Biblioteka
PRIRODNO - MATAMATIČKIH NAUKA

Prof. dr Milan Martinović
Prof. dr Predrag Stanišić
RAČUNARI I PRINCIPI PROGRAMIRANJA
Drugo izdanje

БИБЛИОТЕКА
ТЕХНИЧКОГ ФАКУЛТЕТА
ТИТОГРАД

Glavni urednik
Prof.dr Ilija Vujošević

Urednik edicije
Prof. dr Milojica Jaćimović

Recenzenti
Prof. dr Milo Tomašević
Prof. dr Milenko Mosurović

Likovni urednik
Marina Vešović

Izdavač
Univerzitet Crne Gore,
Cetinjska 2, Podgorica
www.ucg.ac.me

Tiraž
400

Štampa
Pobjeda a.d. – Podgorica

Podgorica 2009

Zabranjeno preštampavanje i fotokopiranje u cjelini ili u djelovima.
Sva prava zadržava izdavač i autor(i)

Инв. бр. 29989

Dr Milan Martinović Dr Predrag Stanišić

RAČUNARI I PRINCIPI PROGRAMIRANJA

II izdanje

Podgorica, 2009.

FOTOKOPIRNICA
STUDENT
ul. Narodnih Heroja br. 2
preko puta KOZMETIK MARKETA
tel.: 069 519 129

Biblioteka
PRIRODNO - MATAMATIČKIH NAUKA

Prof. dr Milan Martinović
Prof. dr Predrag Stanišić
RAČUNARI I PRINCIPI PROGRAMIRANJA
Drugo izdanje

БИБЛИОТЕКА
ТЕХНИЧКОГ ФАКУЛТЕТА
ТИТОГРАД

Glavni urednik
Prof.dr Ilija Vujošević

Urednik edicije
Prof. dr Milojica Jaćimović

Recenzenti
Prof. dr Milo Tomašević
Prof. dr Milenko Mosurović

Likovni urednik
Marina Vešović

Izdavač
Univerzitet Crne Gore,
Cetinjska 2, Podgorica
www.ucg.ac.me

Tiraž
400

Štampa
Pobjeda a.d. – Podgorica

Podgorica 2009

Zabranjeno preštampavanje i fotokopiranje u cjelini ili u djelovima.
Sva prava zadržava izdavač i autor(i)

29989

Dr Milan Martinović Dr Predrag Stanišić

RAČUNARI I PRINCIPI PROGRAMIRANJA

II izdanje

Podgorica, 2009.

Dr Milan Martinović Dr Predrag Stanišić

RAČUNARI I PRINCIPI PROGRAMIRANJA

II izdanje

Sadržaj

Predgovor XI

Predgovor I izdanju XIII

1 Osnovni pojmovi 1
- 1.1 Informatika . . . 1
- 1.2 Informacioni sistem . . . 2
- 1.3 Teorija informacija . . . 4
  - 1.3.1 Količina informacije. Bit . . . 5
- 1.4 Obrada podataka i algoritam . . . 7
- 1.5 Intuitivni i formalni pojam algoritma . . . 9
- 1.6 Dijagrami toka . . . 13
- 1.7 Digitalni i analogni sistem za obradu podataka . . . 15
- 1.8 Ideja o hardveru i softveru . . . 16
- 1.9 Šema računara . . . 18
- 1.10 Razvoj računarske tehnike . . . 20
- 1.11 Zadaci za vježbu . . . 23

2 Brojni sistemi 27
- 2.1 Pozicioni i nepozicioni brojni sistemi . . . 27
  - 2.1.1 Izbor idealne osnove . . . 31
- 2.2 Konverzija broja iz jednog sistema u drugi . . . 33
  - 2.2.1 Konverzija kod kompatibilnih sistema . . . 36
- 2.3 Aritmetičke operacije sa binarnim brojevima . . . 39
- 2.4 Zadaci za vježbu . . . 41

3 Predstavljanje podataka u računaru 43
- 3.1 Predstavljanje neoznačenih cijelih brojeva . . . 44

3.2 Predstavljanje označenih cijelih brojeva 44
3.2.1 Predznak-Veličina predstavljanje 44
3.2.2 Prvi (nepotpuni) komplement 46
3.2.3 Drugi (potpuni) komplement 50
3.2.4 Konverzija pri povećanju mjesta za smještanje broja . . 54
3.2.5 Operacija pomjeranja (shift operacija) 55
3.2.6 BCD kodovi 56
3.3 Predstavljanje racionalnih brojeva 57
3.3.1 Nepokretni zarez 57
3.3.2 Pokretni zarez 58
3.4 Predstavljanje znakova 62
3.4.1 ASCII kodovi 62
3.4.2 Konverzija između ASCII i BCD 63
3.4.3 Ostali kodovi 63
3.5 Matematički pojam kodiranja 64
3.5.1 Azbučna kodiranja 65
3.5.2 Primjer primjene prefiksnog azbučnog kodiranja 67
3.6 Zadaci za vježbu 68

4 Algebra logike **71**
4.1 Pojam Bulove funkcije 71
4.2 Primjeri Bulovih funkcija 73
4.2.1 Konstante 73
4.2.2 Identičko preslikavanje 74
4.2.3 Negacija 75
4.2.4 Konjunkcija 75
4.2.5 Disjunkcija 76
4.2.6 Implikacija 76
4.2.7 Ekskluzivna disjunkcija 76
4.2.8 Šeferova funkcija 77
4.3 Pojam stvarne i fiktivne promjenljive 77
4.4 Formule. Realizacija funkcija formulama 79
4.5 Ekvivalentnost formula i osobine elementarnih funkcija 83
4.6 Princip dualnosti 85
4.7 Savršena disjunktivna normalna forma 87
4.8 Zadaci za vježbu 91
4.9 Rješenja zadataka za vježbu 94

5 Kompletnost i zatvorenost **101**
5.1 Kompletnost 101
5.2 Zatvorenje 104
5.3 Primjeri zatvorenih klasa 105
5.4 Teorema o kompletnosti 111
5.5 Zadaci za vježbu 114
5.6 Rješenja zadataka za vježbu 116

6 Pojam digitalnog kola **123**
6.1 Osnovni pojmovi 123
6.2 Elementi digitalnih integrisanih kola 125
6.2.1 Logički element 125
6.2.2 Idealni logički element 126
6.2.3 Realni logički element 127
6.3 Tehnologija digitalnih kola 134
6.3.1 Relej 134
6.3.2 Vakuumska cijev 134
6.3.3 Tranzistor 135
6.3.4 Realizacija logičkih elemenata 137
6.4 Kombinacione i sekvencijalne mreže 139
6.5 Zadaci za vježbu 141

7 Primjeri digitalnih mreža **143**
7.1 Sabirači 144
7.1.1 Sabirač 144
7.1.2 Potpuni sabirač 144
7.1.3 Paralelni sabirač 147
7.2 Dekoder 148
7.3 Multiplekser 153
7.4 Trostatički baferi 158
7.5 Magistrale digitalnih signala 161
7.6 Pojam bistabilnog kola 164
7.7 SR leč kolo 166
7.8 D leč kolo 170
7.9 Sinhroni flipflopovi SR i D 172
7.10 Flipflopovi JK i T 175
7.11 Rezime o flipflopovima 177
7.12 Stacionarni registri 178

7.13 Pomjerački registri 181
7.14 Serijski sabirač 186
7.15 Serijski i paralelni prenos digitalnih informacija 188
7.16 Brojači 191
7.17 Zadaci za vježbu 194

8 Komponente računarskog sistema **199**
8.1 Aritmetičko–logička jedinica 199
8.2 Memorija 203
8.3 RAM memorije 204
8.3.1 Statičke poluprovodničke memorije 204
8.3.2 Dinamičke poluprovodničke memorije 206
8.3.3 Primjena RAM memorija 207
8.3.4 Organizacija statičke memorije 207
8.4 ROM memorije 209
8.4.1 Pojam ROM memorije 209
8.4.2 ROM memorija sa fiksiranim sadržajem (mask-ROM) . 209
8.4.3 Programabilna ROM memorija (PROM) 211
8.4.4 Programabilna ROM memorija sa mogućnošću brisanja (EPROM) 213
8.4.5 EEPROM 214
8.5 Dvodimenziono dekodiranje adresa memorije 214
8.6 Organizacija centralnog procesora 216
8.6.1 Pojam mikro–operacije i registarskog prenosa 217
8.6.2 Mikro–operacije za prenos između registara 218
8.6.3 Aritmetičke mikro–operacije 220
8.6.4 Logičke mikro–operacije 221
8.6.5 Pomjeračke mikro–operacije 222
8.7 O registrima i izvođenju operacija 222
8.8 Organizacija ulazno–izlaznih jedinica 225
8.8.1 Spoljašnji memorijski uređaji 225
8.8.2 Tastatura 229
8.8.3 Štampači 229
8.8.4 Monitori 230
8.8.5 Prekid. DMA. Drajver 231
8.9 Pitanja i zadaci 232

9 Arhitektura osnovnog računara **235**
9.1 Oblik naredbe osnovnog računara 236
9.2 Kontrolna jedinica i vremenski ciklusi 239
9.2.1 Vremenski ciklusi 243
9.3 Naredbe koje se odnose na memoriju 246
9.4 Naredbe koje se odnose na registre 249
9.5 Ulazno-izlazna jedinica i mogućnost prekida 250
9.6 Ulazno-izlazne naredbe 254
9.7 O sistemu prekida osnovnog računara 256
9.8 Osnovni računar naspram PDP-8 257
9.9 Zadaci za vježbu 259

10 Softver za osnovni računar **261**
10.1 Računarski softver 261
10.2 Primjeri programa za osnovni računar 265
10.2.1 Primjeri programa za ulaz i izlaz 265
10.2.2 Primjer aritmetičkog programa 271
10.2.3 Primjer rada sa potprogramom 272
10.3 Loader 273
10.3.1 Apsolutni loader 273
10.3.2 Loader koji vrši relokaciju 275
10.3.3 Bootstrap looder 277
10.4 Asembler 277
10.4.1 Sistemski softver osnovnog računara 279
10.5 Zadaci za vježbu 280

11 Procesor Intel 8086 **283**
11.1 Uvod 283
11.2 Osnovna arhitektura 284
11.3 Memorija 285
11.4 Registri 285
11.4.1 Programski brojač 286
11.4.2 Status registar 286
11.4.3 Registri opšte namjene 287
11.4.4 Pointerski registri 288
11.4.5 Indeksni registri 288
11.4.6 Segmentni registri 288
11.5 Metode adresiranja 289

11.5.1 Neposredno adresiranje 289
11.5.2 Registarsko adresiranje 289
11.5.3 Registarsko indirektno adresiranje 290
11.5.4 Direktno adresiranje 290
11.5.5 Indeksirano adresiranje 290
11.5.6 Bazno adresiranje 291
11.5.7 Bazno indeksirano adresiranje 291
11.5.8 Bazno indeksirano relativno adresiranje 291
11.6 Instrukcije 292
11.6.1 Instrukcije za prenos podataka 292
11.6.2 Aritmetičke instrukcije 295
11.6.3 Logičke instrukcije i instrukcije pomjeranja i rotiranja . 298
11.6.4 Upravljačke instrukcije 300
11.6.5 Instrukcije za rukovanje nizovima 304
11.6.6 Instrukcije za rukovanje status registrom 305
11.6.7 Razne instrukcije 305
11.7 Upotreba programa debug.exe 306
11.7.1 Primjeri za debug.exe 310
11.8 Asemblerski jezik 8086 315
11.8.1 Format asemblerske linije 316
11.8.2 Segmenti 316
11.8.3 Konstante 318
11.8.4 Zauzimanje memorije za promjenljive 319
11.8.5 Uslovno asembliranje 320
11.8.6 Makroi 320
11.8.7 Potprogrami 322
11.9 Primjeri programa 323
11.9.1 Dodjela vrijednosti 323
11.9.2 Kopiranje vrijednosti 325
11.9.3 Asemblerski oblik IF-THEN-ELSE iskaza 327
11.9.4 Asemblerski oblik FOR iskaza 328
11.9.5 Prenos argumenata preko registara 330
11.9.6 Vraćanje rezultata potprograma preko registara 331
11.10Prekidne rutine 331
11.11Sistemski pozivi i obrada izuzetaka 332
11.11.1 Unutrašnji prekidi 332
11.11.2 Spoljašnji prekidi 333
11.11.3 BIOS servisi 334

11.11.4 DOS prekid 21h 336
11.11.5 Aplikacioni servisi 340
11.12Pisanje obrađivača prekida 342
11.13Pisanje servisa 343
11.14Sistem prekida procesora Intel 8086 344
11.15O mikroprocesoru Intel 80286 347
11.16Tridesetdvobitni mikroprocesori 348
11.17Zadaci za vježbu 351

12 Operativni sistem **355**
12.1 Funkcije operativnog sistema 356
12.2 Tipovi operativnog sistema 357
12.3 Istorija operativnih sistema 358
12.3.1 Prva generacija (1945-55) 358
12.3.2 Druga generacija (1955-65) 358
12.3.3 Treća generacija (1965-80) 359
12.3.4 Četvrta generacija (1980-) 359
12.4 Važniji operativni sistemi 360
12.4.1 CP/M 360
12.4.2 Operativni sistemi DOS i Windows 360
12.4.3 Unix i Linux 361
12.5 Startovanje operativnog sistema 363
12.6 Osnovne komponente operativnog sistema 364
12.6.1 Upravljanje procesima 365
12.6.2 Upravljanje memorijom 366
12.6.3 Upravljanje ulazom i izlazom 367
12.6.4 Upravljanje fajl sistemom 368
12.6.5 Programski interfejs 369
12.6.6 Korisnički interfejs 369
12.7 Pitanja i zadaci 370

13 Uvod u strukture podataka **373**
13.1 Lista 373
13.2 Stek i red 380
13.3 Zadaci za vježbu 386

14 Paradigme programiranja **389**
14.1 Proceduralno programiranje 390
14.2 Objektno-orijentisano programiranje 391
14.3 Funkcionalno programiranje 395
14.4 Logičko programiranje 397
14.5 Rule-based programiranje 399
14.6 Upitni jezici baza podataka 400
14.7 Vizuelno programiranje 402
14.8 Script based programiranje 403
14.9 Programiranje demonstracijom 403
14.10Zaključak 403
14.11Pitanja i zadaci 404

Literatura **405**

Predgovor

Ova knjiga predstavlja drugo izdanje istoimenog udžbenika izdatog od strane Prirodno-matematičkog fakulteta Univerziteta Crne Gore 2004. godine.

Od svog nastanka udžbenik je postao obavezna literatura za kurseve Računari i programiranje i Principi programiranja, na prvoj godini studijskih programa *Matematika*, *Matematika i računarske nauke*, *Računarske nauke* i *Računarstvo i informacione tehnologije* ovog fakulteta, a uvršten je u spisak literature u nekoliko drugih kurseva. Takođe, pokazano je određeno interesovanje za njega i sa drugih srodnih studijskih programa i fakulteta.

Tokom prethodnih pet godina kompletan tiraž je rasprodat, što je motivisalo autore da pristupe izradi novog, dopunjenog, izdanja. U ovom izdanju ispravljene su određene greške i nepreciznosti uočene u I izdanju. Tamo gdje se to autorima učinilo potrebnim, nastojali smo osavremeniti pojmove, pružiti jasnije formulacije i poboljšati izlaganje.

Zahvaljujući našim kolegama, studentima i nastavnicima sa Prirodno-matematičkog fakulteta, koji su nam konstantno pružali svoje vrlo korisne primjedbe, uvrstili smo u novo izdanje i značajan broj njihovih sugestija. Svima njima dugujemo iskrenu zahvalnost.

Kao i kod prvog izdanja, recenzenti prof. dr Milo Tomašević sa Elektrotehničkog fakulteta Univerziteta u Beogradu i prof. dr Milenko Mosurović sa Prirodno-matematičkog fakulteta Univerziteta Crne Gore bili su nam od posebne koristi.

Svjesni da je teško moguće napisati tekst ovog obima bez određenog broja grešaka raznog tipa, pozivamo sve čitaoce da nam pošalju svoje dragocjene sugestije.

U Podgorici, oktobar 2009. Autori

Predgovor I izdanju

Ova knjiga nastala je na osnovu predavanja koja autori drže na predmetima Računari i programiranje i Principi programiranja, na prvoj godini smjerova *B-Primijenjena matematika i računarske nauke* i *C-Računarske nauke*, Prirodno-matematičkog fakulteta u Podgorici.

Namijenjena je dakle, prvenstveno studentima koji izučavaju računarske nauke, mada, po našem mišljenju, ona može biti od koristi i studentim drugih fakulteta na kojima se izučavaju sadržaji bliski sadržaju knjige. Iako sadrži teme iz raznih oblasti, knjiga ne podrazumijeva značajnije prethodno znanje računarskih ili matematičkih disicplina.

Knjiga predstavlja univerzitetski udžbenik i slijedi program pomenutih kurseva. U *prvoj* glavi daje se pregled osnovnih pojmova informatike i računarstva, uz osvrt na istoriju razvoja računarske tehnike. *Druga* i *treća* glava obrađuju brojne sisteme i načine predstavljanja podataka u računaru. U *četvrtoj* obrađuju se glavni pojmovi algebre logike, dok se u *petoj* razmatraju pitanja kompletnosti i zatvorenosti sistema Bulovih funkcija. U *šestoj* glavi uvode se osnovni pojmovi digitalne elektronike, a *sedma* glava sadrži glavne primjere kombinacionih i sekvencijalnih digitalnih kola. *Osma* glava sadrži opis glavnih komponenti računarskog sistem. Želeći ipak da principi funkcionisanja računarskog sistema, kao i programiranja, budu što je moguće jasniji studentima, u *devetoj* glavi detaljno se izlaže arhitektura jednostavnog računara i to do nivoa mikro–operacije. U *desetoj* glavi dati su primjeri programa za osnovni računar, koji uključuju rad sa ulazom/izlazom, potprogramima i prekidima, uz osnovne pojmove o asembleru i loadovanju. Nakon izlaganja principa organizacije i pisanja programa za osnovni računar, u glavi *jedanaest* daje se sistematičan opis arhitekture i asemblerskog jezika procesora 8086, preteče popularne Inteleove 80x86 porodice procesora. Važno svojstvo vertikalne kompatibilnosti Intelovih procesora omogućuje korišćenje sadržaja ove glave, a naročito primjera programa, i na najsavremenijim PC

računarima. U glavi *dvanaest* obrađuju se osnovni koncepti modernih operativnih sistema. Glava *trinaest* daje uvod u neke od važnijih struktura podataka. Konačno, *četrnaesta* glava sadrži pregled glavnih paradigmi programiranja.

Na kraju svake glave nalazi se serija zadataka ili pitanja, koja treba da posluži za bolje razumijevanje i usvajanje gradiva. Autori su se trudili da teme iz prilično različitih oblasti izlože na način prihvatljiv za studente. Koliko su u tome uspjeli, naravno, na studentima je da sude.

Recenzenti ove knjige su prof. dr Milo Tomašević sa Elektrotehničkog fakulteta Univerziteta u Beogradu i docent dr Milenko Mosurović sa Prirodno-matematičkog fakulteta Univerziteta Crne Gore. Njihovi komentari rukopisa i savjeti bili su nam od velike koristi. Želimo da im ovom prilikom izrazimo zahvalnost.

Na kraju, svjesni da je u ovakvim tekstovima teško izbjeći pojavu određenog broja grešaka raznog tipa, želimo da istaknemo da ćemo biti zahvalni za sve eventualne sugestije u odnosu na sadržaj ove knjige.

U Podgorici, april 2004.g. Autori

Glava 1

Osnovni pojmovi

1.1 Informatika

Informatika je naučna disciplina, koja je nastala u drugoj polovina dvadesetoga vijeka. Samu riječ 'informatika' skovao je 1962. godine Filip Drejfis (Philippe Dreyfus), francuski naučnik, kombinujući dvije riječi koje najsažetije definišu predmet istraživanja ove nauke. To su riječi *information* (informacija) i *automatique* (automatski). Dakle, informatika je nauka koja se bavi automatskom obradom podataka, naročito obradom pomoću računara. Ovo se može smatrati definicijom informatike u užem smislu.

Pojam informatike ne mora se nužno vezati za razvoj računarskih tehnologija. Recimo, u širem smislu, informatika je nauka koja proučava zakonitosti u informaciono-dokumentacionoj djelatnosti.

Informatika se koristi raznim savremenim metodama. Njoj se često pristupa kroz opštu teoriju sistema, a koriste se i indukcija, dedukcija, strukturna sistem analiza, simulacioni modeli, regresiona i korelaciona analiza, statističke metode, linearno programiranje, modeli zaliha, teorija uzoraka, itd.

Možemo kazati da se cilj informatike sastoji u razvijanju i implementaciji metoda i sredstava pomoću kojih je moguće na optimalan način prikupljati, skladištiti, obrađivati distribuirati i tumačiti informacije.

Kao posebne grane informatike, tokom vremena izdvojile su se razne grane ove naučne discipline, kao što su:

- organizacija i arhitektura računarskih sistema;

računarima. U glavi *dvanaest* obrađuju se osnovni koncepti modernih operativnih sistema. Glava *trinaest* daje uvod u neke od važnijih struktura podataka. Konačno, *četrnaesta* glava sadrži pregled glavnih paradigmi programiranja.

Na kraju svake glave nalazi se serija zadataka ili pitanja, koja treba da posluži za bolje razumijevanje i usvajanje gradiva. Autori su se trudili da teme iz prilično različitih oblasti izlože na način prihvatljiv za studente. Koliko su u tome uspjeli, naravno, na studentima je da sude.

Recenzenti ove knjige su prof. dr Milo Tomašević sa Elektrotehničkog fakulteta Univerziteta u Beogradu i docent dr Milenko Mosurović sa Prirodno-matematičkog fakulteta Univerziteta Crne Gore. Njihovi komentari rukopisa i savjeti bili su nam od velike koristi. Želimo da im ovom prilikom izrazimo zahvalnost.

Na kraju, svjesni da je u ovakvim tekstovima teško izbjeći pojavu određenog broja grešaka raznog tipa, želimo da istaknemo da ćemo biti zahvalni za sve eventualne sugestije u odnosu na sadržaj ove knjige.

U Podgorici, april 2004.g. Autori

Glava 1

Osnovni pojmovi

1.1 Informatika

Informatika je naučna disciplina, koja je nastala u drugoj polovina dvadesetoga vijeka. Samu riječ 'informatika' skovao je 1962. godine Filip Drejfis (Philippe Dreyfus), francuski naučnik, kombinujući dvije riječi koje najsažetije definišu predmet istraživanja ove nauke. To su riječi *information* (informacija) i *automatique* (automatski). Dakle, informatika je nauka koja se bavi automatskom obradom podataka, naročito obradom pomoću računara. Ovo se može smatrati definicijom informatike u užem smislu.

Pojam informatike ne mora se nužno vezati za razvoj računarskih tehnologija. Recimo, u širem smislu, informatika je nauka koja proučava zakonitosti u informaciono-dokumentacionoj djelatnosti.

Informatika se koristi raznim savremenim metodama. Njoj se često pristupa kroz opštu teoriju sistema, a koriste se i indukcija, dedukcija, strukturna sistem analiza, simulacioni modeli, regresiona i korelaciona analiza, statističke metode, linearno programiranje, modeli zaliha, teorija uzoraka, itd.

Možemo kazati da se cilj informatike sastoji u razvijanju i implementaciji metoda i sredstava pomoću kojih je moguće na optimalan način prikupljati, skladištiti, obrađivati distribuirati i tumačiti informacije.

Kao posebne grane informatike, tokom vremena izdvojile su se razne grane ove naučne discipline, kao što su:

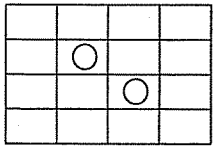
- organizacija i arhitektura računarskih sistema;

Jediničnu količinu informacije nazivamo *bitom.*

Ova razmatranja ukazuju da tražena funkcija f slika poluinterval (0,1] u R^+:

$$f : (0, 1] \to R^+.$$

Pretpostavimo da, umjesto jedne, naša mašina baca dvije loptice (slika 1.3).



Slika 1.3: Eksperiment sa dvije lopte

Bacanje jedne loptice nema nikakve veze sa bacanjem druge. Mašina za bacanje ne zna šta je bacila ranije, niti gdje je ta lopta pala. Tako, na primjer, obije loptice mogu pasti u istu kutiju. Zato kažemo da su to dva *nezavisna događaja.* Intuitivno smo svjesni da je količina informacija o tome gdje su pale dvije lopte jednaka zbiru količina informacija o tome gdje je pala prva a gdje druga lopta.

Možemo posmatrati ishod bacanja dvije loptice kao događaj. U tom prostoru događaja imamo $16 * 16 = 256$ ravnopravnih događaja. Vjerovatnoća svakog od njih je $\frac{1}{256}$, odnosno $\frac{1}{16} * \frac{1}{16}$. Ovo nam daje motivaciju da vjerovatnoću $p(x_1 \cap x_2)$, da će se desiti događaj koji je *presjek* dva nezavisna događaja, možemo definisati kao proizvod vjerovatnoća tih događaja: $p(x_1) * p(x_2)$. Dakle,

$$p(x_1 \cap x_2) = p(x_1) * p(x_2) = p_1 * p_2.$$

Ako imamo količinu informacije $f(p_1)$ i $f(p_2)$ za dva događaja x_1 i x_2, sa vjerovatnoćama p_1 i p_2, tada je prirodno definisati količinu informacije koju nosi presjek događaja kao zbir količina informacije koju nose pojedinačni događaji. Zato mora biti

$$f(p_1 * p_2) = f(p_1) + f(p_2).$$

Kada se na osnovu prethodnih zaključaka skicira grafik funkcije f, lako se vidi da elementarna funkcija koja ima sva gore navedena svojstva je $-\log_2 p$. Znači, za funkciju koja određuje količinu informacije koju nosi neki događaj, možemo uzeti funkciju:

$$f(p) = -\log_2 p.$$

Dalje, vezano za pojam *matematičkog očekivanja,* prirodno se definiše *srednja količina informacije koju nosi neki sistem*:

$$H = \sum_{i=1}^{n} p_i f(p_i).$$

Uvodeći na gornji način funkciju količine informacije, imamo da je

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i.$$

Ova formula naziva se *Šenonovom formulom za srednju količinu informacije.* Šenon je postavio temelje *teorije informacija.*

1.4 Obrada podataka i algoritam

Obrada podataka je djelatnost transformacije podataka u cilju dobijanja izlazne informacije. Postoje razne vrste obrade podataka: ručna (u kojoj se podaci obrađuju korišćenjem ljudskog intelekta), mehanizovana (podaci se takođe obrađuju korišćenjem ljudskog intelekta, ali se neke faze obrade olakšavaju korišćenjem nekog pomoćnog uređaja, recimo kalkulatora) i automatska (obrada se u potpunosti vrši računarom).

Algoritam je precizan propis o tome koje operacije, nad kojim podacima, i u kom redosljedu treba izvršiti da bi se dobio željeni rezultat. Ili, drugim riječima, algoritam je pravilo po kome se izvodi postupak obrade podataka.

Pojasnimo pojam "precizan propis" u ovom kontekstu. Algoritam mora da ima sljedeća svojstva.

- *Ulaz* – skup ulaznih vrijednosti koji mora biti zadat. Vrijednosti moraju biti zadate na jasan način, na primjer korišćenjem neke konačne *azbuke.*

БИБЛИОТЕКА

- *Izlaz* - jasan skup izlaznih vrijednosti koji mora biti dostupan na kraju algoritma. Zadaje se korišćenjem konačne izlazne azbuke.
- *Etapnost* - propis mora biti sastavljen etapno, korak po korak, jedna operacija za drugom.
- *Određenost* – u svakoj fazi algoritma mora se znati koja se operacija izvodi, nad kojim podacima, koji je rezultat i koja je operacija sljedeća.
- *Konačnost* – algoritam se mora završiti u konačnom vremenu.
- *Diskretnost* – algoritam mora biti predstavljen nizom elementarnih koraka.
- *Jednoznačnost* – ni u kojoj fazi njegovog ispunjavanja ne smije se zahtijevati donošenje slobodnih odluka. Sve što je potrebno za izvršavanje nalazi se u ulaznim podacima i samom pravilu.
- *Skladištenje međurezultata* – propis može zahtijevati konačno velik prostor za smještanje međurezultata.
- *Početak* – mora se označiti početak algoritma.
- *Kraj* – algoritam može imati više mogućih krajeva, moraju se jasno označiti.

Izvršilac algoritma mora biti u stanju da u potpunosti razumije sve elementarne korake algoritma i da ih izvrši. Izvršilac algoritma, zavisno od vrste obrade podataka, može biti čovjek ili mašina.

Važna karakteristika algoritma je i njegova širina primjene, tj. veličina klase problema koji se algoritmom rješavaju.

Algoritmi se mogu zapisivati na razne načine. Napišimo primjer algoritma koji ćemo opisati na svom jeziku.

Primjer. Definišimo algoritam kome ulaz čine dva broja X i Y a rezultat je $\frac{X+Y}{X-Y}$ ili informacija da dijeljenje nije moguće.

Na slici 1.4 opisan je jedan konkretni postupak, koji rješava postavljeni problem. On ima sva navedena svojstva algoritma.

Način na koji se opisuje algoritam zavisi od sposobnosti i znanja pisca, ali i od vrste operacija koje su dozvoljene u procesu obrade. Na primjer, ovdje

ULAZ - su dva broja, označimo ih sa X i Y

IZLAZ - je $\frac{X+Y}{X-Y}$ ili poruka da dijeljenje nije moguće.

POČETAK

Korak 1. IZRAČUNAJ X+Y I ZAPAMTI REZULTAT KAO PODATAK M.

Korak 2. IZRAČUNAJ X-Y I ZAPAMTI REZULTAT KAO PODATAK N.

Korak 3. AKO JE N=0 ONDA IDI NA Korak 4, INAČE IDI NA Korak 6

Korak 4. IZLAZ JE OBAVJEŠTENJE DA DIJELJENJE NIJE MOGUĆE

Korak 5. KRAJ

Korak 6. IZRAČUNAJ $\frac{M}{N}$ I ZAPAMTI REZULTAT KAO PODATAK D.

Korak 7. IZLAZ JE D.

Korak 8. KRAJ

Slika 1.4: Algoritam

pretpostavljamo da se operacije izvršavaju jedna za drugom u redosljedu pisanja, osim u Koraku 3, gdje imamo operaciju mijenjanja redosljeda. Takođe, pretpostavljamo da imamo operaciju sabiranja i skladištenja rezultata, operaciju dijeljenja, mogućnost poređenja dva broja, itd.

Ukoliko izvršilac algoritma nije u stanju da izvrši, recimo, operaciju oduzimanja, tada se ona mora definisati preko drugih operacija koje izvršilac razumije, na primjer – korišćenjem negacije i sabiranja.

1.5 Intuitivni i formalni pojam algoritma

Pojam algoritma smo uveli intuitivno. Postoje, međutim, razni matematički modeli algoritma. Ti modeli sadrže matematički precizan opis svih elemenata algoritma. Kada nešto definišemo matematičkim sredstvima, onda možemo matematikom izučavati njegova svojstva.

Jedan od najpoznatijih matematičkih modela algoritma je *Tjuringova mašina*, nazvana po tvorcu, engleskom matematičaru Alenu Tjuringu.

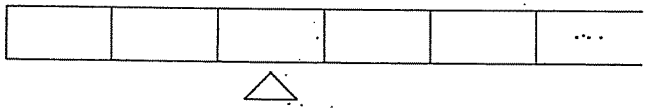
Mašina se sastoji od nekoliko elemenata (slika 1.5). Mašinom upravlja *izvršni uređaj*, koji može da se nalazi u jednom od konačno mnogo stanja

$q_0, q_1, \ldots, q_n$. Jedno stanje, obično q_0, je označeno kao *početno*, a jedno q_f kao *završno stanje*.

Mašina sadrži *traku* sastavljenu od polja, koja je prebrojive dužine (odnosno koja ima polja onoliko koliko ima prirodnih brojeva). Traka ima svoj *početak* sa lijeve strane, dok je sa desne beskonačna. Mašina posjeduje i mehanizam za pokretanje trake.

Mašina sadrži i *glavu*, koja služi za čitanje i pisanje određenog konačnog skupa simbola koji se naziva *radna azbuka*. Azbuka sadrži simbole $s_1, s_2, \ldots, s_y$, kao i simbol * za prazno. Azbuka je, dakle, skup $S = \{*, s_1, s_2, \ldots, s_y\}$. Ona može biti i vrlo prosta, na primjer - može da sadrži samo simbol | i znak za prazno *.

Glava se uvijek nalazi iznad jednog polja trake kojeg nazivamo *radnim*. Operaciju pisanja nekog simbola s_i označavamo sa s_i. Mehanizam za pokretanje trake može da pomjeri traku tako da radno polje postane neko od dva susjedna polja radnom polju, odnosno traka se može pomjerati "ulijevo" i "udesno" od glave. To su operacije L i D.



Slika 1.5: Tjuringova mašina

Postoji i operacija zaustavljanja t. Osim ovom operacijom, mašina se još zaustavlja u slučaju da se radno polje nalazi na početku trake, a izvede se operacija L.

Da rezimiramo, do sada smo definisali operacije

$$O = \{t, *, s_1, s_2, \ldots, s_y, L, D\}.$$

Mašina zna da izvršava operacije oblika "ako se nalaziš u stanju q_i a ispod glave se nalazi simbol s_p, tada upiši simbol s_r, izvrši neku od operacija i pređi u stanje q_j". Ovakva pravila su, u suštini, *funkcija prelaza*

$$\delta : Q \times S \to O \times Q.$$

Ovo se može zapisati nizom četvorki oblika

$$q_i s_p o_u q_j.$$

Mašina počinje sa radom u stanju q_0, a sve ono što se nalazi lijevo od radne pozicije do prvog znaka * smatra se ulazom. Ovo se naziva *početnom konfiguracijom*.

Nakon uvođenja ovih pojmova, Tjuringovu mašinu možemo definisati kao uređenu petorku:

$$TM = \{S, Q, q_0, q_f, \delta\}.$$

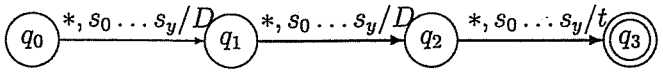
Tjuringovu mašinu čine azbuka, skup stanja, početno stanje, krajnje stanje i funkcija prelaza.

Kao primjer, na slici 1.6 tablicom je opisana Tjuringova mašina koja, nezavisno od početne pozicije, pomjera radno polje za dva mjesta udesno i onda se zaustavlja .

$q_0, *, D, q_1$
q_0, s_0, D, q_1
$\ldots$
q_0, s_y, D, q_1
$q_1, *, D, q_2$
q_1, s_0, D, q_2
$\ldots$
q_1, s_y, D, q_2
$q_2, *, t, q_3$
q_2, s_0, t, q_3
$\ldots$
$q_2, s_y, t, q_3.$

Slika 1.6: Primjer Tjuringove mašine

Tjuringova mašina može se, umjesto tablicom, preglednije opisati *Tjuringovim dijagramom*. To je dijagram koji sadrži grafičke elemente za stanja, kao i prelaze iz stanja u stanje, zavisno od simbola koji se nalazi na radnom polju. On, uz početnu konfiguraciju, potpuno opisuje Tjuringovu



Slika 1.7: Tjuringov dijagram

mašinu. Na slici 1.7 prikazan je dijagram za Tjuringovu mašinu sa slike 1.6.

Naravno, ovaj algoritam može se zapisati i na našem jeziku (slika 1.8).

ULAZ: Stanje trake i položaj glave
IZLAZ: Glava se nalazi dva mjesta udesno
Stanje q_0. Početak rada. Pročitaj znak ispod glave
Ako je to znak $*$ izvrši operaciju d i pređi u stanje q_1
Ako je to znak s_0 izvrši operaciju d i pređi u stanje q_1
...
Ako je to znak s_y izvrši operaciju d i pređi u stanje q_1
Stanje q_1. Pročitaj znak ispod glave
Ako je to znak $*$ izvrši operaciju d i pređi u stanje q_2
Ako je to znak s_0 izvrši operaciju d i pređi u stanje q_2
...
Ako je to znak s_y izvrši operaciju d i pređi u stanje q_2
Stanje q_2. Pročitaj znak ispod glave
Ako je to znak $*$ izvrši operaciju t i pređi u stanje q_3
Ako je to znak s_0 izvrši operaciju t i pređi u stanje q_3
...
Ako je to znak s_y izvrši operaciju t i pređi u stanje q_3
Stanje q_3. Kraj rada

Slika 1.8: Tjuringova mašina kao algoritam

Ovaj opis zadovoljava sve naše zahtjeve date pri definiciji pojma algoritma. Recimo, znamo šta je početak, a šta kraj. Svaki korak je potpuno jasno definisan i uvijek znamo koji je sljedeći, itd.

Intuitivno je jasno da sve za što se može napraviti Tjuringova mašina jeste algoritam (svaka Tjuringova mašina zadovoljava svojstva algoritma). Međutim, pitanje je da li za sve što intuitivno smatramo algoritmom možemo napraviti Tjuringovu mašinu koja radi isti posao.

Iako Tjuringova mašina na prvi pogled djeluje prilično primitivno, pomoću nje je moguće izvršiti sve što može izvršiti najsavremeniji računar!

Naime, prvo su definisani načini kako da zapisujemo podatke na traci, zatim su konstruisane mašine koje obavljaju jednostavne operacije, pa sve složenije i složenije. Nije pronađeno ništa što se intuitivno može smatrati algoritmom, a da se za to ne može napraviti odgovarajuća Tjuringova mašina!

Čerč je prvi izrazio tvrdnju, koja je kasnije nazvana *Čerčovom tezom*, da je intuitivni pojam algoritma ekvivalentan sa Tjuringovom mašinom. Kako intuitivni pojam algoritma nije formalni pojam, to nije moguće dokazati ovu tezu. Čerčova teza ostaje na snazi jer do sada niko nije smislio nešto što bi intuitivno smatrali algoritmom, a da se za to ne može napraviti Tjuringova mašina.

Postoje i drugi pokušaji formalizacije pojma algoritma. Takvi su, na primjer, parcijalno rekurzivne funkcije, normalni algoritmi Markova, grafičke šeme, ... Pokazano je da su svi oni međusobno ekvivalentni, što ide u prilog Čerčovoj tezi.

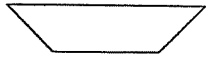
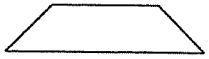
1.6 Dijagrami toka

Naročito je pogodno zapisivati algoritme grafički, pomoću takozvanih "dijagrama toka". To smo već vidjeli i kod Tjuringovih dijagrama. Takvi dijagrami vizuelno ističu pojedine vrste operacija koje se javljaju u obradi podataka.

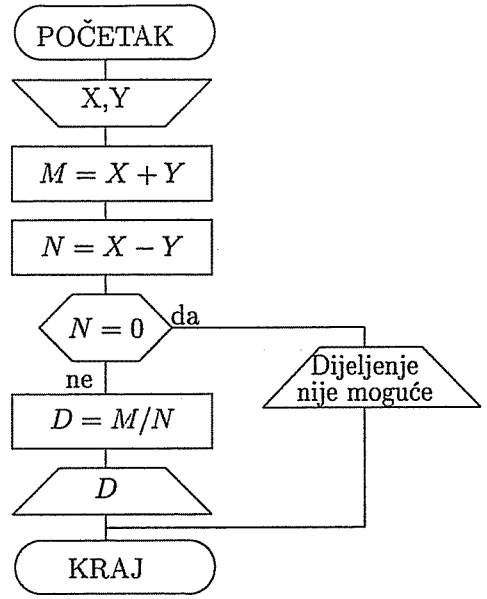
Oni sadrže grafičke elemente za prikazivanje operacija, simbole za označavanje početka i kraja, za označavanje ulaza i izlaza, kao i grafičke simbole za prikazivanje redosljeda izvršavanja operacija.

Postoje razne vrste dijagrama, manje ili više udobnih. Jedna od najčešće korišćenih je vrsta opisana na slici 1.9.

Algoritam opisan na našem jeziku kome ulaz čine dva broja X i Y a rezultat je $\frac{X+Y}{X-Y}$ ili informacija da dijeljenje nije moguće, možemo opisati dijagramom toka, prikazanim na slici 1.10.

POČETAK	oznaka za početak algoritma
KRAJ	oznaka za kraj algoritma
	oznaka za ulaz
	oznaka za izlaz
operacija	simbol za pojedinačnu operaciju, korak algoritma
	operacija grananja. Od ispunjenosti uslova zavisi tok izvršavanja, tj. koja je sljedeća operacija.
	linija koja pokazuje tok algoritma, tj. redosljed izvršavanja operacija. Kako nema strelice, osnovni smjerovi su odozgo nadolje i slijeva udesno.
	prekid ili nastavak toka. Služi za olakšavanja zapisa, najčešće kada algoritam ne može da se zapiše na jednoj stranici.
dio procesa	ovaj simbol označava ranije definisani algoritam koji se može posmatrati u tekućem kao dio procesa.

Slika 1.9: Elementi dijagrama toka



Slika 1.10: Primjer dijagrama toka

1.7 Digitalni i analogni sistem za obradu podataka

Sistem za obradu podataka je uređaj koji služi za obradu podataka, tj. za rješavanje raznih zadataka. U zadatku se pojavljuju informacije raznih vrsta. Da bi uređaj mogao da radi, informacije mu moraju biti predstavljene u pogodnom obliku. Informacije predstavljene u obliku pogodnom za rad određenog uređaja, nazivamo *podacima*. I ranije smo istakli da je podatak nosilac informacije.

Podatak može da bude *analogan (kontinualan)* ili *digitalan (diskretan)*. Za podatak kažemo da je u analognom obliku ako se radi o fizičkoj veličini koja je neprekidnog tipa. Na primjer, količina tečnosti u posudi ili napon u strujnom kolu. Za podatak kažemo da je u diskretnom ili digitalnom obliku ako se radi o veličini koja nije neprekidna, odnosno čije su vrijednosti pojedinačne, dakle strogo razdvojene jedna od druge. Tih vrijednosti ima konačno ili prebrojivo mnogo. Primjeri takvih informacija su, recimo, broj primjeraka knjige, ljudi, životinja, itd.

Ako sistem za obradu podataka radi sa analognim podacima, nazivamo ga *analognim sistemom. Digitalni sistem za obradu podataka* je onaj koji radi sa digitalnim podacima.

Razvoj analognih računara počeo je tridesetih godina XX vijeka, a praktično je završen u pedesetim. Danas se koriste isključivo digitalni računari. Kod digitalnih računara buduće stanje uređaja može da se predvidi na osnovu njegovog sadašnjeg stanja. Osim toga, mala odstupanja u vrijednostima fizičkih veličina ne utiču negativno.

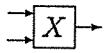
Električni signal čija je amplituda kontinualna po vremenu je primjer analognog signala. Nasuprot tome, ako amplituda signala može da se mijenja naglo ili prekidno - onda je to impulsni signal. Specijalno, ako *impulsni signal* ima samo nekoliko dozvoljenih nivoa amplitude, onda je to *digitalni signal.* Najzad, kada su definisana samo dva različita naponska nivoa - radi se o *binarnom digitalnom signalu.*

1.8 Ideja o hardveru i softveru

Sredstva za automatsku obradu podataka mogu se podijeliti u dvije klase: *hardverska* i *softverska.* U osnovi hardvera je materijalni dio računara, dok osnovu softvera čini logika na osnovu koje su izgrađeni programi koji upravljaju radom računara.

Pod *hardverom* se podrazumijevaju čvrsti, tj. fizički djelovi računara. Umjesto riječi "hardver" ponekad se kaže tehnički sistem računara, aparatni dio ili skup uređaja. *Softver* čini skup programa koji omogućuju korišćenje hardvera. Umjesto termina softver koriste se i izrazi programska podrška ili matematičko obezbjeđenje. Savremeni računari su hardversko-softverski uređaji.

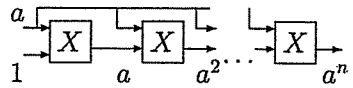
Razmotrimo jedan primjer. Neka imamo mali uređaj koji je u stanju da množi dva broja. On ima dva ulaza i jedan izlaz. Grafički ga možemo predstaviti kao na slici 1.11:



Slika 1.11: Uređaj koji množi dva broja

Pretpostavimo da želimo da riješimo zadatak o računanju veličine $b = a^n$.

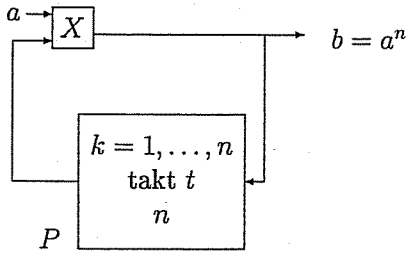
Hardversko rješenje ovog problema možemo izvesti korišćenjem našega automata X, tako sto ćemo izlaz iz jednog automata povezati na ulaz drugog. Rješenje je predstavljeno na slici 1.12:



Slika 1.12: Hardver koji računa $b = a^n$.

Na jednostavan način smo povezali n uređaja. Uočimo da se uređaj pravi samo za konkretno n. Drugim riječima, iako su zadaci o računanju $b = a^{10}$ i $b = a^{15}$ srodni, njih ne možemo riješiti jednim ovakvim uređajem. Za svako n moramo praviti poseban uređaj.

U ovome se ogleda glavni nedostatak hardverskog uređaja, odnosno glavna prednost hardversko-softverskog. Na sljedećoj slici (1.13) prikazano je rješenje koje sadrži i softversku komponentu:



Slika 1.13: Hardversko-softversko izračunavanje $b = a^n$.

