCIJE ARHITEKTURE RAČUNARA U PERIODU OKO 1970. GODINE 197
RADNA MEMORIJA 197
ARHITEKTURA NAREDBI ZA *IBM SYSTEM/360* 198
ARHITEKTURA NAREDBI ZA *DEC PDP11* 200
OSOBINE MAGNETNOG DISKA 202
ORGANIZACIJA SABIRNICE 203
ORGANIZACIJA ASOCIJATIVNE MEMORIJE 206
SKRIVENA MEMORIJA 209
VIRTUELNA MEMORIJA 211
MEMORIJSKA HIJERARHIJA 214
PROBLEM SINHRONIZACIJE 215
10.3. PITANJA 216
11. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 1980. GODINE 219
11.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 1980. GODINE 219
POLUPROVODNIČKE MEMORIJE 219
MIKRO-RAČUNARI 220
PERSONALNI RAČUNARI 223
RAČUNARSKE MREŽE 223
SUPER-RAČUNARI 224
11.2. ISHODI EVOLUCIJE ARHITEKTURE RAČUNARA U PERIODU OKO 1980. GODINE 230
ARHITEKTURA NAREDBI ZA *DEC VAX11/780* 230
ARHITEKTURA NAREDBI ZA *Intel 8086* 231
SEGMENTNA ORGANIZACIJA RADNE MEMORIJE 233
ARHITEKTURA NAREDBI ZA *Intel 80386* 235
PRINCIP RADA MIŠA 237
OSOBINE GRAFIČKIH TERMINALA 238
PRINCIPI RADA LOKALNIH MREŽA 239
VIŠEPROCESORSKI RAČUNARI SA ZAJEDNIČKOM SABIRNICOM 240
11.3. PITANJA 242
12. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 1990. GODINE 244

12.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 1990. GODINE 244
RISC ARHITEKTURA 244
GRANICE RASTA BRZINE I GUSTINE TRANZISTORA NA ČIPU 245
DOMETI PRIMENE PARALELIZMA 246
KONKURENTNO PROGRAMIRANJE 248
KLASIFIKACIJA RAČUNARA 250
OTPORNOST NA KVAROVE 255
RISC PROCESORI 255
12.2. ISHODI EVOLUCIJE ARHITEKTURE RAČUNARA U PERIODU OKO 1990. GODINE 259
IEEE 754 STANDARD ZA ARITMETIKU REALNIH BROJEVA 259
ARHITEKTURA NAREDBI ZA *MIPS* 260
INTEL PENTIUM PRO PROCESOR 261
EVOLUCIJA MASOVNE MEMORIJE 261
UTICAJ *RISC* PROCESORA NA VIRTUELNU MEMORIJU 263
SPOJNE MREŽE 264
BARIJERNA SINHRONIZACIJA 271
12.3. PITANJA 272

13. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 2000. GODINE 274

13.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 2000. GODINE 274
UGRAĐENI RAČUNARI 274
RADNE STANICE 274
SERVERI 275
PARALELIZAM UNUTAR PROCESORA 277
EVOLUCIJA OPTIČKIH DISKOVA 279
13.2. ISHODI EVOLUCIJE ARHITEKTURE RAČUNARA U PERIODU OKO 2000. GODINE 279
ARHITEKTURA NAREDBI ZA *INTEL ITANIUM* 279
13.3. PITANJA 279

14. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 2010. GODINE 281

14.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 2010. GODINE 281
PROCESORI 281
INTERNET 281
GRAFIČKE KARTICE 282
SUPER-RAČUNARI 282
PRENOSNI RAČUNARSKI UREĐAJI 282
14.2. ISHODI EVOLUCIJE ARHITEKTURE RAČUNARA U PERIODU OKO 2010. GODINE 283
ARHITEKTURA NAREDBI ZA AMD *ATHLON 64/OPTERON* 283

14.3. PITANJA 283

15. PROCENA OSOBINA RAČUNARA 284
15.1. NAČIN PROCENE OSOBINA RAČUNARA 284
15.2. PITANJA 286

16. SAVREMENI PERSONALNI RAČUNAR 287
16.1. PROCESOR 287
16.2. MATIČNA PLOČA 288
VEZNI (*BRIDGE*) ČIP 290
BIOS ČIP 290
SABIRNICE 291
16.3. RADNA MEMORIJA 294
16.4. GRAFIČKI ADAPTER 295
16.5. KUĆIŠTE, NAPAJANJE I SISTEM ZA HLAĐENJE 296

LITERATURA 298

INDEKS SLIKA 299

1. UVOD

1.1. POJAM ARHITEKTURE RAČUNARA

Arhitektura računara (*computer architecture*) se bavi problemima upotrebe i pravljenja računara.

Posmatrano sa stanovišta arhitekture računara, upotreba računara se svodi na njegovo programiranje, jer je namena računara da izvršava programe. Način programiranja zavisi od osobina **skupa naredbi** računara. Ovim osobinama se bavi **arhitektura naredbi** (*instruction set architecture*, skraćeno *ISA*).

Cilj pravljenja računara je ostvarenje ili **implementacija** (*implementation*) njegove arhitekture naredbi. Implementacija arhitekture naredbi obuhvata **organizaciju** (*organization*) i **izvedbu** (*hardware*) računara. Organizacija računara se bavi organizacionim komponentama koje obrazuju računar (njihovom namenom i funkcijom), kao i međusobnim odnosima ovih komponenti. Izvedba računara se bavi problemima proizvodnje pomenutih komponenti.

Pojam arhitekture računara obuhvata arhitekturu naredbi i njenu implementaciju, odnosno organizaciju i izvedbu računara.

Između arhitekture naredbi i njene implementacije postoji međuzavisnost, jer implementacija odražava i ograničava arhitekturu naredbi.

1.2. MODEL RAČUNARA

Na arhitekturu naredbi utiču programski jezici koji se koriste za programiranje računara. Za tržišno prihvaćene računare je karakteristično da su prilagođeni procedurnim (imperativnim) programskim jezicima. Njihov tipičan predstavnik je programski jezik C.

PRIMER UPOTREBE PROGRAMSKOG JEZIKA C

Upotreba programskog jezika C može da se ilustruje na primeru nalaženja najvećeg zajedničkog delioca (NZD) dva različita neoznačena broja a i b, za koje važi:

$$a == n \times NZD$$
$$b == m \times NZD$$

Za određivanje najvećeg zajedničkog delioca neoznačenih brojeva a i b potrebno je koeficijente n i m svesti na 1. To se može postići ponavljanjem oduzimanja:

$$|a-b| == |n-m| \times NZD$$

tako da se, nakon svakog oduzimanja, promenljivoj sa većom vrednošću dodeli

razlika. Precizan postupak nalaženja najvećeg zajedničkog delioca opisuje segment programa, izražen programskim jezikom C:

```
unsigned int a = 12;
unsigned int b = 10;
while (a!=b)
 if (a>b)
  a = a-b;
 else
  b = b-a;
```

Po izvršavanju prethodnog segmenta programa važi: a == b == NZD == 2 (radi jednostavnosti, opštost prethodnog segmenta programa je ograničena na određivanje najvećeg zajedničkog delioca neoznačenih brojeva 12 i 10).

OSOBINE PROCEDURNIH PROGRAMSKIH JEZIKA

Procedurni programski jezici omogućuju opisivanje obrada podataka koji pripadaju **celom**, **realnom**, **znakovnom** ili **logičkom** skupu. Ovi skupovi se nazivaju i **prosti tipovi**, jer se njihove vrednosti ne mogu raščlanjivati na prostije sastojke. Prosti tipovi se nazivaju i **osnovni tipovi** (*fundamental types*), jer predstavljaju osnovu za opisivanje svih obrada podataka. Za opisivanje obrada podataka koriste se operacije procedurnih programskih jezika koje omogućuju rukovanje vrednostima prostih tipova. U ovakve operacije spadaju **aritmetičke**, **relacione** i **logičke operacije**.

Opštost opisima obrada podataka daju **promenljive**. Svaku promenljivu karakterišu njeno ime, tip i vrednost. Promenljivoj se dodeljuje vrednost njenog tipa posredstvom **operacije dodele**. Za opštost opisa obrada podataka su važne i **upravljačke operacije**. Zahvaljujući njima redosled obavljanja operacija nije samo sekvencijalni, nego i alternativni i repetitivni.

Promenljive, vrednosti prostih tipova i operacije (upravljačke i one za rukovanje promenljivim i vrednostima prostih tipova) predstavljaju osnovne elemente procedurnih programskih jezika i ujedno daju uopšteni opis računara. Ovome opisu odgovara **model računara** koji obuhvata **memoriju** i **procesor**. Ovaj model podržava izvršavanje programa koji su izraženi procedurnim programskim jezicima.

MEMORIJA

Memorija (*memory*) je sastavljena od niza **lokacija**. Lokacije omogućuju predstavljanje promenljivih prostih tipova, ako mogu da sadrže vrednosti prostih tipova (koje se dodeljuju ovim promenljivim). Svaka lokacija poseduje jednoznačnu (numeričku) oznaku ili **adresu** (brojevi 0, 1, 2, ...), po kojoj se razlikuje od drugih lokacija. Pored adrese, lokacija može da poseduje i posebnu (simboličku) oznaku ili **labelu** (*label*). Kada postoji, labela se koristi umesto adrese. Labela odgovara imenu promenljive koju predstavlja labelirana lokacija.

Za lokacije je važna osobina univerzalnosti, koja podrazumeva da svaka lokacija može da sadrži vrednost bilo kog prostog tipa i adresu. Univerzalnost lokacija se postiže **kodiranjem** (predstavljanjem) svih vrednosti realnog, znakovnog i logičkog tipa vrednostima celog tipa (podskupom celih brojeva, koji pripadaju celom tipu). Zbog univerzalnosti lokacija, javlja se problem višeznačne interpretacije sadržaja lokacija.

Preuzimanje vrednosti promenljive, kao i dodela vrednosti promenljivoj, podrazumeva pristupanje lokacijama, radi **čitanja** (preuzimanja), ili **pisanja** (izmene) sadržaja lokacija.

PROCESOR

Procesor (*processor*) je sastavljen od **sklopova** i **registara**. Sklopovi omogućuju izvršavanje pojedinih operacija (namenjenih za rukovanje vrednostima prostih tipova), a registri sadrže vrednosti prostih tipova koje se obrađuju u toku izvršavanja pojedinih operacija. Registri se razlikuju od memorijskih lokacija po tome što se nalaze u procesoru i po tome što nisu označeni adresom ili labelom, nego posebnom oznakom.

Pre izvršavanja operacije, procesor preuzima **oznaku operacije** i njene **operande**. Operandi su ili vrednosti prostih tipova na koje se operacija odnosi, ili adrese lokacija čiji sadržaji se čitaju ili pišu u toku izvršavanja operacije. Oznaka operacije određuje sklop u kome se izvršava dotična operacija. Oznaka operacije i operandi obrazuju **naredbu** (*instruction*) procesora. Procesor izvršava operacije u toku izvršavanja svojih naredbi. Redosled izvršavanja naredbi je određen programom.

ADRESIRANJE

Operandi se dele na **ulazne** i **izlazne**. Ulazni operandi određuju (ulazne) vrednosti, koje se obrađuju u toku izvršavanja naredbi. Izlazni operandi određuju (izlazne) lokacije, u koje se smeštaju rezultati izvršavanja naredbi.

Ulazni operand se naziva **neposredni** (*immediate*) operand, kada mu odgovara ulazna vrednost. Neposredni operandi su potrebni, radi konstanti. Ulazni operand se naziva **direktni** (*direct*) operand, kada mu odgovara adresa memorijske lokacije sa ulaznom vrednošću, odnosno, naziva se **registarski** (*register*) operand, kada mu odgovara oznaka registra sa ulaznom vrednošću. Izlazni operand se naziva direktni operand, kada mu odgovara adresa izlazne memorijske lokacije, odnosno, naziva se registarski operand, kada mu odgovara oznaka izlaznog registra. Direktni i registarski operandi su potrebni, radi promenljivih prostih tipova. Ulazni operand se naziva **posredni** (*register indirect*) operand, kada mu odgovara oznaka registra sa adresom memorijske lokacije sa ulaznom vrednošću. Izlazni operand se naziva posredni operand, kada mu odgovara oznaka registra sa adresom izlazne memorijske lokacije. Posredni operandi su potrebni, radi pokazivačkih promenljivih. Ulazni operand se naziva **indeksni** (*indexed*) operand, kada mu

odgovaraju posebna vrednost - indeks i oznaka registra, čiji sadržaj u sumi sa indeksom daje adresu memorijske lokacije sa ulaznom vrednošću. Izlazni operand se naziva indeksni operand, kada mu odgovaraju posebna vrednost - indeks i oznaka registra, čiji sadržaj u sumi sa indeksom daje adresu izlazne memorijske lokacije. Indeksni operandi su potrebni, na primer, radi promenljivih složenih tipova (niz ili slog). Određivanje ulaznih vrednosti, odnosno izlaznih lokacija, se naziva i **adresiranje**, pa tako postoje **neposredno**, **direktno**, **registarsko**, **posredno** i **indeksno adresiranje**.

SKUP NAREDBI

Skup naredbi procesora se može svesti na podskup aritmetičkih naredbi, kao i na relacione, logičke, upravljačke i naredbe prebacivanja. Od aritmetičkih naredbi dovoljno je da procesor podrži celobrojno sabiranje i oduzimanje, jer se ostale aritmetičke naredbe mogu izraziti pomoću celobrojnog sabiranja i oduzimanja. Zahvaljujući kodiranju vrednosti realnog, znakovnog i logičkog tipa vrednostima celog tipa, relacione naredbe se svode na celobrojno oduzimanje prvog operanda (PO) relacije od drugog (DO) i zaključivanje o relaciji na osnovu vrednosti i predznaka razlike, jer:

1. relacija DO == PO važi, ako je razlika 0,
2. relacija DO != PO važi, ako je razlika različita od 0,
3. relacija DO < PO važi, ako je razlika negativna,
4. relacija DO >= PO važi, ako je razlika pozitivna ili 0,
5. relacija DO > PO važi, ako je razlika pozitivna, a
6. relacija DO <= PO važi, ako je razlika negativna ili 0.

Logičke naredbe obuhvataju logičko i, logičko ili i logičko ne, a upravljačke naredbe bezuslovnu ili uslovnu izmenu redosleda izvršavanja naredbi programa. Naredbe prebacivanja podržavaju dodelu vrednosti promenljivoj tako što omogućuju prebacivanje sadržaja između registara procesora i memorijskih lokacija.

MAŠINSKE I ASEMBLERSKE NAREDBE

Lokacije mogu da sadrže i naredbe, ako se oznake operacija naredbi (poput njihovih operanada) kodiraju celim brojevima. Ovi brojevi predstavljaju **kodove naredbi**. Kod naredbe sa operandima naredbe obrazuje **mašinsku naredbu**. **Mašinski format naredbe** propisuje kako se interpretiraju cifre celog broja koji predstavlja mašinsku naredbu. Simbolička predstava mašinske naredbe (u kojoj se koriste labele, kao i simboličke oznake za operaciju i operande) se naziva **asemblerska naredba**. Mašinske naredbe pripadaju **mašinskom jeziku**, a asemblerske naredbe pripadaju **asemblerskom jeziku**.

Izvršavanju asemblerskih naredbi obavezno prethodi njihovo prevođenje u mašinski oblik. Prevođenje **asemblerskog programa**, sastavljenog od asemblerskih

naredbi, u **mašinski program**, sastavljen od mašinskih naredbi, obavlja poseban program prevodilac, koji se naziva **asembler**.

Slično asembleru, prevođenje **izvornog programa**, sastavljenog od iskaza procedurnog programskog jezika, u asemblerski (mašinski) program obavlja poseban program prevodilac, koji se naziva **kompajler**.

1.3. FIZIČKA OSNOVA MODELA RAČUNARA

Za pravljenje prethodno opisanog modela računara dovoljna je fizička osnova koja omogućuje predstavljanje celih brojeva, jer su vrednosti prostih tipova i delovi naredbi predstavljeni (kodirani) celim brojevima. Cifre celih brojeva se mogu predstaviti raznim nivoima fizičkih veličina, kao što su električni napon ili jačina magnetnog polja. Praktični razlozi sugerišu korišćenje samo 2 nivoa fizičkih veličina, odnosno 2 nivoa signala. To je dovoljno: za prikaz cifara binarnog brojnog sistema (0 : nula, 1 : jedan), ali i za prikaz logičkih vrednosti (0 : netačno, 1 : tačno). Podudarnost cifara binarnog brojnog sistema i logičkih vrednosti omogućuje da se aritmetičke operacije za binarni brojni sistem opišu logičkim funkcijama, ako se binarne cifre interpretiraju kao logičke vrednosti. To potvrđuje tablica sabiranja za binarni brojni sistem (Slika 1.3.1).

a	b	a+b	Prenos
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Slika 1.3.1 Tablica sabiranja za binarni brojni sistem

Ako se cifre iz tablice sabiranja (Slika 1.3.1) interpretiraju kao logičke vrednosti, a prve dve kolone kao argumenti logičke funkcije, tada treća kolona ove tablice opisuje logičku funkciju isključivo ili (*exclusive or*):

a^b

(oznaka ^ predstavlja operator programskog jezika C za logičku operaciju isključivo ili). Četvrta kolona ove tablice opisuje logičku funkciju i (*and*):

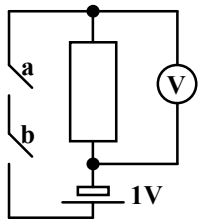
a&b

(oznaka & predstavlja operator programskog jezika C za logičku operaciju i).

Prethodno dozvoljava da se računar napravi kao fizički uređaj koji memoriše logičke vrednosti i obavlja logičke funkcije. Tako, voltmetar iz električnog kola (Slika 1.3.2) pokazuje vrednost logičke funkcije i:

a&b

ako stanja prekidača (0 : otvoren, 1 : zatvoren) zavise od vrednosti logičkih promenljivih a i b.

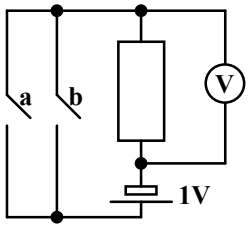


Slika 1.3.2 Električno kolo - ekvivalent logičke funkcije i

Voltmetar iz električnog kola (Slika 1.3.3) pokazuje vrednost logičke funkcije ili (*or*):

a|b

ako stanja prekidača (0 : otvoren, 1 : zatvoren) zavise od vrednosti logičkih promenljivih a i b (oznaka | predstavlja operator programskog jezika C za logičku operaciju ili).

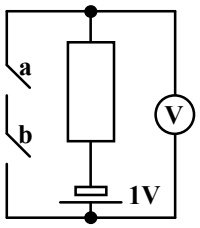


Slika 1.3.3 Električno kolo - ekvivalent logičke funkcije ili

Električna kola omogućuju i ostvarenje logičke negacije. Tako, voltmetar iz električnog kola (Slika 1.3.4) pokazuje vrednost logičke funkcije negirano i (*nand*):

~(a&b)

ako stanja prekidača (0 : otvoren, 1 : zatvoren) zavise od vrednosti logičkih promenljivih a i b (oznaka ~ predstavlja operator programskog jezika C za logičku operaciju negacije).



Slika 1.3.4 Električno kolo - ekvivalent logičke funkcije negirano i

Pomoću logičke funkcije negirano i se mogu izraziti sve ostale logičke funkcije. Ako se kao operator logičke funkcije negirano i odabere oznaka ○:

$$a \circ b \equiv \sim(a \& b)$$

tada sledeći primeri pokazuju kako se pojedine logičke funkcije mogu izraziti pomoću logičke funkcije negirano i:

$$a \,\&\, b \;==\; (a \circ b) \circ (a \circ b)$$
$$a \mid b \;==\; (a \circ a) \circ (b \circ b)$$
$$a \,\hat{}\, b \;==\; ((a \circ a) \circ b) \circ (a \circ (b \circ b))$$
$$a \equiv b \;==\; (a \circ b) \circ ((a \circ a) \circ (b \circ b))$$

Iz prethodnog sledi da se za ostvarenje svih logičkih funkcija može koristiti jedan isti tip električnog kola.

U električnim kolima ulogu prekidača ima tranzistor. Pored ulaznog i izlaznog izvoda, koji su prikazani u sastavu svakog od prekidača na crtežima prethodnih električnih kola, tranzistor ima i treći, upravljački izvod. On određuje provodnost tranzistora, odnosno, stanje prekidača. Na upravljački izvod tranzistora dovodi se, na primer, nivo signala koji odgovara vrednosti neke logičke promenljive, odnosno nivo signala koji odgovara vrednosti neke logičke funkcije, kao što je naponski nivo, prikazan na voltmetrima iz prethodnih električnih kola. Ako se u sastavu prekidača ne prikazuju upravljački izvodi (kao što je to urađeno, radi jednostavnosti, na crtežima prethodnih električnih kola), tada je za svaki prekidač neophodno navesti njegovu upravljačku logičku funkciju, koja se naziva i **prekidačka funkcija**. Podrazumeva se da njena vrednost određuje stanje prekidača, odnosno, da ta vrednost određuje nivo signala na upravljačkom izvodu tranzistora. Argumenti prekidačke funkcije su logičke vrednosti. Oni određuju njenu vrednost, pa, na taj način, omogućuju upravljanje prekidačima. Zato se ovakvi argumenti mogu nazvati **prekidački argumenti**. Na primer, ako logička funkcija:

$$f(a) == a$$

opisuje upravljanje nekim prekidačem, tada od njenog prekidačkog argumenta a zavisi da li je dotični prekidač otvoren ili zatvoren.

Lokacije (i memorijske lokacije i registri procesora) se prave kao nizovi ćelija, nazvanih **biti**. Svakom bitu odgovara poseban fizički uređaj, kao što je flipflop (*flipflop*), sposoban za memorisanje jedne logičke vrednosti, odnosno binarne cifre. Sve lokacije se sastoje od istog i ograničenog broja bita, organizovanih u **bajte** (*byte*) i **reči** (*word*). Ograničena veličina lokacija određuje raspon operanada, odnosno najveći i najmanji operand.

Lokacije se prave pomoću **sekvencijalnih kola** koja omogućuju pamćenje logičkih vrednosti, odnosno binarnih cifara, a sklopovi procesora se prave pomoću **kombinacionih** kola koja ostvaruju pojedine logičke funkcije. Za razliku od kombinacionog kola, čiji izlaz zavisi samo od njegovog ulaza, izlaz sekvencijalnog kola zavisi ne samo od njegovog ulaza, nego i od njegovog stanja.

Iako su principi, na kojima se zasniva rad računara, jednostavni, ipak nije jednostavno od principa stići do pouzdanih i jeftinih komponenti, podesnih za pravljenje praktično upotrebljivog računara. Problemi pravljenja praktično upotrebljivog računara izlaze van okvira ove knjige, jer je ona posvećena samo izgradnji funkcionalno zaokruženog modela računara. Ipak, namena ove knjige je da čitaoca pripremi i za suočavanje sa problemima koji se sreću pri pravljenju praktično upotrebljivog računara.

1.4. PITANJA

1. Šta obuhvata pojam arhitekture računara?
2. Koji prosti (osnovni) tipovi postoje?
3. Koji su osnovni elementi procedurnih programskih jezika?
4. Šta obuhvata model računara?
5. Šta karakteriše memoriju?
6. Kako se može interpretirati sadržaj lokacije memorije?
7. Šta karakteriše procesor?
8. Koja adresiranja postoje?
9. Šta ulazi u sastav mašinske naredbe?
10. Šta ulazi u sastav asemblerske naredbe?
11. Kakav zadatak ima asembler?
12. Kakav zadatak ima kompajler?
13. Šta omogućuje podudarnost cifara binarnog brojnog sistema i logičkih vrednosti?
14. Koja logička funkcija opisuje zbir dve binarne cifre?
15. Šta određuje vrednost prekidačke funkcije?
16. Na šta utiče vrednost prekidačkog argumenta?
17. Šta omogućuju sekvencijalna kola?
18. Šta omogućuju kombinaciona kola?

2. BROJNI SISTEMI I PREDSTAVE BROJEVA

2.1. ARITMETIKA OGRANIČENOG BROJA CIFARA

Celi brojevi se čuvaju u lokacijama u pozicionoj predstavi (najmanje značajna cifra u krajnje desnoj poziciji, a najznačajnija cifra u krajnje levoj poziciji). Poziciona predstava celih brojeva olakšava obavljanje aritmetičkih operacija. Zahvaljujući njoj, postupak sabiranja i oduzimanja, za neoznačene cele brojeve, se svodi na direktnu primenu tablice sabiranja na parove korespondentnih cifara (s desna u levo).

KOMPLEMENT 10 PREDSTAVA OZNAČENIH CELIH BROJEVA

Za označene cele brojeve postupak sabiranja i oduzimanja komplikuje prisustvo predznaka. To ilustruje primer sabiranja označenih celih brojeva raznih predznaka. U ovom slučaju predznak rezultata je jednak predznaku sabirka sa većom apsolutnom vrednošću. Nakon određivanja predznaka, sledi potpisivanje sabiraka na ispravan način, radi oduzimanja apsolutne vrednosti manjeg sabirka od apsolutne vrednosti većeg sabirka. Dopisivanjem ovako dobijene razlike iza prethodno određenog predznaka nastaje traženi zbir.

Važno svojstvo pozicione predstave celih brojeva je da, za ograničen broj cifara, ona dozvoljava predstavljanje označenih celih brojeva kao neoznačenih. Na taj način se eliminišu komplikacije prilikom njihovog sabiranja i oduzimanja, jer se ne mora voditi računa o predznaku. To pokazuje sledeći primer:

$$68+(-57) == 68+(-57)+1000-1000 == 68+943-1000 == 1011-1000 == 11$$

Ako se u prethodnom primeru posmatraju samo 3 najmanje značajne cifre i ako se podrazumeva dodavanje i oduzimanje 10^3, tada se sabiranje označenih celih brojeva 68 i -57 može prikazati kao:

```
 068  (68)
+943  (-57+1000)
 011  (1011-1000)
```

Prenos sa najznačajnije pozicije se zanemaruje, jer se podrazumeva dodavanje i oduzimanje 10^3. Broj 943 je (trocifrena) **komplement** 10 **predstava** broja -57.

Iz prethodnog primera sledi da se određivanje n cifarske komplement 10 predstave negativnog celog broja svodi na njegovo sabiranje sa 10^n. Međutim, pomenuta komplement 10 predstava se može odrediti i na drugi način. On podrazumeva da se za svaku cifru brojnog sistema odredi njen **komplement**, odnosno cifra koja dopunjava posmatranu cifru do najveće cifre u brojnom sistemu.

Slika 2.1.1 sadrži komplemente cifara dekadnog brojnog sistema, čija je najveća cifra 9.

cifra	komplement	komentar
0	9	0+9==9
1	8	1+8==9
2	7	2+7==9
3	6	3+6==9
4	5	4+5==9
5	4	5+4==9
6	3	6+3==9
7	2	7+2==9
8	1	8+1==9
9	0	9+0==9

Slika 2.1.1 Cifre dekadnog brojnog sistema i njihovi komplementi

Formiranje komplement 10 predstave negativnog celog broja započinje dodavanjem ispred cifara apsolutne vrednosti ovog broja vodećih nula do ukupnog broja cifara. Tako nastane **potpuna** apsolutna vrednost negativnog celog broja. Zatim se zamenjuje svaka od cifara potpune apsolutne vrednosti negativnog celog broja komplementom dotične cifre. Postupak zamene cifara njihovim komplementima se zove **komplementiranje**. Komplementiranjem nastaje komplement 9 predstava negativnog celog broja. Uvećavanjem komplement 9 predstave negativnog celog broja za 1 nastaje komplement 10 predstava negativnog celog broja.

Komplement 10 predstava pozitivnog celog broja nastaje njegovim dopunjavanjem s leva vodećim nulama do ukupnog broja cifara.

U slučaju prethodnog primera, trocifrena komplement 10 predstava broja 68 je 068.

Trocifrena komplement 9 predstava broja -57 je 942. Ona nastaje komplementiranjem cifara broja 057 (odnosno komplementiranjem cifara potpune apsolutne vrednosti broja -57). Trocifrena komplement 10 predstava broja -57 je 943. Ona je za 1 veća od trocifrene komplement 9 predstave istog broja.

Važno je naglasiti da komplementiranjem cifara komplement 10 predstave negativnog celog broja i uvećanjem tako dobijenog broja za 1 nastaje potpuna apsolutna vrednost negativnog celog broja.

U komplement 10 predstavi označenih celih brojeva, sve cifre nose vrednost broja, ali najznačajnija cifra ima i dodatnu ulogu, jer ukazuje na predznak broja. Tako, za komplement 10 predstavu pozitivnih celih brojeva, najznačajnija cifra je uvek 0, a za komplement 10 predstavu negativnih celih brojeva, najznačajnija cifra je uvek 9.

Komplement predstava označenih celih brojeva pojednostavljuje procesor, jer omogućuje da se aritmetika označenih celih brojeva obavlja kao i aritmetika neoznačenih celih brojeva (znači, bez posmatranja predznaka brojeva). Tako sabiranje brojeva -7 i 2 može da bude obavljeno kao sabiranje njihovih trocifrenih komplement 10 predstava 993 i 002, uz direktnu primenu tablice sabiranja na parove korespondentnih cifara i bez razmatranja predznaka. Na ovaj način kao rezultat sabiranja dobije se 995, a to je trocifrena komplement 10 predstava broja -5.

U nastavku su prikazane neke vrednosti trocifrene komplement 10 predstave označenih celih brojeva (njima korespondentne vrednosti, sa predznakom, su navedene u zagradama):

099 (+99)
098 (+98)
097 (+97)
096 (+96)
...
002 (+2)
001 (+1)
000 (0)
999 (-1)
998 (-2)
...
903 (-97)
902 (-98)
901 (-99)
900 (-100)

U komplement 10 predstavi, broj negativnih celih brojeva je za jedan veći od broja pozitivnih celih brojeva.

Aritmetičke operacije označenih celih brojeva u komplement 10 predstavi daju označeni celi broj u komplement 10 predstavi. To ilustruje primer oduzimanja:

010 (+10)
-011 (+11)
999 (-1)

Prilikom obavljanja aritmetičkih operacija, prenos sa najznačajnije pozicije u komplement 10 predstavi celih brojeva se uvek zanemaruje.

IZLAZAK VAN OPSEGA

Za aritmetiku ograničenog broja cifara vezan je problem pojave cifara koje izlaze **van opsega**, uvedenog ograničavanjem broja posmatranih cifara. Kod

aritmetike neoznačenih celih brojeva, čije sve cifre nose samo vrednost, do izlaska van opsega dolazi nakon pojave prenosa sa najznačajnije pozicije rezultata:

```
 999
+999
 998
```

U prethodnom primeru se pojavio netačan rezultat, jer se prenos ne može prikazati u posmatrane 3 cifre.

Kod aritmetike označenih celih brojeva u komplement 10 predstavi, čija najznačajnija cifra pokazuje da li je reč o pozitivnom (0) ili negativnom (9) broju, pojava cifre različite od 0 ili 9 u najznačajnijoj poziciji rezultata ukazuje na izlazak van opsega:

```
 050          900
+050   ili   -001
 100          899
```

U prethodnim primerima na pojavu netačnog rezultata ukazuju neodgovarajuće cifre u njegovoj najznačajnijoj poziciji.

Važno je zapaziti da interpretacija rezultata zavisi od interpretacije brojeva. Ako se u pretposlednjem primeru celi brojevi interpretiraju kao označeni u trocifrenoj komplement 10 predstavi, tada je rezultat tačan (u ovom slučaju prenos se zanemaruje). Slično, ako se celi brojevi iz poslednjeg primera interpretiraju kao neoznačeni, tada je rezultat tačan, jer nema prenosa sa najznačajnije pozicije.

POREĐENJE CELIH BROJEVA

Kod određivanja relacija, izlazak van opsega pri celobrojnom oduzimanju prvog operanda (PO) relacije od drugog (DO) ne izaziva probleme, nego pomaže da se izvede ispravan zaključak o važenju relacije.

Za neoznačene cele brojeve:

1. relacija DO == PO važi, ako je rezultat oduzimanja 0,
2. relacija DO != PO važi, ako rezultat oduzimanja nije 0,
3. relacija DO < PO važi, ako se pri oduzimanju javi izlazak van opsega (u obliku prenosa),
4. relacija DO >= PO važi, ako se pri oduzimanju ne javi izlazak van opsega ili je rezultat 0,
5. relacija DO > PO važi, ako se pri oduzimanju ne javi izlazak van opsega i rezultat nije 0, a
6. relacija DO <= PO važi, ako se pri oduzimanju javi izlazak van opsega (u obliku prenosa) ili je rezultat 0.

Za označene cele brojeve:

1. relacija DO == PO važi, ako je rezultat oduzimanja 0,
2. relacija DO != PO važi, ako rezultat nije 0,
3. relacija DO < PO važi, ako je rezultat negativan (najznačajnija cifra 9) ili se pri oduzimanju izađe van opsega (najznačajnija cifra 8),
4. relacija DO >= PO važi, ako je rezultat pozitivan (najznačajnija cifra 0) ili se pri oduzimanju izađe van opsega (najznačajnija cifra 1) ili je rezultat 0,
5. relacija DO > PO važi, ako je rezultat pozitivan (najznačajnija cifra 0) ili se pri oduzimanju izađe van opsega (najznačajnija cifra 1), a
6. relacija DO <= PO važi, ako je rezultat negativan (najznačajnija cifra 9) ili se pri oduzimanju izađe van opsega (najznačajnija cifra 8) ili je rezultat 0.

Važenje prethodnog se može pokazati na primerima svih tipičnih odnosa koji se mogu pojaviti u nekoj relaciji. Za relaciju < sve tipične odnose ilustruju primeri dati tabelarno (Slika 2.1.2).

drugi operand	prvi operand	<	razlika (komplement 10 predstava)
88	99	da	088-099 == 989
-99	99	da	901-099 == 802
-19	19	da	981-019 == 962
-99	-88	da	901-912 == 989
99	88	ne	099-088 == 011
99	-99	ne	099-901 == 198
19	-19	ne	019-981 == 038
-88	-99	ne	912-901 == 011

Slika 2.1.2 Primeri važenja relacije <

VIŠESTRUKA PRECIZNOST

Aritmetika ograničenog broja cifara zahteva **višestruku preciznost**, kada se operacije odnose na vrednosti čiji broj cifara je veći od broja cifara lokacije. U ovom slučaju, delovi vrednosti se čuvaju u raznim lokacijama, a operacije se obavljaju za deo po deo ovih vrednosti (idući od manje značajnih ka značajnijim delovima vrednosti).

ZAKONI ARITMETIKE

Za aritmetiku ograničenog broja cifara ne važe zakoni aritmetike, jer, zbog mogućeg izlaska van opsega, rezultat zavisi od redosleda obavljanja operacija. To potkrepljuje sledeći primer za trocifrenu preciznost:

(200+800)-500 != 200+(800-500)

u kome su navedeni neoznačeni celi brojevi. U toku formiranja levog zbira dolazi do izlaska van opsega, pa je on u opštem slučaju različit od desnog zbira, u kome se ne javlja izlazak van opsega.

2.2. PREDSTAVLJANJE REALNIH BROJEVA

Potpuna poziciona predstava je nepraktična za realne brojeve, jer ona, na primer, podrazumeva da se u broju:

0.000000127

koristi 6 nula za određivanje položaja značajnih cifara iza tačke. Manje prostora zauzima **normalizovana forma**, čiji eksponent određuje položaj značajnih cifara:

1.27×10^{-7}

Još manje prostora je potrebno za **mašinsku normalizovanu formu** (*floating-point*), koja sadrži samo predznak realnog broja, njegovu **frakciju** (*fraction, significand*) i **podešeni eksponent** (*biased exponent*, *excess*). Frakcija obuhvata značajne cifre broja. Podrazumeva se da iza prve od značajnih cifara dolazi decimalna tačka. Eksponent je podešen dodavanjem **konstante podešavanja**, da se ne bi moralo voditi računa o njegovom predznaku i da bi se olakšalo poređenje mašinskih normalizovanih formi. Prethodni broj u mašinskoj normalizovanoj formi sa 10 mesta (MNF10) izgleda:

0031270000_{MNF10}

Posmatrano s leva u desno, prva cifra prethodnog broja sadrži predznak (0 : +, 1 : -). Sledeće dve cifre sadrže podešeni eksponent. Dva mesta omogućuju da eksponent bude u rasponu od -9 do +9. On se podešava dodavanjem konstante podešavanja $10==10^{2-1}$, pa je raspon podešenog eksponenta od 1 do 19. Ovako podešeni eksponent omogućuje poređenje mašinskih normalizovanih formi bez njihovog raspakivanja, pri čemu se mora uračunati uticaj predznaka na rezultat poređenja. Preostalih sedam cifara sadrži frakciju. Podrazumeva se da iza prve cifre frakcije dolazi decimalna tačka.

Predstava realnih brojeva u ograničenom broju cifara ograničava i dijapazon realnih brojeva koji se mogu predstaviti, ali i preciznost, jer iz datog dijapazona može da se precizno predstavi samo ograničen broj realnih brojeva. Greška predstave realnih brojeva, koji ne mogu precizno da se predstave, raste sa veličinom eksponenta i kreće se od 10^{-15} do 10^{3} za MNF10. Tolika je razlika apsolutnih vrednosti 2 najmanja susedna realna broja: 0011000000_{MNF10} (1.0×10^{-9}) i

0011000001_{MNF10} (1.000001×10^{-9}), odnosno 2 najveća susedna realna broja: 0199999998_{MNF10} (9.999998×10^{9}) i 0199999999_{MNF10} (9.999999×10^{9}).

ARITMETIKA MAŠINSKE NORMALIZOVANE FORME

U aritmetici normalizovane forme, predznak, podešeni eksponenti i frakcije se posmatraju odvojeno. Prvo se odredi predznak rezultata. Sabiranje i oduzimanje započinje izjednačavanjem eksponenata (radi ispravnog potpisivanja frakcija). Uvek se manji eksponent izjednačava sa većim, uz pomeranje decimalne tačke u frakciji i, eventualno, odbacivanje prekobrojnih cifara. Zatim sledi sabiranje ili oduzimanje frakcija i, po potrebi, normalizacija rezultata. Tako, sabiranju brojeva:

$$\begin{aligned} &0031270000_{MNF10} && (1.27\times10^{-7})\\ +&0091000001_{MNF10} && (1.000001\times10^{-1}) \end{aligned}$$

obavezno prethodi izjednačavanje eksponenta prvog broja sa eksponentom drugog broja, pri čemu se odbacuju prekobrojne cifre frakcije 2 i 7. Tako nastaju brojevi:

$$\begin{aligned} &0090000001_{MNF10} && (0.000001\times10^{-1}) && \text{(odbačene su prekobrojne cifre 2 i 7)}\\ +&\underline{0091000001}_{MNF10} && (1.000001\times10^{-1})\\ &0091000002_{MNF10} && (1.000002\times10^{-1}) \end{aligned}$$

Kod množenja, eksponenti se sabiraju, uz umanjivanje njihovog zbira za konstantu podešavanja 10, a frakcije se množe. Kod deljenja, eksponenti se oduzimaju, uz dodavanje njihovoj razlici konstante podešavanja 10, a frakcije se dele. U oba slučaja, na kraju, eventualno, sledi normalizacija rezultata. Prethodno ilustruje primer množenja:

$$\begin{aligned} &0112000000_{MNF10} && (2.0\times10^{1})\\ \times&\underline{0124000000}_{MNF10} && (4.0\times10^{2})\\ &0138000000_{MNF10} && (8.0\times10^{3}) \end{aligned}$$

U aritmetici normalizovane forme do izlaska van opsega (s leva, ali i s desna) dolazi samo u eksponentu, jer se prekobrojne cifre frakcije odbacuju. Odbacivanje prekobrojnih cifara frakcije može da uzrokuje ozbiljne greške, naročito, ako u operaciji učestvuju realni brojevi koji se veoma razlikuju po apsolutnoj vrednosti. Primer za takvu vrstu grešaka pruža sabiranje vrlo mnogo malih realnih brojeva. Kada, u toku sabiranja, apsolutna vrednost zbira postane toliko velika, da su sve cifre frakcija preostalih sabiraka prekobrojne, tada u nastavku sabiranja ne dolazi do uvećanja ovog zbira, iako on može biti manji od sume preostalih (zanemarenih) sabiraka.

2.3. BINARNI BROJNI SISTEM

U binarnom brojnom sistemu koriste se samo cifre 0 i 1, njegova **baza** (*radix*) je 2_{10}, a brojevi se predstavljaju u pozicionoj predstavi. Pri prelasku iz decimalnog u binarni brojni sistem i obratno, neophodna je **konverzija brojeva**.

POSTUPAK KONVERZIJE BROJEVA

Konverzije brojeva iz brojnog sistema s jednom bazom u brojni sistem s drugom bazom se zasnivaju na izjednačavanju pozicionih predstava pomenutih brojeva, radi izdvajanja nepoznatih cifara. Pri tome u ovim predstavama se koriste cifre brojnog sistema u kome se računa. Postupak konverzije zahteva da se posebno obavlja konverzija celog, a posebno konverzija razlomljenog dela broja.

Do izdvajanja nepoznatih binarnih cifara, pri konverziji celog broja iz decimalnog u binarni brojni sistem, dolazi kada se konvertovani broj deli bazom binarnog brojnog sistema. U primeru, koji sledi, koriste se cifre decimalnog brojnog sistema, jer se u njemu računa:

$...+d_1 10^1+d_0 10^0$	==	$...+b_1 2^1+b_0 2^0$	
6_{10}	==	$...+b_1 2^1+b_0 2^0$ |:2	
0	==	b_0	(ostatak deljenja)
3_{10}	==	$...+b_2 2^1+b_1 2^0$ |:2	(količnik)
1	==	b_1	(ostatak deljenja)
1	==	$...+b_3 2^1+b_2 2^0$ |:2	(količnik)
1	==	b_2	(ostatak deljenja)
0			(količnik)
6_{10}	==	110_2	

Do izdvajanja nepoznatih binarnih cifara, pri konverziji razlomljenog broja iz decimalnog u binarni brojni sistem, dolazi kada se konvertovani broj množi bazom binarnog brojnog sistema. U sledećem primeru koriste se cifre decimalnog brojnog sistema, jer se u njemu računa:

$d_{-1}10^{-1}+d_{-2}10^{-2}+...$	==	$b_{-1}2^{-1}+b_{-2}2^{-2}+...$	
0.375_{10}	==	$b_{-1}2^{-1}+b_{-2}2^{-2}+...$ |×2	
0	==	b_{-1}	(celi deo proizvoda)
0.75_{10}	==	$b_{-2}2^{-1}+b_{-3}2^{-2}+...$ |×2	(razlomljeni deo proizvoda)
1	==	b_{-2}	(celi deo proizvoda)
0.5_{10}	==	$b_{-3}2^{-1}+b_{-4}2^{-2}+...$ |×2	(razlomljeni deo proizvoda)
1	==	b_{-3}	(celi deo proizvoda)
0			(razlomljeni deo proizvoda)
0.375_{10}	==	0.011_2	

Algoritam konverzije razlomljenog broja iz decimalnog u binarni brojni sistem ne mora da ima kraj. To pokazuje primer konvertovanja razlomljenog broja 0.2_{10}, prikazan tabelarno (Slika 2.3.1).

razlomljeni deo proizvoda	proizvod (×2)	celi deo proizvoda
0.2	0.4	$0 == b_{-1}$
0.4	0.8	$0 == b_{-2}$
0.8	1.6	$1 == b_{-3}$
0.6	1.2	$1 == b_{-4}$
0.2	0.4	$0 == b_{-5}$

Slika 2.3.1 Konvertovanje razlomljenog broja 0.2_{10}

U tabelarno datom primeru (Slika 2.3.1) javlja se beskonačna periodičnost, pa je rezultat konverzije približan:

$0.2_{10} \approx 0.0011_2$

Konverzija celog broja iz binarnog u decimalni brojni sistem se svodi na množenje cifara konvertovanog broja bazom binarnog brojnog sistema. U sledećem primeru koriste se cifre decimalnog brojnog sistema, jer se u njemu računa:

$...+b_2 2^2+b_1 2^1+b_0 2^0 \quad == \quad ((...+b_2)\times 2+b_1)\times 2+b_0$

$X_{10} == 110_2$
$X_{10} == ((b_2)\times 2+b_1)\times 2+b_0$

$X_{10} = b_2 == 1$
$X_{10} = X_{10}\times 2+b_1 == 1\times 2+1 == 3$
$X_{10} = X_{10}\times 2+b_0 == 3\times 2+0 == 6$
$X_{10} == 6$
$6_{10} == 110_2$

Konverzija razlomljenog broja iz binarnog u decimalni brojni sistem se svodi na deljenje cifara konvertovanog broja bazom binarnog brojnog sistema. U sledećem primeru koriste se cifre decimalnog brojnog sistema, jer se u njemu računa:

$b_{-1}2^{-1}+b_{-2}2^{-2}+b_{-3}2^{-3}+... == (b_{-1}+(b_{-2}+(b_{-3}+...)/2)/2)/2$

$X_{10} == 0.0011_2$
$X_{10} == (b_{-1}+(b_{-2}+(b_{-3}+(b_{-4})/2)/2)/2)/2$

X_{10}	=	b_{-4}	==	1		
X_{10}	=	$X_{10}/2+b_{-3}$	==	$1/2+1$	==	1.5
X_{10}	=	$X_{10}/2+b_{-2}$	==	$1.5/2+0$	==	0.75
X_{10}	=	$X_{10}/2+b_{-1}$	==	$0.75/2+0$	==	0.375
X_{10}	=	$X_{10}/2$	==	$0.375/2$	==	0.1875
0.1875_{10}	==	0.0011_2				

Pošto konverzija 0.2_{10} daje približno 0.0011_2, a konverzija 0.0011_2 daje 0.1875_{10}, sledi da je:

$$0.2_{10} \approx 0.1875_{10}$$

odnosno da je greška aproksimacije 0.0125_{10}.

Za skraćeno predstavljanje brojeva iz binarnog brojnog sistema koristi se heksadecimalni brojni sistem, jer jedna heksadecimalna cifra zamenjuje četiri binarne cifre. U heksadecimalnom brojnom sistemu koriste se cifre 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (10_{10}), B (11_{10}), C (12_{10}), D (13_{10}), E (14_{10}) i F (15_{10}). Njegova baza je 16_{10}, a brojevi se predstavljaju u pozicionoj predstavi.

Konverzija broja iz binarnog u heksadecimalni brojni sistem se sastoji od zamena po četiri binarne cifre jednom heksadecimalnom cifrom. Pre toga se binarni broj dopuni s leva vodećim nulama do broja cifara koji je deljiv sa 4:

$$1011111_2 == 5F_{16}$$

$01011111_2 ==$
$0\times2^7+1\times2^6+0\times2^5+1\times2^4+1\times2^3+1\times2^2+1\times2^1+1\times2^0 ==$
$(0\times2^3+1\times2^2+0\times2^1+1\times2^0)\times2^4+(1\times2^3+1\times2^2+1\times2^1+1\times2^0)\times2^0 ==$
$0101\times2^4+1111\times2^0 ==$
$5\times16^1+F\times16^0$

(eksponenti i baza 16 su prikazani u decimalnom brojnom sistemu).

Konverzija broja iz heksadecimalnog u binarni brojni sistem se sastoji od zamene svake heksadecimalne cifre sa 4 odgovarajuće binarne cifre.

2.4. ARITMETIKA OGRANIČENOG BROJA CIFARA U BINARNOM BROJNOM SISTEMU

I u binarnom brojnom sistemu sabiranje i oduzimanje neoznačenih celih brojeva se sastoji od direktne primene tablice sabiranja na parove korespondentnih cifara. U narednim primerima se smatra da je broj binarnih cifra ograničen na 8:

00000101	(5)	00000101	(5)
+00000011	(3)	-00000011	(3)
00001000	(8)	00000010	(2)

KOMPLEMENT 2 PREDSTAVA OZNAČENIH CELIH BROJEVA

Za označene cele brojeve koristi se komplement 2 predstava, radi izbegavanja komplikacija kod njihovog sabiranja i oduzimanja. Komplement 2 predstava pozitivnih celih brojeva nastaje njihovim dopunjavanjem s leva vodećim nulama (do ukupnog broja cifara):

00000010	komplement 2 predstava broja +2

Komplement 2 predstava negativnih celih brojeva nastaje dodavanjem jedinice komplement 1 predstavi negativnih celih brojeva. Komplement 1 predstava negativnih celih brojeva nastaje tako, što se u potpunoj apsolutnoj vrednosti negativnih celih brojeva svaka nula zameni jedinicom i obrnuto:

00000010	potpuna apsolutna vrednost broja -2
11111101	komplement 1 predstava broja -2
11111110	komplement 2 predstava broja -2

U nastavku slede neke vrednosti osmocifrene komplement 2 predstave označenih celih brojeva (njima korespondentne decimalne vrednosti, sa predznakom, su navedene u zagradama):

01111111	(+127)
01111110	(+126)
...	
00000001	(+1)
00000000	(0)
11111111	(-1)
...	
10000001	(-127)
10000000	(-128)

I u komplement 2 predstavi, broj negativnih brojeva je za jedan veći od broja pozitivnih brojeva.

IZLAZAK VAN OPSEGA

Kod neoznačenih celih brojeva, pojava prenosa sa najznačajnijeg mesta ukazuje na izlazak van opsega. Kod označenih celih brojeva u komplement 2 predstavi, pojava neodgovarajuće cifre na najznačajnijem mestu rezultata ukazuje na izlazak van opsega (na primer, kod sabiranja dva pozitivna cela broja, pojava cifre 1 na najznačajnijem mestu zbira nedvosmisleno ukazuje na izlazak van opsega). Svi slučajevi izlaska van opsega, kod sabiranja označenih celih brojeva u komplement 2 predstavi, mogu se pregledno prikazati tabelarno (Slika 2.4.1).

najznačajniji bit prvog sabirka (S_1)	najznačajniji bit drugog sabirka (S_2)	najznačajniji bit zbira (Z)	izlazak van opsega
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Slika 2.4.1 Izlasci van opsega kod sabiranja označenih celih brojeva u komplement 2 predstavi

U tabeli (Slika 2.4.1), cifre iz prve tri kolone se interpretiraju kao celi brojevi, a cifre iz četvrte kolone se interpretiraju kao logičke vrednosti. Ako se cifre u sve četiri kolone interpretiraju kao logičke vrednosti, tada prve tri kolone sadrže argumente, a poslednja vrednosti logičke funkcije izlaska van opsega kod sabiranja:

$((\sim S_1)\&(\sim S_2)\&Z)|(S_1\&S_2\&(\sim Z))$

Svi slučajevi izlaska van opsega, kod oduzimanja označenih celih brojeva u komplement 2 predstavi, mogu se pregledno prikazati tabelarno (Slika 2.4.2).

najznačajniji bit umanjenika (U_1)	najznačajniji bit umanjioca (U_2)	najznačajniji bit razlike (R)	izlazak van opsega
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Slika 2.4.2 Izlasci van opsega kod oduzimanja označenih celih brojeva u komplement 2 predstavi

U tabeli (Slika 2.4.2), cifre iz prve tri kolone se interpretiraju kao celi brojevi, a cifre iz četvrte kolone se interpretiraju kao logičke vrednosti. Ako se cifre u sve četiri kolone interpretiraju kao logičke vrednosti, tada prve tri kolone sadrže argumente, a poslednja vrednosti logičke funkcije izlaska van opsega kod oduzimanja:

$((\sim U_1)\&U_2)\&R)|(U_1\&(\sim U_2)\&(\sim R))$

Interpretacija rezultata aritmetičkih operacija zavisi od interpretacije brojeva. Na primer, kod sabiranja neoznačenih celih brojeva:

$$\begin{array}{rl} 11111111_2 & (255_{10}) \\ \underline{+11111111_2} & (255_{10}) \\ 11111110_2 & (254_{10}) \end{array}$$

dolazi do izlaska van opsega, jer se javlja prenos sa najznačajnijeg mesta binarnog rezultata. Kod sabiranja neoznačenih celih brojeva:

$$\begin{array}{rl} 01000000_2 & (64_{10}) \\ \underline{+01000000_2} & (64_{10}) \\ 10000000_2 & (128_{10}) \end{array}$$

ne dolazi do izlaska van opsega, jer se ne javlja prenos sa najznačajnijeg mesta binarnog rezultata. Ali, izmena interpretacije brojeva u prethodna dva primera povlači za sobom i izmenu interpretacije rezultata. Tako, kod sabiranja označenih celih brojeva u komplement 2 predstavi:

$$\begin{array}{rl} 11111111_2 & (-1_{10}) \\ \underline{+11111111_2} & (-1_{10}) \\ 11111110_2 & (-2_{10}) \end{array}$$

ne dolazi do izlaska van opsega, jer je predznak binarnog rezultata ispravan. Prenos sa najznačajnijeg mesta binarnog rezultata se zanemaruje, jer je reč o komplement 2 predstavi brojeva. Slično, kod sabiranja označenih celih brojeva u komplement 2 predstavi:

$$\begin{array}{rl} 01000000_2 & (+64_{10}) \\ \underline{+01000000_2} & (+64_{10}) \\ 10000000_2 & (-128_{10}) \end{array}$$

dolazi do izlaska van opsega, jer se javlja neodgovarajuća cifra na najznačajnijem mestu rezultata.

POREĐENJE CELIH BROJEVA

I u binarnom brojnom sistemu, određivanje relacija se svodi na oduzimanje prvog operanda (PO) relacije od drugog (DO) i na zaključivanje o važenju relacije na osnovu toga:

1. da li je razlika jednaka nuli ili je različita od nule,
2. da li je razlika negativna ili pozitivna,
3. da li se, pri oduzimanju neoznačenih celih brojeva, javlja izlazak van opsega u obliku prenosa ili ne, odnosno,
4. da li, pri oduzimanju označenih celih brojeva, razlika izlazi van opsega ili ne.

Za opisivanje karakteristika razlike zgodno je uvesti posebne **logičke promenljive**: **N** (nula), **M** (minus), **P** (prenos) i **V** (van opsega). Tako logička promenljiva N sadrži 1, ako je razlika jednaka nuli. U suprotnom, ona sadrži 0. Ova logička promenljiva je jednaka negaciji logičkog ili svih cifara razlike. Logička promenljiva M sadrži 1, ako je razlika negativna. U suprotnom, ona sadrži 0. Ova logička promenljiva je jednaka najznačajnijoj cifri razlike. Logička promenljiva P sadrži 1, ako se pri oduzimanju neoznačenih celih brojeva javi prenos. U suprotnom, ona sadrži 0. Ova logička promenljiva je jednaka prenosu sa najznačajnijeg mesta pri oduzimanju. Logička promenljiva V sadrži 1, ako, pri oduzimanju označenih celih brojeva, razlika izađe van opsega. U suprotnom, ona sadrži 0. Ova logička promenljiva je jednaka vrednosti logičke funkcije izlaska van opsega kod oduzimanja.

Pri određivanju relacija, svakoj od prethodne četiri logičke promenljive dodeljuje se, po oduzimanju, odgovarajuća vrednost. To se radi zbog višeznačne interpretacije operanada relacija. Međutim, zaključak o važenju relacije se oslanja samo na logičke promenljive koje su značajne za važeću interpretaciju poređenih

vrednosti. To se može pokazati tabelama (Slika 2.4.3) i (Slika 2.4.4). Tabela (Slika 2.4.3) sadrži pregled logičkih izraza od čije vrednosti zavisi važenje pojedinih relacija za neoznačene brojeve.

relacija važi	ako je tačan logički izraz
DO == PO	N
DO != PO	~N
DO < PO	P
DO >= PO	~P
DO > PO	(~P)&(~N)
DO <= PO	P|N

Slika 2.4.3 Prikaz uslova važenja relacija za neoznačene brojeve

Tabela (Slika 2.4.4) sadrži pregled logičkih izraza od čije vrednosti zavisi važenje pojedinih relacija za označene brojeve.

relacija važi	ako je tačan logički izraz
DO == PO	N
DO != PO	~N
DO < PO	M^V
DO >= PO	~(M^V)
DO > PO	~(M^V)&(~N)
DO <= PO	(M^V)|N

Slika 2.4.4 Prikaz uslova važenja relacija za označene brojeve

Razumevanje logičkih izraza, navedenih u ovoj tabeli (Slika 2.4.4) za relacije <, >=, >, i <=, zahteva posmatranje slučajeva u kojima su oba operanda pozitivna, odnosno negativna, odnosno u kojima su operandi različitih predznaka. Tabelarni pregled karakterističnih slučajeva važenja relacije < sadrži Slika 2.4.5.

drugi operand	prvi operand	<	M	V
+	+	da	1	0
-	+	da	1 0	0 1
-	-	da	1	0
+	+	ne	0	0
+	-	ne	0 1	0 1
-	-	ne	0	0

Slika 2.4.5 Pregled karakterističnih slučajeva važenja relacije <

Relacija < važi, ako su oba njena operanda pozitivna, a drugi je manji od prvog. Tada je razlika uvek negativna i nije van opsega (M == 1 i V == 0). Relacija < važi i kada je njen drugi operand negativan, a prvi pozitivan. Tada je razlika uvek ili negativna i nije van opsega (M == 1 i V == 0), ili pozitivna i van opsega (M == 0 i V == 1). Relacija < važi i ako su njena oba operanda negativna, a drugi je manji od prvog. Tada je razlika uvek negativna i nije van opsega (M == 1 i V == 0). Relacija < ne važi, kada su oba njena operanda pozitivna, a drugi nije manji od prvog. Tada je razlika uvek pozitivna i nije van opsega (M == 0 i V == 0). Relacija < ne važi ni ako je njen drugi operand pozitivan, a prvi negativan. Tada je razlika uvek ili pozitivna i nije van opsega (M == 0 i V == 0), ili negativna i van opsega (M == 1 i V == 1). Relacija < ne važi ni kada su njena oba operanda negativna, a drugi nije manji od prvog. Tada je razlika uvek pozitivna i nije van opsega (M == 0 i V == 0).

VIŠESTRUKA PRECIZNOST

I u binarnom brojnom sistemu, aritmetika ograničenog broja cifara zahteva višestruku preciznost, kada se operacije odnose na vrednosti čiji broj cifara je veći od broja cifara lokacije. I u ovom slučaju, delovi vrednosti se čuvaju u raznim lokacijama, a operacije se obavljaju za deo po deo ovih vrednosti (idući od manje značajnih ka značajnijim delovima vrednosti).

ZAKONI ARITMETIKE

Takođe, ni u binarnom brojnom sistemu za aritmetiku ograničenog broja cifara ne važe zakoni aritmetike, jer je, zbog mogućnosti izlaska van opsega, rezultat zavisan od redosleda obavljanja operacija. To potkrepljuje sledeći primer za osmobitnu preciznost:

$$(10000100_2+10100000_2)-10000000_2 \; != \; 10000100_2+(10100000_2-10000000_2)$$

u kome su navedeni neoznačeni celi brojevi (u toku formiranja levog zbira dolazi do izlaska van opsega, pa je on u opštem slučaju različit od desnog zbira, u kome se ne javlja izlazak van opsega).

2.5. PREDSTAVLJANJE VREDNOSTI REALNOG TIPA U BINARNOM BROJNOM SISTEMU

Mašinska normalizovana forma se koristi i u binarnom brojnom sistemu za predstavljanje vrednosti realnog tipa. U nastavku se koristi 8 cifarskih pozicija za mašinsku normalizovanu formu (MNF8). U njoj, gledajući s leva u desno, prva pozicija sadrži predznak (0 : +, 1 : -). Naredne 4 pozicije sadrže podešeni eksponent. Kao konstanta podešavanja koristi se vrednost $8==1000_2==2^{4-1}$. Ona se sabira sa eksponentima koji mogu biti u rasponu od -8 do 7, pa se podešeni eksponenti nalaze u dijapazonu od $0000_2==0$ do $1111_2==15$. Poslednje 3 pozicije

sadrže tri manje značajne cifre frakcije. Preostala, najznačajnija cifra frakcije je uvek 1, jer, kod normalizovane forme, tačka dolazi iza prve značajne cifre frakcije, različite od 0. Zato se ova cifra ne prikazuje u frakciji. Znači, frakcija je uvek različita od nule, jer se podrazumeva da je njena najznačajnija cifra 1. Zato se za predstavljanje nule rezerviše nulta vrednost podešenog eksponenta. Slede primeri brojeva, predstavljenih u MNF8:

10101000_{MNF8}==$-1.000_2\times2^{-3}$==-0.125_{10}
00000000_{MNF8}==0.0
01000000_{MNF8}==$+1.000_2\times2^{0}$==$+1.0_{10}$
01001011_{MNF8}==$+1.011_2\times2^{1}$==$+2.75_{10}$
01110001_{MNF8}==$+1.001_2\times2^{6}$==$+72.0_{10}$

I u binarnom brojnom sistemu predstava realnih brojeva u ograničenom broju cifara ograničava i dijapazon realnih brojeva, koji se mogu predstaviti, i preciznost, jer iz datog dijapazona može da se precizno predstavi samo ograničen broj realnih brojeva. Pri tome, greška predstave realnih brojeva, koji ne mogu precizno da se predstave, raste sa veličinom eksponenta i kreće se od 2^{-10} do 2^4 za MNF8.

Prethodno uvedena pravila za aritmetiku normalizovane forme važe i u binarnom brojnom sistemu. Znači, do izlaska van opsega (s leva, ali i s desna) dolazi samo u eksponentu, jer se prekobrojne cifre frakcije odbacuju. Odbacivanje prekobrojnih cifara frakcije može da uzrokuje ozbiljne greške.

VIŠEZNAČNA INTERPRETACIJA CELIH BROJEVA

Kodiranje vrednosti raznih tipova celim brojevima stvara mogućnost višeznačne interpretacije celih brojeva. Na primer, celi binarni broj 01001011 se može interpretirati kao celi decimalni broj 75, ali i kao realan decimalni broj 2.75. Zbog različite interpretacije, neophodna je konverzija, kada se vrednost jednog tipa dodeljuje promenljivoj drugog tipa. Na primer, ako se, bez konverzije, realnoj promenljivoj dodeli binarni broj 01001011, koji odgovara celom decimalnom broju 75, tada će ova promenljiva dobiti vrednost koja odgovara realnom decimalnom broju 2.75, jer binarni broj 01001011 predstavlja i MNF8 za dotični realni broj ($+1.011_2\times2^1$). Zato je potrebno, pre dodele, izvršiti konverziju binarne vrednosti 01001011:

75_{10}==1001011_2==$1.001011_2\times2^6\approx1.001_2\times2^6$==$72.0_{10}$==$01110001_{MNF8}$

Znači, nakon konverzije, realna promenljiva će dobiti vrednost koja odgovara realnom decimalnom broju 72.0, što je približna vrednost realnog decimalnog broja 75.0. Navedeni primer ukazuje da nije uvek moguće precizno konvertovati vrednost celog tipa u MNF, čak i kada se koristi isti broj bita za njihovo reprezentovanje.

Konverzija je neophodna i u obrnutom smeru. Na primer prilikom dodele celobrojnoj promenljivoj vrednosti 01110001_{MNF8}, koji odgovara realnom decimalnom broju 72.0, ova promenljiva će dobiti vrednost koja odgovara celom decimalnom broju 113. Zato je potrebno, pre dodele, izvršiti konverziju vrednosti 01110001_{MNF8}:

$$72.0_{10}==01110001_{MNF8}==1.001_2\times2^6==1001000.0_2==1001000_2==72_{10}$$

Važno je naglasiti da nije uvek moguća konverzija iz MNF u celi tip, čak i kada se koristi isti broj bita za njihovo reprezentovanje, jer MNF pokriva veći dijapazon vrednosti od celog tipa.

2.6. PITANJA

1. Šta omogućuje komplement predstava označenih brojeva?
2. Kako komplement predstava pojednostavljuje procesor?
3. Kada se (ne) zanemaruje prenos sa najznačajnije pozicije prilikom aritmetike celih brojeva?
4. Šta ukazuje na izlazak van opsega?
5. Koje delove sadrži mašinska normalizovana forma?
6. Zašto je uvedena konstanta podešavanja (kako se ona koristi)?
7. Kako se obavlja aritmetika brojeva predstavljenih mašinskom normalizovanom formom?
8. Gde dolazi do izlaska van opsega u aritmetici mašinske normalizovane forme (u eksponentu ili frakciji)?
9. Gde se odbacuju prekobrojne cifre u aritmetici mašinske normalizovane forme (u eksponentu ili frakciji)?
10. Kako se obavljaju konverzije brojeva iz decimalnog u binarni brojni sistem (i obrnuto)?
11. Koji binarni broj se dobije nakon konverzije decimalnog broja 4.25?
12. Koji decimalni broj se dobije nakon konverzije binarnog broja 100.01?
13. Koji heksadecimalni broj se dobije nakon konverzije binarnog broja 11010?
14. Koji binarni broj se dobije nakon konverzije heksadecimalnog broja 1A?
15. Koja je komplement 2 predstava binarnog broja -100?
16. Da li se prilikom sabiranja binarnih brojeva 01000100 i 01110000 javlja izlazak van opsega (za obe interpretacije ovih brojeva)?
17. Na osnovu čega se određuje važenje relacija između (ne)označenih brojeva?
18. Koja je mašinska normalizovana forma (MNF8) binarnog broja -101?
19. Koji binarni broj odgovara mašinskoj normalizovanoj formi 11010010_{MNF8}?

3. ASEMBLERSKO PROGRAMIRANJE

3.1. NIVOI PROGRAMIRANJA

Asemblersko programiranje se zasniva na korišćenju asemblerskih naredbi. Ono se nalazi na višem nivou od mašinskog programiranja, koje koristi mašinske naredbe. U ovom slučaju viši nivo podrazumeva apstrakciju (zanemarivanje). Tako, u primeru asemblerskog programiranja, apstrakcija znači zamenarivanje mašinskih formata naredbi (čije poznavanje je neophodno za mašinsko programiranje, a nebitno za asemblersko programiranje). Na još višem nivou se nalazi programiranje procedurnim programskim jezicima, koje potpuno zanemaruje arhitekturu naredbi, jer se zasniva na vrlo uopštenom modelu računara. Zato se procedurni programski jezici svrstavaju u grupu **programskih jezika visokog nivoa**, dok se mašinski i asemblerski programski jezici svrstavaju u grupu **programskih jezika niskog nivoa**.

Programiranje programskim jezicima niskog nivoa je teže od programiranja programskim jezicima visokog nivoa, jer zahteva detaljno poznavanje arhitekture naredbi. Razmišljanje na nivou arhitekture naredbi odvlači pažnju od suštine rešavanog problema i tako uzrokuje razne greške. Iz istih razloga su programi, pisani programskim jezicima niskog nivoa, manje pregledni i teži za održavanje nego programi pisani programskim jezicima visokog nivoa. Ali, programiranje na nivou arhitekture naredbi ima i svoje prednosti, jer dozvoljava korišćenje svih mogućnosti računara. Zato je programiranje programskim jezicima niskog nivoa neizbežno uvek kada je potrebno na poseban način iskoristiti mogućnosti računara.

Mašinske jezike je opravdano koristiti samo ako ne postoje prevodioci za asemblerske ili procedurne programske jezike (asembleri i kompajleri). Poznavanje mašinskih jezika je neophodno i za pravljenje pomenutih prevodilaca.

Asemblerske jezike je opravdano koristiti, ne samo kada ne postoje prevodioci za procedurne programske jezike (kompajleri) ili kada se ovakvi prevodioci prave, nego i u slučajevima koji zahtevaju da se mogućnosti računara iskoriste do krajnjih granica (za šta je neophodno vladanje arhitekturom naredbi koju sakrivaju programski jezici visokog nivoa). Radi olakšavanja asemblerskog programiranja, razvoj asemblerskih programa je uputno započeti preciziranjem algoritma. Za to je zgodno koristiti, kad god je moguće, sredstva tipična za programske jezike visokog nivoa. Nakon preciziranja algoritma, pažnja se može potpuno koncentrisati na izražavanje algoritma asemblerskim naredbama. Ovakav pristup asemblerskom programiranju smanjuje mogućnost greške, jer u svakom momentu ograničava broj detalja o kojima se mora voditi računa.

Arhitektura naredbi, koju odražava svaki asemblerski jezik, se uvek odnosi na određen procesor, pa se zato svaki asemblerski jezik vezuje za neki procesor. U ovoj knjizi je to hipotetski procesor, nazvan KONCEPT, čije ime ukazuje da je

njegova prevashodna namena da omogući shvatanje principa na kojima se zasniva rad računara.

3.2. ARHITEKTURA NAREDBI PROCESORA KONCEPT

Memorija procesora KONCEPT se sastoji od lokacija koje sadrže po 16 bita. Krajnje desni bit lokacije (označen indeksom 0, na primer B_0) odgovara najmanje značajnoj binarnoj cifri, a krajnje levi bit lokacije (označen indeksom 15, na primer B_{15}) odgovara najznačajnijoj binarnoj cifri. Pristupanje memorijskoj lokaciji podrazumeva pristupanje svim njenim bitima.

16 bita omogućuje prikaz 2^{16} različitih binarnih brojeva. Ako svaki od njih predstavlja adresu samo jedne memorijske lokacije, tada ukupno ima 2^{16} adresa i 2^{16} njima odgovarajućih memorijskih lokacija procesora KONCEPT.

Adrese svih mogućih memorijskih lokacija obrazuju **adresni prostor** procesora KONCEPT. Znači adresni prostor procesora KONCEPT obuhvata 2^{16} različitih adresa i njima pripadnih lokacija (u opštem slučaju adresni prostor procesora obuhvata 2^n različitih adresa/lokacija, ako se adrese sastoje od po n bita).

Procesor KONCEPT ima ukupno 16 **registara opšte namene**, označenih kao `%0`, `%1`, `%2`, `%3`, `%4`, `%5`, `%6`, `%7`, `%8`, `%9`, `%10`, `%11`, `%12`, `%13`, `%14` i `%15`. Svaki od registara opšte namene ovog procesora je organizovan kao memorijska lokacija. Procesor KONCEPT ima i **status registar**, koji je takođe organizovan kao memorijska lokacija. Njegov (najmanje značajan) bit SR_0 sadrži logičku promenljivu N (nula), bit SR_1 logičku promenljivu M (minus), bit SR_2 logičku promenljivu P (prenos), a bit SR_3 logičku promenljivu V (van opsega).

Procesor KONCEPT podržava neposredno, direktno, registarsko, posredno i indeksno adresiranje.

NAREDBE ZA CELOBROJNU ARITMETIKU

U repertoaru naredbi procesora KONCEPT nalaze se naredbe za celobrojnu aritmetiku: SABERI, SABERI_P, DODAJ_1, ODUZMI, ODUZMI_P, ODBIJ_1 i UPOREDI.

SABERI je aritmetička naredba, koja omogućuje sabiranje cele vrednosti, određene prvim (ulaznim) operandom (PO), sa celom vrednošću, određenom drugim (ulaznim) operandom (DO). Zbir se odlaže u lokaciju, određenu drugim (sada izlaznim) operandom. Tabela (Slika 3.2.1) opisuje sabiranje korespondentnih bita prvog i drugog operanda, označenih kao PO_i i DO_i, pri čemu se zbir i prenos označavaju kao Z_i i P_{i+1}, a prenos iz sabiranja prethodnog para bita kao P_i (i = 0, ... , 15):

PO_i	DO_i	P_i	Z_i	P_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Slika 3.2.1 Tabela sabiranja parova bita

Za P_0 se smatra da je uvek 0. Ako se cifre iz svih pet kolona prethodne tabele interpretiraju kao logičke vrednosti, tada iz prethodne tabele sledi važenje sledećih logičkih funkcija:

$Z_i = PO_i \hat{} DO_i \hat{} P_i$
$P_{i+1} = ((PO_i | DO_i) \& P_i) | (PO_i \& DO_i)$

Nakon sabiranja, menjaju se vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$N = \sim(Z_{15}|Z_{14}|\ Z_{13}|Z_{12}|\ Z_{11}|Z_{10}|\ Z_9|Z_8|\ Z_7|Z_6|\ Z_5|Z_4|\ Z_3|Z_2|\ Z_1|Z_0)$
$M = Z_{15}$
$P = P_{16}$
$V = ((\sim PO_{15}) \& (\sim DO_{15}) \& Z_{15}) | (PO_{15} \& DO_{15} \& (\sim Z_{15}))$

(vrednost logičke promenljive V određuje logička funkcija izlaska van opsega kod sabiranja).

SABERI_P je aritmetička naredba, koja omogućuje sabiranje cele vrednosti, određene prvim (ulaznim) operandom (PO) i zatečenog sadržaja logičke promenljive P (prenosa) sa celom vrednošću, određenom drugim (ulaznim) operandom (DO). Zbir se odlaže u lokaciju, određenu drugim (sada izlaznim) operandom. Ova naredba omogućuje sabiranje u višestrukoj preciznosti, jer u zbir uključuje i zatečeni prenos, preostao iza prethodnog sabiranja. Naredba SABERI_P se razlikuje od naredbe SABERI samo po tome što kao vrednost zatečenog prenosa P_0 uzima vrednost logičke promenljive P.

DODAJ_1 je aritmetička naredba, koja se razlikuje od naredbe SABERI samo po tome što se kao vrednost jednog (ulaznog) operanda koristi konstanta 1, pa se ovaj operand ne navodi.

ODUZMI je aritmetička naredba, koja omogućuje oduzimanje cele vrednosti - umanjioca, određene prvim (ulaznim) operandom (PO), od cele vrednosti -

umanjenika, određene drugim (ulaznim) operandom (DO). Razlika se odlaže u lokaciju, određenu drugim (sada izlaznim) operandom. Tabela (Slika 3.2.2) opisuje oduzimanje korespondentnih bita prvog i drugog operanda, označenih kao PO_i i DO_i, pri čemu se razlika i prenos označavaju kao R_i i P_{i+1}, a prenos iz oduzimanja prethodnog para bita kao P_i ($i = 0, \ldots, 15$):

DO_i	PO_i	P_i	R_i	P_{i+1}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Slika 3.2.2 Tabela oduzimanja parova bita

Za P_0 se smatra da je uvek 0. Ako se cifre iz svih pet kolona prethodne tabele interpretiraju kao logičke vrednosti, tada iz prethodne tabele sledi važenje sledećih logičkih funkcija:

$R_i = PO_i \hat{} DO_i \hat{} P_i$
$P_{i+1} = ((DO_i | P_i) \& (\sim PO_i)) | (DO_i \& P_i)$

Nakon oduzimanja, menjaju se i vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$N = \sim(R_{15}|R_{14}|\ R_{13}|R_{12}|\ R_{11}|R_{10}|\ R_9|R_8|\ R_7|R_6|\ R_5|R_4|\ R_3|R_2|\ R_1|R_0)$
$M = R_{15}$
$P = P_{16}$
$V = ((\sim DO_{15}) \& PO_{15} \& R_{15}) | (DO_{15} \& (\sim PO_{15}) \& (\sim R_{15}))$

(vrednost logičke promenljive V određuje logička funkcija izlaska van opsega kod oduzimanja).

ODUZMI_P je aritmetička naredba, koja omogućuje oduzimanje cele vrednosti, određene prvim (ulaznim) operandom (PO) i zatečenog sadržaja logičke promenljive P (prenosa) od cele vrednosti, određene drugim (ulaznim) operandom (DO). Razlika se odlaže u lokaciju, određenu drugim (sada izlaznim) operandom. Ova naredba omogućuje oduzimanje u višestrukoj preciznosti, jer u razliku uključuje i zatečeni prenos, preostao iza prethodnog oduzimanja. Naredba

ODUZMI_P se razlikuje od naredbe ODUZMI samo po tome što kao vrednost zatečenog prenosa P_0 uzima vrednost logičke promenljive P.

ODBIJ_1 je aritmetička naredba, koja se razlikuje od naredbe ODUZMI samo po tome što se kao vrednost jednog (ulaznog) operanda koristi konstanta 1, pa se ovaj operand ne navodi.

UPOREDI je aritmetička naredba koja omogućuje oduzimanje cele vrednosti, određene prvim (ulaznim) operandom (PO) od cele vrednosti, određene drugim (ulaznim) operandom (DO) i postavljanje vrednosti logičkih promenljivih N, M, P i V u skladu sa razlikom. Izvršavanje ove naredbe omogućuje utvrđivanje koja relacija važi između vrednosti njenih ulaznih operanada, tako što u skladu sa razlikom postavlja vrednosti logičkih promenljivih N, M, P i V (za određivanje vrednosti ovih logičkih promenljivih koriste se iste logičke funkcije kao kod aritmetičke naredbe ODUZMI).

NAREDBE ZA RUKOVANJE BITIMA

Procesor KONCEPT podržava logičke naredbe: I, ILI i NE, koje omogućuju pristupanje pojedinim bitima lokacija. Pristupanje pojedinim bitima lokacija je osnovna namena i naredbi LEVO i DESNO, pa se one svrstavaju u istu grupu sa logičkim naredbama.

I je logička naredba, koja omogućuje određivanje "logičkog i" korespondentnih bita binarnih vrednosti, određenih prvim (ulaznim) operandom (PO) i drugim (ulaznim) operandom (DO). Rezultat izvršavanja ove naredbe se odlaže u lokaciju, određenu drugim (sada izlaznim) operandom. Obrazovanja bita rezultata (R_i) od bita ulaznih operanda (PO_i i DO_i, za i = 0, ... , 15), opisuje logička funkcija:

$$R_i = PO_i \& DO_i$$

Nakon izvršavanja naredbe I, menjaju se i vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$$N = \sim(R_{15}|R_{14}|\ R_{13}|R_{12}|\ R_{11}|R_{10}|\ R_9|R_8|\ R_7|R_6|\ R_5|R_4|\ R_3|R_2|\ R_1|R_0)$$
$$M = R_{15}$$
$$P = 0$$
$$V = 0$$

ILI je logička naredba, koja omogućuje određivanje "logičkog ili" korespondentnih bita binarnih vrednosti, određenih prvim (ulaznim) operandom (PO) i drugim (ulaznim) operandom (DO). Rezultat izvršavanja ove naredbe se odlaže u lokaciju, određenu drugim (sada izlaznim) operandom. Obrazovanja bita rezultata (R_i) od bita ulaznih operanda (PO_i i DO_i, za i = 0, ... , 15) opisuje logička funkcija:

$R_i = PO_i|DO_i$

Nakon izvršavanja naredbe ILI, menjaju se i vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$N = \sim(R_{15}|R_{14}|\ R_{13}|R_{12}|\ R_{11}|R_{10}|\ R_9|R_8|\ R_7|R_6|\ R_5|R_4|\ R_3|R_2|\ R_1|R_0)$
$M = R_{15}$
$P = 0$
$V = 0$

NE je logička naredba, koja omogućuje negiranje bita binarne vrednosti, određene prvim (ulaznim) operandom (PO). Rezultat izvršavanja ove naredbe se odlaže u lokaciju, određenu prvim (sada izlaznim) operandom. Obrazovanje bita rezultata (R_i) od bita ulaznog operanda (PO_i, za i = 0, ... , 15) opisuje logička funkcija:

$R_i = \sim PO_i$

Nakon izvršavanja naredbe NE, menjaju se i vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$N = \sim(R_{15}|R_{14}|\ R_{13}|R_{12}|\ R_{11}|R_{10}|\ R_9|R_8|\ R_7|R_6|\ R_5|R_4|\ R_3|R_2|\ R_1|R_0)$
$M = R_{15}$
$P = 0$
$V = 0$

LEVO je naredba, koja omogućuje pomeranje svih bita binarne vrednosti, određene prvim (ulaznim) operandom (PO), za jedno mesto u levo. Rezultat izvršavanja ove naredbe se odlaže u lokaciju određenu prvim (sada izlaznim) operandom. Rezultat pomeranja u levo bita binarne vrednosti ulaznog operanda je jednak njenom množenju brojem 2, pa naredba LEVO omogućuje množenje tim brojem. Obrazovanje bita rezultata (R_i) od bita ulaznog operanda (PO_i, za i = 0, ... , 14) opisuju logičke funkcije:

$R_{i+1} = PO_i$
$R_0 = 0$

Nakon izvršavanja naredbe LEVO, menjaju se i vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$N = \sim(R_{15}|R_{14}|\ R_{13}|R_{12}|\ R_{11}|R_{10}|\ R_9|R_8|\ R_7|R_6|\ R_5|R_4|\ R_3|R_2|\ R_1|R_0)$
$M = R_{15}$
$P = PO_{15}$
$V = PO_{15} \wedge PO_{14}$

DESNO je naredba, koja omogućuje pomeranje svih bita binarne vrednosti, određene prvim (ulaznim) operandom (PO), za jedno mesto u desno. Rezultat izvršavanja ove naredbe se odlaže u lokaciju određenu prvim (sada izlaznim) operandom. Rezultat pomeranja u desno bita binarne vrednosti ulaznog operanda je jednak njenom deljenju brojem 2, pa naredba DESNO omogućuje deljenje označenih brojeva brojem 2 (ako je deljenik negativan, tada se ispravan količnik dobije tek kada se rezultatu pomeranja doda sadržaj koga, nakon pomeranja, ima logička promenljiva P). Obrazovanje bita rezultata (R_i) od bita ulaznog operanda (PO_i, za i = 0, ... , 14) opisuju logičke funkcije:

$R_i = PO_{i+1}$
$R_{15} = PO_{15}$

Nakon izvršavanja naredbe DESNO, menjaju se i vrednosti logičkih promenljivih N, M, P i V. Njihove nove vrednosti određuju logičke funkcije:

$N = \sim(R_{15}|R_{14}|\ R_{13}|R_{12}|\ R_{11}|R_{10}|\ R_9|R_8|\ R_7|R_6|\ R_5|R_4|\ R_3|R_2|\ R_1|R_0)$
$M = R_{15}$
$P = PO_0$
$V = 0$

NAREDBE PREBACIVANJA

Procesor KONCEPT podržava više oblika naredbe prebacivanja PREBACI, koji omogućuju prebacivanje sadržaja između registara procesora i memorijskih lokacija.

PREBACI je naredba, koja omogućuje da se vrednost, određena prvim (ulaznim) operandom, prebaci u lokaciju, određenu drugim (izlaznim) operandom. Izvršavanje ove naredbe ne menja vrednosti logičkih promenljivih N, M, P i V.

UPRAVLJAČKE NAREDBE

Mašinski oblici prethodnih naredbi se smeštaju, u redosledu izvršavanja, u memorijske lokacije sa uzastopnim rastućim adresama. Zahvaljujući tome, adrese ovakvih memorijskih lokacija, koje su ujedno i **adrese** (mašinskih) **naredbi**, smeštenih u dotične memorijske lokacije, ukazuju na redosled izvršavanja ovih naredbi. Znači pomenute naredbe se izvršavaju u rastućem redosledu svojih adresa. Ovakav redosled izvršavanja naredbi je nepromenljiv i nije podesan za slučajeve u kojima obrada podataka zavisi od vrednosti obrađivanih podataka. Kada obrada

podataka zavisi od vrednosti obrađivanih podataka, potrebno je da se redosled izvršavanja naredbi određuje dinamički (u toku izvršavanja). Upravo to omogućuju **uslovne** i **bezuslovne upravljačke naredbe**.

Procesor KONCEPT podržava više uslovnih upravljačkih naredbi. Jedini operand svake uslovne upravljačke naredbe određuje adresu **ciljne naredbe** koja se izvršava nakon izvršavanja pomenute uslovne upravljačke naredbe i to samo ako je ispunjen uslov ove uslovne upravljačke naredbe. U suprotnom, izvršava se mašinska naredba koja sledi iza pomenute uslovne upravljačke naredbe. Slika 3.2.3 sadrži tabelu sa oznakama svih uslovnih upravljačkih naredbi i logičke izraze koji određuju uslov svake od ovih naredbi:

Oznaka uslovne upravljačke naredbe	uslov uslovne upravljačke naredbe
SKOČI_ZA_== ili SKOČI_ZA_N	N
SKOČI_ZA_!= ili SKOČI_ZA_NE_N	~N
SKOČI_ZA_< ili SKOČI_ZA_P	P
SKOČI_ZA_>= ili SKOČI_ZA_NE_P	~P
SKOČI_ZA_>	(~P)&(~N)
SKOČI_ZA_<=	P|N
SKOČI_ZA_±_<	M^V
SKOČI_ZA_±_>=	~(M^V)
SKOČI_ZA_±_>	(~(M^V))&(~N)
SKOČI_ZA_±_<=	(M^V) |N
SKOČI_ZA_M	M
SKOČI_ZA_NE_M	~M
SKOČI_ZA_V	V
SKOČI_ZA_NE_V	~V

Slika 3.2.3 Pregled uslovnih upravljačkih naredbi

Oznake uslovnih upravljačkih naredbi, koje se nalaze u istom redu tabele (Slika 3.2.3), predstavljaju sinonime. Jedini rezultat izvršavanja svake od uslovnih upravljačkih naredbi je, eventualno, izmena redosleda izvršavanja naredbi, dok vrednosti logičkih promenljivih N, M, P i V ostaju neizmenjene. Zbog uloge koju imaju za uslovne naredbe, logičke promenljive N, M, P i V se zajedno nazivaju i **uslovni biti** (*condition codes*).

Za uslovne upravljačke naredbe nije važno kako i kada su postavljeni uslovni biti. Ako je vrednosti ovih bita postavilo izvršavanje naredbe UPOREDI, tada

uslovne upravljačke naredbe SKOČI_ZA_== SKOČI_ZA_!=, SKOČI_ZA_<, SKOČI_ZA_>=, SKOČI_ZA_> i SKOČI_ZA_<= omogućuju dinamičke izmene redosleda izvršavanja naredbi, zavisno od važenja relacija za neoznačene cele brojeve, a uslovne upravljačke naredbe SKOČI_ZA_==, SKOČI_ZA_!=, SKOČI_ZA_±_<, SKOČI_ZA_±_>=, SKOČI_ZA_±_> i SKOČI_ZA_±_<= omogućuju isto za označene cele brojeve.

Procesor KONCEPT podržava bezuslovne upravljačke naredbe: SKOČI, POZOVI i NATRAG.

Bezuslovna upravljačka naredba SKOČI ima jedan operand. On određuje adresu ciljne naredbe koja se izvršava nakon izvršavanja naredbe SKOČI. Jedini rezultat izvršavanja upravljačke naredbe SKOČI je izmena redosleda izvršavanja naredbi (vrednosti uslovnih bita ostaju neizmenjene).

Bezuslovna upravljačka naredba POZOVI ima jedan operand. On određuje adresu ciljne naredbe koja se izvršava nakon izvršavanja naredbe POZOVI. Pored izmene redosleda izvršavanja naredbi, izvršavanje upravljačke naredbe POZOVI menja i sadržaj registra `%15`, tako što u njega smešta adresu naredbe koja sledi iza naredbe POZOVI. Adresa, smeštena u registar `%15`, se naziva **povratna adresa** (*return address*), jer omogućuje da, nakon odlaska na izvršavanje ciljne naredbe (koju određuje operand naredbe POZOVI), usledi povratak na izvršavanje naredbe koja sledi iza naredbe POZOVI. Jedini rezultat izvršavanja upravljačke naredbe POZOVI je izmena redosleda izvršavanja naredbi i smeštanje povratne adrese u registar `%15` (vrednosti uslovnih bita ostaju neizmenjene).

Bezuslovna upravljačka naredba NATRAG nema operanada. Jedini rezultat izvršavanja ove naredbe je izmena redosleda izvršavanja naredbi, tako da se izvršavanje nastavlja od naredbe čiju adresu sadrži registar `%15` (vrednosti uslovnih bita ostaju neizmenjene). Podrazumeva se da registar `%15` sadrži povratnu adresu, koju je prethodno pripremilo izvršavanje naredbe POZOVI.

3.3. ASEMBLERSKI JEZIK KONCEPT

Opis asemblerskog jezika propisuje način sastavljanja (sintaksu) asemblerskih naredbi i određuje značenje (semantiku) asemblerskih naredbi. Sintaksa se zadaje precizno i koncizno u obliku pravila izraženih pomoću *EBNF* (*Extended Backus-Naur Form*) notacije. U ovoj notaciji, ime pravila se navodi sa leve strane strelice (`->`), a sa desne strane strelice se navode prethodno uvedena imena pravila i simboli koji odgovaraju elementima jezika. *EBNF* notacija koristi razmak za razdvajanje delova pravila, vertikalnu crtu (`|`) za označavanje alternativnih delova pravila, okrugle zagrade za grupisanje delova pravila, uglaste zagrade za označavanje delova pravila koji se mogu pojaviti nijednom ili jednom i vitičaste zagrade za označavanje delova pravila koji se mogu pojaviti nijednom, jednom ili više puta. Znakovi, koje koristi *EBNF* notacija (strelica, razmak, vertikalna crta, okrugle, uglaste i vitičaste zagrade), mogu predstavljati simbole jezika, ako se navedu između navodnika. Primer pravila u *EBNF* notaciji je:

```
malo_slovo -> a|b|c|č|ć|d|đ|e|f|g|h|i|j|k|l|m|n|o|p|r|s|š|t|u|v|z|ž
```

Prethodno pravilo označava da se jedno od slova, navedenih sa desne strane strelice, koristi umesto imena `malo_slovo`. Analogno značenje ima i pravilo `cifra`:

```
cifra -> 0|1|2|3|4|5|6|7|8|9
```

Obrazovanje decimalnog broja opisuje pravilo:

```
decimalni_broj -> cifra{cifra}
```

Podrazumeva se da decimalni broj sadrži najviše do 4 cifre.

Obrazovanje heksadecimalnog broja opisuju pravila:

```
heksa_cifra -> cifra|A|B|C|D|E|F
heksadecimalni_broj -> 0x(heksa_cifra){heksa_cifra}
```

(heksadecimalni broj započinje znakovima 0x, iza kojih slede jedna ili više cifara ili velikih slova `A`, `B`, `C`, `D`, `E`, `F`). Podrazumeva se da heksadecimalni broj sadrži najviše do 6 znakova.

Obrazovanje broja opisuje pravilo:

```
broj -> decimalni_broj|heksadecimalni_broj
```

Obrazovanje labele opisuje pravilo:

```
labela -> malo_slovo{malo_slovo|cifra|_}
```

Podrazumeva se da labela može da sadrži najviše do 30 znakova i da je obavezno jedinstvena u asemblerskom programu.

Obrazovanje neposrednog operanda opisuje pravilo:

```
neposredni_operand -> ($broj|$labela)
```

Kao neposredni operand služi vrednost navedenog broja ili adresa memorijske lokacije koju označava navedena labela.

Obrazovanje direktnog operanda opisuje pravilo:

```
direktni_operand -> labela
```

Kao direktni operand služi memorijska lokacija koju označava navedena labela. Ako je direktni operand ulazni, sadržaj ove lokacije predstavlja ulaznu vrednost, a ako je direktni operand izlazni, ova lokacija se koristi kao izlazna lokacija.

Obrazovanje registarskog operanda opisuju pravila:

```
registar -> %(0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15)
registarski_operand -> registar
```

Kao registarski operand služi registar sa navedenom oznakom. Ako je registarski operand ulazni, sadržaj ovog registra predstavlja ulaznu vrednost, a ako je registarski operand izlazni, ovaj registar se koristi kao izlazna lokacija.

Obrazovanje posrednog operanda opisuje pravilo:

```
posredni_operand -> "("registar")"
```

Kao posredni operand služi memorijska lokacija čiju adresu sadrži navedeni registar. Ako je posredni operand ulazni, sadržaj adresirane lokacije predstavlja ulaznu vrednost, a ako je posredni operand izlazni, adresirana lokacija se koristi kao izlazna lokacija.

Obrazovanje indeksnog operanda opisuje pravilo:

```
indeksni_operand -> (broj|labela)"("registar")"
```

Kao indeksni operand služi memorijska lokacija čiju adresu određuje zbir dva sastojka. Jedan sastojak je navedeni broj ili adresa memorijske lokacije koju označava navedena labela, a drugi sastojak je sadržaj navedenog registra. Ako je indeksni operand ulazni, sadržaj adresirane lokacije predstavlja ulaznu vrednost, a ako je indeksni operand izlazni, adresirana lokacija se koristi kao izlazna lokacija.

Obrazovanje razmaka opisuje pravilo:

```
razmak -> " "{" "}
```

(između apostrofa se nalazi razmak).

Obrazovanje nove linije opisuje pravilo:

```
nova_linija -> "početak linije"{" "}
```

(između prvog para apostrofa se nalazi upravljački znak koji omogućuje pozicioniranje na početak linije, a između drugog para apostrofa se nalazi razmak).

Obrazovanje osnovnih asemblerskih naredbi opisuje pravilo:

```
osnovna_naredba -> nova_linija [labela:] razmak
           ((SABERI|SABERI_P) razmak registarski_operand,registarski_operand
           |(ODUZMI|ODUZMI_P) razmak registarski_operand,registarski_operand
           |UPOREDI           razmak registarski_operand,registarski_operand
           |(I|ILI)           razmak registarski_operand,registarski_operand
           |(DODAJ_1|ODBIJ_1) razmak registarski_operand
           |(NE|LEVO|DESNO)   razmak registarski_operand)
```

Obrazovanje asemblerskih naredbi prebacivanja opisuje pravilo:

```
naredba_prebacivanja -> nova_linija [labela:] razmak
           (PREBACI_RR razmak registarski_operand,registarski_operand
           |PREBACI_NR razmak neposredni_operand,registarski_operand
           |PREBACI_DR razmak direktni_operand,registarski_operand
           |PREBACI_PR razmak posredni_operand,registarski_operand
           |PREBACI_IR razmak indeksni_operand,registarski_operand
           |PREBACI_RD razmak registarski_operand,direktni_operand
           |PREBACI_RP razmak registarski_operand,posredni_operand
           |PREBACI_RI razmak registarski_operand,indeksni_operand)
```

Obrazovanje upravljačkih asemblerskih naredbi opisuje pravilo:

```
upravljačka_naredba -> nova_linija [labela:] razmak
                       ((SKOČI
                       |SKOČI_ZA_==
                       |SKOČI_ZA_!=
                       |SKOČI_ZA_<
                       |SKOČI_ZA_>=
                       |SKOČI_ZA_>
                       |SKOČI_ZA_<=
                       |SKOČI_ZA_±_<
                       |SKOČI_ZA_±_>=
                       |SKOČI_ZA_±_>
                       |SKOČI_ZA_±_<=
                       |SKOČI_ZA_N
                       |SKOČI_ZA_NE_N
                       |SKOČI_ZA_P
                       |SKOČI_ZA_NE_P
                       |SKOČI_ZA_M
                       |SKOČI_ZA_NE_M
                       |SKOČI_ZA_V
                       |SKOČI_ZA_NE_V
                       |POZOVI) razmak labela)
                       |NATRAG)
```

Upravljačke asemblerske naredbe koriste adresu labele (koja je navedena na mestu operanda) kao adresu ciljne naredbe.

Asemblerske naredbe u mašinski oblik prevodi asembler. Podrazumeva se da on smešta mašinske oblike naredbi u memorijske lokacije. Međutim, asembleru se mora izričito navesti da zauzme i eventualno inicijalizuje memorijske lokacije za promenljive. To omogućuju **asemblerske direktive**. Asemblerska direktiva `ZAUZMI` omogućuje zauzimanje više memorijskih lokacija. Njen jedini operand određuje broj memorijskih lokacija koje se zauzimaju. Asemblerska direktiva `NAPUNI` omogućuje zauzimanje jedne memorijske lokacije, čiji sadržaj određuje jedini operand ove direktive. Obrazovanje asemblerskih direktiva opisuje pravilo:

```
direktiva -> nova_linija [labela:] razmak
             (ZAUZMI|NAPUNI) razmak broj
```

Važno je uočiti da su direktive upućene asembleru (prevodiocu), a ne procesoru.

Znači, direktive nisu izvršne (one u toku asembliranja izazivaju zauzimanje memorijskih lokacija, u koje se smeštaju podaci). Za razliku od njih, naredbe su upućene procesoru, pa su, prema tome, izvršne (njima odgovaraju memorijske lokacije koje sadrže mašinske oblike naredbi).

Obrazovanje tela asemblerskog programa opisuje pravilo:

```
telo -> {direktiva
        |osnovna_naredba
        |naredba_prebacivanja
        |upravljačka_naredba}
```

Uputno je da, u telu asemblerskog programa, sve direktive prethode naredbama, radi sprečavanja da se sadržaj memorijske lokacije, predviđene za smeštanje podataka, upotrebi kao mašinski oblik naredbe.

Obrazovanje celog asemblerskog programa, ili njegovog samostalnog dela, opisuje pravilo:

```
program -> POČETAK razmak labela telo nova_linija KRAJ
```

Iza reči `POČETAK` i razamaka navodi se **ulazna labela**. Podrazumeva se da u telu asemblerskog programa ulazna labela neposredno prethodi **ulaznoj asemblerskoj naredbi** od koje započinje izvršavanje asemblerskog programa. Izvršavanje asemblerskih naredbi zatim teče sekvencijalno ka poslednjoj asemblerskoj naredbi iz njegovog tela, ako upravljačke naredbe ne izmene redosled izvršavanja naredbi.

3.4. PRIMERI ASEMBLERSKIH PROGRAMA

RAČUNANJE NAJVEĆEG ZAJEDNIČKOG DELIOCA

Računanje najvećeg zajedničkog delioca opisuje asemblerski program:

```
                POČETAK         ulaz
ulaz:           PREBACI_NR      $12,%0
                PREBACI_NR      $10,%1
ponovo:         UPOREDI         %1,%0
                SKOČI_ZA_==     kraj
                SKOČI_ZA_<      manje
veće:           ODUZMI          %1,%0
                SKOČI           ponovo
manje:          ODUZMI          %0,%1
                SKOČI           ponovo
kraj:           SKOČI           kraj
                KRAJ
```

Telo asemblerskog programa započinje opisom punjenja registara `%0` i `%1` početnim vrednostima (`PREBACI_NR`). Zatim sledi opis poređenja (`UPOREDI`) vrednosti registara `%0` i `%1`. Ako su pomenute vrednosti jednake, registri `%0` i `%1` sadrže najveći zajednički delilac, pa sledi odlazak (`SKOČI_ZA_==`) u beskonačnu petlju, koja

označava kraj programa. Ako je vrednost registra `%0` manja od vrednosti registra `%1` (`SKOČI_ZA_<`), vrednost registra `%1` se umanjuje za vrednost registra `%0`. U suprotnom, vrednost registra `%0` se umanjuje za vrednost registra `%1`. U oba slučaja, nakon izmene vrednosti veće promenljive, sledi opet poređenje vrednosti ovih promenljivih.

IZLAZAK VAN OPSEGA KOD NEOZNAČENIH CELIH BROJEVA

Otkrivanje izlaska van opsega, kod sabiranja neoznačenih celih brojeva u dvostrukoj preciznosti, opisuje asemblerski program:

```
                POČETAK         ulaz
a_donji:        NAPUNI          0xFFFF
a_gornji:       NAPUNI          0xFFFF
b_donji:        NAPUNI          0xFFFF
b_gornji:       NAPUNI          0xFFFF
greška:         NAPUNI          0
ulaz:           PREBACI_DR      a_donji,%0
                PREBACI_DR      a_gornji,%1
                PREBACI_DR      b_donji,%2
                PREBACI_DR      b_gornji,%3
                SABERI          %2,%0
                SABERI_P        %3,%1
                SKOČI_ZA_P      van_opsega
                SKOČI           kraj
van_opsega:     PREBACI_NR      $1,%4
                PREBACI_RD      %4,greška
kraj:           SKOČI           kraj
                KRAJ
```

Za promenljive `a` i `b`, čije vrednosti se sabiraju, zauzete su i inicijalizovane po dve lokacije. Za promenljivu `greška`, u koju se smešta indikacija izlaska van opsega (`0` – nije bilo izlaska van opsega, `1` – bilo je izlaska van opsega), zauzeta je jedna lokacija i inicijalizovana na vrednost `0`. Nakon prebacivanja delova sabiraka u registre (`PREBACI_DR`), sabiraju se (`SABERI`) manje značajni (donji) delovi vrednosti promenljivih `a` i `b`. Zatim se sabiraju (`SABERI_P`) značajniji (gornji) delovi vrednosti ovih promenljivih i prenos iz prethodnog sabiranja. Pojava prenosa u drugom sabiranju ukazuje na izlazak van opsega (`SKOČI_ZA_P`), što predstavlja grešku i označava se smeštanjem vrednosti `1` u promenljivu `greška`.

IZLAZAK VAN OPSEGA KOD OZNAČENIH CELIH BROJEVA

Otkrivanje izlaska van opsega, kod sabiranja označenih celih brojeva u dvostrukoj preciznosti, opisuje asemblerski program:

```
                POČETAK         ulaz
a_donji:        NAPUNI          0xFFFF
a_gornji:       NAPUNI          0xFFFF
b_donji:        NAPUNI          0xFFFF
b_gornji:       NAPUNI          0xFFFF
greška:         NAPUNI          0
ulaz:           PREBACI_DR      a_donji,%0
                PREBACI_DR      a_gornji,%1
                PREBACI_DR      b_donji,%2
                PREBACI_DR      b_gornji,%3
                SABERI          %2,%0
                SABERI_P        %3,%1
                SKOČI_ZA_V      van_opsega
                SKOČI           kraj
van_opsega:     PREBACI_NR      $1,%4
                PREBACI_RD      %4,greška
kraj:           SKOČI           kraj
                KRAJ
```

Jedina razlika ovog i prethodnog programa je u načinu otkrivanja izlaska van opsega, nakon sabiranja značajnijih delova vrednosti promenljivih `a` i `b`. U slučaju sabiranja označenih celih brojeva, na izlazak van opsega ukazuje postavljenost logičke promenljive V (`SKOČI_ZA_V`), jer se podrazumeva komplement 2 predstava označenih brojeva.

RUKOVANJE MAŠINSKOM NORMALIZOVANOM FORMOM

Rukovanje mašinskom normalizovanom formom binarne predstave realnih brojeva zahteva pristupanje pojedinim bitima lokacija. Ako se za mašinsku normalizovanu formu koristi cela lokacija (16 bita), tada prvi bit s leva može da sadrži predznak broja, narednih 8 bita mogu da sadrže podešeni eksponent (za podešavanje se koristi konstanta $10000000_2 == 2^{8-1}$), a preostalih 7 bita mogu da sadrže 7 manje značajnih bita frakcije (najznačajniji bit frakcije je uvek 1 i ne prikazuje se). Sledi primer broja predstavljenog u upravo opisanoj mašinskoj normalizovanoj formi:

$-1.5_{10} ==$
$-1.1000000_2 \times 2^0 ==$
1100000001000000_2

Zauzimanje memorijske lokacije za realnu promenljivu i smeštanje u nju prethodne vrednosti u mašinskoj normalizovanoj formi, opisuje naredna asemblerska direktiva:

```
realan:         NAPUNI          0xC040
```

Izmenu samo predznaka vrednosti prethodne realne promenljive, opisuju asemblerske naredbe:

```
                PREBACI_DR      realan,%0
                PREBACI_NR      $0x7FFF,%1
                I               %1,%0
```

a vraćanje predznaka na prethodnu vrednost, opisuje asemblerska naredba:

```
                PREBACI_NR      $0x8000,%1
                ILI             %1,%0
```

Izdvajanje eksponenta i njegovo pomeranje za 7 mesta u desno, radi njegove pripreme za aritmetiku eksponenata, opisuju asemblerske naredbe:

```
                PREBACI_DR      realan,%0
                PREBACI_NR      $0x7F80,%1
                I               %1,%0
                PREBACI_NR      $7,%2
ponovo:         DESNO           %0
                ODBIJ_1         %2
                SKOČI_ZA_NE_N   ponovo
```

RUKOVANJE LOGIČKIM VREDNOSTIMA

Dodelu (logičke) vrednosti relacionog izraza dvema promenljivim, opisuje sekvenca programa, izražena programskim jezikom C:

```
int     a,b;
char    c,d;
...
c = (a!=b);
d = c;
```

Prethodnoj sekvenci programa odgovara sekvenca asemblerskog programa:

```
a:              ZAUZMI          1
b:              ZAUZMI          1
c:              ZAUZMI          1
d:              ZAUZMI          1
                ...
                PREBACI_DR      a,%0
                PREBACI_DR      b,%1
                PREBACI_NR      $1,%2
                UPOREDI         %1,%0
                SKOČI_ZA_!=     dalje
                PREBACI_NR      $0,%2
dalje:          PREBACI_RD      %2,c
                PREBACI_RD      %2,d
```

(konstante `1` i `0` predstavljaju vrednosti logičkih promenljivih `c` i `d`).

RAČUNANJE VREDNOSTI CELOBROJNOG IZRAZA

Dodeljivanje vrednosti celobrojnog izraza promenljivoj opisuje sekvenca programa, izražena programskim jezikom C (podrazumeva se da izraz sadrži samo neoznačene cele vrednosti, za koje izlazak van opsega nije moguć):

```
unsigned a,b,c;
...
c = (a-b)*2+(a+b)/2;
```

Prethodnoj sekvenci programa odgovara sekvenca asemblerskog programa:

```
a:              ZAUZMI          1
b:              ZAUZMI          1
c:              ZAUZMI          1
                ...
                PREBACI_DR      a,%0
                PREBACI_DR      b,%1
                PREBACI_DR      a,%2
                ODUZMI          %1,%2
                LEVO            %2
                SABERI          %1,%0
                DESNO           %0
                SABERI          %0,%2
                PREBACI_RD      %2,c
```

RUKOVANJE NIZOVIMA

Brojanje dana sa srednjom dnevnom temperaturom iz posmatranog intervala temperatura, i to u određenom periodu godine, opisuje sekvenca programa, izražena programskim jezikom C:

```
int     t[365];
int     prvi,poslednji,donja,gornja,i;
int     broj = 0;
...
for (i=prvi;i<=poslednji;i++)
 if ((t[i]>=donja)&&(t[i]<=gornja))
  broj++;
```

(tabela `t` sadrži srednje dnevne temperature za 365 dana u godini, promenljive `prvi` i `poslednji` sadrže redne brojeve dana na granicama zadanog perioda u godini, promenljive `donja` i `gornja` sadrže temperaturne granice posmatranog intervala temperatura, promenljiva `i` služi kao indeks elemenata tabele `t`, a promenljiva `broj` sadrži broj dana u određenom periodu godine, u kojima je srednja dnevna temperatura iz posmatranog intervala). Prethodnoj sekvenci programa odgovara sekvenca asemblerskog programa:

```
t:              ZAUZMI          365
prvi:           ZAUZMI          1
poslednji:      ZAUZMI          1
donja:          ZAUZMI          1
gornja:         ZAUZMI          1
broj:           ZAUZMI          1
                ...
                PREBACI_NR      $0,%0
                PREBACI_DR      prvi,%1
                PREBACI_DR      poslednji,%2
                PREBACI_DR      donja,%3
                PREBACI_DR      gornja,%4
provera:        UPOREDI         %2,%1
                SKOČI_ZA_>      izlaz
ponovo:         UPOREDI         %3,t(%1)
                SKOČI_ZA_±_<    naredni
                UPOREDI         %4,t(%1)
                SKOČI_ZA_±_>    naredni
                DODAJ_1         %0
naredni:        DODAJ_1         %1
                SKOČI           provera
izlaz:          PREBACI_RD      %0,broj
```

(naredba sa labelom `provera` odgovara proveri kraja `for` iskaza, naredba sa labelom `ponovo` i četiri naredbe iza nje odgovaraju `if` iskazu, a naredba sa labelom `naredni` odgovara povećanju indeksa `for` iskaza).

RUKOVANJE SLOGOVIMA

Dodelu vrednosti slogu, koji sadrži predstavu vremena, izraženu brojem sati, minuta i sekundi, opisuje sekvenca programa, izražena programskim jezikom C:

```
struct vreme
 {unsigned sat, minut, sekund;} a,b;
       ...
b = a;
```

Prethodnoj sekvenci programa odgovara sekvenca asemblerskog programa:

```
a:              ZAUZMI          3
b:              ZAUZMI          3
                ...
                PREBACI_NR      $0,%0
                PREBACI_IR      a(%0),%1
                PREBACI_RI      %1,b(%0)
                DODAJ_1         %0
                PREBACI_IR      a(%0),%1
                PREBACI_RI      %1,b(%0)
                DODAJ_1         %0
                PREBACI_IR      a(%0),%1
                PREBACI_RI      %1,b(%0)
```

Za slog promenljive `a` i `b` su zauzete po tri memorijske lokacije (za svako polje po jedna lokacija). Registar `%o` sadrži relativnu udaljenost polja ovih promenljivih od početnih memorijskih lokacija ovih promenljivih.

3.5. POTPROGRAM

Potprogram omogućuje opisivanje izračunavanja vrednosti, odnosno opisivanje rukovanja strukturama podataka (vrednostima složenih tipova). **Opis** potprograma prate njegovi **pozivi**. Razdvajanje opisa i poziva potprograma omogućuje korišćenje (poziv) potprograma samo na osnovu poznavanja njegove namene, a bez poznavanja njegovog opisa. To je naročito važno za postepen razvoj programa, jer znači da se svaki potprogram može koristiti (pozivati) i pre dovršenja njegovog opisa, pod uslovom da je njegova namena precizirana.

Opštost opisu potprograma daje upotreba **parametara**, koja omogućuje da se potprogram odnosi na sve vrednosti iz skupa vrednosti, određenog tipom parametara. Svako izvršavanje potprograma se odnosi na konkretne vrednosti ovih parametara, koje se nazivaju **argumenti**. Njih određuje poziv potprograma. Do izvršavanja potprograma dolazi nakon njegovog poziva. U pozivu potprograma se odredi povratna adresa, od koje se nastavlja izvršavanje programa, nakon izvršavanja potprograma. Izvršavanje potprograma započinje u **ulaznoj tački** opisa potprograma, a završava u **izlaznoj tački** ovog opisa. U ulaznoj tački potprograma se zauzimaju lokacije za **lokalne promenljive** potprograma, a u izlaznoj tački potprograma se ove lokacije oslobađaju. Lokalne promenljive potprograma se nazivaju i **dinamičke**, jer postoje samo u toku izvršavanja potprograma, za razliku od **statičkih** ili **globalnih** promenljivih koje postoje za sve vreme izvršavanja programa i koje su definisane van potprograma. **Područje važenja** (*scope*) lokalnih promenljivih je od mesta njihove definicije do kraja potprograma, a područje važenja globalnih promenljivih je od mesta njihove definicije do kraja programa.

Potprogrami se dele na **funkcije** i **procedure**. Funkcije omogućuju opisivanje izračunavanja vrednosti, odnosno preuzimanje vrednosti promenljive. One imaju jedan ili više **ulaznih parametara** (sa statusom lokalnih promenljivih) i jednu **povratnu vrednost**. Povratna vrednost funkcije se koristi u izrazu, koji sadrži poziv funkcije.

Procedure omogućuju opisivanje izmena vrednosti promenljivih. One mogu da imaju više ulaznih parametara, ali i više **ulazno-izlaznih parametara**.

Računanje najvećeg zajedničkog delioca može biti opisano u obliku funkcije. U nastavku je naveden ovakav opis funkcije, izražen programskim jezikom C:

```
unsigned nzd(unsigned a, unsigned b)
{while (a!=b)
  if (a>b)
   a = a-b;
  else
   b = b-a;
 return(a);};
```

Ulazni parametri `a` i `b`, opisa ove funkcije, određuju vrednosti za koje se računa najveći zajednički delilac. Najveći zajednički delilac se vraća kao povratna vrednost funkcije `nzd`. Ako su `x`, `y` i `z` celobrojne promenljive, tada poziv funkcije `nzd`:

```
z = nzd(12,10);
```

dodeljuje promenljivoj `z` najveći zajednički delilac celih brojeva `12` i `10`, a poziv iste funkcije:

```
z = nzd(x,y);
```

dodeljuje promenljivoj `z` najveći zajednički delilac vrednosti promenljivih `x` i `y`.

Računanje najvećeg zajedničkog delioca može biti opisano i u obliku procedure. U nastavku je naveden ovakav opis procedure, izražen programskim jezikom C:

```
void nzd(unsigned a, unsigned b, unsigned *c)
{while (a!=b)
  if (a>b)
   a = a-b;
  else
   b = b-a;
 *c = a;};
```

Ulazni parametri `a` i `b`, opisa ove procedure, određuju vrednosti za koje se računa najveći zajednički delilac. Najveći zajednički delilac se dodeljuje ulazno-izlaznom parametru `c`. Ako su `x`, `y` i `z` celobrojne promenljive, tada poziv procedure `nzd`:

```
nzd(12,10,&z);
```

dodeljuje promenljivoj `z` najveći zajednički delilac celih brojeva `12` i `10`, a poziv iste procedure:

```
nzd(x,y,&z);
```

dodeljuje promenljivoj `z` najveći zajednički delilac vrednosti promenljivih `x` i `y`.

ASEMBLERSKI OBLIK POTPROGRAMA

Potprogrami nisu isključivo vezani za programske jezike visokog nivoa. Opisi i pozivi potprograma se mogu javiti i u asemblerskom programu, ali su detalji opisivanja i poziva potprograma prepušteni programeru. On se mora pobrinuti za prenos argumenata, za odlazak na izvršavanje potprograma i za povratak iz potprograma, nakon njegovog izvršavanja. Asemblerski jezik KONCEPT, olakšava

zadatak programera, jer sadrži upravljačku naredbu **POZOVI**, koja omogućuje odlazak na izvršavanje potprograma, ali i smeštanje povratne adrese u registar **%15**. Zahvaljujući tome, povratak iz potprograma omogućuje upravljačka naredba **NATRAG**.

Registri procesora KONCEPT (uz izuzetak registra **%15**) mogu biti iskorišćeni za prenos argumenata, za šta je neophodna usaglašenost opisa i poziva potprograma.

Uz pretpostavku da se registar **%1** koristi za smeštanje vrednosti prvog ulaznog parametra, registar **%2** za smeštanje vrednosti drugog ulaznog parametra, a registar **%0** za smeštanje povratne vrednosti funkcije, asemblerski ekvivalent opisa funkcije **nzd** izgleda:

```
nzd:        UPOREDI       %2,%1
            SKOČI_ZA_=    kraj
            SKOČI_ZA_<    manje
veće:       ODUZMI        %2,%1
            SKOČI         nzd
manje:      ODUZMI        %1,%2
            SKOČI         nzd
kraj:       PREBACI_RR    %1,%0
            NATRAG
```

Ako su za celobrojne promenljive **x**, **y** i **z** zauzete i inicijalizovane memorijske lokacije:

```
x:          NAPUNI        9
y:          NAPUNI        3
z:          ZAUZMI        1
```

tada asemblerske naredbe:

```
            PREBACI_NR    $12,%1
            PREBACI_NR    $10,%2
            POZOVI        nzd
            PREBACI_RD    %0,z
```

opisuju poziv asemblerskog ekvivalenta funkcije **nzd**, u kome se izračuna najveći zajednički delilac celih brojeva **10** i **12**, a povratna vrednost ove funkcije dodeli promenljivoj **z**. U prethodnom primeru, u registre **%1** i **%2** se smeste argumenti pre odlaska na izvršavanje potprograma, a nakon povratka iz potprograma sadržaj registra **%0** (povratna vrednost) se smesti u promenljivu **z**. Slično, asemblerske naredbe:

```
            PREBACI_DR    x,%1
            PREBACI_DR    y,%2
            POZOVI        nzd
            PREBACI_RD    %0,z
```

opisuju poziv asemblerskog ekvivalenta funkcije `nzd`, u kome se računa najveći zajednički delilac vrednosti promenljivih `x` i `y`, a povratna vrednost ove funkcije dodeli promenljivoj `z`.

Uz pretpostavku da se registar `%1` koristi za smeštanje vrednosti prvog ulaznog parametra, registar `%2` za smeštanje vrednosti drugog ulaznog parametra, a registar `%3` za smeštanje adrese trećeg ulazno-izlaznog parametra, asemblerski ekvivalent opisa procedure `nzd` izgleda:

```
nzd:            UPOREDI         %2,%1
                SKOČI_ZA_=      kraj
                SKOČI_ZA_<      manje
veće:           ODUZMI          %2,%1
                SKOČI           nzd
manje:          ODUZMI          %1,%2
                SKOČI           nzd
kraj:           PREBACI_RP      %1,(%3)
                NATRAG
```

Ako su za celobrojne promenljive `x`, `y` i `z` zauzete i inicijalizovane memorijske lokacije:

```
x:              NAPUNI          9
y:              NAPUNI          3
z:              ZAUZMI          1
```

tada asemblerske naredbe:

```
                PREBACI_NR      $12,%1
                PREBACI_NR      $10,%2
                PREBACI_NR      $z,%3
                POZOVI          nzd
```

opisuju poziv asemblerskog ekvivalenta procedure `nzd`, u kome se izračuna najveći zajednički delilac celih brojeva `10` i `12` i dodeli ulazno-izlaznoj promenljivoj `z`. U prethodnom primeru, pre odlaska na izvršavanje potprograma, u registre `%1` i `%2` se smeste vrednosti čiji najveći zajednički delilac se računa, a u registar `%3` se smesti adresa memorijske lokacije, namenjene za prihvatanje izračunatog najvećeg zajedničkog delioca. Slično prethodnom, opis poziva asemblerskog ekvivalenta procedure `nzd`, u kome se izračuna najveći zajednički delilac vrednosti promenljivih `x` i `y` i dodeli ulazno-izlaznoj promenljivoj `z`, sadrže asemblerske naredbe:

```
                PREBACI_DR      x,%1
                PREBACI_DR      y,%2
                PREBACI_NR      $z,%3
                POZOVI          nzd
```

3.6. MAKRO

Korišćenje potprograma uvek povećava broj izvršavanih naredbi, čime se produžava vreme izvršavanja programa. Pored toga, korišćenje potprograma nekad izaziva i povećanje dužine programa, što znači da treba više memorijskih lokacija za smeštanje mašinskog oblika programa. Prethodno se može pokazati na primeru opisa i poziva asemblerskog ekvivalenta procedure `nzd`:

```
x:              NAPUNI          9
y:              NAPUNI          3
z:              ZAUZMI          1
                ...
nzd:            UPOREDI         %2,%1
                SKOČI_ZA_==     kraj
                SKOČI_ZA_<      manje
veće:           ODUZMI          %2,%1
                SKOČI           nzd
manje:          ODUZMI          %1,%2
                SKOČI           nzd
kraj:           PREBACI_RP      %1,(%3)
                NATRAG
                ...
                PREBACI_DR      x,%1
                PREBACI_DR      y,%2
                PREBACI_NR      $z,%3
                POZOVI          nzd
```

Ako bi poziv potprograma bio zamenjen opisom potprograma, tada bi prethodnoj sekvenci asemblerskog programa odgovarala sledeća sekvenca:

```
x:              NAPUNI          9
y:              NAPUNI          3
z:              ZAUZMI          1
                ...
nzd:            PREBACI_DR      x,%1
                PREBACI_DR      y,%2
                PREBACI_NR      $z,%3
nzd1:           UPOREDI         %2,%1
                SKOČI_ZA_==     kraj
                SKOČI_ZA_<      manje
veće:           ODUZMI          %2,%1
                SKOČI           nzd1
manje:          ODUZMI          %1,%2
                SKOČI           nzd1
kraj:           PREBACI_RP      %1,(%3)
```

Potprogram, za koga se podrazumeva da se na mestu njegovog poziva pojave naredbe iz njegovog opisa, se naziva **makro** (veliki) potprogram.

Iz prethodnih primera sledi da se, po svakom pozivu, izvrše 2 naredbe manje u slučaju makro potprograma, nego u slučaju običnog potprograma. Zavisnost broja naredbi (odnosno dužine) programa od broja poziva potprograma za ova dva slučaja je prikazana, za prethodni primer, tabelarno (Slika 3.6.1).

broj poziva	broj naredbi za običan potprogram	broj naredbi za makro potprogram
1	13	11
2	17	22
3	21	33

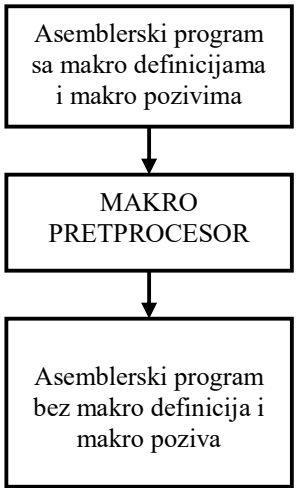
Slika 3.6.1 Zavisnost broja naredbi od broja poziva asemblerskog ekvivalenta potprograma `nzd`

Znači, sa stanovišta dužine programa, u posmatranim slučajevima (Slika 3.6.1) bolji je makro, ako se potprogram koristi samo jedan put.

Upotreba makroa podrazumeva da postoji **makro definicija** sa opisom potprograma i **makro poziv**, koji označava mesto korišćenja makroa. Ako makro definicija, radi opštosti, zahteva parametre, tada odgovarajući makro poziv sadrži adekvatne argumente. Oni zamenjuju parametre u makro definiciji, a onda tako modifikovani opis zamenjuje makro poziv u tekstu programa.

Korišćenje makroa uvek dovodi do skraćenja vremena izvršavanja programa, a dovodi i do smanjenja dužine programa, ako je u programu prisutan samo jedan poziv makroa.

Zamenu ili **supstituciju** makro poziva, eventualno modifikovanim, naredbama iz makro definicije obavlja poseban program koji se naziva **makro pretprocesor** (*macro preprocessor*). On procesira (obrađuje) programski tekst pre asemblera i tako, praveći tekstualne izmene, priprema tekst asemblerskog programa za asembliranje (Slika 3.6.2).



Slika 3.6.2 Uloga makro pretprocesora

U toku izmena programskog teksta, makro pozive zamenjuju naredbe iz makro definicija. Tome eventualno prethodi modifikacija ovih naredbi. U toku nje parametre iz makro definicija zamenjuju argumenti iz makro poziva. Zadatak makro pretprocesora se znatno pojednostavljuje, ako u tekstu asemblerskog programa svaka makro definicija prethodi svojim makro pozivima. Makro definicije i makro pozivi su namenjeni makro pretprocesoru (kao što su direktive namenjene asembleru), pa zato makro pozivi nisu izvršni (kao ni direktive). Zbog toga, makro definicije mogu sadržati i direktive, koje zamenjuju odgovarajuće makro pozive.

OPIS MAKRO DEFINICIJA I MAKRO POZIVA

Za opis obrazovanja makro definicija i makro poziva zgodna je *EBNF* notacija. Tako, pravilo:

```
veliko_slovo -> A|B|C|Č|Ć|D|Đ|E|F|G|H|I|J|K|L|M|N|O|P|R|S|Š|T|U|V|Z|Ž
```

zadaje repertoar znakova, potreban za obrazovanje makro imena. Obrazovanje makro imena opisuje pravilo:

```
ime -> veliko_slovo{veliko_slovo|cifra|_}
```

Podrazumeva se da makro ime sadrži najviše do 30 znakova i da je obavezno jedinstveno u asemblerskom programu.

Obrazovanje makro parametara opisuje pravilo:

```
parametar -> labela|ime
```

Parametar je obavezno jedinstven u makro definiciji.

Obrazovanje makro argumenata opisuje pravilo:

```
argument -> (broj|parametar)[(+|-|*|/)(broj|parametar)]
            |registar
```

Neophodno je da argument bude usaglašen sa ulogom parametra koga zamenjuje. Kada se kao argument pojavi celobrojni izraz, čija oba operanda su brojevi, tada makro pretprocesor izračuna izraz i dobijenu vrednost koristi kao argument.

Obrazovanje makro definicije opisuje pravilo:

```
makro_definicija -> nova_linija ime razmak MAKRO razmak {parametar[,]}
                    telo nova_linija KRAJ
```

(telo makro definicije se ne razlikuje od tela asemblerskog programa).

Obrazovanje makro poziva opisuje pravilo:

```
makro_poziv -> nova_linija [labela:] razmak ime razmak {argument[,]}
```

Obrazovanje proširenog tela asemblerskog programa, odnosno proširenog tela makro definicije, opisuje pravilo:

```
telo -> {direktiva
        |osnovna_naredba
        |naredba_prebacivanja
        |upravljačka_naredba
        |makro_definicija
        |makro_poziv}
```

U makro pozivu se koristi ime prethodno definisanog makroa, sa čijim parametrima se, po broju i mestu, slažu argumenti poziva. Makro pozive zamenjuju tela makroa, u kojima su parametri zamenjeni argumentima. Parametri se mogu pojaviti svugde u telu makro definicije (na primer, na mestu: labele, imena naredbe, imena direktive, operanda, parametra ili argumenta).

PRIMERI MAKRO DEFINICIJA I MAKRO POZIVA

Makro definicija:

```
NZD             MAKRO           a,b,c
                PREBACI_DR      a,%1
                PREBACI_DR      b,%2
                PREBACI_NR      $c,%3
nzd:            UPOREDI         %2,%1
                SKOČI_ZA_==     kraj
                SKOČI_ZA_<      manje
veće:           ODUZMI          %2,%1
                SKOČI           nzd
manje:          ODUZMI          %1,%2
                SKOČI           nzd
kraj:           PREBACI_RP      %1,(%3)
                KRAJ
```

opisuje računanje najvećeg zajedničkog delioca. Makro poziv:

```
                NZD             x,y,z
```

zamenjuju naredbe:

```
                PREBACI_DR      x,%1
                PREBACI_DR      y,%2
                PREBACI_NR      $z,%3
nzd:            UPOREDI         %2,%1
                SKOČI_ZA_==     kraj
                SKOČI_ZA_<      manje
veće:           ODUZMI          %2,%1
                SKOČI           nzd
manje:          ODUZMI          %1,%2
                SKOČI           nzd
kraj:           PREBACI_RP      %1,(%3)
```

Višestruki pozivi makroa `NZD` u jednom asemblerskom programu izazivaju višestruku pojavu istih labela (navedenih u telu ovog makroa). Ovaj problem se rešava ili korišćenjem parametara na mestu spornih labela, ili prepuštanjem makro pretprocesoru da, u ovakvim slučajevima, sam generiše jedinstvene labele.

MAKRO DEFINICIJA SA MAKRO DEFINICIJOM

Telo makro definicije može sadržati drugu makro definiciju. Pri tome poziv unutrašnjeg makroa obavezno sledi iza poziva vanjskog makroa, jer je tek tada unutrašnja makro definicija potpuno oblikovana (njen konačan izgled, u opštem slučaju, zavisi od argumenata poziva vanjskog makroa). To znači, na primer, da je moguće napisati opštu makro definiciju koja opisuje aritmetiku u dvostrukoj preciznosti. Na osnovu ovakve definicije mogu se stvoriti makro definicije koje opisuju sabiranje ili oduzimanje u dvostrukoj preciznosti. Prethodno ilustruje makro definicija:

```
DVOSTRUKO       MAKRO           NAZIV,OPERACIJA1,OPERACIJA2
NAZIV           MAKRO
                OPERACIJA1      %2,%0
                OPERACIJA2      %3,%1
                KRAJ
                KRAJ
```

(prethodna makro definicija podrazumeva da se u registrima `%0` i `%1` nalaze manje značajni i značajniji deo prvog broja u dvostrukoj preciznosti, a da se u registrima `%2` i `%3` nalaze manje značajni i značajniji deo drugog broja u dvostrukoj preciznosti, kao i da se zanemaruje izlazak van opsega). Makro poziv:

```
                DVOSTRUKO       SABERI_2,SABERI,SABERI_P
```

zamenjuje makro definicija:

```
SABERI_2        MAKRO
                SABERI          %2,%0
                SABERI_P        %3,%1
                KRAJ
```

koja opisuje sabiranje u dvostrukoj preciznosti, a makro poziv:

```
                DVOSTRUKO       ODUZMI_2,ODUZMI,ODUZMI_P
```

zamenjuje makro definicija:

```
ODUZMI_2        MAKRO
                ODUZMI          %2,%0
                ODUZMI_P        %3,%1
                KRAJ
```

koja opisuje oduzimanje u dvostrukoj preciznosti.

USLOVNO ASEMBLIRANJE

Makro pretprocesor omogućuje oblikovanje, ne samo makro definicija, nego i drugih delova asemblerskog programa, ako podržava uslovno asembliranje. Za uslovno asembliranje je potrebna uslovna direktiva koja određuje pod kojim uslovom makro pretprocesor propušta na asembliranje deo asemblerskog programa, sadržan u telu uslovne direktive. Ako navedeni uslov nije ispunjen, tada se deo asemblerskog programa iz tela uslovne direktive ne uključuje u asemblerski program.

Obrazovanje uslova uslovne direktive opisuje pravilo:

```
uslov -> (broj|parametar)(==|!=|>|<|>=|<=)(broj|parametar)
```

(uslov uslovne direktive ima oblik relacije, od čijeg važenja zavisi tačnost uslova).

Obrazovanje uslovne direktive opisuje pravilo:

```
uslovna_direktiva -> nova_linija USLOVNO razmak uslov telo nova_linija KRAJ
```

Telo uslovne direktive se obrazuje na isti način kao i telo asemblerskog programa ili makro definicije. Uslovna direktiva je upućena makro pretprocesoru (znači, nije izvršna).

Obrazovanje proširenog tela asemblerskog programa, makro definicije i uslovne direktive opisuje pravilo:

```
telo -> {direktiva
        |osnovna_naredba
        |naredba_prebacivanja
        |upravljačka_naredba
        |makro_definicija
        |makro_poziv
        |uslovna_direktiva }
```

Pomoću uslovne direktive je moguće, na primer, opisati obrazovanje tabele, čiji elementi sadrže vrednosti prvih n stepena broja 2. Obrazovanje ovakve tabele opisuje naredna makro definicija koja sadrži rekurzivne makro pozive. Kraj rekurzije nastupa kada prestane važiti uslov uslovne direktive.

```
TABELA          MAKRO           stepen
                USLOVNO         stepen > 0
                TABELA          stepen/2
                NAPUNI          stepen
                KRAJ
                KRAJ
```

Makro poziv:

```
stepeni:        TABELA          8
```

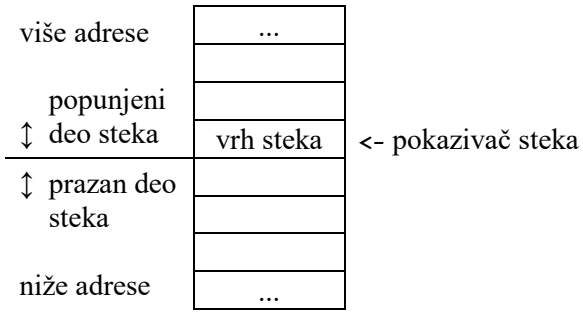
zamenjuju direktive:

```
stepeni:        NAPUNI          1
                NAPUNI          2
                NAPUNI          4
                NAPUNI          8
```

3.7. STEK

U naredbi `POZOVI` je predviđeno čuvanje samo jedne povratne adrese (jer samo toliko prostora ima u registru `%15`). Zato ova naredba direktno ne podržava poziv potprograma iz potprograma, ili rekurzivne pozive potprograma, jer se tada mora sačuvati više (unapred nepoznat broj) povratnih adresa. Povratne adrese se koriste u obrnutom redosledu od redosleda svog nastanka, jer nastaju u pozivu potprograma, a koriste se pri povratku iz potprograma. Znači, kao prva se koristi povratna adresa koja je poslednja nastala (pošto se prvi završava poslednje pozvani potprogram), a kao poslednja povratna adresa koja je prva nastala. Opisani redosled nastajanja i korišćenja povratnih adresa dozvoljava da se za njihovo privremeno čuvanje koristi niz memorijskih lokacija, u koji se povratne adrese smeštaju u jednom smeru, a iz koga se povratne adrese preuzimaju u drugom smeru (podrazumeva se da su adrese memorijskih lokacija iz ovog niza uzastopne i rastuće). Ako se povratne adrese smeštaju u pomenuti niz od njegovog kraja ka njegovom početku, a preuzimaju iz pomenutog niza u suprotnom smeru, počev od lokacije, do koje je niz bio popunjen, tada je za rukovanje nizom jedino važno znati adresu lokacije, do koje je niz popunjen. Od ove lokacije se nastavlja punjenje niza ka njegovom početku, odnosno, od nje se niz prazni ka njegovom kraju. Drugim rečima, sve napunjene lokacije niza su koncentrisane na njegovom kraju, a sve slobodne lokacije na njegovom početku. Ovako organizovan niz memorijskih lokacija se naziva **stek** (*stack*). Numerička adresa lokacije, do koje je stek popunjen, se obično čuva u registru, koji se naziva **pokazivač steka** (*stack pointer*).

Slika 3.7.1 sadrži prikaz izgleda steka.



Slika 3.7.1 Izgled steka

Rukovanje stekom obuhvata: zauzimanje memorijskih lokacija za stek, pripremu steka za korišćenje (odnosno, inicijalizaciju pokazivača steka), smeštanje sadržaja u stek i preuzimanje sadržaja iz steka. Opšti opis rukovanja stekom sadrži naredna makro definicija (za nju se podrazumeva da je registar `%1` radni, odnosno da se njegov sadržaj može slobodno menjati.):

```
STEK            MAKRO           veličina,pokazivač_steka
stek:           ZAUZMI          veličina
PRIPREMI_STEK   MAKRO
                PREBACI_NR      $stek,pokazivač_steka
                PREBACI_NR      $veličina,%1
                SABERI          %1,pokazivač_steka
                KRAJ
NA_STEK         MAKRO           registar
                ODBIJ_1         pokazivač_steka
                PREBACI_RP      registar,(pokazivač_steka)
                KRAJ
SA_STEKA        MAKRO           registar
                PREBACI_PR      (pokazivač_steka),registar
                DODAJ_1         pokazivač_steka
                KRAJ
                KRAJ
```

Jedina dva parametra prethodne makro definicije služe za određivanje: veličine steka i oznake registra pokazivača steka. Oni omogućuju potpuno oblikovanje makro definicija, sadržanih u makro definiciji `STEK`. Makro definicija `STEK` je napravljena uz pretpostavku da je broj lokacija steka uvek veći od broja sadržaja koji se smeštaju na stek. Takođe, podrazumeva se da se stek ispravno koristi, odnosno da se sa steka preuzimaju samo u njega smešteni sadržaji.

Makro poziv:

```
                STEK            0x100,%12
```

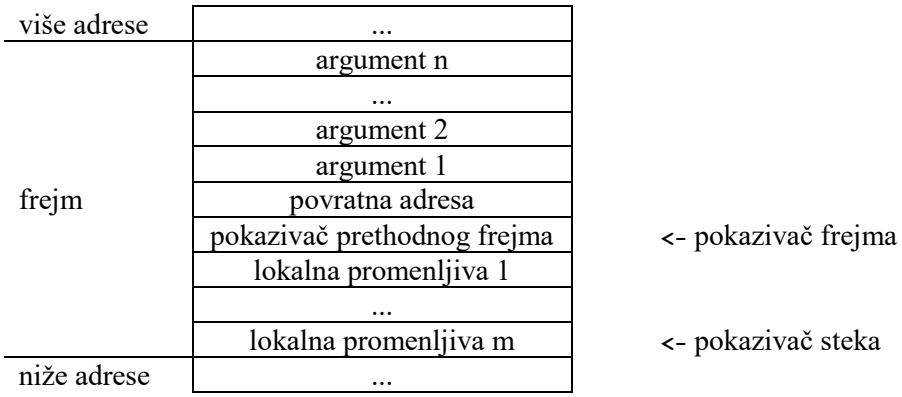
zamenjuju sledeća direktiva i makro definicije:

```
stek:           ZAUZMI          0x100
PRIPREMI_STEK   MAKRO
                PREBACI_NR      $stek,%12
                PREBACI_NR      $0x100,%1
                SABERI          %1,%12
                KRAJ
NA_STEK         MAKRO           registar
                ODBIJ_1         %12
                PREBACI_RP      registar,(%12)
                KRAJ
SA_STEKA        MAKRO           registar
                PREBACI_PR      (%12),registar
                DODAJ_1         %12
                KRAJ
```

koje dovode do zauzimanja 256 memorijskih lokacija (100_{16}) za stek, do određivanja registra `%12` kao pokazivača steka i do oblikovanja makro definicija `PRIPREMI_STEK`, `NA_STEK` i `SA_STEKA` (korišćenje prve od njih obavezno prethodi korišćenju poslednje dve).

FREJM

Stek se koristi, ne samo za čuvanje povratnih adresa, nego i za prenos argumenata u potprogram. Skup lokacija na steku, koje sadrže povratnu adresu i argumente se naziva **frejm** (*stack frame*). Frejm nastaje prilikom poziva potprograma, a nestaje po povratku iz potprograma. Lokacijama frejma se može pristupati posredstvom pokazivača steka, ali tada probleme stvaraju izmene pokazivača steka (koje stalno menjaju udaljenost frejma od vrha steka), kao i istovremeno postojanje više frejmova na steku. Zato je bolje koristiti poseban registar za pristup lokacijama frejma. Takav registar se naziva **pokazivač frejma** (*base pointer*). On pokazuje na baznu lokaciju ili **bazu frejma**. Da bi se za svaki od istovremeno postojećih frejmova mogao odrediti njegov pokazivač, zgodno je uvezati frejmove u listu. To se postiže ako baza najmlađeg frejma sadrži pokazivač prethodnog frejma i tako dalje. Podrazumeva se da aktivni pokazivač frejma pokazuje na bazu najmlađeg frejma. Pored povratne adrese, argumenata i pokazivača prethodnog frejma, frejm može da sadrži i lokalne (dinamičke) promenljive. Slika 3.7.2 sadrži prikaz mogućeg izgleda ovakvog frejma.



Slika 3.7.2 Izgled frejma

Za pristup lokacijama frejma (Slika 3.7.2) zgodno je indeksno adresiranje, koje koristi pokazivač frejma i relativnu udaljnost lokacije frejma od njegove baze. Ako registar `%11` sadrži pokazivač frejma, tada poređenje argumenta 1 i argumenta 2 (Slika 3.7.2) opisuju naredbe:

```
PREBACI_IR     2(%11),%0
PREBACI_IR     3(%11),%1
UPOREDI        %1,%0
```

U prethodnom, a i u sledećem primeru se podrazumeva da su registri `%0` i `%1` radni, odnosno da se njihovi sadržaji mogu slobodno menjati.

Rukovanje frejmom je vezano za poziv potprograma. Ono se može objasniti na primeru za poziv procedure `nzd`, (izražene programskim jezikom C):

```
nzd(x,y,&z);
```

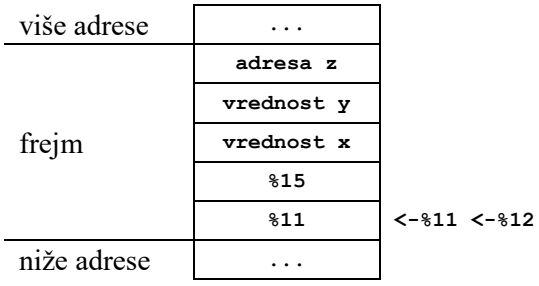
Rukovanje frejmom za prethodni poziv opisuje asemblerska sekvenca:

```
x:             ZAUZMI         1
y:             ZAUZMI         1
z:             ZAUZMI         1
               ...
               PREBACI_NR     $z,%0
               NA_STEK        %0
               PREBACI_DR     y,%0
               NA_STEK        %0
               PREBACI_DR     x,%0
               NA_STEK        %0
               NA_STEK        %15
               POZOVI         nzd
               SA_STEKA       %15
               PREBACI_NR     $3,%0
               SABERI         %0,%12
```

Podrazumeva se da registar `%12` sadrži pokazivač steka, da registar `%11` sadrži pokazivač frejma i da pozvani asemblerski ekvivalent procedure `nzd` izgleda:

```
nzd:        NA_STEK       %11
            PREBACI_RR    %12,%11
            PREBACI_IR    2(%11),%0
            PREBACI_IR    3(%11),%1
ponovo:     UPOREDI       %1,%0
            SKOČI_ZA_==   kraj
            SKOČI_ZA_<    manje
veće:       ODUZMI        %1,%0
            SKOČI         ponovo
manje:      ODUZMI        %0,%1
            SKOČI         ponovo
kraj:       PREBACI_IR    4(%11),%1
            PREBACI_RP    %0,(%1)
            SA_STEKA      %11
            NATRAG
```

Slika 3.7.3 sadrži prikaz stanja na steku, neposredno pre izvršavanja naredbe `UPOREDI`.



Slika 3.7.3 Primer frejma

Gornje (prve) 4 lokacije frejma (Slika 3.7.3) nastanu izvršavanjem naredbi koje prethode naredbi `POZOVI` iz poziva potprograma `nzd`, a preostala lokacija nastane izvršavanjem prve naredbe iz ovoga potprograma. Izvršavanje pretposlednje naredbe ovoga potprograma poništi zadnju lokaciju frejma, a izvršavanja poslednje tri naredbe iz poziva ovoga potprograma ponište prve 4 lokacije frejma.

Korišćenje steka može da uzrokuje **dinamičke greške** u izvršavanju programa (na primer, ako se ispostavi da, za neko izvršavanje programa, broj lokacija steka nije dovoljan za smeštanje svih sadržaja).

U arhitekturi naredbi procesora KONCEPT nisu predviđene posebne naredbe, a ni posebna adresiranja za rukovanje stekom, da bi ovaj procesor bio što jednostavniji. Kada bi procesor KONCEPT podržavao stek, tada bi u njegovom repertoaru naredbi trebalo da postoje naredbe koje su ekvivalentne makroima

`NA_STEK` i `SA_STEKA`. U tom slučaju bi naredbe `POZOVI` i `NATRAG` trebalo da koriste stek umesto registra `%15`.

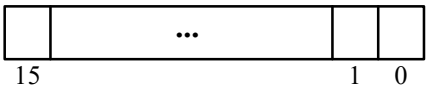
3.8. PITANJA

1. Koji programski jezici niskog i visokog nivoa postoje?
2. U čemu se razlikuju asemblerski i mašinski jezici?
3. Kada je opravdano korišćenje asemblerskog i mašinskog jezika?
4. Koliki adresni prostor omogućuju 2 bita?
5. Koje naredbe omogućuju aritmetiku u višestrukoj preciznosti?
6. Koje naredbe u toku svog izvršavanja ne menjaju uslovne bite?
7. Koje aritmetičke naredbe postoje?
8. Koje naredbe za rukovanje bitima postoje?
9. Koje upravljačke naredbe postoje?
10. Koje kombinacije operanada koriste naredbe prebacivanja?
11. Koje naredbe omogućuju otkrivanje izlaska van opsega?
12. Koje naredbe imaju dva operanda?
13. Koje naredbe imaju jedan operand?
14. Koje naredbe su bez operanada?
15. Koje asemblerske direktive postoje?
16. Šta karakteriše potprogram?
17. Šta karakteriše makro?
18. Šta karakteriše funkciju?
19. Šta karakteriše proceduru?
20. Koje asemblerske naredbe su uvedne radi asemblerskih potprograma?
21. Šta je vezano za poziv asemblerskog potprograma?
22. Šta je vezano za makro poziv?
23. Kako upotreba potprograma utiče na dužinu programa i vreme njegovog izvršavanja?
24. Kako upotreba makroa utiče na dužinu programa i vreme njegovog izvršavanja?
25. Ko obavlja zamenu makro poziva modifikovanim naredbama iz makro definicija?
26. Gde su definisane i kada postoje globalne promenljive?
27. Gde su definisane i kada postoje lokalne promenljive?
28. Šta se smešta na stek?
29. Kako se rukuje stekom?
30. Šta sadrži frejm?

4. MEMORIJA I PROCESOR RAČUNARA KONCEPT

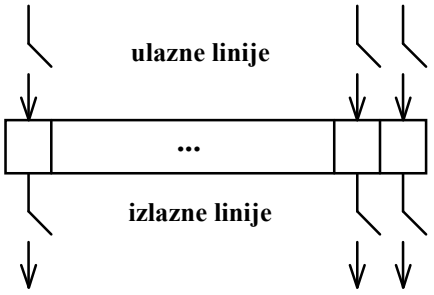
4.1. ORGANIZACIJA MEMORIJE RAČUNARA KONCEPT

Memorijska lokacija ili reč obuhvata 16 bita ili 2 bajta od po 8 bita i odražava pozicionu predstavu binarnog broja. Znači, krajnje desni bit memorijske lokacije sadrži najmanje značajnu cifru, a njen krajnje levi bit sadrži najznačajniju cifru (Slika 4.1.1).



Slika 4.1.1 Raspored bita u memorijskoj lokaciji

Memorijskim lokacijama se pristupa: ili radi čitanja (preuzimanja) sadržaja, ili radi pisanja (izmene) sadržaja. Istovremeno se pristupa svim bitima memorijskih lokacija. Svi biti se paralelno prenose iz lokacije, odnosno u lokaciju (Slika 4.1.2).

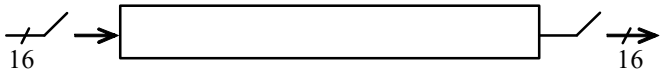


Slika 4.1.2 Principijelni izgled memorijske lokacije

Kada se istovremeno zatvore prekidači na ulaznim linijama, tada nivo signala u svakoj od ulaznih linija (0 ili 1) određuje novi sadržaj bita memorijske lokacije u koji linija ulazi. Slično, kada se istovremeno zatvore prekidači na izlaznim linijama, tada sadržaj bita memorijske lokacije (0 ili 1) određuje nivo signala u izlaznoj liniji koja izlazi iz tog bita. Ulazni prekidači omogućuju pisanje sadržaja memorijske lokacije, a izlazni prekidači omogućuju čitanje sadržaja memorijske lokacije. Zbog načina izvedbe memorijskih lokacija, njihov sadržaj nije definisan u toku pisanja. Zato su pisanje i čitanje sadržaja memorijske lokacije međusobno isključivi, što znači da se ulazni i izlazni prekidači iste memorijske lokacije istovremeno ne zatvaraju.

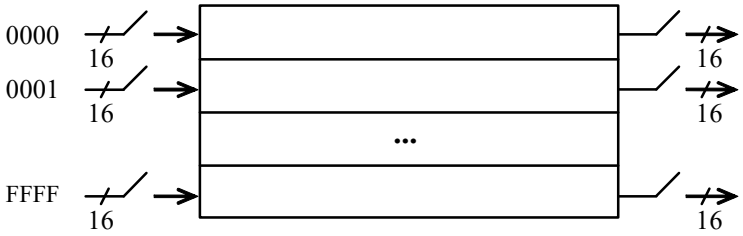
Svi ulazni prekidači jedne memorijske lokacije imaju isto upravljanje i iste argumente (jer su svi istovremeno otvoreni ili zatvoreni). Isto važi i za sve izlazne

prekidače pomenute memorijske lokacije. Zato se svi ulazni prekidači jedne memorijske lokacije predstavljaju samo jednim prekidačem, a svi izlazni prekidači pomenute memorijske lokacije drugim prekidačem (Slika 4.1.3).



Slika 4.1.3 Pojednostavljeni principijelni izgled memorijske lokacije

Pojednostavljeni principijelni prikaz memorije sadrži samo niz memorijskih lokacija od kojih se memorija sastoji. Memorijske lokacije se međusobno razlikuju po adresama, jer svaka memorijska lokacija ima jedinstvenu adresu. To je iskorišćeno u pojednostavljenom principijelnom prikazu memorije i svaka od prikazanih memorijskih lokacija je označena svojom adresom. Adrese su navedene u heksadecimalnom obliku ispred ulaznih linija prikazane memorijske lokacije (Slika 4.1.4).



Slika 4.1.4 Pojednostavljeni principijelni izgled memorije

DEKODIRANJE ADRESA

Jednoznačno upravljanje ulaznim i izlaznim prekidačima neke memorijske lokacije se može ostvariti, ako se osloni na njenu adresu (koja je jednoznačna) i na vrstu pristupanja ovoj memorijskoj lokaciji. Radi toga se pojedini biti `Ai (i = 0, ... , 15)` adrese memorijske lokacije koriste kao prekidački argumenti. Iz istog razloga kao prekidački argumenti se koriste i posebne logičke promenljive `č` (čitanje) i `P` (pisanje), koje određuju vrstu pristupanja memorijskoj lokaciji. Pri tome se podrazumeva da logička promenljiva `č` ima vrednost 1 samo za vreme čitanja, a da logička promenljiva `P` ima vrednost 1 samo za vreme pisanja i da u svakom momentu najviše jedna od ove dve logičke promenljive ima vrednost 1.

Uz prethodne pretpostavke, upravljanje ulaznim prekidačima memorijske lokacije sa adresom 0000_{16} opisuje funkcija:

$$P\&\sim A_{15}\&\sim A_{14}\&\sim A_{13}\&\sim A_{12}\&\sim A_{11}\&\sim A_{10}\&\sim A_9\&\sim A_8\&\sim A_7\&\sim A_6\&\sim A_5\&\sim A_4\&\sim A_3\&\sim A_2\&\sim A_1\&\sim A_0$$

a upravljanje izlaznim prekidačima memorijske lokacije sa adresom 0000_{16} opisuje funkcija:

$$Č\&\sim A_{15}\&\sim A_{14}\&\sim A_{13}\&\sim A_{12}\&\sim A_{11}\&\sim A_{10}\&\sim A_9\&\sim A_8\&\sim A_7\&\sim A_6\&\sim A_5\&\sim A_4\&\sim A_3\&\sim A_2\&\sim A_1\&\sim A_0$$

Slično, upravljanje ulaznim prekidačima memorijske lokacije sa adresom 0001_{16} opisuje funkcija:

$$P\&\sim A_{15}\&\sim A_{14}\&\sim A_{13}\&\sim A_{12}\&\sim A_{11}\&\sim A_{10}\&\sim A_9\&\sim A_8\&\sim A_7\&\sim A_6\&\sim A_5\&\sim A_4\&\sim A_3\&\sim A_2\&\sim A_1\&A_0$$

a upravljanje izlaznim prekidačima memorijske lokacije sa adresom 0001_{16} opisuje funkcija:

$$Č\&\sim A_{15}\&\sim A_{14}\&\sim A_{13}\&\sim A_{12}\&\sim A_{11}\&\sim A_{10}\&\sim A_9\&\sim A_8\&\sim A_7\&\sim A_6\&\sim A_5\&\sim A_4\&\sim A_3\&\sim A_2\&\sim A_1\&A_0$$

Na isti način, upravljanje ulaznim prekidačima memorijske lokacije sa adresom $FFFF_{16}$ opisuje funkcija:

$$P\&A_{15}\&A_{14}\&A_{13}\&A_{12}\&A_{11}\&A_{10}\&A_9\&A_8\&A_7\&A_6\&A_5\&A_4\&A_3\&A_2\&A_1\&A_0$$

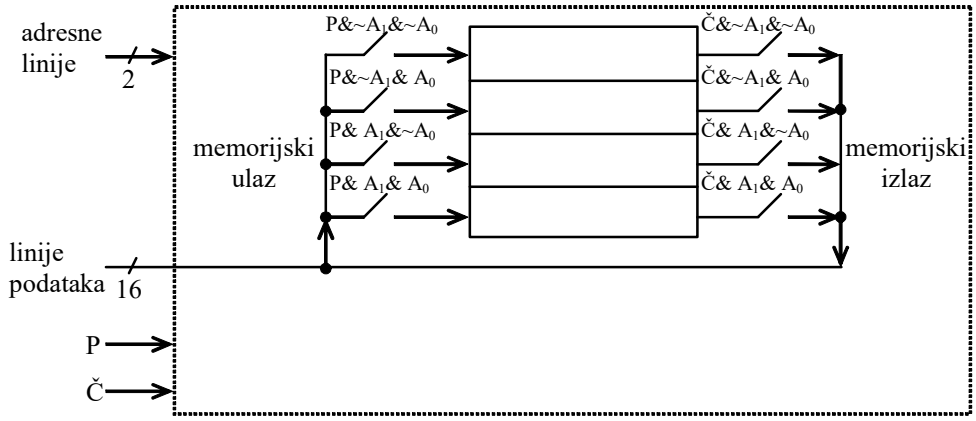
a upravljanje izlaznim prekidačima memorijske lokacije sa adresom $FFFF_{16}$ opisuje funkcija:

$$Č\&A_{15}\&A_{14}\&A_{13}\&A_{12}\&A_{11}\&A_{10}\&A_9\&A_8\&A_7\&A_6\&A_5\&A_4\&A_3\&A_2\&A_1\&A_0$$

Sve funkcije, koje opisuju upravljanje prekidačima memorijskih lokacija, su tako definisane, da je uvek najviše jedna od njih tačna. To znači da su u bilo kom trenutku zatvoreni samo ulazni, ili samo izlazni prekidači najviše jedne memorijske lokacije. Za prethodne funkcije se kaže da **dekodiraju** adresu memorijske lokacije. One se, zato, nazivaju **dekoderske funkcije**. Dekoderske funkcije omogućuju jednoznačno **selektovanje** memorijskih lokacija.

PRINCIPIJELNI IZGLED MEMORIJE SA 4 LOKACIJE

Za opisivanje funkcionisanja memorije podjednako su važni i memorijska adresa i sadržaji logičkih promenljivih `č` i `P`, jer memorijska adresa određuje, odnosno **adresira** memorijsku lokaciju kojoj se pristupa, a vrednosti logičkih promenljivih `č` i `P` određuju da li se toj memorijskoj lokaciji pristupa radi čitanja, ili radi pisanja. U slučaju čitanja, memorijski izlaz zavisi od zatečenog sadržaja adresirane memorijske lokacije, a u slučaju pisanja, memorijski ulaz određuje novi sadržaj adresirane memorijske lokacije. Prethodno navedeno je dovoljno za potpun principijelni prikaz memorije sa 4 lokacije (Slika 4.1.5).



Slika 4.1.5 Principijelni izgled memorije sa 4 lokacije

Memorijskoj adresi odgovaraju adresne linije. One prenose dvobitnu adresu, koja odgovara adresnom prostoru od 2^2, odnosno 4 lokacije. Memorijskom ulazu i izlazu odgovaraju linije podataka. Logičkim promenljivim P i Č odgovaraju upravljačke linije.

Svaka memorijska lokacija može da sadrži ceo broj, realan broj (u mašinskoj normalizovanoj formi), logičku konstantu ili dva znaka (ako kod znaka zauzima jedan bajt). U poslednjem slučaju, racionalno korišćenje memorijskih lokacija podrazumeva programsko smeštanje (pakovanje) znakova u bajte memorijske lokacije i njihovo programsko preuzimanje (raspakivanje) iz bajta memorijske lokacije. Sa stanovišta korisnika, praktičnije je rešenje u kome je moguće direktno pristupiti ne samo reči, nego i bajtu. Ono nije usvojeno za memoriju procesora KONCEPT, jer usložnjava njenu organizaciju.

4.2. KODIRANJE I MAŠINSKI FORMATI NAREDBI PROCESORA KONCEPT

Mašinski format svake naredbe sadrži njen kod i njene operande, odnosno podatke koji omogućuju pristup njenim operandima. Pošto sve naredbe nemaju istu vrstu i broj operanada, mašinski formati raznih naredbi se razlikuju, ne samo po kodovima naredbi koje sadrže, nego i po broju i vrsti operanada. Ako se naredbe razvrstaju na razne tipove naredbi po osnovu broja i vrste svojih operanada, tada svakom tipu naredbe odgovara jedan tip mašinskog formata naredbi.

Sve naredbe procesora KONCEPT se razvrstavaju, po osnovu broja i vrste svojih operanada, u 15 tipova. Prvom tipu pripadaju naredbe **SABERI**, **SABERI_P**, **ODUZMI**, **ODUZMI_P**, **I** i **ILI** sa jednim ulaznim i jednim ulazno-izlaznim registarskim operandom. Drugom tipu pripada naredba **UPOREDI** sa dva ulazna registarska operanda. Trećem tipu pripadaju naredbe **DODAJ_1**, **ODBIJ_1**, **NE**, **LEVO** i **DESNO** sa jednim ulazno-izlaznim registarskim operandom. Za sve naredbe prethodna tri tipa

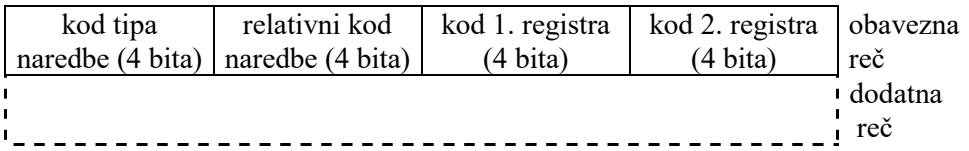
se podrazumeva korišćenje status registra, iako se on ne navodi kao operand. Četvrtom tipu pripada naredba `PREBACI_RR` sa jednim ulaznim i jednim izlaznim registarskim operandom. Petom tipu pripada pripada naredba `PREBACI_NR` sa jednim ulaznim neposrednim i jednim izlaznim registarskim operandom. Šestom tipu pripada naredba `PREBACI_DR` sa jednim ulaznim direktnim i jednim izlaznim registarskim operandom. Sedmom tipu pripada naredba `PREBACI_PR` sa jednim ulaznim posrednim i jednim izlaznim registarskim operandom. Osmom tipu pripada naredba `PREBACI_IR` sa jednim ulaznim indeksnim i jednim izlaznim registarskim operandom. Devetom tipu pripada naredba `PREBACI_RD` sa jednim ulaznim registarskim i jednim izlaznim direktnim operandom. Desetom tipu pripada naredba `PREBACI_RP` sa jednim ulaznim registarskim i jednim izlaznim posrednim operandom. Jedanaestom tipu pripada naredba `PREBACI_RI` sa jednim ulaznim registarskim i jednim izlaznim indeksnim operandom. Dvanaestom tipu pripada naredba `SKOČI` sa jednim operandom. Trinaestom tipu pripadaju naredbe uslovnog skoka, sa jednim operandom. Za ove naredbe se podrazumeva korišćenje status registra, iako se on ne navodi kao operand. Četrnaestom tipu pripada naredba `POZOVI`, sa jednim operandom. Za ovu naredbu se podrazumeva korišćenje registra `%15`, iako se on ne navodi kao operand. Petnaestom tipu pripada naredba `NATRAG`, bez operanada. Za ovu naredbe se podrazumeva korišćenje registra `%15`, iako se on ne navodi kao operand

Za kod svake naredbe je važno da omogući razlikovanje raznih naredbi istog tipa, ali i da omogući razlikovanje naredbi raznih tipova (odnosno razlikovanje raznih tipova njihovih mašinskih formata). Zato se kod naredbe sastoji od **koda tipa naredbe** (koji određuje tip mašinskog formata naredbe) i od **relativnog koda naredbe** (koji omogućuje razlikovanje naredbi istog tipa). Za kodiranje prethodnih petnaest tipova naredbi dovoljna je jedna heksadecimalna cifra. Takođe, jedna heksadecimalna cifra je dovoljna i za relativno kodiranje naredbi unutar svakog od tipova naredbi, jer ni u jednom od njih nema više od šesnaest naredbi. Obe heksadecimalne cifre koda naredbe zauzimaju jedan bajt (čiji značajniji biti sadrže kod tipa naredbe).

Kao što se razni tipovi naredbi međusobno razlikuju po svojim operandima, tako se i razni operandi razlikuju po svojim svojstvima. Postoji devet vrsta operanada, pet ulaznih i četiri izlazna, Šest vrsta operanada uključuje registar (registarski, posredni i indeksni operandi, svi i ulazni i izlazni), a pet vrsta operanda uključuje vrednost, odnosno adresu (neposredni operand, ali i direktni i indeksni operandi, oba i ulazni i izlazni). Zato neke operande karakteriše samo redni broj ili **kod registra** (registarski i posredni), neke samo vrednost (neposredni), neke samo adresa (direktni), a neke i kod registra i adresa (indeksni). Za kod registra je dovoljna jedna heksadecimalna cifra, jer ukupno ima šesnaest registara opšte namene. Za vrednost, odnosno adresu je dovoljna jedna 16 bitna reč, odnosno 4 heksadecimalne cifre.

Iz prethodno navedenog sledi da mašinski format naredbe procesora KONCEPT treba da obuhvati jedan kod naredbe, do dva koda registara i jednu vrednost, odnosno adresu. Iako za svaku naredbu nisu potrebna dva koda registara i jedna vrednost, odnosno adresa, radi pravilnosti, za njih je predviđeno mesto u mašinskom formatu naredbe. Zato mašinski format naredbe obuhvata do dve reči. Prva, je uvek prisutna, pa se naziva **obavezna reč**. Ona u značajnijem, prvom bajtu sadrži kod naredbe, a u manje značajnom, drugom bajtu kodove registara. Značajniji biti drugog bajta obavezne reči sadrže kod 1. registra (ulazno/izlazni ili ulazni registar), a manje značajni biti ovog bajta sadrže kod 2. registra (ulazni registar). Podrazumeva se da drugi bajt obavezne reči nije iskorišćen za one tipove naredbi koji koriste samo jedan ili nijedan registar. Po potrebi, mašinski format naredbe sadrži i drugu, **dodatnu reč**. U njoj se nalazi vrednost neposrednog operanda ili adresa potrebna za direktni i indeksni operand.

Slika 4.2.1 prikazuje izgled mašinskog formata naredbe.



Slika 4.2.1 Izgled mašinskog formata naredbe

Izbor koda tipa naredbe i relativnog koda naredbe je proizvoljan. Kao kod registra služi njegov redni broj (0: `%0`, 1: `%1`, 2: `%2`, 3: `%3`, ...).

KODOVI 1. TIPA NAREDBI

Naredbama prvog tipa odgovaraju kodovi:

10_{16} **SABERI**
11_{16} **SABERI_P**
12_{16} **ODUZMI**
13_{16} **ODUZMI_P**
14_{16} **I**
15_{16} **ILI**

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata samo obaveznu reč. Tako, za asemblersku naredbu:

```
SABERI          %3,%2
```

mašinski format sadrži samo obaveznu reč:

1023_{16}

KODOVI 2. TIPA NAREDBI

Jedinoj naredbi drugog tipa odgovara kod:

```
20₁₆    UPOREDI
```

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata samo obaveznu reč. Tako, za asemblersku naredbu:

```
        UPOREDI     %11,%10
```

mašinski format sadrži samo obaveznu reč:

20AB$_{16}$

KODOVI 3. TIPA NAREDBI

Naredbama trećeg tipa odgovaraju kodovi:

```
30₁₆    DODAJ_1
31₁₆    ODBIJ_1
32₁₆    NE
33₁₆    LEVO
34₁₆    DESNO
```

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata samo obaveznu reč. Tako, za asemblersku naredbu:

```
        NE          %5
```

mašinski format sadrži samo obaveznu reč:

325x$_{16}$

('**x**' označava neiskorišćene manje značajne bite drugog bajta obavezne reči).

KODOVI 4. TIPA NAREDBI

Jedinoj naredbi četvrtog tipa odgovara kod:

```
40₁₆    PREBACI_RR
```

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata samo obaveznu reč. Tako, za asemblersku naredbu:

```
PREBACI_RR      %5,%12
```

mašinski format sadrži samo obaveznu reč:

$40C5_{16}$

KODOVI 5. TIPA NAREDBI

Jedinoj naredbi petog tipa odgovara kod:

50_{16} **PREBACI_NR**

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata obaveznu i dodatnu reč. Tako, za asemblersku naredbu:

```
PREBACI_NR      $6,%12
```

mašinski format sadrži obaveznu i dodatnu reč:

$50Cx_{16}$
0006_{16}

('**x**' označava neiskorišćene manje značajne bite drugog bajta obavezne reči).

Slično, za asemblersku naredbu:

```
PREBACI_NR      $realan,%3
```

mašinski format sadrži obaveznu i dodatnu reč:

$503x_{16}$
$aaaa_{16}$

('**x**' označava neiskorišćene manje značajne bite drugog bajta obavezne reči, dok '**aaaa**' označava adresu labele '**realan**').

KODOVI 6. TIPA NAREDBI

Jedinoj naredbi šestog tipa odgovara kod:

60_{16} **PREBACI_DR**

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata obaveznu i dodatnu reč. Tako, za asemblersku naredbu:

```
PREBACI_DR     realan,%7
```

mašinski format sadrži obaveznu i dodatnu reč:

$607x_{16}$
$aaaa_{16}$

('x' označava neiskorišćene manje značajne bite drugog bajta obavezne reči, dok 'aaaa' označava adresu labele 'realan').

KODOVI 7. TIPA NAREDBI

Jedinoj naredbi sedmog tipa odgovara kod:

```
70₁₆    PREBACI_PR
```

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata samo obaveznu reč. Tako, za asemblersku naredbu:

```
PREBACI_PR     (%5),%12
```

mašinski format sadrži samo obaveznu reč:

$70C5_{16}$

KODOVI 8. TIPA NAREDBI

Jedinoj naredbi osmog tipa odgovara kod:

```
80₁₆    PREBACI_IR
```

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata obaveznu i dodatnu reč. Tako, za asemblersku naredbu:

```
PREBACI_IR     t(%2),%9
```

mašinski format sadrži obaveznu i dodatnu reč:

8092_{16}
$aaaa_{16}$

('aaaa' označava adresu labele 't').

KODOVI 9. TIPA NAREDBI

Jedinoj naredbi devetog tipa odgovara kod:

```
90₁₆    PREBACI_RD
```

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata obaveznu i dodatnu reč. Tako, za asemblersku naredbu:

```
PREBACI_RD     %8,realan
```

mašinski format sadrži obaveznu i dodatnu reč:

908x$_{16}$
aaaa$_{16}$

('**x**' označava neiskorišćene manje značajne bite drugog bajta obavezne reči, dok '**aaaa**' označava adresu labele '**realan**').

KODOVI 10. TIPA NAREDBI

Jedinoj naredbi desetog tipa odgovara kod:

A0$_{16}$ **PREBACI_RP**

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata samo obaveznu reč. Tako, za asemblersku naredbu:

```
PREBACI_RP     %7,(%4)
```

mašinski format sadrži samo obaveznu reč:

A047$_{16}$

KODOVI 11. TIPA NAREDBI

Jedinoj naredbi jedanaestog tipa odgovara kod:

B0$_{16}$ **PREBACI_RI**

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata obaveznu i dodatnu reč. Tako, za asemblersku naredbu:

```
PREBACI_RI     %9,t(%3)
```

mašinski format sadrži obaveznu i dodatnu reč:

B039$_{16}$
aaaa$_{16}$

('**aaaa**' označava adresu labele '**t**').

KODOVI 12. TIPA NAREDBI

Jedinoj naredbi dvanaestog tipa odgovara kod:

$C0_{16}$	**SKOČI**

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata obaveznu i dodatnu reč. Tako, za asemblersku naredbu:

```
SKOČI   kraj
```

mašinski format sadrži obaveznu i dodatnu reč:

$C0xx_{16}$
$aaaa_{16}$

('**xx**' označava neiskorišćen drugi bajt obavezne reči, dok '**aaaa**' označava adresu labele '**kraj**').

KODOVI 13. TIPA NAREDBI

Naredbama trinaestog tipa odgovaraju kodovi:

$D0_{16}$	**SKOČI_ZA_== / SKOČI_ZA_N**
$D1_{16}$	**SKOČI_ZA_!= / SKOČI_ZA_NE_N**
$D2_{16}$	**SKOČI_ZA_< / SKOČI_ZA_P**
$D3_{16}$	**SKOČI_ZA_>= / SKOČI_ZA_NE_P**
$D4_{16}$	**SKOČI_ZA_>**
$D5_{16}$	**SKOČI_ZA_<=**
$D6_{16}$	**SKOČI_ZA_±_<**
$D7_{16}$	**SKOČI_ZA_±_>=**
$D8_{16}$	**SKOČI_ZA_±_>**
$D9_{16}$	**SKOČI_ZA_±_<=**
DA_{16}	**SKOČI_ZA_M**
DB_{16}	**SKOČI_ZA_NE_M**
DC_{16}	**SKOČI_ZA_V**
DD_{16}	**SKOČI_ZA_NE_V**

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata obaveznu i dodatnu reč. Tako, za asemblersku naredbu:

```
SKOČI_ZA_==    kraj
```

mašinski format sadrži obaveznu i dodatnu reč:

$D0xx_{16}$
$aaaa_{16}$

('**xx**' označava neiskorišćen drugi bajt obavezne reči, dok '**aaaa**' označava adresu labele '**kraj**').

KODOVI 14. TIPA NAREDBI

Jedinoj naredbi četrnaestog tipa odgovara kod:

$E0_{16}$ POZOVI

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata obaveznu i dodatnu reč. Tako, za asemblersku naredbu:

```
POZOVI nzd
```

mašinski format sadrži obaveznu i dodatnu reč:

$E0Fx_{16}$
$aaaa_{16}$

('**Fx**' označava registar **%15**, čije korišćenje se podrazumeva, i da je neiskorišćen drugi bajt obavezne reči, dok '**aaaa**' označava adresu labele '**nzd**').

KODOVI 15 TIPA NAREDBI

Jedinoj naredbi petnaestog tipa odgovara kod:

$F0_{16}$ NATRAG

(prva heksadecimalna cifra određuje kod tipa naredbe). Mašinski format ovog tipa naredbi obuhvata samo obaveznu reč. Tako, za asemblersku naredbu:

```
NATRAG
```

mašinski format sadrži samo obaveznu reč:

$F0Fx_{16}$

('**Fx**' označava registar **%15**, čije korišćenje se podrazumeva, i da je neiskorišćen drugi bajt obavezne reči).

4.3. ORGANIZACIJA PROCESORA KONCEPT

Mašinske naredbe se čuvaju u memorijskim lokacijama sa uzastopnim rastućim adresama. To znači, na primer, da se dodatna reč naredbe nalazi uvek u memorijskoj lokaciji koja ima za 1 veću adresu od adrese memorijske lokacije sa obaveznom rečju iste naredbe. Takođe, memorijska lokacija sa obaveznom rečju naredbe ima za 1 veću adresu od adrese memorijske lokacije sa poslednjom rečju njoj prethodeće naredbe. Tako, na primer, sekvenci asemblerskih naredbi:

```
PREBACI_NR      $3,%1
PREBACI_RR      %1,%2
```

odgovaraju mašinske naredbe:

```
501x16  (u lokaciji sa adresom n)
000316  (u lokaciji sa adresom n+1)
402116  (u lokaciji sa adresom n+2)
```

Prva mašinska naredba se sastoji od obavezne reči na adresi n i od dodatne reči na adresi n+1, a druga mašinska naredba se sastoji od obavezne reči na adresi n+2.

Obavljanju svake mašinske naredbe mora da prethodi njeno prebacivanje iz memorije u procesor. To je potrebno da bi procesor dobio iz obavezne reči naredbe kod naredbe i kodove registara (kada su potrebni), a iz dodatne reči operand (kada je naveden u dodatnoj reči naredbe). Procesor može da pročita, odnosno **dobavi** obaveznu reč, ako poseduje njenu adresu. U toku čitanja obavezne reči procesor prvo upućuje njenu adresu po adresnim linijama ka memoriji. Istovremeno on upućuje ka memoriji i signal čitanja po odgovarajućoj upravljačkoj liniji. Zatim procesor preuzima sa linije podataka obaveznu reč. Za njeno čuvanje namenjen je poseban **registar naredbe**.

Kod naredbe određuje operaciju koju procesor treba da obavi i lokacije operanada. Tako, za neposredno adresiranje, operand se nalazi u dodatnoj reči. Za direktno adresiranje, u dodatnoj reči se nalazi adresa operanda, a za indeksno adresiranje, u dodatnoj reči je sastojak adrese operanda. U svakom od ova tri slučaja, da bi dobavio operand, procesor mora pročitati dodatnu reč. Pročitani operand, odnosno njegova adresa ili sastojak njegove adrese se čuva u posebnom **pomoćnom registru**.

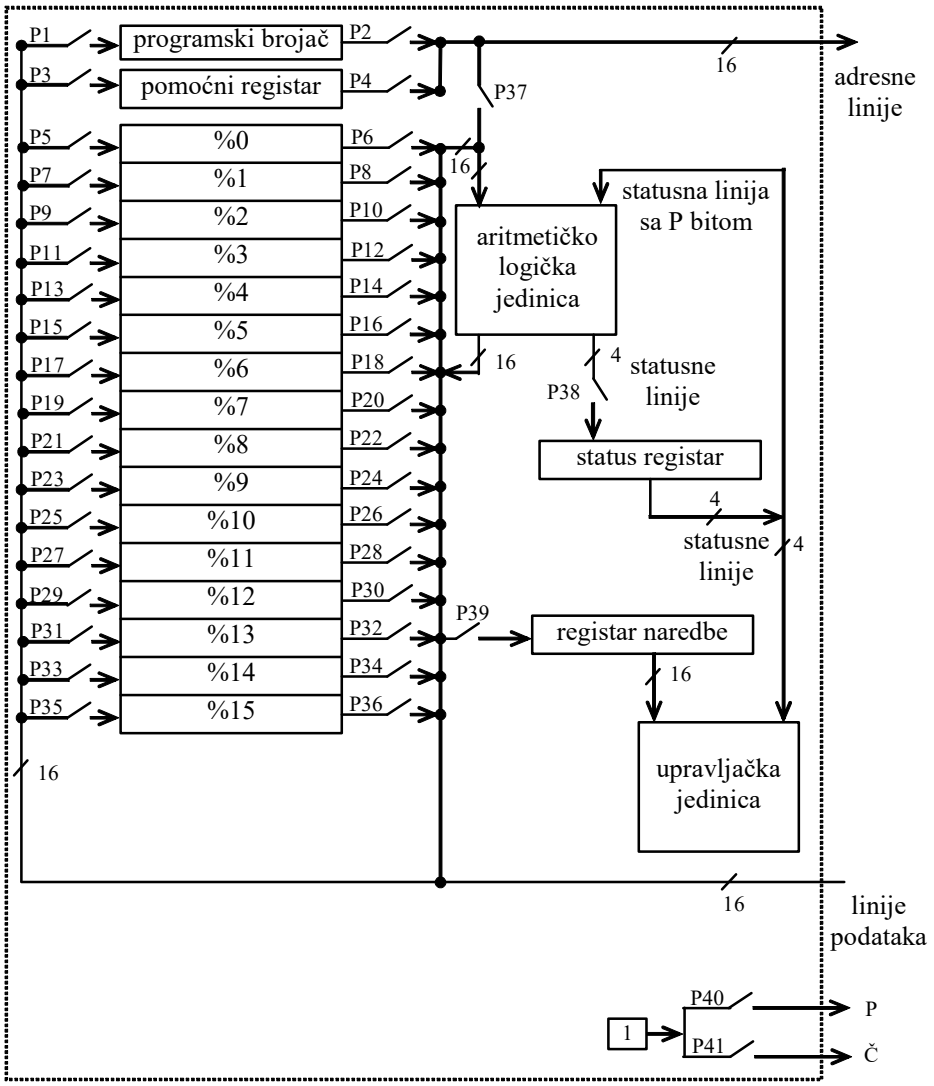
Čitanju dodatne reči prethodi određivanje njene adrese. To se postiže uvećavanjem za 1 adrese prethodno pročitane obavezne reči. Za čuvanje adrese dobavljane reči namenjen je poseban registar, nazvan **programski brojač** (*program counter*), jer uvećavanje njegovog sadržaja za 1 odgovara brojanju dobavljanih reči programa.

Procesor može da pristupi obavljanju operacije tek kada raspolaže kodom naredbe i njenim operandima. Za obavljanje operacija je zadužena **aritmetičko-logička jedinica**.

Delove procesora (na primer, njegove registre) spajaju **vezne linije**. Na njima se nalaze prekidači, čijim zatvaranjem se uspostavlja veza između pojedinih delova procesora. Za upravljanje stanjima prekidača iz veznih linija zadužena je **upravljačka jedinica**.

Registar naredbe, pomoćni registar i programski brojač spadaju u **registre posebne namene**. U ove registre spadaju i status registar, kao i registri iz aritmetičko-logičke i upravljačke jedinice.

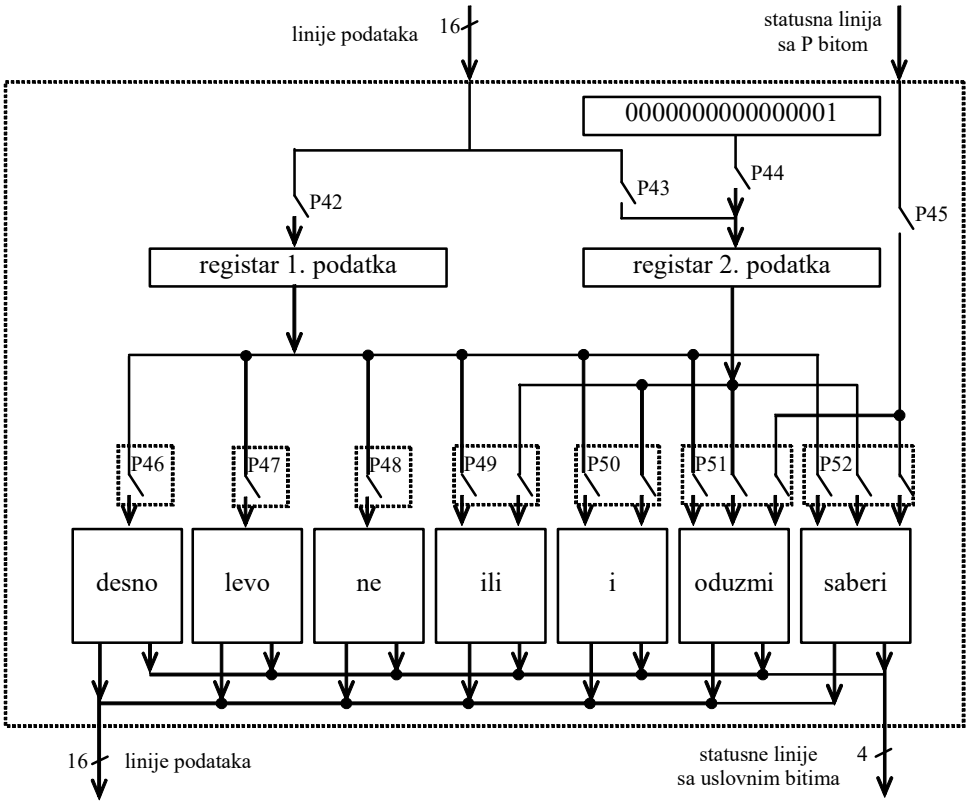
Slika 4.3.1 sadrži prikaz organizacije procesora KONCEPT (prekidač P37 razdvaja adresne linije od linija podataka).



Slika 4.3.1 Organizacija procesora KONCEPT

Obavljanje operacija se svodi na propuštanje podataka, koji odgovaraju pojedinim operandima, kroz odabrane sklopove aritmetičko-logičke jedinice. Na taj način se ovi podaci transformišu u rezultat operacije. Pre transformisanja podaci se

čuvaju u posebnim **registrima 1.** i **2. podatka**. Konstanta 1, kojom se uvećava sadržaj programskog brojača, se čuva u posebnom **registru konstante**. Slika 4.3.2 sadrži prikaz organizacije aritmetičko-logičke jedinice.



Slika 4.3.2 Organizacija aritmetičko-logičke jedinice

Sklopovi **saberi**, **oduzmi**, **i**, **ili**, **ne**, **levo** i **desno** aritmetičko-logičke jedinice ostvaruju logičke funkcije pojedinih naredbi procesora KONCEPT. U aritmetičko-logičku jedinicu ulazi sadržaj P bita status registra radi obavljanja aritmetičkih operacija u dvostrukoj preciznosti, a iz njega izlaze novi sadržaji uslovnih bita (N, P, M i V).

Aktivnost procesora je periodična. U svakom periodu procesor izvrši jednu mašinsku naredbu. Njeno izvršavanje obuhvata dve faze: **fazu dobavljanja** i **fazu obavljanja** naredbe. U fazi dobavljanja se dobavlja obavezna reč mašinske naredbe i smešta u registar naredbe. Pri tome se koristi i uvećava sadržaj programskog brojača. U fazi obavljanja se obavlja naredba čija obavezna reč je dobavljena u fazi dobavljanja. U fazi obavljanja se dobavlja i dodatna reč njenog mašinskog formata, ako postoji. Dobavljanje dodatne reči se ne razlikuje od dobavljanja obavezne reči.

4.4. UPRAVLJANJE PROCESOROM KONCEPT

Upravljanje procesorom KONCEPT se svodi na upravljanje njegovim prekidačima s ciljem njihovog dovođenja u zadana stanja. Na primer, zatvaranje pojedinih prekidača uspostavlja fizički put između izlaza jednog i ulaza drugog registra i omogućuje prenos sadržaja iz prvog u drugi registar. Podrazumeva se da pomenuti prekidači ostaju zatvoreni u toku obavljanja ovakve **elementarne radnje** procesora. Pošto se u toku aktivnosti procesora ponavljaju obavljanja elementarnih radnji poput prethodno pomenute, upravljanje procesorom ima cikličan karakter. U svakom ciklusu ono prevodi prekidače u stanja koja omogućuju obavljanje jedne ili više međusobno nezavisnih elementarnih radnji i zadrži ih u tim stanjima dok se date elementarne radnje ne obave. Pretpostavka je da svi ciklusi imaju isto trajanje.

Na nivou ciklusa je moguće precizno opisati i fazu dobavljanja i fazu obavljanja. Faza dobavljanja se sastoji od tri ciklusa:

1. ciklus: programski brojač → adresne linije (P2)
   1 → č (P41)
   linije podataka → registar naredbe (P39)
2. ciklus: programski brojač → registar 1. podatka (P2, P37, P42)
   1 → registar 2. podatka (P44)
3. ciklus: saberi (P52)
   linije podataka → programski brojač (P1)

Svaki red u opisu ciklusa je posvećen posebnoj elementarnoj radnji procesora. Elementarne radnje iz istog ciklusa se obavljaju istovremeno, a elementarne radnje iz raznih ciklusa se obavljaju sekvencijalno. Za elementarne radnje, namenjene prenosu sadržaja, strelica označava smer u kome se sadržaj prenosi. Na primer, u prvoj elementarnoj radnji 1. ciklusa, sadržaj programskog brojača se prenosi na adresne linije. Sadržaj, koji se prenosi na adresne i druge linije procesora, ostaje prisutan na njima samo u toku trajanja ciklusa. Elementarne radnje omogućuju ne samo prenos sadržaja na raznim relacijama, nego i aktiviranje pojedinih sklopova aritmetičko-logičke jedinice. To je, na primer, slučaj sa prvom elementarnom radnjom 3. ciklusa. Oznake prekidača, čija zatvaranja omogućuju obavljanje neke elementarne radnje, se navode između malih zagrada za tu elementarnu radnju.

U 1. od prethodna 3 ciklusa, sadržaj obavezne reči mašinske naredbe stigne u registar naredbe. Adresu memorijske lokacije sa obaveznom rečju sadrži programski brojač. U 2. ciklusu zatečeni sadržaj programskog brojača stigne u registar 1. podatka, a istovremeno konstanta 1 stigne u registar 2. podatka. Tom konstantom se uveća zatečeni sadržaj programskog brojača u 3. ciklusu, tako da nakon toga programski brojač sadrži adresu naredne memorijske lokacije.

U svakom od ciklusa, koje procesor posveti jednoj naredbi, obavi se elementarni deo dotične naredbe. Zato se može reći da u toku ciklusa procesor

obavi jednu **mikro-naredbu** ukupne naredbe. Više mikro-naredbi obrazuje **mikro-program**. Tako fazi dobavljanja odgovara **mikro-program dobavljanja**, dok fazi obavljanja svakog tipa naredbe odgovara poseban **mikro-program obavljanja**.

MIKRO-PROGRAMI OBAVLJANJA

Mikro-program obavljanja 1. tipa naredbi (`SABERI`, `SABERI_P`, `ODUZMI`, `ODUZMI_P`, `I` i `ILI`) izgleda:

1. ciklus: $\%r_1 \rightarrow$ registar 1. podatka (P_{2r_1+6}, P42)
2. ciklus: $\%r_2 \rightarrow$ registar 2. podatka (P_{2r_2+6}, P43)
3. ciklus: saberi (P52)/(P52, P45)/ oduzmi (P51)/(P51, P45)/ i (P50)/ ili (P49)
   linije podataka $\rightarrow$ $\%r_1$ (P_{2r_1+5})
   statusne linije $\rightarrow$ status registar (P38)

U 1. ciklusu sadržaj registra $\%r_1$, određenog prvim registarskim operandom, stigne u registar 1. podatka. U 2. ciklusu sadržaj registra $\%r_2$, određenog drugim registarskim operandom, stigne u registar 2. podatka. Do obavljanja operacije (određene kodom naredbe) dolazi u 3. ciklusu (u opisu ovog ciklusa, alternativne mikro-naredbe su razdvojene znakom /). Za vreme 3. ciklusa rezultat operacije stigne u registar $\%r_1$, koga određuje prvi registarski operand, a nove vrednosti uslovnih bita stignu u najmanje značajna 4 bita status registra.

Mikro-program obavljanja 2. tipa naredbi (`UPOREDI`) izgleda:

1. ciklus: $\%r_1 \rightarrow$ registar 1. podatka (P_{2r_1+6}, P42)
2. ciklus: $\%r_2 \rightarrow$ registar 2. podatka (P_{2r_2+6}, P43)
3. ciklus: oduzmi (P51)
   statusne linije $\rightarrow$ status registar (P38)

U 1. ciklusu sadržaj registra $\%r_1$, određenog prvim registarskim operandom, stigne u registar 1. podatka. U 2. ciklusu sadržaj registra $\%r_2$, određenog drugim registarskim operandom, stigne u registar 2. podatka. Radi njihovog poređenja, u 3. ciklusu se obavi operacija oduzimanja, a kao rezultat poređenja, nove vrednosti uslovnih bita stignu u najmanje značajna 4 bita status registra.

Mikro-program obavljanja 3. tipa naredbi (`DODAJ_1`, `ODBIJ_1`, `NE`, `LEVO` i `DESNO`) izgleda:

1. ciklus: $\%r_1 \rightarrow$ registar 1. podatka (P_{2r_1+6}, P42)
   $1 \rightarrow$ registar 2. podatka (P44)
2. ciklus: saberi (P52)/ oduzmi (P51)/ ne (P48)/ levo (P47)/ desno (P46)
   linije podataka $\rightarrow$ $\%r_1$ (P_{2r_1+5})
   statusne linije $\rightarrow$ status registar (P38)

U 1. ciklusu sadržaj registra r_1, određenog prvim registarskim operandom, stigne u registar 1. podatka, a istovremeno konstanta 1 stigne u registar 2. podatka. Do obavljanja operacije (određene kodom naredbe) dolazi u 2. ciklusu (u opisu ovog ciklusa, alternativne mikro-naredbe su razdvojene znakom /). Za vreme 2. ciklusa rezultat operacije stigne u registar r_1, koga određuje prvi registarski operand, a nove vrednosti uslovnih bita stignu u najmanje značajna 4 bita status registra.

Mikro-program obavljanja 4. tipa naredbi (**PREBACI_RR**) izgleda:

1. ciklus: r_2 → pomoćni registar (P$2r_2$+6, P3)
2. ciklus: pomoćni registar → r_1 (P4, P37,P$2r_1$+5)

U 1. ciklusu sadržaj registra r_2, određenog drugim registarskim operandom, stigne u pomoćni registar, a u 2. ciklusu ovaj sadržaj iz pomoćnog registra stigne u registar r_1, koga određuje prvi registarski operand.

Mikro-program obavljanja 5. tipa naredbi (**PREBACI_NR**) izgleda:

1. ciklus: programski brojač → adresne linije (P2)
   1 → č (P41)
   linije podataka → r_1 (P$2r_1$+5)
2. ciklus: programski brojač → registar 1. podatka (P2, P37, P42)
   1 → registar 2. podatka (P44)
3. ciklus: saberi (P52)
   linije podataka → programski brojač (P1)

U 1. ciklusu sadržaj dodatne reči mašinske naredbe, sa vrednošću drugog neposrednog operanda, stigne u registar r_1, koga određuje prvi registarski operand naredbe. Adresu memorijske lokacije sa dodatnom rečju sadrži programski brojač. U 2. ciklusu zatečeni sadržaj programskog brojača stigne u registar 1. podatka, a istovremeno konstanta 1 stigne u registar 2. podatka. Tom konstantom se uveća zatečeni sadržaj programskog brojača u 3. ciklusu, tako da nakon toga programski brojač sadrži adresu naredne memorijske lokacije.

Mikro-program obavljanja 6. tipa naredbi (**PREBACI_DR**) izgleda:

1. ciklus: programski brojač → adresne linije (P2)
   1 → č (P41)
   linije podataka → pomoćni registar (P3)
2. ciklus: programski brojač → registar 1. podatka (P2, P37, P42)
   1 → registar 2. podatka (P44)
3. ciklus: saberi (P52)
   linije podataka → programski brojač (P1)
4. ciklus: pomoćni registar → adresne linije (P4)
   1 → č (P41)
   linije podataka → %r_1 (P$2r_1$+5)

U 1. ciklusu sadržaj dodatne reči mašinske naredbe, sa adresom izvorišne lokacije (u kojoj se nalazi vrednost drugog direktnog operanda), stigne u pomoćni registar. Adresu memorijske lokacije sa dodatnom rečju sadrži programski brojač. U 2. ciklusu zatečeni sadržaj programskog brojača stigne u registar 1. podatka, a istovremeno konstanta 1 stigne u registar 2. podatka. Tom konstantom se uveća zatečeni sadržaj programskog brojača u 3. ciklusu, tako da nakon toga programski brojač sadrži adresu naredne memorijske lokacije. U 4. ciklusu iz izvorišne lokacije, čiju adresu sadrži pomoćni registar, stigne vrednost drugog direktnog operanda u registar %r_1, koga određuje prvi registarski operand naredbe.

Mikro-program obavljanja 7. tipa naredbi (**PREBACI_PR**) izgleda:

1. ciklus: %r_2 → pomoćni registar (P$2r_2$+6, P3)
2. ciklus: pomoćni registar → adresne linije (P4)
   1 → č (P41)
   linije podataka → %r_1 (P$2r_1$+5)

U 1. ciklusu iz registra %r_2, određenog drugim registarskim operandom naredbe, stigne adresa izvorišne lokacije (sa vrednošću drugog posrednog operanda) u pomoćni registar. U 2. ciklusu iz izvorišne lokacije, čiju adresu sadrži pomoćni registar, stigne vrednost drugog posrednog operanda u registar %r_1, koga određuje prvi registarski operand naredbe.

Mikro-program obavljanja 8. tipa naredbi (**PREBACI_IR**) izgleda:

1. ciklus: programski brojač → adresne linije (P2)
   1 → č (P41)
   linije podataka → pomoćni registar (P3)
2. ciklus: programski brojač → registar 1. podatka (P2, P37, P42)
   1 → registar 2. podatka (P44)
3. ciklus: saberi (P52)
   linije podataka → programski brojač (P1)
4. ciklus pomoćni registar → registar 1. podatka (P4, P37, P42)
5. ciklus $\%r_2$ → registar 2. podatka (P_{2r_2+6}, P43)
6. ciklus: saberi (P52)
   linije podataka → pomoćni registar (P3)
7. ciklus: pomoćni registar → adresne linije (P4)
   1 → č (P41)
   linije podataka → $\%r_1$ (P_{2r_1+5})

U 1. ciklusu sadržaj dodatne reči mašinske naredbe, sa prvim sastojkom adrese izvorišne lokacije (u kojoj se nalazi vrednost drugog indeksnog operanda), stigne u pomoćni registar. Adresu memorijske lokacije sa dodatnom rečju sadrži programski brojač. U 2. ciklusu zatečeni sadržaj programskog brojača stigne u registar 1. podatka, a istovremeno konstanta 1 stigne u registar 2. podatka. Tom konstantom se uveća zatečeni sadržaj programskog brojača u 3. ciklusu, tako da nakon toga programski brojač sadrži adresu naredne memorijske lokacije. U 4. ciklusu iz pomoćnog registra stigne prvi sastojak adrese izvorišne lokacije u registar 1. podatka, a u 5. ciklusu iz registra $\%r_2$, koga određuje drugi indeksni operand naredbe, stigne drugi sastojak adrese izvorišne lokacije u registar 2. podatka. U 6. ciklusu se saberu pomenuta dva sastojka adrese izvorišne lokacije i ova adresa stigne u pomoćni registar. U 7. ciklusu iz izvorišne lokacije, čiju adresu sadrži pomoćni registar, stigne vrednost drugog indeksnog operanda u registar $\%r_1$, koga određuje prvi registarski operand naredbe.

Mikro-program obavljanja 9. tipa naredbi (**PREBACI_RD**) izgleda:

1. ciklus: programski brojač → adresne linije (P2)
   1 → č (P41)
   linije podataka → pomoćni registar (P3)
2. ciklus: programski brojač → registar 1. podatka (P2, P37, P42)
   1 → registar 2. podatka (P44)
3. ciklus: saberi (P52)
   linije podataka → programski brojač (P1)
4. ciklus: pomoćni registar → adresne linije (P4)
   1 → p (P40)
   $\%r_1$ → linije podataka (P_{2r_1+6})

U 1. ciklusu sadržaj dodatne reči mašinske naredbe, sa adresom odredišne lokacije (koja odgovara prvom direktnom operandu), stigne u pomoćni registar. Adresu memorijske lokacije sa dodatnom rečju sadrži programski brojač. U 2. ciklusu zatečeni sadržaj programskog brojača stigne u registar 1. podatka, a istovremeno konstanta 1 stigne u registar 2. podatka. Tom konstantom se uveća zatečeni sadržaj programskog brojača u 3. ciklusu, tako da nakon toga programski brojač sadrži adresu naredne memorijske lokacije. U 4. ciklusu iz registra $\%r_1$, koga određuje prvi registarski operand naredbe, stigne vrednost u odredišnu lokaciju, čiju adresu ove lokacije sadrži pomoćni registar.

Mikro-program obavljanja 10. tipa naredbi (**PREBACI_RP**) izgleda:

1. ciklus: $\%r_1$ → pomoćni registar (P_{2r_1+6}, P3)
2. ciklus: pomoćni registar → adresne linije (P4)
   1 → **P** (P40)
   $\%r_2$ → linije podataka (P_{2r_2+6})

U 1. ciklusu iz registra $\%r_1$, određenog prvim posrednim operandom naredbe, stigne adresa odredišne lokacije (koja odgovara prvom posrednom operandu) u pomoćni registar. U 2. ciklusu iz registra $\%r_2$, koga određuje drugi registarski operand naredbe, stigne vrednost u odredišnu lokaciju, čiju adresu sadrži pomoćni registar.

Mikro-program obavljanja 11. tipa naredbi (**PREBACI_RI**) izgleda:

1. ciklus: programski brojač → adresne linije (P2)
   1 → **č** (P41)
   linije podataka → pomoćni registar (P3)
2. ciklus: programski brojač → registar 1. podatka (P2, P37, P42)
   1 → registar 2. podatka (P44)
3. ciklus: saberi (P52)
   linije podataka → programski brojač (P1)
4. ciklus pomoćni registar → registar 1. podatka (P4, P37, P42)
5. ciklus $\%r_1$ → registar 2. podatka (P_{2r_1+6}, P43)
6. ciklus: saberi (P52)
   linije podataka → pomoćni registar (P3)
7. ciklus: pomoćni registar → adresne linije (P4)
   1 → **P** (P40)
   $\%r_2$ → linije podataka (P_{2r_2+6})

U 1. ciklusu sadržaj dodatne reči mašinske naredbe, sa prvim sastojkom adrese odredišne lokacije (koja odgovara prvom indeksnom operandu), stigne u pomoćni registar. Adresu memorijske lokacije sa dodatnom rečju sadrži programski brojač.

U 2. ciklusu zatečeni sadržaj programskog brojača stigne u registar 1. podatka, a istovremeno konstanta 1 stigne u registar 2. podatka. Tom konstantom se uveća zatečeni sadržaj programskog brojača u 3. ciklusu, tako da nakon toga programski brojač sadrži adresu naredne memorijske lokacije. U 4. ciklusu iz pomoćnog registra stigne prvi sastojak adrese odredišne lokacije u registar 1. podatka, a u 5. ciklusu iz registra $\%r_1$, koga određuje prvi indeksni operand naredbe, stigne drugi sastojak adrese odredišne lokacije u registar 2. podatka. U 6. ciklusu se saberu pomenuta dva sastojka adrese odredišne lokacije i ova adresa stigne u pomoćni registar. U 7. ciklusu iz registra $\%r_2$, koga određuje drugi registarski operand naredbe, stigne vrednost prvog indeksnog operanda u odredišnu lokaciju, čiju adresu sadrži pomoćni registar.

Mikro-program obavljanja 12. tipa naredbi (`SKOČI`) izgleda:

1. ciklus: programski brojač → adresne linije (P2)
   1 → č (P41)
   linije podataka → pomoćni registar (P3)
2. ciklus: pomoćni registar → programski brojač (P4, P37, P1)

U 1. ciklusu sadržaj dodatne reči mašinske naredbe, sa adresom ciljne naredbe (od koje se nastavlja izvršavanje), stigne u pomoćni registar. Adresu memorijske lokacije sa dodatnom rečju sadrži programski brojač. U 2. ciklusu adresa ciljne naredbe iz pomoćnog registra stigne u programski brojač.

Mikro-program obavljanja 13. tipa naredbi (uslovni skokovi) izgleda:

1. ciklus: programski brojač → adresne linije (P2)
   1 → č (P41)
   linije podataka → pomoćni registar (P3)
2. ciklus: programski brojač → registar 1. podatka (P2, P37, P42)
   1 → registar 2. podatka (P44)
3. ciklus: saberi (P52)
   linije podataka → programski brojač (P1)
4. ciklus: ? pomoćni registar → programski brojač (P4, P37, P1)

U 1. ciklusu sadržaj dodatne reči mašinske naredbe, sa adresom ciljne naredbe (od koje se eventualno nastavlja izvršavanje), stigne u pomoćni registar. Adresu memorijske lokacije sa dodatnom rečju sadrži programski brojač. U 2. ciklusu zatečeni sadržaj programskog brojača stigne u registar 1. podatka, a istovremeno konstanta 1 stigne u registar 2. podatka. Tom konstantom se uveća zatečeni sadržaj programskog brojača u 3. ciklusu, tako da nakon toga programski brojač sadrži adresu naredne naredbe. Ako je ispunjen uslov naredbe uslovnog skoka, u 4. ciklusu adresa ciljne naredbe iz pomoćnog registra stigne u programski brojač.

Mikro-program obavljanja 14. tipa naredbi (`POZOVI`) izgleda:

1. ciklus: programski brojač → adresne linije (P2)
   1 → č (P41)
   linije podataka → pomoćni registar (P3)
2. ciklus: programski brojač → registar 1. podatka (P2, P37, P42)
   1 → registar 2. podatka (P44)
3. ciklus: saberi (P52)
   linije podataka → `%15` (P35)
4. ciklus: pomoćni registar → programski brojač (P4, P37, P1)

U 1. ciklusu sadržaj dodatne reči mašinske naredbe, sa adresom ciljne naredbe (od koje se nastavlja izvršavanje), stigne u pomoćni registar. Adresu memorijske lokacije sa dodatnom rečju sadrži programski brojač. U 2. ciklusu zatečeni sadržaj programskog brojača stigne u registar 1. podatka, a istovremeno konstanta 1 stigne u registar 2. podatka. Tom konstantom se uveća zatečeni sadržaj programskog brojača u 3. ciklusu, a tako dobijena povratna adresa stigne u registar `%15`. U 4. ciklusu adresa ciljne naredbe iz pomoćnog registra stigne u programski brojač.

Mikro-program obavljanja 15. tipa naredbi (`NATRAG`) izgleda:

1. ciklus: `%15` → programski brojač (P36, P1)

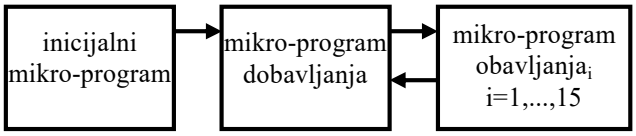
U 1. i jedinom ciklusu povratna adresa iz registra `%15` stigne u programski brojač.

INICIJALNI MIKRO-PROGRAM

Procesor može da započne izvršavanje nekog programa tek kada programski brojač sadrži adresu ulazne naredbe tog programa. Adresa ulazne naredbe, od koje počinje izvršavanje programa, se naziva **ulazna adresa** programa. Znači, da bi procesor započeo izvršavanje nekog programa, na početku rada procesora u programski brojač treba da dospe ulazna adresa tog programa. Pod pretpostavkom da nulta memorijska lokacija uvek sadrži ulaznu adresu programa, koga procesor treba da izvrši, početak rada procesora opisuje **inicijalni mikro-program**:

1. ciklus: 0000_{16} → adresne linije
   1 → č (P41)
   linije podataka → programski brojač (P1)

Inicijalni mikro-program puni ulaznu adresu u programski brojač. Slika 4.4.1 sadrži dijagram koji opisuje nastavak rada procesora.



Slika 4.4.1 Opis prelazaka između mikro-programa

Inicijalni mikro-program aktivira mikro-program dobavljanja, koji aktivira jedan od mikro-programa obavljanja, po čijem završetku se opet aktivira mikro-program dobavljanja.

4.5. UPRAVLJAČKA JEDINICA PROCESORA KONCEPT

Trajanje ciklusa, odnosno vreme obavljanja mikro-naredbi nije zanemarljivo, jer fizički signali putuju konačnom brzinom kroz fizički ekvivalent veznih linija procesora i kroz sklopove aritmetičko-logičke jedinice. Mikro-naredba je obavljena uspešno tek kada fizički signali pređu ceo put, predviđen za tu radnju. Ovakav put nastane zatvaranjem odgovarajućih prekidača. Zadatak upravljačke jedinice je da, pre obavljanja mikro-naredbi, dovede prekidače u potrebno stanje i da ih zadrži u tom stanju dok se pomenute mikro-naredbe ne obave. Upravljačka jedinica utiče na stanja prekidača posredstvom prekidačkih argumenata. Kako su svi prekidački argumenti logičke vrednosti, za smeštanje svakog od ovih argumenata dovoljan je po jedan bit posebnog **upravljačkog registra** (sa oznakom UR i sa oznakom UR_j za bit j). Na taj način od sadržaja upravljačkog registra zavisi stanje pojedinih prekidača. Prema tome, sadržaj upravljačkog registra određuje koja mikro-naredba se obavlja u datom ciklusu i ujedno predstavlja mašinski oblik obavljane mikro-naredbe. Obavljanju mikro-naredbe obavezno prethodi njeno dobavljanje, u toku koga se mašinski oblik mikro-naredbe smešta u upravljački registar. Iz prethodnog sledi da između ponašanja procesora na nivou mikro-naredbi i na nivou naredbi postoji jasna sličnost. Glavna razlika između ponašanja procesora na ova dva nivoa je u obimu posla koji se obavlja.