Ovdje postoji još jedan automat, koji ima svoj *program.* Izvršavanje programa sastoji se u tome da automat može da broji od 1 do n. Očigledno, taj automat mora da ima svoju memoriju u kojoj može da pamti n. Izvršavajući program P ovaj automat može i da *taktuje*, tj. da u pravom trenutku prihvata izlaz iz automata X, kao i da mu predaje međurezultat na njegov drugi ulaz.

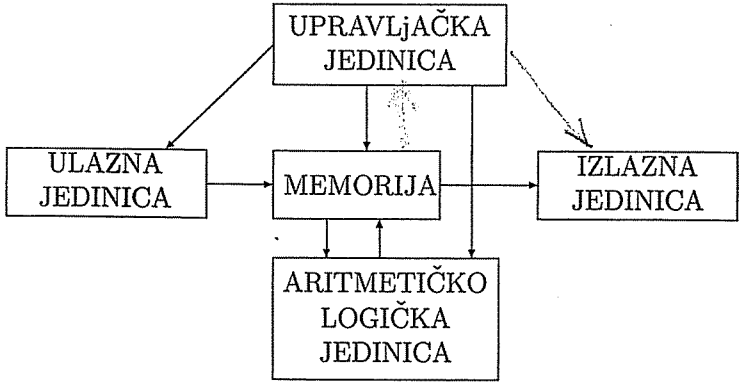
Vidimo da softverska komponenta služi za kontrolu rada hardvera. Važna prednost ovakvog uređaja je u tome što jedan uređaj rješava cijelu klasu srodnih zadataka - dovoljno je samo promijeniti n.

Da ponovimo, hardver računara čine fizičke komponente i pridružena oprema, a softver čine programi napisani za taj računar.

1.9 Šema računara

Iz prethodnog primjera vidimo da digitalni računarski sistem mora sadržati dio za upravljanje sistemom (*upravljacku jedinicu*), dio koji služi za izvršavanje naredbi ili instrukcija (*aritmetičko-logicku jedinicu*), dio za skladištenje podataka (*memoriju*), dio za komunikaciju okruženja (korisnika) sa sistemom (*ulaznu jedinicu*), kao i *izlaznu jedinicu*.

Program na osnovu kojeg radi računar može biti dio ulaza, a kao što ćemo vidjeti malo kasnije, svi današnji računari program drže u istoj memoriji koja služi i za skladištenje međurezultata.



Slika 1.14: Šema računara

Računar se sastoji iz pet djelova: ulazne, izlazne, memorijske, aritmetičko-logičke i upravljačke jedinice (slika 1.14). Opišimo podrobnije njihove funkcije.

Ulazna jedinica prima spoljašnje informacije i u pogodnom obliku ih prenosi u memorijsku jedinicu. Na određeni način, ona ima ulogu prevodioca informacija sa spoljašnjeg jezika na unutrašnji jezik računara. Naravno, proces primanja i prenošenja informacije mora biti usaglašen sa drugim procesima u digitalnom uređaju, te je stoga ulazna jedinica povezana sa upravljačkom jedinicom.

Postoje razni tipovi ulaznih uređaja, zavisno od namjene digitalnog sistema. Takvi uređaji su, na primjer, tastatura, miš, čitaci informacija sa magnetnih traka ili diskova, čitaci optičkih diskova, analogno-digitalni konvertori, uređaji za prenos informacija na daljinu, čitaci bušenih kartica, ...

Izlazna jedinica ima obrnutu ulogu od ulazne. Ona prenosi određene rezultate obrade, odnosno informacije iz memorije računara na spoljašnje uređaje. Ona ima ulogu prevodioca informacija sa unutrašnjeg jezika računara na spoljašnji jezik. I ova se jedinica aktivira samo po uputstvima koje prima iz upravljačke jedinice.

Postoje i razni tipovi izlaznih jedinica: ekrani (monitori), štampači, crtači, magnetni snimači, digitalno-analogni konvertori, ...

Memorijska jedinica ima zadatak da zapamti odnosno sačuva informacije koje dolaze iz ulazne ili aritmetičke jedinice. Ove informacije odnose se ne samo na podatke odrađivanog zadatka, odnosno međurezultate, nego i uputstva odnosno programe u vezi načina obrade primljenih podataka. Otuda je poželjno da memorijska jedinica ima što veći kapacitet, kako bi mogla da primi veliku količinu informacija. Pri tome treba imati u vidu ne samo mogućnost smještanja podataka u memoriju, već i brzo pronalaženje lokacija u njoj kojima će informacija biti predata ili iz kojih će biti uzeta. Oba ova procesa, upisivanje podataka u memoriju, kao i njihovo očitavanje iz memorije, takođe su dirigovani od strane upravljačke jedinice.

Postoji više vrsta memorije, zavisno od vrste organizacije, kao i od tipa memorijskih elemenata. Danas su za veće kapacitete memorija u upotrebi magnetni i optički elementi. Za kratkotrajne memorije, koje su obično manjeg kapaciteta, uglavnom se koriste elektronski memorijski elementi.

Zadatak **aritmetičko-logičke jedinice** je da izvršava predviđene operacije prilikom obrade informacija. Potrebne podatke ova jedinica uzima iz memorije, u koju i vraća dobijene rezultate nakon izvršenih aritmetičkih operacija. Ove operacije obuhvataju četiri osnovne računske radnje, a zastupljeni su i neki drugi matematički postupci (na primjer, logičke operacije). Svi postupci u aritmetičkoj jedinici izvode se prema uputstvima dobijenim iz kontrolne jedinice.

Aritmetičko-logička jedinica sadrži raznovrsna elektronska kola. Tu su, prije svega, logički elementi koji čine sabirače, dekodere, i druge cjeline; kao i memorijski elementi koji sačinjavanju razne registre, brojače, itd.

Upravljačka ili kontrolna jedinica je najsloženiji dio digitalnog uređaja. Ona je povezana sa svim ostalim komponentama digitalnog sistema. Ona im daje uputstva o tome šta, kada i kako treba da rade. One je središte računara. Ova jedinica tumači uputstva (program) i izdaje naredbe kako da se one ostvare. Drugim riječima, zadatak kontrolne jedinice je da organizuje i koordinira djelatnost svih ostalih jedinica u digitalnom sistemu. Recimo, kada neka informacija treba da bude primljena sa uzlaza, kontrolna jedinica aktivira ulaz i stavlja na raspolaganje prenosni put od ulaza do određenog mjesta u memoriji. Kontrolna jedinica sve radnje izvodi saglasno datom programu, koji je takođe smješten u memoriji.

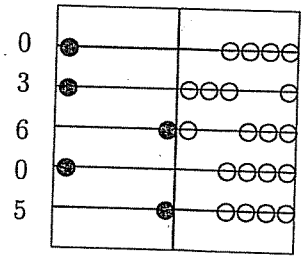
Kao kompleksan uređaj, kontrolna jedinica sadrži velik broj raznovrsnih elektronskih kola, kako sa logičkim tako i sa memorijskim elementima. Posebno treba istaći da u kontrolnoj jedinici postoji *generator taktova*. To je jedno astabilno kolo - na primjer, oscilator stabilisan pomoću kvarcnog kristala. Pored njega, tu su i djelitelji učestanosti, uobličivači napona, itd.

Za ulaznu i izlaznu jedinicu kaže se da su **periferne jedinice**. Aritmetičko-logička i kontrolna jedinicu zajedno čine **centralnu jedinicu** ili **procesor**.

Za ovakav računar, kome se program nalazi u istoj memoriji kao i podaci sa kojima radi, kažemo da je *Von Nojmanovog* tipa.

1.10 Razvoj računarske tehnike

Još prije dvije hiljade godina pojavio se abakus - prvi uređaj za računanje. Sastojao se od nekoliko žica, na kojima su se nalazile kuglice.



Slika 1.15: Abakus

Pomoću rasporeda kuglica zadaje se jedan broj. Broj predstavljaju kuglice bliži sredine, pri čemu tamne kuglice vrijede 5. Tako je na slici 1.15 prikazan broj $n = 3605$.

Na ovom uređaju lako se obavlja sabiranje, a i danas je u upotrebi u nekim zemljama Azije. Abakus koristi digitalni pristup.

Francuski matematičar Paskal (Blaise Pascal) tvorac je prvog uređaja za računanje. Njegova mehanička mašina *Pascaline* mogla je da sabira brojeve od po 8 cifara. Pojedini točkić mašine imao je deset zubaca. Mašina je koristila tegove koji su mogli da se podižu i spuštaju. Prvi primjerak mašine napravljen je 1642. godine. Ukupno je napravljeno oko 50 primjeraka. Ova mašina je, u suštini, bila neautomatizovani, mehanički, kalkulator.

Mašinu koja je mogla da obavlja sabiranje. oduzimanje i množenje napravio je 1670. godine njemački matematičar Lajbnic (Leibniz). Ona je mogla da množi broj koji ima do 5 cifara sa brojem koji ima do 12 cifara, a imala je i pokretni prenosni mehanizam.

Čarls Bebidž (Charles Babbage) počeo je 1834. godine rad na svojoj *Analitičkoj mašini* (Analytical engine), na kojoj je radio dugi niz godina. Planirao je da mašina obavlja serije izračunavanje sa brojevima do 40 cifra i to bez intervencije čovjeka. Osnovna ideja ovog mehanizma je bila dobra, ali njen tvorac nije uspjeo da je dovrši - napravio je neke njene djelove i jednostavniji prototip.

Razvoj računara u današnjem smislu počinje u godinama ispred drugog svjetskog rata. Teorijski rezultati Alena Tjuringa (A.M. Turing) objavljeni su 1936. godine. Njegova Tjuringova mašina, koju smo u osnovnim crtama opisali ranije, značajna je za razvoj računarske tehnike. Ona predstavlja matematičku formalizaciju algoritma i računara u opštem smislu.

U periodu od 1938-1941. godine Konrad Kuze (Zuse) iz Berlina napravio je mašine Z1, Z2 i Z3. To su bili elektro-mehanički kalkulatori, zasnovani na relejima. Oni su mogli da izvršavaju pravolinijske programe, tj. programe u kojima nije bila moguća promjena redosljeda izvršavanja, ono što danas nazivamo naredbom uslovnog skoka. Program se čitao sa bušene trake a ulaz se unosio sa numeričke tastature. Ističemo da ovo nisu bili računari sa programima, nego prije kalkulatori koji su mogli da se programiraju.

Slične Kuzeovim mašinama bile su mašina *ABC* iz 1941. godine, Džona Atansova (John V. Atanasoff), profesora fizike na Univerzitetu države Ajova,

kao i mašina *Harvard Mark I* ili samo *Mark I*, završena 1943 godine na Univerzitetu Harvard. Sve ove mašine su uglavnom tabelirale transcedentne funkcije i rješavale sisteme linearnih jednačina.

U toku drugog svjetskog rata napravljene su i bolje mašine, kakva je bila *Colossus*, koje su služile za dešifrovanje šifrovanih poruka.

Naučnici Ekert (J.P. Eckert) i Mahli (John Mauchly) sa Univerziteta Pensilvanije, izgradili su 1945. godine *Electronic Numerical Integerator and Calculator - ENIAC*. Ovo je bila kompletno elektronska mašina, zbog čega je mnogi smatraju prvim računarom. Sadržala je 18000 vakuumskih cijevi i 1500 releja. Radila je sa bušenim karticama. Komponente su bile povezane kablovima, pa se program zadavao fizičkim rasporedom kablova (po ovome je bio slabiji od Z3). Ovo je bio prilično mukotrpan posao - program se obično unosio 24 sata a onda bi rad ENIAC-a trajao nedjelju dana.

Djelimično zahvaljujući slučaju, upravljački signali ENIAC-a poklapali su se sa njegovim signalima podataka. Zato su izlazne linije podataka jednog uređaja mogle da budu sprovedene u upravljačke linije drugog uređaja. Iz ovih razloga, može se reći da je ENIAC raspolagao mogućnošću uslovnih skokova.

Džon Fon Nojman (John von Neumann) je bio član ekipe koja je radila na razvoju ENIAC-a. On je prvi predložio da se programom manipuliše na isti način kao sa podacima, tj. da se program upiše u memoriju računara. Fon Nojman je kazao da raspored kablova treba da bude stalan, a da program određuje veze izmedu jedinica. Logički koncept rada računara predložen od strane Fon Nojmana važi i danas!

Nešto kasnije, 1952. godine, završena je mašina *EDVAC (Electronic Discrete Variable Automatic Calculator)*, zasnovana na njegovim idejama. Mašina je mogla da radi i sa nizovima brojeva. Prosto rečeno, bila je računar u današnjem smislu. Nema principijelne razlike između računara Edvac i savremenih računara.

Fon Nojman je 1950. godine na Institute for Advanced Study (IAS) napravio mašinu *IAS*. Slične Fon Nojmanovim mašinama bile su engleski *Mark I* iz 1948. i *EDSAC* iz 1949. godine.

Početkom šezdestih godina dolazi do pronalaska tranzistora, što značajno ubrzava razvoj računara, povećavajući im mogućnosti i pouzdanost, a istovremeno smanjujući dimenzije.

Krupan korak u razvoju računara predstavljala je pojava 1964. godine IBM-ove familije računara *System 360*. U odnosu na prethodnike, ovi računari imali su poboljšane mogućnosti, a uticale su i da se tržiste računara proširi. Ovaj računar mogao je da obavlja 500000 sabiranja u sekundi.

Firma DEC je 1965. godine pokazala *PDP-8* - prvi komercijalni miniračunar. Ova mašina bila je prodor prema minijaturizaciji i projektovanju jeftinijih računara, čija je era nastupila pojavom integrisanih kola, krajem šezdesetih godina XX vijeka.

Prve personalne računare proizvela je firma Apple 1977. godine. Ovo doba je i doba izuma mikroprocesora, kada je čitav procesor računara smješten na jednom *čipu*, parčetu silicijuma.

IBM-ov personalni računar napravljen je 1981. godine. IBM PC postao je najbolje prodavani računar. Njegov uspjeh učinio je da Intel 80x86 bude najpopularniji mikroprocesor.

Često se razvoj računara dijeli u nekoliko generacija (slika 1.16), čime se žele istaći glavne prekretnice. Obično se uzima da se prva generacija odnosi na komercijalne elektonske računare.

Generacija	Period	Tehnologija
I generacija	1950-1959	Vakumske cijevi
II generacija	1960-1968	Tranzistori
III generacija	1969-1977	Integrisana kola
IV generacija	1978-	Mikroprocesori

Slika 1.16: Generacije računara

U posljednjih nekoliko godina značajan je razvoj paralelnih računara - računara koji sadrže više od jednog procesora, recimo 2^4, koji zajedno rade.

Naročito je značajan i brz razvoj računarskih mreža, gdje najvažnije mjesto svakako zauzima Internet.

1.11 Zadaci za vježbu

Zadaci služe da se ilustruje pojam algoritma. Potrebne matematičke činjenice poznate su iz srednje škole.

1. Grčki matematičar Euklid iz Aleksandrije predložio je u knjizi "Elementi" (godine 280 prije nove ere) jedan algoritam koji se i danas koristi. Euklidov algoritam za računanje NZD dva broja (najvećeg zajedničkog djelioca) izvodi se pomoću uzastopne primjene pravila: ako je $n_1 > n_2$ onda je $\text{NZD}(n_1, n_2) = \text{NZD}(n_2, n_1 \bmod n_2)$. Sastaviti program na Paskalu za računanje NZD dva broja.

2. Sastaviti program koji učitava prirodan broj i saopštava da li je taj broj prost ili složen. Zanimljivo je da nije pronađen efikasan algoritam (algoritam koji ne bi zahtijevao puno vremena da ga računar izvede) za ispitivanje da li je dati broj prost.

3. Neka $\pi(x)$ označava koliko ima prostih brojeva koji nisu veći od x. Poznata je relacija $\pi(x) \sim \frac{x}{\ln x}$ kad $x \to +\infty$. Sastaviti program u kome se računa $\pi(n)$ kada je $n = 10$, $n = 100$, $n = 1000$ i $n = 10000$ i upoređuju se izračunate vrijednosti sa procjenom za $\pi(n)$ koju daje prethodna relacija.

4. Sastaviti program koji učitava prirodan broj i saopštava njegovo rastavljanje u obliku proizvoda prostih faktora. Zanimljivo je da se ne zna da li postoji efikasan algoritam za faktorizaciju broja u obliku proizvoda prostih brojeva.

5. Za uređenu trojku prirodnih brojeva (x, y, z) kaže se da je Pitagorina trojka ako je $x^2 + y^2 = z^2$. Sve primitivne Pitagorine trojke su (x, y, z) i (y, x, z), gdje je $x = m^2 - n^2$, $y = 2mn$ i $z = m^2 + n^2$, s tim da je $m > n$, da su brojevi m i n uzajamno prosti i da su različite parnosti. Sve Pitagorine trojke imaju oblik (kx, ky, kz), gdje je $k \geq 1$ i (x, y, z) je primitivna Pitagorina trojka. Sastaviti program koji štampa sve Pitagorine trojke takve da je $x \leq 10$ i $y \leq 10$.

6. Da li polinom sa cijelim koeficijentima $p(x) = x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ ima bar jednu cjelobrojnu nulu? Poznato je da (od svih cijelih brojeva) jedino brojevi koji su faktori slobodnog sabirka a_0 mogu eventualno da budu nule. Na primjer, ako je $a_0 = -6$ onda dolaze u obzir samo $x = \pm 1, \pm 2, \pm 3, \pm 6$.

7. Da li polinom sa cijelim koeficijentima $p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ ($a_n \neq 0$) ima bar jednu cjelobrojnu nulu? Poznato je da (među svim racionalnim brojevima) jedino brojevi oblika $x = \pm \frac{p}{q}$ mogu biti racionalne nule, gdje je p faktor slobodnog sabirka a_0, a q je faktor najstarijeg koeficijenta a_n.

8. Da li jednačina sa cijelim koeficijentima $a_1 x_1 + \ldots + a_n x_n = b$ ima bar jedno cjelobrojno rješenje $x_1, \ldots, x_n$? Poznato je tvrđenje: cjelobrojno rješenje $x_1, \ldots, x_n$ postoji ako i samo ako je broj $\text{NZD}(a_1, \ldots, a_n)$ faktor broja b. Sastaviti program u kome se utvrđuje da li cjelobrojno rješenje postoji.

9. Sastaviti program za efikasno računanje stepena $f = a^n$, gdje je n prirodan broj. Uputstvo. Prvo izračunati pomoćne vrijednosti a^2, a^4, ..., a^{2^k} do granice $n < 2^{k+1}$, a zatim pomnožiti odabrane pomoćne vrijednosti. Na primjer, $a^{70} = a^{64} \cdot a^4 \cdot a^2$.

10. Sastaviti program za izvođenje dijeljenja dva prirodna broja $f = \frac{a}{b}$ uz upotrebu samo tri aritmetičke operacije sabiranja, oduzimanja i množenja. Prvo izračunati pomoćne vrijednosti $2b$, $4b$, ..., $2^{k-1}b$, $2^k b$ do granice $a < 2^{k+1}b$, a zatim postepeno umanjivati djeljenik a samo za odabrane pomoćne vrijednosti $2^k b$, $2^{k-1}b$, ..., $4b$, $2b$, b. Predloženi algoritam preuzet je sa egipatskih papirusa iz XVII vijeka prije nove ere čiji je autor Ames (Rindov papirus).

11. Postoji algoritam za računanje količnika dva realna broja $\frac{a}{b}$ uz upotrebu samo tri aritmetičke operacije sa realnim brojevima. Neka je $0 < a < 1$, $0 < b < 1$ i $0 < \frac{a}{b} < 1$. Definišimo dva niza brojeva relacijama $a_0 = a$, $b_0 = b$ i $a_n = (2 - b_{n-1})a_{n-1}$, $b_n = (2 - b_{n-1})b_{n-1}$ za $n \geq 1$. Dokazati da je $\lim_{n\to\infty} a_n = \frac{a}{b}$ i $\lim_{n\to\infty} b_n = 1$. Uzeti član niza a_n čiji je indeks dovoljno velik za približnu vrijednost količnika. Po ovom algoritmu izvođeno je dijeljenje na računaru Edsac (Cambridge, Engleska, 1949. godina), jer su hardverski bile implementirane samo tri operacije: plus, minus i puta.

12. Moguće je računati korijen $y = \sqrt{x}$ ($0 < x < 1$) pomoću tri operacije plus, minus i puta (takođe primijenjeno kod računara Edsac). Neka je $a_0 = x$, $c_0 = x - 1$ i $a_{n+1} = a_n - \frac{1}{2}a_n c_n$, $c_{n+1} = c_n^2\left(\frac{1}{4}c_n - \frac{3}{4}\right)$ za $n \geq 0$. Zapaziti da je $|a_n| < 1$ i $|c_n| < 1$. Dokazati da je $\lim_{n\to\infty} a_n = \sqrt{x}$ i $\lim_{n\to\infty} c_n = 0$.

Sljedeća grupa zadataka služi da se matematičkim sredstvima izrazi tempo kojim su se iz godine u godinu razvijale performanse i karakteristike računara. Prije toga, pogledajmo podatke o nekim značajnim računarima iz prošlosti.

godina	vrsta računara	veličina memorije	učestanost takta	sabiranja za sekundu
1941	Zuse Z3	64 × 22 bita	30 Hz	2
1949	EDSAC	512 × 36 bita	500 KHz	2000
1965	DEC PDP-8	4096 × 12 bita	1 MHz	330.000
1981	IBM PC	64 KB	5 MHz	240.000
1999	Pentium *III*	128 MB	500 MHz	500.000.000

13. Neka t označava vrijeme izraženo u godinama i neka y označava kapacitet memorije (unutrašnje memorije) izražen u bajtima. Na osnovu datih podataka, naći zavisnost $y = y(t)$ oblika $y(t) = ae^{bt}$, tako da aproksimacija bude što je moguće bolja. Drugim riječima, naći vrijednosti a i b po metodi najmanjih kvadrata. Na kraju, odrediti tzv. prosječno vrijeme udvostručavanja za y, tj. odrediti periodu udvostručavanja Δt. Perioda udvostručavanja Δt definisana je sa: za svako t važi jednakost $y(t + \Delta t) = 2y(t)$.

14. Neka y označava učestanost takta računara izraženu u hercima. Na osnovu datih podataka, naći zavisnost oblika $y = ae^{bt}$ i naći periodu udvostručavanja Δt.

15. Neka y označava broj operacija sabiranja koje računar može da obavi za jednu sekundu. Na osnovu datih podataka, naći zavisnost oblika $y = ae^{bt}$ i naći periodu udvostručavanja Δt.

Glava 2

Brojni sistemi

Brojni sistem je *skup simbola* i *pravila* koji služi za prikazivanje brojeva. Skup simbola naziva se i *alfabetom* ili skupom *cifara*. Brojevi se predstavljaju nizom cifara. Brojni sistem mora imati svojstvo da se njime mogu predstaviti svi prirodni brojevi, uključujući nulu.

2.1 Pozicioni i nepozicioni brojni sistemi

Brojne sisteme dijelimo na *pozicione* i *nepozicione*. U nepozicionom sistemu vrijednost cifre ne zavisi od njenog mjesta u nizu cifara koji čini zapis određenog broja. Analogno, sistem u kom vrijednost cifre zavisi od njenog mjesta u nizu cifara koji čini zapis određenog broja - je pozicioni.

Kroz ljudsku istoriju korišćeni su razni brojni sistemi. Prvi sistemi bili su nepozicioni, dok se danas praktično isključivo koriste pozicioni.

Egipćani su već oko 2000 godina prije nove ere imali pozicioni brojni sistem, a Vavilonci i prije njih. Sistem je bio sa osnovom 60, tako da su postojale cifre koje su odgovarale brojevima od 1 do 59. Cifru 0 nisu pisali eksplicitno. Imali su dva simbola koje ćemo označiti sa α i β. Simbol α je označavao 1 a β broj 10. Zapis $\beta\beta\alpha\alpha, \beta\alpha\alpha\alpha, \alpha\alpha\alpha\alpha$ označava broj $22 \cdot 60^2 + 13 \cdot 60 + 4 = 79984$.

U starom Rimu postojao je brojni sistem čije su cifre bili I, V, X, L, C, D i M. Njihove vrijednosti su 1, 5, 10, 50, 100, 500, 1000. Recimo, MMMCCXLVIII je zapis broja 3248. Ovaj sistem je najpoznatiji primjer nepozicionog sistema. U Evropi je sve do XIV vijeka bio glavno sredstvo

za zapis brojeva. Znamo da se i danas ponekad koristi. Osnovni problemi u radu s njim su teškoće u predstavljanju velikih brojeva (potreban je velik broj cifara) i složeno izvođenje aritmetičkih operacija.

Najprostiji način za zapisivanje brojeva je korišćenjem samo jednog znaka, recimo znaka 1. Broj n tada prosto prikazujemo pišući n puta taj znak. Recimo, 111111 prikazuje broj 6. I ovaj brojni sistem je nepozicioni.

Kod pozicionih sistema je važno gdje se cifra nalazi. Ista cifra ne vrijedi isto na dva različita mjesta u nizu cifara koji predstavlja neki broj. Na primjer, u broju 3333 sve cifre 'vrijede' različito - prva 3000, druga 300, treća 30 i četvrta 3. Možemo reći i da njihove *pozicije* vrijede 1000, 100, 10 i 1.

Pozicioni sistemi dijele se na pozicione brojne sisteme *sa osnovom* i *bez osnove*. Pozicioni sistem je sa osnovom ukoliko je u nizu cifara $c_{k-1}\dots c_0$ koji predstavlja neki broj, fiksan količnik vrijednosti pozicije cifre i vrijednosti cifre s njene desne strane. Taj fiksan količnik je *osnova* sistema. Dakle, pozicioni sistem je sa osnovom q ako je

$$\frac{a_t}{a_{t-1}} = q, \text{ za } t = 1,..,k-1.$$

Kod pozicionih sistema bez osnove vrijednosti pozicije se daju pomoću niza tih vrijednosti. Na primjer, u zapisu vremena '17 : 56' vrijednosti cifara koje predstavljaju sate je 60 puta veća od vrijednosti cifara koje predstavljaju minute.

Osnova brojnog sistema najčešće je pozitivan prirodan broj, ali može biti i negativan, racionalan, iracionalan, čak i kompleksan. Mi ćemo dalje razmatrati isključivo sisteme čija je osnova pozitivan prirodan broj veći od jedan.

Očigledan primjer takvog pozicionog sistema je *dekadni* brojni sistem, odnosno sistem sa osnovom 10, kojeg svakodnevno koristimo. U dekadnom sistemu brojevi se zapisuju ciframa iz skupa {0,1,2,...,9}. Za broj 7584 zapisan u dekadnom sistemu imamo:

$$7584 = 7\cdot 1000 + 5\cdot 100 + 8\cdot 10 + 4\cdot 1 = 7\cdot 10^3 + 5\cdot 10^2 + 8\cdot 10^1 + 4\cdot 10^0.$$

Za veličine $10^3, 10^2, 10^1$ i 10^0 kažemo da predstavljaju *težine* ili *vrijednosti pozicija* u zapisu. Očigledno je da je fiksan količnik susjednih težina - on iznosi 10.

Uopšte, broj $c_{k-1}\dots c_0$ može se predstaviti na sljedeći način:

$$c_{k-1}\dots c_0 = \sum_{i=0}^{k-1} c_i \cdot 10^i.$$

Ovaj princip može se uopštiti. Sistem sa osnovom N, $N \geq 2$, ima N cifara $\{0, 1, \dots, N-1\}$. Tada niz cifara $c_{k-1}c_{k-2}\dots c_0$ predstavlja zapis broja

$$c_{k-1}c_{k-2}\dots c_0 = c_{k-1}\cdot N^{k-1} + c_{k-2}\cdot N^{k-2} + \dots + c_0\cdot N^0$$

ili

$$c_{k-1}c_{k-2}\dots c_0 = \sum_{i=0}^{k-1} c_i \cdot N^i.$$

Kada se koristi sistem različit od dekadnog, obično se to naglasi. Oznaka $(c_{k-1}c_{k-2}\dots c_0)_N$ znači da niz cifara $c_{k-1}c_{k-2}\dots c_0$ predstavlja zapis broja u sistemu sa osnovom N.

U računarstvu su posebno interesantni sistemi sa osnovama 2, 8 i 16.

U slučaju $N = 2$, radi se *binarnom* sistemu. On ima dvije cifre: 0 i 1, koje nazivamo *bitovima* (od engleskog *binary digit - binarna cifra*). Zapis $(c_{k-1}c_{k-2}\dots c_0)_2$, pri čemu je $c_i \in \{0,1\}$ za svako $i \in \{0,\dots,k-1\}$, predstavlja zapis broja

$$\sum_{i=0}^{k-1} c_i \cdot 2^i.$$

Na primjer,

$$(1101001)_2 = 1\cdot 2^6 + 1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 = (105)_{10}.$$

U *oktalnom* brojnom sistemu ($N = 8$) cifre su $\{0,1,2,3,4,5,6,7\}$. Imamo da je $(325)_8 = 3\cdot 8^2 + 2\cdot 8^1 + 5\cdot 8^0 = (213)_{10}$.

Kada je $N = 16$, radi se o *heksadekadnom* (ili *heksadecimalnom*) brojnom sistemu. On ima 16 cifara, koje se označavaju simbolima {0,1,2,3,4,5,6,7,8,9,

A, B, C, D, E, F} i kojima u dekadnom sistemu odgovaraju brojevi od 0 do 15. Dakle, A predstavlja dekadno 10, B - 11, ..., F -15. Recimo,

$$(C3F)_{16} = 12 \cdot 16^2 + 3 \cdot 16^1 + 15 \cdot 16^0 = (3135)_{10}.$$

U tabeli 2.1 dati su prirodni brojevi od 0 do 16 u sistemima sa osnovama 10, 2, 8 i 16.

N=10	N=2	N=8	N=16
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Tabela 2.1: Brojni sistemi

Do sada smo govorili isključivo o predstavljanju cijelih brojeva. Broj koji pored cijelog ima i razlomljeni dio može se predstaviti analogno kao u dekadnom sistemu - nizom cifara prije i poslije zareza (tačke):

$$(c_{k-1}c_{k-2}\dots c_0, c_{-1}c_{-2}\dots c_{-l})_N = \sum_{i=-l}^{k-1} c_i \cdot N^i.$$

Cifre su, kao i ranije, iz skupa $\{0, 1, \dots, N-1\}$.

Na primjer,

$$(0,011)_2 = 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = \frac{1}{4} + \frac{1}{8} = (0,375)_{10}$$

$$(11,011)_2 = 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = 2 + 1 + \frac{1}{4} + \frac{1}{8} = (3,375)_{10}.$$

Četiri osnovne artimetičke operacije u ovakvim sistemima izvode se analogno kao u dekadnom sistemu.

Napomenimo da ne moraju u sistemu sa osnovom N cifre uvijek biti od 0 do $N-1$, iako smo mi to do sada podrazumijevali (a što će važiti i u daljem tekstu). Primjer takvog sistema je *balansirani ternarni sistem*. Osnova u ovom sistemu je 3, a cifre su $\{\overline{1}, 0, 1\}$, gdje cifra $\overline{1}$ odgovara dekadnom broju -1. Dekadni brojevi $-2, -1, 0, 1, 2, 3, 4, 5, 6$ u ovom sistemu zapisuju se kao $\overline{1}\,1, \overline{1}, 0, 1, 1\,\overline{1}, 10, 11, 1\,\overline{1}\,\overline{1}, 1\,\overline{1}\,0$.

Kada su u sistemu sa osnovom N cifre od 0 do $N-1$ tada nema duplikata, tj. svaki niz cifara predstavlja tačno jedan broj. Postoje i sistemi u kojima ovo ne važi. Na primjer u sistemu sa osnovom 4 i ciframa $-2, -1, 0, 1, 2$, odnosno $\overline{2}, \overline{1}, 0, 1, 2$, dekadni broj 5 zapisuje se kao 11, a broj 6 kao 12, ali i kao $2\,\overline{2}$.

2.1.1 Izbor idealne osnove

Postavlja se pitanje koji sistem koristiti u računaru. Iako je za ljude prirodan dekadni sistem, tehnološki je vrlo teško napraviti uređaje koji pamte po 10 stanja. Zato je očigledno da treba koristiti sistem koji ima manje cifara.

Da bi ovaj problem preciznije matematički razmotrili, neka je dat broj B. Uvedimo za njega *funkciju informacione zapremine*

$$f(N) = N \cdot k,$$

gdje je k broj mjesta potrebnih za zapis broja B u sistemu sa osnovom N. Dakle, na ovu funkciju utiču osnova N i broj cifara k. Intuitivno, ona nam govori da će predstavljanje nekog broja biti teže kada je njegov zapis duži, ali i kada brojni sistem ima više cifara - iz tehnoloških razloga.

Kako broj B ima k cifara, najveća cifra teži N^{k-1}. To znači da za broj B važi

$$N^{k-1} \leq B < N^k,$$

pa je

$$k \approx \log_N B.$$

Slijedi da naša funkcija f ima oblik

$$f(N) = N \cdot \log_N B$$

odnosno

$$f(N) = \frac{N}{\log_B N}.$$

Njen minimum možemo naći preko prvog izvoda po N.

$$f'(N) = \frac{\log_B N - N \cdot \frac{1}{N \ln B}}{(\log_B N)^2} = \frac{\ln B \log_B N - 1}{\ln B (\log_B N)^2} =$$

$$= \frac{\ln B \frac{\ln N}{\ln B} - 1}{\ln B (\log_B N)^2} = \frac{\ln N - 1}{\ln B (\log_B N)^2}.$$

Iz jednakosti $f'(N) = 0$ dobijamo $\ln N - 1 = 0$, odnosno $N = e \approx 2,72$.

Dakle, funkcija informacione zapremine ima minimum koji ne zavisi od broja B, i koji je između 2 i 3, pa između ta dva broja treba birati idealnu osnovu za korišćenje u računaru.

Iako je e bliže broju 3, u računarima se, osim rijetkih pokušaja, isključivo koristi binarni brojni sistem. To je zato što njegove dvije cifre imaju tehnološki jednostavniju realizaciju. Na primjer, binarne cifre se mogu predstaviti preko visokog (njemu možemo pridružiti cifru 1) i niskog (cifra 0) napona, ili preko jednog smjera magnetnog polja kojem pridružujemo cifru 1 ili drugog (0), preko prekidača koji je zatvoren ili otvoren, itd.

2.2 Konverzija broja iz jednog sistema u drugi

Prirodno se javlja zadatak prevođenja broja iz jednog sistema u drugi. Dakle, ako je dat prikaz jednog broja u sistemu sa osnovom N_1, treba odrediti prikaz tog broja u sistemu sa osnovom N_2.

Naročito je važno kada je N_1 ili N_2 jednako 10, zato što mi znamo izvoditi aritmetičke operacije u dekadnom sistemu.

Posmatraćemo tri slučaja.

Prvi slučaj je $N_2 = 10$. Ovaj slučaj ne predstavlja teškoću - pravilo prevođenja sadržano je u samoj definiciji prikazivanja broja u sistemu sa osnovom N_1. Na primjer, decimalni zapis oktalnog broja 7104 dobijamo računanjem izraza $7 \cdot 8^3 + 1 \cdot 8^2 + 0 \cdot 8^1 + 4 \cdot 8^0$. Tako imamo $(7104)_8 = (3652)_{10}$.

Drugi slučaj je $N_1 = 10$, odnosno prevođenje iz dekadnog sistema. Razmotrimo prvo brojeve bez razlomačkog dijela.

U sistemu sa osnovom N_2 taj broj ima zapis

$$B = (c_{k-1} c_{k-2} \ldots c_0)_{N_2} = \sum_{i=0}^{k-1} c_i \cdot N_2^i.$$

Uočimo da je tada $B = N_2 \sum_{i=0}^{k-2} c_{i+1} \cdot N_2^i + c_0$. Dakle, broj $B_1 = \sum_{i=0}^{k-2} c_{i+1} \cdot N_2^i$ je rezultat cjelobrojnog dijeljenja broja B sa N_2, a cifra c_0 je ostatak pri tom dijeljenju.

Slično, uočivši da je $B_1 = N_2 \sum_{i=0}^{k-3} c_{i+2} \cdot N_2^i + c_1$, imamo da je cifra c_1 ostatak pri dijeljenju broja B_1 sa N_2, a cjelobrojni količnik pri tom dijeljenju - broj $B_2 = \sum_{i=0}^{k-3} c_{i+2} \cdot N_2^i$.

Nastavkom ovog postupka zaključujemo da cifru c_t dobijamo u koraku t iz jednakosti

$$B_t = N_2 \sum_{i=0}^{k-t-2} c_{i+1+t} \cdot N_2^i + c_t.$$

Kada t bude jednako $k-2$ imaćemo

$$B_{k-2} = N_2 \sum_{i=0}^{k-(k-2)-2} c_{i+1+k-2} \cdot N_2^i + c_{k-2} =$$

$$= N_2 \sum_{i=0}^{0} c_{i+k-1} \cdot N_2^i + c_{k-2} = N_2 c_{k-1} + c_{k-2}.$$

Dobijamo, dakle, cifru c_{k-2}. U sljedećem koraku je $B_{k-1} = N_2 \cdot 0 + c_{k-1}$. Dobili smo sve cifre.

Opišimo sada ovaj postupak bez formula. Zadati broj B podijelimo sa brojem N_2, čime dobijamo cjelobrojni količnik i posljednju cifru rezultata (gledano slijeva udesno). Cjelobrojnim dijeljenjem količnika sa brojem N_2 dobijamo novi cjelobrojni količnik i pretposljednju cifru rezultata. Nastavkom ovog postupka, na kraju će cjelobrojni količnik biti 0 a ostatak će biti prva cifra rezultata.

Razmotrimo ovo na primjeru. Želimo da broj 939 zapišemo oktalno.

939:8=117 ostatak 3
117:8=14 ostatak 5
14:8=1 ostatak 6
1:8=0 ostatak 1

Zaključujemo da broj 939 u sistemu sa osnovom 8 ima zapis $(1653)_8$.

Zapis broja 939 u sistemu sa osnovom 16 dobijamo na sljedeći način:

939:16=58 ostatak 11, tj. B
58:16=3 ostatak 10, tj. A
3:16=0 ostatak 3

Rezultat je da broj 939 u sistemu sa osnovom 16 ima zapis $(3AB)_{16}$.

Na kraju, odredimo i zapis ovog broja u sistemu sa osnovom 2.

939:2=469 ostatak 1
469:2=234 ostatak 1
234:2=117 ostatak 0
117:2=58 ostatak 1
58:2=29 ostatak 0
29:2=14 ostatak 1
14:2=7 ostatak 0
7:2=3 ostatak 1
3:2=1 ostatak 1
1:2=0 ostatak 1

Dakle, broj 939 u binarnom sistemu ima zapis $(1110101011)_2$.

Razmotrimo sada zapisivanje pravog razlomka ($0 < X < 1$) u sistemu sa osnovom N_2. Pretpostavimo da taj broj u sistemu sa osnovom N_2 ima konačan zapis

$$(0, c_{-1}c_{-2}\ldots c_{-l})_{N_2} = \sum_{i=-1}^{-l} c_i \cdot N_2^i.$$

Uočimo da je tada $B \cdot N_2 = c_{-1} + \sum_{i=-1}^{-l+1} c_{i-1} \cdot N_2^i$. Znači, množenjem broja B sa N_2 dobili smo broj čiji je cjelobrojni dio cifra c_{-1} a razlomački dio broj $B_{-1} = \sum_{i=-1}^{-l+1} c_{i-1} \cdot N_2^i$ (jer je veći od 0 a manji od 1).

Analogno, uočivši da je $B_{-1} \cdot N_2 = c_{-2} + \sum_{i=-1}^{-l+2} c_{i-2} \cdot N_2^i$, dobijamo cifru c_{-2} kao cjelobrojni dio proizvoda brojeva B_{-1} i N_2, kao i razlomački dio tog proizvoda - broj $B_{-2} = \sum_{i=-1}^{-l+2} c_{i-2} \cdot N_2^i$.

Nastavkom ovog postupka zaključujemo da cifru c_{-t} dobijamo u koraku t iz jednakosti

$$B_{-(t-1)} \cdot N_2 = c_{-t} + \sum_{i=-1}^{-l+t} c_{i-t} \cdot N_2^i.$$

U koraku $l-1$ imamo

$$B_{-(l-2)} \cdot N_2 = c_{-(l-1)} + \sum_{i=-1}^{-l+l-1} c_{i-(l-1)} \cdot N_2^i,$$

odnosno

$$B_{-(l-2)} \cdot N_2 = c_{-(l-1)} + \sum_{i=-1}^{-1} c_{i-l+1} \cdot N_2^i = c_{-(l-1)} + c_{-l} N_2^{-1},$$

pa dobijamo cifru $c_{-(l-1)}$ i $B_{-(l-1)} = c_{-l} N_2^{-1}$.

U sljedećem, posljednjem, koraku dobijamo cifru c_{-l} i $B_{-l} = 0$. Dobili smo sve cifre.

Na primjer, želimo broj 0.6875 predstaviti u binarnom sistemu:

0.6875 *2=1 + 0.375
0.375*2=0+0.75
0.75*2=1+0.5

0.5*2=1+0.

Dakle, $(0.6875)_{10} = (0.1011)_2$.

Nije teško uočiti da je ovaj postupak praktično isti i u slučaju da zapis broja nije konačan. U svakom koraku dobijamo novu cifru, sve do željenog broja cifara. Kao i u dekadnom sistemu, postoje brojevi čije se cifre ponavljaju. Na primjer,

0.3*2=0+0.6
0.6*2=1+0.2
0.2*2=0+0.4
0.4*2=0+0.8
0.8*2=1+0.6
0.6*2=1+0.2
0.2*2=0+0.4
0.4*2=0+0.8
0.8*2=1+0.6
...

Dakle, $(0.3)_{10} = (0.0100\overline{1100}1\overline{1001}\ldots)_2$.

U trećem, odnosno **opštem slučaju,** za proizvoljno N_1 i N_2, prevođenje se vrši preko dekadnog sistema- prvo se iz sistema N_1 broj zapiše u dekadnom sistemu, a onda se iz dekadnog prevede u sistem N_2.

2.2.1 Konverzija kod kompatibilnih sistema

Razmotrimo još i slučaj kada su sistemi ***kompatibilni,*** tj. kada je jedna osnova stepen druge, na primjer $N_1 = N_2^b$. Pokazaćemo da u tom slučaju postoji znatno jednostavniji način konverzije.

Kako u sistemu sa osnovom N_1 postoje cifre $0, 1, 2, \ldots, N_1 - 1$, pri čemu je

$$N_1 - 1 = N_2^b - 1 = (\underbrace{N_2 - 1 N_2 - 1 \ldots N_2 - 1}_{b \text{ puta}})_{N_2},$$

to se u sistemu N_2 mogu zapisati sve cifre sistema N_1.

Kao primjer, slučaj kada je $N_1 = 8$ a $N_2 = 2$ prikazan je u tabeli 2.2. Slično, kada je $N_1 = 16$ a $N_2 = 2$ imamo tabelu 2.3.

okatalni	binarni
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Tabela 2.2: $N_1 = 8, N_2 = 2$

Sličnim rasuđivanjem dolazimo i do zaključka da svakom nizu od b cifara u sistemu N_2 odgovara jedna cifra sistema N_1.

Posmatrajmo sada zapis nekog broja u sistemu sa osnovom N_1 : $(a_{r-1}a_{r-2}\ldots a_0)_{N_1}$. Svaku cifru $a_i, i \in \{0, \ldots, r-1\}$ možemo zapisati u sistemu N_2 sa tačno b cifara $(c_{b-1}^i c_{b-2}^i \ldots c_0^i)_{N_2}$, pri čemu početne cifre mogu biti nule. Tada imamo:

$$(a_{r-1}a_{r-2}\ldots a_0)_{N_1} = \sum_{i=0}^{r-1} a_i N_1^i = \sum_{i=0}^{r-1} \left(\sum_{j=0}^{b-1} c_j^i N_2^j \right) N_1^i =$$

$$= \sum_{i=0}^{r-1} \left(\sum_{j=0}^{b-1} c_j^i N_2^j \right) \left(N_2^b\right)^i = \sum_{i=0}^{r-1} \sum_{j=0}^{b-1} c_j^i N_2^j N_2^{bi} = \sum_{i=0}^{r-1} \sum_{j=0}^{b-1} c_j^i N_2^{bi+j} =$$

$$= c_{b-1}^{r-1} N_2^{(r-1)b+(b-1)} + c_{b-2}^{r-1} N_2^{(r-1)b+(b-2)} + \ldots + c_0^{r-1} N_2^{(r-1)b} +$$

$$\ldots$$

$$+ c_{b-1}^0 N_2^{b-1} + c_{b-2}^0 N_2^{b-2} + \ldots + c_0^0 N_2^0 =$$

$$= \left(c_{b-1}^{r-1} c_{b-2}^{r-1} \ldots c_0^{r-1} \ldots . c_{b-1}^1 c_{b-2}^1 \ldots c_0^1 c_{b-1}^0 c_{b-2}^0 \ldots c_0^0 \right)_{N_2}.$$

Dakle, da bi broj zapisan u sistemu sa osnovom N_1 zapisali u sistemu N_2, pri čemu važi $N_1 = N_2^b$, dovoljno je sve cifre tog broja zapisati pomoću b cifara sistema N_2. I obratno, da bi broj zapisan u sistemu sa osnovom N_2 zapisali u sistemu N_1, pri čemu važi $N_1 = N_2^b$, dovoljno je cifre tog broja

heksa	binarni
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Tabela 2.3: $N_1 = 16, N_2 = 2$

grupisati u grupe od po b cifara (eventualno dodajući nule s lijeve strane) i svaku tu grupu zamijeniti odgovarajućom cifrom sistema N_1.