Dobavljanje mikro-naredbe je vezano za početni deo ciklusa, na primer, za **prvi poluciklus**, dok je obavljanje mikro-naredbe vezano za završni deo ciklusa, za **drugi poluciklus**. Razlikovanje poluciklusa dobavljanja od poluciklusa obavljanja omogućuje posebna logička promenljiva T (takt). Ona periodično menja vrednost, tako da uvek sadrži vrednost 1 u poluciklusu dobavljanja, a vrednost 0 u poluciklusu obavljanja. Fizička realizacija periodičnih izmena vrednosti ove logičke promenljive se oslanja na korišćenje elektronskih oscilatora, koji omogućuju stvaranje signala pravilnog perioda. Pravilna periodičnost izmene vrednosti logičke promenljive T uvodi pojam vremena, pa se izmene njene vrednosti mogu prikazati vremenskim dijagramom (Slika 4.5.1).

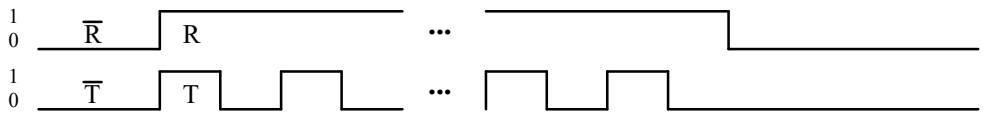
Slika 4.5.1 Vremenski dijagram periodične izmene vrednosti promenljive T

Za čuvanje mašinskih oblika mikro-naredbi potrebna je posebna **mikro-programska memorija**. Veličinu njenih lokacija određuje broj bita u mašinskim oblicima mikro-naredbi. Broj lokacija mikro-programske memorije zavisi od ukupnog broja mikro-naredbi u svim mikro-programima. U slučaju procesora KONCEPT, taj broj ne prelazi 128 mikro-naredbi, pa je za mikro-programsku memoriju dovoljno 128 lokacija. To istovremeno znači da je za adresu lokacije mikro-programske memorije potrebno 7 bita.

U toku dobavljanja mikro-naredbi, u upravljački registar se prebacuju sadržaji odgovarajućih lokacija mikro-programske memorije. Za izvršavanja mikro-programa je potrebno da se njihove mikro-naredbe dobavljaju i obavljaju u redosledu u kome su navedene u mikro-programima. To je moguće ostvariti bez posebnog programskog brojača, ako mašinski oblik svake mikro-naredbe u najmanje značajnih 7 bita sadrži adresu naredne mikro-naredbe, odnosno, adresu lokacije mikro-programske memorije sa mašinskim oblikom naredne mikro-naredbe. Ali, ova, **naredna adresa** ne sme biti u upravljačkom registru za vreme dobavljanja naredne mikro-naredbe, jer se tada smešta novi sadržaj u upravljački registar, pa njegov dotadašnji sadržaj, uključujući i narednu adresu, nije pristupačan. Zato, uz upravljački registar, postoji i **registar sekvence** (sa oznakom `RS` i sa oznakom $\texttt{RS}_j$ za bit `j`). Da bi naredna adresa bila raspoloživa u poluciklusu dobavljanja, ona se prebacuje iz upravljačkog registra u registar sekvence u poluciklusu obavljanja i to neposredno nakon poluciklusa dobavljanja, u toku koga je ona dospela u upravljački registar.

Procesor KONCEPT započinje svoju aktivnost izvršavanjem inicijalnog mikro-programa, ako se, na početku rada procesora, u registru sekvence nalazi adresa ulazne (i jedine) mikro-naredbe inicijalnog mikro-programa. Pod pretpostavkom da je ulazna adresa inicijalnog mikro-programa jednaka 0000000_2, ona će se nalaziti u registru sekvence na početka rada procesora, ako se pre početka rada procesora anulira sadržaj svih registara procesora, pa i registra sekvence.

Otkrivanje početka rada procesora KONCEPT omogućuje posebna logička promenljiva `R`, koja ima vrednost 0 kada procesor ne radi, a vrednost 1 dok god procesor radi. Izmene vrednosti ove logičke promenljive utiču na izmene vrednosti logičke promenljive `T`. Tako, logička promenljiva `T` ima vrednost 0 dok god istu vrednost ima i logička promenljiva `R`. Ove dve logičke promenljive istovremeno izmene vrednost sa 0 na 1. Vrednost logičke promenljive `T` se periodično menja dok god logička promenljiva `R` zadržava vrednost 1. Slika 4.5.2 sadrži vremenski dijagram izmena vrednosti logičkih promenljivih `R` i `T`.



Slika 4.5.2 Vremenski dijagram periodične izmene vrednosti promenljivih R i T

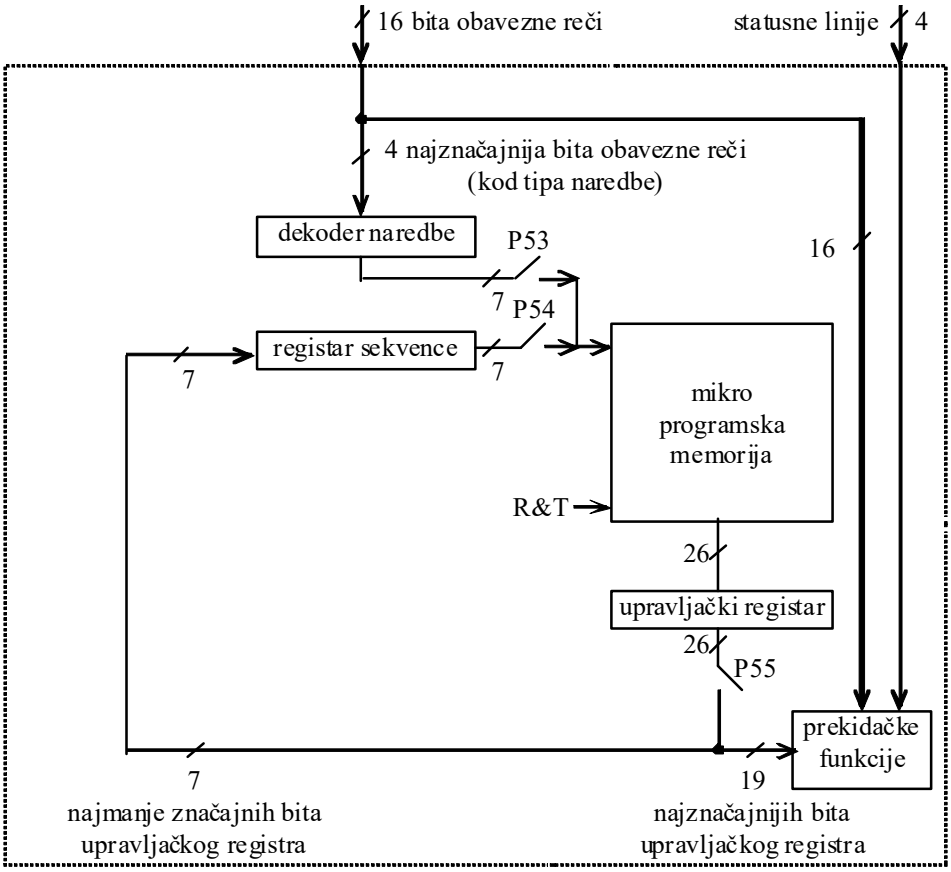
Jedina mikro-naredba inicijalnog mikro-programa kao narednu adresu sadrži ulaznu adresu mikro-programa dobavljanja. Naredna adresa poslednje mikro-naredbe mikro-programa dobavljanja ne može ukazati na mikro-program obavljanja, jer postoji više mikro-programa obavljanja, pa izbor jedne od njihovih ulaznih adresa zavisi od tipa naredbe. Zato se ulazna adresa mikro-programa obavljanja određuje iz koda tipa naredbe. Najjednostavnije rešenje je da kod tipa naredbe bude jednak ulaznoj (7 bitnoj) adresi odgovarajućeg mikro-programa. Ali ono nije prihvatljivo, jer su za kod tipa naredbe predviđena samo 4 bita u obaveznoj reči mašinskog oblika naredbe. Pošto svaki mikro-program ima najviše do 7 mikro-naredbi i ne zahteva više od 7 lokacija, mikro-programska memorija se može podeliti u segmente od po 8 lokacija, tako da je svaki od njih namenjen za jedan mikro-program. U mikro-programskoj memoriji sa 128 lokacija takvih segemenata ima 16 i to je dovoljno za smeštanje 15 mikro-programa obavljanja, mikro-programa dobavljanja i inicijalnog mikro-programa, ako se poslednja dva mikro-programa smeste u isti segment. Za adresiranje 16 segmenata su dovoljna 4 bita. Ako se u segment sa adresom 0000_2 smeste inicijalni mikro-program i mikro-program dobavljanja, tada za adresiranje ostalih segmenata mogu poslužiti 4 bita koda tipa naredbe. Za adresiranje lokacija u segmentu su potrebna još 3 bita. Pošto početna lokacija u svakom segmentu ima adresu 000_2, dodavanjem 3 nule iza 4 bita koda tipa naredbe, nastaje 7 bitna ulazna adresa odgovarajućeg mikro-programa obavljanja. Ovakvu transformaciju koda tipa naredbe u ulaznu adresu odgovarajućeg mikro-programa obavljanja pravi **dekoder naredbe**.

Poslednja mikro-naredba svakog od mikro-programa obavljanja kao narednu adresu koristi ulaznu adresu mikro-programa dobavljanja.

Sadržaji lokacija mikro-programske memorije su nepromenljivi (jer su određeni arhitekturom naredbi procesora KONCEPT). Zato je za njih dovoljno predvideti samo čitanje njihovih lokacija.

Stanje svakog prekidača procesora KONCEPT zavisi od vrednosti odgovarajuće prekidačke funkcije. Sve prekidačke funkcije implementira sklop upravljačke jedinice nazvan **prekidačke funkcije**.

Slika 4.5.3 prikazuje organizaciju upravljačke jedinice procesora KONCEPT.



Slika 4.5.3 Organizacija upravljačke jedinice

U toku rada procesora KONCEPT, naizmenično se ponavljaju ciklusi dobavljanja i obavljanja mikro-naredbi.

U upravljačku jedinicu ulazi snop sa 16 linija, koji polazi od registra naredbe (sa oznakom RN i sa oznakom RN_j za bit j). Njegove 4 najznačajnije linije prenose kod tipa naredbe do dekodera naredbe.

Adresa lokacije mikro-programske memorije dolazi:

1. ili iz dekodera naredbi,
2. ili iz registra sekvence.

Izbor jednog od ova 2 izvora adrese zavisi stanja prekidača P53 i P54.

U bite registra sekvence u poluciklusu obavljanja se prebacuje sadržaj 7 najmanje značajnih bita upravljačkog registra, kada je zatvoren prekidač P55.

Do čitanja lokacije mikro-programske memorije dolazi samo u poluciklusu dobavljanja, kada je tačna funkcija:

R&T

pa njena vrednost određuje stanje jedine upravljačke linije mikro-programske memorije, koja omogućuje čitanje njenih lokacije.

U ciklusu dobavljanja svake mikro-naredbe, upravljački registar se puni sadržajem odgovarajuće lokacije mikro-programske memorije.

4.6. UPRAVLJANJE PREKIDAČIMA IZ UPRAVLJAČKE JEDINICE PROCESORA KONCEPT

Svi prekidači (koji se nalaze i u upravljačkoj jedinici i van nje) se zatvaraju samo kada procesor radi, pa je zato logička promenljiva **R** obavezni argument svih prekidača. Isto važi i za logičku promenljivu **T**, jer ona određuje poluciklus u kome se prekidači zatvaraju. Logičke promenljive **R** i **T** predstavljaju **zajedničke argumente** prekidačkih funkcija svih prekidača. Od njih zavisi stanje prekidača P55, koji omogućuje periodična punjenja registra sekvence. Ova punjenja se dešavaju u poluciklusima obavljanja, pa upravljanje prekidačem P55 opisuje funkcija:

R&~T

Prekidači P53 i P54 određuju odakle stiže adresa u mikro-programsku memoriju. U svakom trenutku najviše jedan od ova dva prekidača sme da bude zatvoren. Prekidač P53 treba da bude zatvoren iza obavljanja mikro-programa dobavljanja, jer tada iz dekodera naredbe treba da stigne do mikro-programske memorije ulazna adresa mikro-programa obavljanja datog tipa naredbe. To je jedina situacija u kojoj prekidač P53 treba da bude zatvoren. Na ovu situaciju može da ukaže sadržaj registra sekvence, ako se na kraju mikro-programa dobavljanja u ovaj registar kao naredna adresa smesti ulazna adresa nekog mikro-programa obavljanja. Kao takva može da posluži bilo koja od ulaznih adresa mikro-programa obavljanja, jer one stižu u mikro-programsku memoriju samo iz dekodera naredbi (pa ne mogu na drugi način postati sadržaj registra sekvence). Na primer, može se odabrati ulazna adresa 0001000_2 mikro-programa obavljanja 1. tipa naredbi. Znači, kada ta adresa stigne u registar sekvence u jednom poluciklusu obavljanja neke mikro-naredbe, tada u sledećem poluciklusu dobavljanja naredne mikro-naredbe treba da bude zatvoren prekidač P53. Prisustvo ove ulazne adrese u registru sekvence detektuje logička funkcija **IZA_DOBAVLJANJA**:

~RS_6&~RS_5&~RS_4&RS_3&~RS_2&~RS_1&~RS_0

pa prekidačka funkcija prekidača P53 izgleda:

R&T&IZA_DOBAVLJANJA

Za prekidač P54 prekidačka funkcija izgleda:

R&T&~IZA_DOBAVLJANJA

jer prekidač P54 treba da bude zatvoren u svim poluciklusima dobavljanja mikro-naredbe, kada tome ne prethodi obavljanje poslednje mikro-naredbe mikro-programa dobavljanja.

Argumenti funkcije **IZA_DOBAVLJANJA** predstavljaju **posebne argumente** prekidačkih funkcija prekidača P53 i P54.

4.7. UPRAVLJANJE PREKIDAČIMA IZVAN UPRAVLJAČKE JEDINICE PROCESORA KONCEPT

Kao posebni argumenti prekidačkih funkcija prekidača izvan upravljačke jedinice procesora KONCEPT služe biti upravljačkog registra (izuzimajući narednu adresu iz najmanje značajnih sedam bita).

Stanje prekidača P1 zavisi od bita 7 upravljačkog registra, ako nije reč o uslovnom skoku. U slučaju uslovnih skokova, prekidač P1 treba da bude zatvoren u toku obavljanja 3. mikro-naredbe iz mikro-programa obavljanja 13. tipa naredbi, što omogućuje bit 25 upravljačkog registra. Međutim, u istom slučaju, u toku obavljanja 4. mikro-naredbe iz mikro-programa obavljanja 13. tipa naredbi, prekidač P1 sme da bude zatvoren samo ako je uslov ispunjen (čime se postiže uslovno obavljanje ove mikro-naredbe). Zato upravljanje ovim prekidačem opisuje funkcija:

R&~T&UR_7&(~USLOVNI_SKOK|USLOV_ISPUNJEN|UR_{25})

Proveru da li se izvršava uslovna upravljačka naredba omogućuju 4 najznačajnija bita registra naredbe RN_i **(i = 15, 14, 13, 12)** u kojima se nalazi kod tipa naredbe. Ovu proveru opisuje funkcija **USLOVNI_SKOK**:

RN_{15}&RN_{14}&~RN_{13}&RN_{12}

Prethodna funkcija ima vrednost 1 samo kada se u registru naredbe nalazi kod tipa naredbe uslovnih upravljačkih naredbi (1101_2). Ova funkcija tako dekodira uslovne upravljačke naredbe.

Proveru ispunjenosti uslova izvršavane uslovne upravljačke naredbe omogućuju logičke funkcije D_i **(i = 0, ... , 13)**. Svaka od njih je vezana za jednu uslovnu upravljačku naredbu i ima vrednost 1, ako se izvršava dotična naredba i ako je uslov te naredbe ispunjen. Pošto relativni kod uslovne upravljačke naredbe sadrže druga (po značaju) 4 bita registra naredbe, a ispunjenost uslova ove naredbe zavisi od sadržaja 4 najmanje značajna bita status registra, kao argumenti logičkih

funkcija D_i služe sadržaji pomenutih bita registra naredbe RN_i (i = 11, 10, 9, 8) i sadržaji uslovnih bita status registra:

```
D0:     ~RN11&~RN10&~RN9&~RN8&N                 (SKOČI_ZA_==)
D1:     ~RN11&~RN10&~RN9& RN8&~N                (SKOČI_ZA_!=)
D2:     ~RN11&~RN10& RN9&~RN8&P                 (SKOČI_ZA_<)
D3:     ~RN11&~RN10& RN9& RN8&~P                (SKOČI_ZA_>=)
D4:     ~RN11& RN10&~RN9&~RN8&(~P&~N)           (SKOČI_ZA_>)
D5:     ~RN11& RN10&~RN9& RN8&(P|N)             (SKOČI_ZA_<=)
D6:     ~RN11& RN10& RN9&~RN8&(M^V)             (SKOČI_ZA_±_<)
D7:     ~RN11& RN10& RN9& RN8&(~(M^V))          (SKOČI_ZA_±_>=)
D8:      RN11&~RN10&~RN9&~RN8&(~(M^V))&~N       (SKOČI_ZA_±_>)
D9:      RN11&~RN10&~RN9& RN8&((M^V)&N)         (SKOČI_ZA_±_<=)
D10:     RN11&~RN10& RN9&~RN8&M                 (SKOČI_ZA_M)
D11:     RN11&~RN10& RN9& RN8&~M                (SKOČI_ZA_NE_M)
D12:     RN11& RN10&~RN9&~RN8&V                 (SKOČI_ZA_V)
D13:     RN11& RN10&~RN9& RN8&~V                (SKOČI_ZA_NE_V)
```

Prethodne funkcije dekodiraju uslov i njegovu ispunjenost, pa proveru ispunjenosti uslova opisuje funkcija **USLOV_ISPUNJEN**:

$D_0|D_1|D_2|D_3|D_4|D_5|D_6|D_7|D_8|D_9|D_{10}|D_{11}|D_{12}|D_{13}$

Slika 4.7.1 sadrži pregled prekidačkih funkcija prekidača P2, P3 i P4.

prekidač	funkcija
P2	$R\&\sim T\&UR_8$
P3	$R\&\sim T\&UR_9$
P4	$R\&\sim T\&UR_{10}$

Slika 4.7.1 Upravljanje prekidačima P2, P3 i P4

Posebne argumente prekidača P2, P3 i P4 čuvaju biti 8, 9 i 10 upravljačkog registra.

Prekidači od P5 do P36 omogućuju pristup registrima opšte namene. Pošto na njihovo upravljanje utiče redni broj registra, kao posebni argumenti ovih prekidača se javljaju sadržaji 8 najmanje značajnih bita registra naredbe RN_i (i = 7, 6, ..., 0). U ovim bitima u 4 značajnije pozicije se nalazi kod prvog registra, a u 4 manje značajne pozicije kod drugog registra operanda. Za izbor jednog od ova dva koda potrebna su dva bita upravljačkog registra (bit 11 omogućuje izbor koda prvog registra, a bit 12 omogućuje izbor koda drugog registra). Dekodiranje jednog od 16 registara opisuju funkcije $REGISTAR_i$ (i = 0, ... , 15):

```
REGISTAR0:      (~RN7&~RN6&~RN5&~RN4&UR11)|(~RN3&~RN2&~RN1&~RN0&UR12)
REGISTAR1:      (~RN7&~RN6&~RN5& RN4&UR11)|(~RN3&~RN2&~RN1& RN0&UR12)
REGISTAR2:      (~RN7&~RN6& RN5&~RN4&UR11)|(~RN3&~RN2& RN1&~RN0&UR12)
REGISTAR3:      (~RN7&~RN6& RN5& RN4&UR11)|(~RN3&~RN2& RN1& RN0&UR12)
REGISTAR4:      (~RN7& RN6&~RN5&~RN4&UR11)|(~RN3& RN2&~RN1&~RN0&UR12)
REGISTAR5:      (~RN7& RN6&~RN5& RN4&UR11)|(~RN3& RN2&~RN1& RN0&UR12)
REGISTAR6:      (~RN7& RN6& RN5&~RN4&UR11)|(~RN3& RN2& RN1&~RN0&UR12)
REGISTAR7:      (~RN7& RN6& RN5& RN4&UR11)|(~RN3& RN2& RN1& RN0&UR12)
REGISTAR8:      ( RN7&~RN6&~RN5&~RN4&UR11)|( RN3&~RN2&~RN1&~RN0&UR12)
REGISTAR9:      ( RN7&~RN6&~RN5& RN4&UR11)|( RN3&~RN2&~RN1& RN0&UR12)
REGISTAR10:     ( RN7&~RN6& RN5&~RN4&UR11)|( RN3&~RN2& RN1&~RN0&UR12)
REGISTAR11:     ( RN7&~RN6& RN5& RN4&UR11)|( RN3&~RN2& RN1& RN0&UR12)
REGISTAR12:     ( RN7& RN6&~RN5&~RN4&UR11)|( RN3& RN2&~RN1&~RN0&UR12)
REGISTAR13:     ( RN7& RN6&~RN5& RN4&UR11)|( RN3& RN2&~RN1& RN0&UR12)
REGISTAR14:     ( RN7& RN6& RN5&~RN4&UR11)|( RN3& RN2& RN1&~RN0&UR12)
REGISTAR15:     ( RN7& RN6& RN5& RN4&UR11)|( RN3& RN2& RN1& RN0&UR12)
```

Nakon dekodiranja registra, potrebno je odrediti da li se zatvara njegov ulazni ili izlazni prekidač. To omogućuju dva bita upravljačkog registra (bit 13 omogućuje izbor ulaznog prekidača, a bit 14 omogućuje izbor izlaznog prekidača).

Slika 4.7.2 sadrži pregled funkcija koje opisuju upravljanje ulaznim prekidačima registara opšte namene.

prekidač	funkcija
P5	R&~T&REGISTAR0&UR13
P7	R&~T&REGISTAR1&UR13
P9	R&~T&REGISTAR2&UR13
P11	R&~T&REGISTAR3&UR13
P13	R&~T&REGISTAR4&UR13
P15	R&~T&REGISTAR5&UR13
P17	R&~T&REGISTAR6&UR13
P19	R&~T&REGISTAR7&UR13
P21	R&~T&REGISTAR8&UR13
P23	R&~T&REGISTAR9&UR13
P25	R&~T&REGISTAR10&UR13
P27	R&~T&REGISTAR11&UR13
P29	R&~T&REGISTAR12&UR13
P31	R&~T&REGISTAR13&UR13
P33	R&~T&REGISTAR14&UR13
P35	R&~T&REGISTAR15&UR13

Slika 4.7.2 Upravljanje ulaznim prekidačima registara opšte namene

Slika 4.7.3 sadrži pregled funkcija koje opisuju upravljanje izlaznim prekidačima registara opšte namene.

prekidač	funkcija
P6	R&~T&REGISTAR$_0$&UR$_{14}$
P8	R&~T&REGISTAR$_1$&UR$_{14}$
P10	R&~T&REGISTAR$_2$&UR$_{14}$
P12	R&~T&REGISTAR$_3$&UR$_{14}$
P14	R&~T&REGISTAR$_4$&UR$_{14}$
P16	R&~T&REGISTAR$_5$&UR$_{14}$
P18	R&~T&REGISTAR$_6$&UR$_{14}$
P20	R&~T&REGISTAR$_7$&UR$_{14}$
P22	R&~T&REGISTAR$_8$&UR$_{14}$
P24	R&~T&REGISTAR$_9$&UR$_{14}$
P26	R&~T&REGISTAR$_{10}$&UR$_{14}$
P28	R&~T&REGISTAR$_{11}$&UR$_{14}$
P30	R&~T&REGISTAR$_{12}$&UR$_{14}$
P32	R&~T&REGISTAR$_{13}$&UR$_{14}$
P34	R&~T&REGISTAR$_{14}$&UR$_{14}$
P36	R&~T&REGISTAR$_{15}$&UR$_{14}$

Slika 4.7.3 Upravljanje izlaznim prekidačima registara opšte namene

Slika 4.7.4 sadrži pregled funkcija koje opisuju upravljanje prekidačima od P37 do P44.

prekidač	funkcija
P37	R&~T&UR$_{15}$
P38	R&~T&UR$_{16}$
P39	R&~T&UR$_{17}$
P40	R&~T&UR$_{18}$
P41	R&~T&UR$_{19}$
P42	R&~T&UR$_{20}$
P43	R&~T&UR$_{21}$
P44	R&~T&UR$_{22}$

Slika 4.7.4 Upravljanje prekidačima od P37 do P44

Posebne argumente prekidača P37, P38, P39, P40, P41, P42, P43 i P44 čuvaju biti 15, 16, 17, 18, 19, 20, 21 i 22 upravljačkog registra.

Na upravljanje prekidačima od P45 do P52, koji omogućuju aktiviranje sklopova aritmetičko-logičke jedinice, utiču kodovi izvršavanih naredbi. Zato se kao njihovi posebni argumenti javljaju sadržaji najznačajnijih 8 bita registra naredbe `RNi (i = 15, 14, ... , 8)` sa kodom naredbe. Slika 4.7.5 sadrži prikaz opisa funkcija za dekodiranje naredbi.

SABERI	~RN_{15}&~RN_{14}&~RN_{13}& RN_{12}&~RN_{11}&~RN_{10}&~RN_9&~RN_8
SABERI_P	~RN_{15}&~RN_{14}&~RN_{13}& RN_{12}&~RN_{11}&~RN_{10}&~RN_9& RN_8
ODUZMI	~RN_{15}&~RN_{14}&~RN_{13}& RN_{12}&~RN_{11}&~RN_{10}& RN_9&~RN_8
ODUZMI_P	~RN_{15}&~RN_{14}&~RN_{13}& RN_{12}&~RN_{11}&~RN_{10}& RN_9& RN_8
I	~RN_{15}&~RN_{14}&~RN_{13}& RN_{12}&~RN_{11}& RN_{10}&~RN_9&~RN_8
ILI	~RN_{15}&~RN_{14}&~RN_{13}& RN_{12}&~RN_{11}& RN_{10}&~RN_9& RN_8
UPOREDI	~RN_{15}&~RN_{14}& RN_{13}&~RN_{12}&~RN_{11}&~RN_{10}&~RN_9&~RN_8
DODAJ_1	~RN_{15}&~RN_{14}& RN_{13}& RN_{12}&~RN_{11}&~RN_{10}&~RN_9&~RN_8
ODBIJ_1	~RN_{15}&~RN_{14}& RN_{13}& RN_{12}&~RN_{11}&~RN_{10}&~RN_9& RN_8
NE	~RN_{15}&~RN_{14}& RN_{13}& RN_{12}&~RN_{11}&~RN_{10}& RN_9&~RN_8
LEVO	~RN_{15}&~RN_{14}& RN_{13}& RN_{12}&~RN_{11}&~RN_{10}& RN_9& RN_8
DESNO	~RN_{15}&~RN_{14}& RN_{13}& RN_{12}&~RN_{11}& RN_{10}&~RN_9&~RN_8

Slika 4.7.5 Funkcije koje opisuju dekodiranje naredbi

Zatvaranje prekidača od P45 do P52 omogućuje bit 23 upravljačkog registra, a funkcije koje opisuju dekodiranje naredbi određuju koji od prekidača će tada biti zatvoren i koji sklop aktiviran. Prekidač P45 omogućuje da u aritmetičkim operacijama učestvuje bit prenosa P. Prekidači P52 zahtevaju dodatni poseban argument (bit 24 upravljčakog registra), da bi se sklop za sabiranje mogao koristiti za uvećanje programskog brojača, kao i za sabiranje sastojaka adrese indeksnog operanda.

Slika 4.7.6 sadrži pregled funkcija koje opisuju upravljanje prekidačima od P45 do P52.

prekidač	funkcija
P45	R&~T&(SABERI_P|ODUZMI_P)&UR_{23}
P46	R&~T&DESNO&UR_{23}
P47	R&~T&LEVO&UR_{23}
P48	R&~T&NE&UR_{23}
P49	R&~T&ILI&UR_{23}
P50	R&~T&I&UR_{23}
P51	R&~T&(ODUZMI|ODUZMI_P|UPOREDI|ODBIJ_1)&UR_{23}
P52	R&~T&(((SABERI|SABERI_P|DODAJ_1)&UR_{23})|UR_{24})

Slika 4.7.6 Upravljanje prekidačima od P45 do P52

Sve prethodno navedene prekidačke funkcije se implementiraju pomoću kombinacionih kola, pa se njihova tačnost istovremeno proverava i utiče na stanje prekidača u pojedinim poluciklusima.

4.8. MAŠINSKI OBLICI MIKRO-PROGRAMA

Početni sadržaj registra sekvence (0000000_2 - ulazna adresa inicijalnog mikro-programa) u poluciklusu dobavljanja izazove zatvaranje prekidača P54 i

punjenje jedine mikro-naredbe inicijalnog mikro-programa u upravljački registar. Po njenom izvršavanju, nastavlja se izvršavanje mikro-programa dobavljanja (sa ulaznom adresom 0000001_2).

Slika 4.8.1 sadrži mašinski oblik inicijalnog i mikro-programa dobavljanja (značajnijih 19 bita mašinskog oblika mikro-naredbe određuje njeno obavljanje, a preostalih 7 bita određuje dobavljanje sledeće mikro-naredbe).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
0000000	00000010000000000001 0000001	P41,P1	P54
0000001	00000010100000000010 0000010	P41,P39,P2	P54
0000010	00010100001000000010 0000011	P44,P42,P37,P2	P54
0000011	01000000000000000001 0001000	P52,P1	P53
0000100	00000000000000000000 0000000		
0000101	00000000000000000000 0000000		
0000110	00000000000000000000 0000000		
0000111	00000000000000000000 0000000		

Slika 4.8.1 Mašinski oblik inicijalnog i mikro-programa dobavljanja.

Slika 4.8.2 sadrži mašinski oblik mikro-programa obavljanja 1. tipa naredbi (P_{ir1} označava izlazni, a P_{ur1} ulazni prekidač prvog registra, dok P_{ir2} označava izlazni prekidač drugog registra,).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
0001000	00000100000010010000 0001001	P42,P_{ir1}	P54
0001001	00001000000010100000 0001010	P43,P_{ir2}	P54
0001010	00100000001001010000 0000001	P52/P51/P50/P49/P45,P38,P_{ur1}	P54
0001011	00000000000000000000 0000000		
0001100	00000000000000000000 0000000		
0001101	00000000000000000000 0000000		
0001110	00000000000000000000 0000000		
0001111	00000000000000000000 0000000		

Slika 4.8.2 Mašinski oblik mikro-programa obavljanja 1. tipa naredbi

Slika 4.8.3 sadrži mašinski oblik mikro-programa obavljanja 2. tipa naredbi (P_{ir1} označava izlazni prekidač prvog, a P_{ir2} izlazni prekidač drugog registra).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
0010000	0000010000010010000 0010001	P42,P_{ir1}	P54
0010001	0000100000010100000 0010010	P43,P_{ir2}	P54
0010010	0010000001000000000 0000001	P51,P38	P54
0010011	0000000000000000000 0000000		
0010100	0000000000000000000 0000000		
0010101	0000000000000000000 0000000		
0010110	0000000000000000000 0000000		
0010111	0000000000000000000 0000000		

Slika 4.8.3 Mašinski oblik mikro-programa obavljanja 2. tipa naredbi

Slika 4.8.4 sadrži mašinski oblik mikro-programa obavljanja 3. tipa naredbi (P_{ir1} označava izlazni, a P_{ur1} označava ulazni prekidač prvog registra).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
0011000	0001010000010010000 0011001	P44,P42,P_{ir1}	P54
0011001	0010000001001010000 0000001	P52/P51/P48/P47/P46,P38,P_{ur1}	P54
0011010	0000000000000000000 0000000		
0011011	0000000000000000000 0000000		
0011100	0000000000000000000 0000000		
0011101	0000000000000000000 0000000		
0011110	0000000000000000000 0000000		
0011111	0000000000000000000 0000000		

Slika 4.8.4 Mašinski oblik mikro-programa obavljanja 3. tipa naredbi

Slika 4.8.5 sadrži mašinski oblik mikro-programa obavljanja 4. tipa naredbi (P_{ir2} označava izlazni prekidač drugog, a P_{ur1} ulazni prekidač prvog registra).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
0100000	0000000000010100100 0100001	P_{ir2},P3	P54
0100001	0000000000101011000 0000001	P37,P_{ur1},P4	P54
0100010	0000000000000000000 0000000		
0100011	0000000000000000000 0000000		
0100100	0000000000000000000 0000000		
0100101	0000000000000000000 0000000		
0100110	0000000000000000000 0000000		
0100111	0000000000000000000 0000000		

Slika 4.8.5 Mašinski oblik mikro-programa obavljanja 4. tipa naredbi

Slika 4.8.6 sadrži mašinski oblik mikro-programa obavljanja 5. tipa naredbi (P_{ur1} označava ulazni prekidač prvog registra).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
0101000	0000001000001010010 0101001	P41,P_{ur1},P2	P54
0101001	0001010000100000010 0101010	P44,P42,P37,P2	P54
0101010	0100000000000000001 0000001	P52,P1	P54
0101011	0000000000000000000 0000000		
0101100	0000000000000000000 0000000		
0101101	0000000000000000000 0000000		
0101110	0000000000000000000 0000000		
0101111	0000000000000000000 0000000		

Slika 4.8.6 Mašinski oblik mikro-programa obavljanja 5. tipa naredbi

Slika 4.8.7 sadrži mašinski oblik mikro-programa obavljanja 6. tipa naredbi (P_{ur1} označava ulazni prekidač prvog registra).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
0110000	0000001000000000110 0110001	P41,P3,P2	P54
0110001	0001010000100000010 0110010	P44,P42,P37,P2	P54
0110010	0100000000000000001 0110011	P52,P1	P54
0110011	0000001000001011000 0000001	P41,P_{ur1},P4	P54
0110100	0000000000000000000 0000000		
0110101	0000000000000000000 0000000		
0110110	0000000000000000000 0000000		
0110111	0000000000000000000 0000000		

Slika 4.8.7 Mašinski oblik mikro-programa obavljanja 6. tipa naredbi

Slika 4.8.8 sadrži mašinski oblik mikro-programa obavljanja 7. tipa naredbi (P_{ir2} označava izlazni prekidač drugog, a P_{ur1} ulazni prekidač prvog registra).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
0111000	0000000000010100100 0111001	P_{ir2},P3	P54
0111001	0000001000001011000 0000001	P41,P_{ur1},P4	P54
0111010	0000000000000000000 0000000		
0111011	0000000000000000000 0000000		
0111100	0000000000000000000 0000000		
0111101	0000000000000000000 0000000		
0111110	0000000000000000000 0000000		
0111111	0000000000000000000 0000000		

Slika 4.8.8 Mašinski oblik mikro-programa obavljanja 7. tipa naredbi

Slika 4.8.9 sadrži mašinski oblik mikro-programa obavljanja 8. tipa naredbi (P_{ir2} označava izlazni prekidač drugog, a P_{ur1} ulazni prekidač prvog registra).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
1000000	0000001000000000110 1000001	P41,P3,P2	P54
1000001	0001010000100000010 1000010	P44,P42,P37,P2	P54
1000010	0100000000000000001 1000011	P52,P1	P54
1000011	0000010000100001000 1000100	P42,P37,P4	P54
1000100	0000100000010100000 1000101	P43,P_{ir2}	P54
1000101	0100000000000000100 1000110	P52,P3	P54
1000110	0000001000001011000 0000001	P41,P_{ur1},P4	P54
1000111	0000000000000000000 0000000		

Slika 4.8.9 Mašinski oblik mikro-programa obavljanja 8. tipa naredbi

Slika 4.8.10 sadrži mašinski oblik mikro-programa obavljanja 9. tipa naredbi (P_{ir1} označava izlazni prekidač prvog registra).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
1001000	0000001000000000110 1001001	P41,P3,P2	P54
1001001	0001010000100000010 1001010	P44,P42,P37,P2	P54
1001010	0100000000000000001 1001011	P52,P1	P54
1001011	0000001000010011000 0000001	P40,P_{ir1},P4	P54
1001100	0000000000000000000 0000000		
1001101	0000000000000000000 0000000		
1001110	0000000000000000000 0000000		
1001111	0000000000000000000 0000000		

Slika 4.8.10 Mašinski oblik mikro-programa obavljanja 9. tipa naredbi

Slika 4.8.11 sadrži mašinski oblik mikro-programa obavljanja 10. tipa naredbi (P_{ir1} označava izlazni prekidač prvog, a P_{ir2} izlazni prekidač drugog registra).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije 2 1 0 5432109876543210987 6543210	značenje jediničnih bita sadržaja lokacije biti 25 -> 7	 biti 6 -> 0
1010000	0000000000010010100 1010001	P_{ir1},P3	P54
1010001	0000000100010101000 0000001	P40,P_{ir2},P4	P54
1010010	0000000000000000000 0000000		
1010011	0000000000000000000 0000000		
1010100	0000000000000000000 0000000		
1010101	0000000000000000000 0000000		
1010110	0000000000000000000 0000000		
1010111	0000000000000000000 0000000		

Slika 4.8.11 Mašinski oblik mikro-programa obavljanja 10. tipa naredbi

Slika 4.8.12 sadrži mašinski oblik mikro-programa obavljanja 11. tipa naredbi (P_{ir1} označava izlazni prekidač prvog, a P_{ir2} izlazni prekidač drugog registra).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije 2 1 0 5432109876543210987 6543210	značenje jediničnih bita sadržaja lokacije biti 25 -> 7	 biti 6 -> 0
1011000	0000001000000000110 1011001	P41,P3,P2	P54
1011001	0001010000100000010 1011010	P44,P42,P37,P2	P54
1011010	0100000000000000001 1011011	P52,P1	P54
1011011	0000010000100001000 1011100	P42,P37,P4	P54
1011100	0000100000010010000 1011101	P43,P_{ir1}	P54
1011101	0100000000000000100 1011110	P52,P3	P54
1011110	0000010000010101000 0000001	P40,P_{ir2},P4	P54
1011111	0000000000000000000 0000000		

Slika 4.8.12 Mašinski oblik mikro-programa obavljanja 10. tipa naredbi

Slika 4.8.13 sadrži mašinski oblik mikro-programa obavljanja 12. tipa naredbi.

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
1100000	0000001000000000110 1100001	P41,P3,P2	P54
1100001	0000000000100001001 0000001	P37,P4,P1	P54
1100010	0000000000000000000 0000000		
1100011	0000000000000000000 0000000		
1100100	0000000000000000000 0000000		
1100101	0000000000000000000 0000000		
1100110	0000000000000000000 0000000		
1100111	0000000000000000000 0000000		

Slika 4.8.13 Mašinski oblik mikro-programa obavljanja 12. tipa naredbi

Slika 4.8.14 sadrži mašinski oblik mikro-programa obavljanja 13. tipa naredbi.

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
1101000	0000001000000000110 1101001	P41,P3,P2	P54
1101001	0001010000100000010 1101010	P44,P42,P37,P2	P54
1101010	1100000000000000001 1101011	P52,P1	P54
1101011	0000000000100001001 0000001	P37,P4,P1	P54
1101100	0000000000000000000 0000000		
1101101	0000000000000000000 0000000		
1101110	0000000000000000000 0000000		
1101111	0000000000000000000 0000000		

Slika 4.8.14 Mašinski oblik mikro-programa obavljanja 13. tipa naredbi

Slika 4.8.15 sadrži mašinski oblik mikro-programa obavljanja 14. tipa naredbi (P_{ur1} označava ulazni prekidač prvog registra).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
1110000	0000001000000000110 1110001	P41,P3,P2	P54
1110001	0001010000100000010 1110010	P44,P42,P37,P2	P54
1110010	0100000000001010000 1110011	P52,P_{ur1}(P35)	P54
1110011	0000000000100001001 0000001	P37,P4,P1	P54
1110100	0000000000000000000 0000000		
1110101	0000000000000000000 0000000		
1110110	0000000000000000000 0000000		
1110111	0000000000000000000 0000000		

Slika 4.8.15 Mašinski oblik mikro-programa obavljanja 14. tipa naredbi

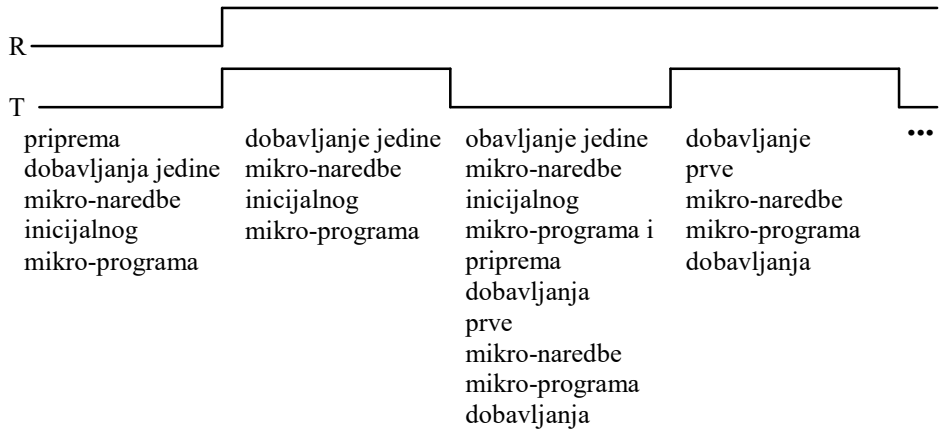
Slika 4.8.16 sadrži mašinski oblik mikro-programa obavljanja 15. tipa naredbi (P_{ir1} označava izlazni prekidač prvog registra).

lokacije mikro-programske memorije			
binarna adresa lokacije	binarni sadržaj lokacije	značenje jediničnih bita sadržaja lokacije	
	2 1 0 5432109876543210987 6543210	biti 25 -> 7	biti 6 -> 0
1111000	0000000000010010001 0000001	P_{ir1}(P36),P1	P54
1111001	0000000000000000000 0000000		
1111010	0000000000000000000 0000000		
1111011	0000000000000000000 0000000		
1111100	0000000000000000000 0000000		
1111101	0000000000000000000 0000000		
1111110	0000000000000000000 0000000		
1111111	0000000000000000000 0000000		

Slika 4.8.16 Mašinski oblik mikro-programa obavljanja 15. tipa naredbi

4.9. RAZMATRANJE RADA PROCESORA KONCEPT

Pre početka rada procesora KONCEPT anuliraju se svi registri procesora, pa i registar sekvence (Slika 4.9.1).



Slika 4.9.1 Vremenski dijagram rada procesora KONCEPT

Početni sadržaj registra sekvence dovodi do punjenja upravljačkog registra sadržajem prve lokacije mikro-programske memorije (sa adresom 0000000_2). Na taj način, u upravljački registar dospe mašinski oblik jedine mikro-naredbe inicijalnog mikro-programa, čije obavljanje dovodi do smeštanja sadržaja prve memorijske lokacije (sa adresom 0000_{16}) u programski brojač. Nakon toga, u upravljački registar se smesti sadržaj druge lokacije mikro-programske memorije (sa adresom 0000001_2). Time se za obavljanje spremi prva mikro-naredba mikro-programa dobavljanja. Prvo izvršavanje ovog mikro-programa dovodi do prebacivanja u registar naredbe sadržaja memorijske lokacije čiju adresu sadrži programski brojač. Neka ova memorijska lokacija sadrži, na primer, obaveznu reč naredbe:

```
PREBACI_NR      $1,%2
```

čija mašinska predstava zauzima dve uzastopne memorijske lokacije, jer obuhvata obaveznu i dodatnu reč: $502x_{16}$ 0001_{16}. Tada iza obavljanja poslednje mikro-naredbe mikro-programa dobavljanja sledi izvršavanje mikro-programa obavljanja naredbi 5. tipa (sa ulaznom adresom 0101000_2), koje dovodi do smeštanja konstante 1 iz dodatne reči u registar `%2`. Nakon izvršavanja ovog mikro-programa, sledi opet obavljanje prve mikro-naredbe mikro-programa dobavljanja (sa adresom 0000001_2).

4.10. PITANJA

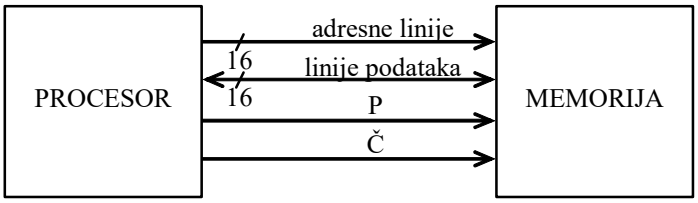
1. Koju adresu dekodira dekoderska funkcija A3&A2&A1&A0 (Ai označava i-ti bit adrese)?
2. Koje linije ulaze u memoriju?
3. Koje linije izlaze iz memorije?
4. Koje naredbe imaju jedan ulazno-izlazni i jedan ulazni registarski operand?

5. U kojim operandima naredbi procesora se koriste registri opšte namene?
6. U kojim operandima naredbi procesora se koriste vrednost/adresa?
7. Da li naredba `SABERI` ima dodatnu reč?
8. U kojoj fazi i u koji registar se smešta obavezna reč?
9. U kojoj fazi i u koji registar se smešta dodatna reč?
10. Šta sadrži procesor?
11. Šta sadrži aritmetičko logička jedinica?
12. Koje registre menja izvršavanje mikro-programa obavljanja naredbe `ODUZMI`?
13. Šta sadrži upravljačka jedinica?
14. Šta karakteriše mikro-programsku memoriju?
15. Koji registri se menjaju u poluciklusu dobavljanja upravljačke jedinice?
16. Koji registri se menjaju u poluciklusu obavljanja upravljačke jedinice?
17. Kako dekoder naredbe transformiše kod naredbe u adresu mikro-programa obavljanja?
18. Gde se koristi ulazna adresa mikro-programa dobavljanja?
19. Odakle dolazi adresa u mikro-programsku memoriju?
20. Koji su zajednički argumenti prekidačkih funkcija?
21. Kako se menja vrednost logičkih promenljivih `T` i `R`?
22. Koji su posebni argumenti prekidačkih funkcija?
23. Šta sadrži programski brojač?
24. Šta sadrži registar sekvence?
25. Šta sadrži upravljački registar?
26. Šta se dešava dok logička promenljive R ima vrednost 0?
27. U kom redosledu se izvršavaju mikro-programi?

5. RAČUNAR KONCEPT

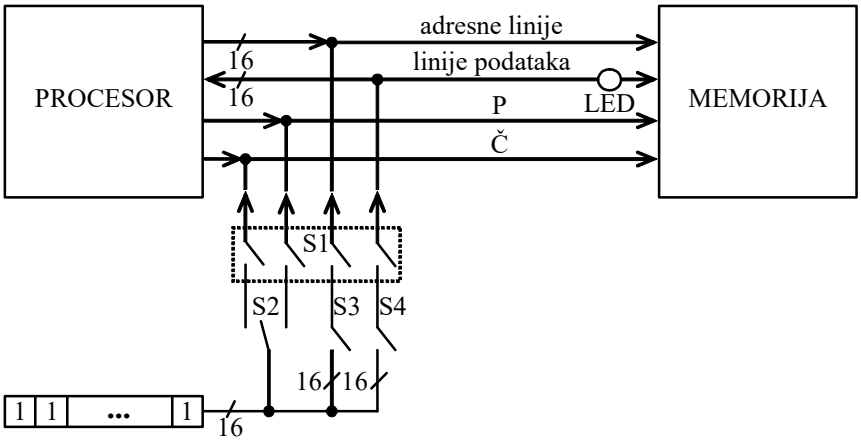
5.1. ORGANIZACIJA RAČUNARA KONCEPT

Slika 5.1.1 sadrži prikaz računara, sastavljenog od procesora i memorije.



Slika 5.1.1 Organizacija računara sastavljenog od procesora i memorije

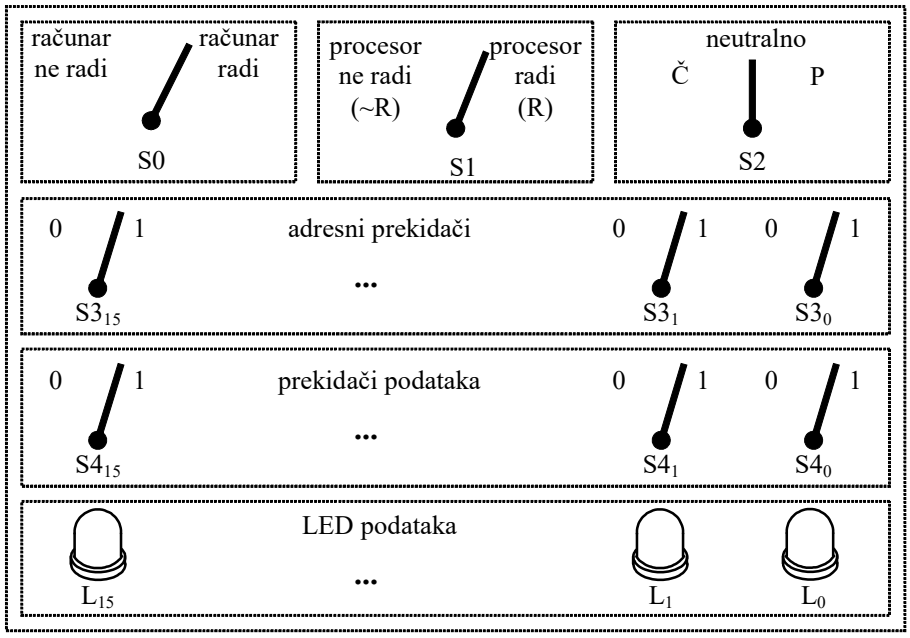
Računar koji se sastoji samo od procesora i memorije je neupotrebljiv, jer nema načina da se izvana izmeni ili pregleda sadržaj memorijskih lokacija, pa ni da se utiče na rad računara, a ni da se taj rad prati. Ovakav računar postaje upotrebljiv, ako se obezbedi vanjski uticaj na nivoe signala u upravljačkim linijama (P i Č), kao i u adresnim linijama i u linijama podataka. Takođe, potrebno je i prikazati nivoe signala u linijama podataka, koji odražavaju binarne, odnosno logičke vrednosti. Za vanjski uticaj na nivoe signala neophodni su mehanički prekidači, a za prikazivanje nivoa signala potrebne su svetlosne diode (Slika 5.1.2).



Slika 5.1.2 Organizacija računara KONCEPT

Oznaka S1 se odnosi na 34 međusobno povezana mehanička prekidača, koji imaju 2 položaja i svi se uvek nalaze u istom položaju. Zatvaranje ovih prekidača zaustavlja rad procesora i omogućuje uticanje na nivoe signala u upravljačkim

linijama (P i Č), ali i u adresnim linijama i u linijama podataka. Otvaranjem ovih prekidača započinje rad procesora. Oznaka S2 se odnosi na mehanički prekidač sa 3 položaja (levo, neutralno, desno), koji omogućuje uticanje na nivoe signala u upravljačkim linijama P i Č. Postavljanje ovog prekidača u desni položaj dovodi na nivo 1 signal u liniji P, a postavljanje ovog prekidača u levi položaj dovodi na nivo 1 signal u liniji Č. Oznaka S3 se odnosi na 16 međusobno nezavisnih mehaničkih prekidača, svaki sa po 2 položaja, koji omogućuju uticanje na nivoe pojedinih signala u adresnim linijama. Zatvaranje svakog od ovih prekidača dovodi na nivo 1 signal u korespondentnoj adresnoj liniji. Oznaka S4 se odnosi na 16 međusobno nezavisnih mehaničkih prekidača, svaki sa po 2 položaja, koji omogućuju uticanje na nivoe pojedinih signala u linijama podataka. Zatvaranje svakog od ovih prekidača dovodi na nivo 1 signal u korespondentnoj liniji podataka. Oznaka LED se odnosi na 16 međusobno nezavisnih svetlosnih dioda, koje omogućuju prikazivanje nivoa pojedinih signala u linijama podataka. Svaka od ovih svetlosnih dioda svetli samo ako je u njenoj liniji signal na nivou 1. Mehanički prekidači i svetlosne diode se nalaze na upravljačkoj tabli (Slika 5.1.3).



Slika 5.1.3 Upravljačka tabla računara KONCEPT

Upravljačka tabla sadrži i prekidač S0, sa 2 položaja. Kada je u desnom položaja, ovaj prekidač omogućuje rad računara, jer tada dovodi napajanje do njegovih sklopova. Ako je ovaj prekidač u suprotnom položaju, tada računar ne

radi, jer su njegovi sklopovi bez napajanja. Za prekidače S3 i S4 se podrazumeva da dovode signal na nivo 1 kada su u desnom položaju. Indeks uz oznake ovih prekidača, kao i uz oznake svetlosnih dioda (L), pokazuje za koju od linija su vezane odgovarajući prekidači, odnosno svetlosne diode.

Pisanje memorijske lokacije počinje dovođenjem prekidača S2 u neutralni, a prekidača S1 u levi položaj. Zatim se prekidači S3 i S4 postave u položaje određene adresom ove memorijske lokacije i njenim novim sadržajem. Potom se, prekidač S2 dovodi u desni položaj, čime se izmeni sadržaj date memorijske lokacije.

Čitanje memorijske lokacije počinje dovođenjem prekidača S2 u neutralni, a prekidača S1 u levi položaj. Zatim se prekidači S3 postave u položaje određene adresom ove memorijske lokacije. Potom se prekidač S2 dovodi u levi položaj. Dok je on u tom položaju, svetlosne diode prikazuju bite date memorijske lokacije.

Do izvršavanja datog programa dolazi nakon prebacivanja prekidača S1 u desni položaj, ako se, prethodno, u prvu memorijsku lokaciju (sa adresom 0000_{16}) smesti ulazna adresa ovog programa. Kraj izvršavanja programa se prepoznaje po pravilnoj izmeni stanja svetlećih dioda, ako se na kraju programa nalazi beskonačna petlja (jer se tada ponavlja izvršavanje iste naredbe, pa se u linijama podataka ponavljaju iste kombinacije nivoa signala, određene njenim mašinskim oblikom).

5.2. ULAZNI I IZLAZNI UREĐAJI RAČUNARA KONCEPT

Upravljačka tabla je nepregledna, pa je njeno korišćenje sporo i bremenito greškama. Zato je bolje binarnu komunikaciju korisnika i računara, koju nameće upravljačka tabla, zameniti znakovnom komunikacijom. To podrazumeva da se za saopštavanje **komandi** računaru koriste nizovi znakova, umesto položaja prekidača.

KOMANDNI JEZIK

Znakovni oblik komandi za pisanje (P) memorijske lokacije, za čitanje (Č) memorijske lokacije i za pokretanje izvršavanja (I) programa, opisuje pravilo komandnog jezika (*command language*), navedeno u *EBNF* notaciji:

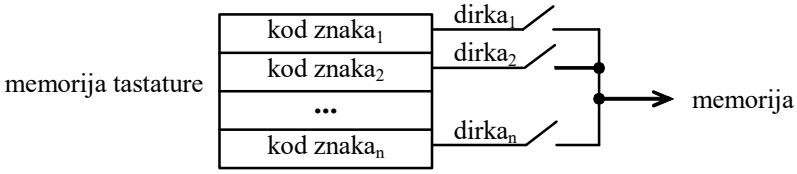
```
komanda -> P razmak broj razmak broj
         |Č razmak broj
         |I razmak broj
```

Brojevi predstavljaju argumente komandi. Prvi od njih označava adresu memorijske lokacije, a drugi označava novi sadržaj ove memorijske lokacije.

TASTATURA

Znakovna komunikacija korisnika i računara zahteva posebne ulazne i izlazne uređaje. Ulazni uređaj ima oblik tastature, sa dirkama koje pripadaju pojedinim znakovima. Pritisak na dirku generiše kod, dodeljen znaku kome dirka pripada. Radi jednostavnosti se može smatrati da su kodovi pojedinih znakova smešteni u

lokacije posebne memorije tastature i da pritisak na dirku zatvara prekidač koji omogućuje čitanje koda znaka iz odgovarajuće lokacije ove memorije (Slika 5.2.1).



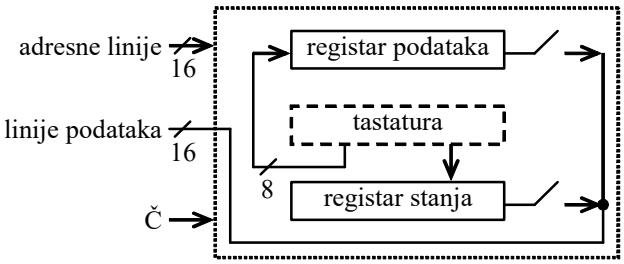
Slika 5.2.1 Princip rada tastature

Sa stanovišta procesora, zgodno je da generisani kod dospe u neku predodređenu memorijsku lokaciju, iz koje procesor može da čita ovaj kod kao i svaki drugi podatak. Prema tome, sa strane procesora, ova lokacija se ne razlikuje od ostalih memorijskih lokacija. Ona se ipak razlikuje od "običnih" memorijskih lokacija, jer je direktno vezana za tastaturu, sa koje prima svoj sadržaj. Zbog toga, ovakva lokacija ne pripada memoriji, nego posebnom sklopu, koji se naziva **kontroler tastature**, a čiji je osnovni zadatak je da posreduje između tastature i procesora. Pomenuta lokacija se naziva **registar podataka** (kontrolera) tastature. Podrazumeva se da registar podataka tastature uvek sadrži kod znaka koji odgovara poslednje pritisnutoj dirki tastature. Da bi se sprečilo da procesor višestruko preuzme kod istog znaka između dva uzastopna pritiska dirki tastature, neophodno je označiti kada se u registru podataka tastature nalazi nepreuzet, a kada već preuzet kod znaka. Radi toga se uvodi **registar stanja** (kontrolera) tastature, koji sa strane procesora liči na "običnu" memorijsku lokaciju, a čiji biti odražavaju stanje kontrolera tastature i automatski se menjaju pri izmenama stanja ovog kontrolera.

Nulti bit registra stanja tastature označava da li registar podataka tastature sadrži preuzet ili nepreuzet kod znaka. Kada je kod znaka preuzet, ovaj bit sadrži vrednost 0, a kada je kod znaka nepreuzet, ovaj bit sadrži vrednost 1. Inicijalni sadržaj ovoga bita je uvek vrednost 0. Pritisak bilo koje dirke tastature izaziva smeštanje koda odgovarajućeg znaka u registar podataka tastature, kao i smeštanje vrednosti 1 u nulti bit registra stanja tastature. Čitanje registra podataka tastature izaziva smeštanje vrednosti 0 u nulti bit registra stanja tastature.

Nema smisla čitati registar podataka tastature dok se u nultom bitu registra stanja tastature ne pojavi vrednost 1. Retko čitanje registra podataka tastature, ređe od pritiskanja dirki tastature, ima za posledicu gubljenje kodova znakova. U ovakvoj situaciji novi kod istiskuje zatečeni iz registra podataka tastature, jer zauzima njegovo mesto, a u nulti bit registra stanja tastature se smešta vrednost 1.

Sa stanovišta procesora, ulazni uređaj u obliku tastature predstavljaju dva registra kontrolera tastature, koje ima smisla samo čitati. Na ovaj način, kontroler tastature sakriva od procesora detalje funkcionisanja tastature i nudi jednoobrazan način pristupanja bilo kojoj tastaturi (Slika 5.2.2).

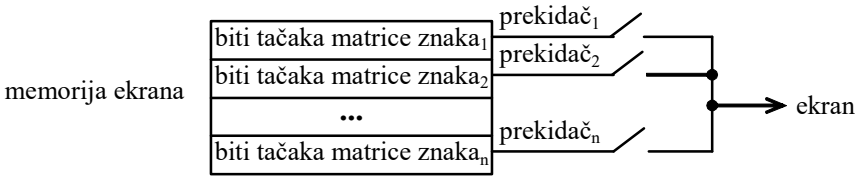


Slika 5.2.2 Organizacija kontrolera tastature

Za kontroler tastature se podrazumeva da 8 bitni kod, generisan u tastaturi, odlazi u 8 najmanje značajnih bita registra podataka tastature.

EKRAN

Izlazni uređaj ima oblik ekrana sa tačkama koje mogu biti osvetljene ili neosvetljene. Ove tačke su grupisane u (pravougaone) matrice, od kojih svaka obrazuje **znakovnu poziciju** za prikazivanje jednog znaka. Oblik znaka obrazuju osvetljene tačke znakovne pozicije, pri čemu se iz koda znaka određuje koje tačke su osvetljene. Radi jednostavnosti se može smatrati da kodovi pojedinih znakova predstavljaju adrese lokacija posebne memorije ekrana. Podrazumeva se da broj bita svake od ovih lokacija odgovara broju tačaka znakovne pozicije, a sadržaj svakog bita određuje osvetljenost korespondentne tačke znakovne pozicije (Slika 5.2.3).



Slika 5.2.3 Princip rada ekrana (kod $znaka_i$ izaziva zatvaranje $prekidača_i$)

Znakovne pozicije ekrana iz iste horizontale obrazuju liniju ekrana.

Znak se uvek prikazuje u znakovnoj poziciji koja sadrži poseban znak, nazvan **kursor** (*cursor*). Novi znak smenjuje kursor, a kursor se pomera horizontalno u sledeću, prvu desnu znakovnu poziciju iste linije. Kada se popune sve pozicije date linije, kursor prelazi u prvu (krajnje levu) znakovnu poziciju naredne (donje) linije. Ako se kursor nalazi u poslednjoj (krajnje desnoj) znakovnoj poziciji poslednje linije, tada se sadržaj svih linija pomera vertikalno na gore, tako što se znak iz svake znakovne pozicije pomera vertikalno u prvu gornju znakovnu poziciju. Opisano pomeranje sadržaja svih linija na gore dovodi do istiskivanja sadržaja prve linije (sa vrha) ekrana, radi oslobađanja prostora u poslednjoj liniji (sa dna) ekrana, da bi kursor prešao u njenu prvu znakovnu poziciju.

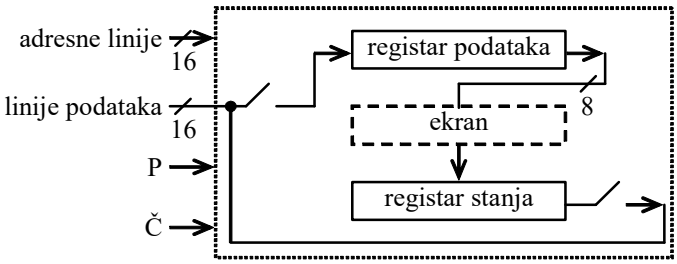
Upravljački znakovi omogućuju izbor znakovne pozicije u kojoj se prikazuje znak, jer po ekranu pomeraju kursor u izabranu znakovnu poziciju. Na primer, upravljački znak "nova linija" (*line feed*) pomera vertikalno kursor po ekranu u znakovnu poziciju ispod znakovne pozicije u kojoj je zatečen kursor. To može izazvati prethodno opisano pomeranje sadržaja svih linija ekrana. Slično tome, upravljački znak "početak linije" (*carriage return*) pomera kursor u prvu znakovnu poziciju linije u kojoj se kursor nalazi.

Sa stanovišta procesora, zgodno je da se kod prikazivanog znaka smesti u neku predodređenu memorijsku lokaciju, iz koje se taj kod upućuje na ekran. Prema tome, sa strane procesora ova lokacija se ne razlikuje od ostalih memorijskih lokacija. Ona se ipak razlikuje od "običnih" memorijskih lokacija, jer je direktno vezana za ekran. Zato ovakva lokacija ne pripada memoriji, nego posebnom sklopu, koji se naziva **kontroler ekrana**. Njegov osnovni zadatak je da posreduje između ekrana i procesora. Pomenuta lokacija se naziva registar podataka (kontrolera) ekrana. Podrazumeva se da registar podataka ekrana sadrži kod znaka koji je poslednji upućen na prikazivanje. Da bi se sprečilo da procesor upisuje u registar podataka ekrana kodove prikazivanih znakova brže nego što se njihovi oblici mogu prikazati, neophodno je označiti kada je prikazan znak, čiji kod je upisan u registar podataka ekrana. Radi toga se uvodi registar stanja (kontrolera) ekrana, koji sa strane procesora liči na "običnu" memorijsku lokaciju, a čiji biti odražavaju stanje kontrolera ekrana i automatski se menjaju pri izmenama stanja ovog kontrolera.

Nulti bit registra stanja ekrana označava da li je znak, čiji kod je u registru podataka ekrana, prikazan ili ne. Kada je znak prikazan, ovaj bit sadrži vrednost 1, a kada znak nije prikazan, ovaj bit sadrži vrednost 0. Inicijalni sadržaj ovoga bita je uvek vrednost 1. Upisivanje koda znaka (koji se prikazuje) u registar podataka ekrana izaziva smeštanje vrednosti 0 u nulti bit registra stanja ekrana. Prikazivanje ovog znaka izaziva smeštanje vrednosti 1 u nulti bit registra stanja ekrana.

Nema smisla upisivati kod novog znaka u registar podataka ekrana, dok se u nultom bitu registra stanja ekrana ne pojavi vrednost 1. Suviše brzo pisanje kodova znakova u registar podataka ekrana, brže od prikazivanja znakova na ekranu, ima za posledicu gubljenje prikazivanih znakova. U ovakvoj situaciji novi kod istiskuje zatečeni iz registra podataka ekrana, zauzimajući njegovo mesto, a u nulti bit registra stanja ekrana se smešta vrednost 0.

Sa stanovišta procesora, izlazni uređaj u obliku ekrana predstavljaju dva registra kontrolera ekrana. Na ovaj način kontroler ekrana sakriva od procesora detalje funkcionisanja ekrana i nudi jednoobrazan način pristupanja bilo kom ekranu (Slika 5.2.4).



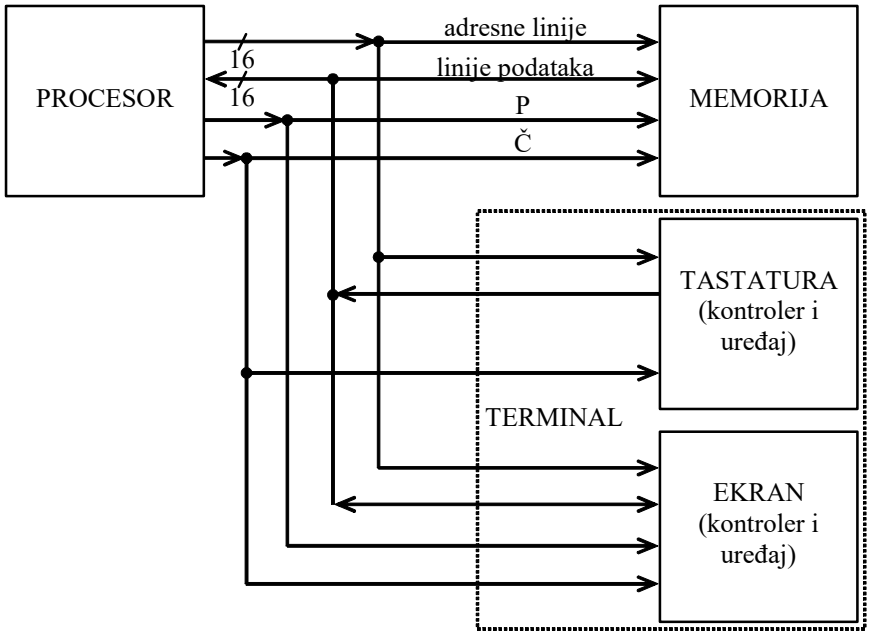
Slika 5.2.4 Organizacija kontrolera ekrana

Za kontroler ekrana se podrazumeva da se 8 bitni kod, koji ide ka ekranu, preuzima iz 8 najmanje značajnih bita registra podataka ekrana.

Funkcionisanje izlaznog uređaja, u obliku matričnog štampača, se suštinski ne razlikuje od prethodno opisanog funkcionisanja ekrana.

RAČUNAR KONCEPT SA ZNAKOVNIM ULAZOM I IZLAZOM

Slika 5.2.5 prikazuje upotrebljivu verziju računara KONCEPT, sastavljenu od procesora, memorije, tastature i ekrana.



Slika 5.2.5 Organizacija računara KONCEPT sa znakovnim ulazom i izlazom

Sa stanovišta procesora, registri podataka i registri stanja kontrolera se ne razlikuju od "običnih" memorijskih lokacija. Oni zamenjuju "obične" memorijske lokacije kojima, po pravilu, odgovaraju najviše adrese.

STANDARDNI ZNAKOVNI KODOVI

Kodiranje znakova regulišu međunarodni dogovori ili standardi, kao što je *ASCII* (*American Standard Code for Information Interchange*).

Kod	Znak	Kod	Znak	Kod	Znak	Kod	Znak
000	*NUL (Null character)*	020	*SP (Space)*	040	@	060	`
001	*SOH (Start of Header)*	021	!	041	A	061	a
002	*STX (Start of Text)*	022	"	042	B	062	b
003	*ETX (End of Text)*	023	#	043	C	063	c
004	*EOT (End of Transmission)*	024	$	044	D	064	d
005	*ENQ (Enquiry)*	025	%	045	E	065	e
006	*ACK (Acknowledgment)*	026	&	046	F	066	f
007	*BEL (Bell)*	027	'	047	G	067	g
008	*BS (Backspace)*	028	(	048	H	068	h
009	*HT (Horizontal Tab)*	029	)	049	I	069	i
00A	*LF (Line Feed)*	02A	*	04A	J	06A	j
00B	*VT (Vertical Tab)*	02B	+	04B	K	06B	k
00C	*FF (Form Feed)*	02C	,	04C	L	06C	l
00D	*CR (Carriage Return)*	02D	-	04D	M	06D	m
00E	*SO (Shift Out)*	02E	.	04E	N	06E	n
00F	*SI (Shift In)*	02F	/	04F	O	06F	o
010	*DLE (Data Link Escape)*	030	0	050	P	070	p
011	*DC1 (XON/Device Control 1)*	031	1	051	Q	071	q
012	*DC2 (Device Control 2)*	032	2	052	R	072	r
013	*DC3 (XOFF/Device Control 3)*	033	3	053	S	073	s
014	*DC4 (Device Control 4)*	034	4	054	T	074	t
015	*NAK (Negative Ack)*	035	5	055	U	075	u
016	*SYN (Synchronous Idle)*	036	6	056	V	076	v
017	*ETB (End of Trans. Block)*	037	7	057	W	077	w
018	*CAN (Cancel)*	038	8	058	X	078	x
019	*EM (End of Medium)*	039	9	059	Y	079	y
01A	*SUB (Substitute)*	03A	:	05A	Z	07A	z
01B	*ESC (Escape)*	03B	;	05B	[	07B	{
01C	*FS (File Separator)*	03C	<	05C	\	07C	|
01D	*GS (Group Separator)*	03D	=	05D	]	07D	}
01E	*RS (Record Separator)*	03E	>	05E	^	07E	~
01F	*US (Unit Separator)*	03F	?	05F	_	07F	*DEL (Delete)*

Slika 5.2.6 Sedmobitna *ASCII* tabela (kodovi su dati heksadecimalno)

U sedmobitnoj verziji *ASCII* standarda (Slika 5.2.6), znakovi nacionalnih alfabeta zamenjuju pojedine od znakova *ASCII* standarda (Š : [, Ć :], Č : ^, Ž : @,

Đ : \, š : {, ć : }, č : ~, ž : ‘, đ : |). Osmobitna verzija *ASCII* standarda koristi dodatnih 128 znakova za reprezentovanje znakova nacionalnih alfabeta. Pošto takvih znakova ima mnogo više od 128, uvedene su kodne strane (*code page*) za grupe pojedinih nacionalnih alfabeta. Svaka kodna strana sadrži 256 znakova, a da bi se kodovi znakova interpretirali u skladu sa kodnom stranom, neophodno je za tekst vezati podatak o važećoj kodnoj strani. Međutim, ni ovaj pristup ne nudi zadovoljavajuće rešenje (naročito ne za jezike poput kineskog). Zato je predložen *UNICODE* standard, koji uvodi šesnaestobitne, odnosno tridesetdvobitne kodove za pojedine znakove. Prvih 256 kodova odgovara *ASCII* kodovima za kodnu stranicu *Latin-1*, a ostali kodovi su podeljeni u zone koje su dodeljene pojedinim nacionalnim alfabetima.

ZNAKOVNA I INTERNA PREDSTAVA CELIH BROJEVA

Zahvaljujući postojanju znakovnih kodova za cifre, brojevi se mogu smeštati u lokacije u **internom** (binarnom), ali i u **znakovnom** obliku. Tako, binarni broj:

10000001_2

može biti smešten u jednu 8 bitnu lokaciju, ali i u 8 ovakvih lokacija:

$31_{16}\ 30_{16}\ 30_{16}\ 30_{16}\ 30_{16}\ 30_{16}\ 30_{16}\ 31_{16}$

U prethodnom primeru svaka binarna cifra je predstavljena svojim znakovnim kodom, pa zauzima jednu 8 bitnu lokaciju.

Brojevi se preuzimaju sa ulaznog uređaja (tastature) u znakovnom obliku i neophodno je da budu u istom obliku radi prikaza na izlaznom uređaju (ekranu). Međutim, aritmetičke operacije zahtevaju da brojevi budu i internom obliku. Zato je neophodna konverzija brojeva iz znakovnog u interni oblik, ali i obrnuto.

Konverzija broja iz znakovnog u interni oblik zahteva (1) da se iz znakovnog koda svake cifre odredi njena interna predstava i (2) da se interna predstava cifre smesti u odgovarajuću poziciju. To se postiže (1) oduzimanjem znakovnog koda cifre nula od znakovnog koda posmatrane cifre datog broja i (2) množenjem tako dobijene razlike odgovarajućim stepenom broja 2 (baze brojnog sistema). Postupak se ponavlja za znakovne kodove svih cifara datog broja idući sa leva u desno.

Konverzija broja iz internog u znakovni oblik zahteva (1) da se za internu predstavu svake cifre odredi njen znakovni kod i (2) da se znakovni kodovi poređaju u odgovarajućem redosledu. To se postiže (1) dodavanjem znakovnog koda cifre nula na vrednost svake cifre iz interne predstave broja i (2) i smeštanjem dobijenog znakovnog koda cifre u odgovarajuću lokaciju.

ZNAKOVNA INTERAKCIJA KORISNIKA I RAČUNARA

Komandu, koja se sastoji od niza znakova, korisnik saopštava računaru pomoću tastature. Pritiskom na odgovarajuću dirku tastature korisnik unosi, s leva u desno, jedan po jedan znak komande. Uneseni znakovi se prikazuju na ekranu, da bi korisnik mogao da uoči eventualno pogrešno unesen znak. Pogrešno uneseni znakovi se poništavaju jedan po jedan, s desna u levo. Znači, prvi se poništava poslednje unesen znak. Poništavanju znakova je posvećena posebna dirka tastature (*Backspace*). Pritisak na dirku za poništavanje generiše kod odgovarajućeg upravljačkog znaka poništavanja.

Prikaz unesenih znakova na ekranu se naziva **eho**, a unos znakova i ispravljanje pogrešno unesenih znakova se naziva **editiranje**. Zbog eha i editiranja, korisnik doživljava tastaturu i ekran kao jedinstven uređaj, koji se naziva terminal.

Oblik neposredne komunikacije korisnika i računara, koji se javlja u editiranju, se naziva **interaktivni rad**.

Nakon ispravnog unosa svih znakova komande, sledi prepoznavanje vrste komande i izdvajanje njenih argumenata. Zatim dolazi ili do izmene sadržaja navedene memorijske lokacije (u slučaju komande P), ili do prikazivanja na ekranu, u heksadecimalnom obliku, sadržaja navedene memorijske lokacije (u slučaju komande Č), ili do pokretanja izvršavanja korisničkog programa (u slučaju komande I). Prepoznavanje vrste komande, izdvajanje njenih argumenata i njeno izvršavanje se zajedno nazivaju **interpretiranje komande**. Interpretiranje komande nastupa tek nakon njenog saopštavanja, znači, nakon unosa svih njenih znakova, odnosno, po završetku editiranja.

Za označavanje kraja saopštavanja komande služi posebna dirka tastature (*enter*). Pritisak na ovu dirku (za kraj unosa) generiše kod odgovarajućeg upravljačkog znaka kraja unosa (*Line Feed* ili *Carriage Return*).

Interpretiranje komandi je u nadležnosti posebnog programa, koji se naziva **interpreter komandi**. Pre preuzimanja komande, interpreter komandi na ekranu prikazuje poseban znak **prompt**, kojim saopštava korisniku da je spreman za preuzimanje komande:

```
for(;;)
 { prikaži prompt i preuzmi komandu;
   if ((prvi znak komande je P)&&
       (sledi heksadecimalni broj)&&
       (sledi heksadecimalni broj))
      { izmeni sadržaj navedene memorijske lokacije }
   else
    if ((prvi znak komande je Č)&&
        (sledi heksadecimalni broj))
       { prikaži sadržaj navedene memorijske lokacije }
    else
     if ((prvi znak komande je I)&&
         (sledi heksadecimalni broj))
        { pokreni izvršavanje korisničkog programa }
     else
        { prikaži poruku greške }
 }
```

U toku prepoznavanja komandi, interpreter komandi sistematski ispituje sve dozvoljene mogućnosti (predviđene komandnim jezikom), a izvršavanju komande pristupa čim prepozna ispravnu komandu i izdvoji njene argumente. Pri izdvajanju, argumenti (brojevi) se prevode iz znakovnog u interni oblik. Obrnuto se dešava kod prikazivanja (u znakovnom obliku) sadržaja memorijskih lokacija.

Za pokretanje izvršavanja programa (komanda I) najbolje je da korisnički program bude u obliku potprograma, čije izvršavanje se pokreće naredbom `POZOVI`, a završava naredbom `NATRAG`. Na taj način se, po izvršavanju korisničkog programa, automatski nastavlja izvršavanje interpretera komandi. Opisani interpreter komandi podrazumeva da se korisnički programi unose u mašinskom obliku.

Preuzimanje znakova (koji obrazuju komande), odnosno prikazivanje znakova (koji predstavljaju sadržaj memorijske lokacije) nalazi se u nadležnosti posebnog programa koji se naziva **drajver** (*driver*) terminala. On se sastoji od 2 potprograma koji podržavaju 2 operacije. Prva, **ulazna operacija** je namenjena za preuzimanje znakova sa terminala (sa tastature), a druga, **izlazna operacija** je namenjena za prikaz znakova na terminalu (na ekranu). Ove dve operacije omogućuju korišćenje terminala bez poznavanja detalja njegovog funkcionisanja. Na taj način, one objedinjuju tastaturu i ekran u jedinstven uređaj.

U ulaznoj operaciji se obavljaju preuzimanje, eho i editiranje preuzetih znakova. Preuzeti znakovi se odlažu u ulaznu memorijsku zonu koja se naziva **ulazni bafer**. Ulazni bafer pripada programu kao što je interpreter komandi, iz koga je ulazna operacija pozvana. Poziv ulazne operacije sadrži, kao argumente, adresu početka i dužinu ulaznog bafera. Opisana ulazna operacija omogućuje preuzimanje samo znakova koji pristignu sa tastature nakon njenog poziva.

Preuzimanje jednog znaka u okviru ulazne operacije opisuje sledeća asemblerska sekvenca:

```
                    PREBACI_NR      $0,%0
čekaj_tastaturu:    PREBACI_DR      registar_stanja_tastature,%1
                    UPOREDI         %1,%0
                    SKOČI_ZA_==     čekaj_tastaturu
                    PREBACI_DR      registar_podataka_tastature,%2
```

Podrazumeva se da je `registar_stanja_tastature` labela registra stanja tastature, da je `registar_podataka_tastature` labela registra podataka tastature, da su registri `%0` i `%1` radni registri, kao i da je preuzeti znak (njegov kod) smešten u registar `%2`.

Eho preuzetog znaka u okviru ulazne operacije opisuje sledeća asemblerska sekvenca:

```
                    PREBACI_NR      $0,%0
čekaj_ekran:        PREBACI_DR      registar_stanja_ ekrana,%1
                    UPOREDI         %1,%0
                    SKOČI_ZA_==     čekaj_ekran
                    PREBACI_RD      %2,registar_podataka_ekrana
```

Podrazumeva se da je `registar_stanja_ekrana` labela registra stanja ekrana, da je `registar_podataka_ekrana` labela registra podataka ekrana, da su registri `%0` i `%1` radni registri, kao i da se znak (njegov kod), čiji eho se vrši, nalazi u registru `%2`.

U obe prethodno navedene asemblerske sekvence ulazne operacije prisutno je **radno čekanje** (*busy waiting*, *polling*). Ono se sastoji od ponavljanja provere da li je ispunjen uslov, neophodan za preuzimanje, odnosno za eho znaka. Ovakva provera se ponavlja do ispunjenja pomenutog uslova.

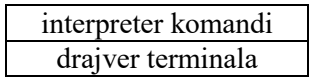
Izvršavanje prethodno navedenih asemblerskih sekvenci se ponavlja dok se ne preuzme ili zadani broj znakova, koji odgovara dužini ulaznog bafera, ili upravljački znak kraja unosa. Preuzimanje upravljačkog znaka poništavanja izaziva, ne samo izbacivanje poslednjeg znaka iz ulaznog bafera, nego i izbacivanje tog znaka sa ekrana. To zahteva vraćanje kursora u znakovnu poziciju koja prethodi njegovoj zatečenoj znakovnoj poziciji i smeštanje razmaka u novu znakovnu poziciju kursora, radi poništavanja u njoj zatečenog znaka.

U izlaznoj operaciji se obavlja samo prikaz znakova. On se ne razlikuje od eha znakova. Prikaz znakova se ponavlja, dok se ne prikaže i poslednji od znakova iz izlazne memorijske zone koja se naziva **izlazni bafer**. Izlazni bafer pripada programu kao što je interpreter komandi, iz koga je izlazna operacija pozvana. Poziv izlazne operacije sadrži, kao argumente, adresu početka i dužinu izlaznog bafera, koja određuje broj prikazivanih znakova.

BIOS

Iz prethodno izloženog proizlazi da su drajver terminala i interpreter komandi ravnopravni delovi računara KONCEPT kao i procesor, memorija ili terminal. Drajver terminala i interpreter komandi se zajedno nazivaju ***BIOS*** (*Basic Input Output System*). *BIOS* predstavlja programski deo računara. On se prikazuje u

obliku hijerarhije slojeva, u kojoj se u višem sloju nalazi interpreter komandi, a u nižem sloju drajver terminala (Slika 5.2.7).



Slika 5.2.7 Slojevita struktura *BIOS*-a

U okviru *BIOS*-a interpreter komandi se nalazi iznad drajvera terminala, jer se oslanja na ovaj drajver tako što poziva njegovu ulaznu operaciju radi preuzimanja znakova komandi i tako što poziva njegovu izlaznu operaciju radi prikaza rezultata izvršavanja komandi.

Ispod *BIOS*-a se nalaze preostali delovi računara, na koje se *BIOS* oslanja. Iznad *BIOS*-a su korisnički programi koji se oslanjaju na *BIOS*, jer pozivaju operacije drajvera terminala.

Prethodno opisana organizacija *BIOS*-a podrazumeva dva nivoa njegovog korišćenja: (1) **interaktivni nivo** i (2) **programski nivo**. Na interaktivnom nivou korisnici direktno koriste komande interpretera komandi, a na programskom nivou se, iz korisničkih programa, pozivaju operacije drajvera terminala.

Verzija računara KONCEPT, koju obrazuju procesor, memorija, terminal i *BIOS*, sadrži upravljačku tablu na kojoj su samo dva prekidača. Prvi od njih je prekidač napajanja. On ima 2 stabilna položaja. Kada je prekidač napajanja u jednom položaju, računar radi (njegovi sklopovi imaju napajanje), a kada se on nalazi u drugom položaju, računar ne radi (njegovi sklopovi nemaju napajanje). Drugi prekidač omogućuje korisniku da pokrene rad procesora iz početka (*reset*). On ima jedan stabilan položaj (u kome logička promenljiva R ima vrednost 1) i jedan nestabilan položaj (u kome logička promenljiva R ima vrednost 0). Dovođenje ovog prekidača u nestabilan položaj uzrokuje da procesor započne rad iz početka, čime se procesor dovodi u poznato stanje. Na ovaj način se, na primer, ponovo aktivira *BIOS*, kada korisnički program upadne u beskonačnu petlju.

Aktiviranje *BIOS*-a znači da se pokrene izvršavanje njegovog interpretera komandi, koji, posredstvom drajvera terminala, stupa u interakciju sa korisnikom. Za automatsko aktiviranje *BIOS*-a neophodno je da prva memorijska lokacija (sa adresom 0000_{16}) sadrži ulaznu adresu *BIOS*-a. Takođe, neophodno je da su sadržaj prve memorijske lokacije i sadržaji memorijskih lokacija sa naredbama *BIOS*-a nepromenljivi, odnosno da se sadržaji ovih lokacija mogu samo čitati.

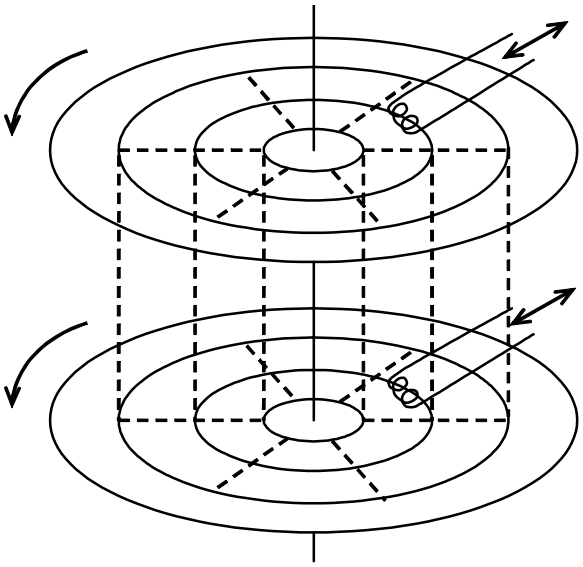
5.3. VRSTE MEMORIJE

Za smeštanje naredbi *BIOS*-a nisu zgodne "obične" memorijske lokacije, pošto njihov sadržaj može biti izmenjen, a nehotične izmene njihovog sadržaja mogu ugroziti funkcionalnost *BIOS*-a, a to znači i upotrebljivost računara KONCEPT. Za rešavanje problema smeštanja naredbi *BIOS*-a nije dovoljno

poznavati samo organizaciju memorije, nego i njena tehnološka svojstva. Po tehnologiji izrade, memorije se razvrstavaju na poluprovodničke i magnetne.

Osnovna podela poluprovodničkih memorija je na *RAM* (*random access memory*) i *ROM* (*read only memory*). Prve dozvoljavaju i pisanje i čitanje svojih lokacija, pa su podesne za memoriju računara. One nisu postojane, jer gube sadržaj pri nestanku napajanja. Druge dozvoljavaju samo čitanje svojih lokacija. One su zato podesne za nultu memorijsku lokaciju (sa adresom 0000_{16}), kao i za memorijske lokacije, predviđene za smeštanje naredbi *BIOS*-a. One su postojane, jer njihov sadržaj, koji je upisan posebnim postupkom, nije zavisan od napajanja.

Magnetnih memorija ima više vrsta, a njihov tipičan predstavnik je **disk** (Slika 5.3.1).



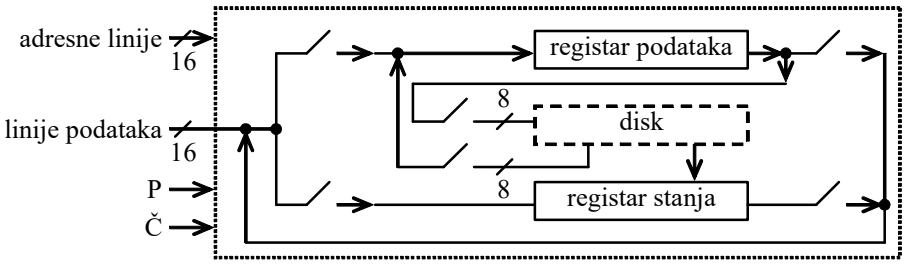
Slika 5.3.1 Principijelni izgled magnetnog diska

Na magnetnom disku memorisani sadržaji se nalaze u obliku magnetnih zapisa, raspoređenih u koncentričnim kružnim **stazama** (*track*) koje **glava** (za čitanje i pisanje) **diska** opisuje na magnetnom premazu rotirajućih ploča diska. I glava diska je pokretna, s tim da se ona pomera radijalno, znači ili od centra ili ka centru rotacije ploča diska. Staze diska su podeljene u **sektore** (*sector*) iste veličine. Radi razlikovanja staza i sektora, svakoj stazi, a i svakom sektoru na njoj, se pridružuje jednoznačan broj. Jedinica čitanja i pisanja diska je **blok**, koji obuhvata sve bajte jednog ili eventualno više sektora. Za čitanje, odnosno za pisanje bloka diska, neophodno je pozicionirati glavu diska iznad odgovarajuće staze i iznad odgovarajućeg sektora na ovoj stazi. Disk sadrži pomoćnu memoriju (u *RAM* izvedbi), veliku bar 1 blok, u koju se smešta sadržaj pri čitanju bloka iz sektora,

odnosno iz koje se preuzima sadržaj pri pisanju bloka u sektor. Pošto dozvoljava i čitanje i pisanje blokova, i disk je podesan za memoriju računara. Disk je, kao i sve magnetne memorije, postojan, jer njegov sadržaj ne zavisi od napajanja.

Iako i poluprovodničke memorije (u *RAM* izvedbi) i diskovi poseduju svojstva potrebna za memoriju računara, ipak ni jedna od ove dve vrste memorije nije idealno prilagođena pomenutoj nameni. Mana poluprovodničkih memorija (u *RAM* izvedbi) je njihova nepostojanost i mali kapacitet. Za razliku od diska, cena bita ove vrste memorija je previsoka za masovnu upotrebu. S druge strane, mana diska je njegova brzina. Za razliku od poluprovodničkih memorija, disk je, zbog svojih pokretnih delova, mnogo sporiji od procesora. Zato poluprovodničke memorije služe za privremeno smeštanje sadržaja koji su potrebni za rad procesora, a diskovi služe za trajno odlaganje sadržaja. Prema tome, memorija se sastoji od **radne** (poluprovodničke) **memorije**, čija brzina je približna brzini procesora, a čiji kapacitet je mali, i **masovne** (magnetne) **memorije**, koja je znatno sporija od procesora, ali je postojana i ima veliki kapacitet. Sadržaji, trajno pohranjeni u masovnoj memoriji, se prebacuju u radnu memoriju samo kada su potrebni za rad procesora. Do prebacivanja u suprotnom smeru dolazi samo radi trajnog pohranjivanja sadržaja. Jedinica prenosa sadržaja između radne i masovne memorije je blok. Prenos bloka na ovoj relaciji nije moguć bez komunikacije između procesora i diska, u toku koje procesor saopštava disku brojeve staze i sektora, a disk, kao sporiji uređaj, diktira dinamiku prenosa.

Komunikacija procesora i diska se suštinski ne razlikuje od komunikacije procesora i tastature (ekrana), pa je prirodno da se osloni na posredovanje **kontrolera diska**. Pri tome, registar podataka (kontrolera) diska omogućuje, ne samo prenos bajta bloka u oba smera, nego i prenos broja staze i broja sektora od procesora ka disku. Nulti bit registra stanja (kontrolera) diska pokazuje kada je komunikacija moguća, a prvi i drugi bit ovog registra služe za označavanje smera prenosa bloka. Kada nulti bit registra stanja diska sadrži vrednost 1, tada procesor može preuzeti ili izmeniti sadržaj registra podataka diska. Registar podataka diska nije spreman za novi prenos sve dok ovaj bit sadrži vrednost 0. Pristupanje procesora registru podataka diska dovodi do smeštanja vrednosti 0 u nulti bit registra stanja diska. Sadržaj ovoga bita postane jednak vrednosti 1, čim registar podataka diska postane spreman za novi prenos. Inicijalni sadržaj nultoga bita registra stanja diska je vrednost 1. Kada prvi bit registra stanja diska sadrži vrednost 1, tada se blok prenosi na disk, a kada drugi bit registra stanja diska sadrži vrednost 1, tada se blok prenosi sa diska. Procesor smešta vrednost 1 u jedan od ova dva bita, zavisno od smera prenosa bloka. Pri tome, on piše u registar stanja diska ne menjajući sadržaj nultog bita ovog registra. Smeštanje vrednosti 1 u prvi ili drugi bit registra stanja diska označava početak prenosa bloka. Postavljeni bit se automatski anulira na kraju prenosa bloka, kada se prenese njegov poslednji bajt. Slika 5.3.2 sadrži prikaz organizacije kontrolera diska.



Slika 5.3.2 Organizacija kontrolera diska

U prenosu bajta između diska i registra podataka diska učestvuje samo 8 najmanje značajnih bita ovog registra. Stanje prekidača na liniji, koja ide od registra podataka diska do diska, zavisi od sadržaja prvog bita registra stanja diska, a stanje prekidača na liniji, koja ide od diska do registra podataka diska, zavisi od sadržaja drugog bita registra stanja diska.

Prenos bloka započinje procesor, smeštanjem oznake smera prenosa u registar stanja diska. Zatim sledi radno čekanje, dok registar podataka diska ne postane spreman za prijem broja staze. Nakon smeštanja broja staze u registar podataka diska sledi radno čekanje, dok ovaj registar ne postane raspoloživ za prijem broja sektora. Po smeštanju broja sektora u registar podataka diska, za svaki bajt prenošenog bloka se ponavlja radno čekanje, dok pomenuti bajt ne dođe sa diska u registar podataka diska, odnosno dok on ne ode iz registra podataka diska na disk, što zavisi od smera prenosa.

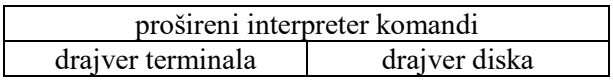
Prethodno opisani postupak korišćenja diska je sakriven u **drajveru diska**, koji brine da pristup registrima kontrolera diska bude u skladu sa pravilima njihove upotrebe. Pored toga, drajver diska pojednostavljuje korišćenje diska, jer dozvoljava da se, umesto brojeva staze i sektora, upotrebljavaju samo redni brojevi blokova koji se prenose na disk i sa diska. Jasno, zadatak drajvera diska je da redni broj bloka pretvori u broj staze i broj sektora. Drajver diska sadrži 2 potprograma. Jedan od njih podržava ulaznu operaciju, namenjenu za čitanje sadržaja bloka, a drugi podržava izlaznu operaciju namenjenu za pisanje sadržaja bloka. Uz poziv ulazne operacije kao argumenti se navode adresa ulaznog bafera i redni broj bloka, čiji sadržaj se prebacuje u ulazni bafer, a uz poziv izlazne operacije se navode adresa izlaznog bafera i redni broj bloka, u koji se prebacuje sadržaj izlaznog bafera. Podrazumeva se da baferi pripadaju programu iz koga se dotična operacija poziva.

Za prenos bloka na disk i sa diska su potrebne posebne komande komandnog jezika. Radi ovih komandi se proširuje pravilo koje opisuje komandni jezik u *EBNF* notaciji:

```
komanda -> P razmak broj razmak broj
          |Č razmak broj
          |I razmak broj
          |N razmak broj razmak broj
          |S razmak broj razmak broj
```

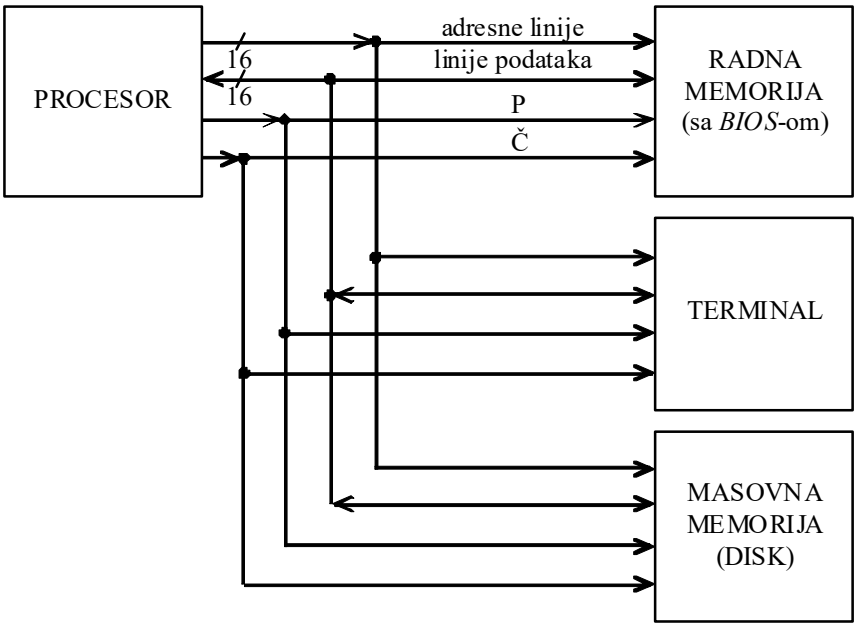
Komanda **N** omogućuje prenos bloka na disk tako što poziva izlaznu operaciju drajvera diska. Komanda **S** omogućuje prenos bloka sa diska tako što poziva ulaznu operaciju drajvera diska. Obe komande imaju dva argumenta u obliku brojeva. Prvi od njih odgovara adresi bafera, a drugi rednom broju bloka koji učestvuje u prenosu. Za komandu **N** u adresiranom baferu mora biti pripremljen sadržaj (na primer, mašinski oblik programa) koji se prenosi na disk i koji kasnije može biti vraćen u radnu memoriju.

Uvođenje prethodne dve komande zahteva proširenje interpretera komandi. Prema tome, u gornjem sloju *BIOS*-a se nalazi prošireni interpreter komandi, a u donjem sloju su drajver terminala i drajver diska (Slika 5.3.3).



Slika 5.3.3 Slojevita struktura proširenog *BIOS*-a

Nova verzija računara KONCEPT sadrži procesor, radnu memoriju, terminal, masovnu memoriju i *BIOS* (Slika 5.3.4).



Slika 5.3.4 Organizacija računara KONCEPT sa radnom i masovnom memorijom

Slika 5.3.4 sadrži kvadrate sa natpisima TERMINAL i MASOVNA MEMORIJA koji označavaju i kontrolere i odgovarajuće fizičke uređaje.

Radna memorija ima manje lokacija od broja koji dopušta adresni prostor računara KONCEPT, zbog visoke cene radne memorije. Najveći deo lokacija radne memorije je u *RAM* izvedbi, a samo mali deo ima *ROM* izvedbu, radi *BIOS*-a.

5.4. KODOVI ZA OTKRIVANJE I POPRAVKU GREŠAKA

Nesavršena izvedba memorije može uzrokovati nepredvidive izmene pojedinih bita i tako izazvati greške u binarnim (internim) predstavama celih brojeva sadržanim u memoriji. Zato, ispravno rukovanje podacima iz memorije zahteva **otkrivanje i popravku grešaka** (*error detection and correction*) u njihovim bitima.

Otkrivanje pogrešnih bita se zasniva na uvođenju **bita provere** (*check bit*) koji zajedno sa bitima podataka obrazuje **kodnu reč** (*codeword*). Da bi biti podataka i bit provere obrazovali kodnu reč, neophodno je da oni zadovoljavaju zadano pravilo. Važenje takvog pravila narušava pojava pogrešnih bita. Znači, otkrivanje pogrešnih bita se zasniva na proveri da li za bite podataka i bit provere važi zadano pravilo.

Za otkrivanje pogrešnih bita obično se uvodi **bit** (provere) **parnosti** (*parity bit*) koji se postavlja tako da kodna reč sadrži paran broj jedinica. U tabeli (Slika

5.4.1) su prikazane sve moguće kodne reči sa dva bita podataka i jednim bitom parnosti.

kodne reči	
biti podataka	bit parnosti
00	0
01	1
10	1
11	0

Slika 5.4.1 Sve kodne reči sa tri bita

Izmena bilo kog bita u bilo kojoj kodnoj reči iz tabele (Slika 5.4.1) daje tri bita koji ne predstavljaju kodnu reč, jer ne sadrže paran broj jedinica. Ali, izmene bilo koja dva bita u bilo kojoj kodnoj reči iz tabele (Slika 5.4.1) daju opet kodnu reč. Ako se uoči da se sve kodne reči iz tabele (Slika 5.4.1) međusobno razlikuju u dva bita, tada sledi zaključak da je broj pogrešnih bita koji se mogu otkriti pomoću ovakvih kodnih reči za jedan manji od broja bita po kojima se ovakve kodne reči razlikuju. Broj različitih bita između bilo koje dve kodne reči se naziva **Hamingova udaljenost** (*Hamming distance*). Znači, ako je Hamingova udaljenost kodnih reči n+1, tada je moguće otkriti do n pogrešnih bita. Drukčije rečeno, tek izmena n+1 bita prevodi jednu kodnu reč u drugu i čini izmene nevidljivim.

Prisustvo više bita parnosti u istoj kodnoj reči omogućuje otkrivanje izmenjenih (pogrešnih) bita, ali i njihovu popravku. Pretpostavka je da se svaki od bita parnosti odnosi na poseban podskup bita podataka. Podrazumeva se da su podskupovi bita podataka tako odabrani da izmena nekog od bita podataka remeti više bita parnosti. Iz toga se zaključuje koji bit podataka je izmenjen, pa se on i popravlja. U tabeli (Slika 5.4.2) su prikazani podskupovi bita podataka i njihovi biti parnosti za kodnu reč sa četiri bita bita podataka i tri bita parnosti. Ovakve kodne reči omogućuju popravku jednog pogrešnog bita.

	redni brojevi bita kodne reči						
biti parnosti	1	2		4			
biti podataka			3		5	6	7
podskup 1. bita parnosti	→		+		+		+
podskup 2. bita parnosti		→	+			+	+
podskup 3. bita parnosti				→	+	+	+
primer kodne reči	0	1	1	0	0	1	1

Slika 5.4.2 Ustrojstvo kodne reči sa četiri bita podataka i tri bita parnosti

Iz tabele (Slika 5.4.2) sledi:

1. ako je netačan samo jedan od bita parnosti, tada je on pogrešan
2. ako su netačni biti parnosti 1 i 2, tada je pogrešan bit podataka 3
3. ako su netačni biti parnosti 1 i 4, tada je pogrešan bit podataka 5
4. ako su netačni biti parnosti 2 i 4, tada je pogrešan bit podataka 6, a
5. ako su netačni biti parnosti 1,2 i 4, tada je pogrešan bit podataka 7

Treba zapaziti da su biti kodne reči iz tabele (Slika 5.4.2) tako numerisani da suma rednih brojeva pogrešnih bita parnosti daje redni broj pogrešnog bita podataka. Podrazumeva se da redni brojevi bita parnosti odgovaraju stepenima broja 2: 2^0 (1), 2^1 (2), 2^2 (4) i tako dalje. Biti podataka se raspoređuju po podskupovima tako da njihov redni broj odgovara sumi rednih brojeva bita parnosti ovih skupova.

Kodne reči, poput onih iz tabele (Slika 5.4.2), predložio je *R. Hamming* 1950. godine, pa se zato kaže da ovakve kodne reči predstavljaju **Hamingov kod** (*Hamming code*).

5.5. OPERATIVNI SISTEM

BIOS olakšava korišćenje računara, jer omogućuje znakovnu komunikaciju korisnika i računara. Ali, korišćenje računara na nivou *BIOS*-a zahteva poznavanje detalja, kao što su, na primer, mašinski formati naredbi. Pored toga, korisnik je prisiljen i da vodi evidenciju o korišćenju masovne i radne memorije. Ova evidencija obuhvata podatke o slobodnim i zauzetim blokovima, odnosno o slobodnim i zauzetim memorijskim lokacijama, pri čemu se za zauzete blokove, odnosno za zauzete memorijske lokacije navodi šta im je sadržaj. Zbog toga je korišćenje računara na nivou *BIOS*-a komplikovano, a to znači i neproduktivno. Za jednostavnije i produktivnije korišćenje računara je neophodno da se korisnička pažnja ne skreće na detalje koji su nevažni sa stanovišta korisnika. Korisnik računara je suštinski zainteresovan za upotrebu sadržaja koji su trajno pohranjeni u blokovima masovne memorije. Za njega je sporedno gde se ti sadržaji nalaze i kakvu internu predstavu imaju. Znači, ako se ovi sadržaji (njihova interna predstava i organizacija) nazovu **datoteka** (*file*), korisnik je zainteresovan za sadržaje datoteka, ali ne i za njihovu internu predstavu i organizaciju.

Svi sadržaji, smešteni u masovnu memoriju, su uobličeni kao datoteke. Prema tome, i mašinski oblici programa se čuvaju u masovnoj memoriji kao datoteke. Pre izvršavanja, mašinski oblici programa se privremeno smeštaju u lokacije radne memorije. Pri tome, za korisnika je sporedno kako i gde se smešta mašinski oblik programa, odnosno kakva je interna predstava ovakvog programa u toku njegovog izvršavanja, jer je korisnik suštinski zainteresovan samo za rezultate angažovanja procesora na pomenutom izvršavanju programa. Znači, ako se angažovanje procesora na izvršavanju programa nazove **proces** (*process*), korisnik je zainteresovan za rezultate aktivnosti procesa, ali ne i za njihovu organizaciju.

STRUKTURA OPERATIVNOG SISTEMA

Koncept datoteke omogućuje razdvajanje upotrebe sadržaja datoteka od potrebe poznavanja njihove organizacije. Zahvaljujući ovom razdvajanju, korisnik ne mora da poznaje način čuvanja sadržaja datoteka, odnosno, organizaciju blokova masovne memorije. Brigu o tome u kojim blokovima je sadržaj datoteke ili koliko je sadržaj datoteke dugačak preuzima poseban programski **modul za rukovanje datotekama**. Da bi koncept datoteke imao praktičnu vrednost, ovaj modul mora da sadrži potprograme koji podržavaju operacije za rukovanje datotekama. Primeri takvih operacija su operacija za stvaranje ili uništenje datoteke, odnosno operacije za preuzimanje i izmenu njenog sadržaja. Za rukovanje datotekama neophodno je razlikovanje datoteka. Zato, svakoj datoteci, pri njenom stvaranju, korisnik pridružuje ime koje je sastavljeno od znakova iz nekog predodređenog skupa.

Ostvarenje koncepta datoteke se zasniva na postojanju **deskriptora datoteke**, koji obuhvata **atribute datoteke**, kao što su njena veličina, redni brojevi blokova sa sadržajem datoteke ili vreme nastanka datoteke. Modul za rukovanje datotekama rukuje deskriptorima datoteka.

Koncept procesa omogućuje razdvajanje upotrebe (izvršavanja) programa od potrebe poznavanja njihove organizacije. Zahvaljujući ovom razdvajanju, korisnik ne mora da poznaje ni detalje organizacije lokacija radne memorije sa mašinskim oblicima naredbi programa, a ni mašinske formate naredbi. Brigu o tome u kojim lokacijama radne memorije su mašinski oblici naredbi programa preuzima poseban programski **modul za rukovanje procesima**. Da bi koncept procesa imao praktičnu vrednost, ovaj modul mora da sadrži potprograme koji podržavaju operacije za rukovanje procesima. Primeri takvih operacija su operacije za stvaranje ili uništenje procesa.

Svaki proces karakterišu mašinski oblik izvršavanog programa, kao i vrednosti promenljivih i sadržaj steka koji se menjaju u toku aktivnosti procesa. Oni zajedno obrazuju **sliku procesa** (*process image*).

Ostvarenje koncepta procesa se zasniva na postojanju **deskriptora procesa**, koji obuhvata **atribute procesa**, kao što su broj lokacija radne memorije potrebnih za smeštanje slike procesa ili evidencija lokacija u koje je ona smeštena. Modul za rukovanje procesima rukuje deskriptorima procesa.

Rukovanje datotekama i procesima se oslanja na rukovanje slobodnim blokovima, odnosno slobodnim memorijskim lokacijama, radi njihovog zauzimanja i oslobađanja. Dok blokove zauzima i oslobađa samo modul za rukovanje datotekama, memorijske lokacije zauzimaju i oslobađaju i modul za rukovanje datotekama i modul za rukovanje procesima. Zato se rukovanje slobodnim blokovima prepušta modulu za rukovanje datotekama, a za rukovanje slobodnim memorijskim lokacijama se uvodi poseban **modul za rukovanje radnom memorijom**, koji sadrži potprograme koji podržavaju operacije za zauzimanje i oslobađanje lokacija radne memorije. Prema tome, na modul za rukovanje radnom memorijom se oslanjaju moduli za rukovanje datotekama i procesima. Takođe,

modul za rukovanje procesima se oslanja na modul za rukovanje datotekama, jer se mašinski oblici programa od kojih nastaju slike procesa nalaze u datotekama koje se nazivaju **izvršne datoteke**. Izvršne datoteke sadrže **inicijalne slike procesa**. U njima se nalaze mašinski oblici programa, njihove ulazne adrese i dužine, podaci o promenljivim i slično

Moduli za rukovanje procesima, datotekama i radnom memorijom se oslanjanju na drajvere. Na primer, sva tri modula koriste drajver terminala za prikaz poruka, a modul za rukovanje datotekama koristi drajver diska za pristupanje blokovima diska. Drajveri obrazuju **modul za rukovanje kontrolerima**. Moduli za rukovanje procesima, datotekama, radnom memorijom i kontrolerima zajedno obrazuju **operativni sistem** (*operating system*). Slika 5.5.1 sadrži prikaz slojevite strukture operativnog sistema.

modul za rukovanje procesima
modul za rukovanje datotekama
modul za rukovanje radnom memorijom
modul za rukovanje kontrolerima

Slika 5.5.1 Slojevita struktura operativnog sistema

Slojevita struktura operativnog sistema odražava međusobni odnos delova od kojih se on sastoji, jer se moduli iz viših slojeva oslanjaju na module iz nižih slojeva tako što pozivaju njihove operacije. Operativni sistem predstavlja programski deo računara. Ispod operativnog sistema su preostali delovi računara, na koje se operativni sistem oslanja. Iznad operativnog sistema su korisnički programi, koji se oslanjaju na operativni sistem, jer pozivaju operacije njegovih modula. Iz strukture operativnog sistema sledi da je njegov zadatak da objedini raznorodne delove računara u skladnu celinu i, uz to, da sakrije od korisnika sve o funkcionisanju računara što nije bitno za njegovo korišćenje.

INTERPRETER KOMANDI OPERATIVNOG SISTEMA

Nakon usvajanja koncepata datoteke i procesa, direktno pristupanje pojedinim blokovima i memorijskim lokacijama postaje, ne samo suvišno, nego i opasno, jer omogućuje proizvoljne izmene deskriptora datoteka i procesa, što može da ugrozi ispravan rad operativnog sistema. Zato komandni jezik, koji se oslanja na operativni sistem, ne sme da sadrži komande za direktno pristupanje blokovima i memorijskim lokacijama. Za jedinu preostalu komandu, namenjenu za pokretanje izvršavanja programa, dovoljno je da odgovara imenu izvršne datoteke. Prema tome, zadatak interpretera komandi je da preuzme ime izvršne datoteke i da od modula za rukovanje procesima zatraži stvaranje odgovarajućeg procesa. Po stvaranju, procesor se posveti stvorenom procesu, tako da nastavak izvršavanja interpretera komandi usledi tek nakon kraja aktivnosti stvorenog procesa. Da bi obavio svoj zadatak, interpreter komandi se oslanja na modul za rukovanje

kontrolerima (na njegov drajver terminala), radi znakovne komunikacije sa korisnikom. Ime, koje ne odgovara izvršnoj datoteci, ne može biti komanda interpretera komandi. Takva datoteka ne sadrži inicijalnu sliku procesa, pa modul za rukovanje procesima na osnovu njenog sadržaja ne može da stvori proces. Prema tome, jedino imena izvršnih datoteka predstavljaju ispravne komande interpretera komandi.

Interpreter komandi spada u korisničke programe i nalazi se u korisničkom sloju iznad operativnog sistema.

Zahvaljujući interpreteru komandi, postoje dva nivoa korišćenja operativnog sistema: (1) interaktivni nivo i (2) programski nivo. Na interaktivnom nivou korisnici koriste komande interpretera komandi operativnog sistema, a na programskom nivou korisnički programi pozivaju operacije pojedinih modula operativnog sistema. Ove operacije se nazivaju **sistemski pozivi**. Komande interpretera komandi operativnog sistema i sistemski pozivi obrazuju dva različita korisnička interfejsa prema operativnom sistemu.

Operativni sistem se prema korisničkim programima odnosi kao prema svojim potprogramima. Očigledan primer takvog odnosa operativnog sistema i korisničkih programa pružaju programi napisani u programskom jeziku C. Svaki C program sačinjavaju potprogrami (funkcije), među kojima se nalazi obavezno i potprogram (funkcija) sa imenom *main*. Podrazumeva se da izvršavanje C programa započinje izvršavanjem potprograma (funkcije) *main*. Njega poziva operativni sistem, započinjući tako aktivnost odgovarajućeg procesa.

SISTEMSKI PROGRAMI

Rad na interaktivnom nivou operativnog sistema je moguć samo ako postoje izvršne datoteke, čija imena služe kao komande interpretera komandi. U suprotnom slučaju, nema mogućnosti za komunikaciju korisnika i računara. Zato uz operativni sistem obavezno dolaze izvršne datoteke, koje odgovaraju posebnim, **sistemskim programima**, kao što su **editor**, **prevodilac** (asembler ili kompajler), **makro pretprocesor**, povezilac ili **linker** (*linker*), punilac ili **louder** (*loader*) i **dibager** (*debugger*). U nadležnosti editora su stvaranja i izmene **tekst** (znakovnih) **datoteka**. Zadatak asemblera je prevođenje asemblerskog programa u mašinski oblik, a zadatak kompajlera je prevođenje programa, napisanog u programskom jeziku visokog nivoa, u mašinski oblik. Oba prevodioca pronalaze program u odgovarajućoj **izvornoj** (programskoj tekst) **datoteci**, a rezultat prevođenja (mašinski oblik programa koji je nespreman za interakciju sa operativnim sistemom) smeštaju u **objektnu datoteku**. Zadatak makro pretprocesora je da vrši tekstualne izmene izvornih datoteka. Pošto njegova aktivnost prethodi aktivnosti prevodilaca, on se često javlja i kao njihov početni deo. Linker stvara izvršnu datoteku. Ona sadrži inicijalnu sliku procesa, odnosno mašinski oblik programa koji je spreman za interakciju sa operativnim sistemom. Louder učestvuje, često kao deo operativnog sistema, u stvaranju slike procesa koja nastaje na osnovu inicijalne

slike procesa. Dibager omogućuje nadgledanje izvršavanja korisničkih programa, radi otkrivanja i otklanjanja grešaka. Kao sistemski programi se javljaju i pomoćni programi za rukovanje datotekama. Oni omogućuju pregledanje i izmene imena postojećih datoteka, prepisivanje sadržaja iz datoteke u datoteku, uništavanje datoteka i slično. Ovakvi pomoćni programi su ponekad deo inerpretera komandi. Svi sistemski programi koriste operativni sistem na nivou sistemskih poziva.

Operativni sistem i sistemski programi stvaraju uopšten model računara, čije poznavanje je sasvim dovoljno, ne samo za korisnike gotovih programa, nego i za aplikacione programere. Ovaj model je nedovoljan samo za sistemske programere, koji se bave razvojem operativnih sistema i sistemskih programa.

ODNOS *BIOS*-A I OPERATIVNOG SISTEMA

BIOS i operativni sistem nude dva različita pogleda na računar. Korisnik *BIOS*-a vidi računar kao mašinu koja rukuje blokovima i memorijskim lokacijama, a korisnik operativnog sistema vidi računar kao mašinu koja rukuje datotekama i procesima. Pošto računar započinje rad izvršavanjem *BIOS*-a, prirodno je da se pokretanje operativnog sistema osloni na *BIOS*. Komande *BIOS*-a su dovoljne da se blokovi, koji sadrže operativni sistem, odnosno, koji sadrže mašinske oblike modula operativnog sistema, prebace sa diska u radnu memoriju, i da se, zatim, pokrene izvršavanje operativnog sistema. Ali, takav način pokretanja operativnog sistema je spor i bremenit greškama, jer se mora prebaciti veliki broj blokova. Zato se pokretanju operativnog sistema posvećuje poseban program, nazvan **inicijalni louder** (*bootstrap loader*). On se pokreće uz pomoć *BIOS*-a. Za izvršni oblik inicijalnog loudera se rezerviše jedan, na primer, nulti blok (*boot block*). Prema tome, na početku rada računara *BIOS* pročita sa nultog bloka izvršni oblik inicijalnog loudera, smesti ga u radnu memoriju i započne njegovo izvršavanje. U toku svog izvršavanja inicijalni louder prebaci u radnu memoriju module operativnog sistema i započne njegovo izvršavanje.

Prethodno opisani način pokretanja operativnog sistema dozvoljava korisniku da, pokretanjem jednog od više inicijalnih loudera, odabere za rad jedan od više operativnih sistema, i na taj način stvori za sebe najbolje radno okruženje.

5.6. PREKLJUČIVANJE

Radno čekanje (dešavanja događaja van procesora), prisutno u radu drajvera, je izvor neracionalnog trošenja procesorskog vremena. Umesto radnog čekanja, bolje je koristiti procesor u **višeprocesnom režimu rada**. U ovom režimu rada u radnoj memoriji istovremeno postoji više slika procesa, a procesor se **prekljućuje** (*context switch, dispatching*) sa procesa čija aktivnost je postala zavisna od dešavanja događaja van procesora, na proces za čiju aktivnost jedino nedostaje procesor. Od istovremeno postojećih procesa, čije slike se istovremeno nalaze u radnoj memoriji, uvek je samo jedan u stanju **'aktivan'**, a ostali su ili u stanju **'čeka'** (dešavanje vanjskog događaja) ili u stanju **'spreman'** (znači, za aktivnost im

nedostaje samo procesor). Za preključivanje je neophodno da bar jedan od neaktivnih istovremeno postojećih procesa bude u stanju 'spreman'. Radi toga postoji poseban **sistemski** proces, koji je uvek ili spreman ili aktivan. Njegov jedini zadatak je da angažuje procesor kada nema drugih istovremeno postojećih procesa. Tada aktivnost ovoga procesa predstavlja radno čekanje nove komande interpretera komandi. Sistemski proces angažuje procesor i kada su svi ostali istovremeno postojeći procesi u stanju 'čeka'. Tada aktivnost ovoga procesa predstavlja radno čekanje dešavanja odgovarajućeg vanjskog događaja. Do preključivanja procesora dolazi (1) na kraju aktivnosti procesa i (2) kada nastavak aktivnosti procesa zavisi od dešavanja vanjskog događaja, kao što je obavljanje ulaza/izlaza vezanog za terminal ili disk. Zato se ovakvo preključivanje naziva **ulazom-izlazom vođeno preključivanje**. Do ulazom-izlazom vođenog preključivanja dolazi, na primer, kada nastavak aktivnosti procesa zavisi od preuzimanja znaka sa tastature terminala. Za ulazom-izlazom vođeno preključivanje je karakteristično da u stanje 'čeka' prelazi aktivni proces (sa koga se procesor preključuje), a da u stanje 'aktivan' prelazi spreman proces (na koga se procesor preključuje). Ovakvo preključivanje se poziva iz drajvera, radi izbegavanja radnog čekanja, ali i iz korisničkog programa, na kraju njegovog izvršavanja, i to posredstvom modula za rukovanje procesima. Zato se potprogram ili operacija preključivanja nalazi u posebnom **modulu za rukovanje procesorom**. Ovaj modul je nazvan tako, jer on odlučuje o angažovanju procesora tako što bira proces kome će se posvetiti procesor. Modul za rukovanje procesorom se nalazi ispod modula za rukovanje kontrolerima, jer operacije drajvera pozivaju njegov potprogram preključivanja. Slika 5.6.1 sadrži prikaz slojeva operativnog sistema, proširenog modulom za rukovanje procesorom.

modul za rukovanje procesima
modul za rukovanje datotekama
modul za rukovanje radnom memorijom
modul za rukovanje kontrolerima
modul za rukovanje procesorom

Slika 5.6.1 Slojevi operativnog sistema

Svi istovremeno postojeći procesi su upućeni na iste resurse računara, kao što su procesor, radna i masovna memorija. Uz pretpostavku da su svi istovremeno postojeći procesi nezavisni (da imaju sopstvene zone radne memorije i sopstvene datoteke, kojima ne pristupaju drugi istovremeno postojeći procesi), kao deljeni resurs se javljaju samo procesorski registri opšte namene. Zato se, u preključivanju procesora, sadržaji ovih registara prvo sačuvaju, na primer u **registarskom baferu** procesa koga procesor napušta, a zatim se ovi registri napune prethodno sačuvanim sadržajima iz registarskog bafera procesa kome se procesor posvećuje.

Podrazumeva se da registarski bafer sadrži posebnu lokaciju za smeštanje sadržaja svakog registra koji je relevantan za preključivanje. Prirodno je da registarski bafer predstavlja deo deskriptora procesa.

Preključivanje procesora se obavlja u toku izvršavanja potprograma preključivanja. Njegovo izvršavanje se poziva u toku aktivnosti jednog procesa, a po završetku ovog izvršavanja aktivnost nastavlja drugi proces. Pretpostavka je da, pre poziva potprograma preključivanja, u registar `%0` bude smeštena adresa registarskog bafera deskriptora procesa sa koga se procesor preključuje, a da u registar `%1` bude smeštena adresa registarskog bafera deskriptora procesa na koga se procesor preključuje. Podrazumeva se da su registri `%0` i `%1` radni registri, tako da njihov prethodni sadržaj nije bitan. Asemblerski oblik potprograma preključivanja izgleda:

```
                    POČETAK       preključivanje
IZBACI              MAKRO         R
                    PREBACI_RP    R,(%0)
                    DODAJ_1       %0
                    KRAJ
UBACI               MAKRO         R
                    PREBACI_PR    (%1),R
                    DODAJ_1       %1
                    KRAJ
preključivanje:     IZBACI        %2
                    IZBACI        %3
                    IZBACI        %4
                    IZBACI        %5
                    IZBACI        %6
                    IZBACI        %7
                    IZBACI        %8
                    IZBACI        %9
                    IZBACI        %10
                    IZBACI        %11
                    IZBACI        %12
                    IZBACI        %13
                    IZBACI        %14
                    IZBACI        %15
                    UBACI         %2
                    UBACI         %3
                    UBACI         %4
                    UBACI         %5
                    UBACI         %6
                    UBACI         %7
                    UBACI         %8
                    UBACI         %9
                    UBACI         %10
                    UBACI         %11
                    UBACI         %12
                    UBACI         %13
                    UBACI         %14
                    UBACI         %15
                    NATRAG
                    KRAJ
```

Prethodni potprogram preključivanja je napravljen pod pretpostavkom da registarski bafer sadrži 14 lokacija, namenjenih za smeštanje sadržaja registara od `%2` do `%15`. Ovaj potprogram se ne brine o sadržaju status registra, jer se podrazumeva da status registar nije ažuran nakon poziva potprograma. Ključnu ulogu u preključivanju ima registar `%15`, jer sadrži povratnu adresu. Izmenom njegovog sadržaja se menja i povratna adresa, a to znači i proces čija aktivnost se nastavlja nakon izvršavanja potprograma preključivanja. Pri tome se podrazumeva da, tokom stvaranja procesa, njegova ulazna adresa bude smeštena u njegov registarski bafer u lokaciju koja je rezervisana za sadržaj registra `%15`. Zahvaljujući tome, nakon prvog preključivanja procesora na ovaj proces, izvršavanje započinje od njegove ulazne naredbe.

5.7. PREKID

Ulazom-izlazom vođeno preključivanje je u potpunosti podređeno povećanju iskorišćenja procesora. Zato, do ulazom-izlazom vođenog preključivanja dolazi samo kada se ustanovi da procesor ne može biti više korisno angažovan u okviru aktivnog procesa. Pri tome, preključivanju obavezno prethodi (1) izmena stanja više istovremeno postojećih procesa i (2) pronalaženje spremnog procesa na koga će se procesor preključiti. Pomenute izmene stanja obuhvataju ne samo aktivan proces, sa koga se procesor preključuje, odnosno spreman proces, na koga se procesor preključuje, nego i procese u stanju 'čeka', ako su se u međuvremenu desili vanjski događaji koji omogućuju nastavljanje njihove aktivnosti. Zato se, u okviru prethodnih izmena stanja procesa, proverava da li se neki od pomenutih vanjskih događaja desio, radi eventualnog prevođenja odgovarajućeg procesa iz stanja 'čeka' u stanje 'spreman'. Ovakve provere dešavanja vanjskih događaja su uvek i samo vezane za ulazom-izlazom vođeno preključivanje. Zato, ovakvo preključivanje ne nudi brzu reakciju procesora na dešavanje vanjskih događaja, jer do pomenute reakcije ne dolazi neposredno po dešavanju vanjskog događaja, nego tek u okviru prvog preključivanja, koje usledi iza dešavanja pomenutog vanjskog događaja. Između dva ulazom-izlazom vođena preključivanja može da se desi više uzastopnih vanjskih događaja iste vrste, kao što su uzastopni pritisci dirki tastature. Posledica je da, na primer, kod znaka može da bude istisnut iz registra podataka tastature pre nego što ga procesor preuzme. Na ovaj način prouzrokovan gubitak koda znaka čini znakovnu komunikaciju nepouzdanom, a interaktivni rad praktično nemogućim. U prethodnom primeru, procesor pravovremeno reaguje na dešavanje vanjskog događaja, ako preuzme svaki kod znaka iz registra podataka tastature neposredno po njegovom prispeću u ovaj registar. To se desi, ako svaki pritisak na dirku tastature izazove izvršavanje posebnog potprograma drajvera terminala, koji je zadužen za preuzimanje znaka iz registra podataka tastature. Za ostvarenje prethodnog neophodno je da izvršavanje pomenutog potprograma drajvera terminala prekine zatečenu aktivnost procesora, ali tako da se na kraju ovog izvršavanja prekinuta aktivnost procesora nastavi, kao da nije bilo njenog

prekidanja. Znači, u opštem slučaju, dešavanje vanjskog događaja, kao što je pritisak na dirku tastature, ili prikazivanje znaka na ekranu, ili kraj prenosa jednog bajta između diska i njegovog kontrolera, izaziva **prekid** (*interrupt*) aktivnosti procesora. U okviru prekida pokreće se izvršavanje potprograma koji se naziva **obrađivač prekida** (*interrupt handler*). Ulazna adresa ovog potprograma se naziva **vektor** (*vector*) obrađivača prekida. Izvršavanje ovoga potprograma se naziva obrada prekida. Prekidi se, po svom poreklu, dele u vrste, kao što su prekid tastature, prekid ekrana ili prekid diska. Svakoj vrsti prekida je pridružen jedan obrađivač prekida. Prekide izazivaju kontroleri nakon dešavanja odgovarajućeg vanjskog događaja. Pri tome oni obaveštavaju procesor o vrsti prekida i tako omogućuju procesoru da započne odgovarajuću obradu prekida.

MEHANIZAM PREKIDA

Svaku vrstu prekida kodira jednoznačan ceo broj. On se naziva **broj vektora**, jer ukazuje na vektor obrađivača prekida, pridruženog dotičnoj vrsti prekida. Za prenos broja vektora od kontrolera ka procesoru podesne su linije podataka (koje povezuju svaki od kontrolera sa procesorom). Pošto ove linije nisu uvek slobodne, kontroler ne upućuje samoinicijativno broj vektora procesoru, nego ga prvo samo obavesti o prekidu. Za to služi posebna linija **najave prekida** (*interrupt request*). Tek kada linije podataka postanu slobodne, procesor zahteva od kontrolera broj vektora. Za to služi posebna linija **potvrde prekida** (*interrupt acknowledge*). Po prijemu, procesor koristi broj vektora kao indeks posebne **tabele** (vektora) **prekida** (*interrupt vector table*), čiji elementi sadrže vektore obrađivača prekida. Na taj način, procesor konvertuje broj vektora u vektor obrađivača prekida, pridruženog odgovarajućoj vrsti prekida.

Tabela prekida se nalazi na predodređenom mestu radne memorije, na primer u prvih n njenih lokacija. Podrazumeva se da je nulta lokacija i dalje rezervisana za ulaznu adresu *BIOS*-a.

Za obradu prekida je dovoljno smestiti vektor obrađivača prekida u programski brojač. Ali, prethodno se mora sačuvati zatečeni sadržaj programskog brojača, da bi se prekinuta aktivnost mogla nastaviti. Isto važi i za status registar, jer se i njegov sadržaj menja u obradi prekida. Za čuvanje zatečenog sadržaja programskog brojača služi registar `%13`, a za čuvanje zatečenog sadržaja status registra služi registar `%14`. Ovakva namena registara `%13` i `%14` onemogućava njihovu upotrebu kao registara opšte namene (isto važi i za registar `%15`, u koga se smešta povratna adresa pri pozivu potprograma). Procesor se ne brine o sadržaju preostalih registara opšte namene, jer je racionalnije da se na početku obrade prekida sačuvaju, a na kraju obrade prekida vrate zatečeni sadržaji samo onih registara opšte namene koji se koriste u obradi prekida.

Obrađivači prekida nisu obični potprogrami, jer je za njihov kraj vezano vraćanje u programski brojač i status registar sadržaja koji su u ovim registrima zatečeni pre početka obrade prekida i sačuvani u registrima `%13` i `%14`. Za vraćanje

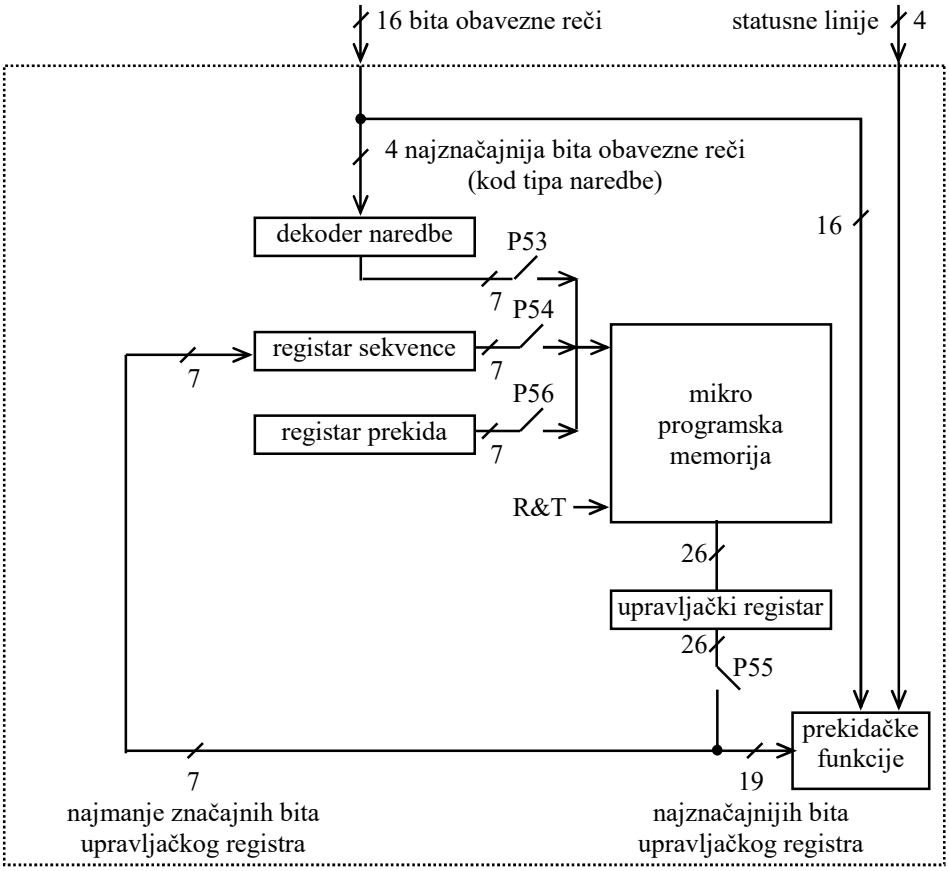
ovih sadržaja u pomenute registre namenjena je posebna naredba procesora KONCEPT, nazvana `NASTAVI`, jer omogućuje nastavljanje prekinute aktivnosti procesora (ona zahteva poseban mikro-program, jer ne pripada ni jednom od postojećih tipova naredbi procesora KONCEPT).

U registrima `%13` i `%14` ima mesta samo za po jedan sadržaj programskog brojača i status registra. Zato procesor KONCEPT ne dozvoljava da jedna obrada prekida bude prekinuta, radi druge obrade prekida. Znači, sve obrade prekida su istog nivoa. Omogućenost, odnosno onemogućenost obrade prekida pokazuje peti bit (SR_4) status registra, koji se naziva **bit prekida**. Prekidi su omogućeni kada ovaj bit sadrži vrednost 1, a onemogućeni kada on sadrži vrednost 0. Procesor automatski onemogućuje obradu novih prekida u toku trajanja svake obrade prekida.

U slučaju prekida raznih nivoa, kada svaku obradu prekida može da prekine hitnija obrada prekida, za čuvanje zatečenih sadržaja programskog brojača i status registra se koristi stek (i tada je neophodna procesorska podrška rukovanju stekom).

IZVEDBA PREKIDA

Precizan opis aktivnosti procesora u slučaju prekida je sadržan u **mikro-programu prekida**. Njegov mašinski oblik sadrži mikro-programska memorija, a njegovu ulaznu adresu sadrži poseban **registar prekida** upravljačke jedinice, čiji sadržaj je stalan (Slika 5.7.1). Uvođenje mikro-programa prekida i mikro-programa naredbe `NASTAVI` zahteva proširenje mikro-programske memorije i druge izmene u skladu sa tim.



Slika 5.7.1 Organizacija upravljačke jedinice koja podržava prekide

Zatvaranje prekidača P56 dovodi do izvršavanja mikro-programa prekida. To se desi samo nakon najave prekida od kontrolera, ako je obrada prekida omogućena, a linije podataka slobodne.

Na najavu prekida od kontrolera ukazuje vrednost 1 logičke promenljive **NAJAVA**. Prekidi su omogućeni kada bit prekida (SR_4) sadrži vrednost 1. Linije podataka su uvek slobodne neposredno pre nego procesor započne fazu dobavljanja naredbe. Znači, ako su prekidi omogućeni, tada, nakon najave prekida, preostaje samo da se sačeka nastupanje faze dobavljanja naredbe, da bi, umesto izvršavanja mikro-programa dobavljanja, započelo izvršavanje mikro-programa prekida.

Do nastupanja faze dobavljanja naredbe dolazi nakon smeštanja ulazne adrese 0000001_2 mikro-programa dobavljanja u registar sekvence. Prisustvo ove ulazne adrese u registru sekvence detektuje logička funkcija **PRE_DOBAVLJANJA**:

$$\sim RS_6 \& \sim RS_5 \& \sim RS_4 \& \sim RS_3 \& \sim RS_2 \& \sim RS_1 \& RS_0$$

Ova logička funkcija ima vrednost 1 samo ako pomenuti biti registra sekvence, sadrže ulaznu adresu 0000001_2 mikro-programa dobavljanja. Prema tome, mogućnost obrade prekida opisuje funkcija **PREKID**:

NAJAVA&SR$_4$&PRE_DOBAVLJANJA

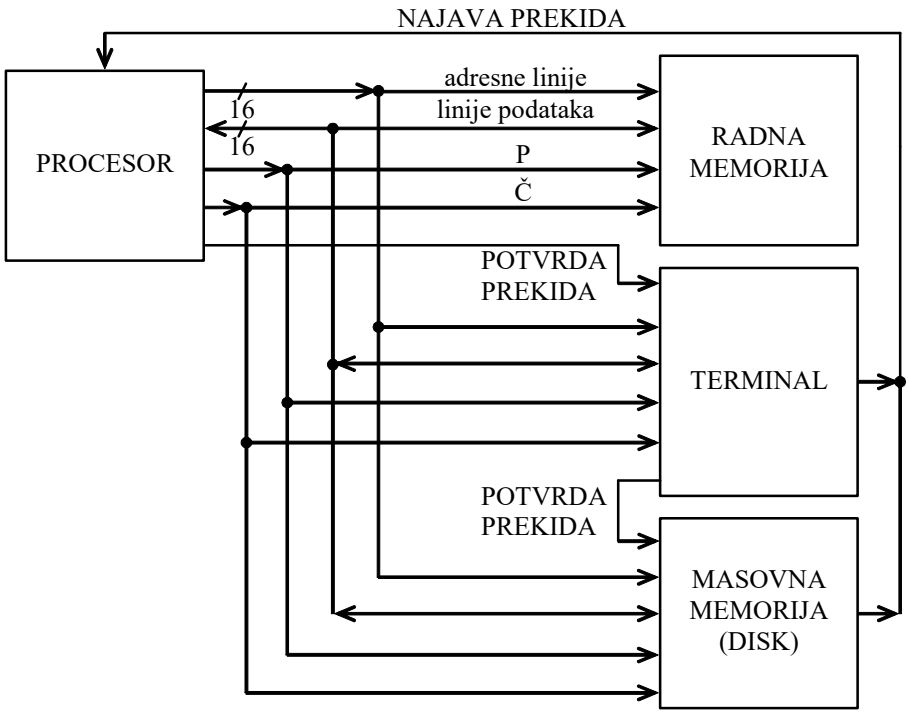
Izvršavanje mikro-programa prekida započinje umesto izvršavanja mikro-programa dobavljanja, ako se prekidač P56 zatvori umesto prekidača P54. Upravljanje prekidačem P56 opisuje funkcija:

R&T&PREKID

Radi sprečavanja istovremenog zatvaranja prekidača P56 i prekidača P54, neophodno je izmeniti upravljanje prekidečem P54. Izmenjeno upravljanje prekidačem P54 opisuje funkcija:

R&T&~IZA_DOBAVLJANJA&~PREKID

Obrade prekida komplikuje činjenica da svaki od kontrolera može istovremeno zatražiti obradu svog prekida. U tom slučaju, redosled obrada prekida zavisi od prioriteta kontrolera koji zahtevaju pomenute obrade. Dodela (statičkih) prioriteta kontrolerima se zasniva na **serijskom povezivanju** (*daisy chaining*) ovih kontrolera linijom potvrde prekida. Na ovaj način, kontroler ima viši prioritet što je bliže procesoru. Najviši prioritet ima kontroler koji je najbliži procesoru, jer do njega prvo stigne potvrda prekida. On tu potvrdu prosleđuje dalje (prema sledećem kontroleru) samo ako nije zahtevao obradu prekida. U suprotnom slučaju, on potvrdu prekida ne prosleđuje dalje, sve dok se njegov prekid ne obradi. Na isti način se, prema svojim sledbenicima, odnose i preostali kontroleri. Slika 5.7.2 prikazuje organizaciju računara KONCEPT koji podržava prekide.



Slika 5.7.2 Organizacija računara KONCEPT koji podržava prekide

Vrednost logičke promenljive **NAJAVA** je uvek jednaka nivou signala u liniji NAJAVA PREKIDA.

Za čuvanje broja vektora u svakom kontroleru je predviđen poseban **registar broja vektora**, čiji izlaz vodi na linije podataka, a stanje čijeg izlaznog prekidača zavisi od nivoa signala na liniji POTVRDA PREKIDA. Prema tome, kada prvi od kontrolera, koji su tražili obradu prekida, primi, preko linije POTVRDA PREKIDA, signal nivoa 1, on automatski po linijama podataka uputi prema procesoru sadržaj svog registra broja vektora. Sadržaj registra broj vektora određuje nivo signala u linijama podataka dok god je u liniji POTVRDA PREKIDA nivo signala 1. Podrazumeva se da kontroleri koji su bliži procesoru, ne ometaju već započetu obradu prekida kontrolera koji su dalje od procesora, nego sačekaju njen kraj, pa tek onda iskoriste svoj privilegovani položaj. Znači, kontroleri reaguju na promenu signala sa nivoa 0 na nivo 1 u liniji POTVRDA PREKIDA.

Mikro-program prekida izgleda:

1. ciklus: programski brojač → %13
2. ciklus: status registar → %14
3. ciklus: 0 → SR_4
4. ciklus: 1 → POTVRDA PREKIDA
   linije podataka → pomoćni registar
5. ciklus: pomoćni registar → adresne linije
   1 → č
   linije podataka → programski brojač

U prvom i drugom ciklusu se sačuvaju zatečeni sadržaji status registra i programskog brojača, a u trećem ciklusu se onemoguće prekidi. Za to je neophodan odgovarajući pristup status registru. U četvrtom ciklusu se preuzme broj vektora. U petom ciklusu preuzeti broj vektora omogući (zahvaljujući tabeli prekida) preuzimanje ulazne adrese obrađivača prekida, koja se smešta u programski brojač. Time se omogući dobavljanje prve naredbe obrađivača prekida i iza toga započne obrada prekida. Podrazumeva se da iza mikro-programa prekida sledi mikro-program dobavljanja.

Naredba NASTAVI je bez operanada, ali se za nju podrazumeva da koristi: programski brojač, status registar i registre %13 i %14. Pošto ona ne pripada ni jednom od postojećih tipova naredbi procesora KONCEPT, ona zahteva sopstveni mikro-program:

1. ciklus: %13 → programski brojač
2. ciklus: %14 → status registar

Izvršavanje ovog mikro-programa omogućuje nastavljanje prekinute aktivnosti procesora i ujedno omogućuje prekide, jer u status registar vraća sadržaj koji je prethodio prekidu, pa je, prema tome, omogućavao prekide. I za ovo je neophodan odgovarajući pristup status registru.

Naredba NASTAVI omogućuje, posredno, izmenu sadržaja status registra (koji je inače nepristupačan na drugi način, jer ne može da se pojavi kao operand bilo koje naredbe). To se postiže, kada se, pre izvršavanja naredbe NASTAVI, izmeni sadržaj registra %14, jer tada izvršavanje ove naredbe smesti u status registar izmenjeni sadržaj registra %14.

ODNOS OBRADE PREKIDA I PREKLJUČIVANJA

Za obradu prekida nije neophodno preključivanje, jer obrada prekida može da se obavi kao deo aktivnosti prekinutog procesa. Ali, obrada prekida može izazvati preključivanje, ako prevede u stanje 'spreman' proces čija aktivnost je hitnija od aktivnosti prekinutog procesa. Radi određivanja procesa čija aktivnost je najhitnija,

uvodi se **prioritet** (*priority*) procesa i podrazumeva se da je aktivnost najprioritetnijeg procesa najhitnija.

Obrađivači prekida obično pripadaju modulu za rukovanje kontrolerima. U nadležnosti ovog modula je i izmena sadržaja tabele prekida. Zahvaljujući tome, obrađivač prekida može da bude i u bilo kom modulu operativnog sistema, pod uslovom da se njegov vektor, posredstvom modula za rukovanje kontrolerima, smesti u odgovarajući element tabele prekida.

Zadatak operativnog sistema je da, na početku rada računara, inicijalizuje tabelu prekida i da zatim inicijalizuje status registar, smeštajući u njegov bit prekida vrednost 1, radi omogućavanja prekida, jer je pretpostavka da su prekidi onemogućeni na početku rada računara.

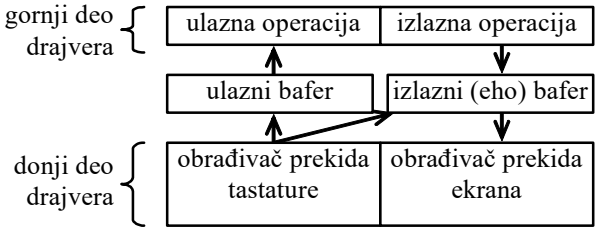
Prekidi izazivaju podelu drajvera na **gornji** i **donji deo**. Između njih se nalaze baferi, namenjeni za razmenu podataka između gornjeg i donjeg dela drajvera. Gornji deo drajvera se sastoji od potprograma koji podržavaju operacije, namenjene slojevima postavljenim iznad sloja sa drajverima. Ovakve operacije omogućuju, na primer, znakovnu komunikaciju između korisnika i računara ili prenos blokova na disk i sa diska. U izvršavanju ovih operacija dolazi do zaustavljanja aktivnosti procesa i do preključivanja procesora, ako nisu ispunjeni uslovi za nastavljanje pomenutih aktivnosti (na primer, ako nisu prispeli očekivani znakovi sa tastature ili ako nije prispeo očekivani blok sa diska). Donji deo drajvera se sastoji od obrađivača prekida, okrenutih kontrolerima koji reaguju na dešavanje vanjskih događaja. Obrade prekida omogućuju nastavljanje prethodno zaustavljenih aktivnosti procesa i dovode do preključivanja procesora na neki od ovih procesa, ako je on prioritetniji od prekinutog procesa. Za obrade prekida je važno da obuhvate samo neodložne stvari, a sve ostalo prepuste procesima. Na taj način se skraćuju periodi u toku kojih su prekidi onemogućeni i tako se skraćuje vreme reakcije na prekide. Procesi koji preuzimaju podatke od obrađivača prekida, radi nastavljaja njihove obrade, se nazivaju **pozadinski** (*background*) **procesi**.

ORGANIZACIJA DRAJVERA TERMINALA

U slučaju drajvera terminala, donji deo drajvera sačinjavaju obrađivač prekida tastature i obrađivač prekida ekrana. Do obrade prekida tastature dolazi čim pristigne znak u registar podataka tastature. U ovoj obradi, prispeli znak se prebacuje u ulazni bafer drajvera terminala. Ulazni bafer omogućuje **unos znakova unapred** (pre nego ih neki proces zatraži), jer u ulaznom baferu prispeli znakovi ostaju sve dok ih ne preuzme neki od procesa, i to posredstvom operacije gornjeg dela drajvera terminala. Ako je ulazni bafer prazan, aktivnost ovih procesa se zaustavlja do njegovog punjenja.

Do obrade prekida ekrana dolazi čim registar podataka ekrana postane spreman za prijem novog znaka (čim je prethodni znak prikazan). U ovoj obradi, registar podataka ekrana se puni znakom iz izlaznog bafera drajvera terminala, jasno, kada on nije prazan. U izlazni bafer znakove smeštaju procesi, opet

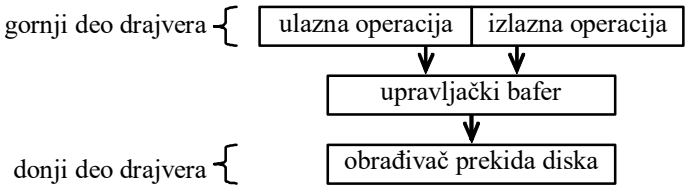
posredstvom potprograma gornjeg dela drajvera terminala. Kada se izlazni bafer napuni, aktivnost ovih procesa se zaustavlja do njegovog pražnjenja. Slika 5.7.3 sadrži prikaz organizacije drajvera terminala.



Slika 5.7.3 Organizacija drajvera terminala

ORGANIZACIJA DRAJVERA DISKA

U slučaju drajvera diska, donji deo drajvera sačinjava obrađivač prekida diska. Do obrade prekida diska dolazi kada kontroler diska završi trenutnu aktivnost i prekidom objavi da je moguć pristup registru podataka diska. U ovoj obradi, obavlja se jedan od koraka u postupku korišćenja diska i to onaj koji je na redu. To znači da prekida (njihovih obrada) ima koliko i ovih koraka: po jedan za saopštavanje broja staze i broja sektora, odnosno po jedan za prenos svakog bajta bloka. Obrađivač prekida diska pronalazi broj staze, broj sektora i adresu zone radne memorije čije lokacije učestvuju u prenosu bajta u **upravljačkom baferu** drajvera diska (koga napune ulazna ili izlazna operacija iz gornjeg dela drajvera diska). Aktivnost procesa, koji je posredstvom operacije iz gornjeg dela drajvera diska zatražio prenos bloka, se nastavlja tek kada se postupak korišćenja diska završi, odnosno, kada se bajti bloka prenesu u željenom smeru. Slika 5.7.4 sadrži prikaz organizacije drajvera diska.



Slika 5.7.4 Organizacija drajvera diska

USKLAĐIVANJE RADA KONTROLERA I UREĐAJA

Prekidi omogućuju istovremeni i nezavisan rad procesora i kontrolera, jer, dok kontroler opslužuje svoj uređaj, procesor je posvećen procesu čija aktivnost ne zavisi od ovog kontrolera. Nakon obavljanja svog zadatka, kontroler prekidom o

tome obaveštava procesor i, eventualno, izaziva njegovo preključivanje. I dok su, tako, aktivnost procesora i kontrolera **asinhrone**, aktivnost kontrolera i uređaja, koga kontroler opslužuje, su **sinhrone**. Za sinhronizaciju kontrolera i pomenutog uređaja koristi se princip **rukovanja** (*handshaking*). Kao što rukovanje nije moguće bez jednog partnera, tako ni saradnja kontrolera i njegovog uređaja nije moguća, ako za nju nisu spremne obe strane. Svoju spremnost za saradnju kontroler označava smeštanjem vrednosti 1 u posebnu logičku promenljivu KONTROLER, koja određuje nivo signala na veznoj liniji, usmerenoj od kontrolera ka uređaju. Na isti način postupa i uređaj sa svojom logičkom promenljivom UREĐAJ, koja određuje nivo signala na veznoj liniji, usmerenoj od uređaja ka kontroleru. I kontroler i uređaj stupaju u međusobnu interakciju samo kada su signali u obe vezne linije na nivou 1. Obaranje signala na nivo 0 u bilo kojoj od ove dve vezne linije, zaustavlja međusobnu interakciju kontrolera i njegovog uređaja.

5.8. SABIRNICA

Suviše veliki broj prekida izaziva značajno trošenje procesorskog vremena (delom na izvršavanje mikro-programa prekida, a delom na obradu prekida). Broj prekida se ne može smanjiti dok god je procesor centralna tačka računara i dok mora da posreduje pri prenosu svih podataka između preostalih delova računara. Znači, tek po stvaranju uslova za direktan kontakt između delova računara, kao što su radna memorija i kontroler diska, postaje moguće značajno smanjenje broja prekida. U ovakvim okolnostima, kontroler diska ne bi morao da prekida procesor, radi prenosa svakog bajta podataka na relaciji radna memorija – registar podataka kontrolera diska. Umesto toga on bi mogao da samostalno obavi prenos svih bajta bloka i da, zatim, prekidom o tome obavesti procesor. To znači da bi umesto 512 prekida bio samo 1 prekid, ako bi blok sadržao 512 bajta.

Za direktan kontakt bilo koja dva dela računara neophodno je postojanje veznih linija koje povezuju sve delove računara. Ovakve vezne linije se nazivaju **sabirnica** ili **magistrala** (*bus*).

Direktan kontakt kontrolera diska i radne memorije se naziva **direktan memorijski pristup** ili skraćeno ***DMA*** (*direct memory access*). Kontroler diska, koji je osposobljen za direktni memorijski pristup, se naziva ***DMA* kontroler**.

Podrazumeva se da i procesor i *DMA* kontroleri pristupaju radnoj memoriji posredstvom sabirnice. U svakom trenutku sabirnicu sme da koristi ili samo procesor ili samo jedan od *DMA* kontrolera (u suprotnom bi došlo do mešanja signala na linijama sabirnice, što bi izazvalo neispravan prenos adresa, podataka i upravljačkih signala). Zato je neophodno obezbediti upravljanje sabirnicom, što se može poveriti procesoru. U tom slučaju, svaki *DMA* kontroler, pre pristupa radnoj memoriji, mora od procesora da zatraži dozvolu za korišćenje sabirnice, na primer tako što će pridružiti vrednost 1 posebnoj logičkoj promenljivoj ZAHTEV (*bus request*). Ova promenljiva određuje nivo signala u istoimenoj veznoj liniji. Procesor dozvoljava korišćenje sabirnice *DMA* kontroleru, kada pridruži vrednost 1 posebnoj

logičkoj promenljivoj DOZVOLA (*bus grant*). Ova promenljiva određuje nivo signala u istoimenoj veznoj liniji.

Direktan memorijski pristup započinje procesor, kada *DMA* kontroleru saopšti:

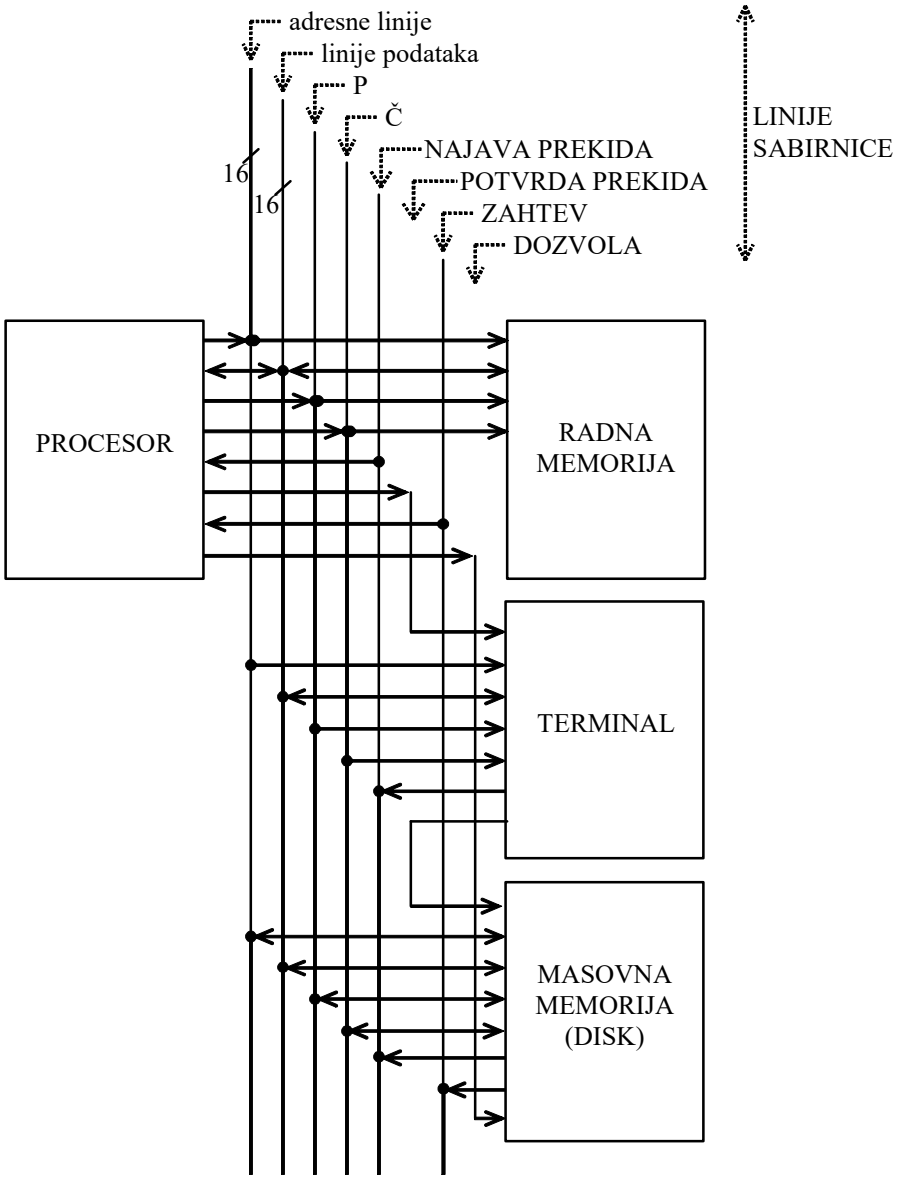
1. broj staze,
2. broj sektora,
3. broj bajta za prenos,
4. adresu prve od lokacija radne memorije koje učestvuju u prenosu i
5. smer prenosa.

Za smeštanje svakog od prva 4 podatka postoji poseban registar u *DMA* kontroleru. To su **registar broja staze**, **registar broja sektora**, **registar broja bajta** i **registar adrese**. Uz to, *DMA* kontroler sadrži još registar stanja i registar podataka. Smer prenosa se saopštava posredstvom registra stanja, a registar podataka učestvuje u prenosu bajta između diska i *DMA* kontrolera, odnosno između *DMA* kontrolera i lokacija radne memorije. Podrazumeva se da se između lokacija radne memorije i registra podataka prenosi svaki od 16 bita, pri čemu opisani *DMA* kontroler koristi samo 8 najmanje značajnih bita.

DMA kontroler traži dozvolu za korišćenje sabirnice, kada postane spreman za prenos bajta na relaciji između lokacije radne memorije i registra podataka. Po dobijanju ove dozvole, on pristupi odgovarajućoj lokaciji radne memorije, radi prenosa pomenutog bajta iz registra podataka ili u njega. Nakon prenosa svakog bajta, *DMA* kontroler oslobađa sabirnicu pridruživanjem vrednosti 0 logičkoj promenljivoj ZAHTEV, što navede procesor da preuzme sabirnicu, pridruživanjem vrednosti 0 logičkoj promenljivoj DOZVOLA. Uz to, *DMA* kontroler umanji za vrednost 1 sadržaj registra broj bajta, da bi otkrio kada su svi bajti preneseni. To se desi kada, nakon umanjenja, ovaj registar sadrži vrednost 0. Tada *DMA* kontroler prekidom obaveštava procesor da je prenos završen. U suprotnom slučaju, on poveća za vrednost 1 sadržaj registra adrese i tako odredi adresu lokacije radne memorije koja učestvuje u narednom prenosu.

Vrlina direktnog memorijskog pristupa je da on smanjuje angažovanje procesora na obradi prekida, jer značajno smanjuje broj prekida. Njegova mana je do on usporava rad procesora kad god mu uzima sabirnicu, jer procesor mora da sačeka da *DMA* kontroler završi korišćenje sabirnice, da bi mogao da pristupi radnoj memoriji. Procesor i *DMA* kontroler ne smetaju jedan drugom samo ako istovremeno nemaju potrebu za korišćenjem sabirnice. To se desi kada procesor pristupa svojim registrima, dok *DMA* kontroler pristupa radnoj memoriji. Procesor može da dozvoli korišćenje sabirnice kad god je u poluciklusu dobavljanja mikro-naredbe, jer mu tada sabirnica nije potrebna. Ali, kada dozvoli *DMA* kontroleru da koristi sabirnicu, procesor ne sme da započne obavljanje mikro-naredbe koja pristupa radnoj memoriji, sve dok mu sabirnica ne postane raspoloživa. To znači da početak obavljanja svake od mikro-naredbi zavisi i od vrednosti logičkih promenljivih DOZVOLA, P i Č.

Slika 5.8.1 prikazuje verziju računara KONCEPT, koji sadrži procesor, radnu memoriju, terminal, masovnu memoriju, sabirnicu i operativni sistem.



Slika 5.8.1 Organizacija računara KONCEPT zasnovanog na sabirnici

Slika 5.8.1 sadrži kvadrate sa natpisima TERMINAL i MASOVNA MEMORIJA koji označavaju i kontrolere i odgovarajuće fizičke uređaje.

U slučaju da postoji više diskova, njihovi *DMA* kontroleri se serijski povezuju linijom DOZVOLA. Ovo se radi iz istih razloga zbog kojih su kontroleri serijski povezani linijom POTVRDA PREKIDA. Prema tome, što je *DMA* kontroler bliži procesoru, on ima viši prioritet u korišćenju sabirnice.

DMA kontroler ima svojstva jednostavnih procesora, jer obavlja aritmetičke operacije i koristi sabirnicu. Dok ovakav kontroler koristi sabirnicu, procesor ne reaguje na prekide.

Vezne linije P, Č, NAJAVA PREKIDA, POTVRDA PREKIDA, ZAHTEV i DOZVOLA spadaju u upravljačke linije sabirnice.

5.9. VIŠEKORISNIČKI RAD

Periodični prekidi omogućuju ravnomernu raspodelu procesorskog vremena između raznih procesa, ako izazivaju **kružno prekljućivanje** (*round robin*) procesora s jednog od ovih procesa na drugi. Periodične prekide izaziva sat. To je digitalni sklop sastavljen od (1) kristalnog oscilatora koji generiše impulse pravilnog perioda i od (2) brojača tih impulsa. Sat generiše prekid kada brojač izbroji zadani broj impulsa. Brojanje prekida (otkucaja) sata je osnova za praćenje proticanja vremena, odnosno za uvođenje pojma **sistemskog vremena** koje se izražava kao broj prekida sata.

Sat je neophodan za **višekorisnički rad**, u toku koga više korisnika istovremeno koristi računar. Pri tome, svaki od njih raspolaže sopstvenim terminalom, posredstvom koga ostvaruje interakciju sa svojim procesima. Zahvaljujući praćenju proticanja sistemskog vremena, moguće je obezbediti periodično preključivanje procesora sa procesa jednog korisnika na proces drugog korisnika i tako stvoriti privid da računar istovremeno opslužuje više korisnika. Ovaj privid se temelji na velikoj brzini procesora.

LOGIČKI I FIZIČKI ADRESNI PROSTORI

Višekorisnički rad stvara mogućnost za nehotično ili namerno međusobno ometanje korisnika. Dovoljno je da proces jednog korisnika neovlašteno izmeni sadržaj memorijskih lokacija koje pripadaju slici procesa drugog korisnika. Zaštita od međusobnog ometanja korisnika se zasniva na sprečavanju procesa da pristupaju memorijskim lokacijama koje ne pripadaju njihovim slikama. To se može ostvariti ako se za svaki proces uvede njegov poseban **logički adresni prostor** i tako razdvoje slike raznih procesa. Slike procesa se i dalje nalaze u običnom ili **fizičkom adresnom prostoru**.

Logički adresni prostor se sastoji od **logičkih adresa** koje se pretvaraju u **fizičke adrese** iz fizičkog adresnog prostora. Pre pretvaranja, proverava se ispravnost logičke adrese, radi detektovanja pokušaja izlaska procesa iz njegovog logičkog adresnog prostora. Podrazumeva se da uvedeni logički adresni prostor obuhvata niz od n memorijskih lokacija sa logičkim adresama od 0 do n-1. Najveća logička adresa se zove **granična adresa**. Ona se menja od procesa do procesa, jer je

veličina logičkog adresnog prostora određena brojem memorijskih lokacija, potrebnih za smeštanje slike procesa. Pojava logičke adrese veće od granične adrese predstavlja pokušaj izlaska van logičkog adresnog prostora procesa (van njegove slike) i tretira se kao greška. Otkrivanje ovakve greške se sastoji od poređenja logičke i granične adrese, a zasniva se na važenju nejednačine:

$$2^n > \Sigma 2^i \quad (n > 0, i = 0, \ldots, n-1)$$

Prethodna nejednačina ukazuje da se poređenje logičke i granične adrese svodi na poređenje korespondentnih bita ovih adresa i to od značajnijih ka manje značajnim bitima. Čim se pronađu različiti korspondentni biti, odmah je jasno da je viša adresa čiji bit sadrži 1. Poređenje bita L_i logičke adrese sa korespondentnim bitom G_i granične adrese opisuju logičke funkcije V_i (više) i N_i (niže) za i = 15, ... , 0:

$$V_i = L_i \& (\sim G_i)$$
$$N_i = (\sim L_i) \& G_i$$

One pokazuju da li je i-ti bit logičke adrese viši ili niži od i-tog bita granične adrese. Uz pomoć prethodnih logičkih funkcija, proveru da li je logička adresa viša od granične adrese opisuje logička funkcija **V**:

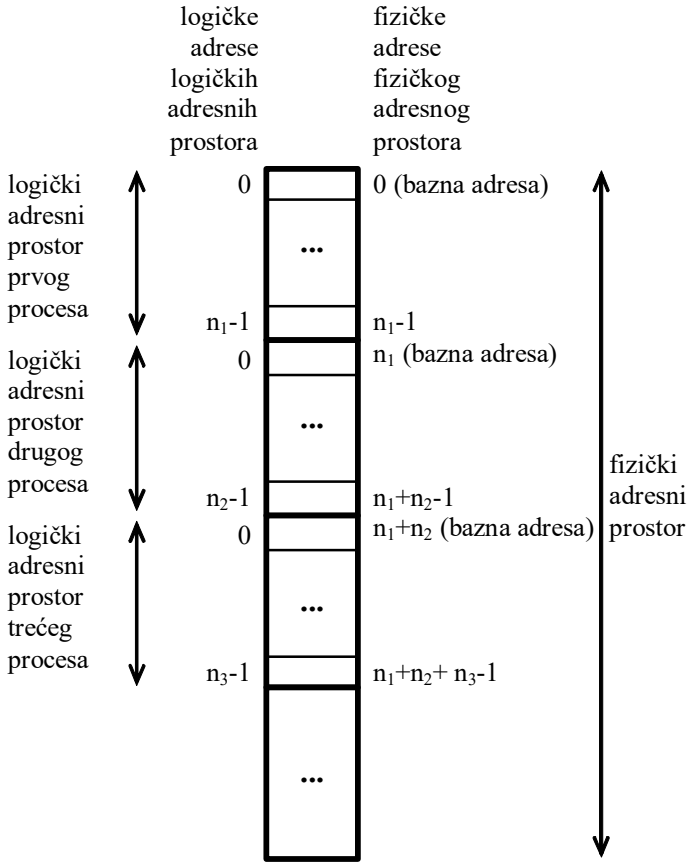
$$V = V_{15} | (\sim N_{15} \& (V_{14} | (\sim N_{14} \& (\ldots (V_1 | (\sim N_1 \& V_0)) \ldots)))))$$

U slučaju greške, kada je logička adresa viša od granične adrese, **V** je tačno i ukazuje na pokušaj izlaska van logičkog adresnog prostora.

PRETVARANJE LOGIČKE ADRESE U FIZIČKU

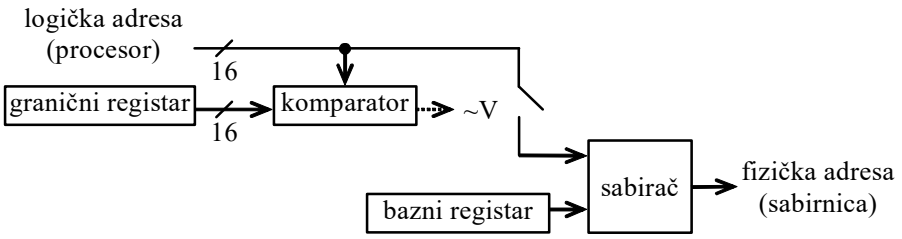
Logičku adresu je moguće pretvoriti u fizičku samo ako je V netačno. Za pretvaranje logičke adrese u fizičku je važno uočiti da se logički adresni prostori raznih procesa mogu istovremeno preslikati u isti fizički adresni prostor tako da se:

1. logičke adrese od 0 do n_1-1 logičkog adresnog prostora prvog procesa preslikaju u fizičke adrese od 0 do n_1-1 fizičkog adresnog prostora,
2. da se logičke adrese od 0 do n_2-1 logičkog adresnog prostora drugog procesa preslikaju u fizičke adrese od n_1 do n_1+n_2-1 fizičkog adresnog prostora i tako dalje (Slika 5.9.1).



Slika 5.9.1 Preslikavanje logičkih adresnih prostora raznih procesa u fizički adresni prostor

Slika 5.9.1 ukazuje da se pretvaranje logičke adrese u fizičku svodi na sabiranje logičke adrese sa **baznom adresom**. Ona odgovara početnoj fizičkoj adresi od koje se logički adresni prostor dotičnog procesa preslikava u fizički adresni prostor. Granična i bazna adresa karakterišu proces, odnosno njegovu sliku i menjaju se od procesa do procesa. Za čuvanje granične i bazne adrese potrebni su posebni registri, koji se nazivaju **granični** (*limit*) i **bazni** (*base*) **registar**. Pošto se sadržaji ovih registara razlikuju od procesa do procesa, oni se menjaju u toku preključivanja, poput sadržaja ostalih procesorskih registara opšte namene. Uz granični i bazni registar, za poređenje logičke i granične adrese potreban je **komparator**, koji implementira logičku funkciju V. Za pretvaranje logičke adrese u fizičku potreban je **sabirač**, koji sabira logičku i baznu adresu. Slika 5.9.2 prikazuje postupak pretvaranja logičke adrese u fizičku.



Slika 5.9.2 Postupak pretvaranja logičke adrese u fizičku

Postupak pretvaranja logičke adrese u fizičku se nalazi u nadležnosti posebnog sklopa koji se označava skraćenicom ***MMU*** (*Memory Management Unit* ili jedinica za upravljanje memorijom). Ovaj sklop može biti u okviru procesora ili van njega. U svakom slučaju, u *MMU* ulazi logička adresa, a iz njega, eventualno, izlazi fizička adresa, koja se upućuje ka sabirnici.

IZUZETAK

Fizička adresa odlazi ka sabirnici samo nakon uspešne provere ispravnosti logičke adrese, znači ako važi ~V. Suprotan slučaj, u kome je provera ispravnosti logičke adrese neuspešna, jer važi V, predstavlja **izuzetak** (*exception*). U slučaju pojave izuzetka, neophodno je prekinuti aktivni proces, jer on zahteva pristupanje nedozvoljenoj memorijskoj lokaciji. Zbog toga su izuzeci podvrsta prekida.

Zadatak *MMU* je da ukaže na izuzetak. Procesor reaguje na izuzetak tako što pokreće **mikro-program izuzetka**, koji se od mikro-programa prekida razlikuje samo po načinu dobavljanja broja vektora. Pošto je izuzetak rezultat aktivnosti procesora, prirodno je da procesor samostalno obezbedi broj vektora, na primer, iz posebnog **registra broja vektora**, koji se nalazi u okviru procesora. Sadržaj ovog registra je predodređen i nepromenljiv. Pomenuti broj vektora indeksira element tabele prekida sa vektorom, odnosno ulaznom adresom **obrađivača izuzetka**. Adresa obrađivača izuzetka se preuzima iz tabele prekida i smešta u programski brojač tokom izvršavanja mikro-programa izuzetka. Zadatak **obrade izuzetka** je da zaustavi nedozvoljenu aktivnost procesa, uz registrovanje mesta njene pojave. Na mesto pojave nedozvoljene aktivnosti procesa ukazuje sadržaj zatečen u programskom brojaču u trenutku izuzetka. Na ovaj način može se otkriti i otkloniti greška koja je izazvala nedozvoljenu aktivnost procesora. Za čuvanje ulazne adrese mikro-programa izuzetka, potrebno je proširiti upravljačku jedinicu procesora KONCEPT posebnim **registrom izuzetka**. Uloga ovoga registra se ne razlikuje od uloge registra prekida. Ipak, prirodno, funkcije njihovih izlaznih prekidača se razlikuju, jer mikro-program izuzetka počinje da se izvršava čim se ustanovi nedozvoljena aktivnost procesa. On, znači, prekida i izvršavanje mikro-programa u toku koga se desi izuzetak. Onemogućavanje izuzetaka nema smisla.

Izuzeci se razlikuju od prekida:

1. po tome što *MMU*, a ne kontroler otkriva pojavu izuzetka,
2. po tome što broj vektora izuzetka pribavlja procesor, a ne kontroler,
3. po tome što obrada izuzetka počinje odmah po njegovom otkrivanju i
4. po tome što izuzeci ne mogu biti onemogućeni.

PRIVILEGOVANI I NEPRIVILEGOVANI REŽIM RADA PROCESORA

Za preuzimanje i izmenu sadržaja graničnog i baznog registra neophodne su posebne naredbe procesora. Ako bi pravo pristupanja ovim registrima imali svi procesi, tada bi oni mogli da svojevoljno preslikavaju svoj logički adresni prostor u fizički i da tako obezvrede zaštitu, odnosno da proveru logičke adrese učine besmislenom. Zato je pravo pristupanja graničnom i baznom registru isključiva privilegija operativnog sistema. Radi uspostavljanja ovakve privilegije, neophodno je razlikovati **privilegovane naredbe** od **neprivilegovanih naredbi** procesora, kao i **privilegovani režim rada** procesora (*kernel*, *superuser mode*) od **neprivilegovanog režima rada** procesora (*user mode*). U privilegovane naredbe procesora spadaju naredbe za preuzimanje i izmenu sadržaja graničnog i baznog registra. Izvršavanje privilegovanih naredbi je dozvoljeno samo u toku privilegovanog režima rada procesora, dok je izvršavanje neprivilegovanih naredbi moguće i u privilegovanom i u neprivilegovanom režimu rada procesora. Privilegovani režim rada procesora je rezervisan samo za izvršavanje operativnog sistema, a neprivilegovani režim rada procesora je predviđen za izvršavanje svih korisničkih programa. Podrazumeva se da se u privilegovanom režimu rada koriste fizičke adrese, jer operativni sistem upravlja celom (fizičkom) radnom memorijom, pa je prirodno i da koristi njene adrese.

Deo fizičkog adresnog prostora, u koji se preslikava logički adresni prostor procesa i kome proces pristupa u neprivilegovanom režimu rada procesora, se naziva **korisnički prostor** (*user space*). Fizički adresni prostor, koga za svoje potrebe koristi operativni sistem u privilegovanom režimu rada procesora, se naziva **sistemski prostor** (*kernel space*).

U procesoru KONCEPT, za označavanje aktuelnog režima rada procesora može da posluži šesti bit status registra (SR_5), koji se naziva **bit privilegije**. Kada je procesor u privilegovanom režimu rada, tada bit privilegije sadrži vrednost 1. U suprotnom slučaju, on sadrži vrednost 0. Da bi se sprečile neovlaštene izmene bita privilegije, naredba `NASTAVI` se, takođe, svrstava u privilegovane naredbe. Pošto ovu naredbu sadrže, kao svoju poslednju naredbu, obrađivači prekida i izuzetaka, pre obrade prekida i izuzetaka neophodno je procesor prevesti u privilegovani režim rada. To zahteva proširenje mikro-programa prekida i izuzetka, radi smeštanja vrednosti 1 u bit privilegije status registra. Za to je neophodan poseban pristup bitu privilegije status registra, u toku koga se njegov sadržaj izmeni, a sadržaj ostalih bita status registra ostane neizmenjen. Zbog korišćenja privilegovane naredbe, obrađivači prekida i izuzetaka ulaze u sastav operativnog sistema.

Pokušaj izvršavanja privilegovanih naredbi u neprivilegovanom režimu rada predstavlja nedozvoljenu aktivnost, na koju, takođe, reaguje mikro-program izuzetka. Ova vrsta izuzetka se otkriva na osnovu koda izvršavane naredbe i sadržaja bita privilegije status registra.

Kao izuzetak se mogu tretirati i pojave kodova nepostojećih naredbi, pa i neispravnih operanada, kao što je, na primer, pojava vrednosti 0 u ulozi delioca za naredbu deljenja kod procesora koji podržavaju ovakvu naredbu. Takođe, kao izuzetak se može tretirati i pojava pogrešnog rezultata, poput izlaska van opsega.

IZVEDBA SISTEMSKIH POZIVA

Neprivilegovani režim rada, rezervisan za izvršavanje korisničkih programa, sprečava da sistemski pozivi budu u obliku običnih poziva potprograma. U ovom slučaju je problematično adresiranje pozivanih operacija operativnog sistema, jer se one nalaze van slike procesa, a to znači i van njegovog korisničkog prostora. Da bi se izbeglo korišćenje nedozvoljenih adresa, operacije operativnog sistema se oblikuju kao obrađivači izuzetaka, a sistemski pozivi se oslanjaju na posebnu naredbu procesora za svesno izazivanje izuzetka (`IZAZOVI`). Na mestu jedinog operanda ovakve naredbe navodi se broj vektora, dodeljenog pozivanoj operaciji operativnog sistema. Izvršavanje ove naredbe se svodi na izvršavanje mikro-programa izuzetka. Prema tome, ovakav sistemski poziv, pored poziva tražene operacije, izaziva i prevođenje procesora iz neprivilegovanog u privilegovani režim rada. Pri tome nema mogućnosti za zloupotrebu prelaska iz neprivilegovanog u privilegovani režim rada procesora, jer je mašinski oblik pozivane operacije van domašaja korisnika. Pošto sistemski pozivi zahtevaju korišćenje specifičnih asemblerskih naredbi, oni se sakrivaju unutar posebnih **sistemskih potprograma**. Ovakvi sistemski potprogrami obrazuju **sistemsku biblioteku**. Zahvaljujući sistemskoj biblioteci, pozivanje operacija operativnog sistema se svodi na pozivanje potprograma iz sistemske biblioteke.

5.10. PITANJA

1. Šta omogućuje upravljačka tabla?
2. Šta omogućuje komandni jezik?
3. Posredstvom kojih registara procesor pristupa kontrolerima?
4. Kada se menja registar stanja kontrolera?
5. Iz kojih registara kontrolera procesor može da čita, a u koje može da piše?
6. Koji standardni znakovni kodovi postoje?
7. Kada se brojevi koriste u znakovnom a kada u internom obliku?
8. Kako se obavlja konverzija brojeva iz internog u znakovni oblik i obrnuto?
9. Kako se interpretiraju komande?
10. Šta je zadatak ulazne/izlazne operacije drajvera?
11. Od čega se sastoji radno čekanje?
12. Šta ulazi u sastav *BIOS*-a?

13. Koliko nivoa korišćenja *BIOS*-a postoji?
14. U kojim tehnologijama se pravi memorija?
15. Koje korake obuhvata prenos bloka diska?
16. Da li je ispravna kodna reč 0010 sa tri bita podataka i jednim bitom parnosti?
17. Koji bit je pogrešan, ako je netačan bit parnosti 1 u kodnoj reči sa četiri bita podataka (sa rednim brojevima 3, 5, 6 i 7) i tri bita parnosti podataka (sa rednim brojevima 1, 2 i 4)?
18. Koje pojmove uvodi operativni sistem?
19. Šta je zadatak operativnog sistema?
20. Šta sadrži slika procesa?
21. Koji deskriptori postoje?
22. Šta omogućuje interpreter komandnog jezika operativnog sistema?
23. Šta obrazuje korisničke interfejse prema operativnom sistemu?
24. Koji sistemski programi postoje?
25. Izvršavanjem kog programa započinje rad računara?
26. Kome korisnik saopštava koji operativni sistem treba da bude pokrenuti na početku rada računara?
27. Šta podrazumeva višeprocesni režim rada?
28. Šta omogućava višeprocesni režim rada?
29. Zašto je uvedeno preključivanje?
30. Šta je zadatak sistemskog procesa?
31. Kada dolazi do ulazom-izlazom vođenog preključivanja?
32. Šta karakteriše ulazom-izlazom vođeno preključivanje?
33. Šta se dešava u toku preključivanja?
34. Koje su mane ulazom-izlazom vođenog preključivanja?
35. Zašto su uvedeni prekidi?
36. Ko izaziva prekide?
37. Šta je vezano za prekide?
38. Šta sadrži tabela prekida?
39. Kojim linijama se prenosi broj vektora prekida od kontrolera ka procesoru?
40. Koje linije su potrebne za podršku prekida?
41. O čemu brine mikro-program prekida?
42. Po čemu se obrađivač prekida razlikuje od običnog potprograma?
43. Kada i kako se onemogućuju prekidi?
44. Čime se proširuje upravljačka jedinica radi podrške prekida?
45. Kada dolazi do obrade prekida?
46. Kako upravljanje prekidima utiče na upravljačku jedinicu?
47. Kojom linijom i zašto se kontroleri serijski povezuju (daisy chaining)?
48. Šta sadrže kontroleri koji podržavaju prekide?
49. Gde se nalazi registar broja vektora?
50. Zašto je uveden prioritet procesa?

51. Koji zadatak imaju pozadinski procesi?
52. Od čega se sastoji drajver terminala koji podržava prekide?
53. Od čega se sastoji drajver diska koji podržava prekide?
54. Šta karakteriše sabirnicu?
55. Koje linije su uvedene radi upravljanja sabirnicom?
56. Šta sadrži *DMA* kontroler?
57. Šta se dešava u toku aktivnosti *DMA* kontrolera?
58. Koje linije serijski povezuju DMA kontrolere?
59. Šta karakteriše višekorisnički rad?
60. Šta je neophodno za višekorisnički rad?
61. Šta sadrži sat?
62. Kako se izražava sistemsko vreme?
63. Šta karakteriše logički adresni prostor?
64. Kako se pretvara logička adresa u fizičku adresu?
65. Šta sadrži MMU?
66. Šta radi MMU?
67. Šta karakteriše izuzetke?
68. Šta uključuje upravljačka jedinica za podršku izuzetaka?
69. Šta je zadatak obrade izuzetaka?
70. Po čemu se razlikuju prekidi i izuzeci?
71. Šta je potrebno za sprečavanje neovlašćenog pristupa graničnom i baznom registru?
72. Koje su privilegovane naredbe?
73. Šta karakteriše privilegovne naredbe?
74. Šta se može izvršavati u privilegovanom režimu rada procesora?
75. Šta se može izvršavati u neprivilegovanom režimu rada procesora?
76. Za šta se koriste izuzeci?
77. Za šta služi naredba IZAZOVI?
78. Na šta se oslanjaju sistemski pozivi?

6. SISTEMSKI PROGRAMI

6.1. EDITOR

Editor je namenjen za zadavanje sadržaja novih tekst datoteka i za izmene sadržaja postojećih tekst datoteka. On koristi ekran terminala za prikazivanje (dela) sadržaja tekst datoteke. Na ekranu je prikazani sadržaj organizovan u linije, iako tekst datoteka može da ima drugačiju unutrašnju organizaciju. Ona, na primer, može da bude organizovana kao niz znakova u kome se nalaze kodovi znakova sadržaja datoteke, kao što su kodovi slova, cifara, posebnih znakova ili upravljačkog znaka "početak linije". U ovako organizovanoj tekst datoteci istoj liniji ekrana pripadaju znakovi datoteke koji se nalaze:

1. između njenog početka i prve pojave pomenutog upravljačkog znaka,
2. između dve uzastopne pojave ovog upravljačkog znaka i
3. između poslednje pojave ovog upravljačkog znaka i kraja tekst datoteke.

Pošto se ekranski prikaz i unutrašnja organizacija tekst datoteke razlikuju, u toku editiranja je neophodno da stanje na ekranu uvek odražava stanje u tekst datoteci.

Aktivnost editora usmerava korisnik pomoću posebnih editorskih komandi. Zadavanje editorskih komandi zahteva korišćenje ili posebnih dirki tastature, ili posebnih kombinacija običnih dirki tastature, ili posebnog pokazivačkog uređaja. Kao komande editora mogu da se koriste i znakovi, ali je tada potrebno razlikovati znakove komandi od znakova sadržaja tekst datoteke. Zato se u ovom slučaju uvode **komandni** i **znakovni režim rada** editora. U komandnom režimu rada, svi znakovi, pristigli sa tastature, predstavljaju znakove komandi. U znakovnom režimu rada, svi preuzeti znakovi postaju deo sadržaja datoteke. Za prelazak iz jednog u drugi režim rada editora potrebna je posebna dirka tastature.

Za ekranski prikaz tekst datoteke, kursor označava znakovnu poziciju ekrana od koje se prikazani sadržaj tekst datoteke menja. Unos svakog znaka izaziva pomeranje kursora u sledeću znakovnu poziciju ekrana, namenjenu za naredni znak, i smeštanje unesenog znaka u prethodnu poziciju kursora. Pri tome, pomeranje kursora izaziva pomeranje svih znakova koji na ekranu dolaze iza njega. Kod unutrašnje organizacije tekst datoteke u obliku niza znakova, kursoru odgovara indeks elementa niza, počev od koga se sadržaj elemenata ovog niza menja. Kod unosa znakova, za svaki novi znak se stvara mesto u nizu pomeranjem sadržaja svih elemenata ovog niza za jedno mesto prema kraju niza, počev od indeksiranog elementa. Zatim se ovaj indeks uveća za vrednost 1. Dok za unutrašnju organizaciju nema razlike između pristizanja upravljačkog znaka "početak linije" i ostalih znakova, za ekranski prikaz, pristizanje pomenutog upravljačkog znaka podrazumeva poseban postupak, jer tada dolazi do pomeranja kursora u prvu znakovnu poziciju naredne linije, uz odgovarajuće pomeranje znakova koji slede iza njega.

Editor se oslanja na drajver terminala koji za vreme aktivnosti editora ne interpretira znakove, ne vrši njihov eho, niti editiranje, nego sve što stigne sa tastature prepušta editoru. Radi podrške editora, drajver terminala, pored običnog, mora da ima i poseban, editorski režim rada.

6.2. ASEMBLER

Asembler je namenjen za analizu asemblerskog programa, sadržanog u programskoj datoteci, radi prepoznavanja pojedinih naredbi i direktiva i radi generisanja njihovih mašinskih oblika. U analizi programske datoteke, asembler preuzima znak po znak njenog sadržaja i proverava da li je preuzeti znak u saglasnosti sa pravilima programskoj jezika koja opisuju obrazovanje asemblerskog programa. Ovakva analiza ima dva nivoa: **leksički** (*lexical*) i **sintaksni** (*syntax*). Cilj **leksičke analize** ili **skeniranja** (*scanning*) je prepoznavanje ispravnih reči u obliku nizova znakova koji ne sadrže separatore, odnosno koji ne sadrže ni razmake, ni upravljačke znakove "početak linije", a obrazovani su po pravilima programskog jezika. Cilj **sintaksne analize** ili **parsiranja** (*parsing*) je prepoznavanje ispravnih rečenica, odnosno redosleda reči, koje su obrazovane po pravilima programskog jezika.

Leksičku i sintaksnu analizu usmeravaju pravila programskog jezika. Tako, na primer, pravilo `program` sugeriše da niz znakova iz programske datoteke treba da započne znakovima `POČETAK`. Iza njih treba da slede razmak i znakovi ulazne labele. Naredno pravilo, `telo`, kao i sledeća pravila, sugerišu da u nastavku treba da sledi upravljački znak "početak linije" i eventualno razmaci, pa ili malo slovo, kojim započinje labela, ili veliko slovo, kojim započinje ime naredbe ili direktive. Nastavak analize zavisi od prepoznatog slučaja i pravila koja se mogu dalje primeniti. Prema tome, asembliranje otpočinje traženjem, od početka programske datoteke, prvog znaka, različitog od separatora. To mora biti znak `P`, a iza njega moraju slediti preostali znakovi reči `POČETAK` i bar jedan separator. Zatim se opet traži znak različit od separatora. To mora biti malo slovo, a iza njega moraju slediti ostali znakovi ulazne labele, kada ih ima (a koji moraju biti ili malo slovo ili cifra ili podvlaka) i bar jedan separator. Zatim se opet traži znak različit od separatora. Ako se pronađe malo slovo, tada je reč o početnom znaku labele, pa se traže dvotačka i separator, a svi pre njih pronađeni znakovi (koji moraju biti ili malo slovo ili cifra ili podvlaka), predstavljaju preostale znakove labele. Zatim se opet traži prvi znak, koji nije separator, a koji mora da bude veliko slovo, jer labela prethodi ili naredbi, ili direktivi. Nastavak asembliranja zavisi od pronađene naredbe ili direktive. U prethodnom opisu analize programske datoteke se prepliću leksička i sintaksna analiza.

Pojava neočekivanog znaka (reči) u bilo kom momentu asembliranja ukazuje na **leksičku** (**sintaksnu**) **grešku** i tada sledi oporavak od ove greške. On se sastoji od pronalaženja prvog znaka od koga se može uspešno nastaviti asembliranje. Oporavak od greške se obično svodi na traženje prve pojave upravljačkog znaka

"početak linije", jer se greška najčešće odnosi na asemblersku naredbu ili direktivu, sadržanu u samo jednoj liniji programskog teksta.

Nakon prepoznavanja naredbe, moguće je generisati njen mašinski oblik. Za generisanje obavezne reči potrebno je raspolagati kodom naredbe. Kodovi naredbi se mogu preuzeti iz **tabele naredbi** (*opcode table*), čiji elementi sadrže imena i kodove naredbi. Slika 6.2.1 sadrži prikaz tabele naredbi za procesor KONCEPT.

Ime naredbe	Heksadecimalni kod naredbe i njena dužina		Ime naredbe	Heksadecimalni kod naredbe i njena dužina	
DESNO	34	1	SKOČI	C0	2
DODAJ_1	30	1	SKOČI_ZA_<	D2	2
I	14	1	SKOČI_ZA_<=	D5	2
ILI	15	1	SKOČI_ZA_!=	D1	2
LEVO	33	1	SKOČI_ZA_==	D0	2
NATRAG	F0	1	SKOČI_ZA_>	D4	2
NE	32	1	SKOČI_ZA_>=	D3	2
ODBIJ_1	31	1	SKOČI_ZA_±_<	D6	2
ODUZMI	12	1	SKOČI_ZA_±_<=	D9	2
ODUZMI_P	13	1	SKOČI_ZA_±_>	D8	2
POZOVI	E0	2	SKOČI_ZA_±_>=	D7	2
PREBACI_DR	60	2	SKOČI_ZA_M	DA	2
PREBACI_IR	80	2	SKOČI_ZA_N	D0	2
PREBACI_NR	50	2	SKOČI_ZA_NE_M	DB	2
PREBACI_PR	70	1	SKOČI_ZA_NE_N	D1	2
PREBACI_RD	90	2	SKOČI_ZA_NE_P	D3	2
PREBACI_RI	B0	2	SKOČI_ZA_NE_V	DD	2
PREBACI_RP	A0	1	SKOČI_ZA_P	D2	2
PREBACI_RR	40	1	SKOČI_ZA_V	DC	2
SABERI	10	1	UPOREDI	20	1
SABERI_P	11	1			

Slika 6.2.1 Tabela naredbi

Tabela naredbi je sortirana po imenima naredbi, radi bržeg, na primer binarnog, pretraživanja.

Kod naredbe, koji je preuzet iz tabele naredbi, se upisuje u značajniji bajt memorijske lokacije koja je rezervisana za generisanje obavezne reči. U preostali bajt se upisuju kodovi registara, koji se koriste u naredbi. Ako operand naredbe zahteva dodatnu reč, odgovarajući sadržaj se upisuje u memorijsku lokaciju koja je rezervisana za generisanje dodatne reči. To ne predstavlja problem, ako se u dodatnu reč smešta heksadecimalni broj koji je naveden u sastavu operanda. Međutim, ako je u sastavu operanda navedena labela, tada određivanje njene adrese nužno prethodi popunjavanju odgovarajuće dodatne reči.

Za labele koje prethode naredbama, kao adresa labele služi adresa početka odgovarajuće naredbe, odnosno adresa memorijske lokacije u kojoj se nalazi obavezna reč ove naredbe. U slučaju da labela prethodi direktivi, kao adresa labele služi adresa početka ove direktive, odnosno adresa prve od memorijskih lokacija koje ova direktiva zauzima za smeštanje podataka. Prema tome, kada se odredi adresa početka naredbe ili direktive, kojoj prethodi labela, ta adresa se pridružuje labeli. Radi čuvanja adresa labela zgodno je uvesti **tabelu labela** (*symbol table*), čiji elementi sadrže labele i njihove adrese. Tabela labela se dopunjava kada se naiđe na **definiciju labele**, odnosno kada se naiđe na liniju u kojoj labela prethodi naredbi ili direktivi. I za tabelu labela je zgodno da bude sortirana po labelama, radi bržeg pretraživanja. Prema tome, kada zatreba adresa labele, dovoljno je pretražiti tabelu labela, radi pronalaženja labele i preuzimanja njene adrese. Međutim, problem se javlja kada se labela koristi pre definisanja, jer tada tabela labela ne sadrži korišćenu labelu, pa se njena adresa ne može preuzeti iz tabele labela. Ovaj problem se javlja kod skokova kod kojih upravljačka naredba prethodi ciljnoj naredbi. Ovakvi skokovi se nazivaju **referenciranje unapred** (*forward references*). Zbog referenciranja unapred, asembliranje se obično organizuje u **dva prolaza**. U prvom prolazu se analizira tekst asemblerskog programa, radi popunjavanja tabele labela. U drugom prolazu se isti tekst analizira još jednom, radi generisanja mašinskih oblika naredbi i direktiva.

U prvom prolazu tabela labela se dopunjava čim se naiđe na definiciju labele. Pošto su za određivanje adresa labela potrebne adrese početaka naredbi ili direktiva kojima labele prethode, pre asembliranja se mora predvideti za koje memorijske lokacije se asemblira asemblerski program, odnosno u koje memorijske lokacije će biti smešten odgovarajući mašinski program. Prirodno je pretpostaviti da se mašinski program smešta od početka logičkog adresnog prostora. Adresi njegovog početka odgovara, znači, nulta logička adresa. Ako se zna adresa početka prve naredbe ili direktive, tada se adrese početaka ostalih naredbi ili direktiva određuju na osnovu činjenice da je adresa početka svake naredbe ili direktive uvek jednaka zbiru adrese početka prethodne naredbe ili direktive i njene dužine. Prema tome, ako se uvede posebna promenljiva **brojač lokacija** (*location counter*) i inicijalizuje na nulu, tada ona sadrži adresu početka prve naredbe ili direktive. Nakon prepoznavanja prve naredbe ili direktive i utvrđivanja njene dužine, sledi uvećavanje vrednost brojača lokacija za tu dužinu, pa on sadrži adresu početka druge naredbe ili direktive. Ako se isti postupak ponovi za svaku naredbu ili direktivu, u brojaču lokacija će se uvek nalaziti adresa početka naredbe ili direktive čija analiza je na redu. Na ovaj način, kada se u toku prepoznavanja naredbe ili direktive otkrije definicija labele, tada njenu adresu sadrži brojač lokacija. Slika 6.2.2 sadrži prikaz uzastopnih vrednosti brojača lokacija pre analize svake naredbe iz asemblerskog programa koji računa najveći zajednički delioc.

Asemblerski program			Brojač lokacija
	POČETAK	ulaz	0
ulaz:	PREBACI_NR	$12,%0	0
	PREBACI_NR	$10,%1	2
ponovo:	UPOREDI	%1,%0	4
	SKOČI_ZA_==	kraj	5
	SKOČI_ZA_<	manje	7
veće:	ODUZMI	%1,%0	9
	SKOČI	ponovo	10
manje:	ODUZMI	%0,%1	12
	SKOČI	ponovo	13
kraj:	SKOČI	kraj	15
	KRAJ		15

Slika 6.2.2 Vrednosti brojača lokacija

Slika 6.2.3 sadrži prikaz tabele labela, nastale u toku prvog prolaza asemblera, za asemblerski program koji računa najveći zajednički delioc (Slika 6.2.2).

Labela	Adresa
kraj	15
manje	12
ponovo	4
ulaz	0
veće	9

Slika 6.2.3 Tabela labela

Mašinski oblik programa, nastao u toku drugog prolaza asemblera, se naziva **objektna sekvenca**. Nju sačinjavaju mašinske naredbe. Slika 6.2.4 sadrži prikaz tabele objektne sekvence, nastale u toku drugog prolaza asemblera, za asemblerski program koji računa najveći zajednički delioc (Slika 6.2.2).

Tabela objektne sekvence				
Adrese lokacija	Objektna sekvenca	Komentar		
			POČETAK	ulaz
0000 0001	5000 000C	ulaz:	PREBACI_NR	$12,%0
0002 0003	5010 000A		PREBACI_NR	$10,%1
0004	2001	ponovo:	UPOREDI	%1,%0
0005 0006	D000 000F		SKOČI_ZA_==	kraj
0007 0008	D200 000C		SKOČI_ZA_<	manje
0009	1201	veće:	ODUZMI	%1,%0
000A 000B	C000 0004		SKOČI	ponovo
000C	1210	manje:	ODUZMI	%0,%1
000D 000E	C000 0004		SKOČI	ponovo
000F 0010	C000 000F	kraj:	SKOČI	kraj
			KRAJ	

(prve dve kolone u tabeli sadrže heksadecimalne brojeve)

Slika 6.2.4 Tabela objektne sekvence

U toku asembliranja se otkrivaju ne samo leksičke ili sintaksne, nego i **semantičke greške**. Otkrivanje semantičkih grešaka je vezano za pretraživanje tabele labela. Na primer:

1. uspešno pretraživanje tabele labela u prvom prolazu asemblera, u toku dopune ove tabele, ukazuje na ponovno definisanje postojeće labele, a
2. neuspešno pretraživanje tabele labela u drugom prolazu asemblera ukazuje na korišćenje nedefinisane labele.

Objektnoj sekvenci odgovara mašinski oblik programa, koji je nespreman za interakciju sa operativnim sistemom. Objektna sekvenca se smešta u objektnu datoteku, koja nije čitljiva kao tekst datoteka, jer njeni svi bajti ne sadrže kodove vidljivih znakova. Pored objektne sekvence, objektna datoteka sadrži i adresu ulazne mašinske naredbe od koje započinje izvršavanje objektne sekvence. Ova ulazna adresa se naziva i **ulazna tačka** (*entry point*) objektne sekvence. Ulazna adresa odgovara ulaznoj labeli (koja prethodi ulaznoj naredbi asembliranog programa).

6.3. MAKRO PRETPROCESOR

Makro pretprocesor je namenjen za analizu programske datoteke, radi prepoznavanja makro definicija, makro poziva i uslovnih direktiva i radi tekstualnih izmena sadržaja ove datoteke. Kada je makro pretprocesor samostalan program, tada on preuzima izvorni tekst asemblerskog programa iz ulazne programske

datoteke, a smešta izmenjeni tekst asemblerskog programa u izlaznu programsku datoteku.

Makro pretprocesiranje se zasniva na leksičkoj i sintaksnoj analizi. Kada se u makro pretprocesiranju prepozna makro definicija, tada se njeno ime smesti u **tabelu makro imena**, a njeno telo se smesti u **tabelu makro tela**. U tabeli makro imena iza svakog imena makroa slede redni broj početnog i redni broj završnog elementa tabele makro tela, u kojima se nalazi telo odgovarajuće makro definicije. Pre smeštanja u tabelu makro tela, telo makro definicije se izmeni tako što se svi parametri zamene njihovim rednim brojevima. Da bi se ovi redni brojevi razlikovali od brojeva prisutnih u telu makro definicije, redni brojevi parametara započinju (posebnim) znakom &. Slika 6.3.1 sadrži prikaz tabele makro imena i tabele makro tela nakon prepoznavanja dve makro definicije.

```
IZBACI          MAKRO       R
                PREBACI_RP  R,(%0)
                DODAJ_1     %0
                KRAJ

UBACI           MAKRO       R
                PREBACI_PR  (%1),R
                DODAJ_1     %1
                KRAJ
```

Tabela makro imena		
IZBACI	1	2
UBACI	3	4

Tabela makro tela		
1	PREBACI_RP	&1,(%0)
2	DODAJ_1	%0
3	PREBACI_PR	(%1),&1
4	DODAJ_1	%1

(druga i treća kolona tabele makro imena sadrže odgovarajuće redne brojeve, navedene u prvoj koloni tabele makro tela)

Slika 6.3.1 Tabela makro imena i tabela makro tela

Nakon prepoznavanja makro poziva, njegovi argumenti se smeste u **tabelu argumenata**, i to tako da se svaki argument nalazi u elementu čiji indeks je jednak rednom broju odgovarajućeg parametra. Na ovaj način se uspostavlja korespondencija parametara i argumenata, koja olakšava zamenu parametara argumentima. Slika 6.3.2 sadrži prikaz tabele argumenata za makro poziv:

```
IZBACI %2
```

redni broj	Argument
1	%2

Slika 6.3.2 Tabela argumenata

Po popunjavanju tabele argumenata, pretražuje se tabela makro imena, radi pronalaženja imena pozivanog makroa. Posredstvom imena se pristupa odgovarajućem telu makro definicije. Zatim se linija sa makro pozivom zameni linijama iz pronađenog tela, u kojima su prethodno redni brojevi parametara zamenjeni argumentima. Pre uvrštavanja linija tela makro definicije u izlaznu programsku datoteku, neophodno je svaku liniju ponovo analizirati, jer linije tela makro definicije mogu sadržati nove makro pozive ili makro definicije. Zbog toga makro pretprocesor sadrži rekurzivne pozive. Rekurzivni pozivi su potrebni i zbog uslovnih direktiva.

Makro pretprocesiranje ili prethodi prvom prolazu asembliranja ili se u njega ugrađuje, ali tako da se asemblerska analiza primenjuje na linije koje su rezultat makro pretprocesiranja.

Za makro pretprocesiranje karakteristične semantičke greške su: poziv nedefinisanog makroa, ponovno definisanje postojećeg makroa i neslaganje broja argumenata makro poziva sa brojem parametara makro definicije. Otkrivanje ovih grešaka se zasniva na korišćenju tabele makro imena, tabele makro tela i tabele argumenata.

6.4. LINKER

Zasebno asembliranje često korišćenih asemblerskih potprograma je dragoceno, jer se tako izbegava ponavljanje asembliranja ovih potprograma zajedno sa asembliranjem programa koji ih pozivaju. Zasebnim asembliranjem programa i potprograma nastaju njihove zasebne objektne datoteke. Zadatak linkera je da zasebne objektne sekvence programa i potprograma, preuzete iz raznih objektnih datoteka, **linkuje** (poveže) u jednu **izvršnu sekvencu** programa, radi stvaranja inicijalne slike procesa i njenog smeštanja u izvršnu datoteku. U postupak linkovanja (povezivanja) linker uključuje i objektne sekvence sistemskih potprograma iz sistemske biblioteke, da bi omogućio interakciju korisničkih programa i operativnog sistema.

PROBLEM RELOKACIJE

Za svaku od zasebnih objektnih sekvenci, koje linker povezuje u jednu izvršnu sekvencu, u toku asembliranja je predviđeno da počinje od nulte logičke adrese. Nakon povezivanja, samo jedna objektna sekvenca počinje od nulte logičke adrese. **Adrese početaka** svih ostalih objektnih sekvenci su **relocirane** (pomerene) za broj memorijskih lokacija koje su zauzele prethodno smeštene objektne sekvence. Ovaj broj memorijskih lokacija predstavlja **konstantu relokacije** za dotičnu objektnu sekvencu. Znači svaka objektna sekvenca ima svoju konstantu

relokacije. **Relokacija** (*relocation*) može da stvori probleme, kada mašinske naredbe iz pomerenih objektnih sekvenci sadrže u svojim dodatnim rečima adrese, jer su te **apsolutne adrese** tačne samo pod pretpostavkom da svaka objektna sekvenca počinje od nulte logičke adrese. Apsolutne adrese objektne sekvence nisu tačne nakon njene relokacije, jer i dalje pokazuju na lokacije koje je asembler predvideo za smeštanje dotične objektne sekvence, a ne na lokacije u koje je objektna sekvenca stvarno dospela nakon linkovanja. Slika 6.4.1 sadrži prikaz objektne sekvence, čije su apsolutne adrese označene strelicom.