Na primjer, broj 939, koji smo razmatrali ranije, ima oktalni zapis $(1653)_8$. Koristeći gornje tabele možemo odmah napisati njegov zapis u binarnom sistemu $(001110101011)_2$, odnosno $(1110101011)_2$. Obratno, imajući njegov binarni zapis $(1110101011)_2$, grupišemo cifre u grupe od po 3 i dopisujemo potreban broj nula sa lijeve strane i lako dobijamo njegov oktalni zapis $(1653)_8$.

Slično, ako želimo heksadecimalni zapis $(3AB)_{16}$ prevesti u binarni, zapišemo cifre prema gornjoj tabeli i dobijamo da je $(3AB)_{16} = (0011\,1010\,1011)_2 = (1110101011)_2$. Opet, ako želimo broj $(1110101011)_2$ prevesti iz binarnog u heksadecimalni sistem, grupišemo cifre u grupe od po 4 (jer je $16 = 2^4$) i dopisujemo potreban broj nula sa lijeve strane i dobijamo njegov heksadecimani zapis $(3AB)_{16}$.

Konverziju iz oktalnog u heksadecimalni sistem i obratno vršimo preko

binarnog sistema.

2.3 Aritmetičke operacije sa binarnim brojevima

Aritmetičke operacije sa binarnim brojevima izvode se analogno kao sa decimalnim. U binarnoj aritmetici broj 2 igra sličnu ulogu kao broj deset u dekadnom sistemu.

Kada sabiramo brojeve u decimalnom sistemu kada se pojavi suma veća od 9 imamo prenos na sljedeće mjesto:

$$\begin{array}{r} 9 \\ +7 \\ \hline 16 \end{array}$$

Isto važi za binarno sabiranje - kada suma dva broja prelazi 1, imamo prenos na sljedeću poziciju. Znači, pri sabiranju koristimo pravila:

$$0+0=0, 0+1=1, 1+0=1 \text{ i } 1+1=0 \text{ (prenos 1)}.$$

Na primjer, ako želimo da saberemo 9 i 7 u binarnom sistemu:

$$\begin{array}{r} 1001 \\ +\ 111 \\ \hline =10000 \end{array}$$

Slično, kada želimo da oduzmemo dva broja, ponekad je potrebno da "pozajmimo" iz kolone slijeva. Koristimo pravila:

$$0-0=0, 0-1=1 \text{ (pozajmi 1)}, 1-0=1 \text{ i } 1-1=0.$$

Na primjer, oduzmimo 3 od 21:

$$\begin{array}{rr} 21 & 10101 \\ -\ 3 & -\ 11 \\ \hline =18 & =10010 \end{array}$$

U decimalnom oduzimanju 10 je pozajmljeno iz kolone 10^1 dodato cifri 1 u koloni 10^0, tj. umjesto 1-3, imamo 11 - 3. Taj broj deset se zatim vraća koloni 10^1 - ne oduzima se 0 od 2 nego 1 od 2. Ista stvar se dešava u binarnom oduzimanju. Dva se pozajmljuje iz 2^2 kolone i dodaje 2^1koloni; 0 - 1 postaje 2 - 1.

I binarno množenje izvodi se analogno dekadnom. Pri množenju koristimo pravila

$$0*0=0, 0*1=0, 1*0=0 \text{ i } 1*1=1.$$

Pomnožimo, na primjer, decimalne brojeve 10 i 11. Njihov binarni zapis je 1010, odnosno 1011.

$$\begin{array}{r} 1010 \\ *\ 1011 \\ \hline 1010 \\ 1010 \\ 0000 \\ \hline 1010 \\ 1101110 \end{array} = (110)_{10}$$

Binarne brojeve, dakle, množimo kao dekadne. Ako množimo dva broja a i b, uzimamo cifru po cifru (s desnog kraja) broja b i na tekući rezultat dodajemo broj a pomnožen sa tom cifrom broja b i stepenom broja 2 - u prvom koraku sa 2^0, u drugom sa 2^1, itd.

Formalno, ako sa p_j označimo tekući rezultat u koraku j, a sa b_j j-tu cifru broja $b = (b_{r-1}b_{r-2}\ldots b_0)_2$, uzimajući da je $p_0 = 0$ imamo da je

$$p_{j+1} = p_j + a \cdot b_j \cdot 2^j.$$

Na kraju, p_r je rezultat. Ovo pravilo (kao i kod dekadnog množenja) direktno slijedi iz pozicionog zapisa brojeva a i b.

Prethodno množenje možemo prikazati i na sljedeći način:

$$\begin{array}{rr} & 1010 \\ & *\ 1011 \\ p_0 & 0000 \\ +ab_0 \cdot 2^0 & 1010 \\ \hline p_1 & 1010 \\ +ab_1 \cdot 2^1 & 10100 \\ \hline p_2 & 11110 \\ +ab_2 \cdot 2^2 & 000000 \\ \hline p_3 & 011110 \\ +ab_3 \cdot 2^3 & 1010000 \\ \hline p_4 & 1101110 \end{array} = (110)_{10}$$

Dijeljenje binarnih brojeva svodi se na oduzimanje. Pravilo je u suštini isto kao i dekadno, posljedica je pozicionog zapisa brojeva. Na primjer, podijelimo brojeve 10000111 i 101 (135 i 5 dekadno).

$$\begin{array}{rl} 10000111 & : 101 = 11011 \ (=27 \text{ dekadno}) \\ -101 & \\ \hline 110 & \\ -\ 101 & \\ \hline 11 & \\ -\ 0 & \\ \hline 111 & \\ -\ 101 & \\ \hline 101 & \\ -\ 101 & \\ \hline 0 & \text{(ostatak)} \end{array}$$

2.4 Zadaci za vježbu

1. Prikazivanje broja u balansiranom ternarnom sistemu. Ulazni podatak je prirodan broj n. Sastaviti program koji za svaki broj u granicama od 1 do n štampa u obliku tabele njegov zapis u balansiranom ternarnom sistemu. Na primjer, ako je $n = 20$ onda tabela treba da glasi:

1	2	3	4	5	6	...	19	20
1	$1\bar{1}$	10	11	$1\bar{1}\bar{1}$	$1\bar{1}0$	...	$1\bar{1}01$	$1\bar{1}1\bar{1}$

2. Konverzija prirodnog broja iz dekadnog brojnog sistema u neki drugi brojni sistem. Neka je prirodan broj n ulazni podatak. Sastaviti program koji će da odštampa zapis tog broja u binarnom, oktalnom i heksadekadnom sistemu.

3. Konverzija pravog razlomka. Sastaviti program u kome se učitava pravi razlomak x $(0 < x < 1)$ prikazan u dekadnom obliku, a određuje se i štampa njegov zapis u binarnom brojnom sistemu. Sastaviti i drugi program koji vrši konverziju u suprotnom smjeru, iz binarnog u dekadni.

4. Aritmetičke operacije sa prirodnim brojevima u oktalnom brojnom sistemu. Učitavaju se dva oktalno prikazana broja n_1 i n_2 i vrsta operacije: sabiranje, oduzimanje ili množenje. Program treba da izračuna rezultat $n_1 + n_2$ ili $n_1 - n_2$ ili $n_1 \cdot n_2$ i da ga saopšti naravno u oktalnom obliku. Neka se program oslanja na tablicu sabiranja i tablicu množenja ($N = 8$):

+	1	2	3	4	5	6	7
1	2	3	4	5	6	7	10
2	3	4	5	6	7	10	11
3	4	5	6	7	10	11	12
4	5	6	7	10	11	12	13
5	6	7	10	11	12	13	14
6	7	10	11	12	13	14	15
7	10	11	12	13	14	15	16

·	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	2	4	6	10	12	14	16
3	3	6	11	14	17	22	25
4	4	10	14	20	24	30	34
5	5	12	17	24	31	36	43
6	6	14	22	30	36	44	52
7	7	16	25	34	43	52	61

Glava 3

Predstavljanje podataka u računaru

U ovom poglavlju razmotrićemo načine za predstavljanje podataka u računaru, i to prvo brojnih a zatim i znakovnih.

Prilikom izučavanja brojnih sistema pokazali smo da za predstavljanje brojeva u računaru treba birati između osnove 2 i 3. Takođe smo istakli da, iz tehnoloških razloga, praktično svi savremeni računari koriste binarni brojni sistem.

U ovom poglavlju razmotrićemo tehnike koje postoje za predstavljanje brojeva u računaru. Jasno je da način predstavljanja brojeva znatno utiče na način na koji se izvršavaju operacije.

Do sada, govoreći o binarnim brojevima nismo razmatrali njihovo smještanje u memoriji računara. Nismo razmatrali nikakva ograničenja u pogledu njihove maksimalne veličine. Pretpostavljali smo da imamo onoliko raspoloživog memorijskog prostora, tj. bita, koliko nam je potrebno. Međutim, u većini računarskih sistema ovo nije slučaj. Brojevi se u računaru obično predstavljaju fiksiranim nizom bita. Veličine su, tipično, 8, 16, 32, 64 i 80 bita, odnosno 1, 2, 4, 8 i 10 *bajta*.

Fiksiran broj bita koji je na raspolaganju određuje opseg brojeva koji se mogu predstaviti u računaru. Na primjer, ako koristimo 8 bita, postoji tačno 256 različitih kombinacija tih bita. To znači da najviše možemo predstaviti 256 različitih brojeva. Slično, ako koristimo 16 bita najviše možemo predstaviti 65536 različitih brojeva. Ako u 8 bita predstavljamo samo prirodne brojeve, tipično ćemo moći predstaviti brojeve od 0 do 255.

Pri predstavljanju brojeva sa ograničenim brojem bita koncept *prekoračenja* (engl. overflow) mora se razmatrati. Prekoračenje nastaje kada rezultat operacije ne može biti predstavljen korišćenjem dozvoljenog broja bita. Na primjer, ako u 8 bita predstavljamo prirodne brojeve od 0 do 255, tada nećemo moći predstaviti rezultat operacije sabiranja brojeva 134 i 197. Rezultat prekoračuje dozvoljeni opseg. Kada se desi prekoračenje, obično se gube bitovi najveće težine. Tada se rezultat prilično razlikuje od tačnog.

Razmotrićemo prvo načine predstavljanja cijelih (neoznačenih i označenih) brojeva, a zatim predstavljanje razlomaka.

3.1 Predstavljanje neoznačenih cijelih brojeva

Kada se fiksirani niz bitova u memoriji koristi samo za pozitivne cjelobrojne vrijednosti, kažemo da se radi o prikazivanju *neoznačenih cijelih brojeva*. U tom slučaju, brojevi se predstavljaju kao obični binarni brojevi sa fiksiranim brojem cifara. Najmanji broj je 0, koji se predstavlja sa $\underbrace{0000\ldots0}_{n}$ a najveći broj 2^n-1 koji se predstavlja sa $\underbrace{1111\ldots1}_{n}$. Znači, opseg dozvoljenih brojeva je

$$[0, 2^n-1],$$

gdje je n broj bita koji se koristi. Za $n=8$ opseg je [0,255], za $n=16$ opseg je [0,65535], za $n=32$ opseg je [0,4294967295], itd.

3.2 Predstavljanje označenih cijelih brojeva

Naravno, osim neoznačenih u računaru je neophodno predstavljati i *označene*, tj. i pozitivne i negativne brojeve. U ovom slučaju, gornji opseg se mora podijeliti na dio za pozitivne i dio za negativne brojeve. U sljedećim sekcijama izložićemo tri načina za predstavljanje označenih cijelih brojeva: Predznak-Veličina predstavljanje, prvi komplement i drugi komplement.

3.2.1 Predznak-Veličina predstavljanje

Najjednostavnija ideja za predstavljanje i pozitivnih i negativnih brojeva je korišćenjem jednog (obično prvog) bita zapisa za predstavljanje znaka. Po konvenciji, uzima se da vrijednost prvog bita 0 predstavlja pozitivni predznak

a 1 negativni. Ostatak, tj. apsolutna vrijednost broja zapisuje se kao pozitivan binarni broj. Ovaj način predstavljanja naziva se *Predznak-Veličina* predstavljanjem.

U ovom načinu predstavljanja, zapis 00000101 predstavlja broj 5 a zapis 10000101 broj -5. U primjeru podrazumijevamo da radimo sa 8 bita. Uopšte, ako imamo n bita, $n-1$ bit se koristi za predstavljanje neoznačenih brojeva - obično od 0 do $2^{n-1}-1$. To znači da imamo $2^{n-1}-1$ pozitivnih i isto toliko negativnih vrijednosti. Dakle, opseg dozvoljenih vrijednosti je od $-(2^{n-1}-1)$ do $2^{n-1}-1$.

Tabela 3.1 daje pregled binarnih zapisa od 0000 do 1111, uz dekadne brojeve koje oni predstavljaju, korišćenjem Predznak-Veličina predstavljanja.

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

Tabela 3.1: Predznak-Veličina predstavljanje

Prednosti ovog metoda ogledaju se u njegovoj intuitivnoj privlačnosti i jednostavnosti, simetričnom opsegu, kao i jednostavnom negiranju broja. Međutim, ovaj metod ima i ozbiljne nedostatke. Naime, broj 0 je predstavljen dva puta, kao +0 i -0, odnosno 00000000 i 10000000, čime smo realno smanjili opseg za jedan broj. Ovo stvara teškoće u logičkim operacijama tipa " *ako je*

x=0 onda ... ". Osim toga, aritmetičke operacije sa brojevima predstavljenim na ovaj način nisu jednostavne, što se naročito odnosi na oduzimanje.

3.2.2 Prvi (nepotpuni) komplement

Drugi metod koji se koristi je metod *prvog* ili *nepotpunog komplementa*. U ovom metodu, negativni broj se dobija od svoje apsolutne vrijednosti, na taj način što se sve cifre tog pozitivnog broja komplementiraju, tj. umjesto cifre 0 napiše cifra 1 a umjesto cifre 1 - cifra 0. Recimo, ukoliko koristimo 8 bita, broj -5 predstavljamo komplementiranjem broja 00000101, čime dobijamo broj 11111010. Pozitivni brojevi se predstavljaju onako kako se zapisuju u binarnom sistemu.

Kao i u Predznak-Veličina predstavljanju, vidimo da u nepotpunom komplementu prva cifra predstavlja znak: 0 predstavlja pozitivne a 1 negativne brojeve. Zato, ako imamo 4 cifre najveći pozitivan broj koji možemo predstaviti je 0111 (dekadno 7) jer zapis 1000 već predstavlja negativan broj -7.

Prikažimo tabelarno (tabela 3.2) binarne zapise od 0000 do 1111, kao i decimalne brojeve koje oni predstavljaju metodom prvog komplementa:

Metod prvog komplementa omogućuje predstavljanje brojeva od $-(2^{n-1}-1)$ do $2^{n-1}-1$. Kada je $n = 4$ od -7 do +7, kada je $n = 8$ od -127 do +127, itd.

Zapis u prvom komplementu pozitivnog broja N odgovara njegovom binarnom zapisu. Iz gornje tabele možemo uočiti sljedeće pravilo: ako imamo pozitivan broj N, manji od 2^{n-1}, zapis broja $-N$ u prvom komplementu (označimo ga sa $\overline{N}$) jednak je zapisu binarnog broja $2^n - 1 - N$. Ovo možemo zapisati na sljedeći način:

$$\overline{N} = 2^n - 1 - N.$$

Na primjer, zapis broja -6 u prvom komplementu jednak je zapisu binarnog broja $2^4 - 1 - 6 = 9$, a to je zapis 1001.

Dokažimo ovo pravilo. Neka imamo binarni broj $c = (c_{n-1}c_{n-2}\dots c_0)_2$. Dakle, $c = \sum_{i=0}^{n-1} c_i \cdot 2^i$. Zapis broja $-c$ u prvom komplementu tada je $a_{n-1}a_{n-2}\dots a_0$, pri čemu važi $c_i + a_i = 1$, za $i = 0, 1, \dots, n-1$. Ovaj zapis predstavlja binarni broj $\sum_{i=0}^{n-1} a_i \cdot 2^i$.

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-7
1001	-6
1010	-5
1011	-4
1100	-3
1101	-2
1110	-1
1111	-0

Tabela 3.2: Prvi komplement

Da bi dokazali da važi gornje pravilo, treba dokazati da je

$$\sum_{i=0}^{n-1} a_i \cdot 2^i = 2^n - 1 - \sum_{i=0}^{n-1} c_i \cdot 2^i.$$

Korišćenjem činjenice da je $c_i + a_i = 1$, za $i = 0, 1, \dots, n-1$, imamo da je

$$\sum_{i=0}^{n-1} a_i \cdot 2^i - \left(2^n - 1 - \sum_{i=0}^{n-1} c_i \cdot 2^i\right) = \left(\sum_{i=0}^{n-1} (a_i + c_i) \cdot 2^i\right) - 2^n + 1 =$$

$$= \left(\sum_{i=0}^{n-1} 2^i\right) - 2^n + 1 = (2^n - 1) - 2^n + 1 = 0,$$

što dokazuje jednakost.

Ukoliko zapis broja $-N$ u prvom komplementu opet komplementiramo dobijamo broj N:

$$\overline{\overline{N}} = 2^n - 1 - (2^n - 1 - N) = N.$$

Što se tiče aritmetičkih operacija u prvom komplementu, sabiranje i oduzimanje se svodi na obično binarno sabiranje, uz određene izmjene.

Uočimo odmah da u sabiranju može da se desi prekoračenje. Razlikujemo dvije vrste prekoračenja- *pravo prekoračenje* i *prekoračenje koje se može korigovati.*

Pravo prekoračenje se dešava kad sabiramo prevelike po apsolutnoj vrijednosti pozitivne i negativne brojeve. Na primjer, saberimo zapise brojeve 6 i 3 u prvom komplementu: 0110+ 0011=1001. Međutim, zapis 1001 u prvom komplementu je zapis broja -6 a ne broja 9! Broj 9 se uopšte i ne može zapisati u prvom komplementu korišćenjem 4 cifre.

Kako zapisi negativnih brojeva počinju sa 1, pri sabiranju zapisa dva negativna broja uvijek imamo prekoračenje. Ako saberemo -6 i -5 dobijamo 1001+ 1010=10011. I ovo je pravo prekoračenje, sabirali smo prevelike negativne brojeve, pravi rezultat se ne može predstaviti sa 4 cifre.

Ako saberemo -1 i -2 dobijamo 1110 + 1101=11011, dakle rezultat je 1011 sa prekoračenjem 1. Ovo prekoračenje je sa korekcijom. Korekcija je prosto pravilo: *saberi prekoračenje na rezultat bez prekoračenja.* Imamo 1011+1=1100 što je zapis broja -3.

Slično važi kada sabiramo 6 i -1 (dakle, kada vršimo oduzimanje). Imamo 0110+1110=10100. I tada imamo prekoračenje sa korekcijom: rezultat je 0100+1=0101, što je zapis broja 5.

Trebamo precizno definisati kad imamo pravo prekoračenje a kada možemo izvesti korekciju. Označimo sa p prenos u rezultatu iz prvog bita slijeva, a sa q prenos koji dolazi do te pozicije. Razmatranjem prethodnih primjera možemo zaključiti da imamo pravo prekoračenje kada su p i q različiti, a nemamo pravo prekoračenje kada su isti. Dakle, kada je $p=1$, $q=0$ ili $p=0$, $q=1$ imamo pravo prekoračenje, a kada je p=0, q=0 ili $p=1, q=1$ nemamo. Korekciju vršimo kada je $p=1, q=1$.

Samo jedan od p i q može biti jednak 1. Ovo je logička funkcija *XOR* odnosno *ekskluzivno ili* (radićemo kasnije). Dakle,

$$overflow = p \text{ XOR } q.$$

Ovaj metod sa sabiranjem prekoračenja naziva se *end-around carry.*

Razmotrimo i množenje brojeva a i b u prvom komplementu. Uočimo prethodno da kada se radi o neoznačenim brojevima sa n bita, za smještanje rezultata množenja potrebno je $2n$ bita. Kada se radi o brojevima predstavljenim sa n bita u prvom komplementu ili Predznak-Veličina predstavljanju, kazali smo da se oni kreću od -(2^{n-1}-1) do 2^{n-1}-1. To znači da se njihov proizvod kreće od $-(2^{2n-2} - 2^n + 1)$ do $2^{2n-2} - 2^n + 1$. Za predstavljanje granica opsega dovoljno je $2n - 1$ bita.

Konceptualno jednostavan način izvođenja množenja je da se negativni brojevi koji se množe komplementiraju, zatim se izvrši množenje, i na kraju, zavisno od znaka, eventualno komplementira rezultat. Međutim, ovakav način množenja se rijetko primjenjuje u računaru zbog tehničkih složenosti koje izaziva.

Obično se pokušava množenje brojeva u prvom komplementu izvesti kao množenje neoznačenih brojeva. Ilustrujmo ovo. Jasno da je za $a > 0$ i $b > 0$ množenje isto kao i binarno, naravno uz vođenje računa o eventualnom prekoračenju. Kada je $a > 0$ i $b < 0$, situacija je nešto složenija. Tada je $b = 2^n - 1 - b'$. Rezultat množenja je tada $P = ab = 2^{2n-1} - 1 - ab'$ (jer proizvod ima $2n - 1$ cifara). Međutim, direktnim množenjem dobijamo $P' = a(2^n - 1 - b') = a2^n - a - ab'$. Razlika je $P - P' = 2^{2n-1} - 1 - ab' - (a2^n - a - ab') = 2^{2n-1} - a2^n - 1 + a = a + (2^{2n-1} - 1 - a2^n)$. Izraz u zagradi je prvi komplement (sa $2n - 1$ cifara) broja $a2^n$, odnosno broja a, kome je dodato n nula s desne strane.

Razmotrimo primjer množenja u prvom komplementu, kada je $a > 0$ i $b < 0$, recimo $a = 7$ i $b = -2$. Zapisi ovih brojeva u prvom komplementu su 0111 i 1101.

	0111
	* 1101
p_0	0000
$+ab_0 \cdot 2^0$	0111
p_1	0111
$+ab_1 \cdot 2^1$	00000
p_2	0111
$+ab_2 \cdot 2^2$	011100
p_3	100011
$+ab_3 \cdot 2^3$	0111000
p_4	1011011

Dakle, rezutat običnog množenja je $P' = 1011011$. Dodajmo mu a, dobijamo $1011011 + 111 = 1100010$. Dodajmo ovom broju komplement broja $a * 2^4 = 1110000$. Rezultat je $P = 1100010 + 0001111 = 1110001$, što je zapis broja -14 u prvom komplementu.

Isti postupak se primjenjuje i kada je $a < 0$ i $b > 0$, samo što se na kraju dodaje komplement broja $b * 2^n$. Nešto je složeniji slučaj $a < 0$ i $b < 0$ (korekcija je složenija).

Vidimo da se aritmetičke operacije u prvom komplementu u suštini svode na obično binarno sabiranje, ali postoji i značajan problem sa korekcijom. Korekcija se najčešće svodi na novo sabiranje, što nije efikasno, s tačke složenosti uređaja i brzine.

Osim toga, značajan nedostatak ovog metoda je i u tome što on gubi jednu mogućnost predstavljanja - vidimo da imamo dva puta predstavljenu nulu.

Možemo zaključiti da, iako ima značajne prednosti u odnosu na Predznak-Veličina predstavljanje, ovaj metod nije idealan i da treba potražiti bolji.

3.2.3 Drugi (potpuni) komplement

Vidjeli smo da se prvi komplement pozitivnog broja dobija oduzimanjem tog broja od $2^n - 1$. Možemo slično pokušati i sa 2^n. Označimo sa N^* broj $2^n - N$. Vidimo da je tada

$$(N^*)^* = 2^n - (2^n - N) = N.$$

Dakle, predstavljanje negativnog broja $-N$ putem zapisa binarnog broja $2^n - N$ ima svojstvo komplementiranja. Ovaj način predstavljanja naziva se *drugim ili potpunim komplementom.*

Vidimo da je $N^* = \overline{N} + 1$. Dakle, drugi komplement broja dobijamo tako što sa prvim komplementom (kao binarnim brojem) saberemo broj 1. Vidjeli smo da ukoliko koristimo 8 bita, broj -5 predstavljamo komplementiranjem broja $N = 00000101$, čime dobijamo zapis $\overline{N} = 11111010$. Zapis broja -5 u drugom komplementu tada je jednak zbiru broja 11111010 i 1. Dobijamo da je drugi komplement broja 5 jednak $N^* = 11111011$.

Tabela 3.3 prikazuje binarne brojeve od 0000 do 1111, kao i brojeve koje oni predstavljaju metodom drugog komplementa:

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Tabela 3.3: Potpuni komplement

U slučaju da smo drugi komplement definisali kroz invertovanje cifrara i sabiranje sa brojem 1, na sličan način kao kod prvog komplementa, može se pokazati da se zapis drugog komplementa pozitivnog broja N dobija računanjem binarnog zapisa broja $2^n - N$.

Možemo koristiti još jednu vezu. Kao i u Predznak-Veličina predstavljanju, vidimo da i u nepotpunom i potpunom komplementu prva cifra predstavlja znak: 0 predstavlja pozitivne a 1 negativne brojeve. Zato možemo smatrati da je težina te cifre -2^{n-1}. Ova pretpostavka, na određeni način, i leži u osnovi metoda komplementacije. Na primjer, kazali smo da je 11111011 zapis broja -5 u potpunom komplementu. Ako prva jedinica teži -2^{n-1} i označava znak, možemo reći da je tim zapisom predstavljen broj $-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0 = -128 + 64 + 32 + 16 + 8 + 2 + 1$, što upravo iznosi -5.

Formalno, zapis $c_{n-1}c_{n-2}\ldots c_0$ u potpunom komplementu je zapis broja

$$-2^{n-1}c_{n-1} + \sum_{i=0}^{n-2} c_i \cdot 2^i.$$

Kako je prva cifra s lijeve strane jednaka 0 za pozitivne brojeve, vidimo da ovaj zapis važi i za njih. Može se dokazati da je naše pravilo nalaženja zapisa negativnog broja putem komplementiranja binarnog zapisa njegove apsolutne vrijednosti i dodavanja broja 1, ekvivalentno sa gornjom formulom.

Ako izračunamo drugi komplement broja 8 (binarno 1000) naći ćemo da je on sam sebi drugi komplement. Međutim, kako on počinje sa 1, prirodno je uzeti da zapis 1000 predstavlja negativni broj -8. Naravno, uopšteno važi ista stvar za binarni broj 2^n kada imamo n cifara.

Prednost drugog komplementa nad prvim i Predznak-Veličina predstavljanjem je u tome što je nula predstavljena samo jednom, čime je povećan opseg. On obuhvata brojeve od $N = -2^{n-1}$ do $N = 2^{n-1} - 1$. Kada je $n = 4$ od -8 do $+7$, kada je $n = 8$ od -128 do $+127$, itd.

Razmotrimo izvođenje aritmetičkih operacija sa brojevima u potpunom komplementu.

Negiranje brojeva predstavljenih u Predznak-Vrijednost metodu i nepotpunom komplementu je vrlo lako - svodi se na invertovanje prve odnosno svih cifara. U potpunom komplementu negiranje se izvodi kako smo definisali ranije: izvrši se invertovanje svih cifara i rezultatu doda broj 1 (kao sabiranje binarnih brojeva). Na primjer:

```
    +18  = 00010010
invertovano 11101101
          + 00000001
            11101110  =-18.
```

Provjerimo da važi i obrnuto, tj. da se negiranjem zapisa broja -18 dobija +18:

```
    -18  = 11101110  (u drugom komplementu)
invertovano 00010001
          + 00000001
            00010010  =18.
```

Sabiranje brojeva u drugom komplementu izvodi se kao prosto sabiranje binarnih brojeva. Oduzimanje broja je sabiranje komplementiranog broja.

Kod drugog komplementa takođe može da dođe do prekoračenja koje se dešava pod istim okolnostima kao kod prvog komplementa. Međutim, za razliku od prvog komplementa, kod drugog ne postoji sabiranje korekcije. Prosto se prekoračenje odbacuje.

Dakle, pravilo za pravo prekoračenje isto je kao kod prvog komplementa: prekoračenje se dešava kada je prenos u posljednju poziciju različit od prenosa iz nje. Prekoračenje može da se desi i pri oduzimanju, čak i oduzimanju manjeg od većeg pozitivnog broja.

Ilustrujmo ovo sa nekoliko primjera.

```
 -7 =  1001                    +3 =  0011
 +5 =  0101                    +4 =  0100
      (1110  =-2                    0111  =7

 -4 =  1100                    -4 =  1100
 -1 =  1111                    +4 =  0100
      11011  =-5 (prenos odbacujemo)    10000  =0

 +5 =  0101                    -7 =  1001
 +4 =  0100                    -6 =  1010
      (1001  =-7 (prekoračenje)     (10011  =3 (prekoračenje)

 +5 =  0101                    +2 =  0010
 -2 =  1110                    -7 =  1001
      10011  =3                     1011  =-5.
```

Što se tiče operacija množenja i dijeljenja brojeva u drugom komplementu, one se praktično potpuno svode na obično binarno množenje i dijeljenje, kao da se radi o neoznačenim brojevima.

Na primjer, u nekim slučajevima množenja imamo korekciju, za koju pravila možemo izvesti analogno kako smo to uradili kod prvog komplementa. Pri izvođenju treba uzeti u obzir da kada se radi o brojevima predstavljenim sa n bita u drugom komplementu, oni se kreću od -2^{n-1} do $2^{n-1} - 1$. To znači da se njihov proizvod kreće od $-2^{2n-2} + 2^{n-1}$ do 2^{2n-2}. Zaključujemo da je za predstavljanje proizvoda potrebno $2n$ bita za gornju granicu, a za donju i manje od toga.

Prednost metoda drugog komplementa nad prvim je i u tome što je korekcija kod množenja obično prostija nego u slučaju prvog komplementa.

3.2.4 Konverzija pri povećanju mjesta za smještanje broja

Često se dešava da treba binarni broj zapisan sa n cifara zapisati sa m cifara, pri čemu je $m > n$. Ukoliko se radi o Predznak-Veličina predstavljanju, takva konverzija se izvodi prosto. Naime, dovoljno je bit koji predstavlja znak prenijeti u krajnji lijevi bit novog broja, a na sve ostale nove cifre postaviti 0. I stari znak treba postaviti na 0.

Na primjer,

```
+18 =          00010010,  predznak-veličina, n = 8 cifara
+18 =  00000000 00010010,  predznak-veličina, m = 16 cifara

-18 =          10010010,  predznak-veličina, n = 8 cifara
-18 =  10000000 00010010,  predznak-veličina, m = 16 cifara
```

Lako je zaključiti koje pravilo važi za prvi komplement. Konverzija broja se izvodi tzv. *proširenjem znaka*: znak (tj. prvi bit s lijeva) starog zapisa iskopirati u sve nove bite. Drugim riječima, ako je broj bio pozitivan, dopuniti broj sa lijeve strane nulama, a ako je negativan - sa jedinicama.

```
+18 =          00010010,  prvi komplement, n = 8 cifara
+18 =  00000000 00010010,  prvi komplement, m = 16 cifara

-18 =          11101101,  prvi komplement, n = 8 cifara
-18 =  11111111 11101101,  prvi komplement, m = 16 cifara.
```

Konverzija broja u drugom komplementu također se izvodi proširenjem znaka.

Na primjer,

```
-18 =          11101110,  drugi komplement, n = 8 cifara
-18 =  11111111 11101110,  drugi komplement, m = 16 cifara.
```

Ovo pravilo možemo dokazati na sljedeći način.

Očigledno je da ono važi za pozitivne brojeve. Neka imamo $n-$tocifreni zapis negativnog broja $1c_{n-2}\dots c_0$ u drugom komplementu. Zapis odgovara

broju $-2^{n-1}+\sum_{i=0}^{n-2} c_i \cdot 2^i$. Ukoliko ovaj zapis dopunimo sa $m-n$ jedinica s lijeve strane $\underbrace{11\dots11}_{m-n}1c_{n-2}\dots c_0$ imamo $m-$tocifreni zapis koji odgovara broju

$$-2^{m-1}+2^{m-2}+2^{m-3}+\ldots+2^n+2^{n-1}+\sum_{i=0}^{n-2} c_i \cdot 2^i.$$

Izračunajmo razliku ova dva broja:

$$-2^{m-1}+2^{m-2}+2^{m-3}+\ldots+2^n+2^{n-1}+\sum_{i=0}^{n-2} c_i \cdot 2^i - \left(-2^{n-1}+\sum_{i=0}^{n-2} c_i \cdot 2^i\right) =$$

$$-2^{m-1}+2^{m-2}+2^{m-3}+\ldots+2^n+2^{n-1}+2^{n-1} = -2^{m-1}+2^{m-2}+2^{m-3}+\ldots+2^n+2^n =$$

$$\begin{aligned} &-2^{m-1}+2^{m-2}+2^{m-3}+\ldots+2^{n+1} = \\ &\ldots \\ &-2^{m-1}+2^{m-2}+2^{m-2} = -2^{m-1}+2^{m-1} = 0. \end{aligned}$$

Dakle, ova dva broja su jednaka, odakle slijedi pravilo.

3.2.5 Operacija pomjeranja (shift operacija)

U primjerima množenja binarnih brojeva pojavljuje se operacija množenja binarnog broja nekim stepenom dvojke. Kao što smo vidjeli u tim primjerima, izvođenje te operacije se svodi na dodavanje određenog broja nula s lijeve strane. Na primjer, ako želimo binarni 101 (dekadno 5) pomnožiti sa 2^3, dovoljno je tom broju dopisati tri nule sa desne strane, čime dobijamo 101000, odnosno dekadno 40.

Ovo svojstvo se lako dokazuje. Neka imamo $n-$tocifreni binarni zapis neoznačenog broja $c_{n-1}c_{n-2}\dots c_0$. Zapis odgovara broju $\sum_{i=0}^{n-1} c_i \cdot 2^i$. Pomnožimo ovaj broj s nekim stepenom dvojke, recimo 2^m. Dobijamo broj $2^m \cdot \sum_{i=0}^{n-1} c_i \cdot 2^i = \sum_{i=0}^{n-1} c_i \cdot 2^m \cdot 2^i = \sum_{i=0}^{n-1} c_i \cdot 2^{m+i} = \sum_{i=m}^{m+n-1} c_i \cdot 2^i = \sum_{i=m}^{m+n-1} c_i \cdot 2^i + \sum_{i=0}^{m-1} 0 \cdot 2^i$, čiji je zapis $c_{n-1}c_{n-2}\dots c_0\underbrace{00..000}_{m}$.

Operacija dodavanja nule sa desne strane naziva se *pomjeranjem ulijevo* (*shift lijevo*). Analogno, dodavanje nule sa lijeve strane naziva se *pomjeranjem udesno* (*shift udesno*). Lako se pokazuje da pomjeranje udesno u stvari znači odbacivanje posljednje cifre, odnosno dijeljenje sa 2.

Dakle, lijevim pomjerajem broja $c_{n-1}c_{n-2}\dots c_0$ dobijamo broj $c_{n-2}\dots c_0 0$, a desnim broj $0c_{n-2}\dots c_1$. Vidimo da sada lijevo pomjeranje nije uvijek množenje sa 2 - gubi se prva cifra koja je mogla biti 1.

Kada radimo sa označenim brojevima, moramo voditi računa o načinu predstavljanja. Zato se operacije pomjeranje drugačije definišu.

Kada radimo u Predznak-Veličina predstavljanju, u broju $c_{n-1}c_{n-2}\dots c_0$ cifru c_{n-1} ne treba dirati. Tako lijevim pomjerajem broja $c_{n-1}c_{n-2}\dots c_0$ dobijamo broj $c_{n-1}c_{n-3}\dots c_0 0$, a desnim broj $c_{n-1}0c_{n-2}\dots c_1$.

U prvom komplementu prirodno je definisati operacije pomjeranja na sljedeći način. Lijevim *cirkularnim pomjerajem* broja $c_{n-1}c_{n-2}\dots c_0$ dobijamo broj $c_{n-2}c_{n-1}\dots c_0 c_{n-1}$, a desnim *pomjeranjem sa propagacijom znaka* broj $c_{n-1}c_{n-1}\dots c_1$.

U drugom komplementu definicije su sljedeće. Lijevim pomjerajem broja $c_{n-1}c_{n-2}\dots c_0$ dobijamo broj $c_{n-2}c_{n-3}\dots c_0 0$, a desnim pomjeranjem sa propagacijom znaka (dakle, isto kao kod prvog komplementa) broj $c_{n-1}c_{n-1}\dots c_1$.

3.2.6 BCD kodovi

Postoji još jedan način za predstavljanje brojeva u računaru, suštinski različit od načina o kojima smo do sada govorili. Osnovna ideja je da se cifre dekadnog broja zamijene nizom bita fiksne dužine. Recimo, dekadnu cifru 0 možemo zamijeniti sa 0000, cifru 1 sa 0001, cifru 2 sa 0010, ..., cifru 9 sa 1001. Tako broj 321 možemo predstaviti kao 001100100001.

Ovo je *Binary Coded Decimal* ili *BCD kod*. Dakle, dekadna cifra se binarno kodira. Ovo kodiranje koje smo naveli kao primjer je takozvano '8421' BCD ili poziciono BCD kodiranje.

Prikažimo još jedan BCD kod, tzv. 'Višak 3' kod. U njemu svaku dekadnu cifru zamjenjujemo četvorocifrenim binarnim zapisom te cifre uvećane za 3. Na primjer, cifru 1 zamjenjujemo zapisom broja 1 + 3, tj. zapisom 0100 (tabela 3.4).

Dekadna cifra	8421 kod	Višak 3 kod
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Tabela 3.4: BCD kodovi

Višak 3 kod ima interesantno svojstvo komplementarnosti: dvije dekadne cifre koje se dopunjavaju do 9 imaju kodove koji se dopunjavaju do 1111. Uvodi se obično neki kod za znak, recimo 0001 za minus. Ako imamo 2 bajta, možemo predstaviti trocifrene označene brojeve. Recimo, -87 predstavljamo kao 0001 0011 1011 1010.

Ovaj kod se koristio kod računara prve i druge generacije a danas se uglavnom koristi kod kalkulatora.

3.3 Predstavljanje racionalnih brojeva

Potrebno je u računaru osim cijelih predstavljati i racionalne brojeve. Kod njih je problem ograničenog prostora još izraženiji. Dva osnovna načina za predstavljanje racionalnih brojeva u računaru su predstavljanje u *nepokretnom* i *pokretnom* zarezu.

3.3.1 Nepokretni zarez

Najočigledniji način za predstavljanje racionalnih brojeva je da se raspoloživi broj bita podijeli na cjeline koje predstavljaju cijeli i razlomački dio broja. Tada kažemo da se radi o *brojevima sa fiksnom preciznošću* ili brojevima *u nepokretnom zarezu*.

БИБЛИОТЕКА ТЕХНИЧКОГ ФАКУЛТЕТА ТИТОГРАД

Razlomački dio tada se predstavlja kao običan pozitivan cijeli broj, a cjelobrojni dio možemo predstavljati na bilo koji od izloženih načina. Na primjer, možemo se odlučiti na Predznak-veličina predstavljanje. Tada ako ukupno imamo 16 bita, možemo izvojiti 7+1 bita za cjelobrojni i 8 bita za razlomački dio, pa broj -13.45 predstavljamo zapisom 1000110100101101. Opet, ako cjelobrojni dio predstavljamo u drugom komplementu, tada ovaj broj ima zapis 1111001100101101.

cjelobrojni dio	razlomački dio

U formatu sa nepokretnim zarezom, na primjer u drugom komplementu, moguće je predstaviti pozitivne i negativne brojeve u opsegu simetričnom na brojnoj osi u odnosu na nulu. Nedostatak ovog metoda je u tome što on omogućuje predstavljanje skupa realnih brojeva u prilično uskom opsegu. Ne mogu se predstaviti veoma veliki, niti veoma mali brojevi. Pri dijeljenju dva velika broja, po pravilu, gubi se razlomački dio rezultata.

3.3.2 Pokretni zarez

U radu sa dekadnim brojevima davno je pronađen metod predstavljanja vrlo velikih i vrlo malih brojeva sa znatno manje cifara nego što je potrebno u običnom pozicionom zapisu. Taj način je tzv. *naučna notacija*. Broja 976 000 000 000 000 možemo zapisati kao $9.76 \cdot 10^{14}$, a broj 0.000 000 000 000 097 6 kao $9.76 \cdot 10^{-14}$. U suštini, tačka se pokreće na željeno mjesto. Zato se ovaj metod naziva predstavljanje brojeva u **pokretnom zarezu**.

Uopšte, brojevi se predstavljaju u obliku

$$\pm m \cdot B^E.$$

Broj m nazivamo *mantisom*, E *eksponentom*, broj B^E *karakteristikom* a broj B *osnovom karakteristike*. Eksponent je cijeli broj.

Jasno je da postoji dosta načina za predstavljanje broja u ovom obliku jer mantisa može biti 97.6, 9.76, 0.976, 0.0976, itd. Obično se uzima da je mantisa manja od jedan i da prva cifra nakon tačke nije 0, tj. uzima se da je mantisa između $\frac{1}{B}$ i 1, odnosno da je *normalizovana*. Dakle,

$$\frac{1}{B} \leq m < 1.$$

Ovako predstavljen broj možemo zapisati u računaru preko niza binarnih cifara, koji se sastoji iz tri dijela

- polje znaka (plus ili minus)
- polje mantise
- polje eksponenta.

Broj B (*osnovu karakteristike*) ne moramo predstavljati jer je on fiksiran.

Eksponent E definiše pravo mjesto tačke u mantisi. Povećanjem broja E za jedan, broj se množi sa B. Znači, ukoliko je B=2 broj se množi sa 2. Obično se ovo smatra nedovoljnim. Zato se uzima neka druga osnova, najčešće stepen dvojke, recimo 16. U daljem izlaganju smatraćemo da je $B = 16$.

Realan broj X tada predstavljamo sa tri veličine s(*sign, znak*), m i E. Ako je $X \geq 0$ tada je $s = 1$, a ako je $X < 0$ tada je $s = -1$. E je cijeli broj a za m važi $\frac{1}{16} \leq m < 1$.

Tada je

$$X = s \cdot m \cdot 16^E.$$

U računaru je ograničen prostor za predstavljanje brojeva, obično je to 4, 8 ili 10 bajta. Uzećemo da imamo 4 bajta, tj. 32 bita za predstavljanje brojeva. Radi lakšeg izlaganja, numerišimo ih od 1 do 32 sa lijeve strane. Tih 32 bita moramo podijeliti na tri dijela, na djelove za predstavljanje s, m i E.

Za s je dovoljan jedan bit, neka to bude bit 1. Uzmimo za predstavljanje eksponenta 7 bita, od bita broj 2 do bita broj 8. Za mantisu tada ostaje 24 bita, od broja 9 do 32.

z	eksponent	mantisa
1	2 ... 8	9 ... 32

Za znak imamo jedan bit - označimo ga sa s_1. Neka bude s_1=0 kada je broj X pozitivan, a s_1=1 kada je negativan.

Eksponent je cijeli broj pa ćemo za njegovo predstavljanje koristiti drugi komplement. Kako imamo 7 bita, možemo predstaviti eksponente od -2^7 do 2^7-1, tj. $-64 \leq E \leq 63$.

Mantisa je pozitivan racionalni broj. Njega predstavljamo sa njegove 24 cifre iza tačke u njegovom zapisu u binarnom sistemu. Ako je $m = (0.m_1m_2\ldots m_{24})_2$ tada u bite od 9 do 32 treba upisati $m_1, m_2, \ldots, m_{24}$.

Definisali smo u potpunosti način predstavljanja broja X u pokretnom zarezu:

$$s e_1e_2e_3e_4e_5e_6e_7 m_1m_2\ldots m_{24}.$$

Zbog pozitivne mantise, ovako se ne može predstaviti nula. Definišimo tada da nulu predstavljamo zapisom u kome su sve cifre 0.

Ilustrujmo ovo sa nekoliko primjera.

Koji broj je predstavljen zapisom 11111001111100...000? Vidimo da je $s_1 = 1$, što znači da je $s = -1$, odnosno zapis predstavlja negativan broj. Eksponent predstavlja sedam cifara 1111100. Na osnovu ranije izloženog, znamo da je ovo zapis broja -4 u drugom komplementu. Znači $E = -4$. Dalje, zapis 11111100...000 predstavlja mantisu $m = (0.11111100\ldots000)_2$, odnosno

$$m = \frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\frac{1}{32}+\frac{1}{64} = \frac{63}{64}.$$

Konačno, zaključujemo da se radi o broju $X = s \cdot m \cdot 16^E = -\frac{63}{64} \cdot 16^{-4} = -0.0001502$.

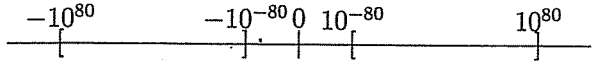
Imamo da je $48.0 = 1 \cdot \frac{3}{16} \cdot 16^2$. Zaključujemo da zapis broja 48.0 u pokretnom zarezu počinje bitom 0. Eksponent je predstavljen sa 7 bita: 0000010, koji su zapis broja 2 u drugom komplementu. Mantisu čine 24 cifre, koje lako izračunavamo imajući u vidu da je $\frac{3}{16} = \frac{1}{8}+\frac{1}{16} = (0.0011)_2$. Dakle, zapis broja 48.0 u pokretnom zarezu je 00000010001100000000000000000000.

Nađimo još i zapis broja 76.8. Imamo da je $76.8 = 1 \cdot 0.3 \cdot 16^2$. Znak i eksponent su isti kao u prethodnom primjeru. Kako je $0.3 = (0.0\overline{1001})_2$, tj. broj 0.3 ima u binarnom zapisu beskonačno cifara, uzećemo za mantisu samo prve 24 iza tačke. Zaključujemo da je zapis broja 76.8 u pokretnom zarezu 00000010010011001100110011001100.

Odredimo opseg brojeva koji se može predstaviti na ovaj način. Posmatrajmo samo pozitivne brojeve. Najmanji broj imamo kada je najmanja

mantisa i eksponent. To su $m = \frac{1}{16}$ i E=-64. Tako je minimalni pozitivan broj koji se može predstaviti na ovaj način: min$=\frac{1}{16} \cdot 16^{-64}$. Slično, najveći pozitivan broj je kada je $m \approx 1$ i $E = 63$, tj. max $\approx 16^{63}$. Dakle raspon pozitivnih brojeva je od $\frac{1}{16} \cdot 16^{-64}$ do 16^{63}, što približno iznosi od 10^{-80} do 10^{80}.

Simetrično, raspon negativnih brojeva je -10^{80} do -10^{-80} (slika 3.1).



Slika 3.1: Raspon brojeva

Brojevi koji su van ova dva segmenta ne mogu se predstaviti u pokretnom zarezu (pod pretpostavkom da se radi u 4 bajta, sa eksponentom dužine 7). Tada imamo pozitivni (odnosno negativni) *overflow* ili *underflow*.

Ako na opisani način hoćemo da predstavimo brojeve A=0.1236 i B=0.1237, njihovi zapisi će se razlikovati. Međutim, ako hoćemo da predstavimo brojeve A i B koji se razlikuju na osmoj dekadnoj cifri iza tačke, dobićemo isti zapis! Ovo je zato što su takvim brojevima eksponenti isti, kao i prve 24 cifre mantise iza tačke. Dakle, te brojeve smatramo istima. Tako da možemo reći da imamo *preciznost* ili *tačnost* od 7 cifara.

Ilustrujmo *gustinu* predstavljenih brojeva na brojnoj osi (slika 3.2).



Slika 3.2: Gustina brojeva

Postoji nekoliko opšteprihvaćenih standarda za predstavljanje brojeva u pokretnom zarezu, kakav je, na primjer, IEEE standard. Princip tih standarda je isti kao izloženi. Varijacije se uglavnom odnose na broj bita za mantisu. Recimo, može se uzeti jedan bit više, 25 umjesto 24. Tada povećavamo preciznost ali smanjujemo opseg - za pozitivne brojeve biće približno od 10^{-38} do 10^{38}.

3.4 Predstavljanje znakova

Osim brojeva, u računaru je potrebno predstaviti i znakove. Postoji nekoliko načina za to, od kojih su danas najpopularniji ASCII i UNICODE rasporedi ili kodovi.

3.4.1 ASCII kodovi

Za zapisivanje jednog štampanog znaka, tj. jednog *karaktera* obično se koristi jedan bajt. Danas je standard ASCII kod, tj. *American Standard Code for Infromation Interchange*. ASCII kod omogućuje kodiranje slova, cifara, znakova interpunkcije. Originalno, to je sedmobitni kod, koji dozvoljava predstavljanje 128 znakova (tabela 3.5).

		Prva četiri bita (viši bitovi)							
Druga četiri bita (niži bitovi)		0000	0001	0010	0011	0100	0101	0110	0111
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0000	**0**	NUL	DLE	Space	0	@	P	‘	p
0001	**1**	SOH	DC1	!	1	A	Q	a	q
0010	**2**	STX	DC2	“	2	B	R	b	r
0011	**3**	ETX	DC3	#	3	C	S	c	s
0100	**4**	EOT	DC4	$	4	D	T	d	t
0101	**5**	ENQ	NAK	%	5	E	U	e	u
0110	**6**	ACK	SYN	&	6	F	V	f	v
0111	**7**	BEL	ETB	‘	7	G	W	g	w
1000	**8**	BS	CAN	(	8	H	X	h	x
1001	**9**	HT	EM	)	9	I	Y	i	y
1010	**A**	LF	SUB	*	:	J	Z	j	z
1011	**B**	VT	ESC	+	;	K	[	k	{
1100	**C**	FF	FS	,	<	L	\	l	|
1101	**D**	CR	GS	-	=	M	]	m	}
1110	**E**	SO	RS	.	>	N		n	~
1111	**F**	SI	US	/	?	O	_	o	DEL

Tabela 3.5: ASCII kod

Prvih 32 koda, kao i posljednji (DEL) predstavljaju kontrolne kodove za upravljanje perifernim uređajima i ne mogu biti odštampani, tačnije nije im pridružen neki standardan simbol. Njima priradaju $(0D)_{16}$ (CR ili carriage return), $(0A)_{16}$ (LF ili line feed), $(0C)_{16}$ (FF ili form feed), $(08)_{16}$ (BS ili backspace).

Dekadne cifre predstavljaju se brojevima od $(30)_{16}$ do $(39)_{16}$. Velika slova engleske abecede su od $(41)_{16}$ do $(5A)_{16}$. Mala slova su od $(61)_{16}$ do $(7A)_{16}$. Primijetimo da se kod za malo slovo razlikuje od koda za veliko slovo za $(20)_{16}$ što olakšava konverziju velikih u mala slova i obratno. Ostale kombinacije predstavljaju znake interpunkcije i specijalne znake.

Ako na numeričkoj tastaturi PC računara otkucate 65 držeći pritisnut taster Alt, ispisaćete slovo 'A'.

Osmi bit se koristi na razne načine. Recimo, može se iskoristiti kao *bit parnosti*, tj. on je 1 ako među ostalih 7 bita ima neparan broj jedinica, inače je 0. Ovo se koristi u prenosu podataka i služi za detektovanje grešaka u prenosu.

Osmi bit se može iskoristiti i za proširenje ASCII tabele, tako da ona može predstaviti 256 znakova umjesto 128. Tako postoje razna proširenja, na primjer za razne jezike. Neka od njih su standardna, neka nestandardna. Neke zemlje koriste kodove od 128 do 255 po svome. Čak je i prvi IBM PC zbog grafike proširio osnovnu ASCII tabelu na nestandardan način. Ovo nestandardno proširenje se i danas puno koristi i može se na određeni način smatrati standardom.

3.4.2 Konverzija između ASCII i BCD

Kazali smo da su u ASCII kodu numerički karakteri predstavljeni od $(30)_{16}$ do $(39)_{16}$. Konverzija između ASCII kodirane cifre i BCD cifre može se izvršiti dodavanjem odnosno oduzimanjem $(30)_{16}$. Oduzmimo $(30)_{16}$ od ASCII koda cifre i dobićemo njen BCD kod, odnosno dodajmo $(30)_{16}$ BCD kodu cifre da dobijemo njen ASCII kod.

3.4.3 Ostali kodovi

Od osmobitnih alfanumeričkih kodova treba pomenuti i EBCDIC američke firme IBM koji je jedno vrijeme bio popularan, a danas se sve manje upotrebljava. EBCDIC je skraćenica od *Extended Binary Coded Decimal Interchange*

Code (prošireni BCD kod za razmjenu) i on je 8-bitna proširenje 4-bitnog BCD kodiranja.

Unicode predstavlja standard koji omogućuje predstavljanje znakova u raznim jezicima. On koristi dva bajta za predstavljanje znakova. Tako jedna kodna strana ima 2^{16}, odnosno 65536 znakova. Osim 16 bita, koji se mogu predstaviti sa 4 heksadecimalne cifre, npr. U+0041 (U+ je standardan prefiks koji označava da se radi o UNICODE karakteru), UNICODE sadrži i dosta dodatnih infromacija o karakteru, kao što su naziv, recimo "LATIN CAPITAL LETTER A", tip karaktera, klasa, pridruženi simbol, itd.

3.5 Matematički pojam kodiranja

Teorija kodiranja je dio matematike, tačnije jedne grane matematike - diskretne matematike. Najšire gledano, kodiranje je funkcija. Na primjer, pravilo drugog komplementa pridružuje svakom cijelom broju od -128 do 127 jedan tačno određen *kod*, tj. niz od 8 binarnih cifara. Dakle, pravilo drugog komplementa *kodira* brojeve od -128 do 127. Ovo kodiranje ima i svojstvo *uzajamne jednoznačnosti*: jednom broju od -128 do 127 pridružen je tačno jedan kod, i obrnuto, svakom 8 cifrenom kodu pridružen tačno jedan broj.

Formalizujmo pojam kodiranja. Neka je $A = \{a_1, a_2, \ldots, a_m\}$. Za skup A kažemo da je *azbuka* a za njegove članove kažemo da su *slova*. *Riječ* u azbuci A je niz od nekoliko slova te azbuke. Skup svih riječi u azbuci A označava se kao $\Omega(A)$. Ove riječi su predmet kodiranja. Često nas ne interesuje kodiranje svih riječi neke azbuke, već samo nekog njegovog podskupa. Označimo sa $\Omega'(A)$ podskup skupa $\Omega(A)$.Članove tog skupa nazivamo *podacima* i njih želimo da kodiramo, tj. da im pridružimo elemente nekog drugog skupa. Taj drugi skup je takođe skup riječi, skup riječi nad azbukom $B = \{b_1, b_2, \ldots, b_n\}$, označimo ga sa $\Omega(B)$.

Tada *kodiranje* možemo definisati kao bilo koju funkciju

$$F : \Omega'(A) \to \Omega(B).$$

Podatak je element skupa $\Omega'(A)$ a njegov *kod* je njemu pridruženi element skupa $\Omega(B)$.

Kod kodiranja se praktično uvijek zahtjeva da je ono *funkcija* a ne uopšteno preslikavanje. To znači da je kodiranje jednoznačno, tj. da podatku pridružuje jedan kod.

Lako je vidjeti da ova definicija obuhvata, recimo, ranije definisane načine predstavljanja brojeva u računaru.

Za kodiranje kažemo da je *binarno* ako je $B = \{0, 1\}$. Drugim riječima, kodiranje je binarno ako su kodovi nizovi binarnih cifara.

Kodiranje je *ravnomjerno* ako su svi kodovi iste dužine, odnosno ako je iste dužine svaka riječ iz $\Omega(B)$, koja je kod nekog podatka. Na primjer, kodiranje 'prvi komplement sa 8 cifara', za brojeve od -127 do +127 je binarno i ravnomjerno.

Dekodiranje je proces inverzan kodiranju. Dekodiranje ne mora biti jednoznačno, mada se u praksi upotrebljavaju kodiranja koja imaju svojstvo da se mogu jednoznačno dekodirati..

Za kodiranje $F : \Omega'(A) \to \Omega(B)$ kažemo da je *uspješno*, ili da dopušta *jednoznačno dekodiranje* ako je funkcija F obostrano jednoznačna. Tada se ne može desiti da dva različita podatka imaju jedan isti kod.

Poseban dio teorije kodiranja je *kriptografija*. U kriptografiji je osnovni zadatak naći kodiranje koje može lako dekodirati onaj ko kodira ili ko ima određena znanja o kodiranju, a da pri tome dekodiranje bude što je moguće teže, tj. praktično nemoguće, za neželjene osobe.

3.5.1 Azbučna kodiranja

Kodiranje je *azbučno* ako je svakom slovu azbuke A pridružena jedna riječ w_k azbuke B i ako se kod riječi iz azbuke A dobija tako što se nadovežu (napišu jedna za drugom) odgovarajuće riječi azbuke B, koje su pridružene slovima azbuke A iz te riječi. Za definisanje azbučnog kodiranja dovoljno je zadati kodne riječi $w_1, \ldots, w_m$ koje odgovaraju slovima $a_1, \ldots, a_m$ azbuke A. Ovo definisanje možemo opisati *šablonom* $\sum$, recimo spiskom $a_1 - w_1, \ldots, a_m - w_m$.

Umjesto azbučno kodiranje može se reći i kodiranje *slovo-po-slovo* ili *znak-po-znak*. Ako u azbučnom kodiranju sve riječi $w_1, \ldots, w_n$ imaju istu dužinu (broj slova) kažemo da se radi o *ravnomjernom azbučnom kodiranju*.

Razmotrimo jedan primjer. Neka su date azbuke $A = \{a_1, a_2\}$ i $B = \{0, 1\}$ i šablon kodiranja

$$\sum : a_1 - 0,\ a_2 - 01.$$

Ovaj kod nije ravnomjeran jer kodne riječi $w_1 = 0$ i $w_2 = 01$ imaju različite dužine. Kod riječi $a_2a_2a_1$ je tada 01010. Sa druge strane, rastumačimo kod 001010001. Odgovor je $a_1a_2a_2a_1a_1a_2$. Može se dokazati da je ovo kodiranje uspješno, tj. da je dekodiranje uvijek jednoznačno.

Neka su sada zadate azbuke $A = \{a_1, a_2, a_3\}$ i $B = \{b_1, b_2, b_3\}$ i šablon kodiranja

$$\sum : a_1 - b_1,\ \ a_2 - b_1b_2,\ \ a_3 - b_3b_1.$$

Ovo azbučno kodiranje ima ista svojstva kao prethodno - nije ravnomjerno a dekodiranje je jednoznačno.

Kodiranje cijelih brojeva u prvom komplementu nije azbučno. Kodiranje cijelih brojeva na 'Višak 3' način jeste azbučno. Njegov šablon kodiranja $\sum$ izražen je tabelom koju smo ranije prikazali. Ovo kodiranje ima svojstvo ravnomjernosti: kodovi cifara, tj. slova dekadne azbuke, su svi dužine četiri. Slično, ASCII tabela je, u suštini, azbučno kodiranje.

Razmotrimo jednu klasu azbučnih kodiranja - klasu prefiksnih azbučnih kodiranja.

Neka je data riječ r sastavljena od q slova $r = c_1c_2 \dots c_q$. Tada je svaka riječ oblika $c_1c_2 \dots c_p$, pri čemu je $1 \leq p \leq q$, *prefiks* ili *početak* riječi r. Na primjer, riječi AV i AVI su prefiksi riječi AVION.

Za azbučno kodiranje kažemo da je *prefiksno* ako nijedna kodna riječ w_i nije prefiks druge kodne riječi w_j. Dakle, u prefiksnom kodiranju riječi se lako razlikuju, ne može se dodavanjem slova jednoj kodnoj riječi w_i dobiti druga kodna riječ w_j.

Dekodiranje kod prefiksnog kodiranja je jednostavno. Neka je data riječ R u azbuci B. Čitamo je sa početka dok ne nađemo niz pročitanih slova koji se poklapa sa nekom kodnom riječi, recimo w_2. Kako je kodiranje prefiksno, sigurni smo da ne možemo dobiti neku drugu kodnu riječ, pa zaključujemo da je prvo slovo originalne riječi slovo a_2. Posmatrajmo sada preostali dio riječi. Na isti način, zaključimo da je drugo slovo, recimo, a_4, itd. U svakom koraku napredujemo u riječi R. Kada je iscrpimo završili smo dekodiranje, tj. sastavili smo originalnu riječ, recimo $a_2a_4 \dots$.

Moguće je da riječ R uopšte nije kod riječi. U tom slučaju, naš bi se postupak zaustavio u nekom koraku kada ne bi mogli identifikovati nijednu kodnu riječ na početku tekuće riječi.

Ovim razmatranjima smo dokazali sljedeću teoremu:

Teorema. Svako prefiksno kodiranje je uspješno, tj. uzajamno jednoznačno. Ili, drugačije formulisano, dekodiranje kod svakog prefiksnog kodiranja je jednoznačno.

Dakle, kod prefiksnog kodiranja ne može se desiti da neki kod rastumačimo na dva načina.

Ima primjera azbučnog kodiranja koje je uspješno a nije prefiksno. Recimo, prethodna dva primjera azbučnog kodiranja, su primjeri takvih kodiranja. Recimo, u prvom primjeru je riječ $w_1 = 0$ prefiks riječi $w_2 = 01$, dakle kodiranje nije prefiksno, a vidjeli smo da je uzajamno jednoznačno.

Kod azbučnog kodiranja obično se uzima da svi članovi skupa $\Omega(A)$ podliježu kodiranju, tj. da je $\Omega'(A) = \Omega(A)$. Za kodne riječi $w_1, \dots, w_m$ kaže se i da su *elementarni kodovi*.

3.5.2 Primjer primjene prefiksnog azbučnog kodiranja

U današnje vrijeme puno se koriste programi za *kompresiju* ili sabijanje podataka. Oni smanjuju prostor koji zauzimaju podaci u spoljašnjoj memoriji ili pri prenosu podataka kroz računarske mreže. Podatke zapisane u ASCII kodu ovi programi prevode u kraće zapise, obično primjenom nekog prefiksnog kodiranja.

Primjer takvog kodiranja je Hafmanov (Huffman) kod. To je prefiksni kod koji karaktere kodira zavisno od učestalosti njihovog pojavljivanja u izvornom tekstu. Slova koja se često pojavljuju kodiraju se kratkim riječima, a ona koja se rjeđe pojavljuju - dužim. Na primjer, ako se slovo '*a*' najčešće pojavljuje možemo ga kodirati riječju 0 a slovo '*dž*' riječju 1100. Hafmanov kod se može predstaviti preko binarnog stabla, što je korisno za dekodiranje.

Neka imamo tekst dužine 100 000 znakova. Pretpostavimo da se u njemu pojavljuju samo slova *a*, *b*, *c*, *d*, *e* i *f*. Neka se ta slova, redom, pojavljuju po 45000, 13000, 12000, 16000, 9000 i 5000 puta. Tada slova možemo kodirati na sljedeći način (tabela 3.6):

Ovo kodiranje je prefiksno. Uporedimo dužine zapisa teksta u ASCII kodu i u Hafmanom kodu. U ASCII kodu svaki karakter će biti predstavljen sa 8 bita, pa će dužina zapisa biti 8*100 000=800 000 binarnih cifara. U Hafmanovom kodu dužina zapisa je 45000*1 + 13000*3 + 12000*3 + 16000*3

a	b	c	d	e	f
0	101	100	111	1101	1100

Tabela 3.6: Primjer prefiksnog kodiranja

+ 9000*4 + 5000*4 = 224 000 binarnih cifara. Dakle zapis smo komprimovali 4 puta, na 25% dužine u odnosu na ASCII kod. U praksi komprimovanje je oko 2 puta, na 50% početne dužine.

Hafmanovo kodiranje je prefiksno, zato nije potrebna nikakva dodatna informacija za dekodiranje teksta izuzev samog šablona. Šablon se obično uključuje u samu kodiranu poruku, najčešće na njen početak.

3.6 Zadaci za vježbu

1. Cio broj N može da bude prikazan u obliku potpunog komplementa (PK) i uz upotrebu dekadnih cifara. Uzmimo da je fiksirana veličina prostora za prikazivanje broja jednaka 9 dekadnih mjesta. Po analogiji sa binarnim slučajem, na prvom mjestu piše 0 ako je $N \geq 0$ i piše 9 ako je $N < 0$. Na primjer, zapis broja $N = -23$ glasi $N^* = 999999977$. Vidi se da je dozvoljeni interval $-10^8 \leq N \leq 10^8 - 1$. Dokazati da je $N^* = N$ ako je $N \geq 0$ i $N^* = 10^9 + N$ ako je $N < 0$.

2. Prilikom sabiranja poništavaju se prenosi preko devetog mjesta. Dokazati da se može izvršiti sabiranje niza brojeva čak i u slučaju da međurezultati leže van dozvoljenog intervala, ako se zna da krajnji rezultat leži u tom intervalu.

3. Formulisati pravilo za množenje PK zapisa sa 10^k, kao i za dijeljenje sa 10^k. Ako je $N^* = 999006573$ onda broj $N : 10^3$ ima zapis 999999006.

4. Ako se za prikazivanje broja koristi prostor čija je veličina 18 dekadnih mjesta onda se kaže da je to broj dvostruke dužine. Sada je recimo $N^* = \underbrace{99\ldots9}_{16}77$ i $N^* = 10^{18} + N$ ako je $N < 0$. Formulisati i dokazati pravilo za konverziju iz jednostruke dužine u dvostruku, kao i za konverziju u suprotnom smjeru.

5. Prilikom množenja dva broja uobičajeno je da se faktori N_1 i N_2 prikazuju kao brojevi jednostruke dužine, a rezultat $N_3 = N_1 \cdot N_2$ kao broj dvostruke

dužine. Na primjer, ako je $N_1 = -5$ i $N_2 = 7$ onda se u memoriji drže vrijednosti $N_1^* = 10^9 - 5$ i $N_2^* = 7$. Njihovim množenjem kao prirodnih brojeva dobiće se $(10^9 - 5) \cdot 7 = 10^{18} - 35 - 10^9(10^9 - 7)$ kao rezultat. S druge strane, kako je očito $N_3 = -35$ to pravilna vrijednost rezultata treba da bude $N_3^* = 10^{18} - 35$. Formulisati pravilo za korekciju rezultata u slučaju da su faktori različitog znaka i u slučaju da su oba negativna.

6. Neka je $-2^{15} \leq N \leq 2^{15} - 1$ i neka je $c_{15}c_{14}\ldots c_1c_0$ zapis broja N u obliku potpunog komplementa, gdje $c_k \in \{0, 1\}$ za $0 \leq k \leq 15$. Stavimo $N^* = (c_{15}c_{14}\ldots c_1c_0)_2$. Dokazati formulu: $N^* = N$ ako je $N \geq 0$ i $N^* = 2^{16} + N$ ako je $N < 0$. Tako da je uvijek $N^* - N \equiv 0 \pmod{2^{16}}$. Takođe zapaziti da je $N = -2^{15} + (c_{14}\ldots c_1c_0)_2$ ako je $N < 0$ tj. $c_{15} = 1$.

7. Neka je $-2^{15} \leq N_1 \leq 2^{15} - 1$, $-2^{15} \leq N_2 \leq 2^{15} - 1$ i $-2^{15} \leq N_1 + N_2 \leq 2^{15} - 1$. Dokazati relaciju $(N_1 + N_2)^* \equiv (N_1^* + N_2^*) \pmod{2^{16}}$. U dokazu koristiti formulu iz prethodnog zadatka. Na navedenoj relaciji temelji se algoritam za sabiranje u potpunom komplementu.

8. Neka je $-2^{15} \leq N_1 \leq 2^{15} - 1$, $-2^{15} \leq N_2 \leq 2^{15} - 1$ i $-2^{15} \leq N_1 \cdot N_2 \leq 2^{15} - 1$. Dokazati relaciju $(N_1 \cdot N_2)^* \equiv (N_1^* \cdot N_2^*) \pmod{2^{16}}$. Na navedenoj relaciji temelji se algoritam za množenje u potpunom komplementu.

9. O relativnoj greški prilikom unošenja realnog broja u memoriju računara. Neka je $x > 0$ broj koji može da bude prikazan u obliku pokretnog zareza i neka je x^* vrijednost njegovog zapisa. Obično je $x^* \neq x$, kao što pokazuje raniji primjer $x = 76{,}8$. Ocijeniti relativnu grešku $\frac{x - x^*}{x}$.

10. Neka je x slučajno izabrani zapis u obliku pokretnog zareza takav da je $0 < x < 1$. Izračunati matematičko očekivanje (prosječnu vrijednost) veličine x. Zapaziti da nije $m_1 = m_2 = m_3 = m_4 = 0$ i da je $-64 \leq E \leq 0$.

11. U bilo kom programskom paketu svakako da mogu da se prikazuju brojevi, na razne načine i sa raznim opsezima. Utvrditi eksperimentalnim sredstvima koje su metode korišćene, koji je najmanji a koji najveći pozitivan i negativan broj koji može da bude prikazan. Isto tako, za realne brojeve utvrditi sa kolikom preciznošću se brojevi prikazuju, odnosno čemu je jednak broj sigurnih dekadnih cifara.

Glava 4

Algebra logike

4.1 Pojam Bulove funkcije

Označimo sa E_2 skup $\{0,1\}$. Za prirodan broj n ($n>0$) sa E_2^n označavamo Dekartov proizvod

$$E_2^n = \underbrace{E_2 \times E_2 \times E_2 \times \ldots \times E_2}_{n}.$$

Dakle, skup E_2^n sadrži uređene n-torke, čije su komponente nule i jedinice. Na primjer, n-torke $\underbrace{(0,0,0,\ldots,0)}_{n}$, $\underbrace{(0,1,1,\ldots,1)}_{n}$ i $\underbrace{(1,1,1,\ldots,1)}_{n}$ su elementi tog skupa.

Bulova funkcija ili *funkcija algebre logike* je funkcija koja preslikava skup E_2^n u E_2:

$$f : E_2^n \to E_2$$

Bulova funkcija definisana je nad n-torkama nula i jedinica a vrijednost funkcije na nekoj n-torci takođe je nula ili jedan.

Ako uvedemo promjenljive $x_1, x_2, \ldots, x_n$ koje uzimaju vrijednost iz skupa E_2, Bulovu funkciju možemo zapisati sa

$$f(x_1, x_2, \ldots, x_n).$$

Tada možemo jedan element iz domena Bulove funkcije predstaviti kao uređenu n-torku

$$(\alpha_1, \alpha_2, \ldots, \alpha_n),$$

gdje $\alpha_i \in E_2$, a vrijednost funkcije na njemu kao

$$f(\alpha_1, \alpha_2, \ldots, \alpha_n).$$

Recimo, vrijednost funkcije na n-torci $(0, 1, \ldots, 1)$ je $f(0, 1, \ldots, 1)$ i može biti jednako 0 ili 1.

Kako je skup $n-$torki sa komponentama 0 ili 1 konačan, domen Bulove funkcije je konačan. Kako $n-$torki nula i jedinica ima $\underbrace{2 \cdot 2 \cdot 2 \cdot \ldots \cdot 2}_{n} = 2^n$, zaključujemo da domen Bulove funkcije ima 2^n elemenata.

Konačnost domena implicira da se Bulova funkcija može predstaviti tabelom koja pored zaglavlja ima 2^n redova. Da se zapiše argument funkcije treba n kolona, dok je još jedna kolona potrebna za vrijednost funkcije. U zaglavlju je prirodno napisati promjenljive $x_1, x_2, \ldots, x_n$ i oznaku za vrijednost funkcije $f(x_1, x_2, \ldots, x_n)$.

Da bi predstavljanje Bulove funkcije tabelom bilo univerzalno moramo još definisati poredak u kom ćemo pisati n-torke. Uočimo da se n-torke nula i jedinica mogu posmatrati i kao binarni n-tocifreni zapisi brojeva od 0 do 2^{n-1}, recimo $\underbrace{(0, 0, \ldots, 0, 1, 1)}_{n}$ možemo posmatrati kao binarni broj $\underbrace{00\ldots011}_{n}$ odnosno 3. Tada se prirodno nameće zaključak da je zgodno n-torke u tabeli poređati u poretku rasta brojeva koje one predstavljaju.

Dakle, Bulovu funkciju zapisujemo na sljedeći način:

x_1	x_2	...	x_n	$f(x_1, x_2, \ldots, x_n)$
0	0	...	0	$f(0, 0, \ldots, 0)$
0	0	...	1	$f(0, 0, \ldots, 1)$
...	...	...	...	...
1	1	...	1	$f(1, 1, \ldots, 1)$

ili

x_1	x_2	...	x_n	$f(x_1, x_2, \ldots, x_n)$
0	0	...	0	$f(0, 0, \ldots, 0)$
0	0	...	1	$f(0, 0, \ldots, 1)$
...	...	...	...	...
1	1	...	1	$f(1, 1, \ldots, 1)$

Kada smo funkcije zapisali na ovaj način, vidimo da je za definiciju funkcije sa n argumenata dovoljno samo dati krajnju desnu kolonu, tj. vektor

$$f(0, 0, \ldots, 0), f(0, 0, \ldots, 1), \ldots, f(1, 1, \ldots, 1).$$

Ovaj vektor sadrži 2^n elemenata koji su jednaki 0 ili 1. Takvih različitih vektora ima 2^{2^n}. Time je dokazana sljedeća teorema.

Teorema 1. Skup Bulovih funkcija sa n argumenata ima 2^{2^n} elemenata.

Skup Bulovih funkcija sa n argumenata označavamo sa P_n a njihov broj sa p_n. Dakle, $p_n = 2^{2^n}$.

Ubuduće ćemo sa P_2 označavati skup svih Bulovih funkcija.

Bulove funkcije nose ime po matematičaru Džoržu Bulu (George Boole), koji ih je uveo 1854. godine u djelu "An Investigation of the Laws of Thought".

4.2 Primjeri Bulovih funkcija

Postoje određene Bulove funkcije koje imaju čestu i važnu primjenu, pa se zato nazivaju *elementarnim funkcijama*. Navešćemo ovdje neke od njih.

4.2.1 Konstante

Bulova funkcija može imati istu vrijednost na svim vrijednostima iz domena. Takve su, na primjer, funkcije zadate tabelama na slici 4.1.

Ovakve funkcije nazivaju se *konstantnim funkcijama*. Naveli smo primjere konstanti sa 1, 2 i 3 argumenta. Konstantnu nula funkciju možemo zapisati sa $f(x_1) = 0$, $f(x_1, x_2) = 0$, $f(x_1, x_2, x_3) = 0$, ili, u opštem slučaju, $f(x_1, x_2, \ldots, x_n) = 0$. Kada nam u kontekstu nije bitan broj argumenata, pišemo

$$f \equiv 0.$$

Slično, funkcije $f(x_1) = 1$, $f(x_1, x_2) = 1$, $f(x_1, x_2, x_3) = 1$ možemo zapisati sa

$$f \equiv 1.$$

x_1	$f(x_1)$
0	0
1	0

x_1	$f(x_1)$
0	1
1	1

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	0
1	0	0
1	1	0

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	1
1	1	1

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Slika 4.1: Konstante

4.2.2 Identičko preslikavanje

Ova funkcija prikazana je tabelom na slici 4.2. Ona ima jedan argument, a vrijednost funkcije jednaka je vrijednosti argumenta. Zato je nazivamo *identičkim preslikavanjem* i označavamo sa

$$f(x) = x.$$

x	$f(x)$
0	0
1	1

Slika 4.2: Identičko preslikavanje

4.2.3 Negacija

I ova funkcija (slika 4.3) ima jedan argument. Vrijednost funkcije je suprotna vrijednosti argumenta, odnosno ona je njegova *negacija*. Zato funkciju nazivamo *negacijom* i označavamo sa

$$f(x) = \overline{x}.$$

x	$f(x)$
0	1
1	0

Slika 4.3: Negacija

Druga oznaka je $f(x) = \neg x$. Ova se funkcija čita kao "ne x"

4.2.4 Konjunkcija

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	0
1	0	0
1	1	1

Slika 4.4: Konjunkcija

Vrijednost ove funkcije, koja ima dva argumenta, je jedan samo kada su oba argumenta jedan. Ova funkcija naziva se *konjunkcijom* ili *logičkim množenjem* i označava sa

$$f(x_1, x_2) = x_1 \& x_2.$$

Ovo čitamo kao "x_1 i x_2". Ponekad se ova funkcija označava i sa

$$f(x_1, x_2) = x_1 \cdot x_2$$

ili

$$f(x_1, x_2) = x_1 x_2.$$

4.2.5 Disjunkcija

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	1

Slika 4.5: Disjunkcija

Vrijednost funkcije na slici 4.5 jednaka je jedan samo kada je bilo koji od argumenata jedan. Ona se naziva *disjunkcijom* ili *logičkim sabiranjem* i označava sa

$$f(x_1, x_2) = x_1 \vee x_2.$$

Čita se "x_1 ili x_2".

4.2.6 Implikacija

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	0
1	1	1

Slika 4.6: Implikacija

Ova funkcija (slika 4.6) se označava sa

$$f(x_1, x_2) = x_1 \rightarrow x_2$$

i naziva *implikacijom* ili *logičkom posljedicom*. Čita se "iz x_1 slijedi x_2".

4.2.7 Ekskluzivna disjunkcija

Ova funkcija je slična disjunkciji sa tim izuzetkom što je jednaka 0 i u slučaju kada su oba argumenta 1. Ona se naziva *ekskluzivnom disjunkcijom* i označava sa

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

Slika 4.7: Ekskluzivna disjunkcija

$$f(x_1, x_2) = x_1 \oplus x_2.$$

Ponekad se naziva i *sabiranjem po modulu 2* i označava sa

$$f(x_1, x_2) = x_1 + x_2.$$

4.2.8 Šeferova funkcija

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	1
1	1	0

Slika 4.8: Šeferova funkcija

Ova funkcija suprotna je konjunkciji. Naziva se *Šeferovom funkcijom* i označava sa

$$f(x_1, x_2) = x_1 \uparrow x_2 \text{ ili } f(x_1, x_2) = x_1 | x_2.$$

4.3 Pojam stvarne i fiktivne promjenljive

Funkcija $f(x_1, \ldots, x_{i-1}, x_i, x_{i+1} \ldots, x_n)$ *stvarno zavisi od promjenljive* x_i ako postoje vrijednosti $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1} \ldots, \alpha_n$ iz {0,1} promjenljivih $x_1, \ldots, x_{i-1}, x_{i+1} \ldots, x_n$, takve da je

$$f(\alpha_1, \ldots, \alpha_{i-1}, 0, \alpha_{i+1} \ldots, \alpha_n) \neq f(\alpha_1, \ldots, \alpha_{i-1}, 1, \alpha_{i+1} \ldots, \alpha_n).$$

Tada promjenljivu x_i nazivamo *stvarnom*. Ako promjenljiva nije stvarna onda je *fiktivna*.

Posmatrajmo funkciju definisanu sljedećom tablicom (slika 4.9).

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1
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	1

Slika 4.9: Primjer fiktivne i stvarne promjenljive

Za vrijednost (α_2, α_3)=(0,0) promjenljivih x_2, x_3 važi da je

$$1 = f(0, \alpha_2, \alpha_3) \neq f(1, \alpha_2, \alpha_3) = 0,$$

što znači da je promjenljiva x_1 stvarna. Slično, za vrijednost (α_1, α_3)=(0,0) promjenljivih x_1, x_3 važi da je

$$1 = f(\alpha_1, 0, \alpha_3) \neq f(\alpha_1, 1, \alpha_3) = 0,$$

pa je i promjenljiva x_2 stvarna. Međutim, za svaku vrijednost α_1, α_2 iz {0,1} promjenljivih x_1, x_2 važi da je

$$f(\alpha_1, \alpha_2, 0) = f(\alpha_1, \alpha_2, 1).$$

Zato je promjenljiva x_3 fiktivna.

Fiktivna promjenljiva nema nikakvog uticaja na vrijednost funkcije. Zato se ona može potpuno isključiti iz definicije funkcije. Ako je x_i fiktivna promjenljiva možemo izbaciti sve redove oblika $\alpha_1, \ldots, \alpha_{i-1}, 1, \alpha_{i+1} \ldots, \alpha_n$ (jer redovi $\alpha_1, \ldots, \alpha_{i-1}, 0, \alpha_{i+1} \ldots, \alpha_n$ sadrže istu informaciju) i nakon toga izbaciti kolonu označenu sa x_i.

Na primjer, izbacivanjem promjenljive x_3 iz prethodne funkcije dobijamo funkciju prikazanu na slici 4.10.

x_1	x_2	$f(x_1, x_2, x_3)$
0	0	1
0	1	0
1	0	0
1	1	1

Slika 4.10: Uklanjanje fiktivne promjenljive

Prirodno je smatrati da su ove dvije funkcije jednake. Dakle, za funkcije f i g kažemo da su *jednake* ako se g može dobiti iz f dodavanjem i uklanjanjem fiktivnih argumenata.

Konstante funkcije 0 i 1 nemaju stvarnih argumenata, zato se iz njih mogu ukloniti svi argumenti.

Uvedimo još jednu definiciju. Funkcija f je *simetrična* u odnosu na argumente $x_1, \ldots, x_k$, $(2 \leq k \leq n)$, ako za svaku permutaciju

$$1 \to j_1, 2 \to j_2, \ldots, k \to j_k$$

važi jednakost:

$$f(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n) = f(x_{j_1}, \ldots, x_{j_k}, x_{k+1}, \ldots, x_n).$$

Analogno se uvodi pojam simetričnosti funkcije u odnosu na proizvoljne promjenljive $x_{i_1}, \ldots, x_{i_k}$.

Na primjer, konstante 0 i 1 su simetrične u odnosu na sve promjenljive.

4.4 Formule. Realizacija funkcija formulama

Kao i u elementarnoj algebri, koristeći elementarne funkcije mogu se definisati formule. Definicija pojma formule je induktivna.

Neka je B neki (ne obavezno konačan) podskup funkcija iz P_2.

1. (*baza indukcije*) Svaka funkcija $f(x_1, \ldots, x_n)$ iz B je *formula nad* B.

2. (*induktivni prelaz*) Neka je $f_0(x_1, \ldots, x_n)$ funkcija iz B i $A_1, A_2, \ldots, A_n$ izrazi koji su ili formule ili promjenljive. Tada je izraz $f_0(A_1, A_2, \ldots, A_n)$ *formula nad* B.

Primjer. Neka je B skup elementarnih funkcija iz primjera 1 do 8. Tada su formule nad B:

$$\overline{(x_1 \vee (x_2 \to x_3)(x_3 \to x_2))}$$

$$(((x_1 \& x_2) + x_1) + x_2)$$

$$(\overline{x}_1 \& (x_2 + x_3)).$$

Sa $\Phi(f_1, \ldots, f_n)$ označavamo formulu koja je izgrađena od funkcija $f_1, \ldots, f_n$. Ako su nam bitne promjenljive, onda pišemo $\Phi(x_1, \ldots, x_n)$.

Ako je Φ formula nad B, tada sve formule koje su upotrebljene za izgradnju te formule nazivamo njenim *podformulama*. Na primjer $(x_1 \& x_2)$ je podformula formule $(((x_1 \& x_2) + x_1) + x_2)$.

Neka je Φ formula nad skupom $B = \{f_1(x_1, \ldots, x_n), \ldots, f_s(x_1, \ldots, x_n)\}$ i neka je $B_1 = \{g_1(x_1, \ldots, x_n), \ldots, g_s(x_1, \ldots, x_n)\}$. Kažemo da formula $\Gamma(g_1, \ldots, g_s)$, koja je dobijena iz formule $\Phi(f_1, \ldots, f_s)$ zamjenom f_i sa g_i, ima *istu strukturu* kao formula Φ.

Na primjer, ako je

$$B = \{\overline{x}_1,\ (x_1 \& x_2)\},\ \Phi = (x_1 \& \overline{(x_2 \& x_3)})$$

$$B_1 = \{\overline{x}_1,\ (x_1 \to x_2)\},\ \Gamma = (x_1 \to \overline{(x_2 \to x_3)}),$$

očigledno je da formule Φ i Γ imaju istu strukturu.

Strukturu formule označavamo sa C. Formula je jednoznačno određena strukturom C i nizom funkcija $f_1, \ldots, f_s$, pa pišemo $\Phi = C[f_1, \ldots, f_s]$.

Pridružimo svakoj formuli $\Phi(x_1, \ldots, x_s)$ nad B jednu funkciju iz P_2, polazeći od induktivne definicije formule:

(Baza indukcije) Ako je formula $\Phi(x_1, \ldots, x_s) = f(x_1, \ldots, x_s)$, $f \in B$, tada formuli Φ pridružimo funkciju f.

(Induktivni korak) Neka je $\Phi(x_1, \ldots, x_s) = f_0(A_1, A_2, \ldots, A_m)$, gdje je A_i ili formula ili simbol promjenljive. Tada je svakom A_i pridružena neka funkcija iz P_2 (ako je A_i formula) ili identička funkcija $f_i = x_{j(i)}$ (ako je A_i promjenljiva). Formuli Φ tada pridružujemo funkciju $f(x_1, \ldots, x_s) = f_0(f_1, f_2, \ldots, f_m)$.

Ako formuli Φ pridružujemo funkciju f_Φ, kažemo i da ta funkcija *realizuje* formulu Φ. Funkciju f_Φ nazivamo i *superpozicijom* funkcija iz B, a proces dobijanja te funkcije iz B nazivamo *operacijom superpozicije*.

Skup svih Bulovih funkcija zajedno sa operacijom superpozicije nad funkcijama naziva se *Bulovom algebrom* ili *algebrom logike* i čini, dakle, jednu algebarsku strukturu.

Kao primjer superpozicije, pridružimo formuli $(((x_1 \& x_2) + x_1) + x_2)$ funkciju. Imamo sljedeće podformule: $(x_1 \& x_2)$, $((x_1 \& x_2) + x_1)$ i konačno $(((x_1 \& x_2) + x_1) + x_2)$. Operaciju superpozicije možemo ilustrovati sljedećom tablicom (slika 4.11).

x_1	x_2	$x_1 \& x_2$	$((x_1 \& x_2) + x_1)$	$(((x_1 \& x_2) + x_1) + x_2)$
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	0	1

Slika 4.11: Operacija superpozicije

Superpozicija je ovdje funkcija od dva argumenta, $g(x_1, x_2)$, i iz tabele vidimo da je $g(x_1, x_2) = x_1 \vee x_2$.

Kao drugi primjer, pridružimo funkciju formuli $(\overline{x}_1 \& (x_2 + x_3))$ (slika 4.12).

Pridružimo funkciju i formuli $\overline{(x_1 \vee (x_2 \to x_3)(x_3 \to x_2))}$. Ne moramo pisati cijelu tablicu, do funkcije f možemo doći razmatranjem slučajeva kada je njena vrijednost 1. Očigledno, ona je jednaka 1 kada je vrijednost formule $(x_1 \vee (x_2 \to x_3)(x_3 \to x_2))$ nula. To važi ako je $x_1 = 0$ i $(x_2 \to x_3)(x_3 \to x_2) = 0$. Formula $(x_2 \to x_3)(x_3 \to x_2)$ ima vrijednost nula ako je $x_2 \to x_3 = 0$ ili $x_3 \to x_2 = 0$. Iz definicije implikacije vidimo da to važi kada je $x_2 = 1$ i $x_3 = 0$ ili kada je $x_2 = 0$ i $x_3 = 1$. Zaključujemo da funkcija f ima vrijednost 1 na trojkama (0,0,1) i (0,1,0). Vidimo da je ova funkcija ista kao funkcija dobijena u prethodnom primjeru.

x_1	x_2	x_3	$\overline{x_1}$	x_2+x_3	$(\overline{x}_1\&(x_2+x_3))$
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	0	0	0

Slika 4.12: Pridruživanje funkcije formuli $(\overline{x}_1\&(x_2+x_3))$

Ovdje definisani jezik formula pogodan je za zapisivanje raznih uslova ili iskaza. Navedimo dva primjera korišćenja Bulovih funkcija, koja to ilustruju.

Primjer. Razmotrimo sabiranje binarnih brojeva sa n cifara:

$$\begin{aligned} & x_n \ldots x_2 x_1 \\ + \; & y_n \ldots y_2 y_1 \\ = & z_{n+1} z_n \ldots z_2 z_1 \end{aligned}$$

Treba izraziti cifre rezultata kroz cifre sabiraka. Uvedimo pomoćne veličine p_i koje označavaju prenos sa pozicije i na poziciju $i+1$. Uzevši u obzir da je x_{n+1}=0, y_{n+1}=0 i $p_0 = 0$ zaključujemo da je

$$z_i = ((x_i + y_i) + p_{i-1}),$$

za $i = 1, \ldots, n+1$.

Veličina p_i definiše se preko prenosa sa pozicije i na poziciju $i+1$: "prenos postoji onda kada su bar dvije veličine od x_i, y_i, p_{i-1} jednake 1". Ovo se korišćenjem elementarnih Bulovih funkcija može zapisati na sljedeći način

$$p_i = ((x_i \& y_i) \vee (x_i \& p_{i-1}) \vee (y_i \& p_{i-1})),$$

ili kraće

$$p_i = ((x_i y_i) \vee (x_i p_{i-1}) \vee (y_i p_{i-1})),$$

za $i = 1, \ldots, n$.

Primjer. Neka u ulazu imamo 3 lifta, prizemlje i n spratova. Zadatak je kako definisati funkciju koja sa prizemlja poziva najbliži slobodan lift. Uvedimo $3n+3$ promjenljivih

$$x_o, \ldots, x_n, y_o, \ldots, y_n, z_o, \ldots, z_n.$$

Promjenljiva x_i označava da je prvi lift slobodan i da se nalazi na i-tom spratu; slično za y_i i z_i.

Razmotrimo sada h_1 – funkciju poziva prvog lifta. Ona ima vrijednost 1 ako prvi lift je slobodan i nema slobodnih liftova na spratovima nižim od onog na kojem se nalazi prvi lift. Analogno definišemo funkcije h_2 i h_3.

Poziv prvog lifta sa prizemlja

$$h_1(x_o, \ldots, x_n, y_o, \ldots, y_n, z_o, \ldots, z_n) = x_o \vee ((\overline{y}_o \& \overline{z}_o) \& (x_1 \vee (\overline{y}_1 \& \overline{z}_1)) \& \ldots)$$

Slično se određuju funkcije poziva drugog i trećeg lifta sa prizemlja:

$$h_2(x_o, \ldots, x_n, y_o, \ldots, y_n, z_o, \ldots, z_n) = y_o \vee ((\overline{x}_o \& \overline{z}_o) \& (y_1 \vee (\overline{x}_1 \& \overline{z}_1)) \& \ldots)$$

$$h_3(x_o, \ldots, x_n, y_o, \ldots, y_n, z_o, \ldots, z_n) = z_o \vee ((\overline{x_o} \& \overline{y_o}) \& (z_1 \vee (\overline{x_1} \& \overline{y_1})) \& \ldots)$$

4.5 Ekvivalentnost formula i osobine elementarnih funkcija

Formule Φ i Γ su *ekvivalentne*, u oznaci $\Phi = \Gamma$, ako je $f_\Phi = f_\Gamma$. Drugačije rečeno, formule su ekvivalente ako su im pridružene iste funkcije.