Tabela objektne sekvence					
Adrese lokacija	Objektna sekvenca	Apsolutna adresa	Komentar		
				POČETAK	ulaz
0000 0001	5000 000C		ulaz:	PREBACI_NR	$12,%0
0002 0003	5010 000A			PREBACI_NR	$10,%1
0004	2001		ponovo:	UPOREDI	%1,%0
0005 0006	D000 000F	<-		SKOČI_ZA_==	kraj
0007 0008	D200 000C	<-		SKOČI_ZA_<	manje
0009	1201		veće:	ODUZMI	%1,%0
000A 000B	C000 0004	<-		SKOČI	ponovo
000C	1210		manje:	ODUZMI	%0,%1
000D 000E	C000 0004	<-		SKOČI	ponovo
000F 0010	C000 000F	<-	kraj:	SKOČI	kraj
				KRAJ	

(prve dve kolone u tabeli sadrže heksadecimalne brojeve)

Slika 6.4.1 Tabela objektne sekvence sa označenim apsolutnim adresama

Apsolutne adrese se javljaju kada se u dodatnoj reči naredbe nalazi adresa koja odgovara labeli. To se desi za operande upravljačkih naredbi, kao i za neposredni, direktni i indeksni operand naredbi prebacivanja.

Ako se objektna sekvenca (Slika 6.4.1) relocira, na primer, za jednu lokaciju, tada sve apsolutne adrese postaju netačne. Ali ako se, nakon relokacije, svaka od prethodnih apsolutnih adresa uveća za vrednost konstante relokacije, koja u ovom primeru ima vrednost 1, apsolutne adrese opet postaju tačne. Znači, **problem relokacije** (*relocation problem*) linker može da reši korekcijom apsolutnih adresa, odnosno uvećavanjem apsolutnih adresa za konstantu relokacije. Ovakva korekcija apsolutnih adresa se naziva **statička relokacija** (*static relocation*). Da bi linker znao gde se u objektnoj sekvenci nalaze apsolutne adrese, asembler mora da generiše **tabelu relokacije** (*relocation dictionary*) sa logičkim adresama lokacija

objektne sekvence, koje sadrže apsolutne adrese. Tabela relokacije se nalazi u objektnoj datoteci zajedno sa objektnom sekvencom. Slika 6.4.2 prikazuje tabelu relokacije za primer objektne sekvence koju sadrži Slika 6.4.1.

Heksadecimalne adrese
0006
0008
000B
000E
0010

Slika 6.4.2 Tabela relokacije

RELATIVNO ADRESIRANJE

Problem relokacije ne postoji, ako ne postoje apsolutne adrese. To se može postići, na primer, kod upravljačkih naredbi, ako se u njihovoj dodatnoj reči navede međusobna udaljenost upravljačke i ciljne naredbe, umesto apsolutne adrese ciljne naredbe. Međusobna udaljenost upravljačke i ciljne naredbe se izražava kao razlika nastala oduzimanjem adrese dodatne reči upravljačke naredbe od adrese obavezne reči ciljne naredbe. Ova udaljenost predstavlja **relativnu adresu** ciljne naredbe u odnosu na upravljačku naredbu. Iz relativne adrese se može odrediti apsolutna adresa ciljne naredbe, kada se relativna adresa sabere sa sadržajem programskog brojača koga programski brojač ima prilikom dobavljanja relativne adrese iz dodatne reči. Ovakvo adresiranje se naziva **relativno adresiranje** (*PC-relative addressing*). Relativno adresiranje predstavlja oblik indeksnog adresiranja, u kome relativna adresa predstavlja indeks i u kome se podrazumeva korišćenje programskog brojača umesto registra opšte namene. Nakon uvođenja relativnog adresiranja, podrazumeva se da se ono koristi u upravljačkim naredbama. To ima uticaja na njihove mikro-programe, pa tako, na primer, mikro-program obavljanja sa relativnim adresiranjem za 12. tip naredbi (`SKOČI`) izgleda:

1. ciklus: programski brojač → adresne linije (P2)
   1 → č (P41)
   linije podataka → pomoćni registar (P3)
2. ciklus programski brojač → registar 1. podatka (P2, P37, P42)
3. ciklus pomoćni registar → registar 2. podatka (P4, P37, P43)
4. ciklus: saberi (P52)
   linije podataka → programski brojač (P1)

Slika 6.4.3 prikazuje primer objektne sekvence u kojoj se podrazumeva korišćenje relativnog adresiranja za upravljačke naredbe.

Tabela objektne sekvence					
Adrese lokacija	Objektna sekvenca	Relativna adresa	Komentar		
				POČETAK	ulaz
0000 0001	5000 000C		ulaz:	PREBACI_NR	$12,%0
0002 0003	5010 000A			PREBACI_NR	$10,%1
0004	2001		ponovo:	UPOREDI	%1,%0
0005 0006	D000 0009	<-		SKOČI_ZA_==	kraj
0007 0008	D200 0004	<-		SKOČI_ZA_<	manje
0009	1201		veće:	ODUZMI	%1,%0
000A 000B	C000 FFF9	<-		SKOČI	ponovo
000C	1210		manje:	ODUZMI	%0,%1
000D 000E	C000 FFF6	<-		SKOČI	ponovo
000F 0010	C000 FFFF	<-	kraj:	SKOČI	kraj
				KRAJ	

(prve dve kolone u tabeli sadrže heksadecimalne brojeve)

Slika 6.4.3 Tabela objektne sekvence sa relativnim adresama, koje su označene strelicama

Relativne adrese su označeni celi brojevi u komplement 2 predstavi. To znači, kada ciljna naredba sledi posle upravljačke naredbe, relativna adresa je pozitivna. Na primer, relativne adrese u lokacijama sa logičkim adresama 6_{16} i 8_{16} (Slika 6.4.3) su pozitivne. Kada ciljna naredba prethodi upravljačkoj naredbi, relativna adresa je negativna. Na primer, relativne adrese u lokacijama sa logičkim adresama B_{16}, E_{16} i 10_{16} (Slika 6.4.3) su negativne.

Na relativne adrese ne utiče relokacija objektne sekvence, jer se relokacijom ne menja relativni položaj mašinskih naredbi u objektnoj sekvenci. Zato relativne adrese ne izazivaju problem relokacije. Problem relokacije ne izazivaju ni apsolutne adrese iz objektnih sekvenci koje nisu relocirane.

PROBLEM SPOLJAŠNJIH REFERENCI

Zasebna asembliranja izazivaju ne samo problem relokacije, nego i problem **spoljašnjih referenci** (*external reference problem*). Spoljašnje reference se javljaju kada se iz programa ili potprograma poziva zasebno asembliran potprogram, jer se tada **referencira** (koristi) **spoljašnja labela** koja nije definisana u programu ili potprogramu pozivaocu, nego u pozivanom potprogramu. Takve spoljašnje labele asembler označava kao nedefinisane labele. Pošto adrese spoljašnjih labela nisu poznate u toku asembliranja, kao njihova adresa koristi se 0. Da bi linker mogao da koriguje ove adrese, asembler formira **tabelu nedefinisanih labela** (*external*

reference table). Njeni elementi sadrže nedefinisane (odnosno spoljašnje) labele i logičke adrese lokacija koje treba korigovati pomoću adresa ovih spoljašnjih labela. I ova tabela ulazi u sastav objektne datoteke. Pomoću tabele nedefinisanih labela, linker može da utvrdi koje lokacije iz objektne sekvence treba korigovati, kao i adrese kojih spoljašnjih labela treba koristiti za ovu korekciju.

Ulogu spoljašnjih labela imaju ulazne labele. Da bi linker mogao da reši problem spoljašnjih referenci, asembler mu posredstvom objektne datoteke prosleđuje **tabelu ulaznih labela** (*entry point table*). Njen element sadrži ulaznu labelu i njenu adresu.

Slika 6.4.4 sadrži objektnu sekvencu programa koji poziva nezavisno asemblirani potprogram `nzd`, radi određivanja najvećeg zajedničkog delioca brojeva 12 i 10. Argumenti se prosleđuju pozivanom potprogramu posredstvom registara `%1` i `%2`. Da bi bio jednostavniji, ovaj program ne koristi povratnu vrednost potprograma `nzd` i završava se beskonačnom petljom. Uz objektnu sekvencu ovog programa su navedene tabela relokacije, tabela nedefinisanih labela i tabela ulaznih labela.

Tabela objektne sekvence			
Adrese lokacija	Objektna sekvenca	Nedefinisana adresa	Komentar
			`POČETAK primer`
`0000` `0001`	`5010` `000C`		`primer: PREBACI_NR $12,%1`
`0002` `0003`	`5020` `0OOA`		`PREBACI_NR $10,%2`
`0004` `0005`	`E0F0` `0000`	`<-`	`POZOVI nzd`
`0006` `0007`	`C000` `FFFE`		`kraj: SKOČI kraj`
			`KRAJ`

Tabela relokacije
-

Tabela nedefinisanih labela	
`nzd`	`0005`

Tabela ulaznih labela	
`primer`	`0000`

(prve dve kolone u tabeli objektne sekvence, kolona u tabeli relokacije i druga kolona u tabelama nedefinisanih i ulaznih labela sadrže heksadecimalne brojeve)

Slika 6.4.4 Tabela objektne sekvence i tabele relokacije, nedefinisanih i ulaznih labela programa

U objektnoj sekvenci koju sadrži Slika 6.4.4 potrebno je korigovati lokaciju sa logičkom adresom 5_{16}, na koju pokazuje strelica. Ova lokacija treba da sadrži relativnu adresu koja se određuje na osnovu adrese spoljašnje labele `nzd`.

Slika 6.4.5 sadrži objektnu sekvencu potprograma `nzd` koji određuje najveći zajednički delioc neoznačenih celih brojeva navedenih u registrima `%1` i `%2`. Najveći zajednički delioc predstavlja povratnu vrednost i prosleđuje se posredstvom registra `%0`. Uz objektnu sekvencu su navedene tabela relokacije, tabela nedefinisanih labela i tabela ulaznih labela.

Tabela objektne sekvence				
Adrese lokacija	Objektna sekvenca	Komentar		
			POČETAK	nzd
0000	2012	nzd:	UPOREDI	%2,%1
0001 0002	D000 0009		SKOČI_ZA_==	kraj
0003 0004	D200 0004		SKOČI_ZA_<	manje
0005	1212	veće:	ODUZMI	%2,%1
0006 0007	C000 FFF9		SKOČI	nzd
0008	1221	manje:	ODUZMI	%1,%2
0009 000A	C000 FFF6		SKOČI	nzd
000B	4001	kraj:	PREBACI_RR	%1,%0
000C	F0F0		NATRAG	
			KRAJ	

Tabela relokacije
-

Tabela nedefinisanih labela	
-	-

Tabela ulaznih labela	
nzd	0000

(prve dve kolone u tabeli objektne sekvence, kolona u tabeli relokacije i druga kolona u tabelama nedefinisanih i ulaznih labela sadrže heksadecimalne brojeve)

Slika 6.4.5 Tabela objektne sekvence i tabele relokacije, nedefinisanih i ulaznih labela potprograma

OBRAZOVANJE IZVRŠNE SEKVENCE

U toku rada linker preuzima objektne sekvence i sve tabele iz objektnih datoteka koje su obuhvaćene linkovanjem. On tada odredi adrese početaka

objektnih sekvenci na osnovu njihovih dužina i formira **tabelu objektnih sekvenci** (*object module table*). U ovoj tabeli se nalaze ulazne labele objektnih sekvenci, dužine objektnih sekveci i adrese njihovih početaka. Slika 6.4.6 sadrži tabelu objektnih sekvenci za program koga sadrži Slika 6.4.4 i potprogram koga sadrži Slika 6.4.5.

Ulazna labela objektne sekvence	Dužina objektne sekvence	Adresa početka objektne sekvence
`primer`	`8`	`0000`
`nzd`	`13`	`0008`

(druga kolona sadrži decimalne, a treća heksadecimalne brojeve)

Slika 6.4.6 Tabela objektnih sekvenci

Adresa početka objektne sekvence je ujedno njena konstanta relokacije, pa se pomoću nje može izvršiti relokacija svih adresa iz njenih tabela. Nakon relokacije, nisu se promenile tabele nedefinisanih i ulaznih labela programa (Slika 6.4.4), jer je konstanta relokacije programa jednaka nuli, dok se adresa labele `nzd` iz tabele ulaznih labela potprograma (Slika 6.4.5) uvećala za vrednost 8, jer je tolika konstanta relokacije potprograma.

Pomoću odgovarajućih konstanti relokacije se, zatim, vrši relokacija apsolutnih adresa pojedinih objektnih sekvenci, radi rešavanja problema relokacije. Da bi se rešio problem spoljašnjih referenci, sve tabele ulaznih labela se spajaju u jednu **tabelu spoljašnjih labela** (*global symbol table*). Slika 6.4.7 sadrži tabelu spoljašnjih labela za program koga sadrži Slika 6.4.4 i potprogram koga sadrži Slika 6.4.5.

Tabela spoljašnjih labela	
`primer`	`0000`
`nzd`	`0008`

(druga kolona u tabeli sadrži heksadecimalne brojeve)

Slika 6.4.7 Tabela spoljašnjih labela

Za svaku od labela iz tabela nedefinisanih labela se pronalazi njena adresa u tabeli spoljašnjih labela. Ova adresa se koristi za korekciju lokacija odgovarajuće objektne sekvence. Ako neka od tabela nedefinisanih labela sadrži labelu koja ne postoji u tabeli spoljašnjih labela, reč je o nedefinisanoj labeli, odnosno o grešci koja se otkriva u toku linkovanja.

Nakon rešavanja problema relokacije i problema spoljašnjih referenci, objektne sekvence se mogu spojiti u jednu izvršnu sekvencu. Slika 6.4.8 sadrži

primer izvršne sekvence za program koga sadrži Slika 6.4.4 i potprogram koga sadrži Slika 6.4.5.

Tabela izvršne sekvence				
Adrese lokacija	Izvršna sekvenca	Komentar		
0000 0001	5010 000C	primer:	PREBACI_NR	$12,%1
0002 0003	5020 0OOA		PREBACI_NR	$10,%2
0004 0005	E0F0 0003		POZOVI	nzd
0006 0007	C000 FFFE	kraj:	SKOČI	kraj
0008	2012	nzd:	UPOREDI	%2,%1
0009 000A	D000 0009		SKOČI_ZA_==	kraj
000B 000C	D200 0004		SKOČI_ZA_<	manje
000D	1212	veće:	ODUZMI	%2,%1
000E 000F	C000 FFF9		SKOČI	nzd
0010	1221	manje:	ODUZMI	%1,%2
0011 0012	C000 FFF6		SKOČI	nzd
0013	4001	kraj:	PREBACI_RR	%1,%0
0014	F0F0		NATRAG	

(prve dve kolone u tabeli sadrže heksadecimalne brojeve)

Slika 6.4.8 Tabela izvršne sekvence

Lokacija sa logičkom adresom 5 izvršne sekvence (Slika 6.4.8) sadrži relativnu adresu 3, koja je određena kao vrednost izraza:

8-5

Adresa 8 odgovara obaveznoj reči ciljne naredbe, a adresa 5 odgovara dodatnoj reči upravljačke naredbe.

Rad linkera se obično obavlja u dva prolaza. Prvi prolaz je namenjen za formiranje tabele objektnih sekvenci, za relokaciju tabela i za formiranje tabele spoljašnjih labela. Drugi prolaz je posvećen rešavanju problema relokacije i problema spoljašnjih referenci i stvaranju izvršne sekvence. Redosled objektnih sekvenci u izvršnoj sekvenci zavisi od redosleda u kome se navode objektne datoteke u komandi kojom se pokreće linker. Ulazna adresa izvršne sekvence je jednaka ulaznoj adresi njene prve objektne sekvence.

6.5. LOUDER

Izvršavanju programa obavezno prethodi njegovo punjenje, koje, između ostalog, obuhvata prepisivanje njegove izvršne sekvence iz izvršne datoteke u radnu memoriju, radi formiranja slike procesa. To obavlja louder. Pre izvršavanja programa neophodno je pripremiti sadržaje graničnog i baznog registra, radi pretvaranja logičkih adresa u fizičke. Postupak pretvaranja logičkih adresa u fizičke se naziva i **dinamička relokacija** (*dynamic relocation*), jer sadržaj baznog registra predstavlja konstantu relokacije, koja se u toku izvršavanja dinamički dodaje na logičke adrese da bi se formirale fizičke adrese. Louder se javlja kao zaseban program, kada se, na primer, izvršni programi prebacuju sa razvojnog računara na ciljni računar, koji nema sistemske programe.

6.6. DIBAGER

Dibager je namenjen za omogućavanje nadgledanja izvršavanja programa, radi praćenja izmena sadržaja lokacija (promenljivih) i otkrivanja šta izaziva pojavu neočekivanih (pogrešnih) sadržaja u njima. Bez dibagera, praćenje izvršavanja programa se svodi na prikazivanje sadržaja promenljivih koje su odabrane unapred, pre izvršavanja programa. Sa dibagerom je moguće prikazivati sadržaje bilo koje promenljive, odabrane u toku izvršavanja programa.

Dibagiranje se zasniva na mogućnosti prekidanja izvršavanja programa ili pre izvršavanja svake, ili samo pre izvršavanja određene, dinamički odabrane asemblerske naredbe, i to radi aktiviranja dibagera. Po aktiviranju, dibager omogućuje pregledanje zatečenih sadržaja memorijskih lokacija, registara opšte namene i nekih registara posebne namene, na primer, programskog brojača ili status registra.

Dibagiranje se oslanja na poseban **koračni režim rada** procesora (*single step*). Procesor se nalazi u ovom režimu rada, kada poseban **bit traga** (*trace bit*) status registra (na primer, bit SR_6 kod procesora KONCEPT) sadrži vrednost 1. Ovaj bit se zove bit traga, jer omogućuje praćenje traga izvršavanja programa.

U koračnom režimu rada procesora se podrazumeva da dobavljanju svake naredbe prethodi izvršavanje mikro-programa izuzetka, radi aktiviranja dibagera koji ima oblik obrađivača izuzetka. Na taj način, koračni režim rada omogućuje nadgledanje izvršavanja svake naredbe, kao i praćenje izmena sadržaja interesantnih lokacija.

Za dibagiranje je zgodno da postoji i posebna **naredba zamke** (*trap*), čije izvršavanje se svodi na izvršavanje mikro-programa izuzetka. Izvršavanje ove naredbe dovodi do aktiviranja dibagera i bez koračnog režima rada procesora. Ovakva naredba ubrzava izvršavanje dibagiranog programa, jer omogućuje da aktiviranje dibagera prethodi izvršavanju samo pojedinih, a ne svih naredbi. Dibager se aktivira pre izvršavanja odabrane naredbe, ako se, na primer, njen mašinski oblik privremeno, u toku dibagiranja, zameni mašinskim oblikom naredbe

zamke. Podrazumeva se da, odmah po svom aktiviranju, dibager umesto mašinskog oblika naredbe zamke vrati prethodno sačuvani (zamenjeni) mašinski oblik odabrane naredbe. Umesto privremenih izmena izvršnog oblika programa, dibager može da smesti adresu odabrane naredbe u neki od posebnih **dibagerskih registara**. Tada do aktiviranja dibagera dolazi kada sadržaj programskog brojača postane jednak sadržaju jednog od dibagerskih registara.

Nadgledanje izvršavanja programa usmerava korisnik pomoću (znakovnih) komandi, upućenih dibageru. Ove komande prevode procesor u koračni režim rada, postavljaju zamku na određeno mesto u programu, prikazuju (uz odgovarajuće interpretiranje) sadržaj lokacija, ali i završavaju izvršavanje programa.

6.7. PITANJA

1. Šta je namena editora?
2. Šta karakteriše editor?
3. Šta je namena asemblera?
4. Koje poslove obavlja asembler?
5. Šta sadrži tabela naredbi?
6. Kada nastane tabela naredbi?
7. Šta sadrži tabela labela?
8. Kada nastane tabela labela?
9. Šta sadrži brojač lokacija?
10. Šta sadrži tabela labela (objasniti na primeru)?
11. Koja adresa odgovara labeli `x` za asemblerske naredbe:
```
   SABERI %0,%1
   NE     %0
x: LEVO   %0
```
   čije mašinske naredbe počinju od adrese 0?
12. Koje greške se javljaju u asembliranju?
13. Šta je namena makro pretprocesora?
14. Koje tabele formira makro pretprocesor?
15. Šta je namena linkera?
16. Koje probleme rešava linker?
17. Šta izaziva pojavu problema relokacije?
18. Šta karakteriše rešavanje problema relokacije?
19. Šta izaziva pojavu problema spoljašnjih referenci?
20. U mašinskim oblicima kojih naredbi se javljaju apsolutne adrese?
21. Koje tabele su potrebne za rešavanje problema spoljašnjih labela?
22. Koliko apsolutnih adresa imaju mašinske naredbe koje odgovaraju asemblerskim naredbama:
```
   SABERI %0,%1
   NE     %0
x: LEVO   %0
```
   (podrazumeva se da ne postoji relativno adresiranje)?
23. Šta karakteriše relativno adresiranje?

24. Koje tabele formira asembler?
25. Koje tabele formira linker?
26. Ko obavlja statičku relokaciju?
27. Ko obavlja dinamičku relokaciju?
28. Šta radi linker u 1. prolazu?
29. Šta radi linker u 2. prolazu?
30. Šta radi asembler u 1. prolazu?
31. Šta radi asembler u 2. prolazu?
32. Koji sistemski programi imaju 1 prolaz?
33. Koji sistemski programi imaju 2 prolaza?
34. Šta je namena loudera?
35. Šta je namena dibagera?
36. Šta je potrebno za rad dibagera?
37. Šta sadrži status registar?

7. EVOLUCIJA ARHITEKTURE RAČUNARA

7.1. PRECIZIRANJE POJMA ARHITEKTURE RAČUNARA

Arhitektura računara se bavi arhitekturom naredbi i organizacijom i izvedbom računara.

Zadatak arhitekture naredbi je da potpuno opiše procesor sa stanovišta korišćenja. Ona opisuje:

1. skup naredbi,
2. vrste operanada,
3. adresiranja,
4. adresni prostor,
5. memorijske lokacije i
6. registre računara.

Skup naredbi sadrži:

1. naredbe prenosa podataka,
2. naredbe za rukovanje bitima,
3. aritmetičke naredbe,
4. upravljačke naredbe,
5. sistemske naredbe (kao što je, na primer, naredba sistemskog poziva) i
6. ulazno-izlazne naredbe.

Vrste operanada određuju predstave vrednosti podržanih tipova podataka, kao što su:

1. celi tip,
2. realni tip (*floating-point*),
3. znakovni tip i
4. logički tip.

U okviru adresiranja opisuju se načini pristupanja operandima, koji su:

1. ili neposredno deo naredbe,
2. ili se nalaze direktno u registrima i memorijskim lokacijama,
3. ili im se pristupa posredstvom adresa, smeštenih u registre i memorijske lokacije, uz eventualno automatsko modifikovanje pomenutih adresa.

Za adresni prostor su bitne vrste i raspon adresa memorijskih lokacija, za memorijske lokacije je bitna njihova veličina, a za registre su bitni njihov broj, veličina i namena.

Organizacija računara se bavi principijelnim ostvarenjem funkcionalnosti računara, odnosno utvrđivanjem vrsta organizacionih komponenti računara, njihovih osobina i načina njihovog kombinovanja. Kao organizacione komponente računara javljaju se procesor, radna memorija, masovna memorija, kontroleri, ulazni i izlazni uređaji, sabirnica, ali i operativni sistem ili sistemski programi, poput kompajlera.

Izvedba računara obuhvata rešavanje tehnoloških i proizvodnih problema, kao što su, na primer, problemi projektovanja i proizvodnje logičkih kola, ali i problemi projektovanja i proizvodnje sistema za napajanje ili sistema za rashlađivanje.

7.2. POKRETAČI RAZVOJA ARHITEKTURE RAČUNARA

Arhitektura računara uvek odražava njegovu namenu, ali tako da upotrebljena tehnologija bude iskorišćena na najcelishodniji način. Zbog toga je, prilikom oblikovanja arhitekture računara, izbor tehničkih rešenja uvek podređen optimizaciji, čiji cilj je ostvarenje najpovoljnijeg odnosa funkcionalnosti računara i njegovih proizvodnih troškova (koji predstavljaju približno dve trećine njegove maloprodajne cene).

Proizvodni troškovi računara se sastoje od **direktnih troškova**, nastalih u toku proizvodnih aktivnosti preduzeća i od **indirektnih troškova**, nastalih u toku preostalih (neproizvodnih) aktivnosti preduzeća. Direktni troškovi su vezani za pravljenje svakog računara i obuhvataju troškove komponenti, troškove rada, troškove garancije i slično. U indirektne troškove spadaju troškovi istraživanja i razvoja, troškovi marketinga, troškovi prodaje, troškovi održavanja proizvodnih pogona, ali i porezi, dobit, kamate na kredite i slično. Iako indirektni troškovi nisu direktno vezani za pravljenje pojedinih računara, oni opterećuju cenu svakog računara, jer se ravnomerno raspoređuju po svakom proizvedenom računaru.

Snižavanje cene računara je vezano za smanjenje proizvodnih troškova. Obaranje direktnih proizvodnih troškova je posledica tehnološkog napretka koji izazove snižavanje cene komponenti ili omogući viši stepen automatizacije proizvodnje i tako dovede do smanjenja troškova rada. Niža cena računara stvara uslove za njegovu primenu u novim oblastima, a tada, zbog širenja tržišta, dolazi do povećanja proizvodnje računara. Značajnije povećanje proizvodnje ima dvostruke posledice. S jedne strane, ono uzrokuje dodatno snižavanje cene, jer se indirektni troškovi raspodeljuju na veći broj proizvedenih računara, pa manje opterećuju cenu svakog od njih. Sa druge strane, ono obezbeđuje dodatna sredstva za istraživanje i razvoj, jer smanjenje udela indirektnih troškova u ceni pojedinog računara ne mora biti proporcionalno povećanju proizvodnje. Pomenuto dodatno snižavanje cene računara i tehnološki napredak, izazvan intenziviranjem istraživanja i razvoja, dovode do novog ciklusa širenja tržišta računara sa već opisanim posledicama. Opisana **pozitivna povratna sprega** između snižavanja cena i proširenja tržišta računara predstavlja snažan zamajac razvoja.

Pojava prvog elektronskog računara opšte namene, podesnog ne samo za numeričke, nego i za poslovne obrade, predstavlja polaznu tačku prethodno opisane razvojne povratne sprege, zasnovane na brzom napredovanju poluprovodničkih tehnologija. Ova povratna sprega je uzrokovala buran razvoj računara, odnosno dovela je do brze evolucije arhitekture računara. U ovoj knjizi se, zato, pojam računar odnosi na elektronski računar opšte namene, a evolucija arhitekture

računara se prati od pojave prvog elektronskog računara opšte namene, proizvedenog za tržište.

7.3. PERIODI EVOLUCIJE

Razvoj arhitekture računara je prolazio kroz više evolucionih faza. One se vezuju za periode dominacije pojedinih poluprovodničkih tehnologija, čiji razvoj je izazvao evoluciju arhitekture računara. Evolucione faze približno obuhvataju prelaze između decenija, računajući od pojave prvog elektronskog računara opšte namene, proizvedenog za tržište. One predstavljaju osnovu za podelu računara u **generacije**, pri čemu se smatra da računari, proizvedeni u okviru jedne evolucione faze, pripadaju istoj generaciji. Podela računara u generacije je uslovna, jer su u svakoj od evolucionih faza postojala preklapanja između dominantne (zrele) i nastupajuće poluprovodničke tehnologije (čiji vrhunac je tek nailazio). Prema tome, u okviru istih generacija računara mešali su se uticaji raznih poluprovodničkih tehnologija. Pored toga, usavršavanje svake od ovih tehnologija se na sličan način odražavalo na arhitekturu računara, što znači da su razne generacije računara prolazile kroz slične razvojne cikluse.

7.4. PITANJA

1. Šta opisuje arhitektura naredbi?
2. Koje vrste naredbi sadrži skup naredbi?
3. Koje tipove podataka podržava arhitektura naredbi?
4. Šta karakteriše operande?
5. Koje organizacione komponente računara postoje?
6. Šta su direktni proizvodni troškovi računara?
7. Šta su indirektni proizvodni troškovi računara?
8. Na čemu se temelji pozitivna povratna sprega koja predstavlja zamajac razvoja računara?
9. Za šta se vezuju evolucione faze arhitekture računara?

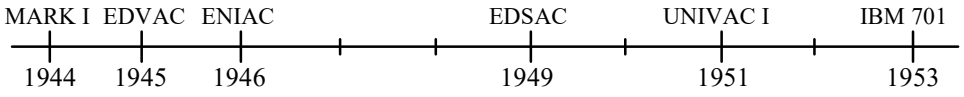
8. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 1950. GODINE

8.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 1950. GODINE

U 1951. godini na tržište Sjedinjenih Američkih Država isporučen je prvi elektronski računar opšte namene, nazvan *UNIVAC I* (*UNIVersal Automatic Computer*). Njegovi autori *J. Presper Eckert* i *John W. Mauchly* su prethodno predvodili grupu sa Pensilvanija univerziteta, koja je 1946. godine završila prvi elektronski računar opšte namene. Ovaj računar je nazvan *ENIAC* (*Electronic Numerical Integrator And Calculator*). Njegova radna memorija je sadržala samo obrađivane podatke, dok su programi ručno zadavani posredstvom upravljačke table. Nameru prethodno pomenute grupe da napravi računar sa radnom memorijom predviđenom za smeštanje i programa i podataka opisao je 1945. godine njen član *John von Neumann*. On je opisani računar nazvao *EDVAC* (*Electronic Discrete Variable Automatic Computer*). U 1946. godini *John von Neumann* je objavio opis usavršene verzije računara sa radnom memorijom predviđenom za smeštanje i programa i podataka, nazvanog *IAS* (po nazivu institucije *Institute for Advanced Study* Prinston univerziteta, u okviru koje je pomenuti opis nastao). U ova dva rada su izloženi suštinski principi funkcionisanja elektronskih računara opšte namene sa radnom memorijom predviđenom za smeštanje i programa i podataka. Prvi ovakav (u punom obimu funkcionalan) računar, pod imenom *EDSAC* (*Electronic Delay Storage Automatic Calculator*), napravila je u Velikoj Britaniji 1949. godine grupa sa Kembridž univerziteta, koju je predvodio *Maurice Wilkes*.

U 1953. godini *IBM* (*International Business Machines Corporation*, tada vodeći proizvođač poslovne opreme, prisutan na tržištu od 1896. godine) isporučio je na tržište Sjedinjenih Američkih Država svoj prvi elektronski računar opšte namene pod imenom *IBM 701*. Prethodno, u 1944. godini, *IBM* je napravio prvi elektro-mehanički računar opšte namene, nazvan *MARK I*. Idejni tvorac projekta ovoga, kao i kasnije napravljena 3 računara, nazvana *MARK II*, *MARK III* i *MARK IV*, je *Howard Aiken* sa Harvard univerziteta. Poslednja dva modela su bili elektronski računari opšte namene sa razdvojenim radnim memorijama za programe i za podatke. Ovakav pristup je nazvan **harvardska arhitektura**.

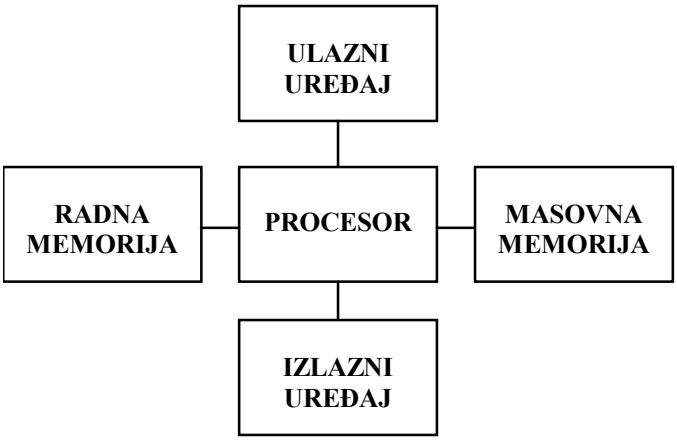
Slika 8.1.1 sadrži prikaz hronologije važnih događaja za razvoj računara.



Slika 8.1.1 Hronologija važnih događaja za razvoj računara

ARHITEKTURA RAČUNARA PRVE GENERACIJE

UNIVAC I i *IBM 701* pripadaju računarima prve generacije. Slika 8.1.2 sadrži prikaz tipične organizacije računara prve generacije.



Slika 8.1.2 Tipična organizacija računara prve generacije

Upravljačke, adresne i linije podataka povezuju procesor i preostale organizacione komponente računara prve generacije (Slika 8.1.2). Procesori prve generacije računara su izrađivani u tehnologiji elektronskih cevi (elektronska cev trioda je napravljena 1906. godine). Radne memorije su zasnivane na raznim tehnologijama, pa su tako korišćene, na primer, i elektrostatičke radne memorije, kod kojih su naelektrisanja tačaka na ekranu katodne cevi predstavljala binarne vrednosti. Jedinice magnetnih traka su korišćene kao masovne memorije. Čitači bušenih traka ili kartica su predstavljali ulazne uređaje, a na mestu izlaznih uređaja su se nalazili bušači traka i kartica, kao i štampači. Centralni položaj procesora u organizaciji je imao za posledicu da bez njegovog učešća nije bio moguć prenos podataka između bilo koje dve organizacione komponente računara prve generacije.

Za prenos podataka između radne memorije, sa jedne strane, i ulaznih i izlaznih uređaja, odnosno, masovne memorije, sa druge strane, postojale su posebne ulazno-izlazne naredbe. Radi njih, je postojao **ulazno-izlazni adresni prostor**, kome su pripadale samo lokacije koje su predstavljale ulazne i izlazne uređaje, odnosno jedinice masovne memorije. Pored ulazno-izlaznog adresnog prostora, postojao je i **memorijski adresni prostor**, kome su pripadale samo lokacije radne memorije.

Prema tome, ista adresa je označavala dve lokacije, zavisno od toga da li je interpretirana kao adresa memorijskog ili adresa ulazno-izlaznog adresnog prostora. Druga interpretacija adresa je bila moguća samo u okviru ulazno-izlaznih naredbi.

Računari prve generacije su bili namenjeni prevashodno za numeričke proračune, kao što su, na primer, računanja tablica trigonometrijskih funkcija. Ipak, oni nisu podržavali aritmetiku pomične tačke (*floating-point arithmetic*), jer je ona bila suviše komplikovana za tehnologiju elektronskih cevi. Oni, takođe, nisu podržavali ni potprograme, a ni rukovanje elementima nizova. Identična obrada uzastopnih elemenata niza je zahtevala ponavljanje modifikacija mašinskih naredbi programa, radi pripremanja u mašinskim naredbama adresa lokacija radne memorije sa vrednostima pojedinih elemenata.

Računari prve generacije su korišćeni na interaktivan način, pri čemu su programeri, posredstvom upravljačke table, bili u neposrednoj komunikaciji sa računarom za vreme izvršavanja svojih programa. Programeri su u početku bili upućeni na korišćenje mašinskog jezika, a kasnije i na korišćenje asemblerskog jezika.

Već kod računara prve generacije brzinu procesora, odnosno broj izvršenih mašinskih naredbi u jedinici vemena, ograničavala je samo raspoloživa tehnologija, dok je brzinu radne memorije, odnosno broj pristupa njenim lokacijama u jedinici vremena ograničavala cena raspoloživih tehnologija. Primena najskuplje tehnologije je bila i ostala prihvatljiva kod procesora, ali ne i kod radne memorije, jer je procesor sastavljen od neuporedivo manje komponenti nego radna memorija. To znači da se već kod računara prve generacije javila ozbiljna disproporcija između brzine procesora i brzine radne memorije (da se i ne pominje disproporcija brzine procesora i brzine ostalih, mehanički zasnovanih organizacionih komponenti računara prve generacije).

MANE RAČUNARA PRVE GENERACIJE

Ozbiljna mana računara prve generacije je bila njihova slaba iskorišćenost. Nju su uzrokovali:

1. interaktivni način rada (dok je programer ispravljao greške u svom programu, računar je bio uglavnom neiskorišćen),
2. modifikovanje programa u toku njihovog izvršavanja (što je zahtevalo punjenje programa pre svakog novog izvršavanja) i
3. učešće procesora u prenosu svakog podatka između bilo koje dve organizacione komponente računara prve generacije (u toku čega je, zbog radnog čekanja, procesorsko vreme neracionalno korišćeno).

8.2. PITANJA

1. Šta je karakterisalo prenos podataka između radne i masovne memorije kod računara prve generacije?
2. Koliko adresnih prostora postoji kod računara prve generacije?
3. Šta je karakterisalo računare prve generacije?
4. Koju manu su imali računari prve generacije

9. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 1960. GODINE

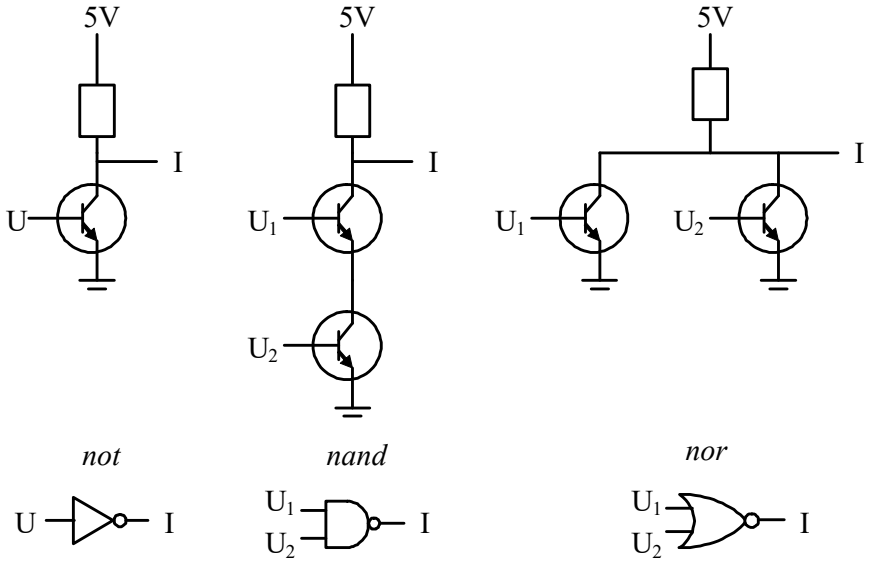
9.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 1960. GODINE

Tehnološku osnovu računara druge generacije su činili diskretni poluprovodnici i magnetne jezgrice (*magnetic core*). Tranzistori su napravljeni 1948. godine, a magnetne jezgrice su napravljane 1949. godine.

DIGITALNA KOLA

Diskretni poluprovodnici su istisnuli elektronske cevi, jer su imali nižu cenu, bili manji, brži i pouzdaniji, a imali su i manju potrošnju energije i manje toplotno zračenje.

Diskretni poluprovodnici su korišćeni za izvedbu **digitalnih kola** (*gate*), sposobnih da računaju vrednosti logičkih funkcija. Slika 9.1.1 prikazuje osnovna digitalna kola *not*, *nand* i *nor* koja odgovaraju logičkim funkcijama *not* (negacija), *nand* (negirano logičko i) i *nor* (negirano logičko ili). Ispod imena ovih kola su nacrtani njihovi simboli.



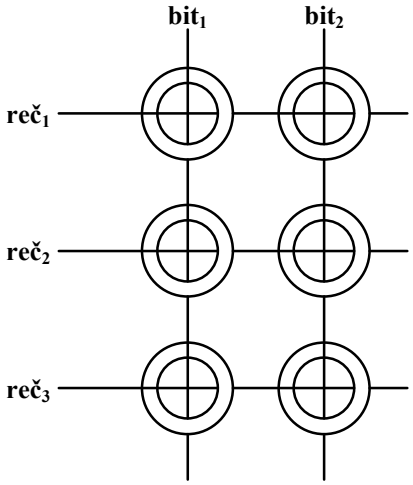
Slika 9.1.1 Osnovna digitalna kola

Za prvo od digitalnih kola (Slika 9.1.1) važi da, kada se ulazni napon U (doveden na bazu tranzistora) poveća sa 0V na 5V, tranzistor provede, pa obori

izlazni napon I sa 5V na 0V. Takođe važi i obrnuto. Tako se ulazna logička vrednost tačno (koju predstavlja napon od 5 volti) invertuje u izlaznu logičku vrednost netačno (koju predstavlja napon od 0 volti) i obrnuto. Na analogan način funkcionišu i preostala dva digitalna kola. Kombinovanjem osnovnih logičkih kola mogu se izvesti složene logičke funkcije.

RADNA MEMORIJA

Magnetne jezgrice su istisnule do tada korišćene memorijske tehnologije, jer su omogućile pravljenje većih i pouzdanijih radnih memorija po nižoj ceni. U tehnologiji magnetnih jezgrica binarne vrednosti su predstavljala dva stanja magnetizacije prstena, napravljenog od magnetnog materijala, kroz koji su prolazili električni provodnici. Slika 9.1.2 sadrži prikaz principijelne organizacije radne memorije, zasnovane na magnetnim jezgricama.



Slika 9.1.2 Principijelna organizacija radne memorije, zasnovane na magnetnim jezgricama

Slika 9.1.2 sadrži primer radne memorije sa 3 reči od po 2 bita. Svaki bit je predstavljen jednom magnetnom jezgricom. Selekciju reči omogućuju provodnici $reč_i$, a pisanje i čitanje bita omogućuju provodnici bit_j. Pisanje jedinice u bit "j" reči "i" je podrazumevalo prevođenje odgovarajuće magnetne jezgrice u željeno magnetno stanje. Do toga je dolazilo ako se i kroz provodnik $reč_i$ i kroz provodnik bit_j u zadanom smeru propusti polovična struja magnetizacije, tako da se u preseku ovih provodnika dobije struja magnetizacije. Čitanje ovako predstavljenih bita reči "i" se sastojalo od propuštanja struje magnetizacije kroz provodnik $reč_i$ u suprotnom smeru od onog koji je korišćen za pisanje i od interpretacije stanja provodnika bit_j. U ovim provodnicima se indukovala struja, ako se pri čitanju menjalo stanje

magnetizacije magnetne jezgrice, pa je čitanje bilo destruktivno. Ovakvu memoriju je karakterisalo:

1. **vreme pristupa** lokaciji (*access time*), koje protekne između postavljanja zatheva za pristupom i obavljanja željenog pristupa (na primer, vreme između postavljanja zahteva za čitanjem lokacije i dobijanja njenog sadržaja) i
2. **vreme ciklusa** (*cycle time*), koje protekne između dva uzastopna pristupa.

Vreme ciklusa je bilo duže od vremena pristupa, jer je uključivalo i obnovu sadržaja, eventualno poništenog pri čitanju lokacije. To znači da je do novog pristupa lokaciji memorije dolazilo tek nakon obnove sadržaja prethodno pročitane lokacije.

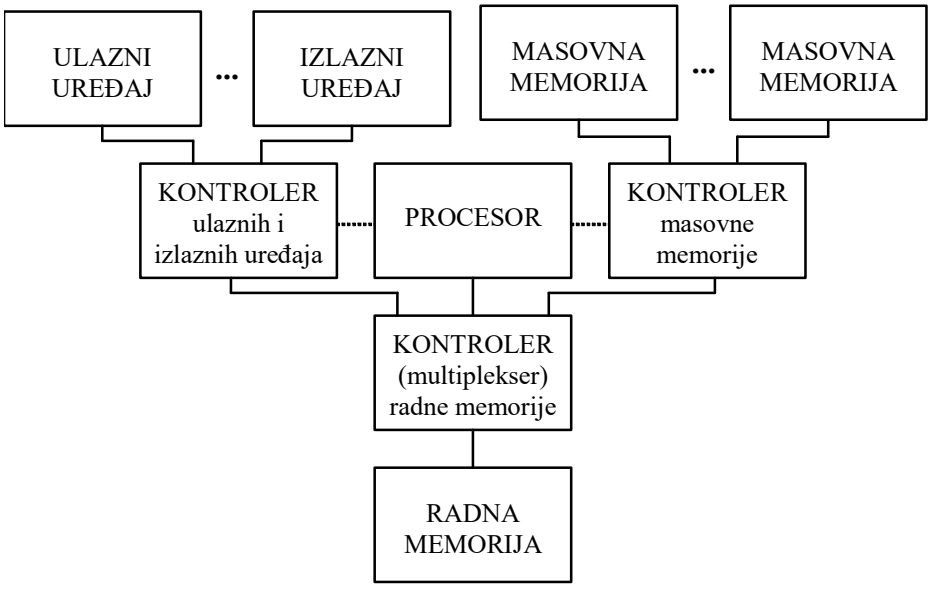
KONTROLERI

Primena poluprovodničke tehnologije je omogućila pojavu kontrolera. Oni su bili specijalizovani za pojedine poslove, kao što je, na primer, upravljenje ulaznim i izlaznim uređajima ili jedinicama masovne memorije. Složenost kontrolera je zavisila od njihove uloge. Tako, na primer, najjednostavniji kontroleri su samo multipleksirali linije, dok su najsloženiji kontroleri upravljali nizom ulaznih i izlaznih uređaja ili jedinica masovne memorije. U drugom slučaju, kontroleri su imali oblik jednostavnijih procesora, posvećenih određenom zadatku. Zato su ovakvi kontroleri nazivani i **procesori posebne namene** (*input output processor*), a njihove osobine su bile u potpunosti podređene njihovom zadatku. Na primer, u skupu naredbi ovakvih kontrolera su se nalazile:

1. upravljačke naredbe,
2. naredbe za rukovanje pojedinim ulaznim i izlaznim uređajima ili jedinicama masovne memorije (poput naredbe za premotavanje magnetne trake) i
3. naredbe za prenos podataka između radne memorije i ulaznog ili izlaznog uređaja, odnosno jedinice masovne memorije.

Procesori posebne namene nisu, za razliku od procesora opšte namene, bili predviđeni, pa ni raspoloživi za izvršavanje korisničkih programa.

Slika 9.1.3 sadrži prikaz tipične organizacije računara druge generacije.



Slika 9.1.3 Tipična organizacija računara druge generacije

Isprekidane linije (Slika 9.1.3) odgovaraju upravljačkim linijama, a pune linije odgovaraju upravljačkim, adresnim i linijama podataka. Kontroler radne memorije je bio zadužen samo za multipleksiranje linija. Na mestu kontrolera ulaznih i izlaznih uređaja i kontrolera masovne memorije su se nalazili procesori posebne namene. Oni su ravnopravno pristupali radnoj memoriji, kao i procesor opšte namene. Zahvaljujući tome, procesor opšte namene je u radnoj memoriji mogao da ostavi ulazno-izlazni program, namenjen za procesor posebne namene i da pokrene njegovu aktivnost, zaustavljajući svoju aktivnost dok traje aktivnost procesora posebne namene.

ARHITEKTURA NAREDBI

Osobine računara druge generacije su zavisile od toga da li su bili namenjeni za poslovne obrade, kao, na primer, *IBM 1401*, ili za numeričke obrade, kao, na primer, *IBM 7094*. U drugom slučaju, procesori su podržavali aritmetiku pomične tačke. Takođe, postojala je i podrška za potprograme, a potreba za modifikovanjem programa je izbegnuta uvođenjem indeksnog adresiranja. Ono je omogućilo da se (efektivna) adresa, na primer, elementa niza, odredi u toku izvršavanja naredbi sabiranjem adrese početka niza, sadržane u mašinskom formatu izvršavane naredbe, i indeksa elementa niza, sadržanog u indeksnom registru.

Kod računara druge generacije je postojala šarenolikost u pogledu broja adresa u okviru mašinskih formata naredbi. Zavisno od računara, broj adresa je varirao od 3 do 0. Ako je u mašinskom formatu naredbe bio predviđen prostor za 3

adrese, tada su se u naredbama, kao što je naredba sabiranja, mogli izričito označiti dva ulazna operanda sa sabircima i izlazni operand, namenjen za zbir. Međutim, ako je bilo predviđeno manje adresa, tada se za neke od operanada podrazumevalo da se nalaze u predodređenim lokacijama. Tako, za naredbe sabiranja kod dvo-adresnih računara se podrazumevalo da jedan operand odgovara jednom sabirku, a da drugi operand odgovara drugom sabirku do obavljanja operacije i zbiru nakon obavljanja operacije. Za naredbe sabiranja jedno-adresnih računara se podrazumevalo da se jedan od sabiraka nalazi u posebnom registru procesora, nazvanom akumulator, gde se smeštao (akumulirao) i zbir. Ili, za naredbe sabiranja nula-adresnih računara se podrazumevalo da se oba sabirka nalaze na vrhu steka, gde se odlagao i zbir. Za jedno-adresne računare, kao što je bio, na primer, *IBM 7094*, vezan je pojam **akumulatorske arhitekture** (*accumulator architecture*), a za nula-adresne računare, kao što je bio, na primer, *Burroughs B5000*, vezan je pojam **stek arhitekture** (*stack architecture*). Skup naredbi računara sa akumulatorskom arhitekturom obavezno je sadržao i naredbe za rukovanje akumulatorima, a skup naredbi računara sa stek arhitekturom obavezno je sadržao i naredbe za rukovanje stekom. Smanjenje broja adresa u mašinskom formatu naredbe je smanjivalo veličinu memorije, potrebne sa smeštanje mašinskih naredbi, ali je, istovremeno, uzrokovalo i povećanje broja mašinskih naredbi u programu, jer su bile neophodne dodatne mašinske naredbe, na primer, za smeštanje potrebnih operanada u akumulator ili u stek. Prema tome, efekat smanjenja broja adresa u mašinskim formatima naredbi je poništavalo povećanje broja mašinskih naredbi u programu.

PROGRAMSKI JEZICI VISOKOG NIVOA

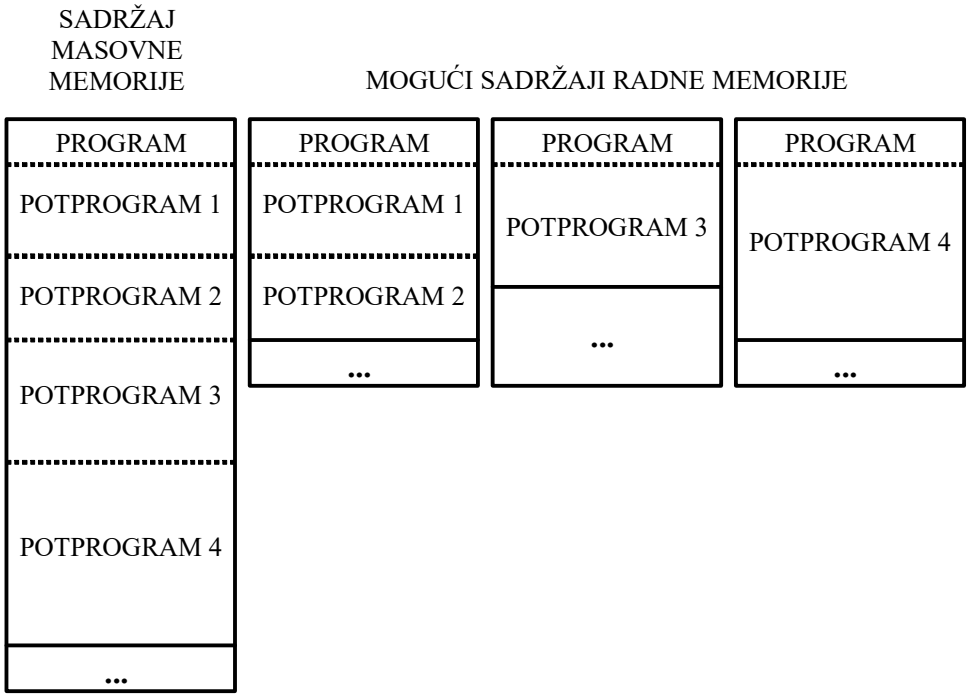
Za računare druge generacije je vezana pojava programskih jezika visokog nivoa, kao što su:

1. *FORTRAN* (*FORmula TRANslation*, čiji razvoj je od 1954. do 1957. godine obavila grupa koju je predvodio *John Backus*, a koju je finansirao *IBM*) i
2. *COBOL* (*COmmon Business Oriented Language*, čija specifikacija je završena 1959. godine, pod nadzorom *CODASYL*-a, *Conference On DAta SYstems Languages*).

Programski jezici visokog nivoa su ponudili uopšteni programski model računara (druge generacije) i tako omogućili njihovo programiranje bez poznavanja detalja njihovog funkcionisanja. Programski jezici visokog nivoa su zahtevali postojanje kompajlera, koji su programe, pisane programskim jezicima visokog nivoa, prevodili u mašinski oblik. Zahvaljujući kompajlerima, programi, pisani programskim jezicima visokog nivoa, su postali prenosivi sa računara na računar. Takođe, pojavile su se biblioteke često korišćenih potprograma i linkeri, namenjeni za spajanje nezavisno prevedenih programa i potprograma.

MEMORIJSKA HIJERARHIJA

Ograničena veličina radne memorije računara druge generacije je terala programere da u radnoj memoriji drže samo deo izvršavanog programa i samo deo obrađivanih podataka, a ostatak u masovnoj memoriji. Radi toga su razvijene i programerske tehnike za **preklapanje** (*overlaying*) delova programa, čije istovremeno prisustvo u radnoj memoriji nije bilo potrebno. Na ovaj način radna i masovna memorija su tretirani kao dva nivoa **memorijske hijerarhije**, kojom je upravljao korisnički program, tako što je inicirao prebacivanje svojih delova sa jednog nivoa memorijske hijerarhije na drugi. Slika 9.1.4 sadrži prikaz primera preklapanja.



Slika 9.1.4 Primeri preklapanja potprograma u radnoj memoriji

OPERATIVNI SISTEMI

Oslanjanje samo na programske jezike visokog nivoa, i iz toga proizašlo nepoznavanje detalja funkcionisanja računara druge generacije, je onemogućilo programere da ove računare koriste na interaktivan način, uobičajen za računare prve generacije. To je imalo za posledicu da je nestao jedan od uzroka neracionalnog korišćenja računarskog vremena. Umesto interaktivnog načina rada, za računare druge generacije je uvedena praksa da programeri svoje programe,

pripremljene na bušenim karticama, predaju operaterima (osoblju, zaduženom za opsluživanje računara). Operateri bi prikupljene programe jedan za drugim puštali na izvršavanje (*batch processing*), a rezultate njihovog izvršavanja, na primer, u obliku štampanih izveštaja, zajedno sa bušenim karticama, vraćali programerima. Na ovaj način, računar je bio zaposlen dok god je bilo korisničkih programa, spremnih za izvršavanje. Radi podrške ovakvog načina rada, razvijeni su i prvi operativni sistemi (*batch monitor*, *batch system*). Njihove mogućnosti su bile vrlo skromne i dozvoljavale su, na primer, samo kompilaciju programa, kao i punjenje i izvršavanje njegovog mašinskog oblika.

MANE RAČUNARA DRUGE GENERACIJE

Računari druge generacije nisu koristili potencijalni paralelizam u radu procesora opšte i posebne namene, jer je procesor opšte namene, nakon pokretanja procesora posebne namene, zaustavljao svoju aktivnost do završetka rada procesora posebne namene. Zbog toga, vreme procesora opšte namene nije bilo najbolje korišćeno. To je, zbog njihove visoke cene, bilo nedopustivo. Pored toga, neinteraktivni način rada je ozbiljno umanjio produktivnost programera, koji su često dugo čekali između predavanja kartica sa svojim programom i dobijanja rezultata njegovog izvršavanja. Na kraju, upravljanje memorijskom hijerarhijom je bio suvišan, a i pretežak posao za mnoge programere.

9.2. PITANJA

1. Šta čini tehnološku osnovu računara druge generacije?
2. Zašto su diskretni poluprovodnici istisnuli elektronske cevi?
3. Šta je karakterisalo radnu memoriju, sastavljenu od magnetnih jezgrica?
4. Zašto su magnetne jezgrice istisnule prvobitne memorijske tehnologije?
5. Kako se obavlja prenos podataka između radne i masovne memorije kod računara druge generacije?
6. Koje naredbe su podržavali procesori posebne namene?
7. Šta je karakterisalo računare druge generacije?
8. Kakvi računari imaju akumulatorsku arhitekturu?
9. Kakvi računari imaju stek arhitekturu?
10. Čija pojava je vezana za računare druge generacije?
11. Koje mane imaju računari druge generacije?

10. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 1970. GODINE

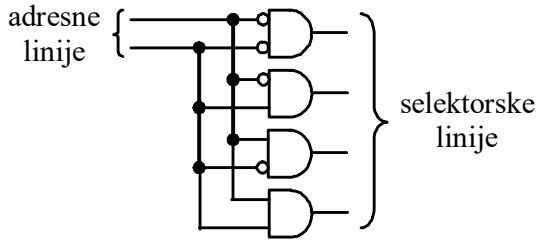
10.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 1970. GODINE

Tehnološku osnovu računara treće generacije su činila integrisana kola (*integrated circuit*) i magnetni diskovi (*magnetic disk*).

INTEGRISANA KOLA

Tehnologija integrisanih kola je omogućila smeštanje minijaturnog elektronskog kola, sastavljenog od nekoliko tranzistora, kondenzatora, dioda ili otpornika, na površinu silicijumske pločice. Ovakva pločica, upakovana u zaštitni omotač sa kontaktnim izvodima, je nazvana čip (*chip*). Ova tehnologija je istisnula tehnologiju diskretnih poluprovodnika, zbog niže cene, veće brzine i veće pouzdanosti, a manje potrošnje energije i manjeg toplotnog zračenja. Primena tehnologije integrisanih kola je izazvala sniženje cene, između ostalog, zato što je omogućila automatizaciju proizvodnje.

Tipična primena integrisanih kola je izvedba kombinacionih kola poput dekodera adrese. U ovakav dekoder ulazi n linija, a iz njega izlazi 2^n linija (Slika 10.1.1). Podrazumeva se da je za bilo koju kombinaciju signala na ulaznim linijama uvek postavljena samo jedna izlazna linija. Znači, kada se na ulaznim linijama dekodera adrese pojavi adresa neke memorijske lokacije, tu lokaciju selektuje njegova postavljena izlazna linija.



Slika 10.1.1 Dekoder 2x4

MAGNETNI DISKOVI

Pojava magnetnih diskova nije označila kraj, nego samo izmenu načina upotrebe magnetnih traka. One su i dalje korišćene za arhiviranje podataka, kao i za prenošenje podataka sa računara na računar, znači za poslove kod kojih je sekvencijalni pristup podacima bio zadovoljavajući. U svim ostalim slučajevima, kod kojih je mogućnost nesekvencijalnog pristupa bila važna, odnosno kod kojih je bilo bitno da pristup podacima bude što brži, magnetni diskovi su istisnuli magnetne trake, tako da je jedinica magnetnog diska postala sinonim za masovnu memoriju.

VELIKI I MINI RAČUNARI

Primena tehnologije integrisanih kola je omogućila proizvođačima računara treće generacije da znatno poprave odnos cene i funkcionalnosti (*price/performance*) svojih proizvoda. To se moglo postići zadržavanjem iste funkcionalnosti, uz smanjenje cene, ili poboljšanjem funkcionalnosti uz zadržavanje cene. Jedna grupa proizvođača računara treće generacije je pomenuti odnos popravila pre svega poboljšavanjem funkcionalnosti svojih proizvoda, dok se druga grupa proizvođača odlučila za skromniju funkcionalnost, ali i za znatno nižu cenu svojih proizvoda. Kao rezultat prvog pristupa nastali su **veliki računari** (*mainframe*), a kao rezultat drugog pristupa nastali su **mini-računari** (*minicomputer*). Pomenuti nazivi su bili posledica, ne samo činjenice da su cene i mogućnosti mini-računara predstavljale samo delić cene i mogućnosti velikih računara, nego i činjenice da su fizičke dimenzije mini-računara bile znatno manje od fizičkih dimenzija velikih računara. Tako, na primer, procesor mini-računara je mogao da stane na jednu štampanu ploču (*printed-circuit board*), dok je procesor velikog računara zauzimao više štampanih ploča.

FAMILIJE RAČUNARA

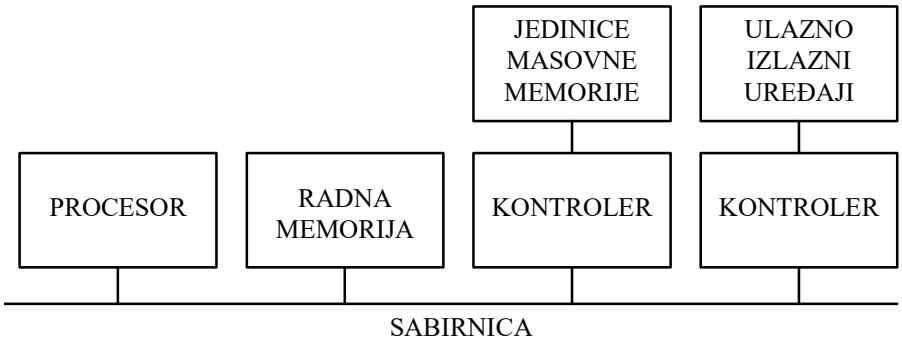
Za pojavu velikih računara je vezana ideja **familije računara**. Svi računari iz jedne familije su imali istu osnovnu arhitekturu naredbi, ali su se razlikovali po ceni i po mogućnostima, kao što su brzina rada procesora, veličina radne i masovne memorije, ili broj ulaznih i izlaznih uređaja. Familija računara je omogućila korisnicima da u nabavci računara krenu od modela sa najnižom cenom i sa najmanjim mogućnostima i da kasnije, u skladu sa povećanjem svojih potreba, bez teškoća pređu na skuplje i moćnije modele. Ideja familije računara je olakšala i posao standardizacije, pa je zato imala pozitivan uticaj i na proizvodnju računara. Zahvaljujući upravo ideji familije računara i ogromnom tržišnom uspehu modela iz familije velikih računara *IBM 360/370*, namenjenih i za poslovne i za numeričke obrade, *IBM* je postao vodeći proizvođač računara uopšte.

OTVORENA ARHITEKTURA

Za razliku od velikih računara, okrenutih tržištu poslovnih i numeričkih obrada, mini-računari su prve primene našli u upravljanju industrijskim procesima, jer su mogli da im budu u potpunosti posvećeni, zahvaljujući svojoj niskoj ceni. Ovakve primene su podrazumevale vezivanje za mini-računare raznih uređaja, kao što su senzori ili izvršni organi. Pošto nisu mogli da proizvedu svaki od takvih uređaja, za proizvođače mini-računara je bilo važno da potaknu nezavisnu proizvodnju pomenutih uređaja za svoje mini-računare. Takođe, pošto nisu mogli ni da pripreme poseban primerak mini-računara za svaku od raznovrsnih primena, za proizvođače mini-računara je bilo važno da korisnici samostalno prilagođavaju mini-računare svojim potrebama. I jedno i drugo je omogućeno zahvaljujući pristupu **otvorene arhitekture** (*open architecture*). Ovaj pristup se zasnivao na

korišćenju sabirnice za povezivanje svih preostalih organizacionih komponenti mini-računara i na publikovanju potpune specifikacije sabirnice. Na taj način, nezavisni proizvođači su mogli da prave nove organizacione komponente mini-računara, odnosno da prave sopstvene kontrolere, posredstvom kojih su njihovi uređaji mogli da budu zakačeni na sabirnicu i da tako budu ugrađeni u mini-računar. Takođe, korisnici su mogli samostalno da nabavljaju organizacione komponente mini-računara od raznih proizvođača i da od njih sklapaju sopstveni mini-računar. Zahvaljujući, između ostalog, i ovakvom pristupu, familija mini-računara *PDP-11* (*Programmed Data Processor*), organizovana oko sabirnice, nazvane *Unibus*, je doživela veliki tržišni uspeh, a njen proizvođač, *DEC* (*Digital Equipment Corporation*), je postao vodeći proizvođač mini-računara.

Slika 10.1.2 sadrži prikaz tipične organizacije mini-računara.



Slika 10.1.2 Tipična organizacija mini-računara

Sabirnica se sastojala od upravljačkih, adresnih i linija podataka, namenjenih za međusobno povezivanje pojedinih parova organizacionih komponenti računara, koji su za nju zakačeni. Jedinice masovne memorije su obuhvatale i jedinice magnentih diskova i jedinice magnetnih traka. Ulazno-izlazni uređaji su obuhvatali čitače bušenih kartica ili traka, štampače, ali i terminale, namenjene za dvosmernu komunikaciju sa računarom. Između kontrolera su postojale velike razlike, na primer, po broju uređaja koje su opsluživali, ili, na primer, po tome da li su imali *DMA* svojstva.

PRODUKTIVNOST PROGRAMERA

Poboljšanje odnosa cene i funkcionalnosti računara treće generacije je imalo za posledicu da je vreme programera postalo dragocenije od vremena računara, pa se prirodno nametnulo pitanje kako funkcionalnost računara treće generacije podrediti potrebama povećanja produktivnosti programera. To se, pre svega, moglo postići pronalaženjem načina za interaktivno korišćenje računara treće generacije, da bi programeri mogli da nadgledaju izvršavanja svojih programa i da tako u njima

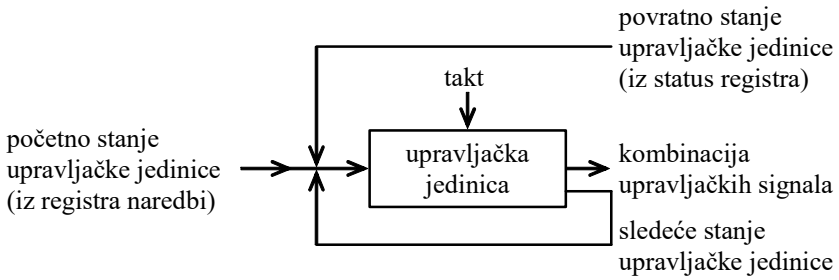
brže otkrivaju i otklanjaju greške. Racionalnijem trošenju programerskog vremena je doprinosila i automatizacija upravljanja računarom, kao što je automatizacija upravljanja memorijskom hijerarhijom. Na produktivnost asemblerskih programera, čiji broj u periodu oko 1970. godine uopšte nije bio zanemarljiv, povoljno je uticalo i proširenje arhitekture naredbi novim naredbama, radi približavanja izražajnosti asemblerskih naredbi iskazima programskih jezika visokog nivoa.

ARHITEKTURA NAREDBI

Izražajnost asemblerskih naredbi se približavala iskazima programskog jezika visokog nivoa na sledeće načine:

1. proširenjem skupa naredbi, radi pokrivanja i opštih i posebnih potreba,
2. podrškom većeg broja tipova podataka,
3. povećanjem broja raspoloživih adresiranja i
4. **ortogonalnošću** (nezavisnošću) **naredbi i adresiranja**, odnosno omogućavanjem slobodnog kombinovanja adresiranja u okviru naredbi, tako da svi operandi mogu biti u registrima (*register-register architecture*), u memorijskim lokacijama (*memory-memory architecture*) ili u bilo kojoj međusobnoj kombinaciji.

Ovakav pristup je doveo do *CISC* (*Complex Instruction Set Computer*) računara sa veoma kompleksnom arhitekturom naredbi. Za upravljanje njihovim procesorima je bilo potrebno generisati mnoštvo sekvenci raznih kombinacija upravljačkih signala. To se praktično nije moglo ostvariti pomoću ožičene (*hardwired*) upravljačke jedinice (Slika 10.1.3), zbog njene preterane kompleksnosti i previsoke cene.



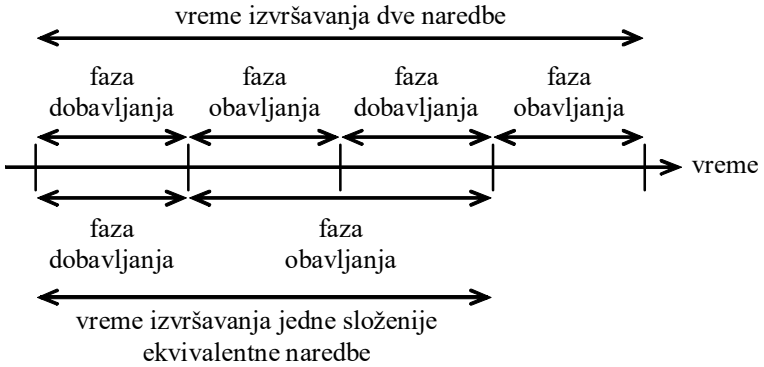
Slika 10.1.3 Organizacija ožičene upravljačke jedinice

MIKRO-PROGRAMIRANJE

Za upravljanje *CISC* procesorioma upotrebljena je mikro-programska (*microprogrammed*) upravljačka jedinica. Za nju je važno uočiti da je vreme pristupa lokacijama njene mikro-programske memorije određivalo dužinu ciklusa procesora, jer je on preuzimao sadržaj jedne od lokacija ove memorije u svakom od svojih ciklusa.

Mikro-programsku upravljačku jedinicu je predložio *Maurice Wilkes* još 1951. godine, ali je njena prva pojava na tržištu vezana za računare *IBM 360/370* familije, jer su tek tada sazreli tehnološki uslovi za njenu upotrebu. Prve mikro-programske upravljačke jedinice računara treće generacije su se oslanjale na mikro-programsku memoriju nepromenljivog sadržaja (*ROM*, *Read Only Memory, Control Store*), koja je imala oblik diodne matrice. Vreme pristupa i vreme ciklusa ovakve memorije se nisu razlikovali i bili su dovoljno kratki, posmatrano sa stanovišta dužine procesorskog ciklusa.

Primena mikro-programske upravljačke jedinice je omogućila veliku fleksibilnost računara, jer je za izmenu funkcionalnosti računara bilo dovoljno napraviti nove mikro-programe. Ovakva aktivnost je nazvana mikro-programiranje. Mikro-programiranje je omogućilo **emulaciju** (oponašanje) jednog računara pomoću drugog računara. Za to je bilo dovoljno napisati interpreter mašinskih naredbi prvog računara u obliku mikro-programa drugog računara. Pored toga, mikro-programiranje je bilo zaslužno i za bolje iskorišćenje procesorskog vremena, jer su složenije naredbe i adresiranja uticali na smanjenje broja pristupa radnoj memoriji radi dobavljanja naredbi i na duže angažovanje procesora nakon svakog od pomenutih dobavljanja (Slika 10.1.4).



Slika 10.1.4 Poređenje izvršavanja dve naredbe i jedne složenije ekvivalentne naredbe

Smanjenje broja pristupa radnoj memoriji je bilo važno, jer je vreme ciklusa radne memorije, napravljene od magnetnih jezgrica, trajalo kao 10 procesorskih ciklusa, pa je svako pristupanje radnoj memoriji zaustavljalo aktivnost procesora u trajanju od 10 njegovih ciklusa. Zato je, na primer, bilo bolje da se u okviru izvršavanja jedne naredbe obradi element niza i pripremi sadržaj indeksnog registra za obradu narednog elementa istog niza, nego da se isti posao obavi u toku izvršavanja dve naredbe. Ovakvo smanjenje broja naredbi u programu je bilo racionalno i sa stanovišta korišćenja radne memorije.

PROMENLJIVI FORMATI MAŠINSKIH NAREDBI

Racionalno korišćenje radne memorije je bilo važno, jer je tehnologija magnetnih jezgrica praktično ograničavala veličinu radnih memorija. One su često imale znatno manje lokacija od broja omogućenog adresnim prostorom, ne samo kod velikih računara sa 24 bitnim adresnim prostorom, kao kod modela iz *IBM 360/370* familije, nego i kod mini-računara sa 16 bitnim adresnim prostorom, kao kod modela iz *PDP-11* familije. Radi skraćenja programa, formati mašinskih naredbi računara treće generacije su bili promenljive dužine. Kraći formati su omogućavali ne samo manje zauzeće radne memorije, nego i brže dobavljanje (zbog smanjenja broja pristupa radnoj memoriji). Dužina ovih formata je zavisila, ne samo od tipa naredbe (broja operanada), nego i od dužine njenih operanada.

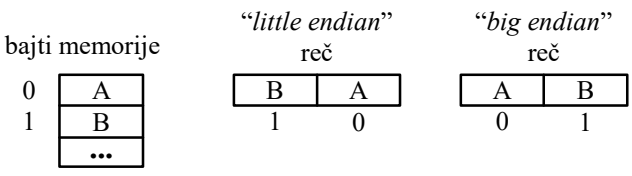
ADRESIRANJA

Za računare treće generacije, koji su za svaku naredbu vezivali samo jednu kombinaciju adresiranja, mašinski formati naredbi nisu sadržali posebne kodove adresiranja, jer su adresiranja mogla biti određena iz koda naredbe. Ovaj pristup je bio tipičan za *IBM*. U slučaju računara koji su dozvoljavali kombinovanje raznih adresiranja u okviru iste naredbe, mašinski format naredbi je obavezno sadržao i kodove adresiranja. Ovaj pristup je bio tipičan za *DEC*. Za drugi pristup je bila karakteristična težnja da se omogući ortogonalnost naredbi i adresiranja, odnosno, da se omogući kombinovanje svih adresiranja u okviru svake naredbe.

Nepovoljna iskustva sa stek i akumulatorskim arhitekturama su uzrokovala da se procesori računara treće generacije ne oslanjaju na stek ili na akumulatore, nego na registre opšte namene. Zato je za ove računare vezan pojam **arhitekture registara opšte namene** (*general purpose register architecture*). U mašinskom formatu naredbi, adrese registara opšte namene su bile ravnopravne adresama lokacija radne memorije, a zahvaljujući promenljivom formatu, naredbe računara treće generacije su sadržale do 3 ovakve adrese.

ORGANIZACIJA RADNE MEMORIJE

Radna memorija računara treće generacije je bila organizovana u 8 bitne bajte. Pored adresiranja svakog bajta, postojala je i mogućnost adresiranja reči. Pri tome se javila razlika u pogledu poretka bajta unutar reči (Slika 10.1.5).



Slika 10.1.5 Dva načina organizovanja bajta u reči

Po jednom pristupu (nazvanom *little endian*), na najmanje značajnom mestu u reči se nalazio bajt sa najnižom adresom, a po drugom pristupu (nazvanom *big endian*), na najmanje značajnom mestu u reči se nalazio bajt sa najvišom adresom. Prvi pristup je usvojio, na primer, *DEC*, a drugi, na primer, *IBM*. Prema tome, prilikom prenosa podataka sa računara jednog proizvođača na računar drugog proizvođača, bilo je neophodno voditi računa o poretku bajta u reči.

Za računare treće generacije je vezano objedinjavanje memorijskog adresnog prostora i ulazno-izlaznog adresnog prostora u jedinstven adresni prostor. Ovakav pristup je nazvan **memorijski preslikani ulaz-izlaz** (*memory-mapped input output*), a prvi ga je primenio *DEC*. Prednost ovoga pristupa je bila u eliminisanju posebnih ulazno-izlaznih naredbi i u jedinstvenom načinu tretiranja svih lokacija, i onih koje su pripadale radnoj memoriji, ali i onih koje su pripadale kontrolerima. Mana memorijski preslikanog ulaza izlaza je bila u smanjenju broja lokacija u radnoj memoriji, jer je izvestan broj adresa dodeljen registrima kontrolera.

MULTIPROGRAMIRANJE

Računari treće generacije su stvorili preduslove za bolju iskorišćenost procesora, jer su omogućili istovremeni rad procesora i kontrolera (*SPOOLing*, *Simultaneous Peripheral Operation On Line*). Znači, nakon pokretanja aktivnosti kontrolera, procesor je mogao nastaviti svoju aktivnost, ako ona nije zavisila od rezultata pokrenute aktivnosti kontrolera. Pri tome su kontroleri koristili mehanizam prekida (*interrupt*), da bi obavestili procesor o završetku svoje aktivnosti (mehanizam prekida je sporadično korišćen i u pojedinim računarima prve i druge generacije, ali je postao opšteprisutan tek u računarima treće generacije).

Aktivnosti procesora i kontrolera su mogle uvek biti preklopljene, ako je bilo moguće na koristan način angažovati procesor, kad god bi nastavak njegove aktivnosti postao zavisan od rezultata pokrenute aktivnosti kontrolera. Pristup **multiprogramiranja** (*multiprogramming*) je ponudio sistematičan način da se procesor uvek drži zaposlen u prethodno opisanoj situaciji. Ideja multiprogramiranja je podrazumevala da se u radnoj memoriji uvek istovremeno nalazi više raznih slika procesa i da se procesor preključuje (*context switch*) sa jednog na drugi proces, čim se, u toku aktivnosti prvog procesa, pokrene aktivnost nekog kontrolera od koje je zavisio nastavak aktivnosti tog procesa. Tom prilikom prvi proces je prelazio iz stanja 'aktivan' u stanje 'čeka', a drugi iz stanja 'spreman' u stanje 'aktivan'. Prema tome, dok su postojali spremni procesi, dotle je postojala mogućnost da se procesor korisno angažuje, nakon što pokrene aktivnost nekog od kontrolera. Ali, istovremeno prisustvo više slika procesa u radnoj memoriji je stvorilo mogućnost za međusobne nenamerne ili zlonamerne interakcije procesa. Za takvu interakciju je bilo dovoljno da se u toku aktivnosti jednog od procesa izmeni sadržaj slike drugog procesa. Zato je uspešna primena multiprogramiranja zavisila od postojanja mehanizma zaštite, namenjenog da, u toku aktivnosti procesa, spreči pristupanje lokacijama radne memorije koje ne pripadaju njegovoj slici. Primer

takvog mehanizma zaštite predstavlja razlikovanje logičkog i fizičkog adresnog prostora i korišćenje graničnog i baznog registra za transformaciju logičkih adresa u fizičke. Pored toga, multiprogramiranje je zahtevalo ili proširenje radne memorije ili njeno racionalnije korišćenje, da bi ona mogla da prihvati što više slika procesa, radi što dužeg korisnog angažovanja procesora. Broj istovremeno prisutnih slika procesa u radnoj memorji je nazvan **stepen multiprogramiranja** (*degree of multiprogramming*).

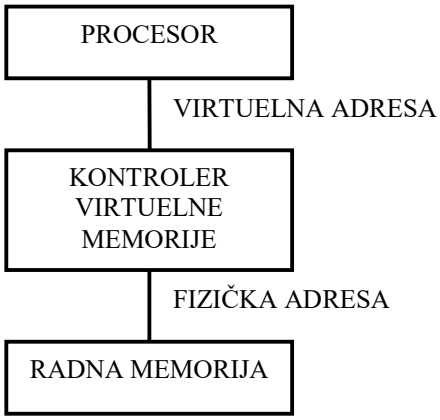
Multiprogramiranje je stvorilo preduslove za ravnomernu raspodelu procesorskog vremena između aktivnosti raznih procesa (*timesharing*). Za ovakvu raspodelu jedino je bilo potrebno izazivati prekide u pravilnim vremenskim intervalima, radi preključivanja procesora sa jednog procesa na drugi. U ovakvim okolnostima, korisnicima su nedostajali još samo terminali za neposrednu komunikaciju sa sopstvenim procesima, radi interpretiranja komandi ili editiranja, na primer. Ovakva vrsta interaktivnog rada nije zahtevala poznavanje detalja funkcionisanja računara treće generacije, nego samo poznavanje pravila korišćenja pojedinih sistemskih programa. Pri tome, zahvaljujući ogromnoj razlici u brzini rada procesora i korisnika, jedan procesor je mogao da opslužи više korisnika, odnosno da, ravnomerno raspodeljujući svoje vreme na svakog od njih, stvori kod korisnika privid da im je računar potpuno posvećen. Ovakva vrsta interaktivnog rada je dovela do raširene upotrebe znakovnih video terminala. Pored toga, ona je omogućila korisnicima da međusobno dele skupe uređaje, kao što su ploteri, na primer, ali je i olakšala razmenu programa i podataka između korisnika, što je stvorilo uslove za razne oblike međusobne saradnje korisnika.

IDEJA VIRTUELNE MEMORIJE

Ograničena veličina radne memorije računara treće generacije, izazvana praktičnim razlozima, nije sprečila širenje primene multiprogramiranja i interaktivnog rada samo zahvaljujući osobini **lokalnosti izvršavanja** programa, odnosno zapažanju da u toku izvršavanja programa procesor ne pristupa podjednako svim lokacijama programa. Praćenje izvršavanja programa je pokazalo da je verovatnoća pristupanja lokacijama programa izrazito neravnomerna, odnosno da je za neke lokacije ubedljivo veća, nego za druge. Tako je zapaženo da, približno, 90% vremena izvršavanja programa otpada na izvršavanje njegovih 10% naredbi, a sličan odnos važi i za pristupanje lokacijama sa podacima. Lokalnost može biti prostorna i vremenska. **Prostorna lokalnost** (*spatial locality*) izvršavanja programa je vezana za izvršavanja delova programa u toku kojih procesor pristupa lokacijama sa uzastopnim adresama. Za ove lokacije se, zbog uzastopnih adresa, kaže da obrazuju **stranicu** (*page*). Prostorna lokalnost je uzrokovana, između ostalog, sekvencijalnim izvršavanjem programa i organizovanjem podataka u nizove i slogove. **Vremenska lokalnost** (*temporal locality*) izvršavanja programa je vezana za izvršavanja delova programa u toku kojih procesor ponavlja pristupanje pojedinim lokacijama iz iste stranice. Ova vrsta lokalnosti je uzrokovana, između

ostalog, postojanjem petlji (*loop*) u programima. Zahvaljujući osobini lokalnosti izvršavanja programa, u toku pojedinih perioda aktivnosti procesa, procesor pristupa samo podskupu svih lokacija slike procesa, obrazovanom od jedne ili više stranica. Ovakav podskup lokacija je nazvan **radni skup** (*working set*). Iskustvo je pokazalo da se radni skup sporo menja u toku vremena. Prema tome, za aktivnost procesa nije potrebno da sve stranice njegove slike budu u radnoj memoriji, nego je dovoljno da se u radnoj memoriji nalaze samo stranice koje obrazuju radni skup. To dalje znači da stepen multiprogramiranja nije određen brojem slika procesa koji istovremeno mogu da stanu u radnu memoriju, nego brojem njihovih radnih skupova koje radna memorija može istovremeno da sadrži. Pri tome se podrazumeva da su slike procesa smeštene u masovnoj memoriji i da se, po potrebi, kopije sadržaja njihovih stranica prebacuju između masovne i radne memorije, kada postanu deo radnog skupa, odnosno kada prestanu biti deo radnog skupa. Pošto je jedinica prenosa između masovne i radne memorije blok, prebacivane kopije sadržaja stranica obuhvataju jedan ili više blokova.

Iz činjenice da se u toku aktivnosti procesa cela njegova slika ne nalazi u radnoj memoriji, nego u masovnoj memoriji, sledi da veličina slike nije ograničena veličinom radne, nego veličinom masovne memorije. Pošto je masovna memorija veća od radne, znači da fizički adresni prostor radne memorije nije dovoljan za adresiranje svih lokacija slike procesa. Zato se uvodi prividni ili **virtuelni** (*virtual*) **adresni prostor**, koga koristi procesor. Veličinu **virtuelnih adresa** (iz virtuelnog adresnog prostora) određuje format mašinskih naredbi procesora. Prema tome, virtuelni adresni prostor ograničava veličinu slika procesa, a fizički adresni prostor ograničava veličinu radnih skupova. Podrazumeva se da su virtuelni i fizički adresni prostori izdeljeni na stranice iste veličine, nazvane (respektivno) **virtuelne** i **fizičke stranice**. Za aktivnost procesora je neophodno da u fizičkim stranicama budu kopije virtuelnih stranica koje obrazuju radni skup, jer procesor može da pristupa samo lokacijama radne memorije. Postojanje dva adresna prostora zahteva da svaka virtuelna adresa, upućena iz procesora, bude pretvorena (*address mapping*) u fizičku pre nego stigne do radne memorije, odnosno do sabirnice. Za ovo pretvaranje je zadužen **kontroler virtuelne memorije** (*MMU*, *Memory Management Unit*). Slika 10.1.6 sadrži prikaz položaja kontrolera virtuelne memorije u organizaciji računara.



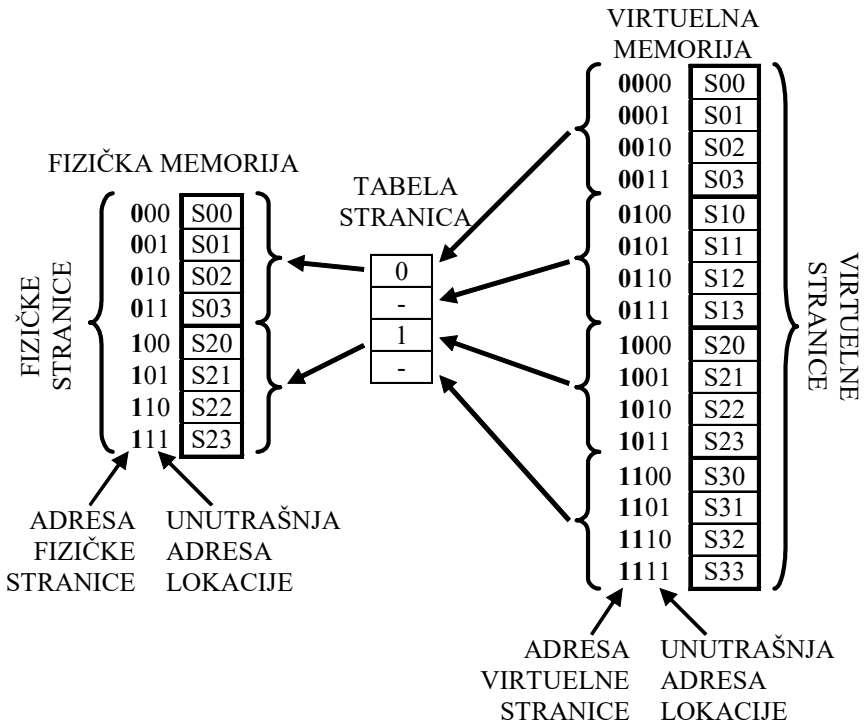
Slika 10.1.6 Položaj kontrolera virtuelne memorije u organizaciji računara

Pretvaranje virtuelne adrese u fizičku ima smisla samo ako neka fizička stranica sadrži kopiju virtuelne stranice na koju se odnosi pomenuta virtuelna adresa. To se utvrđuje na osnovu evidencije o kopijama virtuelnih stranica smeštenim u fizičke stranice. Ako evidencija pokaže da u fizičkim stranicama nema kopije tražene virtuelne stranice, tada je neophodno prebaciti sa masovne memorije kopiju dotične virtuelne stranice u neku od slobodnih fizičkih stranica radne memorije. U slučaju da su sve fizičke stranice zauzete, neka od njih se oslobađa. Ako oslobađana fizička stranica sadrži izmenjenu kopiju virtuelne stranice, pre oslobađanja se izmenjena kopija prebacuje na masovnu memoriju da bi zamenila izvornu virtuelnu stranicu. Pristup, u kome se kopije stranica prebacuju po zahtevu (*demand paging*), podrazumeva izazivanje posebnog **straničnog prekida** (*page fault*), kad god se ustanovi da fizičke stranice ne sadrže kopiju potrebne virtuelne stranice. Izazivanje straničnog prekida dovodi do aktiviranja obrađivača straničnog prekida, zaduženog za prebacivanje kopija virtuelnih stranica. Podrazumeva se da obrađivač straničnog prekida iz adrese virtuelne stranice određuje broj bloka koji je dodeljen dotičnoj virtuelnoj stranici.

Pretvaranje virtuelne adrese u fizičku se zasniva na činjenici da svaka lokacija pripada samo jednoj stranici i u virtuelnom i u fizičkom adresnom prostoru, i da se adresa svake lokacije sastoji od **adrese virtuelne** ili **fizičke stranice** (*page number*), kojoj lokacija pripada, i od **unutrašnje adrese** (*offset*) lokacije u ovoj stranici. Pošto su virtuelna i fizička stranica iste veličine, njihove unutrašnje adrese su jednake, pa je za pretvaranje virtuelne adrese u fizičku potrebno samo zameniti adresu virtuelne stranice adresom fizičke stranice koja sadrži kopiju dotične vitruelne stranice. Za ovo pretvaranje, kao i za evidenciju kopija virtuelnih stranica prisutnih u fizičkim stranicama, potrebna je **tabela stranica** (*page table*). Njeni elementi uspostavljaju korespondenciju između virtuelne stranice i fizičke stranice

koja sadrži njenu kopiju. Broj elemenata tabele stranica je isti kao i broj virtuelnih stranica, pa adresa virtuelne stranice služi kao indeks odgovarajućeg elementa tabele stranica. U njemu se nalazi ili adresa korespondentne fizičke stranice ili oznaka da dotičnoj virtuelnoj stranici ne korespondira fizička stranica.

Slika 10.1.7 sadrži prikaz primera podele virtuelne i radne memorije u virtuelne i fizičke stranice, kao i slučaj preslikavanja virtuelnih stranica sa adresama 00 i 10 u fizičke stranice sa adresama 0 i 1.



Slika 10.1.7 Tabela stranica (sve adrese su binarni brojevi)

Slika 10.1.7 pokazuje da fizička stranica 0 sadrži kopiju virtuelne stranice 00 i da fizička stranica 1 sadrži kopiju virtuelne stranice 10. Kada procesor zatraži pristup lokaciji sa virtuelnom adresom 1011 (lokaciji sa sadržajem S23), kontroler virtuelne memorije konsultuje element tabele stranica sa indeksom 10 i u njemu pronađe adresu fizičke stranice 1. Nakon zamene adrese virtuelne stranice 10 pronađenom adresom korespondentne fizičke stranice 1 od virtuelne adrese 1011 nastane fizička adresa 111. Ona adresira lokaciju sa kopijom traženog sadržaja S23. Da procesor zatraži pristup lokaciji sa virtuelnom adresom 0110, kontroler virtuelne memorije bi ustanovio da u elementu tabele stranica sa indeksom 01 nema adrese fizičke stranice, pa bi izazvao stranični prekid. Aktivirani obrađivač prekida bi

oslobodio jednu, na primer, najranije zauzetu fizičku stranicu i njenu fizičku adresu izbacio iz tabele stranica. Obrađivač prekida bi, zatim, obezbedio da u oslobođenu fizičku stranicu bude smeštena kopija potrebne virtuelne stranice, a u odgovarajući element tabele stranica bi smestio adresu oslobođene fizičke stranice. Nakon obrade straničnog prekida, procesor bi ponovo zatražio pristup lokaciji sa virtuelnom adresom 0110, a kontroler virtuelne memorije bi ponovo započeo pretvaranje date virtuelne adrese u fizičku adresu.

Pretvaranje virtuelne adrese u fizičku usporava procesor, odnosno smanjuje broj izvršavanih mašinskih naredbi u jedinici vremena, jer se tabela stranica nalazi u radnoj memoriji, pa rukovanje tabelom stranica zahteva dodatne pristupe radnoj memoriji. Zato je važno da pristupanje tabeli stranica bude kratkotrajno, pošto od njega zavisi dužina pretvaranja virtuelne adrese u fizičku. To nije bilo moguće postići bez korišćenja brze poluprovodničke memorije. Njena veličina, ograničena cenom, je nedovoljna za smeštanje cele tabele stranica, ali je dovoljna za čuvanje kopija nekoliko njenih elemenata. Da bi se znalo koje kopije elemenata tabele stranica se čuvaju u lokacijama brze memorije, uz njih se, u tim istim lokacijama, čuvaju i indeksi pomenutih elemenata. Prema tome, lokacije brze memorije sadrže adrese korespondentnih parova virtuelnih i fizičkih stranica. Znači, za svaku virtuelnu adresu, nastalu u procesoru, može se proveriti da li se adresa njene virtuelne stranice nalazi u nekoj od lokacija brze memorije. Ako se pronađe takva lokacija, tada ona sadrži i adresu korespondentne fizičke stranice. Pošto se za pristupanje adresama fizičkih stranica iz lokacija brze memorije ne koriste adrese lokacija brze memorije, nego delovi sadržaja ovih lokacija (adrese virtuelnih stranica, prisutnih u njima), ovakva brza memorija je nazvana **asocijativna memorija** (*associative memory, translation lookaside buffer*). Slika 10.1.8 sadrži prikaz primera asocijativne memorije sa 2 lokacije. Sadržaj njenih lokacija odgovara tabeli stranica iz primera koga sadrži Slika 10.1.7.

DVOBITNE ADRESE VIRTUELNIH STRANICA	JEDNOBITNE ADRESE KORESPONDENTNIH FIZIČKIH STRANICA
00	0
10	1

Slika 10.1.8 Asocijativna memorija sa dve trobitne lokacije

Osobina lokalnosti izvršavanja programa omogućuje da se, i uz malu veličinu asocijativne memorije, u njoj dovoljno često pronalaze adrese virtuelnih stranica i njima korespondentne adrese fizičkih stranica. Samo kada adresa virtuelne stranice ne bude pronađena u asocijativnoj memoriji, ona služi kao indeks elementa tabele stranica, smeštene u radnoj memoriji. Ovakvo pristupanje lokacijama radne

memorije, u proseku, suviše ne produžava pretvaranje virtuelne adrese u fizičku, ako je retko, na primer, ako se dešava samo u 10 od 100 slučajeva. Uz ovakvu pretpostavku, **verovatnoća pogotka** (*hit*), odnosno verovatnoća pronalaženja adrese virtuelne stranice u asocijativnoj memoriji je 0.9, a **verovatnoća promašaja** (*miss*) je 0.1. Ako se prihvati da pristupanje asocijativnoj memoriji traje 1 procesorski ciklus, a da pristupanje radnoj memoriji traje 10 procesorskih ciklusa, tada srednje vreme pronalaženja adrese fizičke stranice iznosi:

$$(90\times1+10\times10)/100==0.9\times1+0.1\times10 == 1.9$$

procesorskih ciklusa. U slučaju promašaja, sadržaj asocijativne memorije se ažurira, da bi primio adresu virtuelne stranice, koja je izazvala promašaj, kao i korespondentnu adresu fizičke stranice.

Zahvaljujući tome što u svom sastavu ima asocijativnu memoriju, kontroler virtuelne memorije može dovoljno brzo da obavi pretvaranje virtuelne adrese u fizičku i tako da omogući automatizaciju memorijske hijerarhije, sastavljene od radne i masovne memorije. Preduslov za ovakvu automatizaciju je oslanjanje masovne memorije na tehnologiju magnetnih diskova i mogućnost direktnog pristupanja blokovima diska.

Postavljanje kontrolera virtuelne memorije između procesora i radne memorije je olakšalo međusobnu zaštitu programa, izvršavanih u režimu multiprogramiranja, jer se iste virtuelne adrese iz raznih programa pretvaraju u različite fizičke adrese, pa nema mogućnosti za pristupanje tuđim lokacijama radne memorije. Da bi tabela stranica bila nepristupačna za neovlaštene izmene, neophodno je uvesti privilegovani i neprivilegovani režim rada procesora.

Pojava memorijske hijerarhije je bila rezultat potrebe za memorijom velikog kapaciteta (izraženog brojem bajta), kratkog vremena pristupa (bajtu ili grupi bajta) i niske cene (po jednom bajtu). Pošto su vreme pristupa i cena bili obrnuto proporcionalni, bez memorijske hijerarhije veliki kapacitet su mogle praktično imati samo memorije sa dugačkim vremenom pristupa, jer su jedino one imale prihvatljivo nisku cenu. Međutim, obrazovanje memorijske hijerarhije, sastavljene od memorije velikog kapaciteta i dugačkog vremena pristupa i od memorije malog kapaciteta i kratkog vremena pristupa, je omogućilo da se, po prihvatljivoj ceni, dobiju memorije velikog kapaciteta, a kratkog srednjeg vremena pristupa. Srednje vreme pristupa je kratko, ako sadržaj memorije malog kapaciteta i kratkog vremena pristupa dovoljno često zadovoljava potrebe izvršavanja programa. To se može postići zahvaljujući osobini lokalnosti izvršavanja programa, koja obezbeđuje da se retko pristupa memoriji velikog kapaciteta i dugačkog vremena pristupa. Na taj način mogu da se pomire krajnosti. To potvrđuje i primer virtuelne memorije, čiji kapacitet je srazmeran kapacitetu masovne memorije, a čije srednje vreme pristupa je blisko vremenu pristupa radne memorije. Jasno, prethodno važi pod uslovom da sadržaj radne memorije dovoljno često zadovoljava potrebe izvršavanja programa,

što je moguće zahvaljujući osobini lokalnosti izvršavanja programa. Tako, na primer, ako se potrebni sadržaji pronalaze u radnoj memoriji prosečno u 9999 od 10000 slučajeva, znači, ako je verovatnoća pogotka 0.9999, a verovatnoća promašaja 0.0001, i ako je vreme pristupa radnoj memoriji 10 procesorskih ciklusa, a vreme pristupa masovnoj memoriji 10000 procesorskih ciklusa, tada srednje vreme pristupa virtuelnoj memoriji iznosi:

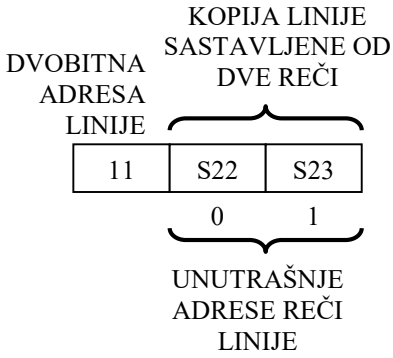
$$0.9999\times10+0.0001\times10000 == 10.999$$

procesorskih ciklusa.

IDEJA SKRIVENE MEMORIJE

Važno je uočiti da je ideja memorijske hijerarhije našla primenu i u kontroleru virtuelne memorije, radi skraćenja srednjeg vremena pretvaranja virtuelne adrese u fizičku, odnosno, radi skraćenja srednjeg vremena pronalaženja adrese fizičke stranice. U ovom slučaju hijerarhiju su činile asocijativna memorija iz kontrolera virtuelne memorije i radna memorija sa tabelom stranica. Pristup, sličan prethodnom, je doveo do obrazovanja memorijske hijerarhije, sastavljene od brze poluprovodničke memorije i radne memorije, s ciljem skraćenja srednjeg vremena pristupa radnoj memoriji. Taj cilj se može ostvariti, ako se u lokacijama brze memorije dovoljno često nalaze kopije sadržaja lokacija radne memorije, potrebne za izvršavanje programa, pa dovoljno često izostaje pristupanje radnoj memoriji. Za brzu memoriju, u ovom slučaju, se koristi naziv **skrivena memorija** ili **keš** (*cache*) memorija, jer njeno postojanje nije vidljivo za programe, odnosno, ne zahteva njihove izmene i jedino utiče na skraćenje vremena njihovog izvršavanja. Skrivena i radna memorija se dele u **linije** (*line*). Svaka od ovih linija se sastoji od istog broja lokacija sa uzastopnim adresama. Između skrivene i radne memorije prebacuju se kopije sadržaja celih linija, jer to sugeriše prostorna lokalnost izvršavanja programa, koja uzrokuje povećavanje verovatnoće pristupanja sadržajima lokacija sa uzastopnim adresama. Prema tome, adresa svake lokacije se sastoji od **adrese linije** (*line number*), kojoj lokacija pripada, i od unutrašnje adrese lokacije u svojoj liniji (*offset*).

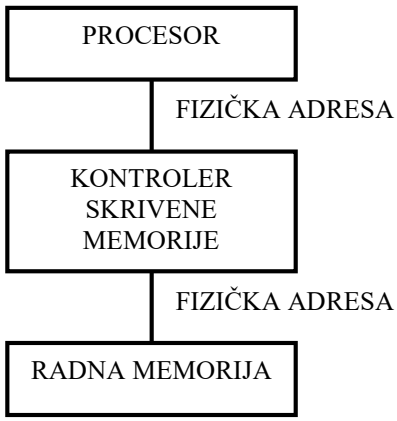
Skrivena memorija je organizovana kao asocijativna memorija, jer ima suviše malo lokacija da bi se adresa linije sa traženom lokacijom radne memorije mogla koristiti za indeksiranje lokacije skrivene memorije. To znači da lokacije skrivene memorije čuvaju ne samo kopiju sadržaja linije, nego i njenu adresu (Slika 10.1.9).



Slika 10.1.9 Skrivena memorija sa jednom lokacijom

Slika 10.1.9 sadrži primer skrivene memorije sa jednom lokacijom u kojoj su adresa linije i kopija sadržaja lokacija sa adresama 110 i 111 (Slika 10.1.7).

Za svaku adresu lokacije radne memorije, emitovanu iz procesora, proverava se da li skrivena memorija sadrži adresu linije sa ovom lokacijom. Ako se u nekoj lokaciji skrivene memorije pronađe adresa pomenute linije, tada ta lokacija sadrži kopiju ove linije, pa nije potreban pristup radnoj memoriji. U suprotnom slučaju, neizbežan je pristup lokaciji radne memorije, kao i ažuriranje skrivene memorije, da bi u njoj uvek bile kopije poslednje korišćenih sadržaja lokacija radne memorije. Skrivena memorija se nalazi u **kontroleru skrivene memorije**, zaduženom za upravljanje memorijskom hijerarhijom, sastavljenom od skrivene i radne memorije (Slika 10.1.10).

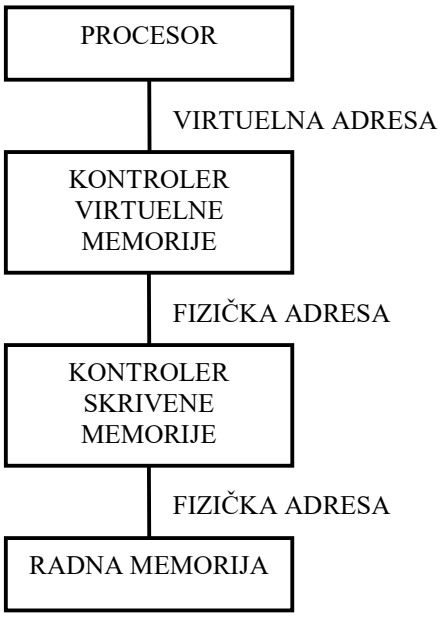


Slika 10.1.10 Položaj kontrolera skrivene memorije u organizaciji računara

ODNOS VIRTUELNE I SKRIVENE MEMORIJE

Između virtuelne i skrivene memorije postoji principijelna sličnost, jer i jedna i druga smanjuju srednje vreme pristupa lokacijama slike procesa. Međutim, između njih postoje i razlike, na primer u pogledu sadržaja lokacija asocijativnih memorija, ili u pogledu reakcije na promašaj prilikom pretraživanja asocijatinvih memorija.

Skrivena memorija se može kombinovati sa virtuelnom memorijom. U tom slučaju ona skraćuje srednje vreme pristupa virtuelnoj memoriji (Slika 10.1.11).



Slika 10.1.11 Kombinacija virtuelne i skrivene memorije

Postavljanje kontrolera skrivene memorije između sabirnice i kontrolera virtuelne memorije omogućuje da do skrivene memorije stižu samo fizičke adrese, čime se izbegava problem pojave višeznačnih virtuelnih adresa u skrivenoj memoriji.

Princip virtuelne memorije, razvijene za računar druge generacije, opisao je *J. Fotheringham* 1961. godine. Upotrebu virtuelne memorije raširili su računari treće generacije, pre svih računari iz *IBM 370* familije. Skrivenu memoriju je predložio *Maurice Wilkes* 1965. godine. Ona je prvi put primenjena 1968. godine u računaru *IBM 360/85*.

OPERATIVNI SISTEM I VIRTUELNA MAŠINA

Koncepti multiprogramiranja, interaktivnog rada i virtuelne memorije su zahtevali veoma složene operativne sisteme, zadužene, na primer, za reagovanje na prekide, za preključivanje, kao i za prebacivanje kopija sadržaja stranica između masovne i radne memorije. Za operativne sisteme računara treće generacije su vezani pojava **sistema datoteka** (*file system*), uvođenje praćenja korišćenja računara treće generacije, radi naplaćivanja usluga, kao i uvođenje pojma **virtuelne mašine** (*virtual machine*). Ovaj pojam je imao dva značenja. Prvo od njih se odnosilo na radno okruženje koje su, sa jedne strane, stvarali operativni sistem, odnosno njegovi sistemski pozivi i interpreter komandnog jezika, a sa druge strane sistemski programi, poput editora, na primer. Drugo značenje ovoga pojma se odnosilo na stvaranje privida da je svakom korisniku raspoloživ primerak celog računara, odnosno njegova cela arhitektura naredbi. Operativnim sistemima računara treće generacije pripada, na primer, operativni sistem *UNIX*, koga su razvili *Ken Thompson* i *Dennis Ritchie*. *UNIX* je razvijen za mini-računare. Za potrebe njegovog razvoja je napravljen programski jezik *C*, čiji je autor *Dennis Ritchie*.

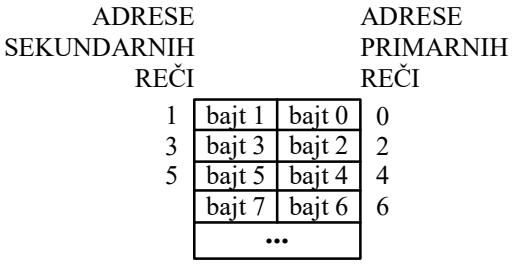
MANA RAČUNARA TREĆE GENERACIJE

Ozbiljna mana računara treće generacije se odnosila na nepredvidivost odziva u interaktivnom radu, što je negativno uticalo na produktivnost programera. Odziv je bio nepredvidiv, jer je zavisio od broja programa na čije izvršavanje je raspodeljivano procesorsko vreme.

10.2. ISHODI EVOLUCIJE ARHITEKTURE RAČUNARA U PERIODU OKO 1970. GODINE

RADNA MEMORIJA

Osnovna organizaciona jedinica radne memorije je osmobitni bajt. Iako svaki bajt poseduje jedinstvenu adresu, jedinica pristupa radnoj memoriji nije bajt, nego reč. Veličina reči je obično jednaka veličini procesorskog registra, da bi se odjednom, u jednom navratu, mogao prebaciti sadržaj celog registra između radne memorije i procesora. Slika 10.2.1 sadrži prikaz organizacije radne memorije, sastavljene od dvobajtnih reči.



Slika 10.2.1 Organizacija radne memorije, sastavljene od dvobajtnih "*little endian*" reči

Reči radne memorije, kojima se može pristupiti u jednom navratu, se nazivaju **primarne reči** (*aligned words*). Za radnu memoriju iz prethodnog primera (Slika 10.2.1) to su reči sa adresama 0, 2, 4, koje obuhvataju bajte 0 i 1, 2 i 3, 4 i 5. Adrese ovih reči su deljive sa 2 bez ostatka. Pristupanje preostalim **sekundarnim rečima** (*non-aligned words*) sa adresama 1, 3, 5, koje obuhvataju bajte 1 i 2, 3 i 4, 5 i 6 (Slika 10.2.1), se odvija u dva navrata. U svakom od njih se pristupi odgovarajućoj primarnoj reči, radi preuzimanja jednog od traženih bajta. Na primer, pristupanje sekundarnoj reči sa adresom 1 zahteva pristupanje primarnim rečima sa adresama 0 i 2, radi dobavljanja bajta sa adresama 1 i 2. Prethodno opisani princip važi i kada je radna memorija računara sastavljena od reči sa 4 ili 8 bajta. Kod ovakvih radnih memorija adrese primarnih reči su deljive bez ostatka sa 4 ili 8.

Ograničavanje pristupa samo na primarne reči radne memorije pojednostavljuje procesor, jer tada nije potreban automatizam za ostvarenje pristupanja sekundarnim rečima.

ARHITEKTURA NAREDBI ZA *IBM SYSTEM/360*

Arhitektura naredbi za *IBM SYSTEM/360* praktično podržava adresni prostor od 24 bita, koji je organizovan u bajte, sa pristupom samo primarnim rečima. Ona omogućuje rukovanje binarnim vrednostima velikim 1, 2 i 4 bajta, realnim vrednostima od 8 bajta (*floating-point*), decimalnim ciframa izraženim kao četvorobitne vrednosti (*packed decimal*) i nizovima znakova (*string*). U ovoj arhitekturi su na raspolaganju 16 registara od 32 bita (registri opšte namene), 4 registra od 64 bita za realne brojeve (*floating-point*) i status registar (*program status word*), koji sadrži programski brojač i uslovne bite (*condition codes*). Ova arhitektura omogućuje privilegovani način rada procesora (*supervisory mode*). Ona podržava 5 adresiranja, koji predstavljaju kombinacije neposrednog, registarskog i indeksnog adresiranja.

Mašinski formati naredbi ove arhitekture su promenljivi. Njihova polja Ri sadrže kod registra i:

8 bita	4 bita	4 bita
kod nar.	R1	R2

Za prethodni RR (*register-register*) mašinski format naredbi se podrazumeva da se 1. operand naredbe nalazi u registru R1, da se 2. operand naredbe nalazi u registru R2 i da se rezultat naredbe smešta u registar R1.

8 bita	4 bita	4 bita	4 bita	12 bita
kod nar.	R1	R2	R3	odstojanje

Za prethodni RX (*register-indexed*) mašinski format naredbi se podrazumeva da se 1. operand naredbe nalazi u registru R1, da se 2. operand naredbe nalazi u memorijskoj lokaciji sa adresom R2+R3+odstojanje i da se rezultat naredbe smešta u registar R1.

8 bita	4 bita	4 bita	4 bita	12 bita
kod nar.	R1	R2	R3	odstojanje

Za prethodni RS (*register-storage*) mašinski format naredbi se podrazumeva da se 1. operand naredbe nalazi u memorijskoj lokaciji sa adresom R3+odstojanje, da se 2. operand naredbe nalazi u registru R2 i da se rezultat naredbe smešta u registar R1.

8 bita	8 bita	4 bita	12 bita
kod nar.	vrednost	R1	odstojanje

Za prethodni SI (*storage-immediate*) mašinski format naredbi se podrazumeva da se vrednost smešta u memorijsku lokaciju sa adresom R1+odstojanje.

8 bita	8 bita	4 bita	12 bita	4 bita	12 bita
kod nar.	dužina	R1	odstojanje1	R2	odstojanje2

Za prethodni SS (*storage-storage*) mašinski format naredbi se podrazumeva da se 1. operand naredbe nalazi u memorijskoj lokaciji sa adresom R1+odstojanje1, da se 2. operand naredbe nalazi u memorijskoj lokaciji sa adresom R2+odstojanje2 i da se rezultat naredbe smešta u memorijsku lokaciju sa adresom R1+odstojanje1 (dužina označava broj memorijskih lokacija na koje se naredbe primenjuje).

Sve naredbe ne koriste sva polja iz prikazanih mašinskih formata naredbi na prethodno opisani način. U takve naredbe spadaju upravljačke naredbe. Na primer, kod uslovnih upravljačkih naredbi RX formata, polje R1 se koristi za izražavanje uslova, a suma sadržaja R2, sadržaja R3 i odstojanja (*displacement*) se smešta u programski brojač, ako je uslov ispunjen.

Arhitektura naredbi za *IBM SYSTEM/360* obuhvata sledeće tipove naredbi:

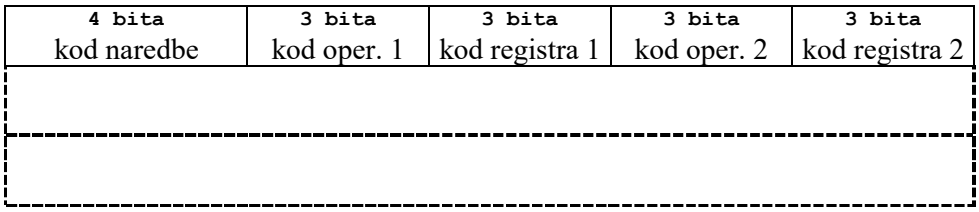
1. naredbe za rukovanje podacima (na primer, za prenos i konverziju podataka),
2. naredbe za rukovanje bitima (logičke naredbe i naredbe pomeranja),
3. naredbe za celobrojnu binarnu aritmetiku (obuhvaćene sve aritmetičke operacije),
4. naredbe za celobrojnu decimalnu aritmetiku (obuhvaćene sve aritmetičke operacije),
5. naredbe za aritmetiku realnih brojeva (obuhvaćene sve aritmetičke operacije),
6. upravljačke naredbe,
7. sistemske naredbe i
8. ulazno-izlazne naredbe.

ARHITEKTURA NAREDBI ZA *DEC PDP11*

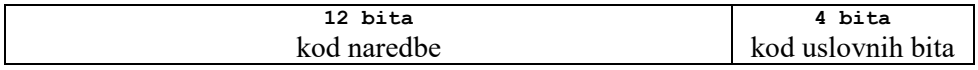
Arhitektura naredbi za *DEC PDP11* podržava adresni prostor od 16 bita, koji je organizovan u bajte, sa pristupom samo primarnim rečima. Ona omogućuje rukovanje binarnim vrednostima velikim 1 i 2 bajta, kao i realnim vrednostima (*floating-point*) velikim 4 i 8 bajta. U ovoj arhitekturi su na raspolaganju 8 registara od 16 bita (registri opšte namene), od kojih jedan služi kao pokazivač steka (*stack pointer*), a drugi kao programski brojač. Pored njih, postoji i status registar (*status word*), koji sadrži uslovne bite (*condition codes*). Ova arhitektura omogućuje privilegovani način rada procesora sa 2 nivoa privilegija (*kernel state, supervisor state*). Ona podržava registarsko adresiranje, posredno adresiranje sa samouvećanjem, posredno adresiranje sa samoumanjenjem, indeksno adresiranje, posredno adresiranje, dvostruko posredno adresiranje sa samouvećanjem, dvostruko posredno adresiranje sa samoumanjenjem i indeksno posredno adresiranje. Sva adresiranja se oslanjanju na registre. Adresiranja sa samouvećanjem (samoumanjenjem) podrazumevaju da se sadržaj korišćenog registra automatski uveća (umanji) nakon adresiranja. Dvostruka indirekcija znači da korišćeni registar sadrži adresu memorijske lokacije sa adresom operanda. Indeksna indirekcija znači da je rezultat indeksnog adresiranja adresa memorijske lokacije sa adresom operanda. Kombinovanje programskog brojača sa posrednim adresiranjem sa samouvećanjem odgovara neposrednom adresiranju. Slično, kombinovanje programskog brojača sa dvostruko posrednim adresiranjem sa samouvećanjem odgovara apsolutnom adresiranju (u kome se koristi apsolutna adresa). Kombinovanje programskog brojača sa indeksnim adresiranjem odgovara relativnom adresiranju, a kombinovanje programskog brojača sa indeksnim posrednim adresiranjem odgovara relativnom posrednom adresiranju.

Mašinski formati naredbi ove arhitekture su promenljivi:

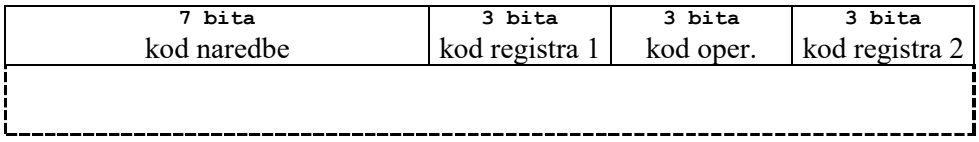
Za prethodni mašinski format naredbi jedini operand je određen kodom operanda (koji određuje adresiranje) i kodom registra (koji određuje registar na koga se oslanja navedeno adresiranje). Zavisno od adresiranja, ovaj mašinski format može zauzeti 2 ili 4 bajta.



Za prethodni mašinski format naredbi 1. operand je određen kodom operanda 1 i kodom registra 1, 2. operand je određen kodom operanda 2 i kodom registra 2, a rezultat naredbe se smešta na odredište koje određuje 2. operand. Zavisno od adresiranja, ovaj mašinski format može zauzeti 2, 4 ili 6 bajta.



Za prethodni mašinski format naredbi se podrazumeva da jedini operand označava uslovne bite koje naredba postavlja na 1 ili na 0.



Za prethodni mašinski format naredbi 1. operand je određen kodom registra 1 (podrazumeva se registarsko adresiranje), 2. operand je određen kodom operanda i kodom registra 2, a rezultat naredbe se smešta na odredište koje određuje 1. ili 2. operand, zavisno od naredbe. Kada ovaj mašinski format odgovara naredbi poziva potprograma, tada kod registra 1 određuje registar u koji se smešta povratna adresa, a kod operanda i kod registra 2 zajedno određuju adresu ciljne naredbe. Zavisno od adresiranja, ovaj mašinski format može zauzeti 2 ili 4 bajta.

13 bita	3 bita
kod naredbe	kod registra

Prethodni mašinski format naredbi odgovara naredbi povratka iz potprograma. Kod registra određuje registar sa povratnom adresom.

8 bita	8 bita
kod naredbe	relativna adresa

Prethodni mašinski format odgovara upravljačkim naredbama za koje se podrazumeva relativno adresiranje. Znači ciljna adresa se određuje kao zbir relativne adrese (*offset*) i sadržaja programskog brojača.

Arhitektura naredbi za *DEC PDP11* obuhvata sledeće tipove naredbi:

1. naredbe za rukovanje podacima (na primer, za prenos i konverziju podataka),
2. naredbe za rukovanje bitima (logičke naredbe i naredbe pomeranja),
3. naredbe za celobrojnu binarnu aritmetiku (obuhvaćene sve aritmetičke operacije),
4. naredbe za aritmetiku realnih brojeva (obuhvaćene sve aritmetičke operacije),
5. upravljačke naredbe i
6. sistemske naredbe.

OSOBINE MAGNETNOG DISKA

Magnetne diskove karakterišu kapacitet i srednje vreme pristupa. Kapacitet diska je određen brojem staza i njihovom iskorišćenošću. Kod formatiranog diska cela staza se ne koristi za smeštanje podataka, pa je zato kapacitet formatiranog diska manji oko 15% od kapaciteta neformatiranog diska. Na stazi formatiranog diska podaci se nalaze u sektorima. Pored sektora sa podacima, na stazi ispred svakog sektora nalazi se prethodnica (*preamble*), a iza svakog sektora sledi kod za korekciju grešaka (*error-correcting code, ecc*) i međusektorski razmak. Prethodnica služi da pripremi glavu diska za pristup sektoru. Kod za korekciju grešaka omogućuje otkrivanje i ispravku eventualnih grešaka po principu Hamingovog koda.

Srednje vreme pristupa (bloku) diska (*average disk access time*) zavisi od **srednjeg vremena pomeranja** (*seek time*) glave diska iznad staza, od **srednjeg vremena rotacije** (*rotational delay*) sektora ispod glave diska, od **vremena prenosa** (*transfer time*) bloka između disk jedinice i njenog kontrolera, kao i od **vremena kontrolera** (*controler time*), koje on troši pri prenosu bloka od i do radne memorije. Prema tome, srednje vreme pristupa diska je jednako sumi srednjeg vremena pomeranja, srednjeg vremena rotacije, vremena prenosa i vremena kontrolera.

ORGANIZACIJA SABIRNICE

Zadatak sabirnice je da poveže razne organizacione komponente računara, radi ostvarenja razmene podataka između pojedinih parova ovih komponenti. Zato u sastav sabirnice ulaze, ne samo linije koje povezuju organizacione komponente računara i elektronika koja omogućuje razmenu signala po linijama, nego i upravljačka logika, koja uređuje pristup linijama sabirnice i razmenu signala po tim linijama.

U svakom od parova organizacionih komponenti, koji razmenjuju podatke posredstvom sabirnice, jedna od komponenti ima ulogu **aktivne strane** (*bus maste*r), a druga ulogu **pasivne strane** (*bus slave*). Ista komponenta u raznim situacijama može da bude i pasivna i aktivna strana (na primer, *DMA* kontroler). Aktivna strana započinje razmenu podataka po **zauzimanju sabirnice**, a pasivna strana učestvuje u ovoj razmeni. Pri tome, zavisno od smera prenosa, postoje dve vrste razmene podataka: **transakcija pisanja** i **transakcija čitanja**. Kod transakcije pisanja, aktivna strana upućuje ka sabirnici adresu podatka, oznaku vrste transakcije i podatak, a kod transakcije čitanja, aktivna strana upućuje na sabirnicu samo adresu podatka i oznaku vrste transakcije. Pasivna strana preuzima sa sabirnice adresu da bi na osnovu nje ustanovila da li se transakcija odnosi na nju. Ako se transakcija odnosi na nju, ona na osnovu oznake vrste transakcije određuje šta se od nje traži. Za transakciju pisanja, ona samo preuzima podatke sa sabirnice. Za transakciju čitanja ona na sabirnicu upućuje adresirani podatak, koga preuzima aktivna strana. Transakcije čitanja su podeljene (*split transaction*), ako sabirnicu mogu da koriste druge aktivna i pasivna strana, nakon što je prva aktivna strana isporučila svojoj pasivnoj strani adresu podatka i oznaku vrste transakcije, a pre nego prva aktivna strana dobije adresirani podatak.

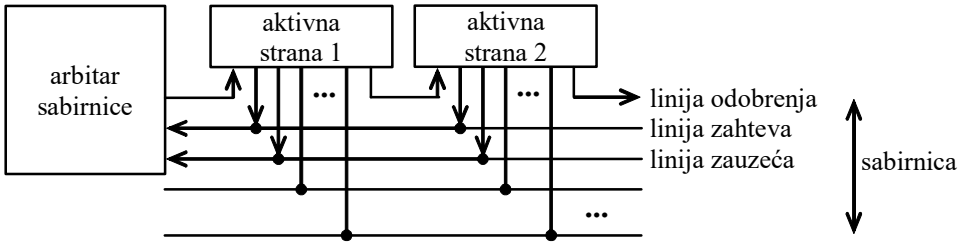
Osobine sabirnice, kao što su broj i vrsta njenih linija, njena propusnost (broj bajta koje ona može da prenese u jedinici vremena), kao i najveći broj organizacionih komponenti koji mogu istovremeno biti na nju zakačeni, zavise od vrste komponenti koje sabirnica povezuje, kao i od njihove međusobne udaljenosti. Ako su unapred poznate karakteristike ovih komponenti i njihove međusobne udaljenosti, tada je vreme sabirnice moguće podeliti na intervale podjednake dužine, tako da transakcije traju unapred zadani broj ovakvih intervala. Sabirnice, kod kojih je aktivnost za njih vezanih komponenti usklađena sa počecima pojedinih intervala, se nazivaju **sinhrone sabirnice** (*synchronous bus*). One u svom sastavu imaju posebnu upravljačku liniju, za prenos sinhronizacionog signala (sa sata). Svaki od ovih signala označava početak jednog intervala. Ako neka od komponenti u ulozi pasivne strane ne može da završi svoj deo transakcije u zadanom broju intervala, tada ona koristi dodatne intervale. Radi ovakvih komponenti, sinhrone sabirnice sadrže posebnu upravljačku liniju, po kojoj pasivna strana u završnom intervalu šalje signal kraja transakcije. Transakcija nije završena dok se taj signal ne pojavi. Dužina intervala je zavisna, ne samo od karaktera komponenti, nego i od

dužine sabirnice, da bi u toku jednog intervala signali mogli da stignu s kraja na kraj sabirnice.

Upravljanje sinhronim sabirnicama je jednostavno, pa se na njega ne troši mnogo vremena. Zato, uz dovoljan broj adresnih i linija podataka, uz kratke intervale, kao i uz obavljanje svih transakcija u minimalnom broju intervala, sinhrone sabirnice imaju veoma visoku propusnost. One se uglavnom koriste za povezivanje procesora i radnih memorija. Tada njihova dužina ne prelazi pola metra.

Za raznovrsne organizacione komponente, koje se nalaze na većim međusobnim udaljenostima (od više metara), praktično je teško ostvariti sinhronizovan rad. Zato se saradnja ovakvih komponenti zasniva na dogovaranju (*handshaking*). Ono se ostvaruje razmenom signala, za šta su neophodne posebne upravljačke linije. Po ovakvim linijama aktivna strana najavljuje pasivnoj početak transakcije, a pasivna strana obaveštava aktivnu stranu o završetku transakcije. Sabirnice, koje zahtevaju dogovaranje aktivne i pasivne strane, se nazivaju **asinhrone sabirnice** (*asynchronous bus*). Poštovanje protokola razmene signala, čije trajanje mora da uzme u obzir najgore pretpostavke, uzrokuje da asinhrone sabirnice imaju manju propusnost od sinhronih sabirnica (ovakva poređenja imaju smisla samo ako su sabirnice izvedene u istoj tehnologiji).

Za uspešan prenos podataka neophodno je sprečiti da više aktivnih strana istovremeno pokrenu svoje transakcije. Zato pokretanju transakcije obavezno prethodi zauzimanje sabirnice. U slučaju kada na istu sabirnicu može biti vezano više procesora, zgodno je da u zauzimanju sabirnice posreduje poseban **arbitar sabirnice** (*arbiter*). Da bi zauzela sabirnicu, aktivna strana upućuje **signal zahteva** (*bus request*) po posebnoj upravljačkoj liniji koja vodi ka arbitru sabirnice. Sabirnica je dodeljena aktivnoj strani do koje stigne **signal odobrenja** (*bus grant*) po posebnoj upravljačkoj liniji koja vodi od arbitra sabirnice. Aktivna strana objavljuje da je zauzela sabirnicu upućivanjem **signala zauzeća** (*bus busy*) po posebnoj upravljačkoj liniji koja vodi ka arbitru sabirnice. Taj signal ostaje aktivan dok god aktivna strana koristi sabirnicu. Ako je sabirnica zauzeta u trenutku pojave signala zahteva, arbitar sabirnice čeka oslobađanje sabirnice, pa tek onda odobrava njeno ponovno zauzimanje. Pri tome, nema smetnje da više aktivnih strana po istoj upravljačkoj liniji upućuje signal zahteva ka arbitru sabirnice, pod uslovom da upravljačka linija, namenjena za signal odobrenja, serijski (*daisy chaining*) povezuje aktivne strane (Slika 10.2.2).



Slika 10.2.2 Serijsko vezivanje aktivnih strana linijom odobrenja

U slučaju serijskog vezivanja aktivnih strana linijom odobrenja, sabirnicu zauzima prva aktivna strana do koje stigne signal odobrenja. Ona taj signal ne propušta dalje. Ovakav način zauzimanja sabirnice je jednostavan, ali ima manu da statički dodeljuje viši prioritet aktivnoj strani koja je fizički bliže arbitru sabirnice. Ovaj problem se izbegava, ako između arbitra sabirnice i svake od aktivnih strana postoje posebne linije za signale zahteva i odobrenja (*independent requesting*). Moguće je i kombinovanje prethodna dva pristupa, tako da svaka od više upravljačkih linija, namenjenih za signal odobrenja, serijski povezuje više aktivnih strana.

Na upravljanje sabirnicom se troši manje vremena, ako se, nakon pripreme prenosa, ne prenosi samo jedan, nego više podataka u nizu. Tada se vreme, potrošeno za pripremu prenosa, raspoređuje na svaki od prenosa u nizu. Zato sabirnice omogućuju, ne samo jedinični (*single word*), nego i grupni (*multiple word*) prenos.

Veličina podatka, koji se u jednom navratu prenosi sabirnicom, zavisi od broja linija podataka. Veći broj linija podataka obezbeđuje veću propusnost sabirnice. Radi snižavanja cene, nekada se linije podataka koriste i za prenos adrese. Ovakvo **vremensko multipleksiranje** linija sabirnice smanjuje njenu propusnost.

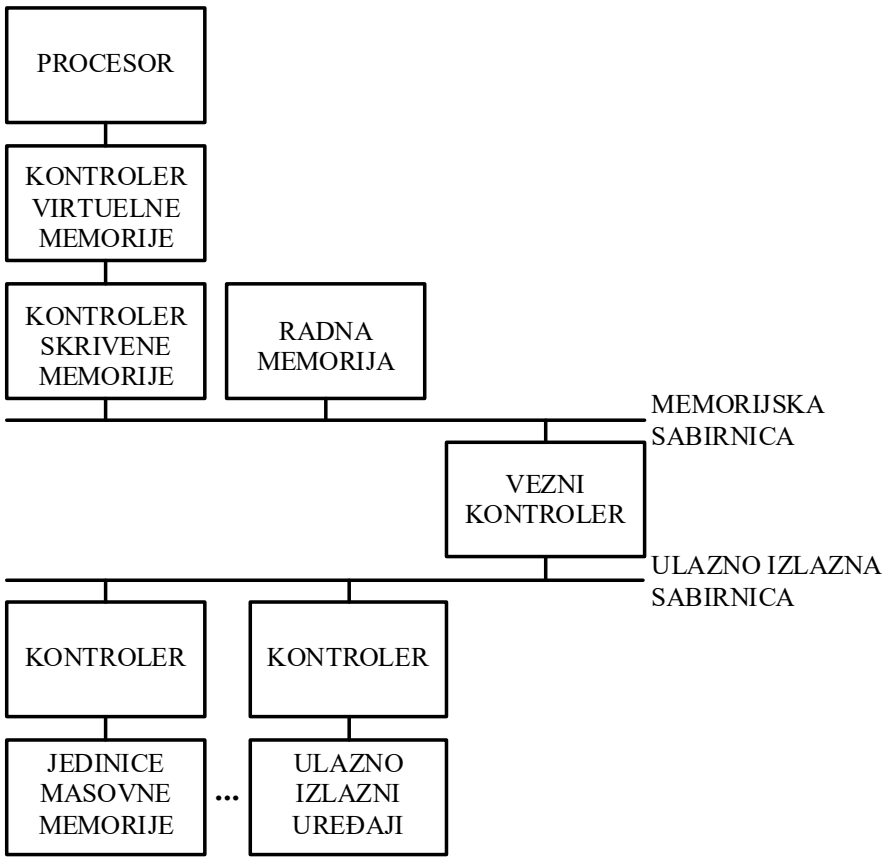
Broj linija, kao i propusnost sabirnice, obrnuto su proporcionalni njenoj dužini. Zato najduže sabirnice, dugačke više metara, sadrže samo od 8 do 16 linija podataka i koriste posebne protokole, radi pravilne interpretacije sadržaja linija podataka.

Namena sabirnice u potpunosti oblikuje njene osobine. Tako se razlikuju:

1. memorijske sabirnice (posvećene povezivanju procesora i radnih memorija),
2. ulazno-izlazne sabirnice (posvećene povezivanju kontrolera masovnih memorija i kontrolera ulaznih i izlaznih uređaja) i
3. sistemske sabirnice (posvećene povezivanju procesora, radnih memorija i kontrolera).

Memorijske sabirnice imaju najveću propusnost i najmanju dužinu, a ulazno-izlazne sabirnice imaju najmanju propusnost i najveću dužinu. Osobine sistemskih sabirnica su između osobina memorijskih i ulazno-izlaznih sabirnica.

Sistemska sabirnica se koristi kao jedina sabirnica kod mini-računara. Memorijska i ulazno-izlazna sabirnica se sreću kod velikih računara, kod kojih je važno da aktivnosti radi ulaza i izlaza podataka što manje ometaju rad procesora. Slika 10.2.3 sadrži prikaz organizacije računara sa memorijskom i ulazno-izlaznom sabirnicom.



Slika 10.2.3 Organizacija računara sa memorijskom i ulazno-izlaznom sabirnicom

Sabirnice su najčešće standardizovana organizaciona komponenta računara.

ORGANIZACIJA ASOCIJATIVNE MEMORIJE

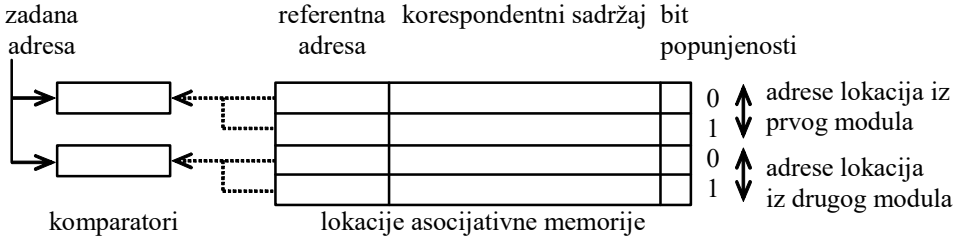
Zadatak asocijativne memorije je da brzo pronađe sadržaj koji korespondira zadanoj adresi. Kada asocijativna memorija opslužuje kontroler virtuelne memorije, ovaj sadržaj predstavlja traženu fizičku adresu, a kada asocijativna memorija opslužuje kontroler skrivene memorije, ovaj sadržaj predstavlja kopiju tražene linije. Između zadane adrese i traženog sadržaja se uspostavi korespondencija, kada

se oni, kao uređeni par, smeste u neku lokaciju asocijativne memorije. Kada zadana adresa postane deo ovakvog uređenog para, ona se naziva **referentna adresa** (*tag*). Pored bita, predviđenih za smeštanje uređenog para, sastavljenog od referentne adrese i njoj korespondentnog sadržaja, svaka lokacija asocijativne memorije sadrži i poseban **bit popunjenosti**, namenjen za podatak o popunjenosti dotične lokacije. Brzo pronalaženje traženog sadržaja omogućuju komparatori, koji zadanu adresu porede sa prisutnom referentnom adresom iz popunjene lokacije asocijativne memorije. Kada je broj komparatora jednak broju lokacija asocijativne memorije (Slika 10.2.4), tada ona ima **punu asocijativnost** (*full associativity*).

Slika 10.2.4 Asocijativna memorija (od 4 lokacije) sa punom asocijativnošću

Kod asocijativne memorije sa punom asocijativnošću, zadana adresa se istovremeno poredi sa svim prisutnim referentnim adresama. Poređenje je uspešno, odnosno ostvaren je pogodak, ako se zadana adresa podudari sa nekom od prisutnih referentnih adresa. Tada sadržaj, uparen sa dotičnom referentnom adresom, predstavlja traženi sadržaj. Ako se zadana adresa ne podudara sa prisutnim referentnim adresama, poređenje je neuspešno, odnosno ostvaren je promašaj. Tada sleduje punjenje neke od lokacija asocijativne memorije novim uređenim parom, kada izvan asocijativne memorije bude pronađen sadržaj koji korespondira zadanoj adresi. Za punjenje se odabira bilo koja od nepopunjenih lokacija asocijativne memorije. Ako su sve lokacije asocijativne memorije popunjene, za punjenje se oslobađa lokacija sa najstarijim uspešnim poređenjem. Ovakav kriterijum izbora lokacije za punjenje je u skladu sa principom lokalnosti po kome lokacija sa uspešnim poređenjem u neposrednoj prošlosti ima veću verovatnoću za uspešno poređenje u neposrednoj budućnosti, od lokacija sa neuspešnim poređenjima u neposrednoj prošlosti. Za brzo pronalaženje lokacije asocijativne memorije sa najstarijim uspešnim poređenjem, zgodno je lokacije uvezati u listu. Na njen početak uvek dolazi lokacija sa najnovijim uspešnim poređenjem, tako da na njen kraj prirodno dospeva lokacija sa najstarijim uspešnim poređenjem. Ovakav pristup predstavlja realizaciju algoritma, označenog skraćenicom *LRU* (*least recently used*). On je primenljiv samo ako je izvezivanje lokacije iz liste, radi njenog uvezivanja na početak liste moguće izvesti bez usporavanja rada asocijativne memorije.

Ekonomski razlozi sugerišu smanjenje broja komparatora, kada asocijativna memorija ima mnogo lokacija. To se može postići modularnom organizacijom asocijativne memorije, u kojoj na svaki modul dolazi po jedan komparator. U odnosu na asocijativnu memoriju sa punom asocijativnošću, broj komparatora se može prepoloviti, ako se lokacije asocijativne memorije organizuju u dva modula sa po dve lokacije (Slika 10.2.5).

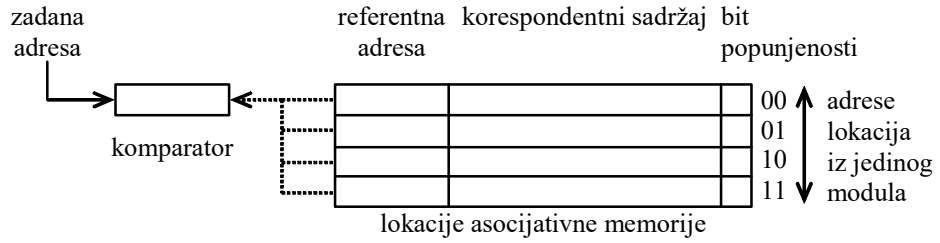


Slika 10.2.5 Asocijativna memorija (od 4 lokacije) sa dvostrukom asocijativnošću

Slika 10.2.5 prikazuje asocijativnu memoriju sa **dvostrukom asocijativnošću** (*2-way set-associativity*). Kada je broj komparatora manji od broja lokacija asocijativne memorije, kao kod asocijativne memorije sa dvostrukom asocijativnošću, tada svaki komparator poredi zadanu adresu sa prisutnom referentnom adresom iz samo jedne lokacije svog modula. Za izbor lokacije koristi se deo bita zadane adrese. Ovi biti tada ne učestvuju u poređenju, jer se podrazumevaju, pa se ne uključuju u referentnu adresu. Referentna adresa može dospeti samo u lokacije koje adresiraju biti isključeni iz nje, jer se ovi biti podrazumevaju samo za dotične lokacije. U primeru koga sadrži Slika 10.2.5 za adresiranje lokacija asocijativne memorije dovoljan je jedan, na primer najmanje značajni bit zadane adrese. Za slučaj da je zadana adresa 11101110, njeni značajniji biti 1110111 se istovremeno porede sa referentnim adresama iz nultih lokacija oba modula.

Lokacije sa istim adresama iz raznih modula asocijativne memorije obrazuju skupove. Lokacije iz istog skupa se uvezuju u listu radi pronalaženja lokacije sa najstarijim uspešnim poređenjem (*LRU* algoritam).

Broj komparatora asocijativne memorije se može svesti na 1, ako sve lokacije asocijativne memorije obrazuju jedan modul (Slika 10.2.6).



Slika 10.2.6 Asocijativna memorija (od 4 lokacije) sa jednostrukom asocijativnošću

Ovakva asocijativna memorija ima **jednostruku asocijativnost** (1-*way set-associativity*, *direct mapped*). U primeru koga sadrži Slika 10.2.6 jedini komparator poredi zadanu adresu sa prisutnom referentnom adresom iz lokacije koju adresiraju dva, na primer, najmanje značajna bita zadane adrese. Pošto se oni podrazumevaju, oni ne učestvuju u poređenju, pa njih ne sadrži referentna adresa. Za slučaj da je zadana adresa 11111100, njeni značajniji biti 111111 se istovremeno porede sa referentnom adresom iz nulte lokacije.

Iskustvo pokazuje da asocijativna memorija sa dvostrukom asocijativnošću nije lošija od dvostruko veće asocijativne memorije sa jednostrukom asocijativnošću. To se objašnjava činjenicom da je svim zadanim adresama, sa istim najmanje značajnim bitima, na raspolaganju samo jedna lokacija asocijativne memorije sa jednostrukom asocijativnošću, a dve lokacije asocijativne memorije sa dvostrukom asocijativnošću.

Na poboljšanje rada asocijativne memorije utiču povećanje broja njenih lokacija, kao i povećanje njenog stepena asocijativnosti. Ipak, značaj povećanja stepena asocijativnosti opada iznad četvorostruke asocijativnosti, između ostalog i zbog povećanja složenosti asocijativne memorije.

Asocijativne memorije se oslanjaju na najbrže (najskuplje) tehnologije, da bi što brže obavljale svoj zadatak, pa to ograničava i broj njihovih lokacija i broj komparatora.

SKRIVENA MEMORIJA

Kontroler skrivene memorije podržava operacije čitanja i pisanja. U toku obavljanja ovih operacija, on proverava da li se u nekoj od lokacija njegove asocijativne memorije nalazi kopija linije sa sadržajem lokacije koju adresiraju njene operacije. Za uspešan rad skrivene memorije je važno da prethodna provera što češće ima pozitivan ishod, na primer da verovatnoća pogotka (*hit*) bude, veća od 0.8, jer mnogo promašaja (*miss*) značajno produžava srednje vreme pristupa. Tako, za asocijativnu memoriju sa vremenom pristupa od 2 procesorska ciklusa, za radnu memoriju sa vremenom pristupa od 10 procesorskih ciklusa i za verovatnoću pogotka 0.9, srednje vreme pristupa je 2.8 procesorskih ciklusa, a za verovatnoću pogotka 0.7, srednje vreme pristupa je 4.4 procesorskih ciklusa. Zapravo, vreme

pristupa u slučaju promašaja je višestruko duže od vremena pristupa u slučaju pogotka, jer uključuje pristup sadržaju lokacije iz radne memorije, kao i prenos kopija sadržaja svih lokacija linije između radne i asocijativne memorije.

Upravljanje skrivenom memorijom se razlikuje za operacije čitanja i pisanja, kao i za slučajeve pogotka i promašaja u okviru neke od ovih operacija. Tako, pogodak kod čitanja sadržaja ne zahteva pristup radnoj memoriji, a promašaj dovodi do pristupa radnoj memoriji, radi prebacivanja kopije sadržaja linije iz radne memorije u asocijativnu memoriju.

Operacija pisanja sadržaja adresirane lokacije je dugotrajnija od operacije čitanja kopije sadržaja adresirane lokacije, jer podrazumeva pristupanje radnoj memoriji. Pri tome, u slučaju pogotka, odmah se menja kopija adresiranog sadržaja u lokaciji skrivene memorije, a pristupanje radnoj memoriji se može odložiti do trenutka kada se pomenuta lokacija asocijativne memorije oslobađa. Tek tada se izbacivana kopija linije upisuje u radnu memoriju (*write back*), i to samo ako je izmenjena. Radi toga, svakoj lokaciji asocijativne memorije se dodaje **bit izmenjenosti** (*modify bit*), koji se pri svakom pisanju postavlja na vrednost 1.

Odlaganje pisanja u radnu memoriju smanjuje broj pristupanja radnoj memoriji, ali po cenu da su sadržaji pojedinih lokacija radne memorije neažurni. Ako je to neprihvatljivo, tada svako pisanje kopije adresiranog sadržaja u lokaciji skrivene memorije prati i pristupanje radnoj memoriji (*write through*).

U slučaju promašaja pri pisanju sadržaja adresirane lokacije, moguće je menjati samo sadržaj adresirane lokacije u radnoj memoriji (*write around*). To je opravdano samo ako nema naknadnih izmena sadržaja iste lokacije. U suprotnom, bolje je da se kopija sadržaja adresirane lokacije radne memorije, nakon izmene, prebaci u skrivenu memoriju (*write allocate*). Pri tome se uvek prebacuje sadržaj cele linije. Slika 10.2.7 sadrži pregled prethodno opisanih slučajeva.

<table>
<tr><td rowspan="2">čitanje</td><td>za pogodak</td><td colspan="2">iz asocijativne memorije</td></tr>
<tr><td>za promašaj</td><td colspan="2">iz radne memorije, uz ažuriranje asocijativne memorije</td></tr>
<tr><td rowspan="4">pisanje</td><td rowspan="2">za pogodak</td><td rowspan="2">u asocijativnu memoriju</td><td>uz odlaganje pisanja u radnu memoriju (write back)</td></tr>
<tr><td>uz pisanje u radnu memoriju (write through)</td></tr>
<tr><td rowspan="2">za promašaj</td><td rowspan="2">u radnu memoriju</td><td>uz ažuriranje asocijativne memorije (write allocate)</td></tr>
<tr><td>bez ažuriranja asocijativne memorije (write around)</td></tr>
</table>

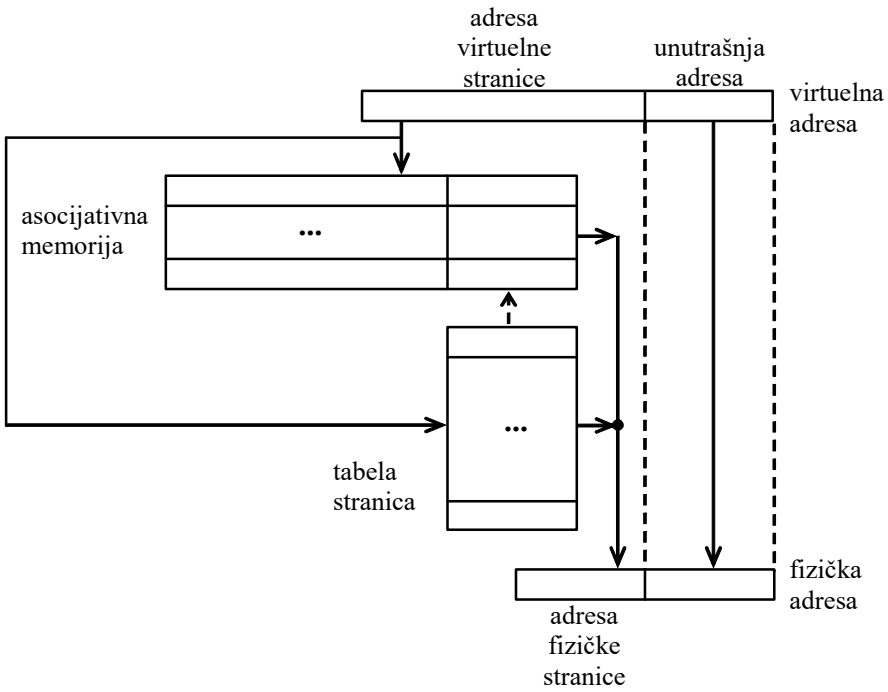
Slika 10.2.7 Pregled slučajeva čitanja i pisanja skrivene memorije

Koncepti skrivene memorije i memorijski preslikanog ulaza izlaza nisu u saglasnosti, jer ulazno-izlazni uređaji mogu da izmene sadržaj lokacija iz radne

memorije, i da tako učine neažurnim njihove kopije u skrivenoj memoriji. Rešenje ovoga problema je da skrivena memorija ne sadrži kopije sadržaja lokacija kojima pristupaju ulazno-izlazni uređaji. Postoji i alternativno rešenje, da kontroleri ulazno-izlaznih uređaja pristupaju radnoj memoriji posredstvom skrivene memorije. Njegova slabost je da tada ovi kontroleri ometaju rad procesora.

VIRTUELNA MEMORIJA

Kontroler virtuelne memorije podržava operacije čitanja i pisanja. U toku obavljanja ovih operacija, on proverava da li se u nekoj od lokacija njegove asocijativne memorije nalazi adresa fizičke stranice koja korespondira zadanoj adresi virtuelne stranice (Slika 10.2.8).



Slika 10.2.8 Pretvaranje virtuelne adrese u fizičku

Pretvaranje virtuelne adrese u fizičku je brže, ako asocijativna memorija sadrži zadanu adresu virtuelne stranice i njoj korespondentnu adresu fizičke stranice. U suprotnom slučaju, proverava se u tabeli stranica da li adresiranoj virtuelnoj stranici odgovara fizička stranica. Ako odgovara, obavi se pretvaranje virtuelne adrese u fizičku, zamenom adrese virtuelne stranice korespondentnom adresom fizičke stranice. Uz to se ažurira asocijativna memorija, tako što se u neku od njenih lokacija (odabranih iz skupa mogućih lokacija po *LRU* algoritmu) smeste

pomenuta adresa virtuelne stranice i njoj korespondentna adresa fizičke stranice. Međutim, ako adresiranoj virtuelnoj stranici ne odgovara fizička stranica, odnosno ako je odgovarajući element tabele stranica nepopunjen, generiše se stranični prekid. Zadatak obrađivača ovog prekida je: (1) da obezbedi slobodnu fizičku stranicu, (2) da obezbedi da se u nju smesti kopija adresirane virtuelne stranice i (3) da ažurira odgovarajuće elemente tabele stranica (radi uparivanja adrese virtuelne stranice i adrese odgovarajuće fizičke stranice), što za sobom povlači i ažuriranje asocijativne memorije. Po završetku obrade straničnog prekida moguće je pretvaranje zadane virtuelne adrese u fizičku adresu.

Na popunjenost elementa tabele stranica ukazuje njegov **bit popunjenosti**.

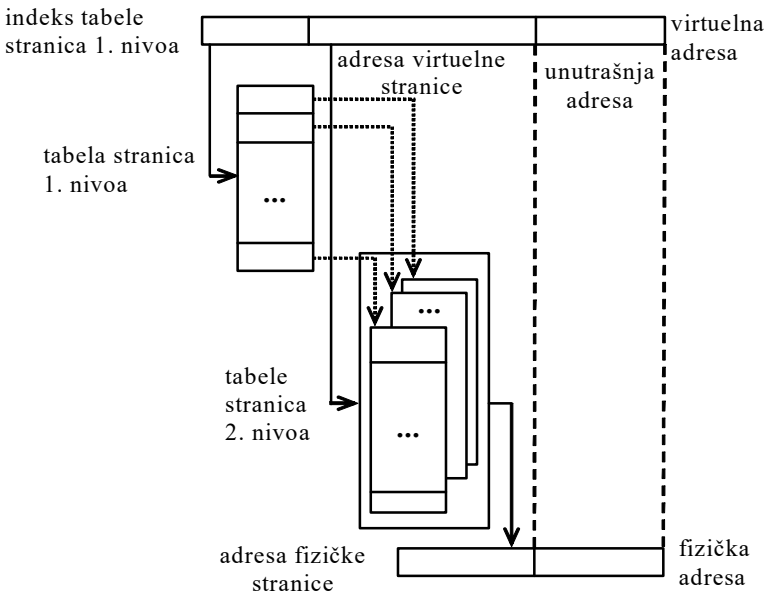
Obezbeđivanje slobodne fizičke stranice podrazumeva oslobađanje jedne od zauzetih fizičkih stranica, ako su sve fizičke stranice zauzete. To obuhvata ažuriranje tabele stranica, radi raskidanja postojeće veze između adrese neke virtuelne stranice i adrese oslobađane fizičke stranice, kao i prebacivanje sadržaja oslobađane fizičke stranice na disk, ako je njen sadržaj izmenjen. Zato svaka izmena sadržaja fizičke stranice mora da ostane zabeležena. Radi toga se za svaku fizičku stranicu veže **bit izmenjenosti** (*modify bit*, *dirty bit*). On se automatski postavlja pri svakoj izmeni sadržaja fizičke stranice. Prilikom izbora fizičke stranice za oslobađanje, dobro je primenjivati *LRU* algoritam. Radi toga je potrebno beležiti svako pristupanje sadržaju, odnosno referenciranje sadržaja fizičke stranice. Zato se za svaku stranicu veže i **bit referenciranja** (*reference bit*). On se automatski postavlja pri svakom pristupanju sadržaju fizičke stranice. Podrazumeva se da se bit referenciranja poništava u pravilnim vremenskim intervalima, da podatak o referenciranju ne bi zastario.

Bit izmenjenosti i bit referenciranja su vezani za fizičku stranicu, pa i za element tabele stranica u kome je njena adresa. To znači da se kopije ovih bita nalaze i u lokaciji asocijativne memorije, kada ona sadrži adresu njihove fizičke stranice. Ovi biti se poništavaju, kada se njihova fizička stranica oslobađa.

Za elemente tabele stranica se, pored bita izmenjenosti i bita referenciranja, vezuje i **bit zabrane prebacivanja** linija sadržaja fizičke stranice u skrivenu memoriju. To je važno za stranice kojima pristupaju kontroleri ulazno-izlaznih uređaja.

Za veliki adresni prostor i malu stranicu, tabela stranica postaje predugačka. Na primer, za adresni prostor od 32 bita i stranicu od 2^9 (512) bajta, tabela stranica ima 2^{23} elemenata i minimalno zauzima 2^{25} bajta (32 megabajta), ako se za svaki element predvide po 4 bajta. Za ovako veliku tabelu nema dovoljno prostora u radnoj memoriji, pogotovo u uslovima multiprogramiranja, kada svakom izvršavanom programu odgovara posebna tabela stranica. Zbog toga se trenutno nekorišćeni delovi tabele stranica smeštaju u masovnu memoriju, što produžava srednje vreme pristupa virtuelnoj memoriji, jer zahteva trošenje vremena na prebacivanje delova tabele stranica između radne i masovne memorije. Zato je veoma važno smanjiti veličinu tabele stranica. To se može postići povećanjem

veličine stranica, ali, za kraće slike procesa koje ne zauzimaju sve stranice virtuelnog adresnog prostora, efikasnije rešenje je podela tabele stranica na više odsečaka iste veličine i korišćenje samo odsečaka čije elemente indeksiraju adrese zauzetih virtuelnih stranica. O svakom odsečku tabele stranica se mora znati da li se koristi i gde se nalazi. Podatke o odsečcima sadrže elementi posebne tabele evidencije posredstvom koje se pristupa korišćenim odsečcima. Za ovakvo rešenje se kaže da uvodi **organizaciju tabele stranica u dva nivoa**, jer se na prvom nivou nalazi tabela evidencije, čiji elementi sadrže adrese odsečaka tabele stranica na drugom nivou. Elementi ovih odsečaka sadrže adrese fizičkih stranica. Tabela evidencije se naziva **tabela stranice sa prvog nivoa**, a odsečci tabele stranice se nazivaju **tabele stranica sa drugog nivoa**. Kada tabela stranica ima dva nivoa, biti virtuelne adrese se razvrstavaju u tri grupe. Najznačajniji biti virtuelne adrese indeksiraju elemente tabele stranica sa prvog nivoa. Preostali biti sadrže adresu virtuelne stranice i unutrašnju adresu. Adresa virtuelne stranice indeksira element tabele sa drugog nivoa. Adresu ove tabele sadrži indeksirani element tabele sa prvog nivoa (Slika 10.2.9).



Slika 10.2.9 Tabela stranica iz dva nivoa

Prednost organizacije tabele stranica u dva nivoa je da ona zahteva postojanje samo tabele sa prvog nivoa i samo korišćenih tabela sa drugog nivoa. A njih ima malo za kratke slike procesa, jer one zauzimaju malo stranica.

Virtuelne adrese koristi procesor, ali je zgodno da ih koriste i kontroleri. Da bi to bilo moguće, neophodno je osposobiti kontrolere za pretvaranje virtuelnih

adresa u fizičke, ali i osposobiti operativni sistem, tako da podržava i ovakvo pretvaranje adresa.

MEMORIJSKA HIJERARHIJA

Ideja memorijske hijerarhije je zasnovana na osobini lokalnosti izvršavanja programa, a njen cilj je da kombinovanjem memorija raznih cena, kapaciteta i vremena pristupa ostvari što veći kapacitet ukupne memorije, po što nižoj ceni i uz što kraće srednje vreme pristupa. Pri tome, na višim nivoima memorijske hijerarhije se nalaze memorije manjeg kapaciteta, više cene i kraćeg vremena pristupa. Ovo vreme je kraće delom zbog skupocenije tehnologije, a delom zbog toga što manja memorija uvodi manje kašnjenja signala.

Memorijska hijerarhija ima četiri osnovna nivoa. Na prvom (najvišem) nivou, nalaze se procesorski registri opšte namene. Zato se kapacitet ovoga nivoa izražava u bajtima, a vreme pristupa odgovara procesorskom ciklusu. Na drugom nivou nalazi se skrivena memorija. Njen kapacitet se izražava u kilobajtima, a vreme pristupa je oko 2 procesorska ciklusa. Na trećem nivou nalazi se radna memorija. Njen kapacitet se izražava u megabajtima, a vreme pristupa je približno 10 procesorskih ciklusa. Na četvrtom (najnižem) nivou nalazi se masovna memorija. Njen kapacitet se izražava u gigabajtima, a vreme pristupa je približno 10^6 procesorskih ciklusa.

Do prebacivanja kopija sadržaja sa nižih na više nivoe dolazi kada se otkrije da na najvišem nivou nema kopije potrebnog sadržaja. Ovo se naziva promašaj, za razliku od pogotka, kada se na najvišem nivou pronađe kopija potrebnog sadržaja. O prebacivanju kopija sadržaja lokacija sa drugog na prvi nivo brine kompajler, odnosno, mašinske naredbe koje on generiše. O prebacivanju kopija sadržaja linija sa trećeg na drugi nivo brine procesor, odnosno njegov kontroler skrivene memorije. O prebacivanju kopija sadržaja stranica sa četvrtog na treći nivo brinu procesor, odnosno njegov kontroler virtuelne memorije i operativni sistem. Prebacivanje kopija sadržaja stranica sa četvrtog nivoa na treći nivo pokreće procesor, ali, pošto je to prebacivanje znatno dugotrajnije od preključivanja, procesor ne čeka na kraj prebacivanja, nego se preključuje na izvršavanje drugog programa. Zato u ovom prebacivanju učestvuje i operativni sistem. Prebacivanje kopije sadržaja linije sa trećeg na drugi nivo je znatno kraće od preključivanja, pa zato procesor pokreće ovakvo prebacivanje i čeka njegov kraj. U svakom slučaju, procesor podržava memorijsku hijerarhiju, (1) ako brzo proverava da li postoji kopija traženog sadržaja u višim nivoima memorijske hijerarhije, i (2) ako izaziva odgovarajući prekid, radi aktiviranja operativnog sistema, kada se na trećem nivou memorijske hijerarhije ne pronađe kopija sadržaja tražene stranice.

Verovatnoća promašaja zavisi od veličine linije/stranice. Ako su linije/stranice suviše male, tada se u njima ne nalaze kopije sadržaja svih lokacija koje su obuhvaćene prostornom lokalnošću izvršavanja programa, pa se tada često javljaju promašaji. S druge strane, ako su linije/stranice prevelike, tada ih može

manje stati u drugi/treći nivo memorijske hijerarhije, pa to uzrokuje česte promašaje. Zato se veličina linija kreće od 4 do 128 bajta, a veličina stranica od 512 do 8192 bajta.

PROBLEM SINHRONIZACIJE

Potreba za bržim reagovanjem računara na vanjske događaje zahteva da se vreme obrade prekida skrati. Zato obrađivač prekida obavlja samo neodložan deo posla, a ostalo prepušta pozadinskom procesu. Pozadinski proces preuzima od obrađivača prekida podatke posredstvom deljenog bafera. Ako obrada prekida prekine pozadinski proces u preuzimanju podataka iz deljenog bafera i izmeni sadržaj ovog bafera, pozadinski proces će preuzeti deo starog i deo novog sadržaja ovog bafera. Na primer, to može da se desi, ako obrađivač prekida smešta u deljeni bafer koordinate pozicije kursora na ekranu, a pozadinski proces preuzima ove koordinate da bi prikazao kursor u odgovarajućoj poziciji na ekranu. Ako je deljeni bafer do prekida sadržao koordinate (0,0) i ako je obrada prekida izmenila ove koordinate na (1,1), tada je pozadinski proces mogao pre prekida da preuzme apscisu 0, a posle prekida ordinatu 1. Rezultat ovakvog sleda događaja je prikazivanje kursora na pogrešnoj poziciji (0,1). Opisani problem se naziva **problem sinhronizacije**. On se rešava onemogućavanjem prekida dok pozadinski proces preuzima sadržaj deljenog bafera. Time se sprečava da za vreme preuzimanja sadržaja ovog bafera taj sadržaj bude izmenjen.

Problem sinhronizacije se javlja i između procesa koji dele neki resurs računara, a predstavlja direktnu posledicu nepredvidivosti preključivanja procesora sa procesa na proces. Preključivanja su nepredvidiva, jer ih izazivaju obrađivači prekida, čija aktiviranja imaju slučajnu prirodu. Tako, ako dva procesa koriste isti štampač, tada njihov međusobni nesinhronizam može da uzrokuje izmešanost znakova koje su oni uputili na isti štampač. Problem deljenih resursa se rešava zauzimanjem deljenog resursa pre njegovog korišćenja samo za jedan proces i oslobađanjem tog resursa nakon njegovog korišćenja. To podrazumeva i privremeno zaustavljanje aktivnosti procesa, kada oni pokušaju da zauzmu prethodno zauzeti deljeni resurs. Zaustavljeni procesi nastavljaju svoju aktivnost jedan po jedan tek kada deljeni resurs bude moguće zauzeti za nekog od njih, što se desi nakon oslobađanja dotičnog resursa. Zauzimanje i oslobađanje resursa podrazumeva izmenu stanja resursa. Stanja resursa su slobodan i zauzet. Ona se mogu reprezentovati vrednostima 1 i 0 (respektivno). Za čuvanje stanja resursa potrebna je posebna deljena memorijska lokacija. Ova lokacija stanja resursa je deljena, jer joj pristupaju razni procesi prilikom zauzimanja i oslobađanja resursa. Zauzimanje resursa podrazumeva izmenu njegovog stanja. Ova izmena se ostvaruje pisanjem vrednosti 0 u lokaciju stanja resursa. I oslobađanje resursa podrazumeva izmenu njegovog stanja. Ova izmena se ostvaruje pisanjem vrednosti 1 u lokaciju stanja resursa. Međutim, prilikom zauzimanja resursa, pre izmene njegovog stanja, mora se proveriti da li je resurs već zauzet. Prema tome, pisanju vrednosti 0 u

lokaciju stanja resursa, mora da prethodi čitanje te lokacije. Pošto su čitanje i pisanje memorijskih lokacija nezavisne operacije, između njih može doći do obrade prekida. To znači da, nakon što jedan proces u okviru zauzimanja resursa pročita lokaciju stanja resursa, obrada prekida može izazvati preključivanje procesora na drugi proces. Ako taj drugi proces pokuša da zauzme isti resurs, i on će pročitati lokaciju stanja istog resursa. Na ovaj način će oba procesa će preuzeti istu vrednost iz lokacije stanja resursa. Ako je preuzeta vrednost 1, oba procesa će zaključiti da je resurs slobodan i zauzeće ga. Nakon toga je moguć međusobni nesinhronizam ovih procesa. Problem se ne bi javio, ako bi u toku zauzimanja resursa čitanje i pisanje lokacije stanja ovog resursa bili nedeljivi. Njihova nedeljivost se može ostvariti eksplicitnim ili implicitnim onemogućenjem prekida. Eksplicitno onemogućenje prekida podrazumeva izmenu bita prekida u registru stanja. Implicitno onemogućenje prekida podrazumeva postojanje mašinske naredbe, čije izvršavanje obuhvata i čitanje i pisanje iste memorijske lokacije. Ako se obezbedi nedeljivost čitanja i pisanja lokacije stanja resursa, tada najviše jedan proces može da zatekne resurs u stanju slobodan, jer se u lokaciju stanja resursa upisuje vrednost 0 odmah po čitanju vrednosti 1. Prema tome, do oslobađanja resursa svi procesi mogu preuzeti samo vrednost 0 iz lokacije stanja resursa.

10.3. PITANJA

1. Šta je činilo tehnološku osnovu računara treće generacije?
2. Zašto je tehnologija integrisanih kola istisnula tehnologiju diskretnih poluprovodnika?
3. Šta je karakterisalo treću generaciju računara?
4. Šta je karakterisalo mini-računare?
5. Šta je karakterisalo arhitekturu naredbi treće generacije računara?
6. Šta karakteriše *CISC* računare?
7. Šta omogućuje mikro-programiranje?
8. Šta je karakterisalo procesore računara treće generacije?
9. Šta je karakterisalo organizaciju radne memorije računara treće generacije?
10. Kako se obavlja prenos podataka između radne i masovne memorije kod računara treće generacije?
11. Šta karakteriše multiprogramiranje?
12. Šta je stepen multiprogramiranja?
13. Šta karakteriše memorijski preslikani ulaz-izlaz?
14. Šta je važno za implementaciju virtuelne memorije?
15. Koji pojmovi su vezani za virtuelnu memoriju?
16. Koje vrste lokalnosti izvršavanja programa postoje?
17. Koje stranice obrazuju radni skup?
18. Šta određuje veličinu adresnog prostora?
19. Ko obavlja pretvaranje virtuelne adrese u fizičku?

20. Šta važi za virtuelne i fizičke strancie?
21. Od čega se sastoji pretvaranje virtuelne adrese u fizičku?
22. Čemu je proporcionalna veličina tabele stranica?
23. Šta sadrže elementi tabele stranica?
24. Da li je moguće translirati virtuelnu adresu 0100 u fizičku adresu 000, ako unutrašnja adresa stranice ima 2 bita, adresa fizičke stranice ima 1 bit, adresa virtuelne stranice ima 2 bita, a tabela stranica sadrži {-,-,1,-} u svojim elementima (krajnje levo je element sa indeksom 0, a krajnje desno je element sa indeksom 3)?
25. Ko izaziva stranični prekid?
26. Kada dolazi do izazivanja straničnog prekida?
27. Zašto je uvedena asocijativna memorija?
28. Šta je karakteristično za asocijativnu memoriju?
29. Šta je potrebno da bi virtuelna memorija omogućila međusobnu zaštitu istovremeno postojećih procesa?
30. Šta karakteriše memorijsku hijerarhiju?
31. Šta karakteriše skriveu memoriju?
32. Koje sličnosti postoje između virtuelne i skrivene memorije?
33. Šta su uveli operativni sistemi računara treće generacije?
34. Šta je zadatak obrađivača straničnog prekida?
35. Koje su mane računara treće generacije?
36. Da li za radnu memoriju sa rečima od 4 bajta, primarna reč može da ima adresu 24?
37. Da li za radnu memoriju sa rečima od 4 bajta, sekundarna reč može da ima adresu 24?
38. Koje su karakteristike formatiranog magnetnog diska?
39. Od čega zavisi sredje vreme pristupa diska?
40. Koji pojmovi su vezani za sabirnicu?
41. Koje osobine ima sabirnica?
42. Kako se dele sabirnice?
43. Koji signali su vezani za arbitar sabirnice?
44. Koje su karakteristike memorijske sabirnice?
45. Koje su karakteristike ulazno-izlazne sabirnice?
46. Kako se dele asocijativne memorije?
47. Koliko komparatora ima asocijativna memorija sa punom asocijativnošću?
48. Šta sadrže lokacije asocijativne memorije?
49. Čemu je proporcionalan kvalitet asocijativne memorije?
50. Kada skrivena memorija ukida pristup radnoj memoriji?
51. Sa kojim konceptom koncept skrivene memorije nije u saglasnosti?
52. Koje bite sadrže lokacije asocijativne memorije iz kontrolera skrivene memorije?

53. Kako se smanjuje veličina tabele stranica?
54. Koje bite poseduju elementi tabele stranica?
55. Koliko delova ima virtuelna adresa kod organizacije tabele stranica u dva nivoa?
56. Koja je prednost organizacije tabele stranica u dva nivoa?
57. Šta čini osnovne nivoe memorijske hijerarhije?
58. Ko brine o prebacivanju kopija sadržaja lokacija sa jednog na drugi nivo memorijske hijerarhije?
59. Kada se javlja problem sinhronizacije?
60. Kako se rešava problem sinhronizacije?

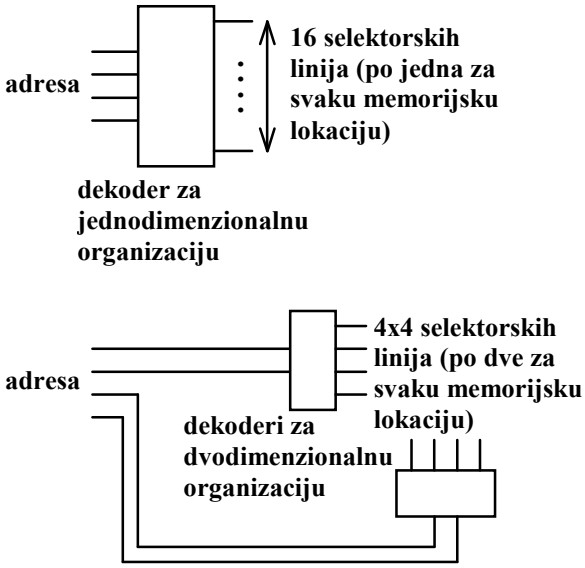
11. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 1980. GODINE

11.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 1980. GODINE

Tehnološku osnovu računara četvrte generacije su činila visoko integrisana kola (*LSI*, *Large Scale Integration*). Tehnologija visoke integracije se oslanjala na gotovo potpuno automatizovano projektovanje i omogućavala je gotovo potpuno automatizovanu masovnu proizvodnju, što je dovelo do velikog snižavanja cene. Pored toga, na ovaj način proizvedeni čipovi (sa stotinama hiljada tranzistora na sebi) su imali prednosti u pogledu pouzdanosti, brzine rada, potrošnje energije i toplotnog zračenja.

POLUPROVODNIČKE MEMORIJE

Zahvaljujući tehnologiji visoke integracije, poluprovodničke memorije (*RAM*, *Random Access Memory*) su istisnule iz upotrebe memorije sa magnetnim jezgricama, jer su omogućile pravljenje većih i pouzdanijih radnih memorija sa kraćim vremenom pristupa i nižom cenom. Po načinu funkcionisanja poluprovodničke memorije su podeljene na **statičke *SRAM*** (*Static RAM*) i **dinamičke *DRAM*** (*Dynamic RAM*). Kod statičkih memorija, za čuvanje binarnih vrednosti su služile ćelije, organizovane kao *flipflop*, koje su sadržale do 6 tranzistora. Kod dinamičkih memorija, za čuvanje binarnih vrednosti su služile ćelije koje su sadržale tranzistor i kondenzator, čiji naboj je predstavljao binarnu vrednost. Ovakve ćelije su zahtevale periodično osvežavanje, da bi se nadoknadilo pražnjenje kondenzatora. Njihovo čitanje je, iz istih razloga, bilo destruktivno. Zato je vreme ciklusa dinamičke memorije bilo duže od vremena njenog pristupa, za razliku od statičke memorije, između čijeg vremena ciklusa i vremena pristupa nije bilo razlike. I statičke i dinamičke memorije su imale dvodimenzionalnu organizaciju, radi obaranja proizvodne cene čipova. Tako su za 2n adresnih linija bila potrebna 2 dekodera adresa. Jedan je dekodirao n adresnih linija reda, a drugi je dekodirao n adresnih linija kolone kvadratne matrice. Oba dekodera su imala n ulaza i 2^n izlaza i bili su jednostavniji, pa i jeftiniji, od dekodera adresa sa 2n ulaza i sa 2^{2n} izlaza, koji je bio neophodan za jednodimenzionalnu organizaciju (Slika 11.1.1).



Slika 11.1.1 Primer jedno i dvodimenzionalne organizacije radne memorije sa 16 lokacija

Za proizvodnju bita *DRAM* čipa bilo je potrebno manje komponenti nego za proizvodnju bita *SRAM* čipa, pa je *DRAM* čip sadržao više bita od *SRAM* čipa. Zbog toga je *DRAM* čip imao veći kapacitet od *SRAM* čipa za istu cenu, odnosno cena jednog bita *DRAM* čipa je bila niža od cene jednog bita *SRAM* čipa. Međutim, vreme pristupa *DRAM* čipa je bilo duže od vremena pristupa *SRAM* čipa. Iz prethodnog je sledilo da su *DRAM* čipovi bili podesni za radnu memoriju, dok su *SRAM* čipovi bili podesni za skrivenu memoriju. Zato je kod *DRAM* čipova u prvom planu bilo povećanje njihovog kapaciteta i snižavanje njihove cene. Tome je doprinosilo i smanjenje broja adresnih linija *DRAM* čipova na polovinu, uz njihovo vremensko multipleksiranje, tako da su adresne linije prvo prenosile adresu reda, a onda adresu kolone. To je još više produžavalo vreme pristupa *DRAM* čipa. Prema tome, za isti način proizvodnje, *DRAM* čip je imao 16 puta veći kapacitet od *SRAM* čipa, od 8 do 16 puta duže vreme pristupa i oko 10 puta nižu cenu bita.

MIKRO-RAČUNARI

Primena tehnologije visoke integracije je stvorila uslove za poboljšanje mogućnosti računara, kao što je, na primer, proširenje adresnog prostora na 32 bita. Takav pristup je doveo do pojave velikih računara četvrte generacije, kao što su računari iz serije *IBM 370-XA* (*eXtended Architecture*), koja je predstavljala poboljšani produžetak *IBM 370* familije računara. Isti pristup je omogućio i pojavu mini-računara četvrte generacije, koji su se, po osobinama kao što je primena koncepata virtuelne i skrivene memorije, ili proširenje adresnog prostora na 32 bita,

približili velikim računarima četvrte generacije. Ipak između velikih i mini-računara je i dalje ostao jaz u pogledu brzine, veličine masovne memorije, broja terminala i, svakako, cene. Grupi mini-računara četvrte generacije su pripadali računari iz serije *DEC VAX* (*Virtual Address eXtension*), nastali kao poboljšani nastavak *PDP11* familije računara. Drugi način primene tehnologije visoke integracije je vodio u smeru snižavanja cene, na račun skromnije funkcionalnosti. Kao rezultat ovakvog pristupa, nastali su **mikro-računari** (*microcomputer*). Pomenuti naziv je bio posledica, ne samo činjenice da su cene i mogućnosti mikro-računara predstavljale samo delić cene i mogućnosti mini i velikih računara četvrte generacije, nego i činjenice da su fizičke dimenzije mikro-računara bile znatno manje od fizičkih dimenzija mini i velikih računara. Tako je celi mikro-računar zauzimao samo jednu štampanu ploču, a njegov mikro-procesor (*microprocessor*) samo jedan čip.

Za proizvođače, koji su iskoristili tehnologiju visoke integracije za snižavanje cena, na račun skromnije funkcionalnosti svojih proizvoda, karakteristične su mikro-procesorske familije čipova. Ovakve familije su nastajale oko pojedinih mikro-procesora i omogućavale su korisnicima da samostalno prave sopstvene mikro-računare. Među ovakvim proizvođačima prednjačio je *Intel*, koji je još 1971. godine izbacio na tržište prvi (četvoro bitni) mikro-procesor *Intel 4004*, a 1972. godine 8 bitni mikro-procesor *Intel 8008*. Njega je pratila *Motorola*, koja je 1974. godine izbacila na tržište svoj 8 bitni mikro-procesor *Motorola 6800*.

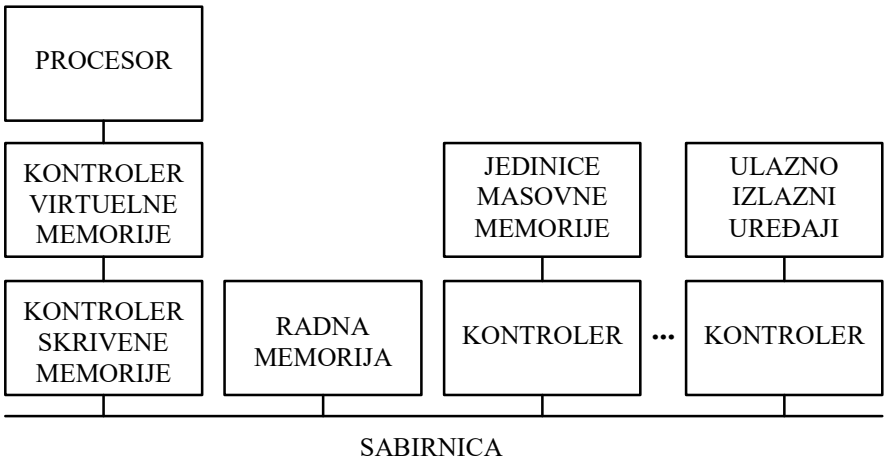
Prvi mikro-procesori su bili veoma skromnih mogućnosti, imali su akumulatorske arhitekture, a njihova pojava je tek nagovestila buran razvoj mikro-procesorskih familija. Do njega je došlo krajem sedamdesetih godina, kada su se pojavili 16 bitni mikro-procesori kao što su *Intel 8086* i *Motorola 68000*. Pojava ova dva mikro-procesora je označila početak razvoja dve familije mikro-procesora (*Intel 80x86* i *Motorola 680x0*), koje su imale ogroman uticaj na tržište. Razvoj mikro-procesora se kretao u smeru poboljšanja njihove funkcionalnosti, čime su se oni približavali mogućnostima mini-računara četvrte generacije. Proširenje mogućnosti mikro-procesora se zasnivalo na korišćenju dodatnih čipova, koji su nazivani periferni procesori (*peripheral processor*), ako nisu bili namenjeni za određeni mikro-procesor, a ako su bili namenjeni za određeni mikro-procesor, tada su nazivani koprocesori (*coprocessor*). Ovakvi čipovi su bili specijalizovani za pojedine poslove, kao što je rukovanje virtuelnom memorijom, ili kao što je aritmetika realnih brojeva (na primer, *Intel 8087 arithmetic coprocessor*). Među čipovima proizvedenim u tehnologiji visoke integracije postojali su i čipovi, namenjeni za sklapanje procesora željenih osobina, kao što je, na primer, bio čip sa segmentnom aritmetičko-logičkom jedinicom (*bit-sliced arithmetic-logic unit*), koji je omogućavao pravljenje aritmetičko-logičke jedinice sa željenom preciznošću.

Zahvaljujući veoma niskoj ceni mikro-računara, prve njihove primene su vezane za razne oblike automatskog upravljanja. Ovakve primene su uticale da proizvođači mikro-računara usvoje pristup otvorene arhitekture. Na primer, *Intel* je takav pristup zasnovao na svojoj *Multibus* sabirnici, a *Motorola* na svojoj *VME*

sabirnici. Zato nije slučajno da se tipična organizacija mikro-računara nije razlikovala od tipične organizacije mini-računara treće generacije. U okviru primena, vezanih za automatsko upravljanje, mikro-računari su bili često potpuno posvećeni veoma jednostavnim poslovima, kao što je bilo, na primer, prikupljanje podataka u režimu **prozivke** (*polling*). U toku prozivke, mikro-procesori su neprekidno proveravali stanja pojedinih izvora podataka, radi otkrivanja prispeća podataka i radi njihovog preuzimanja i čuvanja do trenutka njihovog prosleđivanja. Ovakav način rada je stavljao mikro-procesor u poziciju da posreduje u prenosu svih podataka, pa se tada organizacija mikro-računara nije razlikovala, po načinu rada, od organizacije računara prve generacije.

Primene, vezane za automatsko upravljanje, su dovele do pojave mikro-računara na čipu, nazvanih **mikro-kontroleri** (*microcontroller*), kao što je, na primer, bio mikro-kontroler *Intel 8051*.

Niska cena mikro-računara je omogućila da se oni nađu u ulozi kontrolera, na primer, mini-računara četvrte generacije i da budu potpuno posvećeni poslovima upravljanja uređajima, kao što su ulazni i izlazni uređaji ili jedinice masovne memorije. Tako, u slučaju tipične organizacije mini-računara četvrte generacije, uloga kontrolera ulaznih i izlaznih uređaja, kao i kontrolera jedinica masovne memorije dodeljena je mikro-računarima (Slika 11.1.2).



Slika 11.1.2 Tipična organizacija mini-računara četvrte generacije

Skrivena memorija se zasnivala na *SRAM* čipovima, a radna memorija se zasnivala na *DRAM* čipovima. Ulazno-izlazni uređaji su obuhvatali video terminale i štampače.

PERSONALNI RAČUNARI

Mikro-računari su omogućili interaktivni rad predvidivog odziva, jer su, zahvaljujući niskoj ceni, mogli u potpunosti biti posvećeni jednom korisniku. Za ovakve, **personalne računare** vezana je masovna upotreba **grafičkih terminala**, čiji ekran je sadržao matricu tačaka ili **piksela** (*PIXEL*, *PIcture matriX ELement*), od kojih je svaka bila zasebno pristupačna. Grafički terminali su olakšali interaktivni rad, jer su komunikaciju sa računarom sveli na izbor mogućnosti, prikazanih u obliku pregledno raspoređenih tekstualnih objašnjenja (*menu*) ili crteža (*icon*). Uz prisustvo odgovarajućeg pokazivačkog uređaja (*mouse*), smanjena je upotreba tastature i pojednostavljen je i ubrzan rad korisnika. Za personalne računare je vezana pojava magnetnih disketa (*floppy diskette*), koje su, bar u tom segmentu tržišta, istisnule magnetne trake, preuzimajući od njih arhivsku ulogu. Na tržište personalnih računara najveći uticaj je izvršio *IBM*, kada je 1981. godine izbacio *IBM PC* model personalnog računara, koji je bio baziran na mikro-procesorskoj familiji *Intel 8088* (verzija mikro-procesora *Intel 8086*, sa 8 bitnom sabirnicom). Operativni sistem *MS-DOS* (*MicroSoft Disk Operating System*) ovog personalnog računara je proizveo *Microsoft*. *IBM PC* je imao otvorenu arhitekturu, što je omogućilo mnogim proizvođačima da samostalno prave i prodaju njegove verne kopije (*clone*). Personalni računari su otvorili novi i ogroman segment tržišta, sa izuzetno velikim povratnim uticajem na razvoj računara. Način upotrebe personalnih računara je diktirao da oni imaju sopstveno rashlađivanje i da ne zahtevaju posebno klimatizovane prostorije.

Mana personalnih računara je bila da su onemogućili saradnju korisnika, jer je svaki korisnik imao svoj računar, izolovan od ostalih računara. Da bi se omogućila saradnja korisnika, bilo je neophodno povezati personalne računare. Vezivanje personalnih računara kao terminala za mini ili velike računare posredstvom telefonskih linija je omogućavalo korisnicima da očuvaju oblike saradnje, kao što je razmena poruka i datoteka ili deljenje skupih računarskih resursa. U ovakvim okolnostima, personalni računari su emulirali terminale i omogućavali su korisnicima interaktivni rad u režimu ravnomerne raspodele procesorskog vremena mini i velikih računara. Pri tome, na oba kraja telefonske linije, i sa strane personalnog računara i sa strane mini ili velikog računara, je bilo neophodno prisustvo **modema**. Modem je obavljao modulaciju i demodulaciju, odnosno pretvarao digitalne signale jednosmerne struje u analogne signale naizmenične struje i obrnuto.

RAČUNARSKE MREŽE

Potreba međusobnog povezivanja, ne samo personalnih, nego i mini i velikih računara, je dovela do razvoja lokalnih (*LAN*, *Local Area Network*) i globalnih (*WAN*, *Wide Area Network*) mreža računara. Lokalne mreže su omogućavale međusobno povezivanje računara čija udaljenost nije prelazila nekoliko kilometara. Za to su korišćeni posebni komunikacioni kanali, sa propusnošću od nekoliko

miliona bita u sekundi. Globalne mreže su omogućavale međusobno povezivanje računara udaljenih hiljadama kilometara. Za to su korišćene javne telefonske linije sa propusnošću od najviše nekoliko hiljada bita u sekundi. Prve globalne mreže su razvijene za računare treće generacije (*ARPAnet*, *Advanced Research Project Agency*), ali su ušle u raširenu upotrebu tek za računare četvrte generacije.

Mreže računara su omogućile saradnju korisnika raznih računara, ali pod uslovom da korisnici unapred znaju koju vrstu usluga mogu da dobiju na kome od tuđih računara, kao i da poznaju osobine ne samo svog, nego i osobine tuđih računara. To je otežavalo saradnju korisnika, jer su se računari, povezani mrežom, međusobno veoma razlikovali, pošto su poticali od različitih proizvođača i imali različite operativne sisteme. Da bi se bar donekle umanjile komplikacije saradnje korisnika, čiji računari su bili povezani mrežom, odnosno, da bi se smanjio broj činjenica, čije poznavanje je bilo neophodno za ovakvu saradnju, napravljeni su **mrežni operativni sistemi** (*network operating system*). Oni su objedinili sisteme datoteka raznih računara, povezanih mrežom, u jedinstven sistem datoteka. Mrežni operativni sistemi su predstavljali dogradnju postojećih operativnih sistema. Zahvaljujući mrežnim operativnim sistemima, korisnici su mogli da pristupaju datotekama sa drugih, različitih računara kao svojim datotekama. Zadatak mrežnog operativnog sistema je bio da prepozna kada korisnik želi da pristupi datoteci na tuđem računaru i da to omogući bez obzira na razlike između korisničkog i tuđeg računara. Iako su olakšavali saradnju korisnika, mrežni operativni sistemi nisu potpuno sakrivali od korisnika postojanje mreže, jer nije bilo moguće idealno (potpuno) objediniti različite sisteme datoteka u jedinstven sistem datoteka.

SUPER-RAČUNARI

Zahvaljujući tehnologiji visoke integracije, računari četvrte generacije su bili znatno brži od računara treće generacije. Uprkos tome, oni nisu bili dorasli potrebama velikih naučno tehničkih proračuna, koji su sadržali numeričke obrade ogromnog broja podataka i morali da budu završeni u srazmerno kratkom vremenu. Na primer, obimni proračuni budućih temperatura vazduha, zasnovani na složenim simulacionim modelima atmosfere, mogli su da posluže kao osnova vremenske prognoze samo ako su bili završeni u relativno kratkom roku nakon prikupljanja trenutnih temperatura. Da bi se, u okviru postojeće tehnologije, odgovorilo na ovakve zahteve, bilo je potrebno omogućiti što više istovremenih obrada nezavisnih podataka. To nije bilo moguće bez paralelizma u radu procesora. Ovakav paralelizam se mogao ostvariti (1) uvođenjem više procesora opšte namene u organizaciju računara četvrte generacije (*Processor Level Parallelism, PLP*), ali i (2) preklapanjem rada pojedinih sastavnih delova jednog procesora (*Instruction Level Parallelism, ILP*).

Prvi pristup je zasnovan na mogućnosti pravljenja komercijalnih višeprocesorskih računara sa zajedničkom sabirnicom.

Drugi pristup je zasnovan na zapažanju da se izvršavanje mašinskih naredbi svakako sastoji od više relativno nezavisnih aktivnosti, kao što su:

1. dobavljanje mašinske naredbe
2. njeno dekodiranje
3. dobavljanje njenih operanada
4. izvršavanje operacije
5. odlaganje rezultata

Sekvencijalno obavljanje ovih aktivnosti zapošljava u svakom trenutku samo pojedine delove procesora. Razlaganje izvršavanja naredbe na međusobno nezavisne korake iste dužine, jednake ciklusu procesora, i posvećivanje posebnog dela procesora svakom od ovih koraka, omogućuje **preklapanje rada** ovih delova. Pri tome se podrazumeva da su pomenuti delovi ili **stepeni** (*stage*) procesora povezani u **protočne strukture** (*pipeline*). Svaki stepen iz protočne strukture preuzima od svog prethodnika rezultate njegovog rada, a svom sledbeniku iz protočne strukture prosleđuje rezultat svog rada (Slika 11.1.3).

ciklusi	1. naredba	2. naredba	3. naredba	4. naredba	5. naredba	...
1.	dobavljanje naredbe					
2.	dekodiranje naredbe	dobavljanje naredbe				
3.	dobavljanje operanada	dekodiranje naredbe	dobavljanje naredbe			
4.	izvršavanje naredbe	dobavljanje operanada	dekodiranje naredbe	dobavljanje naredbe		
5.	odlaganje rezultata	izvršavanje naredbe	dobavljanje operanada	dekodiranje naredbe	dobavljanje naredbe	
6.		odlaganje rezultata	izvršavanje naredbe	dobavljanje operanada	dekodiranje naredbe	...
7.			odlaganje rezultata	izvršavanje naredbe	dobavljanje operanada	...
8.				odlaganje rezultata	izvršavanje naredbe	...
9.					odlaganje rezultata	...
...						...

Slika 11.1.3 Preklapajući način rada procesora sa protočnom strukturom

Aktivnost svih zaposlenih stepena započinje istovremeno, jer su svi koraci iste dužine. Znači, u prvom koraku rada procesora, prvi stepen dobavlja prvu mašinsku naredbu. Zatim, u drugom koraku, prvi stepen dobavlja drugu mašinsku

naredbu, dok drugi stepen dekodira prvu mašinsku naredbu. U trećem koraku, prvi stepen dobavlja treću mašinsku naredbu, drugi stepen dekodira drugu mašinsku naredbu, a treći stepen dobavlja prvi operand prve mašinske naredbe, i tako dalje.

Organizacija procesora sa protočnom strukturom zahteva punu međusobnu nezavisnost stepena protočne strukture, radi preklapanja njihovih aktivnosti. Pored toga, ona zahteva i pravovremeno (istovremeno) pripremanje podataka za svaki od zaposlenih stepena. Na primer, neophodno je stvoriti uslove da u istom procesorskom ciklusu budu dobavljeni mašinski oblik jedne i operand druge naredbe. Pri tome, svi stepeni protočne strukture nisu uvek zaposleni, jer izvršavanje različitih naredbi obuhvata različit broj koraka. Na primer, za obavljanje naredbe sabiranja celih brojeva dovoljan je samo 1 korak, dok su za obavljanje naredbe sabiranja realnih (*floating-point*) brojeva potrebna bar 4 koraka: (1) poređenje eksponenata, (2) podešavanje frakcija, (3) sabiranje frakcija i (4) normalizacija rezultata. Zato su, na primer, stepeni, posvećeni pojedinim koracima sabiranja realnih brojeva, uvek nezaposleni, kada se izvršavaju samo naredbe celobrojnog sabiranja.

Ostvarenje međusobne nezavisnosti pojedinih stepena protočne strukture utiče na arhitekturu naredbi. Na primer, kod aritmetičkih naredbi i naredbi za rukovanje bitima se uvodi ograničenje da se njihovi ulazni i izlazni operandi mogu nalaziti samo u registrima procesora. Na taj način se istovremeno mogu dobaviti dva operanda jedne naredbe i odložiti rezultat druge naredbe (što mora biti podržano unutrašnjom organizacinom procesora). Ovo ograničenje negativno utiče na ortogonalnost naredbi i zahteva uvođenje posebnih naredbi prenosa podataka za prebacivanje sadržaja iz lokacija radne memorije u registre procesora (*load*) i obratno (*store*). Međutim, prenos podataka između lokacija radne memorije i registara procesora je u konfliktu sa dobavljanjem naredbi, jer je u oba slučaja potrebno pristupanje radnoj memoriji. Ovakav konflikt se razrešava podelom skrivene memorije na deo za naredbe (*instruction cache*) i deo za podatke (*data cache*). To omogućava da dobavljanje jedne naredbe bude istovremeno sa prenosom podataka u procesorski registar ili iz njega.

Pod pretpostavkom da se posmatraju izvršavanja naredbi koja u proseku obuhvataju k koraka i da se svaki korak završi u toku jednog procesorskog ciklusa sa trajanjem od t vremenskih jedinica, srednje vreme izvršavanja jedne naredbe bez preklapanja je:

$$k \cdot t$$

vremenskih jedinica. Za m izvršavanja naredbi bez preklapanja potrebno je:

$$m \cdot k \cdot t$$

vremenskih jedinica.

Pod prethodnim uslovima, za m izvršavanja naredbi sa preklapanjem ($m>k$) potrebno je:

$$(k-1)\cdot t$$

vremenskih jedinica da se napuni protočna struktura i uspostavi puno preklapanje. Nakon toga u svakom od narednih m ciklusa se završi po jedna naredba. Prema tome, za m izvršavanja naredbi sa preklapanjem potrebno je ukupno:

$$(k-1)\cdot t + m\cdot t$$

vremenskih jedinica. Znači, srednje vreme izvršavanja jedne naredbe sa preklapanjem je:

$$\frac{(k-1)}{m}\cdot t + t$$

vremenskih jedinica. Ovo srednje vreme je približno jednako vremenu jednog procesorskog ciklusa za veliko m, jer je tada preklapanje dugotrajno, pa se vreme punjenja protočne strukture raspodeljuje na veliki broj preklopljenih izvršavanja naredbi. Iz prethodnog sledi da se srednje vreme izvršavanja naredbe smanjuje kada k postaje veće, jer se tada izvršavanje naredbe razlaže na više kraćih koraka, pa procesorski ciklus može biti kraći. Međutim, za veliko k broj stepena protočne strukture je veliki, pa je veliko i kašnjenje signala u njoj. Zato praktični razlozi ograničavaju dužinu protočne strukture.

Puni efekat preklapanja izostaje ako se smanji dugotrajnost preklapanja. Do toga dovodi pojava međuzavisnosti naredbi, kada rezultat izvršavanja jedne naredbe predstavlja operand druge naredbe. To dovodi do zaustavljanja punjenja protočne strukture drugom naredbom, dok njen neophodni operand ne postane raspoloživ. Na ovaj način se protočna struktura prazni. Isti efekat imaju izvršavanja (uslovnih) upravljačkih naredbi.

Verovatnoća međuzavisnosti naredbi je proporcionalna dužini protočne strukture. Što je ona duža, sa više stepena, preklapa se izvršavanje više naredbi, pa tako rastu izgledi da se među njima pojave međusobno zavisne naredbe. Znači, na dugotrajnost preklapanja utiče arhitektura naredbi, jer je lakše ostvariti dugotrajna preklapanja izvršavanja jednostavnijih naredbi (za njih je protočna struktura kraća, pa, pošto se preklapa izvršavanje manje naredbi, manja je i verovatnoća njihove međuzavisnosti).

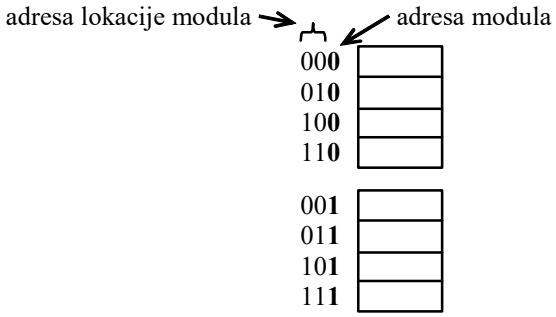
Povećavanju dugotrajnosti preklapanja mogu da doprinesu kompajleri, ako pri generisanju mašinskog programa u njemu raspoređuju mašinske naredbe tako,

da su međusobno zavisne naredbe maksimalno razdvojene. Pri tome ne sme doći do izmene značenja programa.

Dugotrajnost preklapanja se može ostvariti za istovrsne obrade dugačkih nizova podataka, tipičnih za velike naučno tehničke proračune. Za ovakve proračune je tipično i da se izvršavanje iste naredbe ponavlja za dugačke nizove ili **vektore podataka**. Da bi se izbeglo višestruko dobavljanje i dekodiranje iste naredbe, uputno je uvesti **vektorske naredbe**. One omogućuju da se, nakon jednog dobavljanja i dekodiranja, ista operacija primenjuje na vektore podataka. Vektori podataka su smešteni ili u uzastopnim lokacijama radne memorije ili u posebnim vektorskim registrima procesora. U prvom slučaju, mašinski oblik vektorske naredbe sadrži početne adrese vektora podataka, a u drugom slučaju, on sadrži adresu grupe vektorskih registara. Vektorske naredbe se najčešće odnose na aritmetiku realnih brojeva. U toku izvršavanja takvih vektorskih naredbi ostvaruju se dugotrajna preklapanja raznih koraka koji pripadaju obradama raznih elemenata vektora podataka.

Izvršavanje vektorskih naredbi zahteva da se u svakom procesorskom ciklusu pristupa jednom elementu vektora podataka. Ako je korišćena radna memorija sa vremenom pristupa od n procesorskih ciklusa, tada je potrebna **prepletena radna memorija** (*interleaved memory*) u kojoj radnu memoriju sačinjava n nezavisnih memorijskih modula. Pod pretpostavkom da su uzastopni elementi vektora podataka ravnomerno raspodeljeni po lokacijama ovih modula, istovremeno pokretanje čitanja po jednog elementa vektora podataka iz lokacija različitih memorijskih modula, nakon n procesorskih ciklusa, dovodi do preuzimanja n uzastopnih elemenata vektora podataka. Preklapanje isporuke pojedinih od ovih n elemenata sa istovremenim pristupanjem novoj grupi od n uzastopnih elemenata obezbeđuje da srednje vreme pristupa elementu vektora podataka bude približno jednako jednom procesorskom ciklusu.

Ravnomerna raspodela elemenata vektora podataka po raznim memorijskim modulima se automatski ostvaruje kada manje značajni biti adrese memorijske lokacije adresiraju memorijski modul, a značajniji biti ove adrese predstavljaju lokalnu adresu modula (Slika 11.1.4).



Slika 11.1.4 Prepletena radna memorija sa osam lokacija raspodeljenih u dva memorijska modula

Zahvaljujući prethodnoj interpretaciji adresa memorijskih lokacija, prvi element vektora podataka dospeva u nultu lokaciju memorijskog modula 0, drugi u nultu lokaciju memorijskog modula 1 i tako dalje. Prema tome, kada se adresa prvog elementa vektora podataka uputi ka prepletenoj radnoj memoriji (Slika 11.1.4), najznačajniji biti ove adrese istovremeno adresiraju 2 lokacije iz raznih memorijskih modula. Zato je moguće istovremeno pristupiti tim lokacijama. Pod uslovom da se uvek pristupa svim uzastopnim elementima vektora podataka, srednje vreme pristupa svakom elementu odgovara polovini vremena pristupa jednoj lokaciji za posmatrani primer prepletene radne memorije (Slika 11.1.4).

Prepletena radna memorija omogućuje procesoru i kontrolerima da istovremeno pristupaju raznim modulima, jasno, ako postoje nezavisni pristupni putevi, što, takođe, doprinosi bržem radu računara.

Za uspešno korišćenje vektorskih naredbi i prepletene radne memorije, neophodna je podrška kompajlera, radi pretvaranja repetitivnih delova programa, pisanih programskim jezicima visokog nivoa, u vektorske naredbe.

Računari četvrte generacije, čiji su procesori radili u režimu preklapanja i uz to podržavali vektorske naredbe, odskakali su po broju obrađenih podataka u jedinici vremena od ostalih računara četvrte generacije, pa su zaslužili ime **super-računari** (*supercomputer)*.

Super-računari su, pored vektorske jedinice, namenjene za podršku vektorskih naredbi, sadržavali i preklapajuću skalarnu jedinicu, namenjenu za podršku običnih ili skalarnih naredbi. Mogućnosti super-računara su bile veoma impresivne, naročito kada su se posmatrali kratkotrajni periodi u kojima je ostvareno puno preklapanje u radu vektorske jedinice. Pošto su ove mogućnosti bile dorasle zahtevima velikih naučno tehničkih proračuna, super-računari su svojom pojavom otvorili novi segment tržišta računara. Iako je pojava prvih računara čiji su procesori radili u režimu preklapanja vezana za drugu generaciju računara (*IBM 7030* ili *Control Data Corporation CDC 6600*, na primer), a pojava prvih vektorskih računara vezana za treću generaciju računara (*Control Data Corporation*

STring ARray, *CDC STAR-100*, i *Texas Instruments Advanced Scientific Computer*, *TI ASC*), tek je tehnologija visoke integracije stvorila uslove za puni tržišni prodor super-računara. Uspostavljanje stabilnog tržišta super-računara je vezano za pojavu super-računara *Cray-1*, koga je 1976. godine proizveo *Cray Research*. Mogućnosti super-računara su povećavane multipliciranjem procesora, kao u slučaju super-računara *Cray X-MP*, koji se pojavio na tržištu 1985. godine.

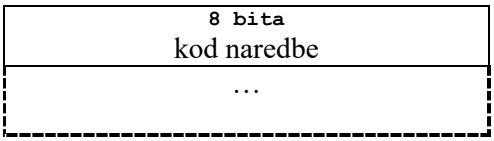
11.2. ISHODI EVOLUCIJE ARHITEKTURE RAČUNARA U PERIODU OKO 1980. GODINE

ARHITEKTURA NAREDBI ZA *DEC VAX11/780*

Arhitektura naredbi za *DEC VAX11/780* podržava adresni prostor od 32 bita, koji je organizovan u bajte. Ona podržava virtuelnu memoriju. Ova arhitektura omogućuje rukovanje binarnim vrednostima velikim 1, 2, 4, 8 i 16 bajta, realnim vrednostima od 4, 8 i 16 bajta (*floating-point*), decimalnim ciframa izraženim kao četvorobitne vrednosti (*binary-coded decimal, BCD*), nizovima znakova (*character string*) i nizovima cifara (*numeric string*). U ovoj arhitekturi je na raspolaganju 16 registara od 32 bita, od kojih jedan služi kao pokazivač frejma (*frame pointer*), drugi služi kao pokazivač steka (*stack pointer*), a treći kao programski brojač. Pored njih, postoji i status registar (*processor status word*), koji sadrži uslovne bite (*condition codes*). Ova arhitektura omogućuje privilegovani način rada procesora sa 3 nivoa privilegija (*kernel mode, executive mode*, *supervisor mode*). Ona podržava registarsko adresiranje, posredno adresiranje sa samouvećanjem, posredno adresiranje sa samoumanjenjem, 4 vrste indeksnog adresiranja, posredno adresiranje, dvostruko posredno adresiranje sa samouvećanjem i 3 vrste indeksnog posrednog adresiranja. Sva prethodna adresiranja se oslanjanju na registre. Pored njih, postoje još 4 adresiranja (*literal addressing mode*), namenjena za male konstante, tako da ukupno ima 16 adresiranja. Adresiranja sa samouvećanjem (samoumanjenjem) podrazumevaju da se sadržaj korišćenog registra automatski uveća (umanji) nakon adresiranja. Dvostruka indirekcija znači da korišćeni registar sadrži adresu memorijske lokacije sa adresom operanda. Indeksna indirekcija znači da je rezultat indeksnog adresiranja adresa memorijske lokacije sa adresom operanda. Kada se indeksno adresiranje koristi za pristupanje elementima nizova, tada se sabiraju početna adresa niza sa proizvodom indeksa traženog elementa i dužine ovog elementa, izražene u bajtima. Ovakav proizvod se naziva **skalirani** (*scaled*) indeks. Kod ostalih vrsta indeksnog adresiranja nema skaliranja, nego se sadržaj registra sabira sa navedenim odstojanjem (*displacement*). Kombinovanje programskog brojača sa posrednim adresiranjem sa samouvećanjem odgovara neposrednom adresiranju. Slično, kombinovanje programskog brojača sa dvostruko posrednim adresiranjem sa samouvećanjem odgovara apsolutnom adresiranju, u kome se koristi apsolutna adresa. Kombinovanje programskog brojača sa indeksnim adresiranjem odgovara relativnom adresiranju, a kombinovanje programskog

brojača sa indeksnim posrednim adresiranjem odgovara relativnom posrednom adresiranju.