Na primjer,

$$0 = (x \& \overline{x})$$

$$(x \to y) = (\overline{y} \to \overline{x})$$

$$\overline{(x_1 \vee (x_2 \to x_3)(x_3 \to x_2))} = \overline{x}_1 (x_2 + x_3).$$

Navedimo nekoliko ekvivalentnosti (jednakosti), koje karakterišu elementarne funkcije. Svaka od njih može se lako dokazati putem pridruživanja funkcija lijevoj i desnoj strani.

Označimo sa $(x \circ y)$ proizvoljnu od funkcija $(x\&y)$, $(x \vee y)$ i $(x + y)$. U jednoj jednakosti $(x \circ y)$ predstavlja jednu funkciju. Tada važe sljedeće jednakosti (*komutativnost* i *asocijativnost*):

1. $(x \circ y) = (y \circ x)$.
2. $((x \circ y) \circ z) = (x \circ (y \circ z))$.

Za konjunkciju i disjunkciju važe zakoni distributivnosti:

3. $((x \vee y)\,\&\,z) = ((x\&z) \vee (y\&z))$.
4. $((x\&y) \vee z) = ((x \vee z)\,\&\,(y \vee z))$.

Za negaciju, konjunkciju i disjunkciju važe:

5. $\overline{\overline{x}} = x, \qquad \overline{x \vee y} = \overline{x}\&\overline{y}, \qquad \overline{x\&y} = \overline{x} \vee \overline{y}$.

Posljednja dva su *De Morganovi* zakoni.

Za konjunkciju i disjunkciju važe i sljedeća svojstva:

6. $\overline{x} \vee x = 1, \qquad x \vee x = x, \qquad x \vee 0 = x, \qquad x \vee 1 = 1,$
7. $\overline{x}\&x = 0, \qquad x\&x = x, \qquad x\&0 = 0, \qquad x\&1 = x.$

Zapis formule u nekim slučajevima možemo skratiti ne pišući zagrade. Recimo, ako uvedemo pravilo da operacija & ima veći prioritet od $\vee$. Tada zapis $(x\&y \vee z)$ označava formulu $((x\&y) \vee z)$. Slično, zbog asocijativnosti, zapis $(x \circ y \circ z)$ označava $((x \circ y) \circ z)$ ili $(x \circ (y \circ z))$. Ovaj zapis nije formula ali se lako prevodi u formulu dopisivanjem zagrada.

U formulama u kojim je spoljašnja funkcija konjunkcija, disjunkcija, sabiranje, implikacija ili funkcija Šefera, spoljašnje zagrade se obično ne stavljaju. Na primjer, zapis $x \to y$ označava formulu $(x \to y)$. Slično važi i za negaciju: dio formule koji se negira ne mora biti u zagradama. Tako zapis $\overline{x \to y}$ označava formulu $\overline{(x \to y)}$.

U daljem izlaganju ćemo često koristiti sljedeće oznake:

$$\mathop{\&}_{i=1}^{n} x_i = x_1\& \ldots \&x_n$$

$$\mathop{\vee}_{i=1}^{n} x_i = x_1 \vee \ldots \vee x_n.$$

Ove oznake imaju smisla i kada je $n = 1$.

Očigledno je da ako je Φ_1 podformula formule Φ, tada ako svako pojavljivanje te podformule u Φ zamijenimo ekvivalentnom formulom Γ_1, dobijamo novu formulu Γ, ekvivalentnu formuli Φ.

Korišćenjem ovog zaključka i prethodnih pravila, lako je dobijati nove jednakosti. Na primjer,

$$x_1 \vee x_1x_2 = x_1 \cdot 1 \vee x_1x_2 = x_1 \cdot (x_2 \vee \overline{x}_2) \vee x_1x_2 =$$
$$x_1x_2 \vee x_1\overline{x}_2 \vee x_1x_2 = x_1x_2 \vee x_1x_2 \vee x_1\overline{x}_2 =$$
$$x_1x_2 \vee x_1\overline{x}_2 = x_1(x_2 \vee \overline{x}_2) = x_1 \cdot 1 = x_1.$$

Na ovaj način dobili smo novu jednakost $x_1 \vee x_1x_2 = x_1$.

4.6 Princip dualnosti

Osim korišćenja ekvivaletnosti formula, postoji još jedan način dobijanja jednakosti, zasnovan na tzv. *principu dualnosti.*

Funkcija $f^*(x_1, \ldots, x_n) = \overline{f}(\overline{x}_1, \ldots, \overline{x}_n)$ naziva se *dualnom funkcijom* funkciji $f(x_1, \ldots, x_n)$. Iz ove definicije lako je zaključiti da se tablica dualne funkcije dobija iz tablice funkcije f tako što se kolona za vrijednost funkcije komplementira (tj. 0 zamijeni sa 1, a 1 sa 0), a zatim ta kolona 'preokrene'. Primjer funkcije i njoj dualne funkcije dat je na slici 4.13.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$	$f^*(x_1, x_2, x_3)$
0	0	0	1	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	1
1	0	1	1	1
1	1	0	0	0
1	1	1	1	0

Slika 4.13: Dualna funkcija

Lako možemo vidjeti da je

- funkcija 0 dualna funkciji 1,
- funkcija 1 dualna funkciji 0,
- funkcija x dualna funkciji x,
- funkcija $\overline{x}$ dualna funkciji $\overline{x}$,
- funkcija $x\&y$ dualna funkciji $x \vee y$,
- funkcija $x \vee y$ dualna funkciji $x\&y$.

Iz definicije dualnosti slijedi

$$f^{**} = (f^*)^* = f,$$

odnosno funkcija je dualna svojoj dualnoj funkciji (*svojstvo uzajamnosti*).

Prirodno se postavlja pitanje: ako je funkcija $f(x_1, \ldots, x_n)$ izražena formulom Φ, kakav oblik ima formula koja realizuje funkciju $f^*(x_1, \ldots, x_n)$? Označimo sa $x_1, \ldots, x_n$ sve različite promjenljive koje se pojavljuju kao argumenti funkcija

$$f_1(x_{11}, \ldots, x_{1p_1}), \ldots, f_m(x_{m1}, \ldots, x_{mp_m}).$$

Teorema 2. Ako je

$$h(x_1, \ldots, x_n) = f(f_1(x_{11}, \ldots, x_{1p_1}), \ldots, f_m(x_{m1}, \ldots, x_{mp_m})),$$

tada je

$$h^*(x_1, \ldots, x_n) = f^*(f_1^*(x_{11}, \ldots, x_{1p_1}), \ldots, f_m^*(x_{m1}, \ldots, x_{mp_m})).$$

Dokaz slijedi iz sljedećih jednakosti

$$\begin{aligned}
&h^*(x_1, \ldots, x_n) = \overline{h}(\overline{x}_1, \ldots, \overline{x}_n) = \\
&\overline{f}(f_1(\overline{x}_{11}, \ldots, \overline{x}_{1p_1}), \ldots, f_m(\overline{x}_{m1}, \ldots, \overline{x}_{mp_m})) = \\
&\overline{f}(\overline{f}_1(\overline{x}_{11}, \ldots, \overline{x}_{1p_1}), \ldots, \overline{f_m}(\overline{x}_{m1}, \ldots, \overline{x}_{mp_m})) = \\
&\overline{f}(\overline{f_1^*}(x_{11}, \ldots, x_{1p_1}), \ldots, \overline{f_m^*}(x_{m1}, \ldots, x_{mp_m})) = \\
&f^*(f_1^*(x_{11}, \ldots, x_{1p_1}), \ldots, f_m^*(x_{m1}, \ldots, x_{mp_m})).
\end{aligned}$$

Iz Teoreme 2, kao direktna posljedica slijedi *princip dualnosti*:

Ako formula

$$\Phi = C[f_1, \ldots, f_s]$$

realizuje funkciju f, tada formula

$$\Phi^* = C[f_1^*, \ldots, f_s^*]$$

*realizuje funkciju f *.*

Dakle, formula Φ^* dobijena zamjenom funkcija $f_1, \ldots, f_s$ redom funkcijama $f_1^*, \ldots, f_s^*$ u formuli Φ, realizuje funkciju dualnu funkciji koju realizuje Φ.

Za formule nad skupom $B = \{0, 1, \overline{x}, x\&y, x \vee y\}$, dualna formula se dobija tako što se na svim mjestima zamijeni 0 sa 1, 1 sa 0, & sa $\vee$ i $\vee$ sa &. Ovo možemo iskazati i na sljedeći način:

Ako je $\Phi = C[0, 1, \overline{x}, x\&y, x \vee y]$ tada je $\Phi^* = C[1, 0, \overline{x}, x \vee y, x\&y]$.

Na primjer, ako je $\Phi = x\&y$, tada je $\Phi^* = x \vee y$. Slično, ako je $\Phi = xy \vee \overline{x}\overline{y}$, tada je $\Phi^* = (x \vee y)(\overline{x} \vee \overline{y})$.

Na kraju, iz principa dualnosti slijedi da ako je

$$\Phi(x_1, \ldots, x_n) = \Gamma(x_1, \ldots, x_n),$$

tada je

$$\Phi^*(x_1, \ldots, x_n) = \Gamma^*(x_1, \ldots, x_n).$$

Tako, na primjer, iz jednakosti $\overline{x\&y} = \overline{x} \vee \overline{y}$ slijedi jednakost $\overline{x \vee y} = \overline{x}\&\overline{y}$.

Princip dualnosti smanjuje za polovinu potrebu izvođenja jednakosti formula.

4.7 Savršena disjunktivna normalna forma

Do sada nismo razmotrili jedno izuzetno važno pitanje. Da li se svaka Bulova funkcija može predstaviti kao formula nad datim skupom funkcija B?

Uvodimo novu oznaku:

$$x^\sigma = x\sigma \vee \overline{x}\,\overline{\sigma},$$

gdje je σ parametar jednak 0 ili 1.

Lako se provjerava da je

$$x^\sigma = \begin{cases} \overline{x}, & \text{ako je } \sigma = 0, \\ x, & \text{ako je } \sigma = 1. \end{cases}$$

Isto tako možemo zaključiti $x^\sigma = 1$ ako i samo ako je $x = \sigma$.

Teorema 3. (Razlaganje funkcije po promjenljivim) Svaku funkciju algebre logike $f(x_1, \ldots, x_n)$ za proizvoljno m $(1 \leq m \leq n)$ moguće je predstaviti u sljedećem obliku:

$$f(x_1, \ldots, x_n) = \bigvee_{(\sigma_1, \ldots, \sigma_m)} x_1^{\sigma_1} \& x_2^{\sigma_2} \& \ldots \& x_m^{\sigma_m} \& f(\sigma_1, \ldots, \sigma_m, x_{m+1}, \ldots, x_n), \quad (*)$$

gdje se suma uzima po svim mogućim m-torkama $\sigma_1, \ldots, \sigma_m$.

Dokaz. Razmotrimo proizvoljnu n-torku $\alpha_1, \ldots, \alpha_n$ vrijednosti promjenljivih $x_1, \ldots, x_n$. Pokazaćemo da lijeva i desna strana jednakosti (*) imaju istu vrijednost, odakle slijedi tvrđenje teoreme. Lijeva strana daje $f(\alpha_1, \ldots, \alpha_n)$, a desna

$$\bigvee_{(\sigma_1, \ldots, \sigma_m)} \alpha_1^{\sigma_1} \& \alpha_2^{\sigma_2} \& \ldots \& \alpha_m^{\sigma_m} \& f(\sigma_1, \ldots, \sigma_m, \alpha_{m+1}, \ldots, \alpha_n) =$$

$$= \alpha_1^{\alpha_1} \& \alpha_2^{\alpha_2} \& \ldots \& \alpha_m^{\alpha_m} \& f(\alpha_1, \ldots, \alpha_m, \alpha_{m+1}, \ldots, \alpha_n) = f(\alpha_1, \ldots, \alpha_n).$$

Kao posljedicu ove teoreme imamo dva specijalna slučaja razlaganja.

1. Razlaganje po jednoj promjenljivoj

$$f(x_1, \ldots, x_{n-1}, x_n) = x_n \& f(x_1, \ldots, x_{n-1}, 0) \vee \overline{x}_n \& f(x_1, \ldots, x_{n-1}, 1)$$

Funkcije $f(x_1, \ldots, x_{n-1}, 0)$ i $f(x_1, \ldots, x_{n-1}, 1)$ nazivaju se *komponentama razlaganja*. Ovo razlaganje je korisno kada se dokazuju svojstva Bulovih funkcija po indukciji.

2. Razlaganje po svim promjenljivim

$$f(x_1, \ldots, x_n) = \bigvee_{(\sigma_1, \ldots, \sigma_n)} x_1^{\sigma_1} \& \ldots \& x_n^{\sigma_n} \& f(\sigma_1, \ldots, \sigma_n)$$

Ako f nije konstanta 0, ovo razlaganje može se zapisati na sljedeći način:

$$\bigvee_{(\sigma_1, \ldots, \sigma_n)} x_1^{\sigma_1} \& \ldots \& x_n^{\sigma_n} \& f(\sigma_1, \ldots, \sigma_n) = \bigvee_{\substack{(\sigma_1, \ldots, \sigma_n) \\ f(\sigma_1, \ldots, \sigma_n) = 1}} x_1^{\sigma_1} \& \ldots \& x_n^{\sigma_n}.$$

Konačno dobijamo

$$f(x_1, \ldots, x_n) = \bigvee_{\substack{(\sigma_1, \ldots, \sigma_n) \\ f(\sigma_1, \ldots, \sigma_n) = 1}} x_1^{\sigma_1} \& \ldots \& x_n^{\sigma_n}.$$

Ovo razlaganje naziva se *savršenom disjunktivnom normalnom formom (SDNF)*.

Teorema 4. Svaka funkcija algebre logike može se predstaviti kao formula nad skupom funkcija koji se sastoji od negacije, konjunkcije i disjunkcije.

Dokaz.

Neka je $f(x_1, \ldots, x_n) \equiv 0$, tada je $f(x_1, \ldots, x_n) = x_1 \& \overline{x}_1$.

Neka je $f(x_1, \ldots, x_n) \neq 0$. Tada funkciju možemo predstaviti njenom savršenom DNF:

$$f(x_1, \ldots, x_n) = \bigvee_{\substack{(\sigma_1, \ldots, \sigma_n) \\ f(\sigma_1, \ldots, \sigma_n) = 1}} x_1^{\sigma_1} \& \ldots \& x_n^{\sigma_n}.$$

Dokaz je završen.

Teorema ima konstruktivni karakter. Ona daje način kako predstaviti proizvoljnu funkciju preko negacije, konjunkcije i disjunkcije. U tablici funkcije (koja nije konstanta 0) prvo uočimo redove $(\sigma_1, \ldots, \sigma_n)$ za koje je $f(\sigma_1, \ldots, \sigma_n) = 1$. Za svaki takav red napišemo logički proizvod $x_1^{\sigma_1} \& \ldots \& x_n^{\sigma_n}$ i na kraju sve takve proizvode povežemo znakom disjunkcije.

Na primjer, predstavimo implikaciju $f(x_1, x_2) = x_1 \to x_2$ savršenom DNF. Ova funkcija jednaka je 1 na tačno tri kombinacije binarnih parova (0,0), (0,1), (1,1), pa važi:

$$f(x_1, x_2) = x_1^0 \& x_2^0 \vee x_1^0 \& x_2^1 \vee x_1^1 \& x_2^1 = \overline{x}_1 \& \overline{x}_2 \vee \overline{x}_1 \& x_2 \vee x_1 \& x_2$$

x_1	x_2	x_3	$f(x_1,x_2,x_3)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Slika 4.14: Primjer za SDNF

Kao sljedeći primjer, nađimo savršenu DNF za funkciju zadatu tablicom na slici 4.14.

Dobijamo

$$f(x_1,x_2,x_3)=\overline{x}_1\&\overline{x}_2\&\overline{x}_3\vee\overline{x}_1\&\overline{x}_2\&x_3\vee x_1\&\overline{x}_2\&x_3\vee x_1\&x_2\&x_3.$$

Savršena DNF je izraz tipa disjunkcije konjunkcija. Postavlja se pitanje, da li je moguće dobiti razlaganje Bulovih funkcija putem konjunkcije disjunkcija? Pokažimo da je to moguće kada f nije konstanta 1. Tada je njoj dualna funkcija f^* različita od konstante 0 pa se može zapisati u obliku SDNF:

$$f^*(x_1,\ldots,x_n)=\bigvee_{\substack{(\sigma_1,\ldots,\sigma_n)\\ f^*(\sigma_1,\ldots,\sigma_n)=1}} x_1^{\sigma_1}\&\ldots\&x_n^{\sigma_n}.$$

Korišćenjem svojstva dualnosti, imamo

$$f^{**}(x_1,\ldots,x_n)=\bigwedge_{\substack{(\sigma_1,\ldots,\sigma_n)\\ f^*(\sigma_1,\ldots,\sigma_n)=1}} (x_1^{\sigma_1}\vee\cdots\vee x_n^{\sigma_n}).$$

Lijeva strana je $f(x_1,\ldots,x_n)$ a desna se može transformisati na sljedeći način:

$$\bigwedge_{\substack{(\sigma_1,\ldots,\sigma_n)\\ f^*(\sigma_1,\ldots,\sigma_n)=1}} (x_1^{\sigma_1}\vee\cdots\vee x_n^{\sigma_n})=\bigwedge_{\substack{(\sigma_1,\ldots,\sigma_n)\\ f(\overline{\sigma}_1,\ldots,\overline{\sigma}_n)=0}} (x_1^{\sigma_1}\vee\cdots\vee x_n^{\sigma_n})=$$

$$=\bigwedge_{\substack{(\sigma_1,\ldots,\sigma_n)\\ f(\sigma_1,\ldots,\sigma_n)=0}} (x_1^{\overline{\sigma}_1}\vee\cdots\vee x_n^{\overline{\sigma}_n}).$$

Na taj način dobijamo razlaganje koje se naziva *savršenom konjunktivnom normalnom formom (SKNF)*:

$$f(x_1,\ldots,x_n)=\bigwedge_{\substack{(\sigma_1,\ldots,\sigma_n)\\ f(\sigma_1,\cdots,\sigma_n)=0}} x_1^{\overline{\sigma}_1}\vee x_2^{\overline{\sigma}_2}\vee\cdots\vee x_n^{\overline{\sigma}_n}.$$

Kao primjer, nađimo savršenu konjunktivnu normalnu formu implikacije $f(x_1,x_2)=x_1\to x_2$. Implikacija ima vrijednost 0 samo na paru (1,0). Zato je

$$f(x_1,x_2)=x_1^{\overline{1}}\vee x_2^{\overline{0}}=x_1^0\vee x_2^1=\overline{x}_1\vee x_2.$$

Za funkciju zadatu tablicom u primjerima za SDNF (slika 4.14) nalazimo SKNF:

$$f(x_1,x_2,x_3)=(x_1\vee\overline{x}_2\vee x_3)\&(x_1\vee\overline{x}_2\vee\overline{x}_3)\&(\overline{x}_1\vee x_2\vee x_3)\&(\overline{x}_1\vee\overline{x}_2\vee x_3).$$

4.8 Zadaci za vježbu

1. Na osnovu funkcija $f(x_1,x_2)$ i $g(x_3,x_4)$ koje su zadate vektorski, odrediti vektorski zapis funkcije h.
   a) $f=1011$, $g=1001$, $h(x_2,x_3,x_4)=f(g(x_3,x_4),x_2)$;
   b) $f=1011$, $g=1001$, $h(x_1,x_2,x_3,x_4)=f(x_1,x_2)\vee g(x_3,x_4)$;
   c) $f=1000$, $g=0111$, $h(x_1,x_2,x_3,x_4)=f(x_1,x_2)\,\&\,g(x_3,x_4)$.

2. a) Funkcija $f(x_1,x_2,x_3)$ zadata je na sljedeći način: ona je jednaka 1 kada je $x_1=1$ ili ako promjenljive x_2 i x_3 imaju istu vrijednost a vrijednost promjenljive x_1 je manja od vrijednosti promjenljive x_3 (u ostalim

slučajevima ova funkcija je jednaka nuli). Sastaviti tablicu funkcije f. b) Sastaviti tablicu funkcije f koja je definisana na sljedeći način: funkcija $f(x_1, x_2, x_3, x_4)$ je jednaka nuli samo za one (x_1, x_2, x_3, x_4) za koje važi nejednakost $x_1 + x_2 > x_3 + 2x_4$.

3. Sastaviti tablicu funkcije zadate pomoću formule:
   a) $\overline{(x \to y) \oplus ((y \to z) \oplus (z \to x))}$;
   b) $\overline{(x \vee y) \vee x\overline{z}} \uparrow (x \sim y)$;
   c) $\overline{x} \to (\overline{z} \sim (y \oplus xz))$. ($\sim$ označava *ekvivalenciju*, tj. funkciju 1001)

4. Da li je sljedeća formula identički istinita (uvijek je jednaka 1) ili je možda identički lažna (uvijek jednaka 0).
   a) $(x \to y) \to ((x \vee z) \to (y \vee z))$;
   b) $((x \oplus y) \sim z)(x \to yz)$;
   c) $((x \vee \overline{y})z \to ((x \sim z) \oplus y))(x(yz))$.

5. Ispitati da li važe sljedeće jednakosti:
   a) $x \vee (y \sim z) = (x \vee y) \sim (x \vee z)$;
   b) $x \to (y \sim z) = (x \to y) \sim (x \to z)$;
   c) $x \,\&\, (y \sim z) = (x \,\&\, y) \sim (x \,\&\, z)$;
   d) $x \to (y \vee z) = (x \to y) \vee (x \to z)$;
   e) $x \to (y \,\&\, z) = (x \to y) \,\&\, (x \to z)$;
   f) $x \oplus (y \to z) = (x \oplus y) \to (x \oplus z)$;
   g) $x \to (y \to z) = (x \to y) \to (x \to z)$.

6. Realizovati funkciju f pomoću formule nad skupom simbola funkcija S. a) $f = x \to y$, $S = \{^{-}, \vee\}$; b) $f(x, y) = x \vee y$, $S = \{\to\}$; c) $f(x, y) = x \sim y$, $S = \{\,\&\,, \to\}$.

7. Uz korišćenje spiska poznatih elementarnih ekvivalentnosti, dokazati da su formule φ i ψ ekvivalentne.
   a) $\varphi = (\overline{x} \,\&\, \overline{z}) \vee (x \,\&\, y) \vee (x \,\&\, \overline{z})$, $\psi = x \,\&\, \overline{\overline{y} \,\&\, z} \vee \overline{x} \,\&\, \overline{z}$;
   b) $\varphi = (x \to y) \to ((x \,\&\, \overline{y}) \oplus (x \sim \overline{y}))$, $\psi = (x \vee y) \,\&\, (\overline{x} \vee \overline{y})$.

8. Za funkciju $f(x, y)$ iz P_2 važi $f(f(x, f(y, z)), f(f(x, y), f(x, z))) = 1$. Dokazati da tada važe sljedeće dvije jednakosti: i) $f(x, x) = 1$, ii) $f(x, f(y, x)) = 1$.

9. Nabrojati suštinske promjenljive funkcije:
   a) $f(x_1, x_2, x_3) = (x_1 \to (x_1 \vee x_2)) \to x_3$.
   b) $f(x_1, x_2) = (x_1 \vee x_2) \to \overline{x_2}$.
   c) $f(x_1, x_2, x_3, x_4) = (x_1 \vee x_2 \vee \overline{x_2}x_3 \vee \overline{x_1}\,\overline{x_2}\,\overline{x_3})x_4$.

10. Pokazati da je x_1 fiktivna promjenljiva za funkciju f (izraziti f formulom u kojoj se x_1 ne pojavljuje).
   a) $f(x_1, x_2, x_3) = (((x_3 \to x_2) \vee x_1)(x_2 \to x_1)x_3\overline{x_1}) \oplus x_3$.
   b) $f(x_1, x_2, x_3) = ((x_1 \vee x_2)(x_1 \vee \overline{x_3}) \to (\overline{x_1} \to x_2\overline{x_3}))x_2$.

11. Odrediti fiktivne promjenljive funkcije f, a zatim konstruisati njoj jednaku funkciju čije su sve promjenljive suštinske. a) $f(x_1, x_2, x_3) = 11110000$. b) $f = 00110011$. c) $f = 00111100$.

12. Da li je funkcija g dualna funkciji f. a) $f = x \oplus y$, $g = x \sim y$. b) $f = x \to y$, $g = y \to x$. c) $f = xy \vee xz \vee yz$, $g = xy \oplus xz \oplus yz$. d) $f = x \oplus y \oplus z$, $g = x \oplus y \oplus z$. e) $f = \overline{x}\,\overline{y}\,z \vee x(y \sim z)$, $g = 01101101$.

13. Koristeći princip dualnosti, konstruisati formulu koja realizuje funkciju dualnu funkciji f. Dobijeni izraz uprostiti (zapisati u DNF ili u obliku polinoma Žegalkina). a) $f = xy \vee yz \vee xt \vee zt$. b) $f = x \cdot 1 \vee y(zt \vee 0) \vee \overline{x}yz$. c) $f = (x \vee y \vee (y\overline{z} \oplus 1)) \to 1$.

14. Da li je funkcija f samodualna. a) $f = \overline{(x \to y) \to xz} \to (y \to z)$. b) $f = (\overline{x} \vee y \vee \overline{z})t \vee \overline{x}y\overline{z}$. c) f=0001 0010 0110 0111.

15. Dokazati da ne postoji samodualna funkcija koja suštinski zavisi od dvije promjenljive.

16. Pomoću ekvivalentnih transformacija prevesti formulu u DNF: $F = (x_1 \vee x_2\overline{x_2})(x_1 \vee x_3)$.

17. Predstaviti funkciju u obliku SDNF. a) $f(x_1, x_2, x_3) = (x_1 \oplus x_2) \to x_2x_3$. b) $f(x_1, x_2, x_3) = 01101100$. c) $f(x_1, x_2, x_3) = 10001110$. d) $f(x_1, x_2) = x_1 \uparrow x_2$.

18. Pomoću transformacija oblika $A = Ax \vee A\overline{x}$ i $A \vee A = A$ preći na SDNF. a) $f(x_1, x_2, x_3) = x_1 \vee \overline{x_2}x_3$. b) $f(x_1, x_2, x_3) = x_1\overline{x_2} \vee \overline{x_1}x_3$. c) $f(x_1, x_2, x_3) = x_1 \vee \overline{x_1}x_2 \vee \overline{x_2}x_3$.

19. Pomoću relacija oblika $x \vee yz = (x \vee y)(x \vee z)$ prevesti funkcije iz prethodnog zadatka u KNF.

20. Dokazati formulu. a) $\overline{x} = x \oplus 1$. b) $(x \oplus y)z = xz \oplus yz$.

21. Koliko puta data funkcija ima vrijednost 1? a) $f(x_1, x_2, \ldots, x_n) = x_1 \ldots x_k \oplus x_{k+1} \ldots x_n$; b) $f(x_1, x_2, \ldots, x_n) = 1 \oplus x_1 \oplus x_1x_2 \oplus \ldots \oplus x_1x_2 \ldots x_n$.

4.9 Rješenja zadataka za vježbu

1. a) Vidi tablicu na slici 4.15a). Odgovor je $h = 11111001$.

2. a) $x_1 = 1$ četiri reda tablice $f = 1$. $0 = x_1 < x_2 = x_3 = 1$ jedan red tablice $f = 1$ (tablica 2.a) na slici 4.15).

3. a) Znamo da je $x \to y = 0$ (netačno) samo u slučaju $x = 1$, $y = 0$, a u ostala tri slučaja je jednako 1 (tačno). Znamo da je $\oplus$ asocijativna pa umjesto $(x \oplus y) \oplus z$ možemo pisati $x \oplus y \oplus z$. Takođe je komutativna pa redosljed nije bitan, recimo $x \oplus y \oplus z = y \oplus z \oplus x$. Izraz od više $\oplus$ je jednak 1 kada je neparan broj članova jednak 1. (tablica 3.a) na slici 4.15)

1. a)

x_2	x_3	x_4	h
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

2. a)

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

3. a)

x	y	z	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Slika 4.15: Rješenja zadataka 1.a), 2.a) i 3.a)

4. a) Da li je moguće da je ovaj izraz jednak 0? Jedino ako je $x \to y = 1$ i $(x \vee z) \to (y \vee z) = 0$. Ovo posljednje je implikacija pa bi moralo $x \vee z = 1$ i $y \vee z = 0$. Posljednje znači $y = 0$ i $z = 0$, a tada je (zbog $x \vee z = 1$) $x = 1$. Međutum, tada ne važi ranije $x \to y = 1$. Kako to nije moguće, zaključujemo da je ova formula identički istinita.

Drugi način da riješimo ovaj zadatak je pomoću tablice. Treći način bio bi pomoću transformacija kao što je $x \to y = \overline{x} \vee y$, a četvrti analizom po z. Naime, u slučaju $z = 0$ ovdje piše $(x \to y) \to (x \to y)$ (što je istinito), a u slučaju $z = 1$ ovdje piše $(x \to y) \to (1 \to 1)$ (što je takođe istinito) ("istina slijedi iz bilo čega").

5. a) Analizirajmo po x. Za $x = 0$ ovo je $y \sim z = y \sim z$ odnosno 1 (tačno). A za $x = 1$ data jednakost se svodi na $1 = 1 \sim 1$ odnosno $1 = 1$. Dakle, jednakost važi, jer se analiza sastoji od dva slučaja koji uzeti zajedno pokrivaju sve moguće slučajeve.

6. a) Odgovor $x \to y = \overline{x} \vee y$. Na lijevoj strani znaka jednakosti pojavljuje se funkcija f koja treba da bude realizovanaa, a na desnoj funkcije iz (dozvoljenog) skupa S. Imamo na raspolaganju funkcije iz skupa S i proizvoljne promjenljive.

b) Odgovor je $x \vee y = (x \to y) \to y$.

c) Odgovor je $x \sim y = (x \to y)\,\&\,(y \to x)$; ekvivalencija je isto što i implikacija u jednom i u drugom smjeru.

7. a) Želimo da i formula ψ izgleda kao $\ldots \vee \ldots \vee \ldots$. Pri kraju formule ψ dio $\overline{x}\,\&\,\overline{z}$ može da bude u zagradi, jer $\&$ ima veći prioritet. Na dio $\overline{\overline{y}\,\&\,z}$ primijenimo formulu za negaciju konjunkcije.

$$\psi = x\,\&\,\overline{\overline{y}\,\&\,z} \vee \overline{x}\,\&\,\overline{z} = (x\,\&\,(y \vee \overline{z})) \vee \overline{x}\,\&\,\overline{z} = (x\,\&\,y) \vee (x\,\&\,\overline{z}) \vee (\overline{x}\,\&\,\overline{z})$$

Primijenili smo i poznati obrazac koji govori da je $a\,\&\,(b \vee c)$ isto što i $(a\,\&\,b) \vee (a\,\&\,c)$. Dokazano je da su ove dvije date formule ekvivalentne, jer smo izraz za ψ doveli do poklapanja sa izrazom za φ.

8. U zadatoj relaciji stavimo $(x, y, z) = (0, 0, 0)$. Ako dopustimo da je $f(0, 0) = 0$, onda ćemo dobiti kontradikciju. Znači da je $f(0, 0) = 1$. Zatim se usmjeravamo na $f(0, 1)$ uz pomoć $(x, y, z) = (0, 0, 0)$ i uz pomoć $(x, y, z) = (0, 1, 1)$. Dalje sami nastavite rješavanje ovog zadatka.

9. a) Dio izraza (zadate funkcije) $x_1 \to (x_1 \vee x_2)$ je uvijek istinit. Zato što pretpostavka x_1 uvijek povlači zaključak $x_1 \vee x_2$; ovom drugom $x_1 \vee x_2$ je "lakše" da bude istinit od ovog prvog x_1. U ovo možemo da se uvjerimo i pomoću transformacija:

$$x_1 \to (x_1 \vee x_2) = \overline{x_1} \vee (x_1 \vee x_2) = 1 \vee x_2 = 1;$$

$x_1 \vee \overline{x_1} = 1$ disjunkcija između nekog iskaza i negacije tog istog iskaza je uvijek istinita; $1 \vee x_2 = 1$ disjunkcija između istine i bilo kog iskaza je sigurno istina. Zato se dati izraz svodi na $1 \to x_3$ što se dalje svodi na prosto x_3. Dobili smo da je $f = x_3$. Vidimo da x_3 utiče na vrijednost funkcije. Dakle, samo x_3 je suštinska promjenljiva funkcije f.

Drugi način da se ovaj zadatak uradi je da sastavimo tablicu date funkcije, a zatim provjerimo da li uvijek važi da je $f(0,x_2,x_3)=f(1,x_2,x_3)$. Ako jeste onda je x_1 fiktivna (a ako nije onda je x_1 suštinska). Slično, po definiciji, možemo ustanoviti da li je x_2 (odnosno x_3) fiktivna ili suštinska.

10. a) $f(x_1,x_2,x_3)=(((x_3\to x_2)\vee x_1)(x_2\to x_1)x_3\overline{x_1})\oplus x_3$

Eliminišemo jednu i drugu implikaciju po poznatom obrascu $x\to y=\overline{x}\vee y$ (osnovni obrazac za implikaciju)

$$f=((\overline{x_3}\vee x_2\vee x_1)(\overline{x_2}\vee x_1)x_3\overline{x_1})\oplus x_3$$

Izvršimo zatim "množenje" sa $x_3\overline{x_1}$, pri čemu znamo da je nekorisno da se piše član koji sadrži dio oblika $x\overline{x}$

$$f=(\overline{x_1}x_3x_2(\overline{x_2}\vee x_1))\oplus x_3$$

Dio ispred znaka $\oplus$ jednak je nuli, jer da bi on bio jednak jedinici trebalo bi da je istovremeno $\overline{x_1}=1$ i $x_3=1$ i $x_2=1$ i $\overline{x_2}\vee x_1=1$

$$f=0\oplus x_3=x_3.$$

Konačno imamo $f=x_3$.

11. a) Napisati prvo tablicu zadate funkcije. Odatle je očito da je $f=\overline{x_1}$. Slijedi da su za f fiktivne x_2 i x_3. Imamo: $f=g$ gdje je $g(x_1)=\overline{x_1}$.

12. a) Načinimo tablice za f i g (slika 4.16). Da li se, kada se vektor vrijednosti za f preokrene i komplementira, dobije vektor g? Odgovor – da.

b) Odgovor je ne.

x	y	f
0	0	0
0	1	1
1	0	1
1	1	0

x	y	g
0	0	1
0	1	0
1	0	0
1	1	1

Slika 4.16: Tablice funkcija f i g (zadatak 12.a))

13. a) $f=xy\vee yz\vee xt\vee zt=(x\vee z)y\vee(x\vee z)t=(x\vee z)(y\vee t)$.

Znamo da je $(a\vee b)^*=ab$, $(ab)^*=a\vee b$, kao i princip dualnosti: $(f(f_1,f_2))^*=f^*(f_1^*,f_2^*)$; f_1 mi je $x\vee z$. Tako $f^*=xz\vee yt$, ovo je odgovor.

14. a) Prvo dva puta koristimo osnovni obrazac za implikaciju. Koriste se i obrasci za negaciju konjunkcije i za negaciju disjunkcije (De Morganovi obrasci).

$$\overline{(x\to y)\to xz}=\overline{(\overline{x}\vee y)\to xz}=\overline{\overline{\overline{x}\vee y}\vee xz}=\overline{x\overline{y}\vee xz}=(\overline{x}\vee y)(\overline{x}\vee\overline{z})=$$

$$=\overline{x}\vee\overline{x}\,\overline{z}\vee y\overline{x}\vee y\overline{z}=\overline{x}\vee y\overline{z}$$

Zato što pored $\overline{x}$ nema koristi da se u disjunkciji piše član $\overline{x}\,\overline{z}$ (sadrži dodatni faktor $\overline{z}$), a takođe ni $y\overline{x}$ (dodato je y). Drugim riječima, važi formula $a\vee ab=a$, koja se lako dokazuje. Tako,

$$f=(\overline{x}\vee y\overline{z})\to(y\to z)=\overline{\overline{x}\vee y\overline{z}}\vee(y\to z)=x\overline{y}\vee xz\vee\overline{y}\vee z=\overline{y}\vee z.$$

Iz ovoga što smo dobili da je $f(x,y,z)=\overline{y}\vee z$ lako slijedi da funkcija f nije samodualna, jer ne ispunjava uslov $f(\overline{x},\overline{y},\overline{z})=\overline{f}(x,y,z)$. Odgovor: nije.

b) Odgovor je – jeste.

c) Odgovor je – nije.

15. Ako je f samodualna onda je dovoljno da se zadaju samo dvije njene vrijednosti i to $f(0,0)$ i $f(0,1)$ pa da time čitava funkcija f bude definisana odnosno da budu definisane sve četiri njene vrijednosti. Kako zadati $p=f(0,0)$ i $q=f(0,1)$. Za ovo postoje četiri mogućnosti (4.17). Prva mogućnost (da i p i q budu nula) ustvari daje funkciju $f=x$, a za tu funkciju je očito da suštinski zavisi od samo jedne promjenljive. Slično i dalje: II $f=y$, III $f=\overline{y}$, IV $f=\overline{x}$. Dokaz je završen.

x	y	f
0	0	p
0	1	q
1	0	$\overline{q}$
1	1	$\overline{p}$

I	II	III	IV
0	0	1	1
0	1	0	1
1	0	1	0
1	1	0	0

Slika 4.17: Zadatak 15)

16. $F=(x_1\vee x_2\overline{x_2})(x_1\vee x_3)=(x_1\vee 0)(x_1\vee x_3)=x_1(x_1\vee x_3)=x_1$;

poznat je obrazac $a(a \vee b) = a$, koji se uostalom lako dokazuje (provjerava).

17. b) $f = 01101100 = \overline{x_1}\,\overline{x_2}x_3 \vee \overline{x_1}x_2\overline{x_3} \vee x_1\overline{x_2}\,\overline{x_3} \vee x_1\overline{x_2}x_3$.

Prvo se po zadatom vektoru vrijednosti napiše tablica zadate funkcije (slika 4.18), a u drugom koraku se po tablici nađe SDNF. Upravo, na račun svake jedinice kod f doći će po jedan član. Prva jedinica je za $(x_1, x_2, x_3) = (0, 0, 1)$ pa njoj sljeduje $\overline{x_1}\,\overline{x_2}x_3$, itd.

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Slika 4.18: Zadatak 17.b)

18. a) $f(x_1, x_2, x_3) = x_1 \vee \overline{x_2}x_3$. Već imamo DNF a treba da dobijemo SDNF. Treba svaka elementarna konjunkcija da ima puni rang, tj. da ima rang tri, budući da se radi o funkciji od tri promjenljive. Ispod x_1 "kriju se" $x_1\overline{x_2}\,\overline{x_3}$, $x_1\overline{x_2}x_3$, $x_1x_2\overline{x_3}$, $x_1x_2x_3$ (dakle, četiri izraza). Ispod $\overline{x_2}x_3$ kriju se $\overline{x_1}\,\overline{x_2}x_3$, $x_1\overline{x_2}x_3$ (dva izraza). Ovih šest izraza treba sastaviti pomoću $\vee$, pri čemu se jedan ponavlja pa ga nećemo dvaput pisati. Odgovor je:

$$f(x_1, x_2, x_3) = x_1\overline{x_2}\,\overline{x_3} \vee x_1\overline{x_2}x_3 \vee x_1x_2\overline{x_3} \vee x_1x_2x_3 \vee \overline{x_1}\,\overline{x_2}x_3.$$

19. a) $f(x_1, x_2, x_3) = x_1 \vee \overline{x_2}x_3 = (x_1 \vee \overline{x_2})(x_1 \vee x_3)$.

Znamo da se KNF razlikuje od DNF po tome što & i $\vee$ zamijene uloge.

20. a), b) Dokazati provjerom.

21. a) Posmatrajmo izraz $x_1 \ldots x_k \oplus x_{k+1} \ldots x_n$ ili recimo $x_1x_2x_3 \oplus x_4x_5x_6x_7x_8x_9x_{10}$ i analizirajmo kada je taj izraz jednak 1. Budući da izraz ima oblik $a \oplus b$ trebalo bi da bude oblika $1 \oplus 0$ ili kao $0 \oplus 1$. Razmotrimo prvo za $1 \oplus 0$. Važi da je $x_1 \,\&\, x_2 \,\&\, x_3 = 1$ samo u jednom slučaju (ako su sva tri

učesnika jednaka 1), a inače je naravno jednako 0. Dok $x_4x_5x_6x_7x_8x_9x_{10} = 0$ važi u svim slučajevima, osim slučaja kada su svi $x_4, \ldots, x_{10}$ jednaki 1, tj. važi u $2^7 - 1$ slučajeva. Zatim gledamo za $0 \oplus 1$. Na konkretnom primjeru $n = 10$, $k = 3$ uočili smo zakonitost. Odgovor je

$$1 \cdot (2^{n-k} - 1) + (2^k - 1) \cdot 1 = 2^{n-k} + 2^k - 2.$$

Glava 5

Kompletnost i zatvorenost

Vidjeli smo da se svaka funkcija algebre logike može izraziti u vidu formule preko elementarnih funkcija negacije, konjunkcije i disjunkcije. U vezi s tim, prirodno se nameće pitanje, u kojoj mjeri je postojanje takvih funkcija slučajno? U ovom poglavlju ćemo prvo pokazati da takvih sistema elementarnih funkcija ima više, a kasnije dati iscrpan odgovor na ovo pitanje.

5.1 Kompletnost

Za sistem funkcija $\{f_1, f_2, \ldots, f_n, \ldots\}$ iz P_2 kažemo da je (funkcionalno) *kompletan* ako se svaka Bulova funkcija može predstaviti kao formula nad datim skupom funkcija.

Očigledni primjeri kompletnih sistema su sam skup P_2 i skup koji sadrži negaciju, konjunkciju i disjunkciju. Nije svaki skup funkcija kompletan - recimo, sistem koji sadrži samo konstante funkcije 0 i 1 nije kompletan.

Sljedeća teorema govori o tome da se pitanje kompletnosti jednog sistema može svesti na kompletnost drugog sistema.

Teorema 5. Data su dva sistema funkcija $B = \{f_1, f_2, \ldots, f_n, \ldots\}$ i $G = \{g_1, g_2, \ldots, g_n, \ldots\}$ takva da je B kompletan sistem i da se svaka funkcija iz B može izraziti kao formula nad G. Tada je i sistem G kompletan sistem.

Dokaz. Neka je h proizvoljna funkcija iz P_2. Kako je sistem B kompletan, nju je moguće izraziti formulom nad B, tj.

$$h = C[f_1, \ldots, f_n, \ldots].$$

Po uslovu teoreme je

$$f_1 = C_1[g_1, \ldots, g_n, \ldots],$$

$$f_2 = C_2[g_1, \ldots, g_n, \ldots],$$

$$\ldots.$$

Zato možemo sva pojavljivanja funkcija $f_1, f_2, \ldots, f_n, \ldots$ u formuli $C[f_1, \ldots, f_n, \ldots]$ zamijeniti njihovim formulama nad G. To možemo zapisati na sljedeći način:

$$C[f_1, \ldots, f_n, \ldots] = C[C_1[g_1, \ldots, g_n, \ldots], C_2[g_1, \ldots, g_n, \ldots], \ldots].$$

Izraz na desnoj strani je formula nad B, odnosno

$$C[C_1[g_1, \ldots, g_n, \ldots], C_2[g_1, \ldots, g_n, \ldots], \ldots] = C'[g_1, \ldots, g_n, \ldots]$$

pa na kraju imamo

$$h = C'[g_1, \ldots, g_n, \ldots],$$

čime je teorema dokazana.

Primijetimo još da se ne mogu uvijek zamjene obaviti istovremeno jer prilikom zamjene može doći do nove pojave simbola $f_1, f_2, \ldots, f_n, \ldots$. Na primjer, neka je

$$f_1 = x_1 \& x_2, \; g_1 = x_1 \vee x_2, \; g_2 = x_1 + x_2$$

i

$$h = x_1 \& (x_2 \& x_3).$$

Tada je

$$f_1 = x_1 \& x_2 = x_1 + x_2 + x_1 \vee x_2$$

i

$$h = x_1 \& (x_2 \& x_3) = x_1 + (x_2 \& x_3) + (x_1 \vee (x_2 \& x_3)) =$$
$$= x_1 + x_2 + x_3 + (x_2 \vee x_3) + (x_1 \vee (x_2 + x_3 + (x_2 \vee x_3))).$$

Primjenom prethodne teoreme može se dokazati kompletnost velikog broja sistema funkcija. Na primjer, sistem $B_1 = \{\overline{x}, x_1 \& x_2\}$ je kompletan, jer se disjunkcija može izraziti preko negacije i konjunkcije (DeMorganovi zakoni). Tada iz principa dualnosti slijedi i da je sistem $B_2 = \{\overline{x}, x_1 \vee x_2\}$ kompletan.

Kako je $x_1 \uparrow x_1 = \overline{x}_1$, $(x_1 \uparrow x_2) \uparrow (x_1 \uparrow x_2) = \overline{x_1 \uparrow x_2} = x_1 \& x_2$, sistem $B_3 = \{x_1 \uparrow x_2\}$ je kompletan.

Sistem $B_4 = \{0, \; 1, \; x_1 + x_2, \; x_1 x_2\}$ je kompletan, jer je $x_1 + 1 = \overline{x}_1$. Formula nad B_4, poslije uklanjanja zagrada i prostih algebarskih transformacija, prelazi u polinom po modulu 2.