Mašinski format naredbi ove arhitekture izgleda:



Mašinski format naredbi obavezno obuhvata bajt sa kodom naredbe. Iza slede eventualno još jedan bajt sa kodom naredbe i bajti za operande. Operanada može biti od 0 do 6. Za sve operande su dozvoljena sva adresiranja, jasno, kada imaju smisla, tako da je ostvarena puna ortogonalnost naredbi i adresiranja. Adresiranja se kodiraju kao kod arhitekture naredbi za *DEC PDP11*, uz razliku da kodovi operanda i registra zahtevaju po 4 bita.

Arhitektura naredbi za *DEC VAX11/780* obuhvata sledeće tipove naredbi:

1. naredbe za rukovanje podacima (na primer, za prenos i konverziju podataka),
2. naredbe za rukovanje bitima (logičke naredbe i naredbe pomeranja),
3. naredbe za celobrojnu binarnu aritmetiku (obuhvaćene sve aritmetičke operacije),
4. naredbe za celobrojnu decimalnu aritmetiku (obuhvaćene sve aritmetičke operacije),
5. naredbe za aritmetiku realnih brojeva (obuhvaćene sve aritmetičke operacije),
6. upravljačke naredbe i
7. sistemske naredbe.

ARHITEKTURA NAREDBI ZA *INTEL 8086*

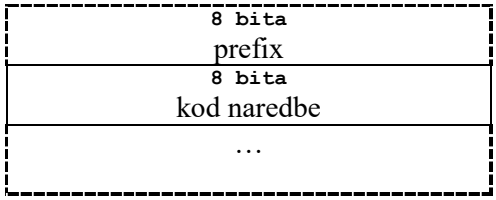
Arhitektura naredbi za *INTEL 8086* podržava adresni prostor od 20 bita, koji je organizovan u bajte. Ovaj adresni prostor nije raspoloživ ceo, nego su uvek raspoloživa samo njegova 4 segmenta sa 16 bitnim lokalnim adresnim prostorima: segment naredbi (*code segment*), segment steka (*stack segment*), segment podataka (*data segment*) i dodatni segment (*extra segment*). Prvi segment je namenjen za pristup mašinskim naredbama, drugi za pristup steku, a poslednja dva za pristup podacima. Položaj pristupačnih segmenata određuju njihove početne ili bazne (*base*) adrese koje se nalaze u 16 bitnim segmentnim registrima *CS* (*code segment*), *SS* (*stack segment*), *DS* (*data segment*) i *ES* (*extra segment*). Zavisno od sadržaja njihovih segmentnih registara, segmenti se mogu i preklapati. Adresa memorijske lokacije se određuje kao suma skalirane bazne adrese segmenta i njegove unutrašnje (*offset*) adrese. Svrha skaliranja bazne adrese je da se ona dopuni sa 4 bita i postane

20 bitna adresa pre sabiranja. Zato se skaliranje sastoji od množenja bazne adrese sa 16, odnosno, od njenog dopunjavanja s desna sa 4 nule.

Ova arhitektura omogućuje rukovanje binarnim vrednostima velikim 1 i 2 bajta, decimalnim ciframa izraženim kao četvorobitne vrednosti (*packed decimal*) i nizovima znakova (*character string*). Pored 4 segmentna registra, ova arhitektura nudi još 4 registra podataka *AX*, *BX*, *CX* i *DX* od 16 bita. Značajniji (*high*) i manje značajan (*low*) bajt svakog od ovih registara se može posmatrati kao zaseban registar od 8 bita. Njihove oznake su *AH*, *AL*, *BH*, *BL*, *CH*, *CL*, *DH* i *DL* (respektivno). Uloga registara podataka je delom unapred određena, jer se oni koriste kao podrazumevajući registri. Tako su *AX* i *DX* podrazumevajući registri za množenje, deljenje i ulaz-izlaz, a *CX* registar je podrazumevajući za rukovanje nizovima znakova. Postoje i 4 adresna registra od 16 bita: *SP* (*stack pointer*), *BP* (*base pointer*, *frame pointer*), *SI* (*source index register*) i *DI* (*destination index register*). I njihova uloga je delom unapred određena. Na kraju, postoje još programski brojač *IP* (*instruction pointer*) od 16 bita i status registar *FLAGS* od 8 bita koji sadrži uslovne bite (*condition code*).

Arhitektura naredbi za *INTEL 8086* podržava neposredno adresiranje, registarsko adresiranje, direktno adresiranje, posredno adresiranje (pomoću registara *BX*, *BP*, *SI* i *DI*) i 3 vrste indeksnog adresiranja. Dve vrste indeksnog adresiranja se razlikuju samo po upotrebljenim registrima (*indexed* adresiranje koristi registre *SI* i *DI*, a *base* adresiranje koristi registre *BX* i *BP*). Treća uključuje po dva registra u indeksno adresiranje (*based indexed* adresiranje obuhvata parove registara, sastavljene od *BX* ili *BP* registara i od *SI* ili *DI* registara).

Mašinski format naredbi ove arhitekture izgleda:



Mašinski format naredbi obavezno obuhvata bajt sa kodom naredbe. Njemu može da prethodi prefiks koji, na primer, menja podrazumevajući segment za naredbu sa ovim prefiksom. Pri tome se podrazumeva da se svaka naredba i svako od njenih adresiranja odnose na neki segment. Tako se upravljačke naredbe odnose na segment naredbi, naredbe za rukovanje stekom na segment steka, a ostale naredbe uglavnom na segment podataka ili na dodatni segment. Iza koda naredbe mogu da slede do 4 dodatna bajta različitog značenja. Oni mogu da sadrže:

1. unutrašnju adresu od 16 bita i baznu adresu segmenta od 16 bita,
2. oznake dva registra (od kojih je jedan namenjen za posredno adresiranje) i, eventualno, odstojanje (*displacement*) od 8 ili 16 bita,
3. neposredni operand od 8 ili 16 bita, ali i

4. relativnu adresu od 8 bita.

Vrsta dozvoljenih operanada se menja od naredbe do naredbe, tako da ne postoji ortogonalnost naredbi i adresiranja.

Arhitektura naredbi za *INTEL 8086* obuhvata sledeće tipove naredbi:

1. naredbe za rukovanje podacima (na primer, za prenos i konverziju podataka),
2. naredbe za rukovanje bitima (logičke naredbe i naredbe pomeranja),
3. naredbe za celobrojnu binarnu aritmetiku (obuhvaćene sve aritmetičke operacije),
4. naredbe za celobrojnu decimalnu aritmetiku (obuhvaćene sve aritmetičke operacije),
5. upravljačke naredbe,
6. sistemske naredbe i
7. ulazno-izlazne naredbe.

SEGMENTNA ORGANIZACIJA RADNE MEMORIJE

Segmentna organizacija radne memorije, pomenuta kod arhitekture naredbi za *INTEL 8086*, omogućuje racionalno korišćenje radne memorije. Na primer, ako jedan segment sadrži mašinske naredbe, drugi stek a treći podatke, tada segment naredbi mogu da dele slike svih procesa koje odgovaraju izvršavanju istog programa. Za takvo korišćenje segmenata važno je osigurati zaštitu koja sprečava:

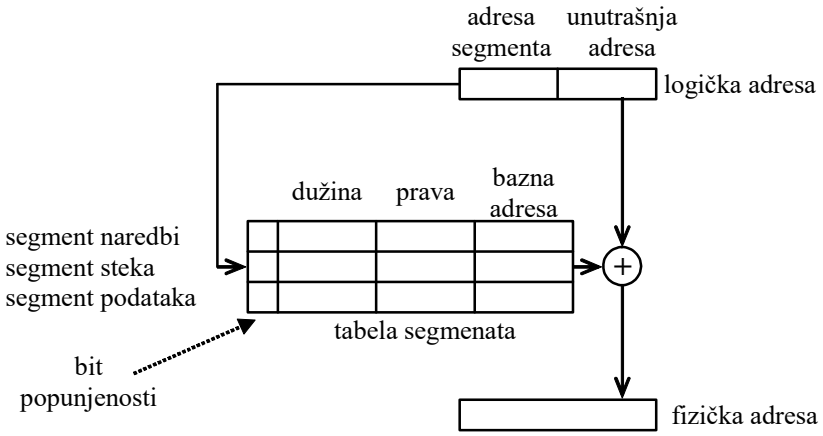
1. da proces izlazi van svojih segmenata i
2. da se segmenti nenamenski koriste.

Zaštita od izlaska van segmenta se može ostvariti primenom ideje graničnog i baznog registra, jer svaki segment ima svoju dužinu i svoju baznu adresu.

Segment naredbi se namenski koristi, ako mu se pristupa samo radi izvršavanja naredbi, dok se segmenti steka i podataka namenski koriste ako im se pristupa radi čitanja ili pisanja memorijskih lokacija. Sprečavanje nenamenskog korišćenja segmenata zahteva uvođenje **prava pristupa** segmentu, kao što su pravo izvršavanja, pravo čitanja i pravo pisanja segmenta.

Ispravno korišćenje segmenata podrazumeva da se za svaki proces veže posebna **tabela segmenata**. Za svaki segment procesa u ovakvoj tabeli segmenata postoji poseban element sa baznom adresom segmenta, dužinom segmenta, pravima pristupa segmentu i podatkom da li je element popunjen.

Logičke adrese iz adresnog prostora segmentirane radne memorije se sastoje od **adrese segmenta** i od njegove unutrašnje adrese. Adresa segmenta indeksira element tabele segmenata, radi korišćenja njegovog sadržaja za proveru da unutrašnja adresa nije veća od dužine segmenta i da proces ima pravo na traženi pristup adresiranoj lokaciji segmenta. Ako je provera uspešna, fizička adresa nastaje sabiranjem (ili spajanjem) bazne adrese segmenta i njegove unutrašnje adrese (Slika 11.2.1).

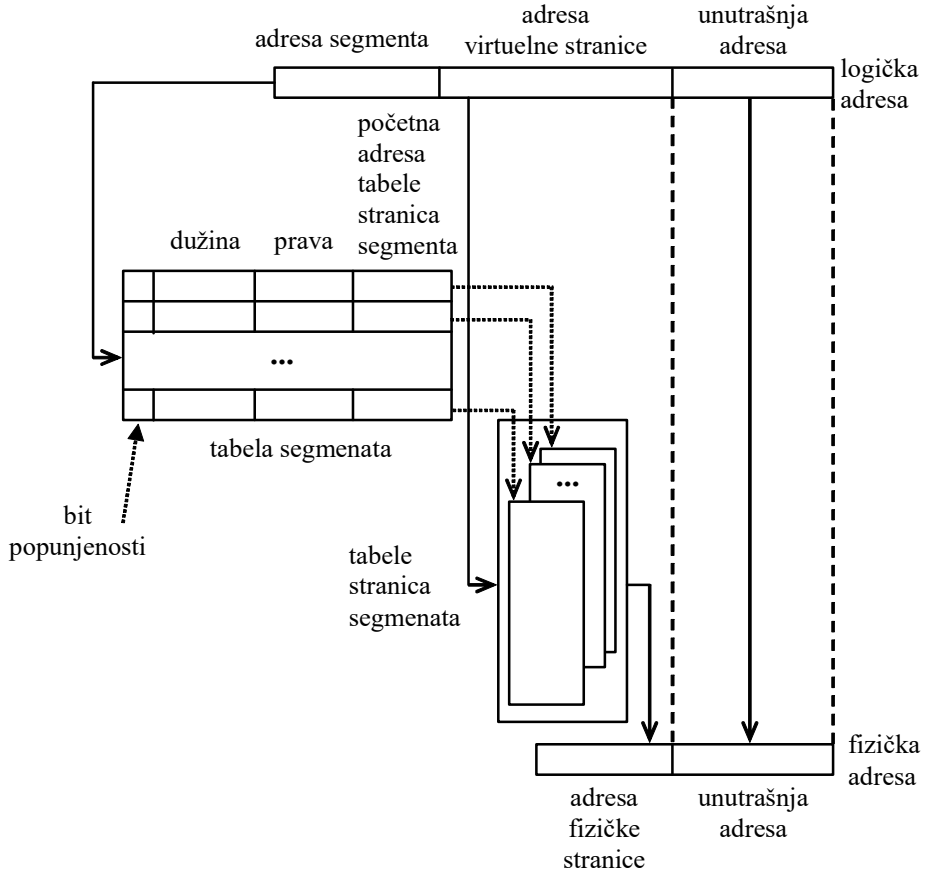


Slika 11.2.1 Pretvaranje logičke adrese u fizičku kod segmentirane radne memorije

Prethodno opisana **osnovna segmentacija** se pretvara u **punu segmentaciju**, ako se broj segmenata poveća, tako da biblioteke potprograma ili grupe promenljivih mogu biti smeštene u zasebne segmente. Na ovaj način je moguće ostvariti deljenje pojedinih biblioteka potprograma ili grupa promenljivih između raznih procesa. U ovom slučaju ima smisla pojedinim procesima davati samo pojedinačna prava pristupa pojedinim segmentima, na primer samo pravo čitanja deljenog segmenta sa grupom promenljivih. Mogućnost deljenja velike biblioteke potprograma je uslov za racionalno korišćenje ne samo radne, nego i masovne memorije. Takva **deljena biblioteka** (*shared library*) se ne mora uopšte linkovati za pojedine programe, pa na taj način ne mora zauzimati prostor u raznim izvršnim datotekama. Dovoljno je da svaka od izvršnih datoteka, koja sadrži pozive potprograma deljene biblioteke, sadrži i informaciju da je deljena biblioteka potrebna za vreme izvršavanja programa. Pomenuta informacija omogućuje da se na početku izvršavanja programa proveri da li se potrebna deljena biblioteka nalazi u nekom od segmenata radne memorije. Ako se ne nalazi, ona se puni u segment radne memorije koji je prethodno za nju obezbeđen. Nakon provere i njenog punjenja, u odgovarajući element tabele segmenata se smeštaju dužina segmenta deljene biblioteke, prava pristupa ovom segmentu i njegova bazna adresa. Posle toga mogu se pozivati potprogrami ove biblioteke kao da je ona statički linkovana za izvršavani program. Povezivanje biblioteke potprograma za program neposredno pre njegovog izvršavanja se naziva **dinamičko linkovanje** (*dynamic linking*). Deljene biblioteke se nazivaju i **dinamički linkovane biblioteke** ili ***DLL*** (*dynamic link library*).

Za segmentaciju je zgodno da se osloni na virtuelnu memoriju da bi segmenti mogli biti veći od fizičke memorije. Ovakav spoj se naziva **stranična segmentacija** (*paged segmentation*), jer se segmenti sastoje od celog broja stranica, pa se njihova dužina izražava kao broj stranica. U ovom slučaju elementi tabele segmenata ne

sadrže baznu adresu segmenta, nego početnu adresu tabele stranica dotičnog segmenta, a logička adresa se sastoji od adrese segmenta, od adrese virtuelne stranice i od unutrašnje adrese. Adresa segmenta indeksira element tabele segmenata koji omogućuje proveru da li je adresa virtuelne stranice ispravna i da li je traženi pristup dozvoljen. U slučaju provere sa pozitivnim ishodom, indeksirani element tabele segmenata sadrži početnu adresu tabele stranica segmenta. Njene elemente indeksira adresa virutelne stranice, radi pronalaženja adrese fizičke stranice i njenog spajanja sa unutrašnjom adresom (Slika 11.2.2).



Slika 11.2.2 Pretvaranje logičke adrese u fizičku kod stranične segmentacije

ARHITEKTURA NAREDBI ZA *INTEL 80386*

Arhitektura naredbi za *INTEL 80386* podržava fizički adresni prostor od 24 bita, koji je organizovan u bajte. Ona podržava segmentaciju, virtuelnu memoriju i straničnu segmentaciju. Segmentacija uvodi logički adresni prostor od 16383 segmenta sa 20 bitnim lokalnim adresnim prostorima. Istovremeno je pristupačno

samo 6 segmenta: segment naredbi, segment steka i 4 segmenta podataka. Podaci o segmentima se nalaze u tabelama segmenata i obuhvataju 32 bitnu baznu adresu, segmenta, njegova prava pristupa i njegovu 20 bitnu dužinu (izraženu u jedinicama 1 bajt ili 4 kilobajta). Postoji jedna globalna tabela segmenata za celi računar, a za svaki proces može biti vezana lokalna tabela segmenata. Adrese pristupačnih segmenata se nalaze u 16 bitnim segmentnim registrima *CS* (*code segment*), *SS* (*stack segment*), *DS* (*data segment*), *ES* (*extra segment*), *FS* i *GS*. Sadržaji ovih registara indeksiraju tabelu segmenata, radi pretvaranja logičke adrese u lineranu adresu, koja odgovara fizičkoj ili virtuelnoj adresi, što zavisi od okolnosti. Logička adresa se sastoji od 16 bitne adrese segmenata, koja se nalazi u nekom od segmentnih registara, i od 32 bitne unutrašnje (*offset*) adrese segmenta. Linearna adresa ima 32 bita i nastaje kao suma 32 bitne bazne adrese segmenta i njegove 32 bitne unutrašnje adrese.

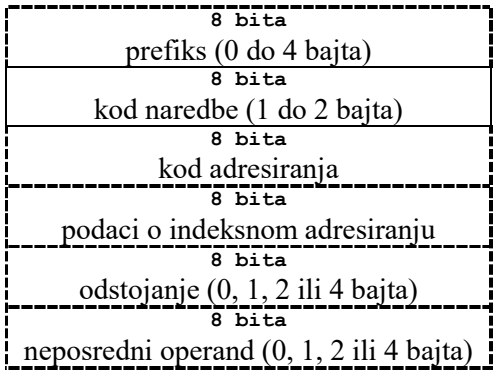
Virtuelna memorija ima tabelu stranica u dva nivoa. Virtuelna adresa se sastoji od 10 bitnog indeksa tabele stranica sa prvog nivoa, 10 bitne adrese virtuelne stranice i 12 bitne unutrašnje adrese.

Stranična segmentacija kombinuje prethodno opisanu segmentaciju i virtuelnu memoriju, tako da se prvo logička adresa pretvara u linearnu, odnosno virtuelnu adresu, a onda se virtuelna adresa pretvara u fizičku adresu.

Ova arhitektura omogućuje rukovanje binarnim vrednostima velikim 1, 2 i 4 bajta, decimalnim ciframa izraženim kao četvorobitne vrednosti (*binary coded decimal, BCD*) i nizovima znakova (*character string*). Pored 6 segmentnih registra, ova arhitektura nudi još 4 registra podataka *EAX*, *EBX*, *ECX* i *EDX* od 32 bita. Postoje i 4 adresna registra od 32 bita: *ESP* (*stack pointer*), *EBP (base pointer*, *frame pointer*), *ESI* (*source index register*) i *EDI* (*destination index register*). Na kraju, postoje još programski brojač *EIP* (*instruction pointer*) od 32 bita i status registar *EFLAGS* od 32 bita koji sadrži uslovne bite (*condition code*). Ova arhitektura omogućuje privilegovani način rada procesora sa 3 nivoa privilegovanog rada (*kernel, system services*, *custom extensions*).

Arhitektura naredbi za *INTEL 80386* podržava neposredno adresiranje, registarsko adresiranje, direktno adresiranje, posredno adresiranje i 3 vrste indeksnog adresiranja. Dve vrste indeksnog adresiranja se razlikuju po tome što jedna ne uključuje, a druga uključuje skaliranje. Treća vrsta indeksnog adresiranja podrazumeva istovremeno korišćenje obe prethodne vrste indeksnog adresiranja.

Mašinski format naredbi ove arhitekture izgleda:



Mašinski format naredbi obavezno obuhvata bajt sa kodom naredbe. Njemu mogu da prethode do 4 bajta prefiksa koji, na primer, omogućuju menjanje podrazumevajućeg segmenta za naredbu čiji kod sledi iza prefiksa. Pri tome se podrazumeva da se svaka naredba i svako od njenih adresiranja odnose na neki segment. Tako se upravljačke naredbe odnose na segment naredbi, naredbe za rukovanje stekom na segment steka, a ostale naredbe uglavnom na segment podataka ili na dodatne segmente. Iza koda naredbe mogu da slede kod adresiranja, podaci o indeksnom adresiranju, poput podatka o skaliranju, iznos odstojanja (*displacement*) i neposredni operand.

Arhitektura naredbi za *INTEL 80386* obuhvata sledeće tipove naredbi:

1. naredbe za rukovanje podacima (na primer, za prenos i konverziju podataka),
2. naredbe za rukovanje bitima (logičke naredbe i naredbe pomeranja),
3. naredbe za celobrojnu binarnu aritmetiku (obuhvaćene sve aritmetičke operacije),
4. naredbe za celobrojnu decimalnu aritmetiku (obuhvaćene sve aritmetičke operacije),
5. upravljačke naredbe,
6. sistemske naredbe i
7. ulazno-izlazne naredbe.

Ova arhitektura nudi posebnu podršku multiprogramiranju (*multitasking*) i dibagiranju. Ona emulira arhitekturu naredbi za *INTEL 8086*.

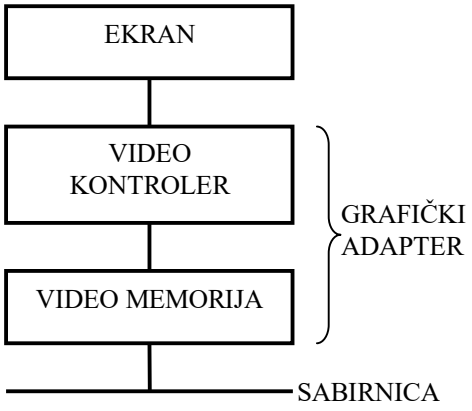
PRINCIP RADA MIŠA

Miš registruje svoje pomeranje po x ili y osi pomoću mehaničkih ili nekih drugih davača pozicije. Kada to pomeranje bude veće od neke unapred zadane vrednosti, on ka računaru šalje poruku sa podacima o relativnoj promeni pozicije. Ovakva poruka sadrži izmenu x i y pozicije, kao i stanje dirki miša. Do slanja ovakve poruke dolazi i kada se izmeni stanje bilo koje dirke miša.

OSOBINE GRAFIČKIH TERMINALA

Osobine grafičkog terminala zavise od njegovog **grafičkog adaptera** koji u svom sastavu ima poluprovodničku **video memoriju** i **video kontroler**. Video memorija sadrži stanja piksela koja se prikazuju na ekranu, a video kontroler upravlja prikazivanjem ovih piksela na ekranu. Od broja lokacija video memorije zavisi broj piksela koji može biti prikazan. Ako je grafički terminal crno beli (monohromatski), za predstavljanje svakog piksela dovoljan je 1 bit video memorije, ili, eventualno, 1 bajt za predstavljanje raznih nijansi sivog. Za grafičke terminale u boji potrebna su 3 bajta za predstavljanje jednog piksela, po jedan bajt za predstavljanje nijanse svake od 3 osnovne boje.

Video memorija ima dva ulaza (*dual port memory*). Jedan od njih je okrenut sabirnici i omogućuje pristupanje lokacijama video memorije, radi preuzimanja ili izmena njihovih sadržaja. To je potrebno za operacije, kao što je, na primer, pomeranje prozora po ekranu grafičkog terminala (*bit BLT operation*, *bit BLock Transfer*). Drugi ulaz video memorije je okrenut ka video kontroleru (ekranu) i omogućuje periodično osvežavanje ekrana (Slika 11.2.3).



Slika 11.2.3 Dvoulazna video memorija i video kontroler

Propusnost drugog ulaza je znatno viša od propusnosti prvog ulaza, zbog velike brzine osvežavanja ekrana. Zbog toga, na ekranu može da se pojavi samo delimično izmenjen sadržaj video memorije. Da bi se to sprečilo, i ujedno podržala animacija, u grafički terminal se ugrađuju dve video memorije, tako da se sadržaj jedne prikazuje na ekranu, dok se menja sadržaj druge. Pošto veličina video memorije značajno utiče na cenu grafičkog terminala, važno je pronaći način da se smanji broj bita po pikselu. To se može postići, na primer, kada istovremeno nisu potrebne sve nijanse boja iz palete, nego samo njihov mali podskup. Tako, ako su potrebne samo 4 odabrane nijanse od, na primer, 2^{24} nijansi, tada su za svaki piksel

dovoljna 2 bita. Pri tome, sadržaji ova 2 bita indeksiraju elemente posebne **tabele boja** (*color palette*, *color table*, *color map*, *video look-up table*), koja sadrži potpune kodove nijansi (znači, svaki od 4 elementa ove tabele je velik 3 bajta). U ovakvoj organizaciji video memorije, zadatak programera je da utiče i na piksele, ali i na tabelu boja.

Grafički terminal se nalazi neposredno uz računar (nije fizički udaljen od njega), jer je video memorija grafičkog terminala vezana za sabirnicu računara.

PRINCIPI RADA LOKALNIH MREŽA

Lokalne mreže omogućuju prenos poruka, sastavljenih od jednog ili više paketa. Prenos poruka se odvija u skladu sa posebnim komunikacionim protokolima, koji precizno opisuju ponašanje pošiljaoca i primaoca poruke. Na primer, ovakvi protokoli određuju kako se poruke rastavljaju na pakete u toku slanja i kako se od paketa obrazuju poruke u toku prijema. Dve najpoznatije vrste lokanih mreža su **eternet** (*Ethernet*) i **mreža sa putujućim žetonima** (*token ring*, *token bus*). I jedna i druga vrsta lokalnih mreža se sastoji od prenosnika signala i od kontrolera koji upravljaju prenosom signala. One se razlikuju po načinu na koji kontroleri stiču pravo da emituju pakete. I kod jedne i kod druge vrste lokalnih mreža u svakom momentu najviše jedan kontroler ima pravo emitovanja, a svi ostali kontroleri slušaju. Za mreže sa putujućim žetonom se podrazumeva da se svi paketi prenose od kontrolera do kontrolera u predodređenom redosledu i uvek u istom smeru. Pri tome, između kontrolera cirkuliše jedan poseban paket sa žetonom. Kontroler, koji primi ovaj paket, ima pravo da emituje jedan svoj paket. Nakon toga, on prosleđuje dalje paket sa žetonom, što uradi i kada nema sopstveni paket za emitovanje.

Za eternet mreže je karakteristično da kontroleri prate stanje prenosnika signala i da započinju emitovanje čim ustanove da nema tuđe emisije. Pri tome, oni nastavljaju da prate stanje prenosnika signala i u toku svoje emisije, radi, eventualnog, otkrivanja **kolizije**, odnosno, istovremenog početka više emisija. U slučaju kolizije, svaki od kontrolera, koji su istovremeno započeli emisiju, odlaže emitovanje za period slučajne dužine, radi izbegavanja ponovne kolizije. Ako nema kolizije, kontroler nastavlja sa emitovanjem do kraja paketa, koji ukupno sadrži do 1530 bajta. Ovakav način dobijanja prava za emitovanje se označava skraćnicom *CSMA/CD* (*Carrier Sense Multiple Access with Collision Detection*).

Eternet mreže su najraširenija vrsta lokalnih mreža. Njihova dužina se kreće oko jednog kilometra. Na jednu eternet mrežu se kači do 1024 računara, što je definisano formatom paketa.

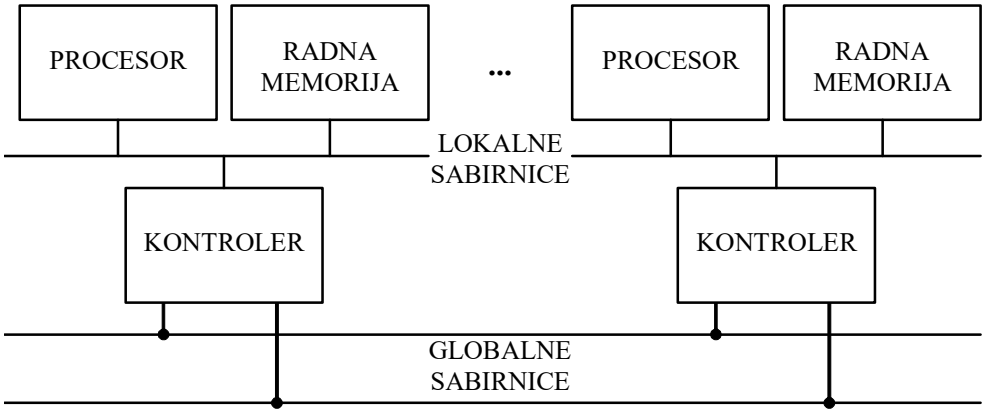
Eternet mreže nisu dobre za velika opterećenja (preko 50%), jer im tada propusnost opada brže od povećanja opterećenja zbog uticaja kolizija, čiji broj ubrzano raste na većim opterećenjima. Mreže sa putujućim žetonom su dobre za velika opterećenja, jer i za najveća opterećenja garantuju minimalnu propusnost

svakom pošiljaocu poruka. Mreže sa putujućim žetonom nisu dobre za mala opterećenja, jer ne omogućuju dobro iskorišćenje propusnosti.

VIŠEPROCESORSKI RAČUNARI SA ZAJEDNIČKOM SABIRNICOM

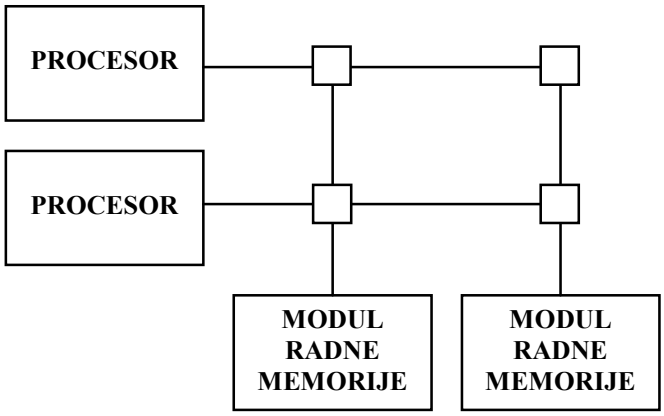
Standardne sabirnice, koje sadrže arbitar i podržavaju više aktivnih strana, pružaju prirodnu osnovu za izgradnju višeprocesorskih računara, jer dozvoljavaju da više procesora bude istovremeno zakačeno na njih. Mana ovakvog rešenja je da se na sabirnicu može zakačiti mali broj procesora, zbog pojave problema **zagušenja sabirnice**. Ovaj problem može da se reši korišćenjem više sabirnica, ali po cenu duže razmene podataka između organizacionih komponenti računara, zakačenih na razne sabirnice. Pri tome se podrazumeva da postoji jedinstven adresni prostor, tako da svaki procesor može da pristupi bilo kojoj lokaciji radne memorije, uz različita vremena pristupa raznim lokacijama.

Slika 11.2.4 sadrži prikaz organizacije višeprocesorskog računara sa više lokalnih i globalnih sabirnica. Na ovoj ideji je zasnovan, na primer, *NonStop* multiprocesor, koga je krajem sedamdesetih godina proizveo *Tandem*.



Slika 11.2.4 Organizacija višeprocesorskog računara sa više lokalnih i globalnih sabirnica

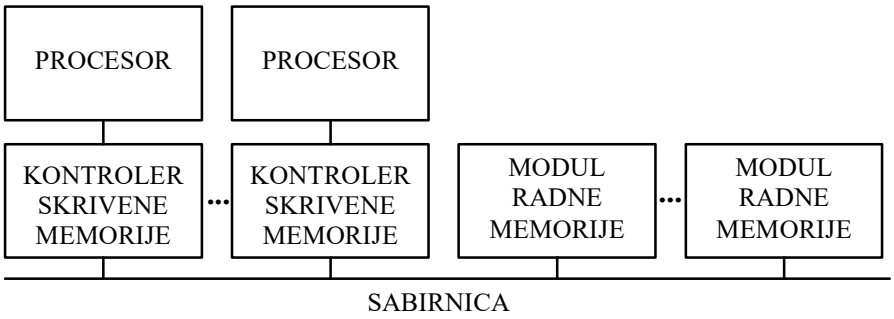
Ideja korišćenja više sabirnica ima krajnost u **unakrsnoj matrici** (*crossbar*), Ona obezbeđuje da između svakog procesora i modula radne memorije postoji posebna sabirnica (Slika 11.2.5).



Slika 11.2.5 Unakrsna matrica

Kvadrati na mestima susreta horizontalnih i vertikalnih sabirnica unaksne matrice simbolizuju prekidače, koji omogućuju dinamičko uspostavljanje direktinih veza između pojedinih procesora i modula radne memorije. Ovakvi prekidači podržavaju istovremenu vezu svakog procesora sa različitim modulom radne memorije. Unakrsne matrice su korišćene kod velikih računara treće generacije za povezivanje raznih organizacionih komponenti (*Burroughs B5000*, *CDC Cyber-70 model 74*). Mana unakrsnih matrica je da za veliki broj prekidača one imaju visoku cenu, veliko kašnjenje i malu pouzdanost.

Skrivena memorija doprinosi smanjuju opterećenja sabirnice, jer dok god se u skrivenoj memoriji nalaze kopije sadržaja adresiranih lokacija, nema potrebe za izlazak na sabirnicu, radi pristupanja radnoj memoriji. Zato se skrivena memorija koristi za izbegavanje zagušenja sabirnice u višeprocesorskim računarima, kod kojih su svi procesori zakačeni na istu sabirnicu (Slika 11.2.6).



Slika 11.2.6 Višeprocesorski računar sa jednom sabirnicom i skrivenim memorijama

Da bi skrivene memorije mogle biti korišćene za odlaganje zagušenja sabirnice (Slika 11.2.6), neophodno je obezbediti **usaglašenost sadržaja** raznih skrivenih memorija (*cache coherency*). Do neusaglašenosti sadržaja raznih skrivenih memorija dolazi kada razni procesori istovremeno menjaju razne kopije sadržaja iste lokacije. Ovakve neusaglašenosti se sprečavaju, ako svi kontroleri skrivenih memorija prate aktivnosti na sabirnici (*snoopy cache*), radi poništavanja (*write invalidate*) ili izmene (*write broadcast*) kopija sadržaja lokacija, čije adrese su se pojavile na sabirnici u okviru transakcije pisanja. Pri tome je radna memorija izdeljena u module, radi primene podeljenih transakcija. *Symmetry* multiprocesor, koga je krajem osamdesetih godina na tržište izbacio *Sequent*, predstavlja primer multiprocesora u okviru koga su mogla da se zakače do 32 *Intel 80386* procesora na istu sabirnicu sa 64 linije podataka. Pri tome se između svakog od ovih procesora i sabirnice nalazila posebna 64 kilobajtna skrivena memorija (*write back*, *write invalidate*).

Osobenost višeprocesorskih računara je da se kod njih problem sinhronizacije procesa ne može rešiti onemogućenjem prekida, jer kod njih nesinhronizam procesa nije posledica prekida i time izazvanih nepredvidivih preključivanja, nego je posledica stvarnog paralelizma procesa. Pošto, dva procesa mogu biti istovremeno aktivna na raznim procesorima višeprocesorskog računara, oni mogu pristupati deljenim memorijskim lokacijama u nepredvidivom redosledu. To znači da, u primeru zauzimanja resursa, dva procesa mogu istovremeno započeti zauzimanje istog resursa. U ovakvim okolnostima neophodnu nedeljivost čitanja i pisanja lokacije stanja resursa obezbeđuje zaključavanje sabirnice za vreme obavljanja ovih operacija. To se može ostvariti pomoću posebne **upravljačke linije za zaključavanje sabirnice** (*lock*). Ovo zaključavanje sabirnice omogućuje posebna mašinska naredba (*read-modify-write*) koja zaključa sabirnicu, na primer, u toku čitanja i pisanja jedne memorijske lokacije.

Važno pitanje je šta uraditi sa procesom višeprocesorskog računara nakon neuspešnog pokušaja zaključavanja nekog resursa. Ako će resurs biti zauzet kraće od vremena potrebnog za preključivanje, tada ima smisla ponavljati pokušaje zaključavanja resursa dok se u tome ne uspe. Ovakvo ponavljanje zaključavanja resursa predstavlja oblik radnog čekanja (*spinning*, *spin lock*, *spin waiting*). Ono je prihvatljivo samo ako je kratkotrajno. Međutim, čak i kada je kratkotrajno, ono može da zaguši sabirnicu, ako se više procesa, sa raznih procesora, nalazi istovremeno u ovakvom radnom čekanju.

11.3. PITANJA

1. Šta je činilo tehnološku osnovu računara četvrte generacije?
2. Zašto je tehnologija visoko integrisanih kola istisnula tehnologiju integrisanih kola?
3. Zašto su poluprovodničke memorije istisnule iz upotrebe memorije sa magnetnim jezgricama?

4. Šta karakteriše poluprovodničke memorije?
5. Šta je karakterisalo četvrtu generaciju računara?
6. Šta je mikro-kontroler?
7. Šta je karakterisalo personalne računare?
8. Šta je karakterisalo računarske mreže?
9. Šta je karakteristično za protočnu strukturu?
10. Čemu je približno jednako srednje vreme izvršavanja jedne naredbe u režimu rada sa dugotrajnim preklapanjem?
11. Šta doprinosi dugotrajnosti preklapanja?
12. Šta karakteriše super-računare?
13. Šta kod segmentne organizacije radne memorije dele procesi, koji odgovaraju istom programu?
14. Koliko delova ima logička adresa iz segmentirane radne memorije?
15. Šta sadrže elementi tabele segmenata?
16. Koja su prava pristupa segmentu?
17. Kako se pretvara logička adresa iz segmentirane radne memorije u fizičku?
18. Šta omogućuje puna segmentacija?
19. Koliko delova ima logička adresa kod stranične segmentacije?
20. Kako se pretvara logička adresa kod stranične segmentiacije u fizičku?
21. Zašto je uvedena stranična segmentacija?
22. Kada miš šalje poruku?
23. Šta karakteriše video memoriju grafičkog terminala?
24. Šta određuje tabela boja grafičkog terminala?
25. Koliko se može prikazati nijansi boja i koliko je potrebno bita za piksel, ako tabela boja ima 4 elementa?
26. Koje su najpoznatije vrste lokalnih mreža?
27. Po čemu se razlikuju etetnet i mreže sa putujućim žetonom?
28. Koje su lokalne mreže dobre za velika, a koja za mala optrećenja?
29. Kako se rešava problem zagušenja sabirnice kod višeprocesorskih računara sa zajedničkom sabirnicom?
30. Kako se rešava problem sinhronizacije procesa kod višeprocesorskih računara sa zajedničkom sabirnicom?

12. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 1990. GODINE

12.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 1990. GODINE

Pojava tehnologije veoma visoke integracije (*VLSI*, *Very Large Scale Integration*), koja je omogućila smeštanje više miliona tranzistora na jedan čip, stvorila je uslove za dalje poboljšavanje mogućnosti računara. Tako su se pojavili još moćniji super-računari, poput *Cray Y-MP*, na primer, a veliki računari su dobili neke osobine super-računara, kao što su vektorske naredbe, kod računara iz *IBM ESA/370* serije, nastale kao proširenje *IBM 370* familije računara. Najveći mini-računari su se približili, po mogućnostima, velikim računarima, kao *DEC VAX 8800*, a mikro-računari, zasnovani, na primer, na mikro-procesorima *Intel 80486* ili *Motorola 68040*, su dobili obeležja mini i velikih računara, kao što je podrška aritmetici realnih brojeva, na primer. Ipak, cene super, velikih, mini i mikro-računara su se i dalje međusobno razlikovale približno za po red veličine. To je bilo posledica činjenice da su veliki računari bili namenjeni za najveće poslovne obrade, koje su zahtevale podršku stotinama terminala ili disk jedinica, dok je zadatak mikro-računara bio da opslužuju samo po jednog korisnika, bez obzira da li je reč o slabijem personalnom računaru sa *MS-DOS* operativnim sistemom ili o moćnijoj radnoj stanici sa *UNIX* operativnim sistemom. Mini-računari su pokrivali primene između prethodne dve krajnosti, a korišćenje super-računara je bilo ograničeno na velike naučno tehničke proračune.

RISC ARHITEKTURA

Zahvaljujući primeni tehnologije veoma visoke integracije, mikro-programske memorije su postale dovoljno velike da podrže i najsloženiju arhitekturu naredbi, pa je složenost arhitekture naredbi velikih, mini i mikro-računara dostigla svoj vrhunac. Skupovi naredbi su sadržali više stotina naredbi, a podržani su na desetine adresiranja i tipova podataka. Međutim, tehnologija veoma visoke integracije je omogućila i znatno skraćenje procesorskih ciklusa, pod uslovom da se upravljanje procesorom osloni na ožičenu upravljačku jedinicu. Radi toga je bilo neophodno temeljito pojednostavljenje *CISC* arhitekture naredbi. Ovo pojednostavljenje se zasnivalo na smanjenju broja naredbi u skupu naredbi, kao i na smanjenju broja podržanih adresiranja i tipova podataka. Računari sa pojednostavljenom arhitekturom naredbi su označeni skraćenicom *RISC* (*Reduced Instruction Set Computer*). Ovako pojednostavljena arhitektura naredbi je dozvolila da se upravljanje procesorom prepusti ožičenoj upravljačkoj jedinici i da se u značajnijoj meri ostvari preklapanje rada nezavisnih delova procesora. Sličan pristup se već pokazao delotvornim kod procesora super-računara. Njihova arhitektura naredbi je takođe bila jednostavna i ona je usmerila razvoj *RISC*

arhitekture. Na taj način je ne samo skraćen procesorski ciklus, nego je i smanjen prosečan broj procesorskih ciklusa po naredbi sa 5 do 10, koliko je iznosio za *CISC* procesore, na svega 1 do 2 procesorska ciklusa, koliko je iznosio za *RISC* procesore. Ujedno je i značajno olakšano projektovanje procesora.

Na neosporne prednosti *RISC* pristupa ukazali su:

1. rezultati *IBM 801* projekta, koga je krajem sedamdesetih godina vodio *John Cocke*,
2. osobine *RISC-I* i *RISC-II* procesora, koje je na Berkli univerzitetu početkom osamdesetih godina razvila grupa na čijem čelu je bio *David A. Patterson* i
3. osobine *MIPS* procesora, koji je na Stanford univerzitetu početkom osamdesetih godina razvila grupa na čijem čelu je bio *John L. Hennessy*.

Nakon toga, svi veliki proizvođači računara su usvojili *RISC* pristup. Tako su nastali *RISC* procesori: *Motorola 88000*, *Intel i860*, *IBM RS/6000* i *DEC ALPHA*, navedeni u hronološkom redosledu pojave na tržištu.

GRANICE RASTA BRZINE I GUSTINE TRANZISTORA NA ČIPU

Skraćenje procesorskog ciklusa, koje je omogućila tehnologija veoma visoke integracije, je bilo prirodna posledica smanjenja veličine tranzistora, odnosno, povećanja njihovog broja na čipu. Smanjenje veličine tranzistora po svim dimenzijama na polovinu, uz isti napon i snagu, učetvorostručuje brzinu tranzistora, uz napomenu da pod tim uslovima 4 puta više tranzistora na čipu emituje i 4 puta veću toplotu. Veća brzina tranzistora znači manje kašnjenje signala, pri njihovom prolasku kroz tranzistor, a to dalje znači da signal ima veću brzinu prostiranja i da mu je potrebno kraće vreme da prođe kroz čip. Zato procesorski ciklus može da bude kraći. Skraćenju procesorskog ciklusa je doprinosilo i smanjenje veličine čipa, odnosno, skraćenje ukupnog puta signala, što je, takođe, bila usputna posledica *RISC* pristupa. Smanjenje veličine čipa je izazvalo i smanjenje njegove proizvodne cene, jer je, zbog tehnologije proizvodnje, na istu površinu (*wafer*) moglo da stane više čipova (*die*), pa su se na njih raspoređivali približno konstantni troškovi proizvodnje.

Ubrzanje računara, zasnovano na skraćenju procesorskog ciklusa ima svoja fizička ograničenja, jer je skraćenje procesorskog ciklusa ograničeno činjenicom da brzina prostiranja signala ne može preći brzinu svetlosti od oko 3×10^8 metara u sekundi. Znači, **procesorski ciklus mora biti toliko dugačak da u toku trajanja ciklusa signal može da pređe put od svog izvorišta do svog odredišta**. To ograničava veličinu računara, odnosno njegov **prečnik** (*diameter*). Prečnik računara je jednak polovini puta koji signali mogu da pređu u jednom procesorskom ciklusu, jer se podrazumeva, na primer, da signali odlaze od procesora ka radnoj memoriji, i da se vraćaju, od radne memorije ka procesoru. Tako je ciklus *DEC ALPHA 21164* procesora, koji se pojavio na tržištu 1995. godine, iznosio oko 3.33 nano (10^{-9}) sekunde. Za to vreme signal je mogao da pređe dužinu od oko jednog

metra, ako se kretao brzinom svetlosti, pa je polovina te dužine predstavljala prečnik računara, zasnovanog na ovom procesoru.

I povećanje brzine tranzistora i njihovog broja na čipu imaju svoja fizička ograničenja. Tako, u proteklom periodu razvoja poluprovodničkih tehnologija, brzina tranzistora i njihov broj na procesorskom čipu su rasli približno 60% godišnje. Broj tranzistora na memorijskom *DRAM* čipu se povećavao za oko 120% godišnje, uz povećanje njihove brzine za oko 3% godišnje. Ovakvu pravilnost razvoja poluprovodničkih tehnologija je sredinom šezdesetih godina uočio *Gordon Moore*, jedan od osnivača *Intel*-a. Međutim, takva razvojna linija ne može da bude stalna, bar ne u pomenutim poluprovodničkim tehnologijama. Minijaturizacija tranzistora je moguća samo do granice nakon koje elektronske komponente postaju nepouzdane zbog međusobne interakcije. Slično, povećanje brzine tranzistora je moguće samo do granice nakon koje emitovanje toplote postaje nepremostiv problem. Upravo zbog problema hlađenja, brzina tranzistrora na *DRAM* čipovima je rasla znatno sporije od njihovog broja na čipu.

DOMETI PRIMENE PARALELIZMA

Pobrojana fizička ograničenja povećanja brzine računara su postala važna, kada su njihovi dometi bili nedovoljni za rešavanje praktično značajnih problema, kao što je, na primer, problem analize radarskih i televizijskih slika sa satelita. Svaka ovakva slika se sastojala od 10^{10} piksela, a njena analiza je obuhvatala prijem i obradu pojedinih piksela slike. Obrada piksela je zahtevala obavljanje oko 10^3 operacija po pikselu. Pošto je svakog dana sa satelita pristizalo na stotine slika, za obradu svih piksela je bilo potrebno dnevno obaviti oko 10^{15} operacija, ili, približno, 10^{10} operacija u sekundi. Uz pretpostavku da je za svaku od ovih operacija dovoljan jedan procesorski ciklus, računar bi mogao da obradi sve slike, pristigle sa satelita, ako bi imao procesorski ciklus 10^{-10} sekundi, odnosno, ako njegov prečnik ne bi bio veći od 1.5 centimetra! Jasno, rešavanje problema, kao što je analiza satelitskih slika, se moralo osloniti na istovremenu, odnosno paralelnu obradu nezavisnih podataka.

Serijski prenos piksela isključuje mogućnost istovremenog prijema više piksela. Međutim, obrade pojedinih piksela su nezavisne, pa mogu biti istovremene. Prema tome, prijem piksela predstavlja **sekvencijalan deo** analize satelitske slike, koga nije moguće skratiti primenom paralelizma, dok obrada pojedinih piksela predstavlja **nesekvencijalan deo** analize satelitske slike, koga je moguće skratiti primenom paralelizma.

Sagledavanje dometa primene paralelizma u analizi radarskih i televizijskih slika sa satelita zahteva da se uporede vreme serijske analize slike sa vremenom njene paralelne analize. U toku serijske analize slike, od ukupnog vremena t njene analize, s-ti deo ($0<s<1$) otpada na prijem svih njenih piksela, a (1-s)-ti deo otpada na serijsku obradu svih piksela satelitske slike,. Znači, proizvod:

$$s \cdot t$$

odgovara vremenu prijema svih piksela satelitske slike, a proizvod:

$$(1-s) \cdot t$$

odgovara vremenu serijske obrade svih piksela slike. Uz pretpostavku da se istovremeno obavljaju obrade n piksela ($1<n \leq 10^{10}$), tada vreme paralelne obrade svih piksela satelitske slike iznosi:

$$\frac{(1-s) \cdot t}{n}$$

dok se vreme prijema ovih piksela ne menja. Prema tome, ukupno vreme paralelne analize satelitske slike iznosi:

$$s \cdot t + \frac{1-s}{n} \cdot t$$

Primena paralelizma u analizi satelitske slike dovodi do poboljšanja koje se izražava kao odnos vremena potrebnog za serijsku i vremena potrebnog za paralelnu analizu slike:

$$\frac{t}{s \cdot t + \frac{1-s}{n} \cdot t} = \frac{1}{s + \frac{1-s}{n}} = \frac{1}{\frac{s \cdot n + 1 - s}{n}} = \frac{n}{1 + s \cdot (n-1)}$$

Prethodni odnos se može primeniti na svaku obradu podataka koja ima sekvencijalan deo, na koga primena paralelizma nema uticaja, i nesekvencijalan deo, na kog primena paralelizma ima uticaja. Prema tome, veličina poboljšanja, izazvanog primenom paralelizma, zavisi od dva faktora. Jedan je udeo (s) koji u ukupnom vremenu obrade svih podataka ima njen sekvencijalan deo. Drugi je mogući stepen paralelizma (n) koji određuje najveći mogući broj istovremenih obrada pojedinih podataka. Pri tome, izraz:

$$\frac{1}{s}$$

predstavlja graničnu vrednost pomenutog poboljšanja. Stvarno poboljšanje se

približava svojoj graničnoj vrednosti samo za visok stepen paralelizma, kada parametar n ima veoma veliku vrednost.

Na prethodno opisane domete primene paralelizma ukazao je *G. M. Amdahl* još 1967. godine.

Primena paralelizma dovodi do značajnijeg skraćenja obrada podataka samo ako je njen sekvencijalan deo veoma kratak i ako je mogući stepen paralelizma visok. Upravo te osobine ima analiza satelitskih slika. Kod nje su postojali uslovi za masovni paralelizam, jer je stepen paralelizma mogao da raste do 10^{10}. Pored toga, sve istovremene obrade su

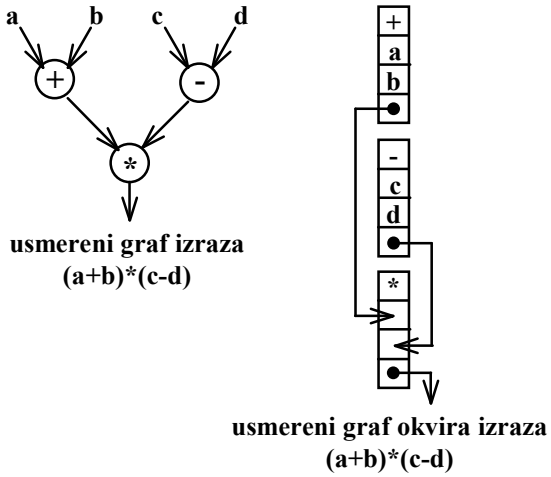
1. identične, jer su pikseli obrađivani na isti način i
2. jednostavne, jer su obrađivane jednobitne vrednosti, koje su reprezentovale pojedine piksele.

To je dozvoljavalo da se napravi paralelni računar sa više od 10000 jednostavnih jednobitnih procesora, koji su nazvani **procesni elementi**. Oni su mogli da u dovoljno kratkom vremenu obrade sve piksele slike. Uz ne suviše kratak procesorski ciklus od 10^{-6} sekundi, svaki od ovakvih procesnih elemenata mogao je da obavi 10^6 jednocikličnih operacija u sekundi, pa je 10^4 procesnih elemenata moglo da obavi 10^{10} operacija u sekundi. Međutim, mana ovakvog paralelnog računara je da je on bio specijalizovan samo za rešavanje problema analize satelitskih slika. Iz prethodnog sledi da se uspeh u primeni paralelnih računara ostvaruje na račun njihove opštosti, odnosno da zavisi od njihove specijalizovanosti za rešavanje pojedinih klasa problema.

KONKURENTNO PROGRAMIRANJE

Uspešna primena paralelnih računara je zavisila, pre svega, od njihovog programiranja. Za programiranje paralelnih računara nisu bili podesni programski jezici visokog nivoa, jer su oni predstavljali apstrakciju računara koji sekvencijalno izvršavaju programe, odnosno za koje se podrazumeva da izvršavaju naredbe jednu za drugom u redosledu određenom u programu. Redosled izvršavanja naredbi je bio važan, jer su naredbe opisivale izmene stanja promenljivih, a konačno stanje ovih promenljivih je zavisilo od redosleda izmena njihovih stanja. Ovakav **model računanja** (*computational model*) je nazvan **model upravljačkog toka** (*control flow model*, *control driven model*), jer su na redosled izvršavanja naredbi uticale upravljačke naredbe, izmenom stanja programskog brojača. Znači, nakon opisivanja sekvencijalnog rešenja problema pomoću programskog jezika visokog nivoa, programer je morao da ga prilagodi paralelnom računaru pronalaženjem nezavisnih delova programa, koji su mogli da se izvršavaju paralelno. Nakon njihovog pronalaženja, sledilo je i preuređenje programa, radi stvaranja uslova da potencijalni paralelizam postane moguć. Potreba izričitog opisivanja paralelizma je izazvala dogradnju postojećih sekvencijalnih programskih jezika visokog nivoa. Tako su nastali **konkurentni programski jezici** (*concurrent programming language*).

Korišćenje konkurentnih programskih jezika je složenije od korišćenja sekvencijalnih programskih jezika. Zato se javila potreba za pronalaženjem načina izražavanja algoritama kod kojih je paralelizam prirodna posledica pravljenja algoritma, a ne rezultat posebnog napora programera. Tako je, na primer, nastao **model toka podataka** (*data flow model*, *data driven model*). On je podrazumevao da se međuzavisnost operacija, sadržanih u obradi podataka, označi vezivanjem izlaznih operanada jedne operacije za ulazne operande druge operacije. Tako nastaje usmereni graf, u čiji čvorovima se nalaze okviri. Svaki okvir sadrži polje za kod operacije, polja za ulazne operande i polje za izlazni operand (Slika 12.1.1).



Slika 12.1.1 Princip toka podataka

Model toka podataka je uspostavio redosled samo za međusobno zavisne operacije. Sve druge operacije, čiji ulazni operandi su raspoloživi, odnosno, čiji okviri su popunjeni, mogu da se izvršavaju u bilo kom redosledu, znači i paralelno. Iako je model toka podataka nudio prirodni paralelizam, on nije posedovao opštost modela upravljačkog toka, pa nije doživeo ozbiljniju praktičnu primenu. Za ovaj model su bile potrebne posebne organizacije računara, okrenute popunjavanju okvira i selekciji popunjenih okvira, kao i posebni funkcionalni programski jezici.

Nepostojanje zgodnog načina za opisivanje paralelizma i iz toga proizašla visoka cena razvoja konkurentnih programa su bili donekle ublaženi pojavom oblika paralelizma koji su nevidljivi za programere. U ovakve oblike paralelizma je spadalo preklapanje rada stepena protočne strukture procesora (*ILP*), koje je dovodilo do smanjenja srednjeg broja ciklusa po naredbi, ubrzavajući tako izvršavanje korisničkih programa i to bez ikakve intervencije programera. Ipak, prethodno je zahtevalo asistenciju kompajlera, u čijoj nadležnosti je bilo

raspoređivanje mašinskih naredbi na način koji ne menja funkcionalnost programa, a koji omogućuje veće preklapanje rada stepena protočne strukture procesora.

Pojava paralelizma, koji je bio nevidljiv za programera, se javljala i kao posledica prisustva više od jednog procesora opšte namene u računaru (*PLP*). Ovi procesori su omogućavali:

1. ili povećanje propusnosti računara, odnosno povećanje broja izvršenih programa u jedinici vremena,
2. ili povećanje raspoloživosti računara, odnosno, sprečavali su da kvar jednog procesora onemogući rad računara.

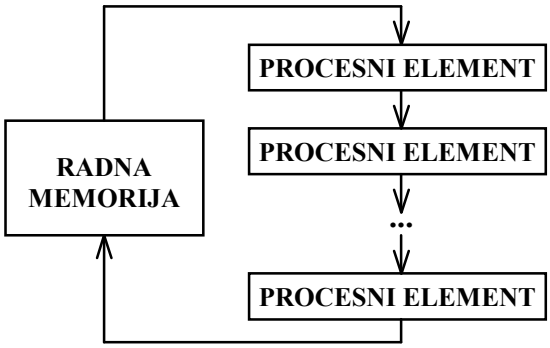
Više procesora opšte namene su ugrađivani još u računare prve generacije, radi povećavanja njihove raspoloživosti. Takva praksa je postala redovna kod najmoćnijih računara već od treće generacije računara, radi povećanja njihove propusnosti. Prisustvo više procesora u organizaciji računara nije zahtevalo bilo kakvu intervenciju programera, jer je rukovanje ovim procesorima bilo u nadležnosti operativnog sistema.

KLASIFIKACIJA RAČUNARA

Programeri su mogli svesno da iskoriste prisustvo više procesora za istovremeno izvršavanje raznih programa, posvećenih rešavanju pojedinih potproblema istog problema. Ovakva izvršavanja su obrazovala protočnu strukturu, ako su rezultati jednog izvršavanja prosleđivani drugom posredstvom datoteka. Na primer, takvu strukturu su mogli da obrazuju pretprocesor, kompajler i linker programa napisanih programskim jezikom C, ako su se istovremeno izvršavali na raznim procesorima, pripremajući za izvršavanje razne izvorne C programe. Ovakav paralelizam je karakterisala saradnja nekoliko procesora koji su izvršavali razne programe, radi obrade različitih podataka, a međusobnu saradnju su ostvarivali povremenom razmenom podataka. Ovaj paralelizam se suštinski razlikovao od masovnog paralelizma, karakterističnog po angažovanju velikog broja procesnih elemenata na identičnoj obradi različitih podataka, kao kod analize satelitskih slika. Pošto je u prvom slučaju svaki procesor izvršavao poseban program, radi obrade posebnih podataka, ovakav paralelni računar je karakterisalo **više tokova naredbi i više tokova podataka**. Engleska skraćenica za ovaj naziv je *MIMD* (*Multiple Instruction stream Multiple Data stream*). U slučaju masovnog paralelizma, svaki procesni element je izvršavao isti program, radi obrade posebnih podataka, pa je ovakav paralelni računar karakterisao **jedan tok naredbi i više tokova podataka**. Engleska skraćenica za ovaj naziv je *SIMD* (*Single Instruction stream Multiple Data stream*). *SIMD* računar predstavlja poopštenje vektorske jedinice procesora super-računara, kod kojih je više raznih podataka bilo obrađivano na isti način, ali u režimu preklapanja. Računar sa jednim procesorom je karakterisao samo **jedan tok naredbi i jedan tok podataka**. Engleska skraćenica za ovaj naziv je *SISD* (*Single Instruction stream Single Data stream*). Preostali slučaj kombinovanja posmatranih brojeva tokova naredbi i podataka je odgovarao

paralelnom računaru sa **više tokova naredbi i sa jednim tokom podataka**. Engleska skraćenica za ovaj naziv je *MISD* (*Multiple Instruction stream Single Data stream*).

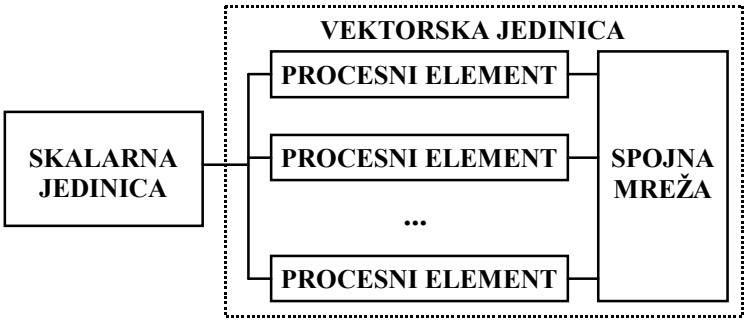
Prethodnu klasifikaciju aritektura računara je predložio *M. J. Flynn* još 1966. godine. Pri tome je *MISD* računar uveden više da upotpuni klasifikaciju, nego što je imao praktičnu vrednost. U *MISD* računare bi spadao, na primer, paralelni računar, specijalizovan za prepoznavanje oblika (*pattern recognition*), kod koga svi procesori preuzimaju iste podatke, ali svaki od njih ih obrađuje na različit način, radi provere da li podaci pripadaju određenoj vrsti, odnosno da li odgovaraju određenom obliku. Definiciji *MISD* računara donekle odgovara sistolički računar (*systolic array*), sastavljen od prostorno pravilno raspoređenih procesnih elemenata, koji su međusobno povezani, da bi kroz njih prolazili podaci u toku svoje obrade (Slika 12.1.2).



Slika 12.1.2 Sistolički računar

Sistolički računari su usko specijalizovani za rešavanje određenih problema, kao što je brza furijeova transformacija, na primer.

Jedini tok naredbi *SIMD* računara obrazuju skalarne i vektorske naredbe. One dolaze do skalarne jedinice, koja izvršava skalarne naredbe, a vektorske naredbe upućuje procesnim elementima, od kojih je sastavljena vektorska jedinica (Slika 12.1.3).

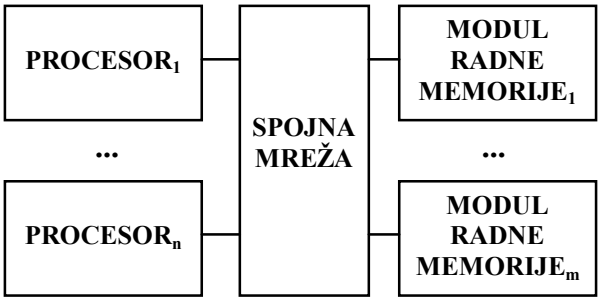


Slika 12.1.3 *SIMD* računar

Skalarna jedinica upravlja procesnim elementima iz vektorske jedinice. Svaki od procesnih elemenata u svom sastavu sadrži sopstvenu radnu memoriju, kojoj može da pristupa i skalarna jedinica. Svi aktivni procesni elementi sinhrono (istovremeno) izvršavaju istu vektorsku naredbu, obrađujući pri tome podatak iz sopstvene radne memorije. Zato nema potrebe za posebnim programskim usklađivanjem rada procesnih elemenata. Radi međusobne saradnje, procesni elementi su povezani posebnom **spojnom mrežom** (*interconnection network*). Veze koje spajaju skalarnu jedinicu sa procesnim elementima predstavljaju snopove adresnih, upravljačkih i linija podataka, posredstvom kojih skalarna jedinica pristupa lokacijama lokalnih radnih memorija procesnih elemenata, odnosno upravlja radom procesnih elemenata.

SIMD računari su specijalizovani za rešavanje pojedinih klasa problema, kao što je obrada satelitskih slika. Oni predstavljaju dodatak računarima opšte namene. Naredbe izvršavanog programa sa računara opšte namene dolaze do skalarne jedinice *SIMD* računara, a u obrnutom smeru, nakon izvršavanja programa, teku rezultati obrade. Za *SIMD* računare je izražen problem pouzdanosti, jer što je veći broj procesnih elemenata, to je manja verovatnoća da su svi oni istovremeno ispravni.

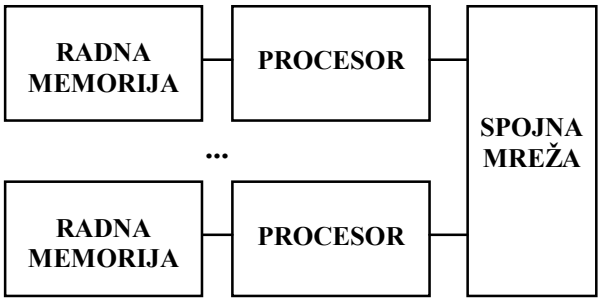
Za razliku od *SIMD* računara, kod kojeg je rad procesnih elemenata sinhron, kod *MIMD* računara procesori su nezavisni, pa nema vremenske usklađenosti njihovog rada. Zato je osnovni problem kod *MIMD* računara kako programski uskladiti, odnosno sinhronizovati rad procesora, tako da međusobno uspešno sarađuju. Pošto se saradnja procesora ostvaruje razmenom podataka, zadatak sinhronizacije je da osigura konzistentnost podataka, odnosno da spreči da procesori pristupaju podacima čiju obradu drugi procesori nisu završili. To se ostvaruje ili poštovanjem strogih pravila pri pristupanju lokacijama zajedničke radne memorije, kojoj svi procesori ravnopravno pristupaju, ili poštovanjem protokola razmene poruka po komunikacionim linijama koje povezuju procesore. U prvom slučaju, zajednička radna memorija je sastavljena od nezavisnih memorijskih modula, tako da razni procesori mogu istovremeno da pristupaju raznim modulima (Slika 12.1.4).



Slika 12.1.4 Multiprocesor

Procesori pod istim uslovima pristupaju lokacijama zajedničke radne memorije, ako su podjednako udaljeni od njenih modula. Pri tome, procesori i moduli zajedničke radne memorije moraju da budu na što kraćoj međusobnoj udaljenosti, da bi vreme pristupa lokacijama zajedničke radne memorije bilo što kraće. Zbog toga se za ovakvu organizaciju kaže da je **čvrsto spregnuta** (*tightly coupled*), a ovakvi *MIMD* računari su nazvani **multiprocesori** (*multiprocessor*). Zajednička radna memorija olakšava programiranje, jer bilo koji procesor može da izvršava bilo koji od nezavisnih programa i, pri tome, da pristupa bilo kom podatku. To oslobađa programera potrebe da poznaje sve detalje organizacije multiprocesora. Međutim, praktični razlozi sprečavaju uključivanje velikog broja procesora u multiprocesor, jer tada zajednička radna memorija i spojna mreža postaju usko grlo, čije otklanjanje ima suviše visoku cenu. Problem **proširljivosti** (*scalability*) multiprocesora se donekle ublažava uvođenjem modula zajedničke radne memorije sa različitim vremenima pristupa. Ovakvi multiprocesori se označavaju skraćenicom *NUMA* (*Non Uniform Memory Access*). Kod *NUMA* pristupa svaki od modula zajedničke radne memorije se pridružuje jednom od procesora, tako da je vreme pristupa lokacijama sopstvenih modula znatno kraće od vremena pristupa lokacijama tuđih modula. *NUMA* pristup se oslanja na pretpostavku da procesori najčešće pristupaju svom modulu zajedničke radne memorije. Mana ovoga pristupa je da programer mora da poznaje organizaciju *NUMA* multiprocesora, da bi na najbolji način rasporedio programe i podatke po modulima radne memorije. Ovo je neophodno, jer svi procesori nisu u identičnom položaju, pa ne mogu da izvršavaju sve programe i da pristupaju svim podacima pod istim uslovima.

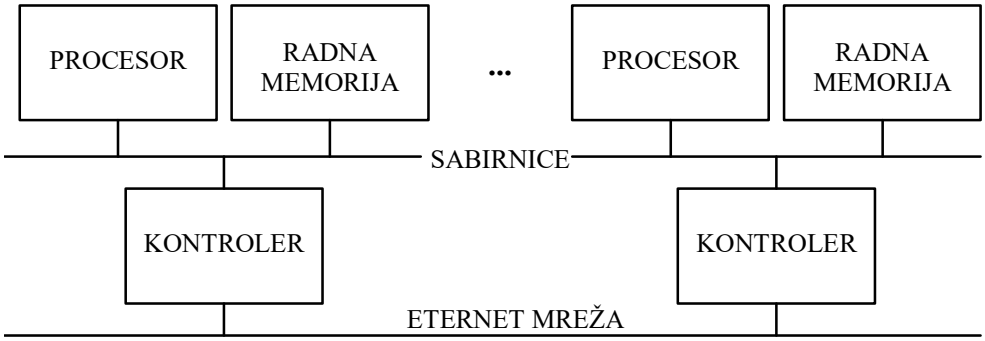
Problem proširljivosti nije izražen kod *MIMD* računara u kojima se saradnja procesora zasniva na razmeni poruka. U ovom slučaju, svaki procesor poseduje sopstvenu radnu memoriju, kojoj je samo on pristupa, a spojna mreža povezuje procesore (Slika 12.1.5).



Slika 12.1.5 Multiračunar

Sopstvena radna memorija daje procesorima veliku samostalnost. Pri tome, na mestima svakog od parova sastavljenih od procesora i radne memorije može da bude potpun računar. Zbog međusobne nezavisnosti, ovakvi računari mogu da budu na velikim međusobnim udaljenostima, pa se za ovakvu organizaciju kaže da je **labavo spregnuta** (*loosely coupled*). Ovakvi *MIMD* računari su nazvani **multiračunari** (*multicomputer*). Pošto razmena poruka po potencijalno veoma dugačkim komunikacionim linijama predstavlja jedini način saradnje računara koji ulaze u sastav multiračunara, oni su međusobno prilično izolovani. To zahteva od programera da poznaju detalje organizacije multiračunara, radi ispravne raspodele programa i podataka. Ovde je važnost ispravne raspodele veća nego kod *NUMA* multiprocesora, jer nema mogućnosti za direktno pristupanje programima ili podacima iz tuđih radnih memorija, pa je odgovornost programera veća, a njegov zadatak teži.

Računari, koji čine multiračunar, su tipično međusobno povezani eternet lokalnom mrežom (Slika 12.1.6).



Slika 12.1.6 Multiračunar organizovan oko eternet lokalne mreže

Multiračunar može sadržati više eternet mreža, spojenih pomoću posebnih uređaja (*bridge*, *gateway*, *router*). Više spojenih eternet mreža obrazuju internet.

Broj, vrstu i prostorni raspored računara, zakačenih za lokalnu mrežu, određuje i menja po svojim potrebama korisnik. Znači, lokalne mreže omogućuju korisniku da oblikuje organizaciju ovakvog multiračunara po svojoj volji.

OTPORNOST NA KVAROVE

Kvar pojedinih računara iz multiračunara ne izaziva obavezno kraj rada celog multiračunara, nego, eventualno, samo njegovo usporenje, odnosno produženje vremena odziva. Zato je multiračunar prirodna osnova za primene kod kojih je važan nastavak rada i nakon pojave pojedinačnih kvarova, odnosno za primene koje zahtevaju posedovanje **otpornosti na kvarove** (*fault tolerance*). Ovakve primene se oslanjaju na više računara, koji mogu, na primer, istovremeno da obavljaju iste poslove i da razmenjuju rezultate svojih aktivnosti, radi njihovog poređenja. Ako se pojave različiti rezultati, tada se, kao ispravan, usvaja rezultat koga je proizvela većina računara (*majority voting*). Moguće je i da samo jedan, glavni računar obavlja zadani posao, a da dodatni, rezervni (*stand by*) računar prati njegov rad. Glavni računar periodično prolazi kroz tačku oporavka, u kojoj razmenjuje poruke sa rezervnim računarom, da bi dokazao da je ispravan, i da bi ostavio podatke o stanju obrade. Na osnovu ovih podataka, u slučaju kvara glavnog računara, rezervni računar može da rekonstruiše obradu od poslednje tačke oporavka.

RISC PROCESORI

RISC procesori su u potpunosti podređeni povećanju brzine izvršavanja programa. Njihova pojava je usledila nakon što su programski jezici visokog nivoa istisnuli iz široke upotrebe asemblerske jezike, pa je tako opao značaj izražajnosti asemblerskih naredbi, a briga o racionalnom korišćenju mogućnosti arhitekture naredbi je prešla u isključivu nadležnost kompajlera.

Skupovi naredbi *RISC* procesora sadrže uglavnom samo neophodne naredbe, kao što su naredbe prenosa podataka između lokacija radne memorije i registara procesora (*load*, *store*), naredbe celobrojne aritmetike i aritmetike realnih brojeva, kao i logičke i upravljačke naredbe. Uz to, *RISC* procesori uglavnom dozvoljavaju samo neposredno, registarsko, i indeksno adresiranje. Oni uglavnom podržavaju samo celobrojne i realne tipove podataka (raznih preciznosti).

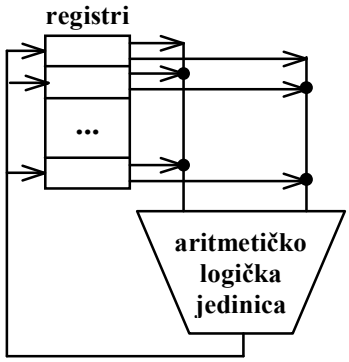
Izvedba *RISC* procesora zauzima mnogo manje prostora, nego izvedba *CISC* procesora. To se direktno koristi za:

1. skraćivanje procesorskog ciklusa,
2. povećanje broja registara opšte namene,
3. proširenje adresnog prostora,
4. povećanje paralelizma u radu procesora,
5. ugradnju skrivene memorije u procesor i
6. prebacivanje upravljanja virtuelnom memorijom u nadležnost procesora.

Veći broj registara opšte namene (32 i više) omogućuje držanje vrednosti više promenljivih u procesoru i tako, smanjujući potrebu za pristupanjem radnoj

memoriji, ubrzava izvršavanje programa. Isti efekat ima i uvođenje više skupova registara (*register window*), pri čemu svaki skup registara podržava izvršavanje različitog potprograma, čime se izbegava korišćenje steka u radnoj memoriji. Veći adresni prostor, na primer, 64 bita, kao kod *DEC ALPHA* procesora, je potreban, jer se prosečna veličina programa udvostruči svake godine, odnosno zauzme jedan bit više u adresnom prostoru.

Za *RISC* procesore je važno povećanje paralelizma u njihovom radu. To se postiže, na primer, uvođenjem više nezavisnih linija (*three-bus architecture*) koje od registara vode do aritmetičko-logičke jedinice i natrag (Slika 12.1.7).



Slika 12.1.7 Vezivanje registara i aritmetičko-logičke jedinice pomoću 3 sabirnice

Ovakva veza (*datapath*) registara i aritmetičko-logičke jedinice omogućuje da u jednom procesorskom ciklusu sadržaji dva registra stignu do aritmetičko-logičke jedinice, kao i da se rezultat njene aktivnosti smesti u treći registar.

Paralelizam u radu *RISC* procesora se povećava, na primer, tako što se prepušta stepenu njegove protočne strukture, koji je zadužen za dobavljanje naredbi, da uvećava programski brojač. To podrazumeva da je ovaj stepen osposobljen za sabiranje. Uvođenje više nezavisnih stepeni za izvršavanje operacija celobrojne aritmetike ili aritmetike realnih brojeva omogućuje paralelno obavljanje više naredbi za celobrojnu aritmetiku i za aritmetiku realnih brojeva. Procesori koji mogu da izvršavaju istovremeno više naredbi, na primer do 4, kao *IBM RS/6000* i *DEC ALPHA*, se nazivaju **super-skalari** (*superscalar*). Kod super-skalara se podrazumeva da dobavljanje naredbe, njeno dekodiranje, dobavljanje operanada i odlaganje rezultata traju kraće od izvršavanja operacije. Zahvaljujući tome, moguće je istovremeno pokrenuti više nezavisnih naredbi (*multiple instruction issue*) koje će istovremeno zauzimati različite stepene procesora, namenjene za izvršavanje raznih operacija (celobrojne aritmetike, aritmetike realnih brojeva ili prenosa podataka između lokacija radne memorije i registara).

Istovremeno izvršavanje više naredbi komplikuje reakciju na prekide. Problem se javlja zbog istovremenog izvršavanja više operacija i teškoće da se

spolja precizno ustanovi šta se u procesoru dešavalo u trenutku prekida. Prekidi su **precizni** (*precise interrupt*), ako pokretanju mehanizma prekida prethodi završetak svih započetih izvršavanja naredbi. U suprotnom su prekidi **neprecizni** (*imprecise interrupt*).

Ugradnja skrivene memorije u *RISC* procesore doprinosi povećanju brzine izvršavanja programa. U tom pogledu, posebno je važno razdvajanje skrivene memorije na deo namenjen samo za naredbe i na deo namenjen samo za podatke, jer tada mogu da se potpuno preklapaju dobavljanje naredbe i prenos podataka između lokacije radne memorije i registra procesora. Na skrivenu memoriju se može sa uspehom primeniti i ideja memorijske hijerarhije, pa se tako javljaju skrivene memorije u dva nivoa. Na višem nivou je manja skrivena memorija, sa kraćim vremenom pristupa, a na nižem nivou je veća skrivena memorija sa dužim vremenom pristupa. Gornji nivo skrivene memorije se približava svojim vremenom pristupa potrebama procesora, a donji nivo svojim kapacitetom utiče na smanjenje verovatnoće promašaja. Dva nivoa skrivene memorije su, na primer, ugrađena u *RISC* procesor *DEC ALPHA 21164*. Kod njega se na višem nivou nalaze dve skrivene memorije, svaka sa po 8 kilobajta, jedna namenjena za mašinske naredbe, a druga za podatke, dok se na nižem nivou nalazi jedna skrivena memorija sa 96 kilobajta.

Brzini izvršavanja programa doprinosi i ubrzanje pretvaranja virtuelne adrese u fizičku, koje je posledica prebacivanja upravljanja virtuelnom memorijom u nadležnost procesora.

Formati mašinskih naredbi *RISC* procesora su pravilni, znači imaju isti raspored polja i istu dužinu za sve naredbe. To omogućuje brže dekodiranje i veće preklapanje obavljanja naredbi, jer u ovakvim mašinskim naredbama nema dodatnih reči, kao kod mašinskih formata promenljive dužine, koje bi izazvale odlaganje dobavljanja naredne mašinske naredbe. Iz istih razloga, za svaki tip mašinske naredbe su, u njenom mašinskom kodu, predodređena adresiranja. Tako je indeksno adresiranje predviđeno samo za naredbe prenosa podataka (*load* i *store*), koje jedine mogu da pristupe lokacijama radne memorije, radi prenosa njihovog sadržaja u registre procesora, ili u obrnutom smeru. Aritmetičke i logičke naredbe pristupaju samo sadržajima registara procesora i, eventualno, neposrednim operandima, prisutnim u mašinskom formatu naredbe. Za upravljačke naredbe je predviđeno ili registarsko adresiranje, kod koga je odredišna adresa u nekom od registara procesora, ili relativno adresiranje, kod koga odredišna adresa nastaje kao zbir neposredne vrednosti iz mašinskog formata naredbe i sadržaja nekog od registara procesora. Ovakvo ograničeno korišćenje adresiranja doprinosi preklapanju izvršavanja naredbi, jer se tako izbegava da se podudare, na primer, iste vrste pristupanja radnoj memoriji u toku izvršavanja raznih naredbi. Nepromenljivost mašinskih formata naredbi uzrokuje da je veličina neposrednih operanada ograničena, jer je za njih na raspolaganju samo deo bita iz mašinskog formata naredbi.

Veličina ubrzanja rada *RISC* procesora zavisi od kompajlera. Njegov zadatak je ne samo da generiše najbolje nizove mašinskih naredbi, nego i da mašinske naredbe u ovim nizovima tako rasporedi, da se ostvari što dugotrajnije preklapanje njihovih izvršavanja. Tako, kompajler razdvaja međusobno zavisne naredbe, kod kojih, na primer, rezultat izvršavanja jedne naredbe predstavlja ulazni operand druge i čije preklapanje nije moguće. Međusobno zavisne naredbe se razdvajaju umetanjem između njih drugih, nezavisnih mašinskih naredbi. Ovakvo, **statičko raspoređivanje** je naročito važno za uspešan rad super-skalara. Pošto postojanje status registra sputava slobodu kompajlera kod statičkog raspoređivanja, *RISC* procesori uglavnom ne sadrže status registre, pa ni uslovne bite. Zato se rezultat (vrednost 0 ili 1) provere važenja nekog uslova (veće, manje i slično) odlaže u zadani registar opšte namene, što je u nadležnosti posebnih naredbi. Uz njih obično postoje i uslovne upravljačke naredbe, koje dovode do izmene toka izvršavanja programa, zavisno od toga da li zadani registar sadrži vrednost 0 ili vrednost 1. Alternativa je i da se naredbe provere i uslovne upravljačke naredbe spoje u jednu naredbu, čije izvršavanje zahteva dvostruko više ciklusa od preostalih naredbi. Pošto i izvršavanje upravljačkih naredbi skraćuje dužinu preklapanja, kompajler ima zadatak da izbacuje ovakve naredbe iz mašinskog programa, kad god je to moguće. Na primer, kompajleri prevode petlje (*loop*) sa unapred zadanim i nepromenljivim brojem ponavljanja njihovog tela, tako, da zadani broj puta ponove generisanje tela takvih petlji (*loop unrolling*). Na taj način upravljačke naredbe postaju suvišne, a generisani niz mašinskih naredbi osigurava dugotrajno preklapanje njihovog izvršavanja. Negativan uticaj upravljačkih naredbi na dužinu preklapanja se neutrališe i preklapanjem njihovog izvršavanja sa izvršavanjem naredbi koje ne zavise od izvršavanja pomenutih upravljačkih naredbi. Zadatak kompajlera je i da obezbedi da se najčešće korišćene vrednosti nalaze u registrima procesora, jer se tako izbegava pristupanje radnoj memoriji.

Uslovne upravljačke naredbe su posebno problematične za *RISC* procesore, jer protočni režim rada ovih procesora uzrokuje da se dobavljanje nove naredbe preklapa sa dekodiranjem uslovne upravljačke naredbe. Prema tome, da bi se izbegao zastoj (*stall*) punjenja protočne strukture, odnosno, da bi se izbeglo skraćenje dužine preklapanja, pre izvršavanja uslovne upravljačke naredbe mora se odlučiti da li se dobavlja naredba iza nje ili njena ciljna naredba. Takva odluka se zasniva na **dinamičkom predviđanju** ishoda izvršavanja uslovnih upravljačkih naredbi (*dynamic branch prediction*). Loše predviđanje dovodi do izvršavanja pogrešnih naredbi. Ono se prekida čim bude poznat ishod izvršavanja uslovne upravljačke naredbe, ali tada sleduje poništavanje efekata pogrešno izvršenih naredbi. Da bi to bilo moguće, neophodno je koristiti ili privremene lokacije za smeštanje rezultata izvršavanja pogrešnih naredbi, ili sačuvati vrednosti koje su ovi rezultati izmenili.

Dinamičko predviđanje ishoda izvršavanja uslovnih upravljačkih naredbi se zasniva na pamćenju ishoda njihovih prethodnih izvršavanja i na pretpostavci da će

ponovljena izvršavanja imati isti ishod. Kompajler može da doprinese tačnosti dinamičkog predviđanja, ako postoji način da ostavi podatak o sigurnim ishodima izvršavanja uslovnih upravljačkih naredbi.

Povećanju dužine preklapanja naredbi može da doprinese i **izvršavanje naredbi van redosleda** (*out of order*). Na primer, kada se dobavi zavisna naredba, za čije izvršavanje je potreban rezultat izvršavanja prethodne naredbe, umesto zastoja u punjenju protočne strukture, može se nastaviti sa dobavljanjem sledeće naredbe. Na taj način, odlaganje izvršavanja zavisne naredbe ne izaziva zastoj u punjenju protočne strukture, dok iza nje ima nezavisnih naredbi. Radi otkrivanja zavisnosti naredbi, u toku njihovog obavljanja procesor vodi evidenciju (*scoreborading*) o korišćenju svojih resursa, kao što su registri. Za registre ovakva evidencija obuhvata podatak da li obavljane naredbe čitaju ili pišu neki od registara. Kada se dobavi nova naredba koja čita iz nekog registra, a za taj registar je već evidentirano da jedna od obavljanih naredbi piše u njega, tada je jasno da nova naredba zavisi od jedne od izvršavanih naredbi. Međuzavisnost naredbi se može otkloniti, kada one ne prosleđuju jedna drugoj rezultate svog rada, nego samo koriste isti registar. Tako, ako obavljana naredba čita iz nekog registra, a nova naredba piše u isti registar, tada je međuzavisnost ovih naredbi moguće otkloniti korišćenjem rezervnog registra procesora za drugu naredbu (*register renaming*, *Tomasulo algorithm*).

Uz *RISC* procesore su neophodne velike radne memorije, jer su njihovi mašinski programi znatno duži od mašinskih programa *CISC* procesora.

Za *RISC* procesore je karakteristično da podržavaju memorijski preslikani ulaz-izlaz, kao i da poštuju *IEEE* 754 standard za aritmetiku realnih brojeva.

12.2. ISHODI EVOLUCIJE ARHITEKTURE RAČUNARA U PERIODU OKO 1990. GODINE

IEEE 754 STANDARD ZA ARITMETIKU REALNIH BROJEVA

IEEE 754 standard definiše 3 formata mašinske normalizovane forme realnih brojeva:

1. jednostruka preciznost od 32 bita sa najznačajnijim bitom za predznak, narednih 8 bita za podešeni eksponent (*excess 127 exponent*) i preostalih 23 za frakciju,
2. dvostruka preciznost od 64 bita sa najznačajnijim bitom za predznak, narednih 11 bita za podešeni eksponent (*excess 1023 exponent*) i preostalih 52 za frakciju,
3. proširena preciznost od 80 bita koja je namenjena za interno korišćenje u toku računanja, radi smanjivanja greške, izazvane odbacivanjem prekobrojnih cifara.

Ovaj standard definiše i **denormalizovane** (*denormalized*) **brojeve**. Oni se koriste za predstavljanje manjih vrednosti od onih koje se mogu predstaviti

mašinskom normalizovanom formom. Na taj način se problem potkoračenja (*underflow*) ublažava. IEEE 754 standard uvodi i oznake za beskonačno (*infinity*) i neodređeno (*not a number*), koje nastaju kao rezultat deljenja konačnog broja nulom ili deljenja beskonačnog sa beskonačnim. Znači, ove oznake mogu da se koriste u aritmetičkim operacijama realnih brojeva.

ARHITEKTURA NAREDBI ZA *MIPS*

RISC procesori, kao što su *MIPS*, *INTEL 860*, *MOTOROLA 88000* i *SPARC*, imaju sličnu arhitekturu. Arhitektura naredbi za *MIPS* podržava adresni prostor od 32 bita, koji je organizovan u bajte. Ona omogućuje rukovanje binarnim vrednostima velikim 1, 2 i 4 bajta, kao i realnim vrednostima od 4 i 8 bajta (*floating-point*). U ovoj arhitekturi je na raspolaganju 16 registara od 32 bita ili 16 registara od 64 bita za realne brojeve. Ova arhitektura podržava virtuelnu memoriju. Ona podržava neposredno, registarsko i indeksno adresiranje.

Mašinski formati naredbi ove arhitekture imaju fiksnu veličinu.

6 bita	5 bita	5 bita	5 bita	5 bita	6 bita
kod naredbe	ulazni reg1	ulazni reg2	izlazni reg	konstanta	kod naredbe

Prethodni mašinski format naredbi odgovara aritmetičkim i logičkim naredbama, čiji ulazni operandi su u registrima označenim kao ulazni reg1 i ulazni reg2, a izlazni operand se smešta u registar označen kao izlazni reg. Ovaj mašinski format odgovara i naredbama pomeranja koje koriste polje konstanta.

6 bita	5 bita	5 bita	16 bita
kod naredbe	ulazni reg	izlazni reg	konstanta

Prethodni mašinski format naredbi odgovara naredbama prenosa podataka.

6 bita	5 bita	5 bita	16 bita
kod naredbe	ulazni reg1	ulazni reg2	konstanta

Prethodni mašinski format odgovara upravljačkim naredbama. Za uslovne upravljačke naredbe se podrazumeva: (1) da njihov uslov zavisi od rezultata poređenja sadržaja dva registra ili (2) da je taj uslov određen sadržajem registra, u kome se nalazi kod logičke vrednosti (0 ili 1).

6 bita	26 bita
kod naredbe	konstanta

Prethodni mašinski format odgovara bezuslovnim upravljačkim naredbama.

Arhitektura naredbi za *MIPS* obuhvata sledeće tipove naredbi:

1. naredbe za rukovanje podacima (za prenos podataka),
2. naredbe za rukovanje bitima (logičke naredbe i naredbe pomeranja),
3. naredbe za celobrojnu binarnu aritmetiku (obuhvaćene sve aritmetičke operacije),
4. naredbe za aritmetiku realnih brojeva (obuhvaćene sve aritmetičke operacije),
5. upravljačke naredbe i
6. sistemske naredbe.

INTEL PENTIUM PRO PROCESOR

Arhitektura naredbi za *INTEL PENTIUM Pro* je proširena, u donosu na arhitekturu naredbi za *INTEL 80386*, naredbama za aritmetiku realnih brojeva, naredbama za rukovanje potrošnjom energije, naredbama za multimedijalne primene (*MMX*), kao i manjim brojem drugih naredbi različite namene. Naredbe za rukovanje potrošnjom energije su namenjene, pre svega, računarima sa baterijskim napajanjem. Naredbe za multimedijalne primene olakšavaju obradu zvuka i slike, jer podržavaju *SIMD* model obrade podataka.

INTEL PENTUIM Pro procesor je super-skalar koji može da izvrši do 3 naredbe u jednom procesorskom ciklusu (*three-way superscalar*). Unutar ovoga procesora se nalazi *RISC* jezgro sa 12-stepenom protočnom strukturom, koja dozvoljava izvršavanje naredbi van redosleda. Naredbe *RISC* jezgra predstavljaju mikro-naredbe za naredbe *INTEL PENTIUM Pro* procesora. Ovaj procesor se oslanja na dva nivoa skrivene memorije. Prvi nivo sačinjavaju skrivena memorija za podatke (*L1 data cache*) i skrivena memorija za naredbe (*L1 instruction cache*), svaka od po 8 kilobajta. Drugi nivo (*L2 cache*) se sastoji od 96 kilobajtne statičke poluprovodničke memorije, spojene sa procesorom posebnom 64-bitnom sabirnicom.

EVOLUCIJA MASOVNE MEMORIJE

Tehnologija magnetnih diskova se uspostavila kao glavni oslonac masovne memorije, jer je uspešno pratila porast zahteva, zahvaljujući svom stalnom usavršavanju. Tako, kapacitet diskova prosečno godišnje raste za približno 60%, a njihovo srednje vreme pristupa se smanjuje za oko 3% godišnje. Rast kapaciteta se zasniva na povećanju gustine magnetnog zapisa. Zahvaljujući tome, dimenzije magnetnih diskova su se smanjivale, kao i njihovo srednje vreme pristupa, jer su se smanjivale kako udaljenosti susednih staza, tako i njihova dužina. Zato su glave diskova za kraće vreme prelazile sa staze na stazu, kao što je i svaka staza prolazila za kraće vreme ispod njih.

Racionalno korišćenje spoljnih staza diska, čija dužina je veća od dužine unutrašnjih staza, je dovelo do povećanja broja sektora na spoljašnjim stazama.

Srednje vreme pristupa diska se može smanjiti na svoj k-ti deo, ako se koristi k diskova u nizu (*disk array*), tako da se istovremeno pristupa blokovima na raznim

diskovima. Ovaj pristup se zasniva na ideji koja je primenjena kod prepletene radne memorije. On je poznat pod skraćenicom *RAID* (*redundant array of independent disks*). Za *RAID* je standardizovano 6 različitih organizacija, označenih kao *RAID* 0, *RAID* 1, *RAID* 2, *RAID* 3, *RAID* 4 i *RAID* 5. Prva podržava smanjenje srednjeg vremena pristupa, jer omogućuje raspodelu susednih blokova na 4 diska. Znači, u proseku je moguće pristupiti nizu od 4 susedna bloka za vreme pristupa jednom bloku. Međutim, mana prve organizacije je nedovoljna pouzdanost, jer je ona upotrebljiva samo ako su sva 4 diska ispravna. Ovde je važno uočiti da je verovatnoća da sva 4 diska budu ispravna manja od verovatnoće da bilo koji od njih bude pojedinačno ispravan. Druga i ostale organizacije povećavaju pouzdanost, tako što uvode **redundantne** (dodatne, rezervne) diskove. Na primer, druga organizacija koristi minimalno 2 diska, 1 kao osnovni i 1 kao rezervni, tako da rezervni disk predstavlja kopiju osnovnog diska. Preostale organizacije povećavaju pouzdanost korišćenjem kodova za otkrivanje i oporavak od grešaka.

Tehnologija optičkih diskova nudi veći kapacitet od tehnologije magnetnih disketa, pa su zato optički diskovi preuzeli od magnetnih disketa arhivsku ulogu. Primer optičkog diska je *CD* (*compact disc*). *CD* koristi lasersku tehnologiju da izmeni optička svojstva podloge na koju se u obliku spirale upisuju podaci. Postoji više standarda koji opisuju način zapisivanja podataka na disku. Prvobitni standard je opisivao samo diskove koji na sebi sadrže audio materijal (*Red Book*, *IEC 908*). Kasnije nadogradnje standarda (*Blue Book*, *Yellow Book*, itd.) su donele mogućnosti zapisa podataka (*CD-ROM*) , video materijala (*Video CD*) i fotografija (*Photo CD*) i mogućnost dopisivanja podataka (*multisession, CD-XA*). Radi univerzalne upotrebe optičkih diskova, za organizaciju podataka, odnosno za sistem datoteka optičkog diska postoji međunarodni standard *ISO 9660*. Optički disk sa podacima zapisanim po ovom standardu može da se koristi na svim računarima koji podržavaju ovaj standard. Postoje 3 vrste *CD* optičkih diskova, označene kao *CD* (*read only*), *CD-R* (*recordables*) i *CD-RW* (*rewritables*). Prva vrsta se proizvodi pomoću posebnih matrica i ovakve optičke diskove računari mogu samo čitati. Druga vrsta *CD* optičkih diskova koristi tehnologiju koja omogućuje računarima da upisuju podatke na njih. Međutim, jednom upisani podaci na ovakav optički disk ne mogu biti kasnije fizički izbrisani. Prilikom dopisivanja podataka, pravi se novi sadržaj optičkog diska, u kome mogu biti izostavljene neke prethodno upisane datoteke, što stvara iluziju da se datoteke mogu izbrisati. Treća vrsta *CD* optičkih diskova koristi tehnologiju koja dozvoljava višestruko pisanje i brisanje podataka.

Optički diskovi sa oznakom *DVD* (*digital versatile disk*) koriste noviju tehnologiju, pa imaju višestruko veći kapacitet od *CD* optičkih diskova (4.7 gigabajta i više prema 700 megabajta). Postoje dva standarda za *DVD* optičke diskove na koje se može upisivati, *DVD+R* i *DVD-R*. Većina novijih uređaja podržava oba standarda.

Tehnologija poluprovodničkih diskova (*SSD*, *Solid State Disk*), zasnovanih na *FLASH* memorijskim čipovima, nudi znatno manju potrošnju energije od prethodne dve tehnologije i znatno kraće vreme pristupa, ali uz višu cenu.

UTICAJ *RISC* PROCESORA NA VIRTUELNU MEMORIJU

Adresni prostor od 64 bita, koga uvode pojedini *RISC* procesori, izaziva pojavu suviše velikih tabela stranica, tako da one prevazilaze ukupnu memoriju računara. Na primer, za stranice velike 2^{12} (4 kilobajta), tabela stranica ima 2^{52} elemenata, odnosno zauzima 2^{55} bajta, ako njeni elementi zauzimaju po 8 bajta. Zato se, umesto tabele stranica, koristi **invertovana tabela stranica** (*inverted page table*). Broj njenih elemenata je jednak broju fizičkih stranica radne memorije. Znači, svakoj fizičkoj stranici odgovara jedan element ove tabele, tako da indeks elementa adresira odgovarajuću fizičku stranicu.

Na invertovanu tabelu stranica se primenjuje princip rada asocijativne memorije sa jednostrukom asocijativnošću. Znači, deo bita iz zadane adrese virtuelne stranice služi kao indeks elementa invertovane tabele stranica i, ujedno, kao adresa odgovarajuće fizičke stranice. Ako indeksirani element sadrži, kao referentnu adresu, preostale bite iz zadane adrese virtuelne stranice, tada se njena kopija nalazi u odgovarajućoj fizičkoj stranici.

Svaki element invertovane tabele stranica sadrži i bit popunjenosti dotičnog elementa uz bite (referentne) adrese virtuelne stranice, čija kopija se nalazi u odgovarajućoj fizičkoj stranici (Slika 12.2.1).

000		
001		
010		
011		
100		
101		
110		
111		

Slika 12.2.1 Organizacija invertovane tabele stranica

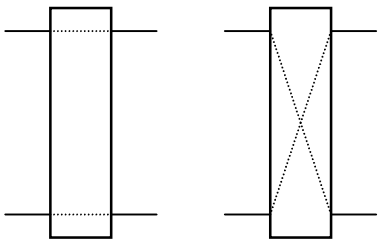
Za indeksiranje elemenata invertovane tabele stranica koju sadrži Slika 12.2.1 mogu poslužiti 3 najmanje značajna bita adrese virtuelne stranice. Preostali biti predstavljaju referentni deo njene adrese. Ako indeksirani element invertovane tabele stranica ne sadrži referentni deo adrese virtuelne stranice, tada njoj ne korespondira fizička stranica. U tom slučaju neophodno je: (1) osloboditi fizičku stranicu koja odgovara indeksiranom elementu, (2) u nju smestiti kopiju dotične virtuelne stranice i (3) referentni deo njene adrese smestiti u indeksirani element.

Organizacija invertovane tabele stranica iz prethodnog primera (Slika 12.2.1) podrazumeva da svakoj adresi virtuelne stranice odgovara samo jedan element invertovane tabele stranica, kao i da sve adrese virtuelnih stranica sa ista 3 najmanje značajna bita koriste isti element invertovane tabele stranica.

SPOJNE MREŽE

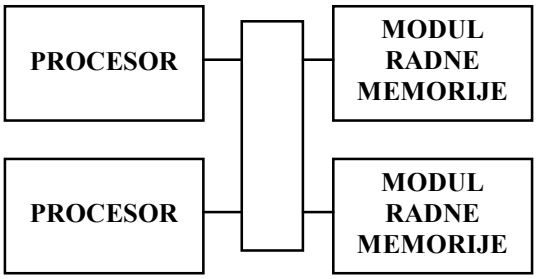
Spojne mreže omogućuju povezivanje raznih organizacionih komponenti računara. One omogućuju serijski prenos podataka koji su organizovani u poruke. Spojne mreže imaju pravilan oblik i decentralizovano upravljanje. Njihov pravilan oblik omogućuje automatsko određivanje puta podataka kroz spojnu mrežu. Upravljanje spojnim mrežama je decentralizovano, jer se odluke o izboru puta donose lokalno u okviru spojne mreže, a na osnovu adresnog dela poruke koja se serijski prenosi. Osobine spojnih mreža zavise od komponenti koje one povezuju.

Za spojne mreže, koje povezuju procesore i module radne memorije, bitno je da ponude što veći broj nezavisnih puteva između procesora i modula radne memorije, da bi što više procesora moglo istovremeno da pristupa različitim modulima radne memorije. Pri tome je važno da ovakve spojne mreže uzrokuju što manje kašnjenje. Ovakve spojne mreže se zasnivaju na 2×2 prekidačima, koji omogućuju paralelnu i unakrsnu vezu između svoja 2 ulaza i svoja 2 izlaza (Slika 12.2.2).



Slika 12.2.2 Stanja prekidača sa 2 ulaza i 2 izlaza

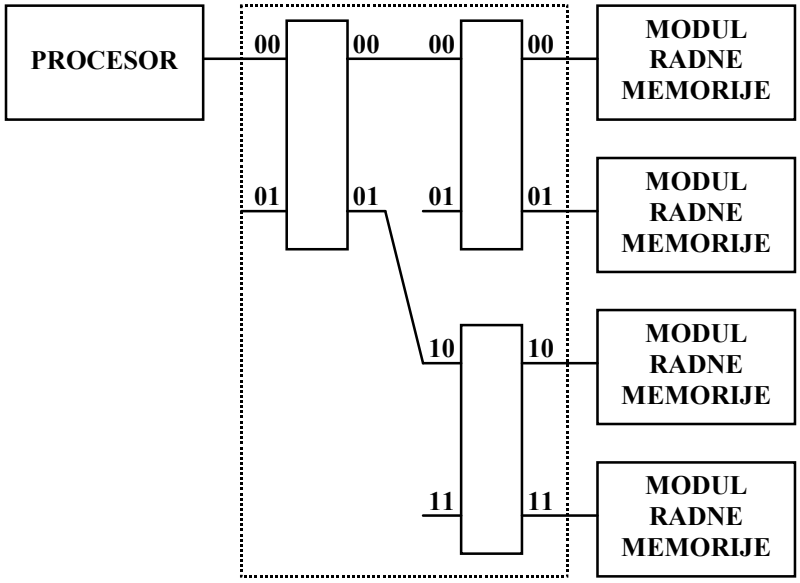
Pomoću ovakvih prekidača, moguće je ostvariti istovremenu vezu između 2 procesora i 2 modula radne memorije (Slika 12.2.3).



Slika 12.2.3 Primer upotrebe prekidača sa 2 ulaza i 2 izlaza

Za povezivanje više od 2 procesora sa više od 2 modula radne memorije, potrebno je više 2×2 prekidača, raspoređenih u više stepeni. Zato se ovakva spojna mreža naziva **višestepena prekidačka mreža** (*multistage switching network*).

Pravila raspoređivanja prekidača u višestepenoj prekidačkoj mreži, kao i pravila njihovog međusobnog povezivanja se dobro uočavaju na primeru povezivanja jednog procesora sa 4 modula radne memorije (Slika 12.2.4).



Slika 12.2.4 Princip rada dvostepene prekidačke mreže

Da bi se ostvarila ovakva veza, jedini procesor se veže za ulaz jednog prekidača. Pri tome se izlazi ovog prekidača vezuju za ulaze druga dva prekidača, čiji izlazi su vezani za po jedan modul radne memorije. Prvi od pomenuta 3

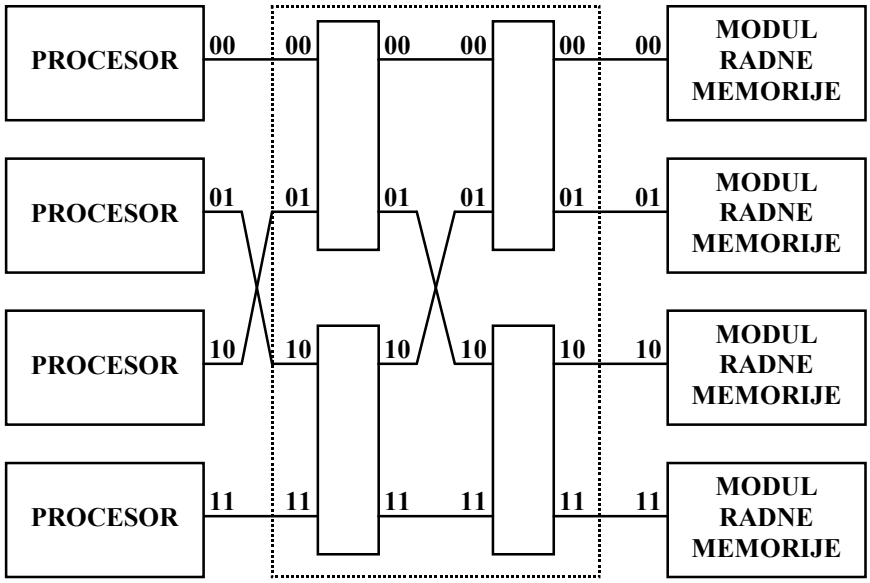
prekidača pripada prvom stepenu višestepene prekidačke mreže, a preostala 2 prekidača pripadaju njenom drugom stepenu.

Ulazi i izlazi prekidača (Slika 12.2.4) su označeni binarnim brojevima, da bi se istaklo pravilo povezivanja prekidača iz raznih stepeni višestepene prekidačke mreže. Po ovom pravilu, izlaz prekidača iz jednog stepena se povezuje sa ulazom prekidača iz sledećeg stepena, tako da binarna oznaka ulaza nastaje kružnim pomeranjem za jedno mesto u levo bita iz binarne oznake izlaza. Ovakav način povezivanja prekidača iz raznih stepeni se naziva **izmešani spoj** (*shuffle exchange*).

Podrazumeva se da je binarna oznaka svakog modula radne memorije (Slika 12.2.4) jednaka binarnoj oznaci izlaza prekidača iz drugog stepena, na koji je dati modul povezan. Ovakva binarna oznaka ulazi u sastav poruke koja odgovara transakciji čitanja ili pisanja. Ovu poruku upućuje procesor ka odgovarajućem modulu radne memorije. Izmešani spoj omogućuje svakom prekidaču da odredi svoje stanje na osnovu bita binarne oznake modula radne memorije, sadržane u poruci koja je stigla do njega. Na taj način svaki prekidač daje svoj doprinos povezivanju procesora i traženog modula radne memorije. Pri tome, prvi bit ovakve binarne oznake određuje stanje prekidača iz prvog stepena, a drugi bit određuje stanje prekidača iz drugog stepena. Na isti način se tretiraju i ostali biti, ako ima više od 2 stepena. Podrazumeva se da svaki od prekidača usmerava poruku na gornji izlaz, ako njemu namenjeni bit sadrži vrednost 0. Inače, poruka odlazi na donji izlaz prekidača. Stanje prekidača određuje poruka koja prva stigne do jednog od njegova 2 ulaza. Znači, poruka koja odgovara transakciji, usmerenoj ka modulu radne memorije sa binarnom oznakom 11, prolazi kroz donji izlaz jedinog prekidača iz prvog stepena i kroz donji izlaz donjeg prekidača iz drugog stepena. Na taj način se uspostavlja veza između procesora i pomenutog modula radne memorije. Ova veza opstaje dok se zahtevana transakcija ne obavi.

Opisani način upravljanja prekidačima omogućuje svakom prekidaču da samostalno određuje svoje stanje, što je primer decentralizovanog upravljanja.

Za vezivanje 4 procesora i 4 modula radne memorije dovoljna su 4 prekidača, raspoređena u 2 stepena (Slika 12.2.5).



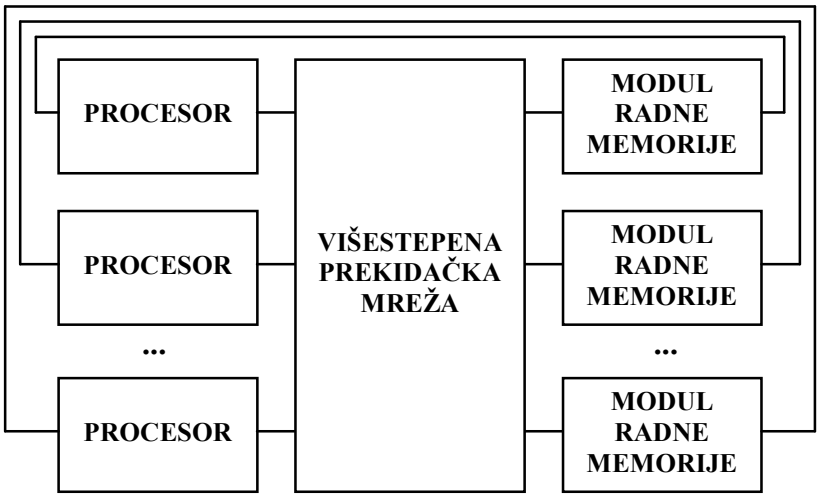
Slika 12.2.5 Primer upotrebe dvostepene prekidačke mreže

Spojna mreža koju sadrži Slika 12.2.5 se naziva **omega višestepena prekidačka mreža**. Ona omogućuje, na primer, istovremene veze procesora i modula radne memorije sa istim binarnim oznakama. Ove binarne oznake su navedene uz procesore i uz module radne memorije. Omega višestepena prekidačka mreža spada u **blokirajuće višestepene prekidačke mreže**, jer ne omogućuje istovremenu vezu bilo kog procesora i bilo kog modula radne memorije. Na primer, dok je uspostavljena veza između procesora 00 i modula radne memorije 01, dotle je blokirana, odnosno onemogućena veza između procesora 10 i modula radne memorije 00. Postoje i **neblokirajuće višestepene prekidačke mreže**, poput **Beneš višestepenih prekidačkih mreža**. Takođe, postoje i višestepene prekidačke mreže sa različitim brojem ulaza i izlaza, poput ***Banian* višestepenih prekidačkih mreža**.

Prednost višestepenih prekidačkih mreža u odnosu na unakrsne matrice je da uz znatno manje prekidača ostvaruju približno istu funkciju. U prethodnom primeru omega višestepene prekidačke mreže potrebna su samo 4 prekidača za povezivanje 4 procesora sa 4 modula radne memorije, za razliku od 16 prekidačkih blokova, koje za istu funkciju koristi unakrsna matrica. Ipak, proširljivost multiprocesora, zasnovanih na višestepenim prekidačkim mrežama, je ograničena brojem prekidača, jer je kašnjenje višestepene prekidačke mreže proporcionalno broju prekidača.

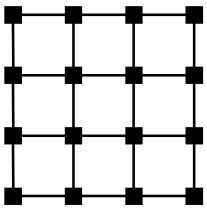
BBN Butterfly, koga je na tržište sredinom osamdesetih godina izbacio *BBN LAB.*, predstavlja primer multiprocesora, zasnovanog na višestepenoj prekidačkoj mreži. Njegova posebnost je da posebna sabirnica spaja svaki procesor sa jednim od modula radne memorije, zbog čega *BBN Butterfly* spada u *NUMA* multiprocesore.

Kod njega je vreme pristupa lokalnom modulu radne memorije, posredstvom posebne sabirnice, 2 mikrosekunde, dok je vreme pristupa preostalim modulima radne memorije, posredstvom višestepene prekidačke mreže, 6 mikrosekundi. Osnovni gradivni modul *BBN Butterfly* sadrži 16 procesora *Motorola 68000* i 16 modula radne memorije, od kojih svaki ima 1 megabajt. *BBN Butterfly* se može proširivati do 96 procesora i 96 modula radne memorije (Slika 12.2.6).



Slika 12.2.6 Organizacija *BBN Butterfly* multiprocesora

Višestepene prekidačke mreže spadaju u **dinamičke spojne mreže**, jer uspostavljaju i raskidaju spojne puteve između procesora i modula radne memorije. **Statičke spojne mreže** uspostavljaju trajnu (nepromenljivu) vezu između svojih čvorova. One su podesne za povezivanje procesnih elemenata iz vektorske jedinice kod *SIMD* računara, prilagođenih rešavanju određene klase problema, kod koje je unapred poznat oblik saradnje procesnih elemenata. To dozvoljava da se direktno povežu procesni elementi čija saradnja je neophodna. **Dvodimenzionalna mreža** (*mesh*, *grid*) predstavlja tipičan primer statičke spojne mreže (Slika 12.2.7).

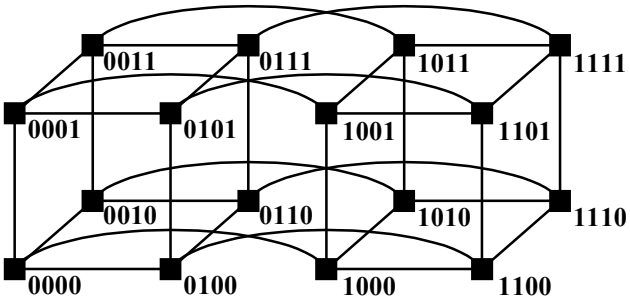


Slika 12.2.7 Primer dvodimenzionalne mreže

Zacrnjeni kvadrati iz čvorova dvodimenzionalne mreže (Slika 12.2.7) predstavljaju komunikacione procesore. Za njih vezani procesni elementi nisu prikazani. Na taj način svaki procesni element je vezan direktno za svoje horizontalne i vertikalne susede, pa samo njima može direktno da šalje poruke, odnosno samo od njih može direktno da prima poruke. Rubni procesni elementi mogu biti povezani na razne načine. Kod spiralnog načina povezivanja, procesni element na desnom kraju svake horizontale je vezan za procesni element na levom kraju naredne horizontale, a kod cilindričnog načina povezivanja, međusobno su spojeni rubni procesni elementi sa istih horizontala, odnosno vertikala.

Dvodimenzionalne mreže su korišćene za povezivanje procesnih elemenata iz vektorskih jedinica dva namenska *SIMD* računara, oba proizvedena za potrebe *NASA* (*National Aeronautics and Space Administration*). Prvi, *ILLIAC IV* (*ILLInois Automatic Computer*), je bio namenjen za matrične operacije i za rešavanje parcijalnih diferencijalnih jednačina. On je bio proizveden krajem šezdesetih godina, imao je 8×8 procesnih elemenata, namenjenih za aritmetiku realnih brojeva u 64 bita. U okviru svakog od ovih procesnih elemenata bila je lokalna radna memorija od 16 kilobajta. Drugi, *MPP* (*Massively Parallel Processor*), je bio namenjen za analizu satelitskih slika. On je bio proizveden krajem sedamdesetih godina, imao je 128×128 procesnih elemenata, namenjenih za jednobitne operacije. U okviru svakog od ovih procesnih elemenata bila je lokalna radna memorija od 1024 bita.

Statičke spojne mreže mogu uspešno da podrže i razmenu poruka između bilo koja 2 čvora i bez potpunog međusobnog povezivanja svih čvorova, pod uslovom da se na jednostavan i uniforman način pronalaze najkraći putevi između bilo kog para čvorova. To omogućuju komunikacioni procesori iz čvorova statičke spojne mreže, koji podržavaju razmenu poruka. Ako se ovakvi čvorovi označe binarnim brojevima, i ako su direktno povezani samo čvorovi čije oznake se razlikuju u samo jednoj binarnoj cifri, tada svaki komunikacioni procesor može da odredi najkraći put za bilo koju poruku samo na osnovu binarnih oznaka izvorišnog i odredišnog čvora iz poruke koju prenosi. Na najkraći put ukazuju različiti parovi korespondentnih binarnih cifara u oznakama izvorišnog i odredišnog čvora. Ako ima n>1 ovakvih parova, tada se između izvorišnog i odredišnog čvora nalazi n-1 čvorova, čije posredovanje je neophodno za uspešnu razmenu poruka. Pri tome, binarna oznaka prvog posrednika se razlikuje u samo jednom bitu od binarne oznake izvorišta, i sve tako do poslednjeg posrednika, čija binarna oznaka se razlikuje u jednom bitu od binarne oznake odredišta. Prema tome, izmenom jednog i to proizvoljno odabranog bita binarne oznake izvorišta, po kome se ona razlikuje od binarne oznake odredišta, nastaje ili binarna oznaka posrednika ili binarna oznaka odredišta. Ovakve statičke spojne mreže sa, na primer, 8 čvorova obrazuju (trodimenzionalnu) kocku, a sa 16 čvorova obrazuju četvorodimenzionalnu **hiperkocku** (*hypercube*). Slika 12.2.8 sadrži prikaz četvorodimenzionalne hiperkocke.



Slika 12.2.8 Primer četvorodimenzionalne hiperkocke

Crni kvadrati u čvorovima četvorodimenzionalne hiperkocke (Slika 12.2.8) predstavljaju komunikacione procesore.

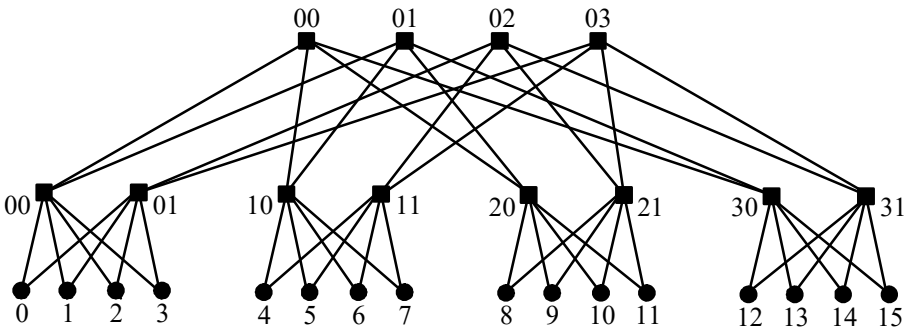
Dimenzije hiperkocke određuju najveću moguću od minimalnih udaljenosti bilo koja 2 čvora. Ovu udaljenost obrazuju ivice hiperkocke. Za četvorodimenzionalnu hiperkocku ta udaljenost je 4. Tako, na primer, za čvorove sa binarnim oznakama 0000 i 1111, rezultat logičke operacije isključivo ili je 1111. Prema tome, izmena bilo kog bita binarne oznake izvorišnog čvora daje binarnu oznaku čvora posrednika, koji je na putu ka odredišnom čvoru. Ponavljanje ovoga postupka za binarne oznake posredničkih i odredišnog čvora omogućuje određivanje binarnih oznaka svih posredničkih čvorova koji su na najkraćem putu do odredišnog čvora.

Statička spojna mreža u obliku hiperkocke se koristi i u *SIMD* i u *MIMD* računarima. *CM-1* (*Connection Machine*) predstavlja primer *SIMD* paralelnog računara, koga je na tržište početkom osamdesetih godina izbacila *Thinking Machines Corporation*. On je namenjen za simboličke obrade, karakteristične za probleme veštačke inteligencije, kao što su uparivanja (*pattern matching*), ili pretraživanja (*search*), a koje se oslanjaju na kompleksne pokazivačke strukture i međusobnu komunikaciju slučajno odabranih procesnih elemenata. Za povezivanje procesnih elemenata, osnovni gradivni modul *CM-1* sadrži dvanaestdimenzionalnu hiperkocku sa 4096 čvorova, međusobno povezanih sa 24576 dvosmernih linija, koje obrazuju ivice hiperkocke. U svakom od ovih čvorova se nalazi jedan komunikacioni procesor i 16 jednobitnih procesnih elemenata, svaki sa po 4096 bita lokalne radne memorije. Ovi procesni elementi su povezani u dvodimenzionalnu mrežu. Komunikacioni procesori posreduju u razmeni poruka između procesnih elemenata iz raznih čvorova. Prema tome, osnovni gradivni modul *CM-1* sadrži 65536 jednobitnih procesnih elemenata i 268 megabita radne memorije.

Intel hypercube ili *iPSC* (*Intel Personal SuperComputer*), koji se pojavio sredinom osamdesetih godina, predstavlja primer multiračunara, zasnovanog na hiperkocki. Osnovni gradivni modul *iPSC* sadrži petodimenzionalnu hiperkocku od 32 čvora, a moguće je kombinovanje do 4 ovakva gradivna modula. U svakom

čvoru se nalazi *Intel 80286* procesor sa koprocesorom za aritmetiku realnih brojeva i sa 512 kilobajta lokalne radne memorije.

Debelo drvo (*fat-tree*) predstavlja drugi primer statičke spojne mreže koja podržava razmenu poruka između bilo koja 2 čvora (Slika 12.2.9).



Slika 12.2.9 Primer debelog drveta za 16 čvorova

Crni kvadrati (Slika 12.2.9) predstavljaju **prekidače** (*switch*), a crni krugovi predstavljaju procesore koji su na njih povezani. Podrazumeva se da je propusnost linija iznad prekidača dvostruko veća od propusnosti linija ispod prekidača.

Debelo drvo je korišćeno u *CM-5* (*Connection Machine*) multiprocesoru koga je na tržište sredinom devedesetih godina izbacila *Thinking Machines Corporation*. I ovaj multiprocesor je bio napravljen od tržišno raspoloživih mikroprocesora.

BARIJERNA SINHRONIZACIJA

Primena multiprocesora u složenim iteracionim numeričkim proračunima je dovela do potrebe za posebnom vrstom usklađivanja aktivnosti više procesa koji su pridruženi različitim procesorima. U ovakvim iteracionim proračunima svaki proces može da pređe na narednu iteraciju, tek kada svi procesi završe tekuću iteraciju. Ovakvo usklađivanje aktivnosti se naziva **barijerna sinhronizacija** (*barrier synchronization*) i zahteva brojanje procesa koji su završili tekuću iteraciju, odnosno, koji su stigli do sinhronizacione barijere. Radi toga je potrebno odabrati posebnu lokaciju zajedničke radne memorije, nazvanu brojač i nameniti je za brojanje procesa, pristiglih do sinhronizacione barijere. Kada proces stigne do sinhronizacione barijere, on preuzima i uveća sadržaj brojača za vrednost 1. Pri tome se ne sme dozvoliti da više procesa, jedan za drugim, pročitaju isti sadržaj brojača i da zatim, nakon uvećanja pročitanog sadržaja za vredost 1, opet, jedan za drugim, upišu uvećani sadržaj natrag, jer tada brojač neće sadržati ispravan broj procesa, pristiglih do sinhronizacione barijere. Znači, potrebna je nedeljivost čitanja, uvećanja i pisanja vrednosti brojača. Sinhronizaciona barijera se može ostvariti pomoću procesorske naredbe **čitaj-uvećaj-piši** (*fetch-and-add*). Uslov je

da izvršavanje te naredbe zauzme brojač, koji je naveden kao operand naredbe, tako da između čitanja i pisanja ove lokacije nisu dozvoljeni drugi pristupi brojaču. Ovakvo zauzimanje brojača može da obezbedi spojna mreža, ali i poseban kontroler memorijskog modula. Za ispravno brojanje procesa, pristiglih do sinhronizacione barijere, je važno da brojač u početku sadrži vrednost 0. Pri tome, svaki od procesa, pristiglih do sinhronizacione barijere, ostaje uz barijeru u radnom čekanju dok god je sadržaj brojača manji od ukupnog broja procesa. Kada i poslednji proces pristigne do sinhronizacione barijere, sadržaj brojača više nije manji od ukupnog broja procesa, pa svi procesi mogu preći na narednu iteraciju. Pri tome, sadržaj brojača ne sme biti anuliran, da bi svaki od procesa, koji se nalaze u radnom čekanju, ustanovio da su svi procesi stigli do sinhronizacione barijere. U međuvremenu, procesi koji su prešli u narednu iteraciju mogu da uvećaju brojač. Zato, da bi se isti brojač koristio za sinhronizacionu barijeru i u narednoj iteraciji, neophodno je da, prilikom narednog dolaska do sinhronizacione barijere, procesi ostanu u radnom čekanju dok je sadržaj brojača veći od ukupnog broja procesa. Pri tome se podrazumeva da poslednji proces, koji na kraju naredne iteracije stigne do sinhronizacione barijere, anulira brojač. Precizan opis protokola, koga mora da poštuje svaki od procesa, da bi se ostvarila sinhronizaciona barijera, opisuje funkcija `barrier`, napisana programskim jezikom C:

```
void barrier(int *counter)
 { char state;
   state = (*counter < NUMBER);
   if(fetch_and_add(counter) == (2*NUMBER-1))
    *counter = 0;
   while(state == (*counter < NUMBER)); };
```

Za izvršavanje funkcije `fetch_and_add` se podrazumeva da odgovara izvršavanju procesorske naredbe za čitanje i uvećavanje. Poziv ove funkcije vraća zatečenu vrednost brojača i, uz to, uvećava brojač za 1. Konstanta `NUMBER` odgovara ukupnom broju procesa (procesora), parametar `counter` predstavlja brojač, a lokalna promenljiva `state` određuje za svaki proces posebno da li on čeka dok je sadržaj brojača manji ili dok nije manji od ukupnog broja procesa. U toku izvršavanja funkcije `barrier` procesi se nalaze u radnom čekanju.

12.3. PITANJA

1. Šta je uzrokovalo pojavu *RISC* arhitekture?
2. Šta je omogućila *RISC* arhitektura?
3. Koja su fizička ograničenja skraćenja procesorskog ciklusa kod poluprovodničke tehnologije?
4. Od čega zavisi prečnik računara?
5. Da li skrivena memorija utiče na prečnik računara?
6. Od čega zavisi veličina ubrzanja obrade podataka, nastala primenom paralelizma?

7. Koji modeli računanja postoje?
8. Koji oblici paralelizma su nevidljivi za programere?
9. Kako se na osnovu broja tokova naredbi i broja tokova podataka dele paralelni računari?
10. U koju klasu računara spadaju sistolički računari?
11. Od čega se sastoje *SIMD* računari?
12. Šta je slaba tačka *SIMD* računara?
13. U čemu se razlikuju multiprocesori i multiračunari?
14. Šta karakteriše *NUMA* multiprocesore?
15. Na čemu se zasniva otpornost na kvarove?
16. Šta karakteriše *RISC* procesore?
17. Šta su super-skalari?
18. Kakvi prekidi mogu biti kod super-skalara?
19. Kakvi su formati mašinskih naredbi kod *RISC* procesora?
20. Za šta su zaduženi kompajleri kod *RISC* procesora?
21. Za šta su zaduženi *RISC* procesori?
22. Šta podržava *IEEE 754* standard za aritmetiku realnih brojeva?
23. Šta je karakteristično za *INTEL PENTIUM Pro* procesor?
24. Šta omogućuje *RAID*?
25. Koje tehnologije masovnih memorija postoje?
26. Šta je karakteristično za invertovanu tabelu stranica?
27. Šta je karakteristično za spojne mreže?
28. Kako se dele spojne mreže?
29. Kako se dele višestepene prekidačke mreže?
30. Šta je karakteristično za višestepene prekidačke mreže?
31. Koje vrste statičkih spojnih mreža postoje?
32. Šta karakteriše hiperkocku?
33. Šta omogućuje barijerna sinhronizacija?
34. Na čemu se zasniva barijerna sinhronizacija?

13. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 2000. GODINE

13.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 2000. GODINE

Nastavak usavršavanja tehnologije veoma visoke integracije je omogućio smeštanje na desetine miliona tranzistora na jedan čip. To je omogućilo da se u mikro-procesorskoj tehnoligiji realizuju ideje koje su nekada bile isključivo vezane za velike i super-računare. Tako je izbrisana granica između mikro, mini, velikih, a delom i super-računara, pa se kao prirodnija nametnula podela na **ugrađene** (*embedded*) **računare**, **radne stanice** (*desktop computer*, *workstation*) i **servere**. Ugrađeni računari predstavljaju upravljački deo raznih uređaja. Radne stanice opslužuju pojedinačne korisnike i omogućuju ne samo njihov lokalni rad, nego i pristup Internetu. Serveri posredstvom Interneta pružaju razne usluge korisnicima radnih stanica.

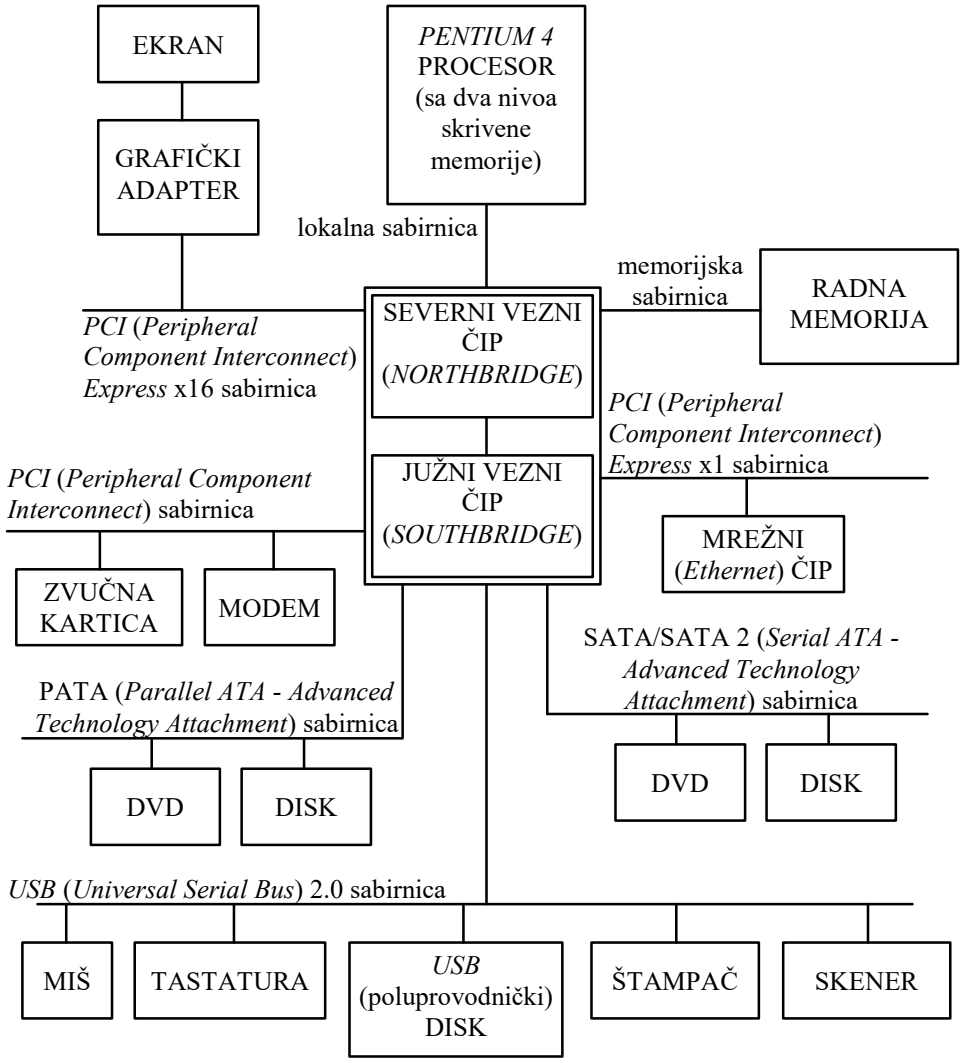
UGRAĐENI RAČUNARI

Ugrađeni računari su vezani za primene sa više ili manje strogim ograničenjima na vreme koje je raspoloživo za obradu podataka (*real-time systems*). Ovakve primene često podrazumevaju obradu govornih, video i drugih signala. Specifični zahtevi ovakvih obrada, kao što su precizni numerički proračuni za vrlo dugačke nizove podataka, su doveli do pojave posebne vrste procesora, koji se označavaju skraćenicom *DSP* (*digital signal processor*). Za ugrađene računare je važna mala potrošnja energije i mogućnost rada u nepovoljnim uslovima visoke ili niske temperature, visoke vlažnosti, vibracija, prisustva prašine i agresivnih materija, jer uređaji, u koje su oni ugrađeni, obično imaju baterijsko napajanje i predviđeni su za rad na otvorenom prostoru ili u industrijskim pogonima.

Posebnu podvrstu ugrađenih računara predstavljaju računari za jednokratnu upotrebu (*disposable computer*). Primer njihove upotrebe pružaju *RFID* (*Radio Frequency IDentification*) čipovi koji se ugrađuju u proizvode radi njihove identifikacije. Kada se nađu u zoni dovoljno jakih radio signala, ovi čipovi indukuju dovoljno energije da emituju identifikacioni broj sadržan u sebi ili obave neku drugu kratkotrajnu operaciju.

RADNE STANICE

Radne stanice imaju bitne karakteristike nekadašnjih velikih računara, kao što su veliki adresni prostor, virtuelna memorija, više sabirnica ili velika masovna memorija. Slika 13.1.1 sadrži prikaz organizacije radne stanice, zasnovane na *PENTIUM 4* procesoru.



Slika 13.1.1 Organizacija radne stanice koja je zasnovana na *PENTIUM 4* procesoru

SERVERI

Serveri se dele na multiprocesore i multiračunare. Multiprocesore karakteriše **deljena radna memorija** (*shared-memory systems*). Ona je sastavljena od više modula. Njoj, posredstvom spojne mreže, mogu pristupati svi procesori. Njihova saradnja se ostvaruje razmenom podataka posredstvom istih memorijskih lokacija. To stvara probleme, jer razni procesori pristupaju istim memorijskim lokacijama u nepredvidivom redosledu, koga još komplikuje činjenica da sadržaj iste memorijske lokacije može biti repliciran u više skrivenih memorija. Zato je potrebno definisati

pravila ponašanja memorije (*memory semantycs*) koja ograničavaju mogući redosled pristupanja istim memorijskim lokacijama. Ovakva pravila oblikuju **modele konzistentnosti** (*consistency models*). Tako, **stroga konzistentnost** (*strict consistency*) podrazumeva potpunu serijalizaciju pristupa, jer garantuje da svako čitanje iste memorijske lokacije preuzima poslednju vrednost, koja je upisana u tu lokaciju. Stroga konzistentnost je suviše kruta, jer smanjuje stepen paralelizma u multiprocesoru. **Sekvencijalna konzistetnost** (*sequential consistency*) dozvoljava da svako čitanje iste memorijske lokacije ne preuzima poslednju vrednost, koja je upisana u nju, nego može da preuzme i neku stariju vrednost, uz garanciju da svaki procesor vidi isti redosled upisivanja vrednosti za svaku memorijsku lokaciju. Postoje i drugi, još labaviji modeli konzistentnosti, koji su lakši za izvedbu, a daju dovoljno pravilnosti za pisanje programa sa predvidivim ponašanjem.

Posebnu podvrstu multiprocesora predstavljaju COMA (*cache only memory access*) multiprocesori. Njihova celokupna deljena radna memorija se ponaša kao skrivena memorija. Znači svaki procesor ima njenu kopiju. O konzistentnosti ovih kopija se brinu posebni protokoli (*cache coherence protocols*).

Primer multiprocesora je *SEQUENT NUMA-Q 2000*. Njegov osnovni gradivni modul sadrži 4 *INTEL PENTIUM Pro* procesora i 4 giga bajta radne memorije. Ove gradivne module objedinjuju u multiprocesor posebni kontroleri koji su međusobno povezani. Svaki od ovih kontrolera poseduje skrivenu memoriju od 32 megabajta, a svi zajedno su odgovorni za usaglašenost sadržaja svih skrivenih memorija.

Multiračunare karakteriše **distribuirana radna memorija** (*distributed-memory system*). Pošto svaki računar ima sopstvenu lokalnu radnu memoriju, koja je nepristupačna za druge računare, razmena poruka je jedini način njihove saradnje. Razmena poruka se oslanja na posebne biblioteke, kao što je ***MPI*** (*Message Passing Interface*) biblioteka. One realizuju odgovarajuće komunikacione protokole i olakšavaju komunikaciju procesa, aktivnih na raznim računarima, omogućujući njihovu sinhronizaiciju.

Uslovna podela multiračunara se zasniva na karakteristikama računara od kojih su multiračunari sastavljeni. U klasi super-računara se nalaze multiračunari, sastavljeni od procesora sa sopstvenom lokalnom radnom memorijom, koji su povezani brzim, namenski razvijenim spojnim mrežama. Ovakvi multiračunari se označavaju skraćenicom ***MPP*** (*massively parallel processors*). U klasi **klastera** (*cluster*, *cluster of workstations*) se nalaze multiračunari koji su sastavljeni od radnih stanica, povezanih komercijalnim spojnim mrežama. Kao komercijalne spojne mreže koriste se eternet mreže, **prekidači** (*switch*) i **sistemske mreže** (*SAN*, *storage or system area network*). Prekidači omogućuju prenos poruka između linija koje su za njih spojene. Njihova komercijalna primena je vezana za pojavu prekidača na čipu. Sistemske mreže istiskuju ulazno-izlazne sabirnice. One su izvorno namenjene za povezivanje jedinica masovne memorije sa računarom na udaljenostima do 100 metara.

Primer *MPP* multiračunara je *CRAY T3E*. On sadrži do 2048 procesora *DEC ALPHA 21164*, svaki sa do 2 gigabajta radne memorije. Ovi procesori su povezani dvosmernom trodimenzionalnom mrežom (*full-duplex 3D torus*) u kojoj je svaki čvor povezan sa 4 suseda iz svoje ravni i sa po jednim susedom iz gornje i donje ravni. Za vezu 512 čvorova ovakva trodimenzionalna mreža ima 8×8×8 čvorova. Pored toga, procesori su povezani i posebnom mrežom za razmenu poruka, na koju su zakačeni i razni ulazno-izlazni uređaji i uređaji masovne memorije.

Primer klaster multiračunara je *GOOGLE*. On je namenjen za pretraživanje Interneta (*search engine*). Jedan njegov klaster se sastoji se od 5120 radnih stanica. U sastav jedne radne stanice ulazi *PENTIUM 4* procesor, 512 megabajta radne memorije, 80 gigabajtni disk i napajanje sa hlađenjem. Radne stanice su raspoređene u 64 ormara. Svaki ormar sadrži 80 radnih stanica i prekidač koji ih povezuje međusobno. Prekidači iz ormara su povezani sa dva prekidača koji *GOOGLE* klaster povezuju na Internet.

PARALELIZAM UNUTAR PROCESORA

Razvoj tehnologije visokog stepena integracije je prvo omogućio uvođenje protočne strukture u procesor, radi izvršavanja jedne procesorske naredbe u približno jednom procesorskom ciklusu, a zatim je omogućio uvođenje više protočnih struktura u procesor (super-skalari) i izvršavanje više procesorskih naredbi u jednom procesorskom ciklusu (*multiple-issue processors*). Takva vrsta paralelizma unutar procesora ne zahteva programersku asistenciju, jer je na nivou procesorskih naredbi (*instruction level parallelism*) i moguća je između nezavisnih naredbi. Broj istovremeno izvršenih naredbi u jednom procesorskom ciklusu zavisi od broja protočnih struktura procesora, ali i od broja raspoloživih nezavisnih naredbi.

Jedan način da se poveća broj raspoloživih nezavisnih naredbi je da izvršavanje pojedinih naredbi započne pre nego što je poznato da li će one uopšte biti izvršavane (*speculative execution*). Na primer, ako u programu postoji grananje sa izračunavanjem vrednosti različitog aritmetičkog izraza u svakoj grani, tada se mogu izvršiti naredbe radi računanja vrednosti oba aritmetička izraza pre nego je poznato koja od grana će biti odabrana u izvršavanju programa. To znači da bi za iskaz:

```
if (a>b)
   a = a - b;
else
   b = b - a;
```

(iz primera računanja najvećeg zajedničkog delioca) bile izvršene naredbe koje računaju vrednost oba aritmetička izraza pre nego je poznato koji od njih će biti potreban. Efekti izvršavanja pomenutih naredbi se prihvataju tek kada se ispostavi da je njihovo izvršavanje bilo potrebno. U suprotnom se javlja problem ako je

njihovo izvršavanje izazvalo izuzetak. Zato se obrada ovakvog izuzetka mora odložiti do trenutka kada je sigurno da je izvršavanje naredbe bilo potrebno.

Drugi način da se poveća broj raspoloživih nezavisnih naredbi je da programer označi nezavisne delove svog programa, koji se nazivaju **niti** (*thread*). Tada je redosled izvršavanja naredbi precizno određen samo unutar svake niti, ali ne i između njih. Pošto su naredbe iz raznih niti istog programa međusobno nezavisne, one se mogu izvršavati u proizvoljnom redosledu (*thread level parallelism*). Prema tome, izvršavanja naredbi iz raznih niti istog programa mogu istovremeno počinjati (*simultaneous multithreading*), kad god za to postoje raspoloživi resursi u procesoru.

Nezavisne naredbe može pronalaziti ili procesor ili kompajler. Praksa je pokazala da prvi pristup komplikuje i usporava procesor. Zato je prirodno takav posao prepustiti kompajleru. Njegov zadatak je (1) da generiše naredbe koje će na najbolji način uposliti procesor i iskoristiti njegov paralelizam i (2) da među njima pronađe nezavisne naredbe čije izvršavanje može istovremeno započeti i odvijati se paralelno. Kada pronađe takve naredbe, kompajler ih može spakovati kao jednu veliku naredbu. Za procesore koji mogu da prihvate ovakvu veliku naredbu se kaže da imaju **dugačak mašinski format naredbe**. Engleska skraćenica ovog naziva je *VLIW* (*very long instruction word*). Pošto je paralelizam između ovakvih naredbi eksplicitno označen, računari sa ovakvim procesorima se nazivaju **računari sa eksplicitnim paralelnim naredbama**. Engleska skraćenica ovog naziva je *EPIC* (*explicitly parallel instruction computers*). Da bi preklapanje izvršavanja ovakvih naredbi bilo jednostavnije i bez zadrški, potrebno je izbeći uslovne izmene u redosledu izvršavanja naredbi. Izbegavanje grananja u programu se zasniva na konceptu **predikatskih naredbi** (*predicated instruction*). Njihovo izvršavanje započinje proverom važenja zadanog predikata (uslova) koga sadrži poseban jednobitni **predikatski registar** (*predicate register*). Ako predikat važi, izvršavanje predikatske naredbe se nastavlja, inače se obustavlja. Predikatski registar postavljaju posebne naredbe, uvedene umesto uslovnih upravljačkih naredbi. U ovakvoj situaciji, uslovnu upravljačku naredbu zamenjuje posebna naredba koja postavi zadani predikatski registar u skladu sa važenjem datog uslova. Naredbama iz jedne grane programa odgovaraju ekvivalentne predikatske naredbe za čije izvršavanje je neophodno da zadani predikatski registar sadrži vrednost 1, dok naredbama iz druge grane programa odgovaraju ekvivalentne predikatske naredbe za čije izvršavanje je neophodno da zadani predikatski registar sadrži vrednost 0. Tako, u slučaju iskaza:

```
if (a>b)
   a = a - b;
else
   b = b - a;
```

predikatski registar bi postavile naredbe koje određuju važenje uslova, ako je uslov tačan. Prvom iskazu dodele bi odgovarale predikatske naredbe za čije izvršavanje je

potrebno da predikatski registar bude postavljen, a drugom iskazu dodele bi odgovarale predikatske naredbe za čije izvršavanje je potrebno da predikatski registar ne bude postavljen.

EVOLUCIJA OPTIČKIH DISKOVA

Povećanje kapaciteta optičkih diskova je vezano za pojavu optičkih diskova sa oznakama *Blu-ray* (25 i više gigabajta) i *HD DVD* (15 i više gigabajta). Noviji standardi sistema datoteka, *UDF* (*Universal Disk Format*, *ISO/IEC 13346*) i *ISO/IEC 13490*, donose unapređenja vezana za nazive datoteka, njihovu maksimalnu veličinu, kao i za dopisivanje podataka.

13.2. ISHODI EVOLUCIJE ARHITEKTURE RAČUNARA U PERIODU OKO 2000. GODINE

ARHITEKTURA NAREDBI ZA *INTEL ITANIUM*

INTEL ITANIUM spada u *RISC EPIC* procesore. Arhitektura naredbi ovog procesora podržava 64-bitni adresni prostor. U ovoj arhitekturi na raspolaganju su 128 registara opšte namene od 64 bita, 128 registra za realne brojeve od 82 bita, 64 jednobitna predikatska registra (*predicate registers*), 8 registara za indirektne skokove (*branch registers*) od 64 bita i razni drugi namenski registri. Ukupno ima 111 tipova naredbi.

Sve naredbe su predikatske i dobavljaju se u grupama koje se nazivaju svežnjevi. Mašinski format svežnja (*bundle*) naredbi ove arhitekture izgleda:

41 bit	41 bit	41 bit	5 bita

Njegova prva 3 polja su namenjena za 3 mašinske naredbe. Svaka od ovih mašinskih naredbi u 4 najznačajnija bita sadrži kod tipa naredbe, a u 6 najmanje značajnih bita sadrži kod predikatskog registra. Namena ostalih bita zavisi od tipa naredbe. Tako, na primer, ostali biti mogu sadržati relativni kod naredbe i kodove registara sa operandima. U poslednjem polju svežnja se nalazi 5 bitni kod svežnja. On olakšava dekodiranje naredbi iz svežnja. Pošto se podrazumeva eksplicitno označavanje paralelnih naredbi, koje mogu obuhvatati naredbe iz jednog ili više svežnjeva, kod svežnja ujedno označava kraj grupe paralelnih naredbi u svežnju. Na taj način grupu paralelnih naredbi čine sve naredbe koje se nalaze između dve uzastopne oznake kraja u istom ili raznim svežnjevima, zavisno od slučaja.

Implementacija ove arhitekture se oslanja na 3 nivoa skrivene memorije.

13.3. PITANJA

1. Kakva podela računara se nametnula u evolucionom periodu oko 2000. godine?
2. Koji pojmovi su vezani za ugrađene račnare?

3. Šta karakteriše radne stanice?
4. Kako se dele serveri?
5. Koji modeli konzistentnosti deljene radne memorije postoje?
6. Kako se dele multiračunari?
7. Koje spojne mreže koriste klasteri?
8. Kako se povećava broj raspoloživih nezavisnih naredbi, koje se mogu istovremeno izvršavati?

14. EVOLUCIONI PERIOD ARHITEKTURE RAČUNARA OKO 2010. GODINE

14.1. SVOJSTVA ARHITEKTURE RAČUNARA U PERIODU OKO 2010. GODINE

Nastavak usavršavanja tehnologije veoma visoke integracije i time uzrokovano povećavanje broja tranzistora na jednom čipu su omogućili pojavu procesora sa dva jezgra, prvo *IBM POWER4 PowerPC*, a nešto kasnije kasnije i *AMD Athlon64 X2*. Prvi dvojezgarni procesori su sadržali dva odvojena jezgra, sa kasnijom tendencijom veće integracije jezgara. Postojanje dva jezgra u jednom procesoru je omogućilo paralelno izvršavanje dva nezavisna niza naredbi. Dalji razvoj je doveo do procesora sa još više jezgara (3,4,6,8,10,12,...), koji omogućavaju još viši nivo paralelizma. Predstavljene su i hibridne arhitekture višejezgarnih procesora, gde se u jednom čipu kombinuje više različitih jezgara. Primer takvog procesora je *Cell*, koji u sebi sadrži jedno *PowerPC* jezgro i osam specijalizovanih *SIMD* jezgara, međusobno povezanih brzom sabirnicom.

PROCESORI

Na tržištu personalnih računara vodeće mesto imaju procesori sa x86 arhitekturom naredbi, ali sve veći značaj dobijaju *RISC* procesori sa *ARM* arhitekturom naredbi, najviše zahvaljujući širenju prenosnih računarskih uređaja. Jedna od osnovnih karakteristika ovih procesora je smanjena potrošnja električne energije. I za *ARM* procesore je karakteristično širenje adresnog prostora sa 32 na 64 bita, kao i prelaz na veći broj jezgara. Razvoj i proizvodnja *ARM* procesora su razdvojeni, jer kompanija, koja razvija *ARM* arhitekturu naredbi, ne proizvodi *ARM* procesore, nego prodaje licence za njihovu proizvodnju drugim kompanijama, pa prizvođača *ARM* procesora ima više. U komercijalnoj upotrebi se nalaze i *IBM POWER*8 12-jezgarni procesor, *SPARC* (*Fujitsu SPARC64 X+*, *Oracle SPARC T5*) 16-jezgarni procesor i *Loongson 3B* 8-jezgarni procesor (*MIPS64* arhitektura naredbi sa hardverski podržanom emulacijom x86 arhitekture naredbi).

INTERNET

Uloga Interneta u povezivanju računara je naglasila značaj pitanja međusobne zaštite njegovih korisnika i uzrokovala potrebu pronalaženja načina anonimnog pristupa globalnoj računarskoj mreži, radi sakrivanja pristupa Internetu jednih korisnika od drugih. To je dovelo do razvoja *TOR*-a (*The Onion Router*), prve javno dostupne mreže računara za anonimno korišćenje Internet servisa, ali i do podsticaja razvoja hardvera otvorenog koda (*open-source hardware*), firmvera otvorenog koda (*open-source firmware*, *Coreboot*) za postojeći hardver, kao i do šire upotrebe softvera otvorenog koda (*open-source software*).

GRAFIČKE KARTICE

Razvoj grafičkih kartica je doveo do povećanja broja jednostavnih jezgara u njima, kako bi se operacije trodimenzionalnog prikaza što više ubrzale. Sa povećavanjem funkcionalnost ovih jezgara, postala je moguća njihova upotreba i za proračune koji nisu vezani samo za grafički prikaz. Ovakva jezgra imaju *SIMD* svojstva, jer omogućuju istovremeno izvršavanje iste operacije za nizove različitih podataka. To je dovelo do pojave *GPGPU* (*General-Purpose computing on Graphics Processing Units*), odnosno do upotrebe grafičkih kartica u proračunima opšte namene. Ovakva upotreba je uglavnom bila vezana za računarske razvojne laboratorije, sve do 2007. godine, kada je predstavljena prva verzija *Nvidia CUDA* (*Compute Unified Device Architecture*) harvdersko-softverske arhitekture. Ona omogućava programiranje jezgara *Nvidia* grafičkih kartica posredstvom namenskih proširenja C programskog jezika. Godinu dana kasnije razvijena je i *OpenCL* (*Open Computing Language*) 1.0 specifikacija. Ona omogućava *GPGPU* programiranje na svakoj grafičkoj kartici za koju postoji razvijen *OpenCL* drajver. Kasnije su se pojavile *GPGPU* kartice čija primarna svrha nisu grafičke primene, nego ubrzanje proračuna (*Ageia PhysX*, *Nvidia Tesla*, *Intel Xeon Phi*).

SUPER-RAČUNARI

Super-računari se uglavnom oslanjaju na tržišno dostupne procesore i komponente. Kako se broj čvorova (procesora) u njima povećava, tako se sve više teži ka korišćenju komponenti sa smanjenom potrošnjom, odnosno sa boljim odnosom računarske snage i potrošnje električne energije. Zaključno sa sredinom 2015. godine, preko 85% najbržih super-računara koristi 64-bitne procesora sa x86 arhitekturom naredbi, dok 97% najbržih super-računara koristi *GNU/Linux* operativne sisteme. U 2008. godini je napravljen *IBM Roadrunner*, prvi super-računar koji koristi hibridne čvorove, jer se svaki od njih sastoji od 64-bitnog x86 procesora i *PowerXCell* procesora (za ubrzanje proračuna). Današnji najbrži super-računari za ubrzanje proračuna često koriste grafičke kartice, kao i kartice posebno napravljene za tu namenu (*Intel Xeon Phi*, *Nvidia Tesla*).

PRENOSNI RAČUNARSKI UREĐAJI

Razvoj prenosnih računarskih uređaja, prevashodno mobilnih telefona, je doveo prvo do pojave namenskih operativnh sistema za prenosne uređaje, a zatim i do toga da mobilni telefoni počnu dobijati mnoga svojstva personalnih računara. Današnji mobilni telefoni i tableti često u sebi sadrže dvo, četvoro pa i osmojezgarne procesore, kao i više gigabajta *RAM* memorije i predstavljaju minijatrune personalne računare. U mobilnim telefonima se uglavnom koriste *ARM* procesori (zbog smanjene potrošnje električne energije), ali ima i onih koji su zasnovani na x86 procesorima. Među operativnim sistemima mobilnih računarskih uređaja su najrasprostranjeniji *Android* (zasnovan na *Linux* jezgru), i *iOS* (zasnovan na *OSX Darwin* jezgru). Jedna od osnovnih karakteristika ovih uređaja je da se

interakcija sa korisnikom ostvaruje preko ekrana osetljivog na dodir, za razliku od personalnih računara koji uglavnom koriste kombinaciju tastature i miša.

14.2. ISHODI EVOLUCIJE ARHITEKTURE RAČUNARA U PERIODU OKO 2010. GODINE

ARHITEKTURA NAREDBI ZA AMD *ATHLON 64/OPTERON*

Opteron i *Athlon 64* su prvi x86 kompatibilni procesori čija arhitektura naredbi podržava 64 bitni adresni prostor. Iako je reč o 64 bitnim procesorima, oni su unazad kompatibilni sa x86 procesorima i mogu izvršavati i 32 bitne i 16 bitne naredbe ranijih generacija procesora. Ovi procesori su prvi x86 kompatibilni procesori koji u sebi sadrže ugrađeni kontroler memorije. U ovoj arhitekturi naredbi na raspolaganju su 51 sistemski registar (za sistemske programe, kao što su operativni sistem ili dibager), 16 registara opšte namene sa 64 bita (čiji 32, 16 i 8 bitni delovi se mogu koristiti kao posebni registri), 8 registara za aritmetiku realnih brojeva sa 64 bita, 16 registara za *SIMD* naredbe sa 256 bita, kao i nekoliko statusnih i kontrolnih registara.

Mašinski format naredbe uključuje i posebne prefiks i postfiks bajtove koji dodatno određuju ponašanje naredbe, što komplikuje izgled mašinskog formata naredbe. Mašinski format naredbe se može sastojati od jednog do 15 bajtova. U okviru njega su opisani operacija, ulazni i izlazni operandi, kao i modifikatori naredbe (predstavljeni prefiks i postfiks bajtovima).

14.3. PITANJA

1. Šta karakteriše procesore u evolucionom periodu oko 2010. godine?
2. Šta karakteriše Internet u evolucionom periodu oko 2010. godine?
3. Šta omogućuju *GPGPU* kartice?
4. Po čemu se razlikuju *CUDA* i *OpenCL*?
5. Zašto se u mobilnim telefonima pretežno koriste *ARM* procesori?

15. PROCENA OSOBINA RAČUNARA

15.1. NAČIN PROCENE OSOBINA RAČUNARA

Za uspešan rad računara, neophodno je usaglasiti raznorodne komponente koje ulaze u njegov sastav. Raznorodnost ovih komponenti otežava njihovo usaglašavanje, ali i procenu ukupnih osobina računara.

Sa stanovišta procene osobina računara, procesor karakterišu veličina procesorskih naredbi i vreme njihovog izvršavanja. Vreme izvršavanja procesorske naredbe je određeno trajanjem procesorskog ciklusa i njihovim brojem u procesorskoj naredbi. Pošto različite procesorske naredbe sadrže različit broj procesorskih ciklusa, procesor karakteriše prosečno vreme izvršavanja procesorskih naredbi. Ono se obično ne određuje kao aritmetička sredina, nego kao suma proizvoda verovatnoća izvršavanja pojedinih tipova procesorskih naredbi i vremena potrebnih za njihovo izvršavanje. Ovakve verovatnoće se određuju empirijski. Primer tako određenih verovatnoća izvršavanja pojedinih tipova procesorskih naredbi, poznat pod nazivom *Gibson mix*, dodeljuje naredbama prenosa između registara procesora i lokacija radne memorije verovatnoću 0.31, uslovnim naredbama verovatnoću 0.17, naredbama celobrojne aritmetike verovatnoću 0.07, naredbama aritmetike realnih brojeva verovatnoću 0.12, a ostalim naredbama verovatnoću 0.33. Ako se zna srednji broj ciklusa po procesorskoj naredbi, tada se procesor karakteriše vremenom njegovog ciklusa, ili, još češće, taktom, odnosno frekvencijom (brojem procesorskih ciklusa u jedinici vremena). Količnik frekvencije procesora i srednjeg broja procesorskih ciklusa po procesorskoj naredbi predstavlja srednji broj (frekvenciju) naredbi koje procesor može da izvrši u jedinici vremena. Ovakav srednji broj naredbi se obično izražava u milionima naredbi u sekundi. Engleska skraćenica naziva ovakve jedinice je *MIPS* (*million instructions per second*). Na primer, *RISC* procesor sa procesorskim ciklusom od 10^{-8} sekundi ima frekvenciju 100 megaherca. Ako prosečna naredba ovoga procesora traje 2 procesorska ciklusa, tada ovaj procesor može u proseku da izvrši 50 miliona procesorskih naredbi u sekundi (50 *MIPS*). Važno je naglasiti da *MIPS* nije podesna mera za međusobno poređenje procesora, zbog velikih razlika između skupova naredbi različitih vrsta procesora. Na primer, *CISC* procesori, kao što su procesori iz *DEC VAX* serije, omogućuju da se, u okviru izvršavanja jedne procesorske naredbe, preuzmu dva operanda iz radne memorije, obavi njihovo sabiranje, i zbir smesti u radnu memoriju. Za isti posao su, kod tipičnih *RISC* procesora, potrebne 4 procesorske naredbe. Iz istih razloga nije uputno porediti procesore ni na osnovu broja operacija aritmetike realnih brojeva koje oni mogu da obave u jedinici vremena. Engleska skraćenica naziva ovakve jedinice je *megaFLOPS* ili *MFLOPS* (*million floating-point operations per second*).

Sa stanovišta procene osobina računara, radnu memoriju karakterišu veličina njene reči, ukupan broj reči i vreme ciklusa. Recipročna vrednost vremena

memorijskog ciklusa određuje propusnost radne memorije, odnosno broj čitanja/pisanja lokacija radne memorije u jedinici vremena.

Sa stanovišta procene osobina računara, sabirnicu karakteriše propusnost (*bandwidth*), koja se izražava brojem bajta u sekundi.

Sa stanovišta procene osobina računara, disk, kao sinonim za masovnu memoriju, karakterišu veličina bloka, ukupan broj blokova i srednje vreme pristupa (bloku diska). Ovo vreme je jednako sumi srednjeg vremena pomeranja, srednjeg vremena rotacije, vremena prenosa i vremena kontrolera. Srednje vreme pomeranja obično daje proizvođač, podrazumevajući ravnomerno pristupanje svim blokovima diska. Recipročna vrednost srednjeg vremena pristupa određuje propusnost diska, odnosno broj čitanja/pisanja blokova diska u jedinici vremena.

Na primeru prethodne četiri organizacione komponente računara se može pokazati problematičnost njihovog usaglašavanja. Da bi procesor mogao da radi punom brzinom, do njega mora da stigne odgovarajući broj mašinskih naredbi i podataka. To nije moguće bez odgovarajuće propusnosti sabirnice i radne memorije. Skrivena memorija može da dovede u sklad pomenute komponente, ali za dovoljno visoku verovatnoću pogodaka. Važno je uočiti da ta verovatnoća zavisi od izvršavanog programa. Poseban problem predstavlja propusnost diska, jer ona zavisi i od osobina operativnog sistema i od karakteristka izvršavanog programa. Prethodno ukazuje da uspešnost računara podjednako zavisi od:

1. osobina i usaglašenosti njegovih organizacionih komponenti,
2. od osobina izvršavanog programa, odnosno od toga da li u njemu preovlađuju proračuni, interaktivni rad ili ulaz i izlaz podataka,
3. od operativnog sistema, ali i
4. od kompajlera.

Zbog uticaja više raznorodnih uzroka, teško je predvideti (proceniti) ponašanje računara u raznim situacijama. Iz istih razloga je problematično i poređenje računara na osnovu osobina njihovih pojedinih organizacionih komponenti. Zato se provera uspešnosti arhitekture računara, kao i poređenje raznih računara, zasnivaju na posmatranju računara kao celine, odnosno na posmatranju vremena potrebnog za izvšavanje reprezentativnih programa (*benchmark*). Za ovakva poređenja je važno da na raznim računarima budu obezbeđeni isti uslovi, znači isti operativni sistem i isti kompajler. Nekada se, kao reprezentativni programi, koriste delovi odabranih programa (*kernel*, *livermore loop*, *linpack*), za koje se pretpostavlja da mogu da pokažu osobine računara kao celine. S istom namerom se prave i posebni programi (*sintetic benchmark*, *whetstone*, *dhrystone*), u kojima se nastoji obezbediti da izvršavanja pojedinih tipova naredbi budu u odnosu zapaženom kod određene klase korisničkih programa. Međutim, najbolji reprezentativni programi su programi koje korisnik upotrebljava, jer jedino oni mogu ukazati na računar koji najbolje odgovara korisnikovim potrebama.

15.2. PITANJA

1. Od čega zavisi vreme izvršavanja procesorske naredbe?
2. Šta određuje količnik frekvencije procesora i srednjeg broja procesorskih ciklusa po procesorskoj naredbi?
3. Šta određuje recipročna vrednost vremena memorijskog ciklusa?
4. Šta određuje recipročna vrednost srednjeg vremena pristupa diska?
5. Od čega zavisi uspešnost računara?

16. SAVREMENI PERSONALNI RAČUNAR

Fizički delovi savremenog personalnog računara su: matična ploča sa kontrolerima, procesor, radna memorija, uređaji masovne memorije, grafički, zvučni i mrežni adapter, tastatura, miš, monitor, napajanje i kućište sa rashladnim sistemom. Najčešće korišćeni operativni sistemi na personalnim računarima su razne verzije operativnih sistema *Microsoft Windows*, *UNIX*, *GNU/Linux* i *Apple Mac OS X*.

16.1. PROCESOR

Dva najznačajnija proizvođača procesora za personalne računare (*x86* kompatibilni procesori) su *Intel* i *AMD*. Ostali proizvođači, kao što su *VIA*, *SiS* i *Transmeta*, pokrivaju veoma mali deo tržišta, i to uglavnom vezan za usko specijalizovane primene. Poslednje dve godine u svet personalnih računara je ušla još jedna procesorska arhitektura koja nije kompatibilna sa *x86*. U pitanju su procesori zasnovani na *ARM* arhitekturi, koji su se prvobitno koristili u mobilnim uređajima, ali se sve više pojavljuju kao osnova *tablet* i *netbook* računara. Bitna karakteristika svih procesora za personalne računare je da nominalno imaju istu arhitekturu naredbi (osnovne naredbe, *SSE - Streaming SIMD Extensions*, odnosno *3DNow*!). Praksa pokazuje da arhitekture naredbi procesora raznih proizvođača nisu identične, jer postoje programi koji se izvršavaju korektno na procesorima jednog poizvođača, a ne izvršavaju se korektno na procesorima drugog, i obratno.

Svaki procesor karakteriše njegov tip podnožija (*socket*). I *Intel* i *AMD* u svojoj ponudi imaju više familija procesora koji koriste različita podnožja. Tako, *Intel* u svojoj ponudi ima (između ostalih) familije procesora *Pentium*, *Celeron*, *Xeon*, *Core*, *Core 2* i *Core i7* (i derivati *Core i3* i *Core i*5), dok *AMD* ima (između ostalih) familije procesora *Athlon*, *Opetron*, *Turion*, *Sempron*, *Phenom* i *Phenom II*.

Još jedna bitna karakteristika procesora je njegova frekvencija. Frekvencije savremenih procesora dostižu do 3,8 GHz, iako su uz pomoć specijalnih rashladnih sistema moguće i više frekvencije (pri tome, treba imati u vidu da novije arhitekture procesora postižu bolje rezultate od starijih, iako rade na nižim frekvencijama). Pošto danas raširene tehnologije polako dostižu maksimum u pogledu frekvencije procesora, traže se drugi načini povećanja brzine izvršavanja programa. Tako se uvode 64-bitne arhitekture (*Athlon64*, krajem 2003. godine) i u jedan čip se smešta više procesora (*dual-core* procesori, sredina 2005. godine i *quad-core* procesori, sredina 2006. godine). Trenutno se radi na razvoju osmo i dvanaesto-jezgarnih *x86* procesora. Treba napomenuti i da se u oblasti numeričkih proračuna, umesto centralnog procesora, sve više koriste procesori sa grafičkih kartica koji već sada poseduju više desetina/stotina jezgara.

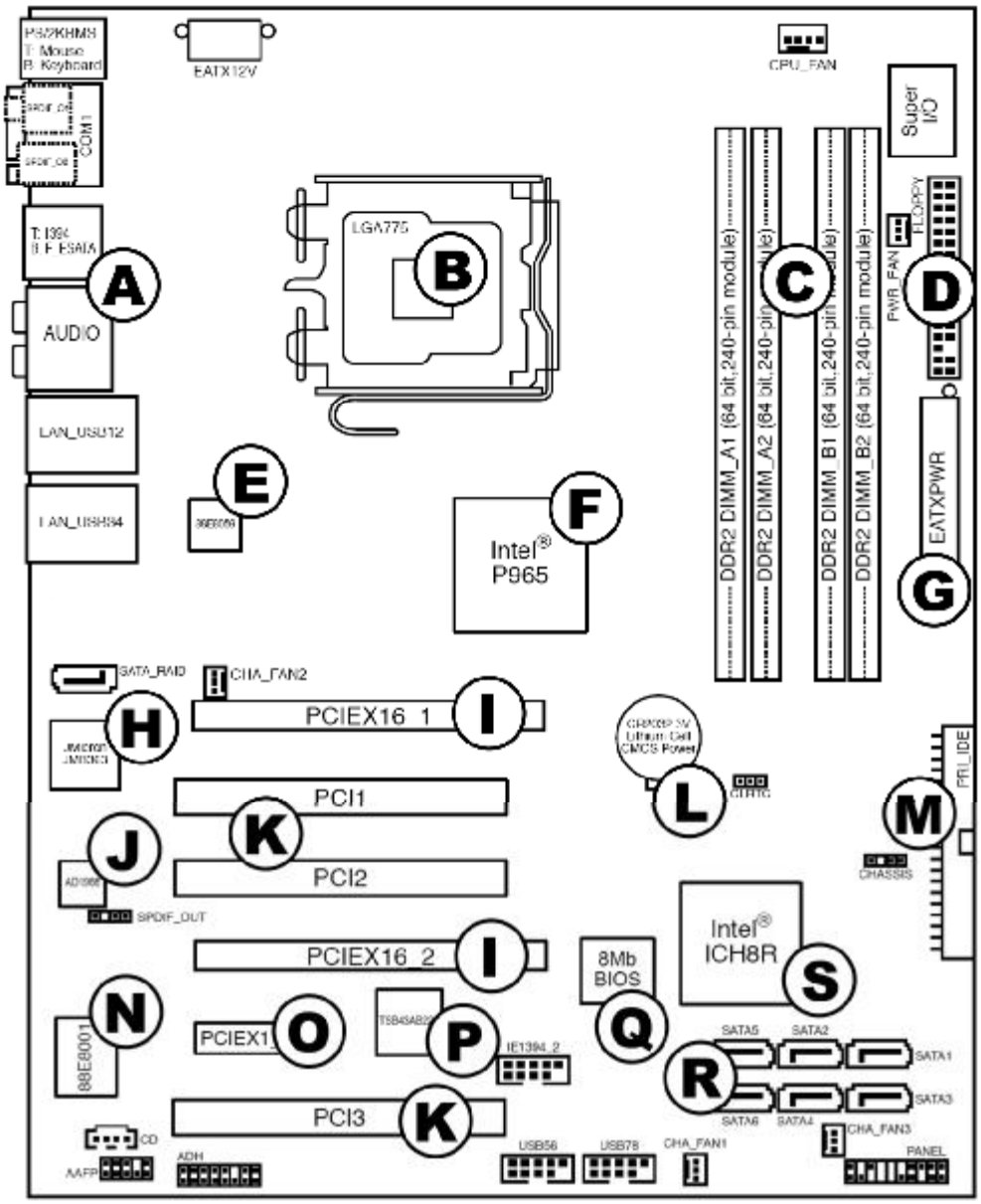
16.2. MATIČNA PLOČA

Matična ploča povezuje ostale delove računara. Na njoj se nalaze: podnožije za procesor iz određene familije, konektori za memorijske module, konektori za ulazno-izlazne uređaje, konektori za uređaje masovne memorije, vezni (*bridge*) čip(ovi), *BIOS* čip, konektori za napajanje, itd. Njih povezuju fizičke linije izvedene u višeslojnoj štampi na matičnoj ploči. Pored nabrojanih osnovnih delova, matična ploča može sadržati i razne dodatne delove, kao na primer, ugrađeni *RAID* kontroler, uređaje za podešavanje pojedinih parametara matične ploče u toku rada računara, konektor za dodatni grafički adapter, itd.

Postoji više standarda matičnih ploča koje propisuju njihovi proizvođači. Ovi standardi određuju vrstu napajanja i dimenzije ploča. Danas su najviše u upotrebi standardi *ATX* (*Advanced Technology Extended*) i *microATX*. 2003. godine je predložen *BTX* (*Balanced Technology Extended*) standard, ali je napušten krajem 2006. godine. Prenosivi personalni računari obično koriste specijalizovane matične ploče, prilagođene potrebama takvih računara.

Slika 16.2.1 prikazuje pojednostavljeni izgled matične ploče za *Core 2 Duo* procesor sa *LGA775* podnožjem. Ova matična ploča sadrži:

1. konektore za *USB*, *Ethernet*, *FireWire*, *eSATA*, audio ulaz i izlaz, tastaturu i miša (A),
2. podnožje za procesor (B),
3. konektore za memoriju (C),
4. konektor za *floppy* disk (D),
5. gigabitni *LAN PCIe* kontroler (E),
6. severni vezni čip (F),
7. konektor za napajanje (G),
8. *SATA II RAID* i *PATA* kontroler (H),
9. *PCIe* x16 konektore (I),
10. *HD* audio kontroler (J),
11. *PCI* konektore (K),
12. litijumska baterija za *RAM* memoriju *BIOS*-a i kratkospojnik za njeno brisanje(L),
13. *PATA* konektor (M),
14. gigabitni *LAN PCI* kontroler (N),
15. *PCIe* x1 konektor (O),
16. *FireWire* kontroler (P),
17. *BIOS* čip (Q),
18. *SATA* konektore (R)
19. južni vezni čip (S) i
20. više konektora za priključenje dodatnih ventilatora, napajanja i uređaja.



Slika 16.2.1 Pojednostavljeni izgled matične ploče

VEZNI (*BRIDGE*) ČIP

Vezni (*Bridge*) čip služi da poveže sve hardverske delove matične ploče u jednu celinu. Zbog svoje kompleksnosti, ovaj čip se obično sastoji iz dva dela, nazvana severni (*northbridge*) i južni (*southbridge*). Ova dva dela veznog čipa međusobno povezuje specijalizova brza sabirnica. Generalno, za severni vezni čip su povezani procesor, radna memorija i grafički adapter. Za južni vezni čip su povezani ostali uređaji. Postoje vezni čipovi koji nemaju prethodno navedenu organizaciju.

Severni vezni čip u sebi sadrži memorijski kontroler, kontroler brze sabirnice za procesor i kontroler sabirnice za grafički adapter (*AGP* ili *PCIe*). Južni vezni čip u sebi obično sadrži kontroler ulazno-izlazne sabirnice (*PCI* i/ili *PCIe*), *USB* kontroler, kontroler sabirnice masovne memorije (*PATA* i/ili *SATA*), kontroler *floppy* diska, a ostvaruje vezu sa *BIOS* čipom, kao i sa tastaturom i mišem.

Pored prethodnog, vezni čipovi mogu u sebi imati i neke uređaje. Tako, severni vezni čip u sebi može sadržati integrisani grafički adapter koji nema sopstvenu memoriju, nego koristi deo radne memorije. Južni vezni čip u sebi može sadržati, na primer, integrisani zvučni adapter, integrisani mrežni adapter i/ili integrisani modem. Svi ovi uređaji su interno spojeni na odgovarajuće sabirnice.

Počevši od familije procesora *Athlon64* iz 2003. godine, procesori kompanije *AMD* su obuhvatili veći deo funkcionalnosti severnog veznog čipa (prvenstveno komunikacija sa radnom memorijom). Zahvaljujući tome, vezni čip ove i kasnijih familija *AMD* procesora ima samo jedan deo. Memorijski kontroler je ugrađen u procesor i u *Intel*-ovoj *Core i7* arhitekturi, i novijoj *Sandy Bridge*.

BIOS ČIP

BIOS (*Basic Input/Output System*) čip je *ROM* (*Flash*) memorija koja sadrži program. Rad računara započinje izvršavanjem ovog programa. To se desi kada se računar uključi. Osnovna funkcija ovog programa je da omogući punjenje operativnog sistema sa nekog od uređaja masovne memorije u radnu memoriju i njegovo pokretanje. Pored toga, *BIOS* omogućuje podešavanje raznih parametara matične ploče posredstvom svog *Setup* programa. Podešavanja utiču na rad raznih delova računara, omogućuju izbor uređaja sa koga će se pokrenuti operativni sistem, utiču na postupak isključivanja računara i omogućuju zadavanje šifre za pokretanje računara, kao i vraćanje parametara na inicijalno stanje. Pored ovog osnovnog *BIOS*-a, razni uređaji (grafički adapteri, na primer) mogu imati i svoj posebni *BIOS*, koji se obično nadograđuje na postojeći. Parametri koji se podešavaju pomoću *BIOS*-a se čuvaju u posebnoj *RAM* memoriji koja se napaja iz baterije sa matične ploče.

Već duže vreme postoje napori da se arhitektura *BIOS*-a, zacrtana osamdesetih godina prošlog veka, unapredi. Tako, danas je na raspolaganju još nekoliko rešenja: *EFI* (odnosno, *UEFI - Unified Extensible Firmware Interface*), *LinuxBIOS* i *Open Firmware* (i njegove implementacije, kao što su *OpenBOOT*,

OpenBIOS ili *Coreboot*). *EFI* je danas zastupljen u *Intel*-baziranim *Apple Macintosh* računarima i pojedinim serverima. *Open Firmware* ima svoju primenu u *SUN*-ovim i *IBM*-ovim serverima i *PowerPC*-baziranim *Apple Macintosh* računarima. *LinuxBIOS* se koristi u *OLPC* (*One Laptop Per Child*) projektu.

SABIRNICE

Danas najčešće korišćena ulazno–izlazna sabirnica je *PCI* (*Peripheral Component Interconnect*) sabirnica. Ova sabirnica ostvaruje paralelnu vezu *PCI* kontrolera i odgovarajućih *PCI* kartica. Postoji 32-bitna ili 64-bitna verzija *PCI* sabirnice, pri čemu je 32-bitna u mnogo široj upotrebi. *PCI* sabirnica radi na 33MHz (poslednja revizija je uvela i brzinu od 66MHz). Ona koristi konektore od 18 za 32-bitne, odnosno 188 pinova za 64-bitne kartice. *PCI* kontroler po uključenju računara preuzima od *PCI* kartica njihove zahteve za hardverskim resursima (memorijski i ulazno-izlazni adresni prostor, prekidi), i na osnovu toga svakoj kartici automatski dodeljuje tražene resurse (*plug and play*). Ovu sabirnicu najčešće koriste zvučne i mrežne katrice, kontroleri diskova (dodatni *ATA* kontroler, ili *SCSI* kontroler, na primer), *USB* i *FireWire* kontroleri, starije grafičke kartice, itd.

Većina grafičkih kartica, proizvednih u periodu od 1998. do 2005. godine, koristi *AGP* (*Accelerated Graphics Port*) sabirnicu. Ova sabirnica je uvedena pošto se pokazalo da *PCI* sabirnica ne može podržati protok podataka izazvan pojavom većih rezolucija grafičkih adaptera i uvođenjem 3D funkcija. *AGP* sabirnica radi na 66MHz i koristi konektore od 18 pinova. Poslednja revizija ove sabirnice omogućava prenos 8 podataka u okviru jednog ciklusa (u oznaci *AGPx8*).

PCI i naročito *AGP* sabirnice danas sve više zamenjuje novija sabirnica, nazvana *PCIe* (*PCI Express*). Za razliku od *PCI* i *AGP* sabirnica, koje koriste paralelnu vezu ka periferijskim uređajima, *PCIe* koristi serijsku vezu. Pored toga, ona omogućuje povezivanje i odvezivanje uređaja u toku rada računara (*hot-plug*). Veza sa uređajima kod *PCIe* sabirnice može biti jednostruka (*PCIe x1*), ili višestruka (*PCIe x2, x4, x8, x16, x32*), u zavisnosti od toga koliko serijskih kanala podržava uređaj. U skladu sa tim, postoje i odgovarajući konektori (x1, x2, x4, x8, x16, x32), pri čemu se uređaji uvek mogu zakačiti za svoj ili veći konektor. Tako, na primer, grafičke kartice koriste uglavnom x16 konektore, dok mrežne kartice koriste x1 konektore. Zbog velikog broja uređaja baziranih na *PCI* sabirnici, kao i zbog toga što za *PCIe* sabirnicu još uvek ne postoji šira proizvodnja za neke kategorije uređaja (zvučne kartice, na primer), današnje matične ploče na sebi obično imaju i *PCI* i *PCIe* sabirnicu. *PCIe* sabirnica je trenutno najkorišćenija kod grafičkih adaptera (koji su ranije koristili *AGP* sabirnicu), kod dodatnih kontrolera hard diskova i kod mrežnih krtica. Očekuje se da u narednom periodu i ostali uređaji pređu na ovu sabirnicu. Početkom 2007. godine je usvojena i specifikacija *PCIe 2.0* sabirnice, kao i specifikacija *External PCI Express* koja omogućava

povezivanje eksternih uređaja preko *PCIe* magistrale. Finalna verzija *PCIe 3.0* specifikacije se očekuje u toku 2009/2010.

ATA (*Advanced Technology Attachment*) sabirnica je standardna sabirnica za povezivanje uređaja masovne memorije (hard diskovi, *CD* i *DVD* uređaji) na računar. Ova sabirnica se naziva još i *ATAPI* (*Advanced Technology Attachment Packet Interface*) i *IDE* (*Integrated Drive Electronics*). Ona ostvaruje paralelnu vezu ka uređajima i to putem 40-žilnog, odnosno 80-žilnog kabla (Slika 16.2.2).

Slika 16.2.2 *ATA* konektor

SATA (*Serial ATA*) sabirnica je danas praktično zamenila *ATA* sabirnicu. Ona ostvaruje serijsku vezu ka uređajima i koristi 7-žilni kabl za povezivanje (Slika 16.2.3). Nakon uvođenja *SATA* sabirnice, standardna *ATA* sabirnica je preimenovana u *PATA* (*Parallel ATA*). I *SATA* sabirnica podržava povezivanje i odvezivanje uređaja tokom rada računara. Danas se već uveliko prave hard diskovi za ovu sabirnicu, a postoji i veći izbor optičkih uređaja. Otuda se na novim matičnim pločama nalaze sve ređe nalazi *PATA* sabirnica, pošto se njena uloga uglavnom svela na povezivanje *CD* i *DVD* uređaja sa *PATA* sabirnicom. Trenutno aktuelna verzija sabirnice je *SATA/300* (ili, češće pominjana kao *SATA II*). Sredinom 2004. godine je ustanovljena i *eSATA* (*External SATA*) sabirnica, namenjena za povezivanje spoljnih uređaja. 2009 godine je predstavljena *SATA 3.0* sabirnica koja omogućava brzine prenosa do 6Gb/s.

Slika 16.2.3 *SATA* konektori

SCSI (*Small Computer System Interface*) sabirnica se danas uglavnom koristi za brze diskove kod serverskih računara. Zasnovana je na 8-bitnoj ili 16-bitnoj paralelnoj vezi sa uređajima i omogućava povezivanje do 8 SCSI uređaja na jedan konektor, tako što se uređaji povezuju jedan na drugi (*SCSI chain*). Da bi se *SCSI* uređaji koristili, potreban je i *SCSI* kontroler, koji se na računar povezuje preko *PCI* sabirnice. Postoji više vrsta *SCSI* konektora koji imaju od 50 do 80 pinova, sa

različitim rasporedima (Slika 16.2.4). 2003. godine je ustanovljena i *SAS* (*Serial Attached SCSI*) sabirnica, zasnovana na serijskoj vezi ka uređajima, koja omogućava povezivanje kako *SAS* diskova, tako i *SATA* diskova.

Slika 16.2.4 *SCSI* konektori

Još 90-tih godina prošlog veka se ukazala potreba da se relativno spori spoljni uređaji povežu za računar. Za njih je bilo neekonomično da koriste komplikovanije sabirnice kao što je *PCI*. To je dovelo do definisanja *USB* (*Universal Serial Bus*) sabirnice, koja je zasnovana na serijskoj vezi sa uređajima. Ova sabirnica omogućuje povezivanje i odvezivanje uređaja u toku rada računara. U uređaje koji koriste ovu sabirnicu spadaju tastature, miševi, štampači, skeneri, *FLASH* diskovi, *web* kamere, fotoaparati, itd. Poslednja revizija ove sabirnice, *USB 2.0*, je donela značajno povećanje protoka, tako da se danas ova sabirnica koristi i za povezivanje uređaja masovne memorije, kao što su hard diskovi i *CD* i *DVD* uređaji. Konektori postoje u tri varijante, standardni *USB A* i *B* konektori i *mini-USB* konektori, namenjeni malim uređajima, kao što su PDA uređaji, fotoaparati ili mobilni telefoni (Slika 16.2.5). *USB* kontroler se u računaru obično nalazi ugrađen u vezne čipove, a može se naći i kao poseban kontroler povezan preko *PCI* sabirnice. 2009 godine je predstavljena *USB 3.0* sabirnica koja omogućava brzinu do 5Gb/s.

Slika 16.2.5 *USB* konektori

FireWire (ili *IEEE 1394*) sabirnica je prvobitno bila namenjena za povezivanje brzih spoljnih uređaja na računar, sa mogućnošću povezivanja i odvezivanja uređaja tokom rada računara. Veza sa uređajima je serijska, a koriste se dva tipa konektora, sa 4 i sa 6 pinova (Slika 16.2.6). Ova sabirnica se danas najviše koristi za povezivanje audio/video opreme (video kamere, kamkoderi), dok je ostale njene primene preuzela *USB 2.0* sabirnica. *FireWire* kontroler se u računaru obično nalazi kao poseban kontroler povezan preko *PCI* sabirnice, a postoje i rešenja gde se on nalazi na matičnoj ploči.

Slika 16.2.6 *FireWire* konektori

Do uvođenja *USB* sabirnice i masovnije pojave *USB* miševa i tastatura, za povezivanje ovih uređaja korišćeni su 6-pinski *PS/2* priključci i posebna serijska veza sa veznim čipovima.

16.3. RADNA MEMORIJA

U današnjim personalnim računarima koristi se radna memorija u obliku *DIMM* (*Dual In-Line Memory Module*) modula. Kod njih se memorijski čipovi nalaze sa obe strane štampane ploče. Najviše su u upotrebi *DDR* (*Double Data Rate*) i *DDR2 SDRAM* (*Synchronous DRAM*) moduli (Slika 16.3.1). Oni omogućuju da se u okviru jednog ciklusa obave dve operacije za dve različite memorijske lokacije. Time se udvostručuje brzina *SDRAM* memorije. Tako, u računaru sa memorijskom sabirnicom na 100 MHz, *DDR* memorija efektivno radi na 200 MHz. Oznake čipova ovih memorija sadrže frekvenciju rada, pa tako *DDR* čipovi koje rade na 200 MHz imaju oznaku DDR-200. Odgovarajući memorijski moduli sastavljeni od ovih čipova imaju oznaku koja sadrži maksimalnu brzinu protoka, pa tako memorijski modul sastavljen od DDR-200 čipova ima oznaku PC1600, pošto je njegova maksimalna brzina protoka 1600 Mbit/s. *DDR2* memorija je poboljšana varijanta *DDR* memorije po pitanju frekvencije, protoka i potrošnje. Nomenklatura je identična onoj korištenoj kod *DDR* memorijskih modula. *DDR DIMM* moduli sadrže 184 pina pomoću kojih se povezuju sa matičnom pločom, dok *DDR2 DIMM* moduli imaju 80 pinova. Kako bi se izbeglo da se u matičnu ploču stavi pogrešan memorijski modul, svaka vrsta modula ima jedan ili više zareza između pinova, koji se moraju poklapati sa ispupčenjima u ležištu memorijskog modula. Laptop računari, zbog svojih malih dimenzija, koriste posebne *SO-DIMM* (*Small Outline DIMM*) memorijske module, koji su manji po dimenzijama od standardnih *DIMM* modula. Sredinom 2007. godine je uvedena i *DDR3* memorija koja uvodi dalje povećanje protoka i smanjenje potrošnje.

Slika 16.3.1 *DDR* memorijski modul

16.4. GRAFIČKI ADAPTER

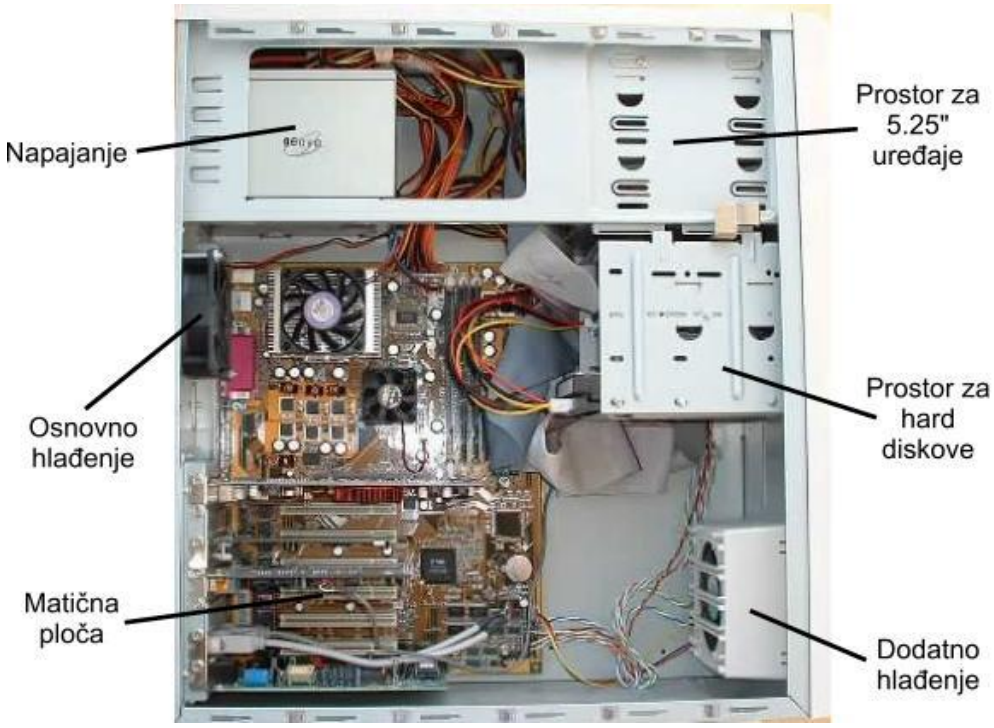
Grafički adapter omogućava prikaz slike na monitoru ili nekom drugom izlaznom uređaju (projektor, TV). Grafički adapteri, proizvedeni u poslednjih godinu dana, gotovo isključivo koriste *PCIe* sabirnicu, najčešće *PCIe x16*, dok je ranije u širokoj upotrebi bila *AGP* sabirnica. Dva najveća proizvođača grafičkih adaptera su *NVIDIA* i *AMD* (*ATI*), dok ostali zauzimaju veoma mali udeo na tržištu (*VIA/S3*, *Matrox*). Grafički adapteri na sebi imaju sopstveni procesor kao i sopstvenu radnu memoriju (koriste se kako *DDR2*/*DDR3* memorijski čipovi, tako i posebni *GDDR* memorijski čipovi). Pored posebnih grafičkih adaptera, na pojedinim matičnim pločama (naročito u kategoriji prenosnih računara) su prisutni i integrisani grafički adapteri (u okviru veznih čipova), koji kao radnu memoriju koriste deo radne memorije računara (u ovoj kategoriji adaptera kao veoma zastupljen proizvođač se nalazi i *Intel*). Brzi razvoj grafičkih adaptera za personalne računare je počeo devedesetih godina prošlog veka kada su se pojavili prvi grafički adapteri koji su omogućavali ubrzavanje nekih proračuna vezanih za prikaz trodimenzionalnih objekata na ekranu. Početkom 2010 se pojavio i prvi *x86* procesor sa integrisanim grafičkim adapterom – *Intel Core i5 661*. Krajem 2010. godine *Intel* je predstavio novu arhitekturu *Sandy Bridge* koja takođe sadrži grafičko jezgro unutar procesora, dok je *AMD* uradio sličnu stvar sa *AMD Fusion* arhitekturom početkom 2011.

Današnji moderni grafički adapteri poseduju veći broj jezgara (više desetina/stotina), kao i do 2GB radne memorije i optimizovani su za paralelno izvršavanje velikog broja proračuna. Upravo ova karakteristika je dovela do njihove upotrebe za mnogo širu kategoriju numeričkih proračuna (*GPGPU - General Purpose GPU*). Pokazalo se da, kada su u pitanju intenzivni numerički proračuni, procesori u grafičkim adapterima mogu doneti znatna ubrzanja u odnosu na izvršavanje istog proračuna na procesoru računara. *NVIDIA* je za svoje grafičke adaptere početkom 2007. ponudila *CUDA* (*Compute Unified Device Architecture*) platformu koja omogućava izvršavanje *C*-ovskih programa na grafičkom adapteru. Pošto *CUDA* platforma nije u potpunosti otvorena, razvijena je i *OpenCL* (*Open*

Computing Language) platforma koja omogućava transparentno iskorišćavanje kako višejezgarnih centralnih procesora, tako i procesora u grafičkom adapteru.

16.5. KUĆIŠTE, NAPAJANJE I SISTEM ZA HLAĐENJE

Kućišta za PC računare se dele na položena (*desktop*) i uspravna (*tower*) kućušta. Danas se najviše koriste uspravna kućišta (Slika 16.5.1). Ona se dele na mala, srednja i velika (*MiniTower*, *MidiTower* i *BigTower*). Veličina kućišta zavisi od broja potrebnih uređaja. Tako, za serverske računare se koriste velika kućišta, dok se za kućne i kancelarijske primene uglavnom koriste srednja i mala kućišta.



Slika 16.5.1 Primer uspravnog kućišta

Kućište obično dolazi sa ugrađenim napajanjem i osnovnim sistemom za hlađenje. Napajanje treba da odgovara maksimalnoj ukupnoj potrošnji svih organizacionih komponenti računara. *ATX* napajanja omogućavaju softversku kontrolu uključivanja i isključivanja računara. Osnovni sistem za hlađenje obično nije u stanju da održi dovoljno nisku temperaturu u kućištu tokom rada računara, pa posebno hlađenje obično imaju procesor, vezni čipovi i grafički adapter. Posebno hlađenje je često poželjno i za hard diskove i memorijske module. Da bi se proizvedena toplota izbacila iz kućišta, neophodno je da se na njemu nalazi dva ili

više ventilatora, kako za ubacivanje svežeg vazduha, tako i za izbacivanje toplog vazduha. Novija rešenja hlađenja komponenti su zasnovana na hlađenju tečnostima, čime se dobija dosta tiši rad računara.

LITERATURA

U pisanju ove knjige korišćeni su sledeći izvori:

***TANENBAUM, A.S, AUSTIN, T.: Structured Computer Organization**, sixth edition, Upper Saddle River, NJ: Pearson/Prentice Hall, 2012.*

***HENNESSY, J.L., PATTERSON D.A.: Computer Architecture A Quantitative Approach**, fifth edition, San Francisco, CA: Morgan Kaufman, 20011.*

***PATTERSON D.A., HENNESSY, J.L.: Computer Organization and Design, The Hardware / Software Interface**, revised fourth edition, San Francisco, CA: Morgan Kaufman, 20011.*

Prethodno pomenute knjige omogućuju potpunije upoznavanje materije vezane za arhitekturu računara. One sadrže i iscrpan pregled literature iz oblasti arhitekture računara.

INDEKS SLIKA

Slika 1.3.1 Tablica sabiranja za binarni brojni sistem 5
Slika 1.3.2 Električno kolo - ekvivalent logičke funkcije i 6
Slika 1.3.3 Električno kolo - ekvivalent logičke funkcije ili 6
Slika 1.3.4 Električno kolo - ekvivalent logičke funkcije negirano i 7
Slika 2.1.1 Cifre dekadnog brojnog sistema i njihovi komplementi 10
Slika 2.1.2 Primeri važenja relacije < 13
Slika 2.3.1 Konvertovanje razlomljenog broja 0.2_{10} 17
Slika 2.4.1 Izlasci van opsega kod sabiranja označenih celih brojeva u 20
Slika 2.4.2 Izlasci van opsega kod oduzimanja označenih celih brojeva u komplement 2 predstavi 21
Slika 2.4.3 Prikaz uslova važenja relacija za neoznačene brojeve 23
Slika 2.4.4 Prikaz uslova važenja relacija za označene brojeve 23
Slika 2.4.5 Pregled karakterističnih slučajeva važenja relacije < 23
Slika 3.2.1 Tabela sabiranja parova bita 29
Slika 3.2.2 Tabela oduzimanja parova bita 30
Slika 3.2.3 Pregled uslovnih upravljačkih naredbi 34
Slika 3.6.1 Zavisnost broja naredbi od broja poziva asemblerskog ekvivalenta potprograma **`nzd`** 50
Slika 3.6.2 Uloga makro pretprocesora 50
Slika 3.7.1 Izgled steka 56
Slika 3.7.2 Izgled frejma 58
Slika 3.7.3 Primer frejma 59
Slika 4.1.1 Raspored bita u memorijskoj lokaciji 61
Slika 4.1.2 Principijelni izgled memorijske lokacije 61
Slika 4.1.3 Pojednostavljeni principijelni izgled memorijske lokacije 62
Slika 4.1.4 Pojednostavljeni principijelni izgled memorije 62
Slika 4.1.5 Principijelni izgled memorije sa 4 lokacije 64
Slika 4.2.1 Izgled mašinskog formata naredbe 66
Slika 4.3.1 Organizacija procesora KONCEPT 74
Slika 4.3.2 Organizacija aritmetičko-logičke jedinice 75
Slika 4.4.1 Opis prelazaka između mikro-programa 84
Slika 4.5.1 Vremenski dijagram periodične izmene vrednosti promenljive T 85
Slika 4.5.2 Vremenski dijagram periodične izmene vrednosti promenljivih R i T 86
Slika 4.5.3 Organizacija upravljačke jedinice 87
Slika 4.7.1 Upravljanje prekidačima P2, P3 i P4 90
Slika 4.7.2 Upravljanje ulaznim prekidačima registara opšte namene 91
Slika 4.7.3 Upravljanje izlaznim prekidačima registara opšte namene 92
Slika 4.7.4 Upravljanje prekidačima od P37 do P44 92
Slika 4.7.5 Funkcije koje opisuju dekodiranje naredbi 93
Slika 4.7.6 Upravljanje prekidačima od P45 do P52 93
Slika 4.8.1 Mašinski oblik inicijalnog i mikro-programa dobavljanja. 94
Slika 4.8.2 Mašinski oblik mikro-programa obavljanja 1. tipa naredbi 94

Slika 4.8.3 Mašinski oblik mikro-programa obavljanja 2. tipa naredbi 95
Slika 4.8.4 Mašinski oblik mikro-programa obavljanja 3. tipa naredbi 95
Slika 4.8.5 Mašinski oblik mikro-programa obavljanja 4. tipa naredbi 96
Slika 4.8.6 Mašinski oblik mikro-programa obavljanja 5. tipa naredbi 96
Slika 4.8.7 Mašinski oblik mikro-programa obavljanja 6. tipa naredbi 97
Slika 4.8.8 Mašinski oblik mikro-programa obavljanja 7. tipa naredbi 97
Slika 4.8.9 Mašinski oblik mikro-programa obavljanja 8. tipa naredbi 98
Slika 4.8.10 Mašinski oblik mikro-programa obavljanja 9. tipa naredbi 98
Slika 4.8.11 Mašinski oblik mikro-programa obavljanja 10. tipa naredbi 99
Slika 4.8.12 Mašinski oblik mikro-programa obavljanja 10. tipa naredbi 99
Slika 4.8.13 Mašinski oblik mikro-programa obavljanja 12. tipa naredbi 100
Slika 4.8.14 Mašinski oblik mikro-programa obavljanja 13. tipa naredbi 100
Slika 4.8.15 Mašinski oblik mikro-programa obavljanja 14. tipa naredbi 101
Slika 4.8.16 Mašinski oblik mikro-programa obavljanja 15. tipa naredbi 101
Slika 4.9.1 Vremenski dijagram rada procesora KONCEPT 102
Slika 5.1.1 Organizacija računara sastavljenog od procesora i memorije 104
Slika 5.1.2 Organizacija računara KONCEPT 104
Slika 5.1.3 Upravljačka tabla računara KONCEPT 105
Slika 5.2.1 Princip rada tastature 107
Slika 5.2.2 Organizacija kontrolera tastature 108
Slika 5.2.3 Princip rada ekrana (kod $znaka_i$ izaziva zatvaranje $prekidača_i$) 108
Slika 5.2.4 Organizacija kontrolera ekrana 110
Slika 5.2.5 Organizacija računara KONCEPT sa znakovnim ulazom i izlazom 110
Slika 5.2.6 Sedmobitna *ASCII* tabela (kodovi su dati heksadecimalno) 111
Slika 5.2.7 Slojevita struktura *BIOS*-a 116
Slika 5.3.1 Principijelni izgled magnetnog diska 117
Slika 5.3.2 Organizacija kontrolera diska 119
Slika 5.3.3 Slojevita struktura proširenog *BIOS*-a 120
Slika 5.3.4 Organizacija računara KONCEPT sa radnom i masovnom memorijom 121
Slika 5.4.1 Sve kodne reči sa tri bita 122
Slika 5.4.2 Ustrojstvo kodne reči sa četiri bita podataka i tri bita parnosti 122
Slika 5.5.1 Slojevita struktura operativnog sistema 125
Slika 5.6.1 Slojevi operativnog sistema 128
Slika 5.7.1 Organizacija upravljačke jedinice koja podržava prekide 133
Slika 5.7.2 Organizacija računara KONCEPT koji podržava prekide 135
Slika 5.7.3 Organizacija drajvera terminala 138
Slika 5.7.4 Organizacija drajvera diska 138
Slika 5.8.1 Organizacija računara KONCEPT zasnovanog na sabirnici 141
Slika 5.9.1 Preslikavanje logičkih adresnih prostora raznih procesa u fizički adresni prostor 144
Slika 5.9.2 Postupak pretvaranja logičke adrese u fizičku 145
Slika 6.2.1 Tabela naredbi 152
Slika 6.2.2 Vrednosti brojača lokacija 154
Slika 6.2.3 Tabela labela 154
Slika 6.2.4 Tabela objektne sekvence 155
Slika 6.3.1 Tabela makro imena i tabela makro tela 156

Slika 6.3.2 Tabela argumenata 157
Slika 6.4.1 Tabela objektne sekvence sa označenim apsolutnim adresama 158
Slika 6.4.2 Tabela relokacije 159
Slika 6.4.3 Tabela objektne sekvence sa relativnim adresama, koje su označene strelicama 160
Slika 6.4.4 Tabela objektne sekvence i tabele relokacije, nedefinisanih i ulaznih labela programa 161
Slika 6.4.5 Tabela objektne sekvence i tabele relokacije, nedefinisanih i ulaznih labela potprograma 162
Slika 6.4.6 Tabela objektnih sekvenci 163
Slika 6.4.7 Tabela spoljašnjih labela 163
Slika 6.4.8 Tabela izvršne sekvence 164
Slika 8.1.1 Hronologija važnih događaja za razvoj računara 171
Slika 8.1.2 Tipična organizacija računara prve generacije 172
Slika 9.1.1 Osnovna digitalna kola 174
Slika 9.1.2 Principijelna organizacija radne memorije, zasnovane na magnetnim jezgricama 175
Slika 9.1.3 Tipična organizacija računara druge generacije 177
Slika 9.1.4 Primeri preklapanja potprograma u radnoj memoriji 179
Slika 10.1.1 Dekoder 2x4 181
Slika 10.1.2 Tipična organizacija mini-računara 183
Slika 10.1.3 Organizacija ožičene upravljačke jedinice 184
Slika 10.1.4 Poređenje izvršavanja dve naredbe i jedne složenije ekvivalentne naredbe 185
Slika 10.1.5 Dva načina organizovanja bajta u reči 186
Slika 10.1.6 Položaj kontrolera virtuelne memorije u organizaciji računara 190
Slika 10.1.7 Tabela stranica (sve adrese su binarni brojevi) 191
Slika 10.1.8 Asocijativna memorija sa dve trobitne lokacije 192
Slika 10.1.9 Skrivena memorija sa jednom lokacijom 195
Slika 10.1.10 Položaj kontrolera skrivene memorije u organizaciji računara 195
Slika 10.1.11 Kombinacija virtuelne i skrivene memorije 196
Slika 10.2.1 Organizacija radne memorije, sastavljene od dvobajtnih "*little endian*" reči 198
Slika 10.2.2 Serijsko vezivanje aktivnih strana linijom odobrenja 205
Slika 10.2.3 Organizacija računara sa memorijskom i ulazno-izlaznom sabirnicom 206
Slika 10.2.4 Asocijativna memorija (od 4 lokacije) sa punom asocijativnošću 207
Slika 10.2.5 Asocijativna memorija (od 4 lokacije) sa dvostrukom asocijativnošću 208
Slika 10.2.6 Asocijativna memorija (od 4 lokacije) sa jednostrukom asocijativnošću 209
Slika 10.2.7 Pregled slučajeva čitanja i pisanja skrivene memorije 210
Slika 10.2.8 Pretvaranje virtuelne adrese u fizičku 211
Slika 10.2.9 Tabela stranica iz dva nivoa 213
Slika 11.1.1 Primer jedno i dvodimenzionalne organizacije radne memorije sa 16 lokacija 220
Slika 11.1.2 Tipična organizacija mini-računara četvrte generacije 222
Slika 11.1.3 Preklapajući način rada procesora sa protočnom strukturom 225
Slika 11.1.4 Prepletena radna memorija sa osam lokacija raspodeljenih u dva memorijska modula 229
Slika 11.2.1 Pretvaranje logičke adrese u fizičku kod segmentirane radne memorije 234

Slika 11.2.2 Pretvaranje logičke adrese u fizičku kod stranične segmentacije 235
Slika 11.2.3 Dvoulazna video memorija i video kontroler 238
Slika 11.2.4 Organizacija višeprocesorskog računara sa više lokalnih i globalnih sabirnica 240
Slika 11.2.5 Unakrsna matrica 241
Slika 11.2.6 Višeprocesorski računar sa jednom sabirnicom i skrivenim memorijama 241
Slika 12.1.1 Princip toka podataka 249
Slika 12.1.2 Sistolički računar 251
Slika 12.1.3 *SIMD* računar 252
Slika 12.1.4 Multiprocesor 253
Slika 12.1.5 Multiračunar 254
Slika 12.1.6 Multiračunar organizovan oko eternet lokalne mreže 254
Slika 12.1.7 Vezivanje registara i aritmetičko-logičke jedinice pomoću 3 sabirnice 256
Slika 12.2.1 Organizacija invertovane tabele stranica 263
Slika 12.2.2 Stanja prekidača sa 2 ulaza i 2 izlaza 264
Slika 12.2.3 Primer upotrebe prekidača sa 2 ulaza i 2 izlaza 265
Slika 12.2.4 Princip rada dvostepene prekidačke mreže 265
Slika 12.2.5 Primer upotrebe dvostepene prekidačke mreže 267
Slika 12.2.6 Organizacija *BBN Butterfly* multiprocesora 268
Slika 12.2.7 Primer dvodimenzionalne mreže 268
Slika 12.2.8 Primer četvorodimenzionalne hiperkocke 270
Slika 12.2.9 Primer debelog drveta za 16 čvorova 271
Slika 13.1.1 Organizacija radne stanice koja je zasnovana na *PENTIUM 4* procesoru 275
Slika 16.2.1 Pojednostavljeni izgled matične ploče 289
Slika 16.2.2 *ATA* konektor 292
Slika 16.2.3 *SATA* konektori 292
Slika 16.2.4 *SCSI* konektori 293
Slika 16.2.5 *USB* konektori 293
Slika 16.2.6 *FireWire* konektori 294
Slika 16.3.1 *DDR* memorijski modul 295
Slika 16.5.1 Primer uspravnog kućišta 296