Teorema 6. Svaka funkcija iz P_2 može biti izražena putem polinoma po modulu 2 (polinoma Žegalkina).

Nađimo broj polinoma Žegalkina od promjenljivih $x_1, \ldots, x_n$, tj. broj izraza oblika

$$\sum_{(i_1, \ldots, i_s)} a_{i_1 \ldots i_s} x_{i_1} \ldots x_{i_s}.$$

Broj članova $x_{i_1} \ldots x_{i_s}$ jednak je broju podskupova $i_1, \ldots, i_s$ skupa $1, \ldots, n$, tj. on iznosi 2^n. Kako je $a_{i_1 \ldots i_s}$ jednako 0 ili 1, traženi broj polinoma jednak je 2^{2^n}, što je jednako broju Bulovih funkcija od promjenljivih $x_1, \ldots, x_n$. S druge strane, lako se vidi da različitim podskupovima odgovaraju različite funkcije. Odavde slijedi jedinstvenost predstavljanja funkcija putem polinoma Žegalkina.

Kao primjer, predstavimo disjunkciju $x_1 \vee x_2$ putem polinoma Žegalkina, tj. nađimo koeficijente a, b, c i d u izrazu:

$$x_1 \vee x_2 = a x_1 x_2 + b x_1 + c x_2 + d.$$

Ako stavimo $x_1 = x_2 = 0$ dobijamo d=0. Ako uvrstimo $x_1 = 0$ i $x_2 = 1$ dobijamo c=1. Iz $x_1 = 1$ i $x_2 = 0$ imamo b=1. Na kraju, iz $x_1 = 1$ i $x_2 = 1$ imamo $1 = a + b + c$, odnosno a=1. Zaključujemo da je

$$x_1 \vee x_2 = x_1 x_2 + x_1 + x_2.$$

5.2 Zatvorenje

Neka je B neki skup Bulovih funkcija. *Zatvorenjem* skupa B, u oznaci $[B]$ nazivamo skup svih Bulovih funkcija koje se mogu predstaviti kao formule nad B.

Lako je vidjeti da je operacija zatvorenja invarijantna u odnosu na dodavanje i uklanjanje fiktivnih promjenljivih.

Primjer. Ako je $B = P_2$, tada je $[B] = P_2$.

Primjer. Ako je B=$\{1, x_1+x_2\}$, tada je [B] skup svih linearnih Bulovih funkcija L, tj. funkcija oblika

$$f(x_1, \ldots, x_n) = c_0 + c_1x_1 + \ldots + c_nx_n,$$

gdje je $c_i = 0, 1$ $(i = 0, 1, \ldots, n)$; stvarne promjenljive ulaze sa koeficijentom 1, fiktivne sa 0.

Napišimo nekoliko glavnih svojstava zatvorenja:

- B$\subseteq$[B]
- $[[B]] = [B]$
- ako je $B_1 \subseteq B_2$, tada je $[B_1] \subseteq [B_2]$
- $[B_1] \cup [B_2] \subseteq [B_1 \cup B_2]$.

Za skup (*klasu*) Bulovih funkcija B kažemo da je *zatvoren* ako je [B]=B.

Očigledan primjer zatvorene klase je P_2. Zatvorena je i klasa linearnih funkcija, jer je linearni izraz sastavljen od linearnih izraza- linearan.

Klasa B=$\{1, x_1+x_2\}$ nije zatvorena.

Očigledno je da je svako zatvorenje neke klase [B] zatvoreno. Ovo daje mogućnost dobijanja mnogih primjera zatvorenih klasa.

U terminima zatvorenja možemo dati ekvivalentnu definiciju kompletnosti: sistem B je kompletan ako je [B]= P_2.

5.3 Primjeri zatvorenih klasa

1. Klasa T_0

Navedimo nekoliko važnih primjera zatvorenih klasa.
T_0 je klasa svih funkcija koje čuvaju 0, tj. svih funkcija f takvih da je

$$f(0, \ldots, 0) = 0.$$

Očigledno je da funkcije 0, x, konjunkcija, disjunkcija i sabiranje pripadaju klasi T_0, a konstanta 1 i negacija ne pripadaju.

Kako tačno polovina svih Bulovih funkcija ima vrijednost 0 na $(0, \ldots, 0)$, to tačno polovina svih Bulovih funkcija pripada klasi T_0. Dakle, klasa T_0 ima $\frac{1}{2}2^{2^n}$ elemenata.

Dokažimo da je T_0 zatvorena klasa. Kako identičko preslikavanje, tj. funkcija x, pripada T_0, dovoljno je dokazati da funkcija $F = f(f_1, \ldots, f_n)$ pripada T_0 ako funkcije $f, f_1, \ldots, f_n$ pripadaju T_0. To slijedi iz

$$F(0, \ldots, 0) = f(f_1(0, \ldots, 0), \ldots, f_n(0, \ldots, 0)) = f(0, \ldots, 0) = 0.$$

2. Klasa T_1

T_1 je klasa svih funkcija koje čuvaju 1, tj. svih funkcija f takvih da je

$$f(1, \ldots, 1) = 1.$$

Funkcije 1, x, konjunkcija, disjunkcija pripadaju klasi T_1, a konstanta 0, negacija i sabiranje ne pripadaju.

Lako je vidjeti da klasa T_1 sadrži funkcije dualne funkcijama iz T_0. Kažemo zato da je klasa T_1 dualna klasi T_0. Zato nije teško prenijeti sve rezultate za klasu T_0 u klasu T_1. Dakle, T_1 je zatvorena klasa koja ima $\frac{1}{2}2^{2^n}$ elemenata.

3. Klasa S

Klasi S funkcija pripadaju funkcije za koje je $f = f^*$, tj. koje su samodualne. Funkcije x i negacija su samodualne. I funkcija

$$f(x_1, x_2, x_3) = x_1x_2 \vee x_1x_3 \vee x_2x_3$$

je samodualna jer je

$$f^*(x_1,x_2,x_3) = (x_1 \vee x_2)\&(x_1 \vee x_3)\&(x_2 \vee x_3) = x_1x_2 \vee x_1x_3 \vee x_2x_3 = f(x_1,x_2,x_3).$$

Za samodualnu funkciju važi da je $f(x_1,\ldots,x_n) = \overline{f}(\overline{x_1},\ldots,\overline{x_n})$, odnosno da na vrijednostima $(\alpha_1,\ldots,\alpha_n)$ i $(\overline{\alpha_1},\ldots,\overline{\alpha_n})$, koje nazivamo *suprotnima*, funkcija ima suprotne vrijednosti. To znači da je samodualna funkcija u potpunosti definisana prvom polovinom tablice. Odavde zaključujemo da je broj samodualnih funkcija jednak $2^{2^{n-1}} = 2^{\frac{2^n}{2}} = \sqrt{2^{2^n}}$.

Dokažimo da je klasa S zatvorena. Kako identičko preslikavanje, tj. funkcija x, pripada S, dovoljno je dokazati da funkcija

$$\Phi = f(f_1,\ldots,f_n)$$

pripada S ako funkcije $f, f_1,\ldots,f_n$ pripadaju S. Uz upotrebu principa dualnosti, to slijedi neposredno iz

$$\Phi^* = f^*(f_1^*,\ldots,f_n^*) = f(f_1,\ldots,f_n) = \Phi.$$

Lema 1. (O nesamodualnoj funkciji) Ako $f \notin$S, tada se iz nje korišćenjem funkcija x i negacije može dobiti konstanta.

Dokaz. Kako, po pretpostavci, f nije samodualna funkcija, to postoji n-torka $(\alpha_1,\ldots,\alpha_n)$ za koju je

$$f(\alpha_1,\ldots,\alpha_n) = f(\overline{\alpha_1},\ldots,\overline{\alpha_n}).$$

Definišimo funkcije

$$\varphi_i(x) = x^{\alpha_i},$$

za $i = 1,\ldots,n$ i funkciju

$$\varphi(x) = f(\varphi_1(x),\ldots,\varphi_n(x)).$$

Tada imamo

$$\varphi(0) = f(\varphi_1(0),\ldots,\varphi_n(0)) = f(0^{\alpha_1},\ldots,0^{\alpha_n}) = f(\overline{\alpha_1},\ldots,\overline{\alpha_n}) =$$
$$= f(\alpha_1,\ldots,\alpha_n) = f(1^{\alpha_1},\ldots,1^{\alpha_n}) = f(\varphi_1(1),\ldots,\varphi_n(1)) = \varphi(1),$$

odnosno funkcija φ je konstanta. Vidimo da se x_i supstituiše sa x^{α_i}.

4. Klasa M

Ako uvedemo vektorski zapis n-torki:

$$\tilde{\alpha} = (\alpha_1,\cdots,\alpha_n),$$

tada umjesto

$$f(\alpha_1,\cdots,\alpha_n)$$

možemo pisati

$$f(\tilde{\alpha}).$$

Kažemo da za dvije n-torke $\tilde{\alpha},\tilde{\beta}$ važi *relacija poretka* $\tilde{\alpha} \preceq \tilde{\beta}$, ako je $\alpha_i \leq \beta_i$, za svako i=1, ...,n. Na primjer, $(0,1,0,1) \preceq (1,1,0,1)$.

Očigledno je da za relaciju poretka n-torki važi tranzitivnost - ako je $\tilde{\alpha} \preceq \tilde{\beta}$ i $\tilde{\beta} \preceq \tilde{\gamma}$, tada je i $\tilde{\alpha} \preceq \tilde{\gamma}$. Primijetimo da se ne mogu svake dvije n-torke uporediti. Na primjer, za n-torke (0,1) i (1,0) ne važi ni (0,1) $\preceq$(1,0) niti (1,0) $\preceq$(0,1). Zbog toga je skup svih n-torki *parcijalno uređen* u odnosu relaciju poretka.

Funkcija f je *monotona* ako za svake dvije n-torke za koje je $\tilde{\alpha} \preceq \tilde{\beta}$, važi

$$f(\tilde{\alpha}) \leq f(\tilde{\beta}).$$

Lako je uočiti da su monotone funkcije 0, 1, x, disjunkcija i konjunkcija.

Označimo sa M skup svih monotonih funkcija. Pokažimo da je klasa monotonih funkcija zatvorena. Kako identičko preslikavanje, tj. funkcija x, pripada M, dovoljno je dokazati da funkcija

$$\Phi = f(f_1,\ldots,f_m)$$

pripada M ako funkcije $f, f_1,\ldots,f_m$ pripadaju M.

Neka su date funkcije

$$\Phi, f_1,\ldots,f_m$$

i n-torke njihovih promjenljivih

$$\tilde{x} = (x_1,\ldots,x_n), \quad \tilde{x}_1 = (x_{11},\ldots,x_{1p_1}),\ldots,\tilde{x}_m = (x_{m1},\ldots,x_{mp_m}),$$

pri čemu se skup promjenljivih funkcije Φ sastoji isključivo od promjenljivih koje se pojavljuju u funkcijama $f_1,\ldots,f_m$.

Dalje, neka su $\tilde{\alpha}$ i $\tilde{\beta}$ dvije n-torke promjenljivih $\tilde{x}$, pri čemu je $\tilde{\alpha} \preceq \tilde{\beta}$. Te n-torke određuju vrijednosti $\tilde{\alpha}_1, \tilde{\beta}_1, \ldots, \tilde{\alpha}_m, \tilde{\beta}_m$, promjenljivih $\tilde{x}_1, \ldots, \tilde{x}_m$, takve da je $\tilde{\alpha}_1 \preceq \tilde{\beta}_1, \ldots, \tilde{\alpha}_m \preceq \tilde{\beta}_m$. Zbog monotonosti funkcija $f_1, \ldots, f_m$, važi da je

$$f_1(\tilde{\alpha}_1) \leq f_1(\tilde{\beta}_1), \ldots, f_m(\tilde{\alpha}_m) \leq f_m(\tilde{\beta}_m).$$

Odavde slijedi da je

$$(f_1(\tilde{\alpha}_1), \ldots, f_m(\tilde{\alpha}_m)) \preceq (f_1(\tilde{\beta}_1), \ldots, f_m(\tilde{\beta}_m)),$$

a kako je i funkcija f monotona, imamo

$$f(f_1(\tilde{\alpha}_1), \ldots, f_m(\tilde{\alpha}_m)) \leq f(f_1(\tilde{\beta}_1), \ldots, f_m(\tilde{\beta}_m)),$$

odnosno

$$\Phi(\tilde{\alpha}) = f(f_1(\tilde{\alpha}_1), \ldots, f_m(\tilde{\alpha}_m)) \leq f(f_1(\tilde{\beta}_1), \ldots, f_m(\tilde{\beta}_m)) = \Phi(\tilde{\beta}),$$

što dokazuje tvrđenje.

Kažemo da su n-torke $\tilde{\alpha}$ i $\tilde{\beta}$ *susjedne* po i-toj koordinati ako je

$$\tilde{\alpha} = (\alpha_1, \cdots, \alpha_{i-1}, \alpha_i, \alpha_{i+1}, \cdots, \alpha_n)$$

$$\tilde{\beta} = (\alpha_1, \cdots, \alpha_{i-1}, \overline{\alpha}_i, \alpha_{i+1}, \cdots, \alpha_n).$$

Lema 2. (O nemonotonoj funkciji) Ako $f \notin M$, tada se iz f korišćenjem konstanti 0 i 1 i funkcije x može dobiti negacija.

Dokaz. Pokažimo prvo da kod nemonotone funkcije uvijek postoje susjedne n-torke $\tilde{\alpha}$ i $\tilde{\beta}$, takve da je $f(\tilde{\alpha}) > f(\tilde{\beta})$.

Kako je f nemonotona, po definiciji postoje n-torke $\tilde{\alpha}_1$ i $\tilde{\beta}_1$, takve da je $\tilde{\alpha}_1 \preceq \tilde{\beta}_1$ i $f(\tilde{\alpha}_1) > f(\tilde{\beta}_1)$. Ukoliko su $\tilde{\alpha}_1$ i $\tilde{\beta}_1$ susjedne n-torke, dokazali smo tvrđenje. Ukoliko nisu, n-torka $\tilde{\beta}_1$ razlikuje se od n-torke $\tilde{\alpha}_1$ u t koordinata, $t > 1$, pri čemu važi da tih t koordinata imaju vrijednost 0 u n-torki $\tilde{\alpha}$ a vrijednost 1 u n-torki $\tilde{\beta}$. Zato možemo napraviti t-1 susjednih n-torki $\tilde{\alpha}_2$, $\ldots, \tilde{\alpha}_t$, takvih da je

$$\tilde{\alpha}_1 \preceq \tilde{\alpha}_2 \preceq \ldots \preceq \tilde{\alpha}_t \preceq \tilde{\beta}_1.$$

Kako je po pretpostavci $f(\tilde{\alpha}_1) > f(\tilde{\beta}_1)$, postoji bar jedan par susjednih n-torki, označimo ih sa $\tilde{\alpha}$ i $\tilde{\beta}$, takvih da je $f(\tilde{\alpha}) > f(\tilde{\beta})$.

Sada možemo nastaviti dokaz. Neka je nađeni par n-torki $\tilde{\alpha}$ i $\tilde{\beta}$ susjedan po i-toj koordinati, i

$$\tilde{\alpha} = (\alpha_1, \ldots, \alpha_{i-1}, 0, \alpha_{i+1}, \ldots, \alpha_n),$$

$$\tilde{\beta} = (\alpha_1, \ldots, \alpha_{i-1}, 1, \alpha_{i+1}, \ldots, \alpha_n).$$

Definišimo funkciju

$$\varphi(x) = f(\alpha_1, \ldots, \alpha_{i-1}, x, \alpha_{i+1}, \ldots, \alpha_n).$$

Imamo

$$\varphi(0) = f(\alpha_1, \cdots, \alpha_{i-1}, 0, \alpha_{i+1}, \cdots, \alpha_n) =$$

$$= f(\tilde{\alpha}) > f(\tilde{\beta}) =$$

$$= f(\alpha_1, \cdots, \alpha_{i-1}, 1, \alpha_{i+1}, \cdots, \alpha_n) = \varphi(1).$$

Za funkciju $\varphi(x)$ važi $\varphi(0) = 1$ i $\varphi(1) = 0$, što znači da je $\varphi(x) = \overline{x}$. Lema je dokazana.

Vidimo da supstitucija

$$\begin{pmatrix} x_1 & \ldots & x_{i-1} & x_i & x_{i+1} & \ldots & x_n \\ \alpha_1 & \ldots & \alpha_{i-1} & x & \alpha_{i+1} & \ldots & \alpha_n \end{pmatrix}$$

svodi funkciju na negaciju.

5. Klasa L

Pokazali smo ranije da je klasa linearnih funkcija L zatvorena. Ona očigledno sadrži konstante 0, 1, identičku funkciju x, negaciju i sabiranje. Konjunkcija i disjunkcija ne pripadaju toj klasi.

Lema 3. (O nelinearnoj funkciji) Ako $f \notin L$, tada se iz nje korišćenjem konstanti 0 i 1, funkcija x i negacije, i eventualno pomoću negacije funkcije f, može dobiti konjunkcija.

Dokaz. Razmotrimo polinom Žegalkina za funkciju f:

$$f(x_1, \ldots, x_n) = \sum_{\{i_1,\ldots,i_s\}\subset\{1,\ldots,n\}} a_{i_1\ldots i_s} x_{i_1} \ldots x_{i_s}.$$

Polinom Žegalkina nije linearan, pa postoji u njemu član koji sadrži bar dva množitelja. Bez ograničavanja opštosti možemo smatrati da su dva od tih množitelja x_1 i x_2. Tada se polinom (a time i funkcija f) može zapisati u obliku

$$\sum_{\{i_1,\ldots,i_s\}} a_{i_1\ldots i_s} x_{i_1} \ldots x_{i_s} = x_1 x_2 f_1(x_3, \ldots, x_n) + x_1 f_2(x_3, \ldots, x_n) +$$

$$x_2 f_3(x_3, \ldots, x_n) + f_4(x_3, \ldots, x_n),$$

pri čemu, zbog jedinstvenosti polinoma, funkcija f_1 nije konstanta 0.

Kako f_1 nije konstanta 0, postoje $\alpha_3, \ldots, \alpha_n$, takvi da je

$$f_1(\alpha_3, \ldots, \alpha_n) = 1.$$

Posmatrajmo funkciju

$$\varphi(x_1, x_2) = f(x_1, x_2, \alpha_3, \ldots, \alpha_n) = x_1 x_2 + \alpha \cdot x_1 + \beta \cdot x_2 + \gamma,$$

gdje su α, β, γ konstante jednake 0 ili 1.

Na osnovu funkcije $\varphi(x_1, x_2)$ definišimo funkciju

$$\psi(x_1, x_2) = \varphi(x_1 + \beta, x_2 + \alpha) + \alpha\beta + \gamma.$$

Očigledno je da je

$$\varphi(x_1 + \beta, x_2 + \alpha) + \alpha\beta + \gamma =$$
$$= (x_1 + \beta)(x_2 + \alpha) + \alpha(x_1 + \beta) + \beta(x_2 + \alpha) + \gamma + \alpha\beta + \gamma = x_1 x_2.$$

Slijedi da je $\psi(x_1, x_2) = x_1 x_2$, što dokazuje lemu. Vidimo da supstitucija koja se koristi glasi:

$$\begin{pmatrix} x_1 & x_2 & x_3 & \ldots & x_n \\ x_1^{\overline{\beta}} & x_2^{\overline{\alpha}} & \alpha_3 & \ldots & \alpha_n \end{pmatrix}.$$

Zatvorene klase T_0, T_1, S, M i L uzajamno su različite, što ilustruje tabela 5.1.

	T_0	T_1	S	M	L
0	+	-	-	+	+
1	-	+	-	+	+
$\overline{x}$	-	-	+	-	+

Tabela 5.1: Različitost klasa T_0, T_1, S, M i L

5.4 Teorema o kompletnosti

Prethodni primjeri zatvorenih klasa dovode nas do rješenja jednog od osnovnih pitanja algebre logike – koji su potrebni i dovoljni uslovi da bi sistem funkcija B bio kompletan.

Teorema 7. (Teorema o kompletnosti) Da bi sistem funkcija B bio kompletan, potrebno je i dovoljno da B nije podskup nijedne od klasa T_0, T_1, S, M i L, tj. da on sadrži bar po jednu funkciju koja nije iz T_0, T_1, S, M i L.

Dokaz.

Neophodnost. Neka je B kompletan sistem, tj. $[B] = P_2$. Pretpostavimo da je B sadržan u nekoj od naznačenih klasa, označimo je sa G. Dakle, B je podskup od G, $B \subseteq G$. Tada je na osnovu svojstava zatvorenja:

$$P_2 = [B] \subseteq [G] = G \subseteq P_2.$$

Odavde slijedi da je $G = P_2$, što nije tačno.

Dovoljnost. Neka je B skup funkcija koji nije podskup nijedne od klasa T_0, T_1, S, M i L. Treba da dokažemo da je B kompletan sistem. Kako B nije podskup pomenutih klasa, iz B je moguće izdvojiti podskup B', koji ne sadrži više od 5 funkcija, koji ima isto svojstvo. B' može imati i manje od 5 funkcija jer funkcija može da bude element dvije ili više klasa. Ne ograničavajući opštost, pretpostavimo da je

$$B' = \{f_0, f_1, f_M, f_S, f_L\},$$

pri čemu

$$f_0 \notin T_0, f_1 \notin T_1, f_M \notin M, f_S \notin S, f_L \notin L.$$

Takođe, možemo smatrati da sve te funkcije zavise od istih promjenljivih $x_1, x_2, \ldots, x_n$.

Dokaz dovoljnosti ćemo dati u tri etape.

1. Konstruisanje konstanti 0 i 1 pomoću funkcija f_0, f_1 i f_S.

Razmotrimo funkciju f_0 na n-torki $(1, \ldots, 1)$. Moguća su dva slučaja:

a) $f_0(1, \ldots, 1) = 1$. Tada funkcija $\varphi(x) = f_0(x, \ldots, x)$ jeste konstanta 1, jer je $\varphi(0) = f_0(0, \ldots, 0) = 1$ i $\varphi(1) = f_0(1, \ldots, 1) = 1$. Drugu konstantu dobijamo iz $f_1(1, \ldots, 1) = 0$.

b) $f_0(1, \ldots, 1) = 0$. U ovom slučaju funkcija $\varphi(x) = f_0(x, \ldots, x)$ jeste negacija, jer je $\varphi(0) = f_0(0, \ldots, 0) = 1$ i $\varphi(1) = f_0(1, \ldots, 1) = 0$. Posmatrajmo sada funkciju f_S. Kako imamo negaciju, prema lemi o nesamodualnoj funkciji, iz nje možemo dobiti konstantu. Iz dobijene konstante, kako imamo negaciju, možemo dobiti drugu konstantu.

Dakle, u oba slučaja smo dobili konstante 0 i 1.

2. Konstruisanje negacije pomoću konstanti 0 i 1 i nemonotone funkcije f_M. Ovo se izvodi pomoću leme o nemonotonoj funkciji.

3. Konstruisanje konjunkcije pomoću konstanti 0 i 1 i nelinearne funkcije f_L, eventualno koristiti i negaciju. Ovo se izvodi pomoću leme o nelinearnoj funkciji.

Dakle, pomoću formula nad B' (što znači i nad B) realizovali smo funkcije negaciju i konjunkciju. Slijedi da je sistem B kompletan, čime je dovoljnost dokazana.

Posljedica 1. Svaka zatvorena klasa funkcija iz P_2 koja je različita od P_2 sadrži se u bar jednoj od klasa T_0, T_1, S, M i L.

Klasa B funkcija iz P_2 naziva se *maksimalnom* ako ona nije kompletna i ako za proizvoljnu funkciju f iz P_2 koja nije u B, važi da je klasa $\{f\} \cup B$ kompletna.

Iz definicije slijedi da je svaka maksimalna klasa zatvorena.

Posljedica 2. U algebri logike postoji samo 5 maksimalnih klasa. To su T_0, T_1, S, M i L.

Kao primjer teoreme o funkcionalnoj kompletnosti, pokažimo da je kompletan sistem funkcija

$$f_1 = x_1x_2, \quad f_2 = 0, \quad f_3 = 1 \text{ i } f_4 = x_1 + x_2 + x_3.$$

Imamo: $f_3 \notin T_0, f_2 \notin T_1, f_2 \notin S, f_4 \notin M, f_1 \notin L$. S druge strane, uklanjanje proizvoljne funkcije daje nekompletan sistem:

$$\{f_2, f_3, f_4\} \subset L, \quad \{f_1, f_3, f_4\} \subset T_1,$$

$$\{f_1, f_2, f_4\} \subset T_0, \quad \{f_1, f_2, f_3\} \subset M.$$

Iz dokaza teoreme o kompletnosti neposredno slijedi sljedeća teorema.

Teorema 8. Iz svakog kompletnog u P_2 sistema B funkcija moguće je izdvojiti kompletni podsistem, koji sadrži ne više od četiri funkcije.

Dokaz. Vidjeli smo da je moguće iz B izdvojiti kompletni podsistem, koji sadrži ne više od pet funkcija. Pri tome smo vidjeli da je funkcija f_0, koja ne pripada klasi T_0, ili nesamodualna (slučaj a), jer je $f_0(0, \ldots, 0) = f_0(1, \ldots, 1)$, ili ne čuva jedinicu i nije monotona (slučaj b): $f_0(0, \ldots, 0) > f_0(1, \ldots, 1)$. U prvom slučaju kompletan je sistem $\{f_0, f_1, f_M, f_L\}$, a u drugom $\{f_0, f_S, f_L\}$.

Prethodni primjer pokazuje da se konstanta četiri ne može umanjiti.

Teorema o funkcionalnoj kompletnosti ne daje samo kriterijum kompletnosti. Ona omogućuje (uz korišćenje SDNF ili SKNF) predstavljanje proizvoljne funkcije f formulom preko funkcija kompletnog sistema B.

Opišimo još par rezultata koje je dao E. Post.

Sistem funkcija $\{f_1, f_2, \ldots, f_s, \ldots\}$ zatvorene klase B naziva se *kompletnim u B*, ako se njegovo zatvorenje poklapa sa B. Drugim riječima, sistem je kompletan u B ako svaka formula nad B može biti izražena u vidu formule kroz funkcije datog sistema.

Sistem funkcija $\{f_1, f_2, \ldots, f_s, \ldots\}$ zatvorene klase B naziva se *bazom B*, ako je on kompletan u B, pri čemu nijedan njegov podsistem nije kompletan u B.

Na primjer, sistem funkcija

$$f_1 = x_1x_2, \quad f_2 = 0, \quad f_3 = 1 \text{ i } f_4 = x_1 + x_2 + x_3$$

je baza u P_2. Moguće je dokazati da je sistem funkcija $\{0, 1, x_1\&x_2, x_1 \vee x_2\}$ baza klase M.

Teorema 9. Svaka zatvorena klasa u P_2 ima konačnu bazu.

Teorema 10. Kardinalnost skupa zatvorenih klasa u P_2 je prebrojiva.

Za sistem funkcija B kažemo da je *funkcionalno kompletan u slabom smislu* ako bilo koja Bulova funkcija može da bude prikazana pomoću formule nad skupom $B \cup \{0, 1\}$, tj. kao superpozicija konstanti i funkcija iz B. Vidi se da iz kompletnosti u običnom smislu slijedi kompletnost u slabom smislu.

Teorema 11. Sistem je kompletan u slabom smislu ako i samo ako u tom sistemu postoji bar jedna nemonotona i bar jedna nelinearna funkcija.

5.5 Zadaci za vježbu

1. Da li je skup A zatvorena klasa? a) $A = \{0, 1\}$; b) $A = \{\overline{x}\}$; c) $A = \{1, \overline{x}\}$; d) $A = \{x_1, x_1x_2, x_1x_2x_3, \ldots, x_1x_2 \ldots x_n, \ldots\}$; e) $A = \{0, x_1, x_1 \vee x_2, x_1 \vee x_2 \vee x_3, \ldots\}$; f) $A = \{x_1, x_1 \oplus x_2, x_1 \oplus x_2 \oplus x_3, \ldots\}$.

2. Svodeći neki sistem funkcija za koji je odranije poznato da je kompletan na zadati sistem funkcija A, dokazati da je zadati sistem funkcija A kompletan. Drugim riječima, izraziti svaku funkciju iz ranijeg skupa kao superpoziciju u kojoj učestvuju samo članovi skupa A. a) $A = \{xy \oplus z, (x \sim y) \oplus z\}$; b) $A = \{x \to y, \overline{x \oplus y \oplus z}\}$.

3. Dokazati: $x_1 \oplus x_2 \oplus \ldots \oplus x_n = 1$ ako i samo ako neparan broj promjenljivih x_1, x_2, ..., x_n ima vrijednost 1. Zato se i kaže "po modulu 2".

4. Metodom neodređenih koeficijenata odrediti polinom Žegalkina za funkciju. a) $f = 1001$. Uputstvo. Napisati opšti oblik za $n = 2$, pa prvo odrediti a, zatim b i c, na kraju d. b) $f = 01101000$. Uputstvo. Prvo odrediti a, zatim b, c i d, onda e, f i g, na kraju h. Za određivanje a služi $f(0, 0, 0)$. Za određivanje b služi $f(1, 0, 0)$; već je poznato a. Za određivanje c služi $f(0, 1, 0)$; već su poznati a i b. Itd.

5. Dokazati da je funkcija $h(x_1, x_2, x_3) = x_1x_2 \vee x_1x_3 \vee x_2x_3$ samodualna.

6. Iz nesamodualne funkcije f, tako što se na mjestima promjenljivih uvrste x i $\overline{x}$, dobiti konstantu. a) $f(x, y) = x \to y$; b) $f(x, y) = x \oplus y$; c) $f(x, y) = x \uparrow y$; d) $f(x, y, z) = 00111001$.

7. Neka je $A = \{1, x \oplus y\}$. Pokazati da je $[A] = L$.

8. Razlažući funkciju f u polinom Žegalkina, razjasniti da li je ona linearna. a) $f(x_1, x_2, x_3) = (x_1x_2 \vee \overline{x_1}\,\overline{x_2}) \oplus x_3$; b) $f(x_1, x_2) = x_1x_2(x_1 \oplus x_2)$.

9. Neka $f(x_1, x_2, x_3) \in L \cap S$, $f(0, 0, 0) = f(0, 0, 1)$ i $f(\alpha_1, \alpha_2, \alpha_3) = 0$. Čemu je jednako $f(\overline{\alpha_1}, \overline{\alpha_2}, \alpha_3)$?

10. Za koje n funkcija $f(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n \oplus 1$ pripada skupu $T_1 \setminus T_0$?

11. Naći funkciju $f(x, x, \ldots, x)$ ako je: a) $f(x_1, x_2, \ldots, x_n) \in T_1 \setminus T_0$; b) $f(x_1, x_2, \ldots, x_n) \in (L \cap S) \setminus T_1$; c) $f(x_1, x_2, \ldots, x_n) \in S \setminus T_0$.

12. Koje od ovih funkcija su monotone? a) $x \to (x \to y)$; b) $x \to (y \to x)$; c) $xy(x \oplus y)$; d) $xy \oplus yz \oplus zx \oplus z$; e) $f(x_1, x_2, x_3) = 00110111$.

13. Za one funkcije iz prethodnog zadatka koje nisu monotone postići ono što predviđa lema o nemonotonoj funkciji (umjesto $x_1, \ldots, x_n$ uvrste se 0, 1 i x na pogodan način, pa se tako dobije funkcija $\overline{x}$).

14. Koliko postoji monotonih funkcija $f(x_1, x_2, x_3)$ takvih da je $f(0, 1, 1) = f(1, 0, 1) = 1$ i $f(0, 0, 1) = 0$?

15. Izračunati broj funkcija u svakom od sljedećih skupova: a) $M \setminus (T_1 \cap T_0)$, b) $M \setminus (T_1 \cup T_0)$, c) $M \cap L$, ako se radi jedino o funkcijama koje zavise samo od promjenljivih $x_1, x_2, \ldots, x_n$.

16. Ako $f \in M$ onda i $f^* \in M$. Dokazati.

17. Koristeći (Postov) kriterijum kompletnosti (tj. teoremu o kompletnosti), razjasniti da li je zadati sistem A kompletan. a) $A = \{x \to y, x \to \overline{y} \cdot z\}$; b)

$A = \{x \cdot \overline{y}, \overline{x} \sim yz\}$; c) $A = \{0, 1, x(y \sim z) \vee \overline{x}(y \oplus z)\}$; d) $A = \{01101001, 10001101, 00011100\}$.

18. Dokazati da svaka funkcija iz klase L pripada bar jednoj od klasa T_0, T_1 i S.

5.6 Rješenja zadataka za vježbu

1. a) Ovaj skup $A = \{0, 1\}$ jeste zatvorena klasa. Zaista, dvije date funkcije imaju po nula argumenata pa praktično nije moguća nikakva superpozicija niti supstitucija na mjestu promjenljive, tako da se ne mogu dobiti nove funkcije.

b) Dati skup nije zatvorena klasa. Zaista, $\overline{x}$ s jedne strane jeste superpozicija u okviru onoga što imamo na raspolaganju a s druge strane $\notin A$.

c) Nije zatvorena klasa. Zaista, $\overline{1} = 0 \notin A$.

d) Odgovor – jeste.

e) Jeste. Zato što je disjunkcija komutativna, za nju važi i obrazac $x \vee x = x$, a i funkcija 0 koja je prisutna u datom skupu neće narušiti zatvorenost (neće se uz pomoć nje "izaći" iz A, dobiti nešto novo), jer je $0 \vee x = x \in A$.

f) Nije.

2. a) Od dvije date funkcije $xy \oplus z$ i $(x \sim y) \oplus z$ mi ćemo superpozicijama izgraditi funkcije $\overline{x}$ i $x \,\&\, y$, čime će dokaz biti završen, jer znamo da je sistem $\{\overline{x}, x \,\&\, y\}$ kompletan. Prvo, $(x \sim y) \oplus z$ daje $(x \sim x) \oplus z = 1 \oplus z = \overline{z}$; izgradili smo negaciju. Drugo, iz $xy \oplus z$ imamo $x\overline{y} \oplus x$ (imamo pravo da pišemo $\overline{y}$ jer smo maločas izgradili negaciju) $= x\overline{y} \oplus x1 = x(\overline{y} \oplus 1) = xy$; i konjunkciju smo izgradili.

b) Sada se svođenje vrši na kompletni sistem $\{x \vee y, \overline{x}\}$. Sljedeće dvije formule rješavaju postavljeni zadatak: $(x \to y) \to y = x \vee y$, $\overline{x \oplus x \oplus x} = \overline{x}$. Naravno da je korisno da se osvjedočimo da su te dvije formule ispravne. Uočite šablon koji je prisutan i u jednoj i u drugoj formuli: na lijevoj strani imamo izraz koji je dozvoljenog oblika (koriste se zadate funkcije i proizvoljno imena promjenljivih), a na desnoj strani imamo članove kompletnog sistema.

3. Dokaz se izvodi pomoću principa matematičke indukcije. Za $n = 1$ ovdje piše: $x_1 = 1$ ako i samo ako je jedna promjenljiva x_1 vrijednosti 1. Za $n = 2$

tvrđenje koje treba da bude dokazano glasi: važi $x_1 \oplus x_2 = 1$ tada i samo tada kada je među promjenljivim x_1, x_2 njih neparno jednako 1. I jedno i drugo (i $n = 1$ i $n = 2$) je tačno. Da bismo dokazali za $n > 2$, izvršimo indukcijski korak (prelazak sa n na $n + 1$).

Indukcijski korak lako se sprovodi - u osnovi se svodi na sljedeće: paran broj + 1 = neparan broj, neparan broj + 1 = paran broj. Na kraju, kada gledamo kolika je vrijednost izraza oblika $a \oplus b \oplus \ldots \oplus c$ samo treba prebrojati koliko je od učesnika $a, b, \ldots, c$ vrijednosti 1. Ako ih je neparno onda je izraz jednak 1, a ako ih je parno onda je čitav izraz jednak 0.

4. a) Konstruišimo tablicu funkcije f (slika 5.1). Opšti oblik polinoma po modulu dva (polinoma Žegalkina) za $n = 2$ glasi u opštim oznakama $f(x_1, x_2) = a \oplus a_1x_1 \oplus a_2x_2 \oplus a_{12}x_1x_2$ ili (prostije)

$$f(x_1, x_2) = a \oplus bx_1 \oplus cx_2 \oplus dx_1x_2.$$

$1 = f(0, 0) = a$ (iz tablice se vidi da je $f(0, 0) = 1$; iz ranijeg predstavljanja se vidi da je $f(0, 0) = a$; ostale sabirke iz tog predstavljanja ne pišemo jer konjunkcija u kojoj je bar nešto jednako nuli "propada"; pa $a = 1$). Slično iz $0 = f(1, 0) = 1 \oplus b$ imamo $b = 1$. Iz $0 = f(0, 1) = 1 \oplus c$ nalazimo da je $c = 1$ (iskoristili smo to što su a i b već određeni). Konačno, iz $1 = f(1, 1) = a \oplus bx_1 \oplus cx_2 \oplus dx_1x_2 = 1 \oplus 1 \oplus 1 \oplus d$ dobijamo $d = 0$ (gledamo da li ima paran ili neparan broj jedinica, vidi prethodni zadatak)

Odgovor: $f(x_1, x_2) = 1 \oplus x_1 \oplus x_2$.

x_1	x_2	f
0	0	1
0	1	0
1	0	0
1	1	1

Slika 5.1: Zadatak 4.a)

b) Opšti oblik polinoma Žegalkina za $n = 3$ glasi

$$f(x_1, x_2, x_3) = a \oplus bx_1 \oplus cx_2 \oplus dx_3 \oplus ex_1x_2 \oplus fx_1x_3 \oplus gx_2x_3 \oplus hx_1x_2x_3.$$

Bitan je redosljed određivanja koeficijenata $a, b, \ldots, h$; taj redosljed se ne poklapa sa redosljedom čitanja tablice. Prvo se određuje koeficijent a, drugo

oni uz koje stoji jedan faktor, zatim oni uz koje stoje dva faktora, na kraju h. Iz konstrukcije se vidi da polinom postoji i da je jedinstven. Dalje sami riješite ovaj zadatak.

5. Ovo se dokazuje bilo po definiciji samodualnosti bilo uz korišćenje principa dualnosti (naći izraz za dualnu funkciju h^*).

6. a) $f(x,y) = x \to y$. $f(x,x) = x \to x = 1$. Posljednja jednakost tj. $f(x,x) = 1$ predstavlja rješenje zadatka. Naravno da se treba osvjedočiti da je ispravno $x \to x = 1$. Dakle, na mjestu prve promjenljive stavili smo x, druge isto x, a poslije toga se ta funkcija svela na 1 (dobili smo konstantu, što se i tražilo).

Lema o nesamodualnoj funkciji nam garantuje da se cilj može postići, tj. možemo dobiti konstantu, ali nam sama po sebi ne daje "uputstvo" kako taj cilj postići. Naše je da nađemo "pravi" način.

Slično rasuđivanje treba imati u vidu i kada se govori o lemi o nesamodualnoj funkciji.

b) $f(x,y) = x \oplus y$. $f(x,x) = x \oplus x = 0$.

c) $f(x,y) = x \uparrow y$. $f(x,\overline{x}) = x \uparrow \overline{x} = 1$. Uvrstili smo x ili $\overline{x}$, a dobili smo konstantu. Osvjedočiti se da je relacija $x \uparrow \overline{x} = 1$ ispravna.

d) Pogledati dokaz leme. Naći trojku (α, β, γ) na kojoj je narušen uslov samodualnosti. Pa na mjestima x, y i z uvrstiti redom x^α, y^β i z^γ, znajući da je $x^0 = \overline{x}$, $x^1 = x$.

7. $A = \{1, x \oplus y\}$, da je $[A] = L$. Obije date funkcije su linearne tako da je $A \subset L$. Zato je $[A] \subset [L]$; praveći sve moguće superpozicije nad manjim skupom A stići će se do manjeg (ili do istog) nego ako se kompozicije prave nad L. Klasa L je zatvorena odnosno imamo $[L] = L$. Dakle $[A] \subset L$. S druge strane, lako se vidi da se pomoću kompozicija nad dvije zadate funkcije 1 i $x \oplus y$ može dobiti bilo koji linearni izraz kakvi su recimo $1 \oplus x_2 \oplus x_3$ ili $x_1 \oplus x_6 \oplus x_7 \oplus x_8$ (može se dobiti bilo koja linearna funkcija). Dokazano.

8. Uputstvo. Predstavljanje u obliku polinoma Žegalkina dobija se metodom neodređenih koeficijenata. Ako se u tom predstavljanju pojavljuje bar neki "nelinearni" član (x_1x_2 ili $x_1x_2x_3$ ili slično) onda ta funkcija nije linearna. U suprotnom slučaju jeste linearna.

9. f je linearna $\Rightarrow f = a \oplus a_1x_1 \oplus a_2x_2 \oplus a_3x_3$ (nema nelinearnih članova)

$f(0,0,0) = f(0,0,1) \Rightarrow a_3 = 0$ tj. $f = a \oplus a_1x_1 \oplus a_2x_2$ tj. f ne zavisi od x_3

f je samodualna $\Rightarrow f(\overline{\alpha_1}, \overline{\alpha_2}, \overline{\alpha_3}) = \overline{f}(\alpha_1, \alpha_2, \alpha_3) = \overline{0} = 1$

Kako nije bitno čemu je jednako x_3, to je i $f(\overline{\alpha_1}, \overline{\alpha_2}, \alpha_3) = 1$, što i jeste odgovor.

10. Treba da bude da je f u T_1 (da je $f(1,\ldots,1) = 1$) i da je f van T_0 (da je $f(0,\ldots,0) \neq 0$ odnosno $= 1$). Ovo drugo je uvijek ispunjeno jer je $0 \oplus 0 \oplus \ldots \oplus 0 \oplus 1 = 1$. Znamo pravilo o parnosti odnosno "mod 2". U vezi toga, izbrojimo da u izrazu $f(1,\ldots,1) = x_1 \oplus x_2 \oplus \ldots \oplus x_n \oplus 1$ ima $n+1$ jedinica, a taj izraz treba da bude $= 1$. Odgovor: za parne n.

11. a) Kako je dato $f(x_1, x_2, \ldots, x_n) \in T_1 \setminus T_0$, to je $f(x, x, \ldots, x) = 1$.

b) $f \in L \Rightarrow f = a \oplus a_1x_1 \oplus a_2x_2 \oplus \ldots \oplus a_nx_n$, gdje $a, a_1, \ldots, a_n \in \{0,1\}$, pa je zato $g(x) = f(x, x, \ldots, x) = a \oplus bx$ gdje $a, b \in \{0,1\}$; dva po dva x se "skrate" kod računanja $\oplus$. $f \in S$ pa je zato $g = 01$ ili $g = 10$. $f \notin T_1$ pa mora biti $g = 10$. Odgovor: $f(x, x, \ldots, x) = \overline{x}$.

c) $f \in S \Rightarrow g = 01$ ili $g = 10$. $f \notin T_0 \Rightarrow g = 10$. Odgovor: $g(x) = f(x, x, \ldots, x) = \overline{x}$. Zapaziti da iz $f(x_1, x_2, \ldots, x_n) \in S$ slijedi da i $g(x) = f(x, x, \ldots, x) \in S$, i slično.

12. a) $x \to (x \to y) = \overline{x} \vee (x \to y) = \overline{x} \vee \overline{x} \vee y = \overline{x} \vee y$. Sastavimo tablicu funkcije. Ispitati uslov $f(0,0) \leq f(0,1) \leq f(1,1)$, $f(0,0) \leq f(1,0) \leq f(1,1)$. Uslov nije ispunjen jer je $f(0,0) = 1$, $f(1,0) = 0$. Zaključujemo da razmatrana funkcija nije monotona.

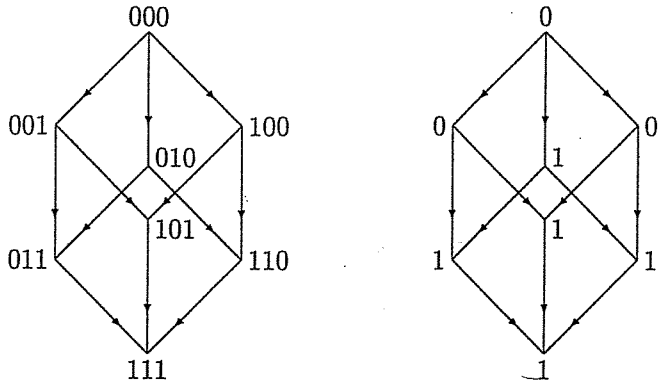
b) $x \to (y \to x) = \overline{x} \vee (y \to x) = \overline{x} \vee \overline{y} \vee x = 1 \vee \overline{y} = 1$, jeste monotona.

c) Važi jednakost $xy(x \oplus y) = 0$. Zaista, da bi vrijednost funkcije bila jednaka jedan, tj. da bi bilo $xy(x \oplus y) = 1$, treba $x = 1$, $y = 1$ i $x \oplus y = 1$, što je (zajedno) nemoguće, budući da je $1 \oplus 1 = 0$. Funkcija jeste monotona.

d) $f(1,1,0) = 1$, $f(1,1,1) = 0$. Nije monotona.

e) Vidi sliku 5.2. Nacrtajmo jediničnu kocku i prikažimo vrijednosti ove funkcije. U cilju ispitivanja monotonosti nacrtajmo i strelice koje povezuju uređene trojke koje su u relaciji $\preceq$. Zapaziti da smo povezali samo susjedne uređene trojke koje su u relaciji, a ne bilo koje uređene trojke koje su u relaciji. Ovo je u redu, budući da je $\preceq$ jedna tranzitivna relacija. Ispitati uslov: kad god je $A \preceq B$ tada je $f(A) \leq f(B)$. Jeste monotona.

13. a) $f(x,y) = x \to (x \to y) = \overline{x} \vee y$, $f(x,0) = \overline{x} \vee 0 = \overline{x}$.



Slika 5.2: Zadatak 12.e)

d) $f(x,y,z) = xy \oplus yz \oplus zx \oplus z$, $f(1,1,z) = 1 \oplus z \oplus z \oplus z = 1 \oplus z = \overline{z}$.

Funkcije pod b), c) i e) su monotone, pa nisu potrebna nikakva razmatranja.

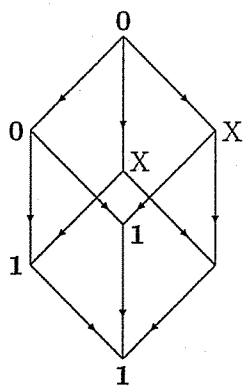
14. Ovaj zadatak možemo riješiti pomoću jedinične kocke i uslova $A \preceq B$ povlači $f(A) \leq f(B)$. Prvo treba prikazati zadate vrijednosti funkcije, a zatim upisati vrijednosti koje uslov određuje. Na kraju, ostale vrijednosti izabremo vodeći računa opet o uslovu. Odgovor: $N = 5$ (slika 5.3).

15. a) Funkcija f je po uslovu monotona, a ako $f \notin T_1$ onda za njenu vrijednost u zadnjem tjemenu jedinične n–dimenzione kocke imamo $f(1,\ldots,1) = 0$ pa odavde slijedi da mora biti $f \equiv 0$. Funkcija f je sigurno monotona, a ako $f \notin T_0$ onda $f(0,\ldots,0) = 1$, pa je zato $f \equiv 1$. Odgovor: $N = 2$.

b) $f \in M$, $f \notin T_1$, $f \notin T_0$. Odgovor: $N = 0$ (nijedna).

c) $f \in L$ pa $f = a \oplus a_1x_1 \oplus a_2x_2 \oplus \ldots \oplus a_nx_n$. Lako se vidi da prisustvo $x_1 \oplus x_2$ ili $x_2 \oplus x_4 \oplus x_6$ ili slično nije spojivo sa $f \in M$. Takođe i prisustvo $1 \oplus x_1$ onemogućava da $f \in M$. Tako da preostaju sljedeće mogućnosti: da bude $f = x_1$ ili ... ili $f = x_n$ ili $f = 0$ ili $f = 1$. Odgovor: $N = n + 2$.

16. Neka su α i β binarne promjenljive, tj. neka $\alpha, \beta \in \{0,1\}$. Lako se vidi da važi $\alpha \leq \beta \Leftrightarrow \overline{\alpha} \geq \overline{\beta}$. Odavde se lako dobija sljedeće: $(\alpha_1,\ldots,\alpha_n) \preceq (\beta_1,\ldots,\beta_n) \Leftrightarrow (\overline{\beta_1},\ldots,\overline{\beta_n}) \preceq (\overline{\alpha_1},\ldots,\overline{\alpha_n})$. Uvedimo oznake $\tilde{\alpha} = (\alpha_1,\ldots,\alpha_n)$, $\tilde{\beta} = (\beta_1,\ldots,\beta_n)$. Neka bude $\tilde{\alpha} \preceq \tilde{\beta}$. Tada



Slika 5.3: Zadatak 14

je $(\overline{\beta_1},\ldots,\overline{\beta_n}) \preceq (\overline{\alpha_1},\ldots,\overline{\alpha_n})$. Funkcija f je po uslovu monotona $\Rightarrow$ $f(\overline{\beta_1},\ldots,\overline{\beta_n}) \leq f(\overline{\alpha_1},\ldots,\overline{\alpha_n})$. Primijenimo komplement na posljednju relaciju:

$$\overline{f}(\overline{\alpha_1},\ldots,\overline{\alpha_n}) \leq \overline{f}(\overline{\beta_1},\ldots,\overline{\beta_n}). \qquad (*)$$

S druge strane, po definiciji dualne funkcije imamo:

$$f^*(\alpha_1,\ldots,\alpha_n) = \overline{f}(\overline{\alpha_1},\ldots,\overline{\alpha_n}), \quad f^*(\beta_1,\ldots,\beta_n) = \overline{f}(\overline{\beta_1},\ldots,\overline{\beta_n}).$$

Tako da $(*)$ govori da je

$$f^*(\alpha_1,\ldots,\alpha_n) \leq f^*(\beta_1,\ldots,\beta_n).$$

Mi smo pokazali da $\tilde{\alpha} \preceq \tilde{\beta}$ povlači $f^*(\tilde{\alpha}) \leq f^*(\tilde{\beta})$. Time smo mi po definiciji pokazali da je funkcija f^* monotona.

17. a) $A = \{x \to y, x \to \overline{y} \cdot z\}$. Za svaku od ove dvije funkcije treba razjasniti njen odnos prema svakoj od pet klasa T_0, T_1, S, L i M, tj. da li funkcija pripada ili ne pripada klasi. Kako se razjašnjava taj odnos – o tome smo govorili u prethodnim zadacima.

18. Dopustimo da postoji funkcija $f = f(x_1,\ldots,x_n)$ takva da $f \in L$, $f \notin T_0$, $f \notin T_1$ i $f \notin S$. $f \notin T_0$ pa $f(0,\ldots,0) = 1$. $f \notin T_1$ pa $f(1,\ldots,1) = 0$. $f \in L$ pa $f = a \oplus y_1 \oplus \ldots \oplus y_k$, gdje $y_1,\ldots,y_k \in \{x_1,x_2,\ldots,x_n\}$; $a \in \{0,1\}$; $y_i \neq y_j$ za $i \neq j$; $0 \leq k \leq n$. U vezi $f(0,\ldots,0) = 1$ imamo $a = 1$ tako da je

$$f = 1 \oplus y_1 \oplus \ldots \oplus y_k.$$

U vezi $f(1,\ldots,1)=0$ imamo da je broj k neparan. Recimo, $f=1\oplus x_6$ ili $f=1\oplus x_1\oplus x_3\oplus x_7$ ili slično.

$$f^*(x_1,\ldots,x_n)=\overline{f}(\overline{x_1},\ldots,\overline{x_n})=\overline{1\oplus\overline{y_1}\oplus\ldots\oplus\overline{y_k}}= \qquad \{\,\overline{u}=u\oplus 1\,\}$$

$$\overline{1\oplus(y_1\oplus 1)\oplus\ldots\oplus(y_k\oplus 1)}= \qquad \{\text{ paran broj jedinica se skrati }\}$$

$$\overline{y_1\oplus\ldots\oplus y_k}=1\oplus y_1\oplus\ldots\oplus y_k=f(x_1,\ldots,x_n)$$

Ovo govori da je $f^*=f$ tj. da $f\in S$. Dobili smo kontradikciju. Dokaz je završen.

Glava 6

Pojam digitalnog kola

Savremeni računari su digitalni uređaji. Njihov rad zasnovan je na realizaciji raznih funkcija pomoću digitalnih elemenata. U ovom poglavlju razmotrićemo osnovne pojmove digitalne elektronike.

6.1 Osnovni pojmovi

Elektronika je grana inženjerstva i primijenjene fizike, koja se bavi dizajnom i primjenom uređaja čiji rad zavisi od protoka elektrona u cilju generisanja, prenosa, prijema i skladištenja informacije. Najčešća klasa tih uređaja su elektronska kola, uređaji koji sadrže neprekidni put električne struje, sastavljen od provodnika i provodnih uređaja, uključujući izvor elektromotorne sile koji vodi struju kroz kolo. Kolo ovog tipa je zatvoreno kolo a kolo u kome put nije neprekidan je otvoreno kolo.

Uobičajeni termin za signal koji je kontinualan po vremenu i po amplitudi je *analogni signal.* Kola koja operišu sa analognim električnim signalima nazivaju se *analognim kolima.*

Jednu važnu klasu analognih signala predstavljaju *impulsni signali.* Oni su kontinualni po vremenu, ali im se amplituda može naglo mijenjati, pa signal u nekim slučajevima ne može imati bilo koju vrijednost iz dozvoljenog intervala. Primjeri impulsnih signala su periodične ili aperiodične povorke pravougaonih impulsa. Kola koja generišu ili obrađuju impulsne signale nazivaju se *impulsnim kolima.* Primjeri impulsnih kola su generatori impulsa i povorki impulsa, fliplopovi i tajmeri.

Digitalni signali su jedna uža klasa impulsnih signala koji imaju samo nekoliko dozvoljenih amplitudskih nivoa. Najčešće se koriste *binarni* digitalni signali, gdje su definisana samo dva različita naponska nivoa.

Električna kola koja obrađuju binarne digitalne signale nazivaju se *digitalnim kolima*.

Uvijek postoji određena tolerancija komponenti i napona napajanja, tj. teško je garantovati precizne naponske nivoe, pa se obično umjesto dva naponska nivoa definišu dva naponska *opsega*, koji se interpretiraju kao logička jedinica i logička nula. Naponski opsezi koji definišu logičku nulu i logičku jedinicu razdvojeni su *prelaznom zonom* u kojoj se nalaze signali koji ne predstavljaju ni logičku nulu ni logičku jedinicu, pa stoga nisu dozvoljeni u normalnom radu digitalnog kola.

Digitalna kola su, kao i analogna, sastavljena od aktivnih elemenata (tranzistori) i pasivnih elemenata (otpornici i vrlo rijetko kondenzatori). Analogna kola vrlo često se izrađuju i u diskretnoj tehnologiji, dok se digitalna kola danas prave isključivo u tehnologiji integrisanih kola.

Treba reći da su digitalna kola korišćena dosta prije integrisane tehnologije, pa i tranzistorske tehnologije. Prvi električni element koji je korišćen za realizaciju digitalnih kola bio je kontrolisani prekidač (relej). Sa pojavom elektronskih cijevi napravljena su impulsna i digitalna kola koja su omogućila veću brzinu rada. Prvi digitalni računar, napravljen početkom pedesetih godina, imao je sve digitalne elemente realizovane pomoću elektronskih cijevi. Sa pojavom tranzistora, digitalna kola se minijaturizuju i postaju brža. Glavni napredak u razvoju digitalnih kola došao je poslije pronalaska tehnologije integrisanih kola koja je omogućila smanjenje dimenzija i cijene, uz istovremeno povećanje brzine i kompleksnosti digitalnih kola.

Digitalna kola se prema načinu formiranja izlaznog signala dijele na *kombinaciona* i *sekvencijalna*. Kod kombinacionih digitalnih kola, signal na izlazu kola zavisi samo od trenutnih vrijednosti ulaznih signala. Kod sekvencijalnih kola, stanje na izlazu zavisi od trenutnog stanja na ulazima, ali i od prethodnih stanja na ulazima. Sekvencijalna kola se dalje dijele na *sinhrona* i *asinhrona*. Kod sinhronih kola, sve promjene dešavaju se istovremeno pod dejstvom kontrolnog signala takta. Kod asinhronih kola promjene se mogu dešavati u proizvoljnom trenutku i određene su samo osobinama upotrebljenih elemenata i vremenom pojavljivanja pobude.

Već smo kazali da se danas digitalna kola prave isključivo u tehnologiji integrisanih kola. *Integrisano kolo* je kolo napravljeno na jednom parčetu silicujuma. Parče se na engleskom kaže *chip*, pa otuda već odomaćena riječ *čip*. Integrisana kola imaju veliki broj (desetine i stotine miliona) tranzistora i veličine su nekoliko centimetara.

Bez obzira na veličinu, integrisano kolo je prvo dio većeg komada, kružnog oblika, koji se naziva *vaferom*. Vafer je veličine do 30-ak centimetara i sadrži određeni broj (nekoliko desetina ili nekoliko stotina, zavisno od veličine) istovjetnih kopija integrisanog kola. Iz vafera se izdvajaju pojedinačna integrisana kola i testiraju. Ona koja prođu test pakuju se u ambalažu, pri čemu se određeni broj signalnih linija (ulazne i izlazne) povezuje sa nožicama na ambalaži, takozvanim *pinovima*.

Kompletno integrisano kolo sa ambalažom i pinovima često se takođe naziva čipom, a nekad čak i integrisanim kolom. Iako ovi nazivi nisu korektni, dosta ljudi ih koristi.

Zavisno od broja elemenata integrisana kola se dijele na: kola sa malim stepenom integracije (*SSI*, small scale integration), koja imaju do 20 elemenata; kola sa srednjim stepenom integracije (*MSI*, medium scale integration), koja imaju od 20 do 200 elemenata; kola sa velikim stepenom integracije (*LSI*, large scale integration), koja imaju od 200 do 200,000 elemenata i kola sa vrlo velikim stepenom integracije (*VLSI*, very large scale integration), koja imaju preko 200,000 elemenata (tipično nekoliko miliona, danas se već radi o stotinama miliona).

6.2 Elementi digitalnih integrisanih kola

Digitalna koja konstruišu se od manjih elemenata, koje nazivamo *logičkim elementima*.

6.2.1 Logički element

Logički element je fizički uređaj (digitalno kolo) čija jednačina odgovara jednoj (tipično elementarnoj) Bulovoj funkciji. To je elementarno kolo, od takvih kola grade se složenija kola.

Na primjer, *i*–kolo ima dva ulaza a i b i ima izlaz y, s tim da je $y = a \,\&\, b$. Slično, invertor ima jedan ulaz a i ima izlaz y, s tim da je $y = \overline{a}$.

Može se govoriti o idealnom i o realnom logičkom elementu. Zamišljeni ili idealni logički element se skoro ne razlikuje od odgovarajuće logičke funkcije, jedino što ima svoj grafički simbol. Fizički ili realni logički elementi proučavaju se u okviru digitalne elektronike.

6.2.2 Idealni logički element

Idealni logički element realizuje jednu unaprijed zadatu logičku funkciju. Navedimo njegove glavne osobine.

U idealnom logičkom elementu logičkoj nuli odgovara napon nula, a logičkoj jedinici napon napajanja. Izlazni napon zavisi naravno od ulaznog napona, odnosno od ulaznih napona. Pod *karakteristikom prenosa* podrazumijeva se oblik te zavisnosti.

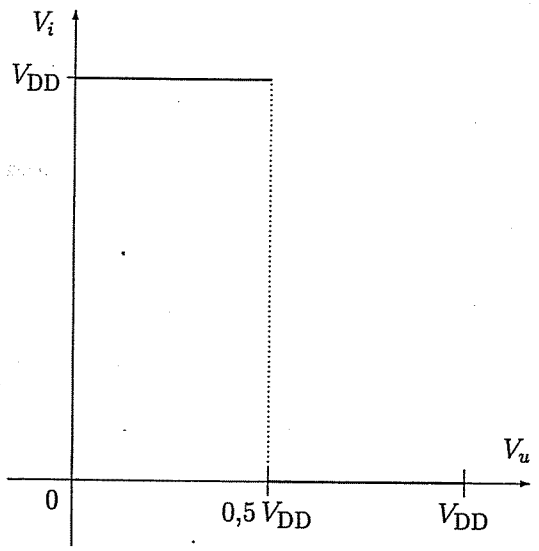
Prelazak izlaznog napona sa jednog na drugi nivo izvodi se naglo i to kada je ulazni napon jednak polovini napona napajanja.

Razmotrimo ovo na primjeru invertora. Kazali smo da je invertor digitalno kolo, logički element koji odgovara Bulovoj funkciji negaciji. Kod invertora važi da ako je ulazni signal visok, onda izlazni treba da bude nizak, i obrnuto. Na slici 6.1 prikazana je idealna karakteristika prenosa invertora.

Na slici su sa V_u i V_i označeni ulazni odnosno izlazni napon. Sa V_{DD} označen je napon napajanja, obično 5V (pet volti). Neformalno možemo reći da je ovo grafik funkcije $y = y(x)$ za koju važi: kada je $0 < x < 1/2$ tada je $y = 1$ i kada je $1/2 < x < 1$ tada je $y = 0$.

Ulazni i izlazni napon zavise od vremena t. Dinamičke karakteristike odnose se na oblik tih zavisnosti. Izlaz se odaziva ulazu tek poslije određenog vremena kašnjenja. Važno svojstvo idealnog logičkog elementa je da je vrijeme potrebno za prelazak iz jednog u drugo logičko stanje beskonačno kratko - kašnjenje je jednako nula sekundi.

Navedimo i druga svojstva idealnog logičkog elementa. Izlazna impedansa (otpornost) idealnog logičkog elementa jednaka je nuli, a njegova ulazna impedansa beskonačno je velika. To znači da element lako saopštava izlaznu vrijednost, a dešavanja unutar elementa ne odražavaju se na ulazne vrijednosti.



Slika 6.1: Idealna karakteristika prenosa invertora

Takođe smatramo da idealni logički element nema nikakvu potrošnju i da je njegova cijena nula.

Iz velikog broja razloga, koje nije teško razumjeti, idealne karakteristike ne mogu da budu zadovoljene u praksi, bez obzira o kakvoj tehnologiji se radi. Stoga moramo razmotriti karakteristike realnih elemenata koje mogu biti ostvarene u praksi.

6.2.3 Realni logički element

Kod idealnog logičkog elementa logičkoj jedinici odgovarao je napon napajanja. Realnije je uzeti da logičkoj jedinici odgovara jedan interval od v_3 do v_4 ili jedna *naponska zona* (jedan *naponski opseg*). Ako je naponski nivo (ako je amplitudski nivo) između v_3 i v_4 onda se ta vrijednost napona tumači kao logička jedinica.

Slično, i logičkoj nuli (umjesto idealne vrijednosti nula) možemo pridružiti jedan interval napona, recimo od v_1 do v_2.

Naravno da je $v_1 < v_2 < v_3 < v_4$. Za interval od v_2 do v_3 kaže se da obrazuje prelaznu zonu. Signali iz prelazne zone ne predstavljaju ni logičku nulu ni logičku jedinicu, pa prema tome nisu dozvoljeni kao rezultat u normalnom radu digitalnog kola.

Zavisno od familije logičkih kola, neutralni naponski nivo (napon nula, uzemljenje) ne mora da se odnosi ni na logičku nulu ni na logičku jedinicu.

Svako realno digitalno logičko kolo treba da zadovoljava bar neke osobine idealnih logičkih kola. Navedimo važnije:

1. Izlazni signal mora da bude unaprijed definisana jednoznačna funkcija ulaznih signala. Ta funkcija i predstavlja logičku funkciju kola.

2. Karakteristika prenosa ulaz–izlaz treba da bude jako nelinearna. Drugim riječima. karakteristika prenosa u prelaznoj zoni treba da bude strma što je više moguće. Tako da su onda normalni nivoi izlaznog napona skoncentrisani u dvije uske oblasti, u dva logička nivoa.

3. Prolaskom kroz logičko kolo nastaje regeneracija amplitudskih nivoa. Drugim riječima, nove ulazne vrijednosti svaki put će izazvati postavljanje izlaza na odgovarajući nivo, kada prođe vrijeme odziva kola.

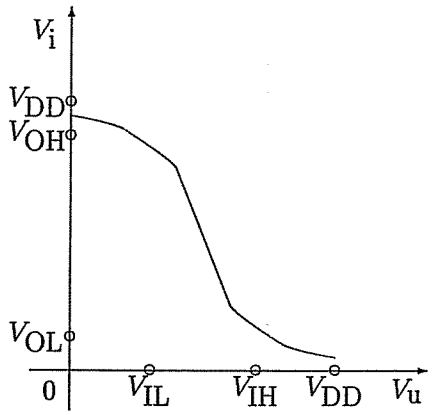
4. Logičko kolo treba da ima osobine jednostranosti i usmjerenosti. Promjena na izlazu kola ne treba da izaziva nikakvu naknadnu promjenu na ulazima kola. Signali se prostiru od ulaza prema izlazu.

5. Logičko kolo može da ima više ulaznih priključaka. Izlazni signal logičkog kola može da bude razveden odnosno može da bude poslat kao ulaz na više narednih kola.

Polazeći od osobina idealnog logičkog elementa i poželjnih karakteristika realnih logičkih elemenata, izvedeni su neki pokazatelji mjere vrijednosti stvarnih logičkih elemenata, odnosno definisane su njihove glavne ili osnovne karakteristike na koje treba obratiti pažnju. Navedimo osnovne karakteristike realnih logičkih elemenata.

1. Karakteristike prenosa

Karakteristike prenosa realnih logičkih elemenata samo aproksimiraju idealnu karakteristiku sa slike 6.1.



Slika 6.2: Realna karakteristika prenosa invertora

Na slici 6.2 prikazana je tipična karakteristika prenosa realnog invertorskog kola. Uočavaju se dvije bitne razlike između idealne i realne karakteristike prenosa. Prvo, prelazak sa jednog na drugi logički nivo nije jasno definisan, već postoji prelazna zona između stanja logičke nule i logičke jedinice. Drugo, nivo logičke nule nije jednak nula volti a nivo logičke jedinice nije jednak naponu napajanja.

Podsjetimo se da je kod invertora $y(x) = \overline{x}$, tj. $y(0) = 1$ i $y(1) = 0$ i da se tokom normalnog rada kola vrijednost signala može nalaziti u prelaznoj zoni samo tokom promjene logičkog stanja signala.

Značajne su četiri naponske vrijednosti V_{IL}, V_{IH}, V_{OL} i V_{OH}. Oznake I, O, L, H potiču od engleskih riječi *input, output, low, high*, koje znače *ulaz, izlaz, nizak, visok*. V_{IL} predstavlja najveći dozvoljeni napon na ulazu kola koji će se smatrati logičkom nulom (najveći dozvoljeni nivo logičke nule na ulazu). V_{IH} predstavlja najmanji dozvoljeni nivo logičke jedinice na ulazu kola. Naponski nivo V_{OL} predstavlja najveći dozvoljeni nivo logičke nule na izlazu kola (garantovana gornja granica za nulu na izlazu). Slično, napon V_{OH} predstavlja najmanji dopušteni nivo logičke jedinice na izlazu kola (garantovana donja granica za 1 na izlazu). Izlazni signali iznad V_{OH} smatraju se da su jednaki 1.

Na primjer, može da bude $V_{IL} = 0,6V$; $V_{IH} = 1,5V$; $V_{OL} = 0,1V$; $V_{OH} = 3,6V$; gdje je napajanje od 5V. Ove brojne vrijednosti odnose se na kola u bipolarnoj tehnologiji (TTL familija).

Uobičajeno je da se u složenim digitalnim kolima pomoću izlaza jednog logičkog kola pobuđuju ulazi narednih logičkih kola (izlaz iz kola K_1 dovodi se kao ulaz na kolo K_2). Da bi cio niz kola ispravno funkcionisao neophodno je da budu zadovoljeni uslovi $V_{OL} < V_{IL}$ i $V_{OH} > V_{IH}$.

2. Margine šuma

Neželjena promjena naponskog nivoa u čvoru odnosno u tački gdje su logički nivoi bitni (u tzv. logičkim čvorovima kola) naziva se šumom. Ako je amplituda šuma na ulazu logičkog kola mala, onda će izlaz biti ispravan. Ako je, pak, amplituda neželjene promjene na ulazu logičkog kola velika, onda ona može da izazove logičku grešku.

Pod pojmom margine šuma podrazumijeva se dozvoljena promjena naponskih nivoa na ulazu kola koja neće izazvati nepoželjnu promjenu na izlazu. Amplituda dozvoljene promjene zavisi i od logičkog stanja na ulazu, tako da postoje dvije margine šuma, NM_1 za logičku jedinicu i NM_0 za logičku nulu. Na osnovu ranije rečenog, margina šuma za logičku jedinicu iznosi $NM_1 = V_{OH} - V_{IH}$, a za logičku nulu $NM_0 = V_{IL} - V_{OL}$.

3. Faktor grananja na ulazu i izlazu

Ulazna impedansa realnog logičkog kola nije beskonačno velika, a njegova izlazna impedansa nije nula. Zbog toga se prilikom sprezanja logičkih kola (radi obrazovanja složenih digitalnih mreža) pojavljuje problem opterećivanja izlaza.

Postoje faktor grananja na izlazu i faktor grananja na ulazu.

Faktor grananja na izlazu kola (engl. *fan-out*) jeste broj nezavisnih ulaznih priključaka koji mogu da budu priključeni na izlaz tog kola, tj. na koliko mjesta može da se razvede izlazni signal logičkog kola, a da tako ne budu narušene dozvoljene varijacije logičkih nivoa. Označavamo ga sa N. Recimo, može da bude $N = 10$ (TTL familija).

Kada se računa faktor grananja na izlazu onda treba uočiti da sva kola ne opterećuju jednako prethodno kolo. Zato se u okviru jedne familije logičkih

kola definiše tzv. standardno opterećenje pomoću koga se određuje uticaj svakog ulaza na izlaz prethodnog kola.

Faktor grananja na ulazu (engl. *fan-in*) predstavlja dozvoljeni broj nezavisnih ulaznih priključaka u kolo. U većini slučajeva ograničen je samo praktičnim razlozima kao što je broj nožica na kućištu ili male potrebe za kolima sa velikim brojem ulaza i slično. Ipak, kod nekih familija logičkih kola broj ulaza ograničen je i zbog degradacije električnih karakteristika.

4. Dinamičke karakteristike

Pod dinamičkim karakteristikama podrazumijevaju se vremenske karakteristike, odnosno pokazatelji koji se odnose na vrijeme t.

Prelazak iz jednog u drugo logičko stanje kod realnog logičkog kola ne može se obaviti beskonačno brzo. Razlozi za to su višestruki. Prije svega, u svakom kolu postoje kapaciteti. Kao što je poznato iz teorije električnih kola, napon na kapacitetu ne može da se mijenja trenutno već se takve promjene vrše po eksponencijalnom zakonu. Osim toga, struje kroz elemente su konačne. Jačina struje je ograničena zahtjevima da kolo ima što manju disipaciju (potrošnju); da se kolo ne pregrije. Iz tih razloga, promjena nivoa na izlazu kola obavlja se za konačno vrijeme i kasni za promjenom nivoa na ulazu.

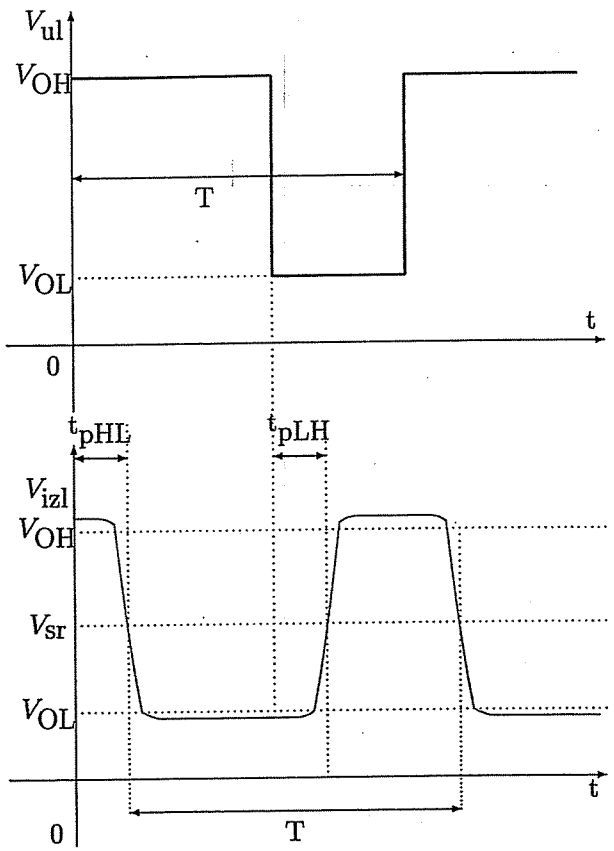
Kao ilustraciju posmatrajmo invertor. U prvom dijelu slike 6.3 vidimo idealizovani pobudni signal logičkog invertora, koji predstavlja povorku pravougaonih impulsa.

Izlazni signal realnog invertora imaće tipičan oblik, koji je takođe prikazan na slici 6.3.

Na vremenskim dijagramima izlaznog signala mogu se uočiti karakteristični intervali koji definišu kašnjenje odziva za pobudom.

Vrijeme kašnjenja opadajuće ivice t_{pHL} predstavlja vrijeme za koje opadajuća ivica izlaznog signala kasni za pobudom koja ju je izazvala. Definiše se kao vrijeme između trenutka promjene ulaznog signala i trenutka kada izlazni signal opadne do srednje vrijednosti $V_{sr} = (V_{OL} + V_{OH}) / 2$.

Vrijeme kašnjenja rastuće ivice t_{pLH} predstavlja vrijeme između trenutka promjene ulaznog signala i trenutka kada izlazni signal dostigne srednju vrijednost napona V_{sr}.



Slika 6.3: Odziv realnog invertora na idealizovanu pobudu

Vrijeme kašnjenja mjeri se jedinicama nano-sekundama. Recimo, može da bude jednako 10 ns.

Postoje i druge dinamičke karakteristike.

5. Disipacija

Svako realno logičko kolo mora imati neku potrošnju (*disipaciju*). Međutim, disipaciju kola nije uvijek lako odrediti, jer će se kolo (zavisno od logičkog stanja) nalaziti u različitim uslovima rada. Stoga se pri definiciji disipacije obično uzima da se kolo pobuđuje povorkom pravougaonih impulsa takvih da je trajanje impulsa i pauze jednako. Tada je struja izvora za napajanje jednaka aritmetičkoj sredini struja u jednom i drugom logičkom stanju.

Označimo sa V_{DD} napon napajanja i sa I_{DD} struju kroz izvor napajanja. Tada je prosječna snaga disipacije P_D jednaka

$$P_D = \frac{V_{DD} \cdot (I_{DDmin} + I_{DDmax})}{2}.$$

6. Proizvod snage i kašnjenja

Snaga disipacije P_D logičkog kola obično je povezana sa njegovom najvećom mogućom brzinom rada. Tako kola čija je brzina veća rade sa većim strujama. Zato se kod njih nepoželjni kapaciteti brže pune i prazne ili je radni režim tranzistora takav da je disipacija veća.

Imajući to u vidu, prilikom projektovanja logičkih kola uvijek se pravi određeni kompromis između potrošnje i brzine. Neka t_p označava vrijeme kašnjenja. Da se ocijeni mjera vrijednosti tog kompromisa, definiše se pokazatelj *proizvod snage i kašnjenja*, PDP (engl. power delay product). Kao što i naziv kaže, važi:

$$PDP = P_D \cdot t_p.$$

Navedeni kompromis je bolji ako je PDP manji. Veličina PDP izražava se očito u jedinicama $W \cdot sec = J$. Savremena logička kola imaju PDP reda veličine pJ, jer su tipične vrijednosti snage disipacije reda mW a tipične vrijednosti vremena kašnjenja su reda ns.

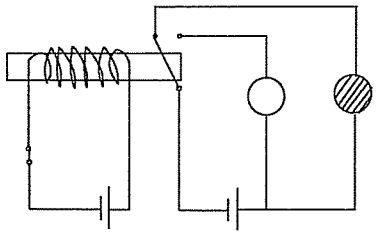
Vidjećemo da je vrijeme kašnjenja ili odziva logičkog elementa po pravilu znatno manje od taktnog vremena; taktno vrijeme ili perioda takta jednaka je primjera radi $1\mu s$, kada je učestanost takta jednaka 1MHz.

6.3 Tehnologija digitalnih kola

U digitalnoj elektronici korišćene su i koriste se razne tehnologije za izradu digitalnih kola. Opisaćemo neke od njih, koji se koriste kao 0-1 prekidači (engl. switch): relej, vakuumsku cijev i tranzistor.

6.3.1 Relej

Kod starih računara kao 0-1 prekidač koristio se relej. Njegovu ulogu je nešto kasnije preuzela vakuumska cijev. Relej ili kalem sastoji se od metalne šipke i namotaja žice oko nje (slika 6.4). Kada se kroz namotaje propusti struja, onda se stvara elektromagnetno polje što prouzrokuje pomjeranje šipke. Zato se uključi jedan ili više prekidača. S druge strane, kada prestane struja, šipka se vrati u polazni položaj i onda se prekidač isključi. Za vraćanje u polazni položaj koristi se opruga.



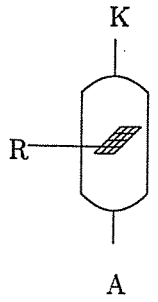
Slika 6.4: Relej

Prekidači releja mogu da se koriste da upravljaju drugim uređajima. Prekidači mogu da omoguće prolazak struje kroz namotaje drugih releja, itd. Tako se pružaju mogućnosti za složene logičke kombinacije, koje mogu da se upotrijebe u automatizaciji ili za uređaje za računanje.

6.3.2 Vakuumska cijev

Vakuumsku cijev čini jedna zatopljena staklena cijev u kojoj je vakuum, zajedno sa tri elektronska elementa - katodom, anodom, rešetkom (slika 6.5). Skraćenice za te elemente su K, A, R. Na elemente se može djelovati putem žice ili elektrode ili kontakta.

Ako se na katodu i anodu primijeni naponska razlika onda između njih teče struja, i to od katode prema anodi. Neka je sada između katode i anode



Slika 6.5: Vakuumska cijev

umetnuta rešetka. Tada se može upravljati strujnim tokom između anode i katode, mijenjajući napon na rešetki između visokog i niskog napona. Dakle, ako za napon na rešetki postoje dvije mogućnosti, onda vakuumska cijev djeluje kao prekidač u odnosu na katodu i anodu (u odnosu na svoje dvije pločice). Takvim prekidačem upravlja se elektronski.

Osim prekidačkog svojstva, vakuumska cijev ima i svojstvo strujnog pojačivača. Dovoljan je mali napon na rešetki da se postigne relativno velika struja između dvije pločice.

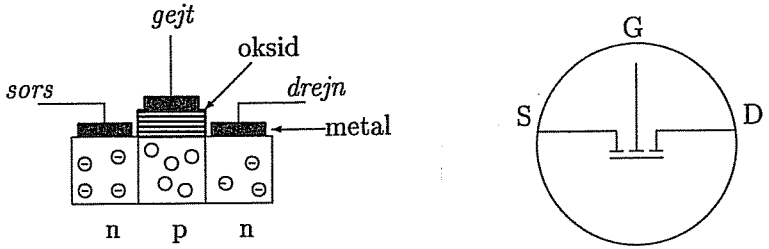
Vakuumska cijev zamijenjena je tranzistorom, koji se upotrebljava u savremenim integrisanim kolima.

6.3.3 Tranzistor

Tranzistor je nastao na osnovu tehnologije poluprovodnika. Materijali kao što su, na primjer, bakar i aluminijum su provodnici - provode struju sa malo otpora. S druge strane, postoje materijali koji ne provode struju - takva je recimo plastika. Poluprovodnici su slični provodnicima - sa malim uticajem može se postići da on provodi elektricitet. Primjeri takvih materijala su silicijum i germanijum.

Postoji nekoliko tipova tranzistora, opišimo najpopularniji od njih, tzv. $n-p-n$ tranzistor (slika 6.6). U njemu je između dva pojasa tipa n stavljen pojas tipa p. Prilikom izrade tranzistora poluprovodničkoj podlozi dodaju se primjese materijala. U jednom slučaju ti materijali postižu da se u određenim područjima (n) stvori višak elektrona, tj. višak negativnog punjenja. U dru-

gom slučaju stvaraju se područja (p) koja imaju manjak elektrona, odnosno kažemo da imaju *šupljine*, tj. višak pozitivnog punjenja.



Slika 6.6: MOS tranzistor i simbol MOS tranzistora

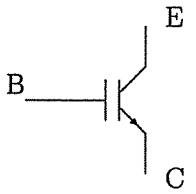
Oba n pojasa priključeni su na spoljašnji kontakt (elektrodu). I p sloj ima svoj spoljašnji metalni priključak, samo što je p pojas razdvojen od svoje elektrode jednim slojem oksida S_iO_2 (koji predstavlja izolator). Kaže se zato da su ovakvi tranistori napravljeni u MOS (metal oxide semiconductor) tehnologiji.

U tranzistoru postoje tri karakteristične tačke - sors, drain, gejt (engl. source, drain, gate). Sors i drejn se odnose na n pojas, a gejt na srednji pojas p.

Način rada tranzistora zavisi od gejta. Postoje dvije mogućnosti. Prva je da nije primijenjen napon na gejt. Tada srednji pojas djeluje kao prepreka između sorsa i drejna. Kroz tranzistor (između sorsa i drejna) tada ne protiče praktično nikakva struja. Druga mogućnost je da je na gejt doveden napon veći od tzv. prekidnog napona (praga napona). Prekidni napon može da bude, recimo, $1V$. Pod uticajem električnog polja, u srednjem pojasu elektroni bivaju privučeni u oblast neposredno ispod oksida. Dakle stvara se provodni put ili *kanal* između sorsa i drejna - sada su elektronska vrata otvorena. Sada, ako postoji potencijalna (naponska) razlika između sorsa i drejna onda će se pojaviti struja kroz kanal.

Prema tome, tranzistor se kratko može opisati na sljedeći način. Ako je gejt nula (tj. nema struje) tada su sors i drejn razdvojeni, a ako je gejt jednak 1 onda su oni kratko spojeni (ovdje treba uzeti u obzir i naponske nivoe na sorsu, onosno drejnu).

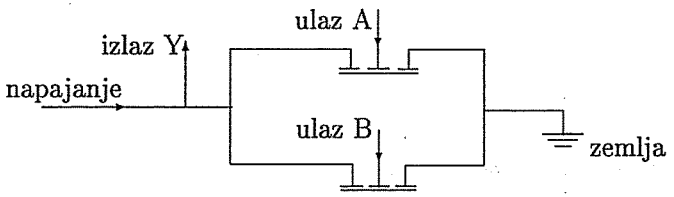
Slično važi i za transistor izrađen u bipolarnoj tehnologiji. Kod njega se sors, drejn i gejt nazivaju emiterom, bazom i kolektorom (engl. emitter, base, collector). Kada se baza optereti tada struja teče od emitera ka kolektoru. Simbol bipolarnog tranzistora dat je na slici 6.7.



Slika 6.7: Simbol bipolarnog tranzistora

6.3.4 Realizacija logičkih elemenata

Sada ćemo vidjeti kako se od tranzistora može sastaviti električno kolo koje odgovara nekoj unaprijed zadatoj (elementarnoj) Bulovoj funkciji. Za takvo kolo se kaže da predstavlja jedan *logički elelement*. Slika 6.8 prikazuje kako se kombinovanjem dva MOS tranzistora može sastaviti NILI kolo.



Slika 6.8: $Y = \overline{A \vee B}$

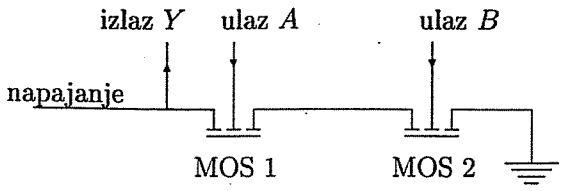
Izvor napajanja može, preko ulaza A ili ulaza B, da bude spojen sa uzemljenjem, čime bi se izlaz Y ostavio bez napona. Zaključujemo da izlaz predstavlja negaciju disjunkcije ulaza, tj.

$$Y = \overline{A \vee B}.$$

Izvor napajanja je jedna baterija od $5V$.

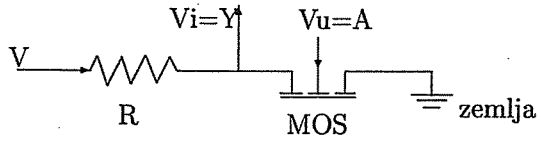
Na sljedećoj slici (6.9), dva tranzistora neće više biti postavljeni paralelno već serijski. Vidimo da se sada radi o negaciji konjunkcije, odnosno o NI kolu:

$$Y = \overline{A \& B}.$$



Slika 6.9: $Y = \overline{A\&B}$

Posmatrajmo sada slučaj sličnog kola samo sa jednim tranzistorom. Razmotrićemo ovaj slučaj malo detaljnije.



Slika 6.10: $Y = \overline{A}$

Na slici 6.10 je prikazan *invertor* sa MOS tranzistorom, kojem odgovara jednačina $Y = \overline{A}$ (negacija). Na slici je V - napon napajanja; R - otpor; V_u - ulazni napon; V_i - izlazni napon. Ako je ulazni napon visok (tj. $A = 1$) onda tranzistor provodi struju, tj. napajanje je spojeno sa zemljom , pa će u tački Y biti nizak napon ($Y = 0$). Slično, ako je $A = 0$ onda tranzistor ne provodi struju, napajanje je odvojeno od zemlje, izlaz postepeno biva preko otpornika opterećen od strane napajanja, tako da će postati $Y = 1$.

Otpornik služi da se izbjegne kratak spoj napajanja i uzemljenja. Kada je $A = 1$ disipacija je velika, zbog prisustva otpornika. Kada je $A = 0$ onda prisustvo otpornika usporava rad prikazanog kola, odnosno potrebno je dosta vremena da se uspostavi naponski nivo Y.

Mi smo ovdje govorili o MOS tranzistoru čiji je raspored $n - p - n$ tako da se obrazuje kanal tip N, zato je ovo NMOS tranzistor. Slično, PMOS tranzistor je sa rasporedom $p - n - p$. Bolje karakteristike imaju kola iz CMOS familije (complementary), kod kojih se na istoj podlozi formiraju PMOS i NMOS tranzistori. CMOS invertor se dobija od NMOS invertora

(koji se, kao što smo vidjeli, sastoji od NMOS tranzistora i otpornika), tako što se otpornik zamijeni jednim PMOS tranzistorom.

Ovi primjeri ilustrovali su kako se u tranzistorskoj tehnologiji mogu napraviti osnovni logički elementi. U daljem izlaganju manje ćemo obraćati pažnju na tehnologiju i familiju u kojoj je logičko kolo izgrađeno, a više na njegovu jednačinu, odnosno funkciju koju realizuje. Svakom osnovnom logičkom elementu odgovara njegov grafički simbol. U tabeli 6.1 dajemo pregled tih simbola.

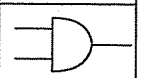
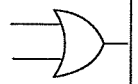
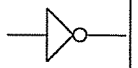
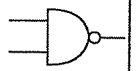
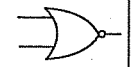
Funkcija	Naziv	Simbol
$Y = A\&B$	AND, I	
$Y = A \vee B$	OR, ILI	
$Y = \overline{A}$	NOT, NE	
$Y = \overline{A\&B}$	NI, NAND	
$Y = \overline{A \vee B}$	NILI, NOR	
$Y = A \oplus B$ $Y = A \underline{\vee} B$	XOR, EX-OR, exclusive OR ISKLJUČIVO ILI, ILI A ILI B	

Tabela 6.1: Osnovni logički elementi

6.4 Kombinacione i sekvencijalne mreže

Za *kombinacione* mreže (kola) karakteristično je da ne sadrže povratnu spregu. To znači da izlazni signal sa bilo kog kola u mreži ne smije da

bude doveden na ulaz mreže, kako ne bi uticao na ulaz tog istog kola. Sa druge strane, digitalne mreže sa povratnom spregom najčešće se ponašaju kao *sekvencijalne mreže.* Glavni primjeri kombinacionih mreža su sabirači, dekoderi, multiplekseri i magistrale.

Za praktičnu realizaciju kombinacione mreže neophodno je koristiti realne karakteristike logičkih kola, kako bi realizovana mreža pouzdano generisala zadatu funkciju. Karakteristike koje imaju najveći uticaj za ispravan i pouzdan rad logičke mreže su: a) familija logičkih kola; b) ulazno opterećenje; c) izlazni faktor grananja; d) vrijeme propagacije uzlazne i silazne ivice signala; i e) margine šuma logičke nule i jedinice.

Izbor familije logičkih kola uslovljen je u prvom redu zahtijevanom brzinom rada mreže, a zatim otpornošću na smetnje, potrošnjom električne energije i cijenom.

Sekvencijalne mreže (često nazivane i sekvencijalnim mašinama ili sekvencijalnim automatima) razlikuju se od kombinacionih mreža po tome što izlazni signali iz sekvencijalnih mreža zavise ne samo od tekućih vrijednosti ulaznih promjenljivih, već i od redosljeda (sekvence) generisanja ulaznih signala. Činjenica da je potrebno pamtiti redosljed aktiviranja ulaznih signala, uslovljava da se u sekvencijalnim mrežama moraju koristiti memorijski elementi.

Kao memorijski elementi u sekvencijalnim mrežama mogu da se koriste sve vrste flipflopova ili leč kola. Ako svi flipflopovi u mreži istovremeno mijenjaju stanje nakon uzlazne ili silazne ivice zajedničkog taktnog impulsa onda je mreža *sinhrona.* Ulazni signali u sinhronu mrežu odnosno memorijski elementi sinhrone mreže ne smiju da mijenjaju stanje za vrijeme dok je taktni impuls aktivan. Ako se u mreži koristi više od jednog taktnog impulsa ili ako neki od memorijskih elemenata može da promijeni stanje (promjenom neke od ulaznih promjenljivih) nezavisno od taktnog impulsa, onda je mreža *asinhrona.*

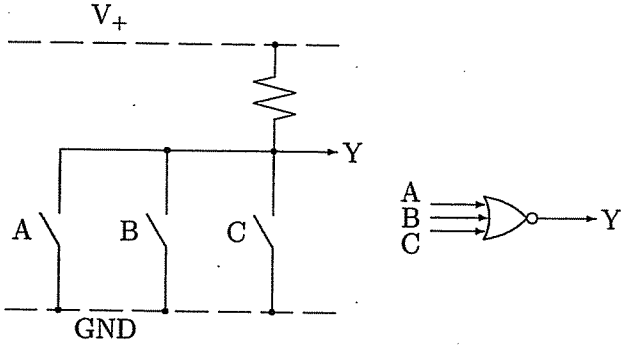
Glavni primjeri sekvencijalnih mreža su stacionarni registri, pomjerački registri i brojači.

Kao što ćemo uskoro vidjeti, dizajniranje digitalnih kola je naporan posao. Zato su razvijeni posebni programi koji omogućuju grafičko unošenje shema. Ovi alati olakšavaju dizajn, analizu, kao i testiranje digitalnih kola putem simulacije. Neki od njih posjeduju posebno razvijene jezike (na primjer VDHL) za specifikaciju digitalnih kola.

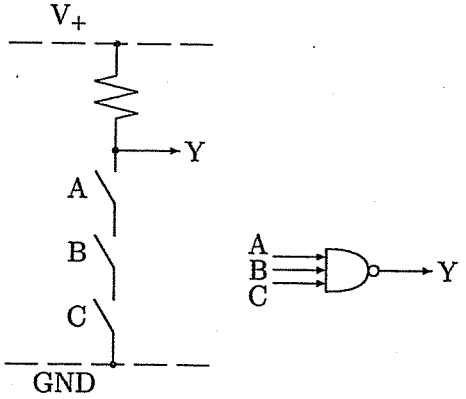
6.5 Zadaci za vježbu

U ovoj sekciji navodimo dva primjera logičkih elemenata. Na osnovu šeme sastavićemo jednačinu.

1. Na slici je prikazana mreža koja ima tri ulazna prekidača A, B i C i ima jednu izlaznu liniju Y (nizak napon ili visok napon). Sa V_+ označen je napon napajanja a sa GND označeno je uzemljenje. Lako se vidi da važi jednačina $Y = \overline{A \vee B \vee C}$, tako da smo konstruisali jedan NILI–element. Grafički simbol elementa nacrtan je pored šeme.



2. V. šemu i grafički simbol. Lako se vidi da za razmatranu mrežu važi jednačina $Y = \overline{A \& B \& C}$, tako da imamo jedan NI–element.



Glava 7

Primjeri digitalnih mreža

U ovom poglavlju ćemo razmotriti glavne primjere digitalnih mreža, kako kombinacionih, tako i sekvencijalnih.

Kombinaciona ili logička mreža održava jednu ili više logičkih, tj. Bulovih, funkcija. Logičke mreže su sastavljene od logičkih elemenata koji odražavaju pojedine Bulove funkcije, recimo I, ILI, NE, ISKLJUČIVO ILI.

Možemo razdvojiti nekoliko faza u procesu konstrukcije logičke mreže. To su:

- definisanje zavisnosti izlaznih od ulaznih promjenljivih;
- definisanje tablica;
- pravljenje izraza;
- sinteza;
- dodjela simbola.

Definicijom mreže zadaju se njene ulazne i izlazne promjenljive, i definiše zavisnost izlaznih promjenljivih od ulaznih. Na osnovu definisane zavisnosti, za svaku izlaznu promjenljivu pravi se tablica odgovarajuće Bulove funkcije. Iz tablice se, zatim, sastavlja funkcionalni izraz (ili izrazi) u kome učestvuju logički elementi koji nam stoje na raspolaganju, recimo I, ILI, NE, ISKLJUČIVO ILI. Uopšte uzevši, mogući su razni izrazi koji odražavaju zavisnost izlazne promjenljive od ulaznih. Nastojimo da izraz bude što je moguće

jednostavniji, jer će tada i rezultujuća mreža biti jednostavnija. Najzad, slijedeći napisani izraz, vršimo sintezu mreže. Mreža se posmatra kao jedna cjelina, pa joj se obično dodjeljuje neki simbol, grafički znak ili naziv.

7.1 Sabirači

Kao prvi primjer kombinacione mreže razmotrićemo mrežu koja omogućuje da se izvrši sabiranje dva binarno zapisana broja. Riješićemo tri varijante: a) sabirač; b) potpuni sabirač; c) paralelni sabirač. Nazivi na engleskom, koji se često koriste, su semi-adder, full adder i parallel adder.

7.1.1 Sabirač

Sabirač ima dvije ulazne promjenljive, x i y, i dvije izlazne promjenljive, z i q. Ulazne promjeljive predstavljaju dvije binarne cifre koje sabiramo, z je zbir, a q je prenos. Dakle, radi se o jednocifrenom binarnom sabiranju. Zavisnost izlaznih od ulaznih promjenljivih možemo zadati tabelom 7.1.

x	y	z	q
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabela 7.1: Tabela za sabirač

Lako se vidi da su odgovarajući izrazi za funkcije $z = z(x, y)$ i $q = q(x, y)$ dati formulama:

$$z = x \oplus y$$

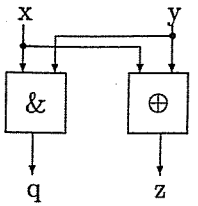
i

$$q = xy.$$

Sada možemo izvršiti sintezu - mreža sabirača data je na slici 7.1.

7.1.2 Potpuni sabirač

Kod *potpunog sabirača*, ulazne promjenljive su x,y,p, i one označavaju cifru prvog sabirka, cifru drugog sabirka i stari prenos. Izlazne promjenljive su z



Slika 7.1: Mreža sabirača

i q - odgovarajuća cifra zbira i novi prenos. Dakle, potpuni sabirač realizuje sabiranje dva binarna broja na jednoj koordinati. Na rezultat koji je jedna binarna cifra i prenos, utiču cifre koje se sabiraju i stari prenos.

Zavisnost izlaznih od ulaznih promjenljivih kod potpunog sabirača možemo zadati tabelom 7.2.

x	y	p	z	q
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

Tabela 7.2: Tabela za potpuni sabirač

Možemo lako provjeriti da važe izrazi:

$$z = x \oplus y \oplus p$$

i

$$q = (x \oplus y)p \vee xy.$$

Mreža punog sabirača data je na slici 7.2.

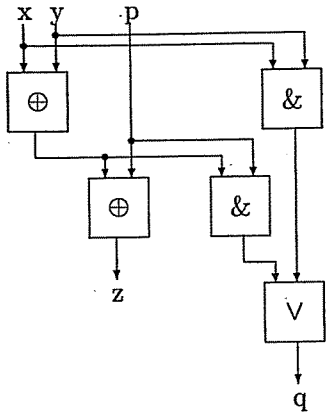
Primijetimo da mogu da posluže i sljedeći izrazi, u kojima se pojavljuje pomoćna veličina f:

$$f = \overline{x}y \vee x\overline{y},$$

$$z = \overline{f}p \vee f\overline{p},$$

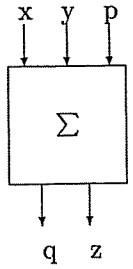
$$q = fp \vee xy.$$

Naravno, ukoliko bismo za sintezu koristili ove izraze, mreža bi izgledala drugačije nego na slici 7.2.



Slika 7.2: Mreža potpunog sabirača

Najzad, kako ćemo potpuni sabirač koristiti niže, uvedimo njegov grafički simbol (slika 7.3).



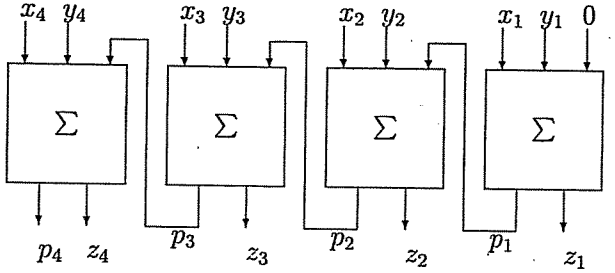
Slika 7.3: Grafički simbol potpunog sabirača

7.1.3 Paralelni sabirač

Paralelni sabirač je kombinaciona mreža koja odgovara operaciji binarnog sabiranja dva binarno zapisana broja.

Pretpostavimo da sabirci imaju po 4 bita. Označimo cifre prvog sa x_4, x_3, x_2, x_1 a drugog sa y_4, y_3, y_2, y_1. Cifre rezultata označimo sa z_4, z_3, z_2, z_1. Dakle, treba da bude $x_4x_3x_2x_1 + y_4y_3y_2y_1 = z_4z_3z_2z_1$, na primjer $0100 + 0101 = 1001$.

Za konstrukciju paralelnog sabirača možemo upotrijebiti četiri potpuna sabirača $\sum$, kao što je prikazano na slici 7.4.



Slika 7.4: Paralelni sabirač

Primijetimo nekoliko važnih stvari. Kao prvo, signal za prenos, koji se kaskadno prenosi od bita najmanje težine do bita najveće težine, usporava rad mreže. Za sabirač sa slike 7.4, u slučaju da je jedan sabirak 1111 a drugi 0001, izlazni signal prenosa p_4 će kasniti u odnosu na postavljanje ulaznih signala za vrijeme $4t$, gdje je t vrijeme propagacije signala za prenos kroz jedan stepen potpunog sabirača.

Uočimo još da je umjesto krajnjeg desnog potpunog sabirača $\sum$ moguće koristiti sabirač.

Najzad, ako je izlaz $p_4 = 1$, onda se time saopštava da je došlo do prekoračenja, tj. da zbir ima više od 4 cifre.

7.2 Dekoder

Kao drugi primjer logičke (kombinacione) mreže razmotrimo potpuni binarni dekoder. *Dekoder* je kombinaciona mreža sa određenim brojem ulaza i izlaza, pri čemu svaka dozvoljena kombinacija ulaznih promjenljivih aktivira tačno jedan izlaz. Za dekoder sa n ulaza kažemo da je *potpun* ako ima 2^n izlaza, odnosno da je *nepotpun* ako ima manje od 2^n izlaza. Potpuni dekoderi često se nazivaju i *binarnim*, zato što tada ulazne promjenljive možemo shvatiti kao binarno kodiran broj.

Dakle, potpuni binarni dekoder ima n ulaznih veličina, označimo ih kao $x_1, \ldots, x_n$, i $N = 2^n$ izlaznih veličina, koje možemo označiti sa $y_1, \ldots, y_N$. Sada je potrebno definisati zavisnost izlaznih od ulaznih promjenljivih.

Sve izlazne promjenljive treba da su jedake nuli osim jedne koje treba da je jednaka jedinici. Koja od 2^n izlaznih promjenljivih treba da je jednaka jedinici zavisi od ulaznih promjenljivih. Najčešće se izrađuju dekoderi kod kojih je $n = 2, n = 3$, ili $n = 4$.

Klasičan primjer upotrebe dekodera kod digitalnih računara je raspoznavanje naredbe. Naime, rad računara se odvija po programu koji je zapisan u mašinskom jeziku. Jedna po jedna naredba dolazi na red za izvršavanje u procesoru. Naredbe su binarno zapisane (kodirane) i procesor treba da rastumači (dekodira) koja naredba treba da se izvrši, tj. pusti tačno jedan kontrolni (upravljački) signal koji odgovara toj naredbi. Na osnovu tog upravljačkog signala počeće da se izvršava naredba.

Potrebno određivanje koja je naredba sljedeća izvodi se pomoću dekodera. Ako pretpostavimo da su naredbe nekog računara kodirane sa četiri bita, tj. oblika su $x_1x_2x_3x_4$, dekoder ukupno ima 4 ulaza ($n = 4$). Različitih naredbi tada ima $N = 2^4$, što znači da dekoder ima 16 izlaza.

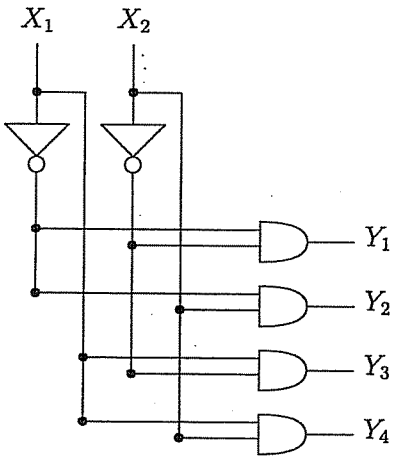
Zbog jednostavnosti, razmotrimo prvo dekoder u slučaju kada je $n = 2$. Mreža treba da bude takva da prilikom različitih kombinacija ulaza x_1 i x_2 uvijek jedan od izlaza y_1, y_2, y_3, y_4 bude aktivan (jednak jedan), a ostala tri neaktivni (jednaki nuli). Naravno, očekujemo i da različite kombinacije vrijednosti ulaznih promjenljivih treba da daju različite izlaze.

A_1	A_0	Y_1	Y_2	Y_3	Y_4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Kako znamo svojstva Bulovih funkcija, lako je sastaviti potrebne jednačine korišćenjem funkcija $\&, \vee, \overline{x}$. Na primjer, znamo da funkcija $f = x_1\&x_2$ ima svojstvo da je jednaka jedinici samo za jednu kombinaciju - kada je $x_1 = 1, x_2 = 1$. Slično svojstvo ima funkcija $f = x_1\&\overline{x}_2$ - jednaka je jedinici samo za kombinaciju $x_1 = 1, x_2 = 0$. Zaključujemo da željene jednačine dekodera glase:

$$y_1 = \overline{x}_1\&\overline{x}_2, \quad y_2 = \overline{x}_1\&x_2, \quad y_3 = x_1\&\overline{x}_2, \quad y_4 = x_1\&x_2.$$

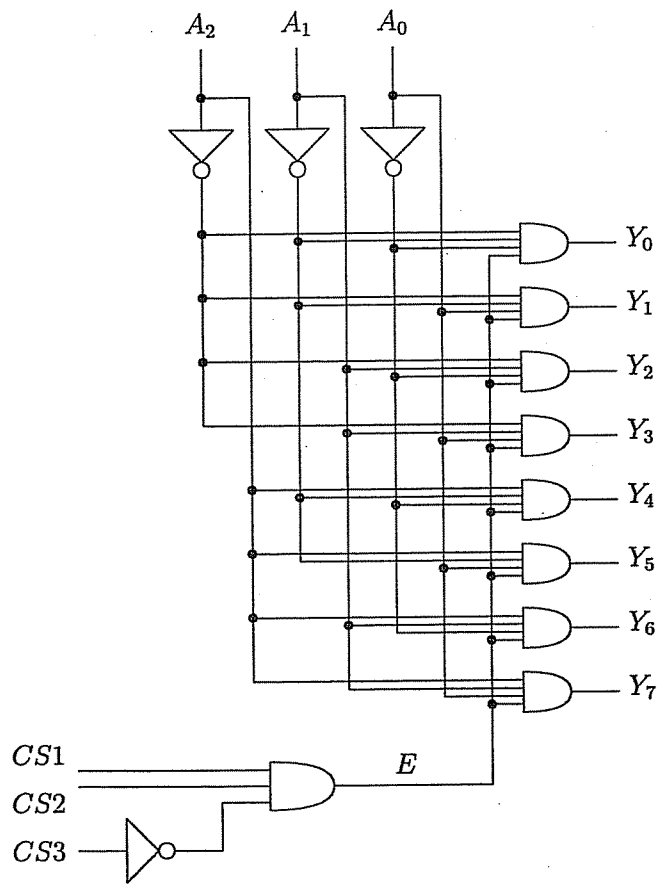
Sada je lako sastaviti šemu dekodera (slika 7.5).

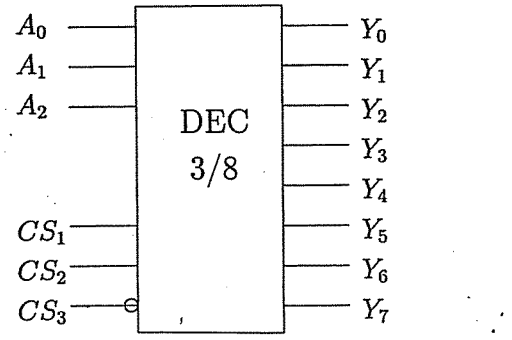


Slika 7.5: Dekoder 2/4

Razmotrimo slučaj kada je $n = 3$, tj. konstruišimo 3/8 dekoder. Označimo njegove ulaze kao A_0, A_1, A_2 a izlaze kao $Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7$. Analogno sa prethodnim slučajem, jednačine možemo zapisati kao:

$$Y_0 = \overline{A}_2\overline{A}_1\overline{A}_0, \quad Y_1 = \overline{A}_2\overline{A}_1A_0, \quad Y_2 = \overline{A}_2A_1\overline{A}_0, \quad Y_3 = \overline{A}_2A_1A_0,$$





Slika 7.6: Šema i simbol dekodera 3/8

$$Y_4 = A_2\overline{A}_1\overline{A}_0,\ \ Y_5 = A_2\overline{A}_1A_0,\ \ Y_6 = A_2A_1\overline{A}_0,\ \ Y_7 = A_2A_1A_0.$$

A_2	A_1	A_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Često se izlazi dekodera uslovljavaju postojanjem sinhronizacionog signala E. Signal E, koji se naziva i *signalom dozvole* (engl. enable), treba držati na nultom (neaktivnom) nivou za vrijeme dok ulazne promjenljive mijenjaju vrijednost, a na nivou 1 (aktivan) kada su ulazne promjenljive stabilne. Za $E = 0$, bez obzira na kombinacije vrijednosti ulaznih promjenljivih, izlazi dekodera će biti neaktivni. Sa ovim dodatkom, prethodne jednačine postaju:

$$Y_0 = \overline{A}_2\overline{A}_1\overline{A}_0E,\ \ Y_1 = \overline{A}_2\overline{A}_1A_0E,\ \ Y_2 = \overline{A}_2A_1\overline{A}_0E,\ \ Y_3 = \overline{A}_2A_1A_0E,$$

$$Y_4 = A_2\overline{A}_1\overline{A}_0E,\ \ Y_5 = A_2\overline{A}_1A_0E,\ \ Y_6 = A_2A_1\overline{A}_0E,\ \ Y_7 = A_2A_1A_0E.$$

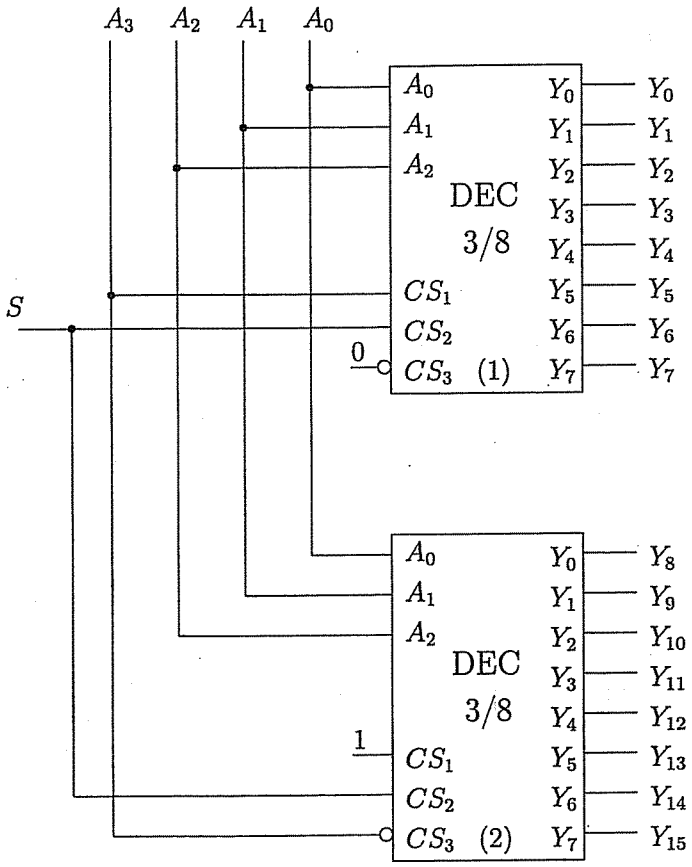
Sa svoje strane, signal E generiše se kombinacijom kontrolnih signala tipa CS odnosno signala *selekcije čipa* (engl. chip select). Uzmimo da u konkretnom slučaju imamo tri takva signala, CS_1, CS_2, CS_3, koji su vezani tako da je

$$E = CS_1 \cdot CS_2 \cdot \overline{CS}_3.$$

Šema ovako definisanog dekodera data je na slici 7.6.

Dekoderi se u šemama digitalnih sistema označavaju pravougaonim simbolima sa naznačenim ulaznim, izlaznim i kontrolnim signalima. Unutar pravougaonika upisuje se vrsta dekodera. Simbol konstruisanog 3/8 dekodera takođe je dat je na slici 7.6.

Dekoderi se, kao integrisane komponente, izrađuju u različitim pakovanjima, odnosno sa različitim brojem ulaza i izlaza. Najčešće su 2/4, 3/8 i 4/16 dekoderi.



Slika 7.7: Dekoder 4/16 realizovan dekoderima 3/8

Kaskadnim vezivanjem više dekodera moguće je ostvariti dekodovanje binarnih brojeva sa proizvoljnim brojem cifara. Kontrolni ulazi mogu se koristiti za međusobno vezivanje dekodera kako bi se dekodovale višebitne ulazne riječi. Tako se, recimo, četvorobitni binarni broj može dekodirati korišćenjem dva 3/8 dekodera (slika 7.7).

Signal bita najveće težine (A_3) dovodi se na kontolni ulaz CS_1 prvog dekodera i $\overline{CS}_3$ drugog dekodera. Time se postiže da kada je $A_3 = 1$ bude aktivan samo prvi dekoder, a kada je $A_3 = 0$ - drugi. Tada se dekodiranje svodi na 3/8 dekoder sa ulazima A_2, A_1, A_0.

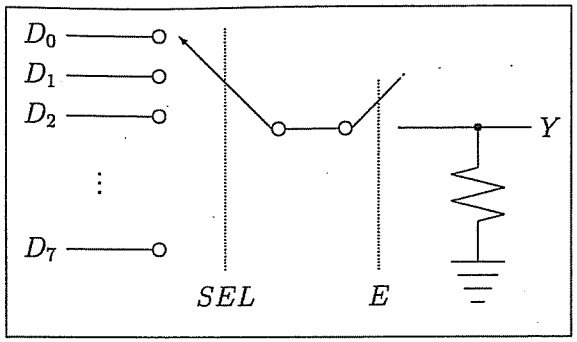
Kontrolni ulaz CS_2 može da se koristi za priključivanje zajedničkog sinhronizacionog signala S. Očigledno, na kontrolni ulaz $\overline{CS}_3$ prvog dekodera treba dovesti signal 1, a na kontrolni ulaz CS_1 drugog dekodera signal 1.

Na kraju, napomenimo da postoji i *enkoder*, mreža koja je inverzna mreži dekodera. Dakle, ulazne promjenljive enkodera su $y_1, y_2, \ldots, y_N$, a njegove izlazne promjenljive su $x_1, \ldots, x_n$. Ovdje je $n \geq 1$ i $N = 2^n$. Jasno, sve nabrojane ulazne i izlazne veličine su bulovske, što znači da uzimaju vrijednosti iz skupa $\{0, 1\}$. Važe iste zavisnosti između promjenljivih koju su specifikovane na početku ovog naslova. Naravno, razlika je u tome što sada treba promjenljive $x_1, \ldots, x_n$ izraziti preko $y_1, y_2, \ldots, y_N$, a ne obrnuto, kao kod dekodera. Tako, na primjer, u slučaju $n = 3$ jednačine enkodera imaju oblik $x_1 = x_1(y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$, $x_2 = x_2(y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$ i $x_3 = x_3(y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$. Za vježbu, sastavite te tri jednačine.

7.3 Multiplekser

Multiplekser je kombinaciona mreža koja obavlja funkciju digitalnog višepoložajnog prekidača. Na slici 7.8 prikazana je funkcionalna šema osmoulaznog multipleksera.

Signalom selekcije SEL sa slike 7.8 prekidač se postavlja u željeni položaj tako da se bira tačno jedan od osam ulaznih signala. Izabrani ulazni signal prenosi se na izlaz Y. Na slici vidimo da postoji još i signal dozvole E. Dakle, možemo kazati da željeni ulazni signal biva priključen na izlazni priključak Y ukoliko je aktiviran prekidač dozvole E. Ako je E neaktivan onda će izlaz biti logička nula, bez obzira na položaj prekidača SEL i vrijednosti ulaznih promjenljivih D_0 do D_7.



Slika 7.8: Funkcionalna šema osmoulaznog multipleksera.

Kako imamo 8 ulaznih mogućnosti, signal SEL sastoji se od tri binarna signala S_0, S_1 i S_2. Na primjer, ako je $S_0 = S_1 = S_2 = 0$, tada se na izlaz priključuje signal D_0, a ako je $S_0 = S_1 = S_2 = 1$ - signal D_7. Naravno, ovo važi samo ako je $E = 1$.

Na osnovu ove definicije multipleksera, sada treba sastaviti njegovu logičku funkciju, tj. izraz za Y.

$$Y = (D_0 \overline{S_2}\,\overline{S_1}\,\overline{S_0} \vee D_1 \overline{S_2}\,\overline{S_1}\,S_0 \vee D_2 \overline{S_2}\,S_1\,\overline{S_0} \vee D_3 \overline{S_2}\,S_1\,S_0 \vee$$
$$D_4\,S_2\,\overline{S_1}\,\overline{S_0} \vee D_5\,S_2\,\overline{S_1}\,S_0 \vee D_6\,S_2\,S_1\,\overline{S_0} \vee D_7\,S_2\,S_1\,S_0) \cdot E \qquad (*)$$

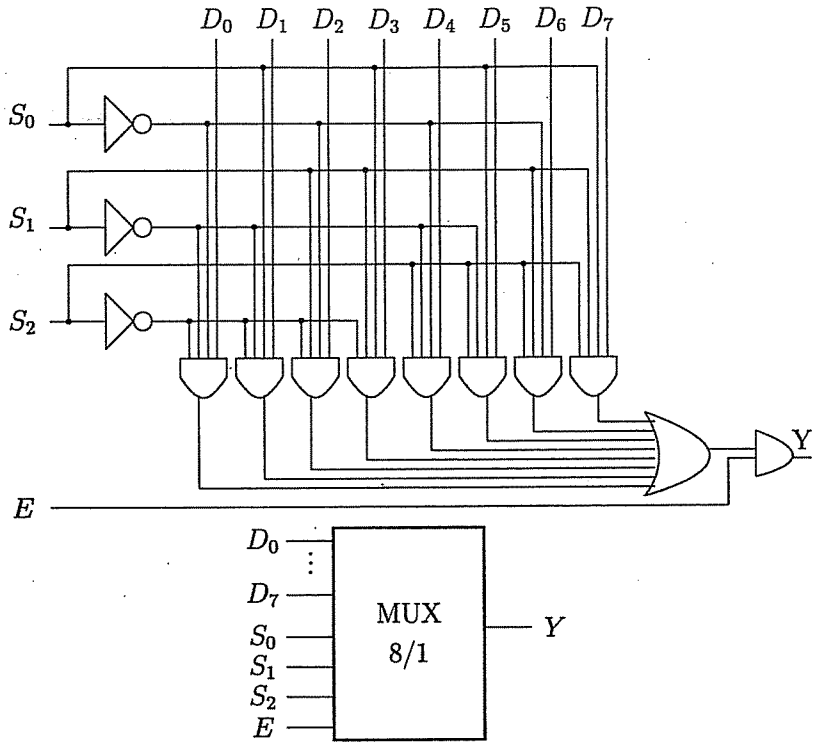
Na osnovu izraza $(*)$, lako se izvrši sinteza multipleksera (slika 7.9). Na slici je prikazan i simbol digitalnog multipleksera sa osam ulaza.

Kao integrisane komponente, multiplekseri se izrađuju sa dva, četiri, osam ili šesnaest ulaza. U slučaju četiri ulaza (četiri ulaza podataka) D_0 do D_3 imamo dva selekciona ulaza S_0 i S_1, a izlazni signal Y dat je jednačinom:

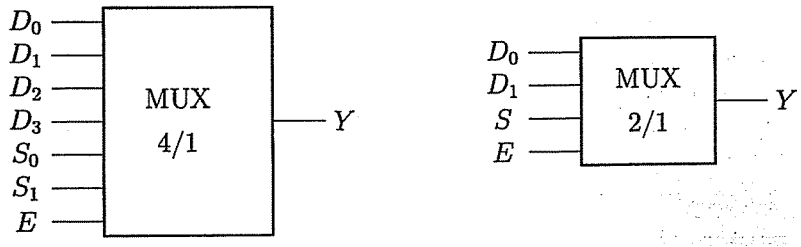
$$Y = (D_0 \overline{S_1}\,\overline{S_0} \vee D_1 \overline{S_1}\,S_0 \vee D_2\,S_1\,\overline{S_0} \vee D_3\,S_1\,S_0) \cdot E.$$

Slično, u slučaju dva ulaza odgovarajući izraz za izlaz Y je

$$Y = (D_0 \overline{S} \vee D_1\,S) \cdot E.$$



Slika 7.9: Logička šema i simbol osmoulaznog multipleksera.



Slika 7.10: Simboli četvoroulaznog i dvoulaznog multipleksera.

Za vježbu, sastavite tablicu za $Y = Y(D_0, D_1, S, E)$.

Simboli 4/1 i 2/1 multipleksera dati su na slici 7.10.

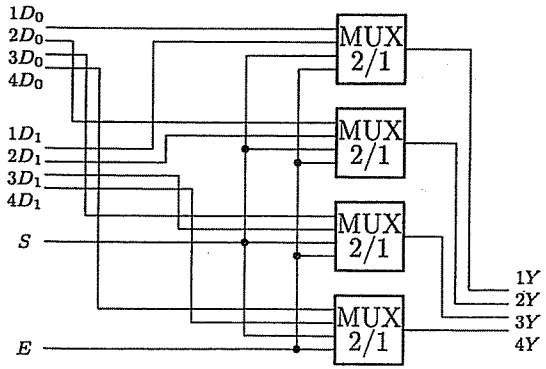
Multiplekseri se (radi univerzalnosti) obično izrađuju da sadrže i direktni izlaz Y i komplementirani izlaz $\overline{Y}$. Signal dozvole izlaza E istovremeno aktivira ili deaktivira oba izlaza.

U digitalnim sistemima, gdje se koriste višebitne informacije, po jedan multiplekser se koristi za svaki bit, dok su selekcioni ulazi i signal dozvole izlaza zajednički za sve multipleksere. Neka se vrši selekcija između n informacija D_0 do D_{n-1} i neka je veličina jedne informacije k bita. Tako, recimo, može biti $n = 2$ i $k = 4$, ili $n = 2$ i $k = 8$.

Razmotrimo detaljnije primjer takvog kola kada je $n = 2$ i $k = 4$. Kažemo da se radi o *četvorobitnom dvoulaznom multiplekseru*. Tada ulaze možemo označiti sa $1D_0$, $2D_0$, $3D_0$, $4D_0$, $1D_1$, $2D_1$, $3D_1$, $4D_1$.

Jasno je da n određuje broj binarnih komponenti S_0 do S_{m-1} signala selekcije SEL. Taj broj mora biti jednak $m = \log_2 n$. U našem slučaju, kako je $n = 2$, selekcija će imati jedan bit - S.

Princip rada četvorobitnog dvoulaznog multipleksera je sljedeći: ako je $S = 0$ na izlaze Y_0, Y_1, Y_2, Y_3 propuštaju se signali $1D_0$, $2D_0$, $3D_0$, $4D_0$, a kada je $S = 1$ - signali $1D_1$, $2D_1$, $3D_1$, $4D_1$. Logička šema data je na slici 7.11.



Slika 7.11: Multiplekser za dvije informacije od po četiri bita.

U literaturi i u katalozima integrisanih komponenti, multiplekser se ponekad naziva i selektorom podataka (engl. *Data Selector*). Selekcioni ulazi nazivaju se tada adresnim ulazima i označavaju sa A, B, C, ...

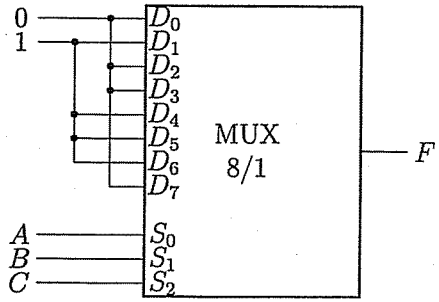
Multiplekser može da koristi i za realizaciju (za sintezu) logičkih funkcija. Pogledajmo na primjeru funkcije

$$F = \overline{C}\,\overline{B}\,A \vee C\,\overline{B} \vee C\,B\,\overline{A}.$$

Da bi iskoristili multiplekser za njenu realizaciju, potrebno je dopuniti funkciju do normalne forme:

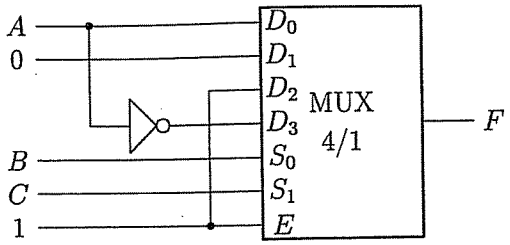
$$F = \overline{C}\,\overline{B}\,A \vee C\,\overline{B}\,\overline{A} \vee C\,\overline{B}\,A \vee C\,B\,\overline{A}.$$

Na slici 7.12 prikazana je realizacija ove funkcije pomoću 8/1 multipleksera. Ulazne veličine (A, B, C) spojene su na selektore multipleksera. U zavisnosti od njihovih vrijednosti biće propušteno 0 ili 1 na izlaz. Kako funkcija treba da ima vrijednost 1 za kombinacije ulaznih promjenljivih $(0, 0, 1)$, $(1, 0, 0)$, $(1, 0, 1)$, $(1, 1, 0)$, treba dovesti 1 na ulaze D_1, D_4, D_5 i D_6, a 0 na ostale.



Slika 7.12: Realizacija funkcije putem 8/1 multipleksera

Istu funkciju moguće je realizovati i pomoću 4/1 multipleksera (slika 7.13). U ovom slučaju na selekcione ulaze S_0 i S_1 dovode se redom B i C. Na ulaze D_0, D_1, D_2 i D_3 dovode se 0, 1, $\overline{A}$ ili A, u zavisnosti od izgleda SDNF funkcije F.



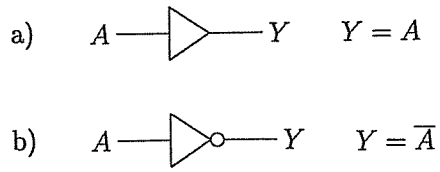
Slika 7.13: Realizacija funkcije putem 4/1 multipleksera

Na primjer, za ulaz D_0 razmatramo da li su u SDNF prisutni članovi $\overline{A}\,\overline{B}\,\overline{C}$ i $A\,\overline{B}\,\overline{C}$. Ako su oba prisutni, tada na ulaz D_0 dovodimo 1. Ako su oba odsutna, onda dovodimo 0. U slučaju da je prisutan samo jedan od ovih članova, tada dovodimo $\overline{A}$ ili A. Da bi upravo izložena analiza bila razumljiva, treba imati u vidu već ranije navedenu jednačinu multipleksera 4/1:

$$F = D_0\,\overline{S_1}\,\overline{S_0} \vee D_1\,\overline{S_1}\,S_0 \vee D_2\,S_1\,\overline{S_0} \vee D_3\,S_1\,S_0.$$

7.4 Trostatički baferi

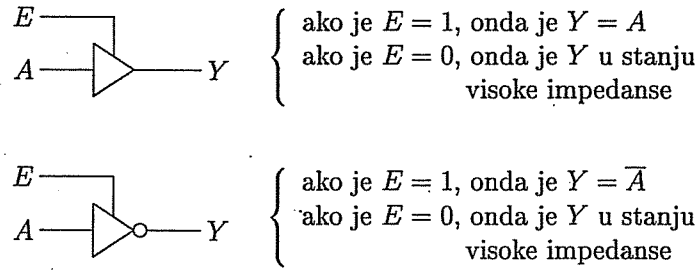
Logička kola *a*) i *b*) sa slike 7.14 poznata su odranije. Za kolo a) kaže se da je *neinvertujući pojačivač* ili *neinvertujući bafer* ili *bafer*. Za kolo b) kaže se da je *invertujući bafer* ili *invertor*.



Slika 7.14: Neinvertujući i invertujući bafer

Znamo da postoje razne tehnologije izrade logičkih kola, na primjer TTL i CMOS. Jedno logičko kolo odražava jednu logičku funkciju (Bulovu funkciju). Osim takvih kola, u mnogim tehnologijama izrađuju se i *modifikovana* ili *trostatička (logička) kola*. Izlaz trostatičkog kola nalazi se u jednom od tri moguća stanja (položaja). Izlazni signal može da bude logička nula, logička jedinica ili može da bude u stanju visoke impedanse (visokog otpora).

Trostatička kola mogu da budu logička I, ILI, NI, NILI, invertori i neinvertujući pojačivači. Ukoliko trostatički invertor odnosno neinvertujući pojačivač ima povećani izlazni faktor grananja u odnosu na standardna kola date familije, onda se takvo kolo naziva *trostatičkim baferom* ili *trostatičkim drajverom*.



Slika 7.15: Trostatički baferi

Na slici 7.15 prikazani su standardni simboli za trostatičke bafere. Kolo *c*) je neinvertujući trostatički bafer, a kolo *d*) invertujući trostatički bafer.

Signal koji prebacuje kolo u stanje visoke impedanse naziva se *signalom dozvole* ili *signalom aktiviranja*. Najčešće se obilježava slovom E, po engleskoj riječi *enable*, što znači dozvola. I invertujući i neinvertujući trostatički bafer aktivni su kada je signal dozvole E jednak jedinici, a za $E = 0$ oni su u stanju visoke impedanse. Bafer *c*) za $E = 1$ predstavlja neinvertujući pojačivač, a bafer *d*) za $E = 1$ predstavlja invertor.

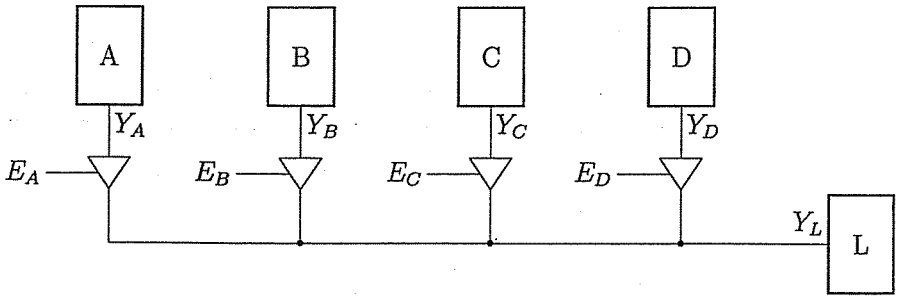
Trebalo bi nacrtati šemu sa tranzistorima za c) i d). Sa šeme se vidi da izlazu Y prethode gornji tranzistor T_1 (kome prethodi izvor napajanja) i donji tranzistor T_2 (kome prethodi uzemljenje). Neka bude $E = 0$. Tada i T_1 i T_2 postaju neprovodni ili zakočeni. Struja kroz jedan i drugi tranzistor je

izuzetno mala; recimo, ona ima red veličine $10^{-15}A$. Izlaz Y je razdvojen i od izvora za napajanje i od zemlje (od mase). Izlazna impedansa je reda veličine recimo $10^9\Omega$. Zato se kaže da je tada izlaz Y u stanju visoke impedanse ili u trećem stanju ili u neopterećenom stanju. Izlaz Y svakako se dovodi na ulaz nekog drugog (narednog) kola. Ako je $E = 0$ onda Y ne opterećuje naredno kolo. Kao da je linija Y odspojena ili otkačena od narednog kola. Matematičar bi kazao da je vrijednost Y nedefinisana kada je $E = 0$.

Prosto rečeno, vrijednost napona na izlaznoj liniji Y praktično je jednaka nula volti kad dozvola nije data. Znamo da nije obavezno da se u svakoj familiji logička nula poklapa sa nula volti, odnosno sa fizičkom nulom.

Neka se izlazi iz više digitalnih uređaja šalju posredstvom zajedničke linije prema jednom digitalnom uređaju, recimo prema digitalnom uređaju L, naravno ne više od jednog izlaza u isto vrijeme. Ovakvo slanje ostvaruje se po pravilu uz pomoć trostatičkih bafera.

Na slici 7.16 prikazan je digitalni sistem u kome se sa logičkih mreža A, B, C i D signali Y_A, Y_B, Y_C i Y_D šalju logičkoj mreži L, preko zajedničke linije. Signali dozvole E_A do E_D moraju biti generisani u različitim vremenskim intervalima, kako ne bi dolazilo do "sudaranja" na zajedničkoj izlaznoj liniji. Aktiviranje više od jednog trostatičkog bafera priključenog na istu liniju može da dovede do trajnog oštećenja bafera u slučaju da neki od aktiviranih bafera šalje jedinicu a neki drugi šalje nulu.



Slika 7.16: Prosleđivanje signala zajedničkom linijom

Svaka izlazna mreža Y_A, Y_B, Y_C i Y_D ima svoj upravljački signal E_A, E_B, E_C i E_D. Upravljački signal određuje da li je njena izlazna impedansa

niska ili visoka. Uvijek samo jedna od izlaznih mreža, spojenih na zajedničku liniju, ima nisku izlaznu impedansu (nije zatvorena). Zadatak je onoga koji planira mrežu da bude osigurano da se izvrši izlaz samo iz jednog člana. U protivnom se mogu uništiti izlazni krugovi tropoložajnog člana velikim strujama u trenutku podijeljenog logičkog nivoa na njihovim izlazima.

Dakle, važi:

$$\begin{cases} \text{ako je } E_A = 1,\ E_B = 0,\ E_C = 0,\ E_D = 0 \text{ onda je } Y_L = Y_A; \\ \text{ako je } E_A = 0,\ E_B = 1,\ E_C = 0,\ E_D = 0 \text{ onda je } Y_L = Y_B; \\ \text{ako je } E_A = 0,\ E_B = 0,\ E_C = 1,\ E_D = 0 \text{ onda je } Y_L = Y_C; \\ \text{ako je } E_A = 0,\ E_B = 0,\ E_C = 0,\ E_D = 1 \text{ onda je } Y_L = Y_D. \end{cases}$$

Za mrežu oblika slike 7.16 kaže se da ostvaruje funkciju žičano ILI (engl. wired OR). U prethodnom naslovu smo vidjeli da multiplekser takođe može da posluži kada treba da se vrši izbor jednog izlaznog signala (među više izlaznih signala) koji će biti poslat. Slične okolnosti važe kada umjesto jedne prijemne mreže (kakva je mreža L na slici 7.16) postoji nekoliko prijemnih mreža i treba da upućeni signal dospije samo do jedne, izabrane, prijemne mreže.

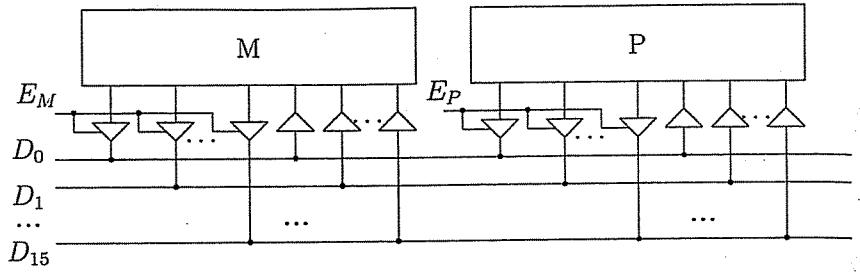
7.5 Magistrale digitalnih signala

Magistrala (*sabirnica*, engl. *bus*, *high-way*) je prenosni put ili spojni put između dvije mreže ili između dva podsistema u digitalnom sistemu (od jednog podsistema ka drugom podsistemu) ili između dva registra. Magistrala se sastoji od određenog broja paralelnih vodova ili linija. Broj tih linija naziva se *širinom magistrale*. Recimo, širina magistrale može da bude četiri ili osam linija. I jednu samu liniju možemo smatrati magistralom.

U računarskom sistemu ima mnogo podsistema i nije ekonomično da od bilo kog podsistema ka bilo kom drugom podsistemu postoji posebna veza. Zato se magistrala definiše kao prenosni put za komunikaciju između nekoliko podsistema.

Mikroprocesorski sistem može da ima adresnu magistralu od recimo 20 linija, magistralu podataka od 16 linija i upravljačku magistralu od 8 linija. Svaka magistrala prenosi specifičnu vrstu informacija. Sve tri magistrale zajedno u tom slučaju predstavljaju sistemsku magistralu. Na sistemsku magistralu priključeni su podsistemi koji međusobno razmjenjuju informacije. Na

primjer, procesor P treba da šalje podatke u memoriju M i da prima podatke iz memorije, odnosno P upisuje podatke u M i P čita podatke iz M. Te dvije operacije se u računarskom sistemu u principu ne obavljaju u isto vrijeme. Zato ne bi bilo ekonomično da se koriste dva odvojena prenosna puta.



Slika 7.17: Priključivanje digitalnih podsistema na magistralu

Trostatički baferi omogućavaju realizaciju dvosmjerne magistrale. Slika 7.17 prikazuje priključivanje digitalnih podsistema na magistralu. Upravljački signali E_M i E_P definišu da li će se jedna riječ (veličine 16 bita) prenijeti iz memorije u procesor ili će pak ona biti prenesena u suprotnom smjeru (iz procesora u memoriju). Prenos se ostvaruje preko magistrale podataka čija je širina jednaka šesnaest bita, tj. šesnaest linija- linije od D_0 do D_{15}.

Ako je $E_M = 0$ i $E_P = 0$ onda se ne vrši nikakav prenos. Ako je $E_M = 1$ i $E_P = 0$ onda M šalje jednu riječ u P. Ako je $E_M = 0$ i $E_P = 1$ onda P šalje jednu riječ u M. Ako je $E_M = 1$ i $E_P = 1$ onda rad mreže nije definisan i ona može da pregori.

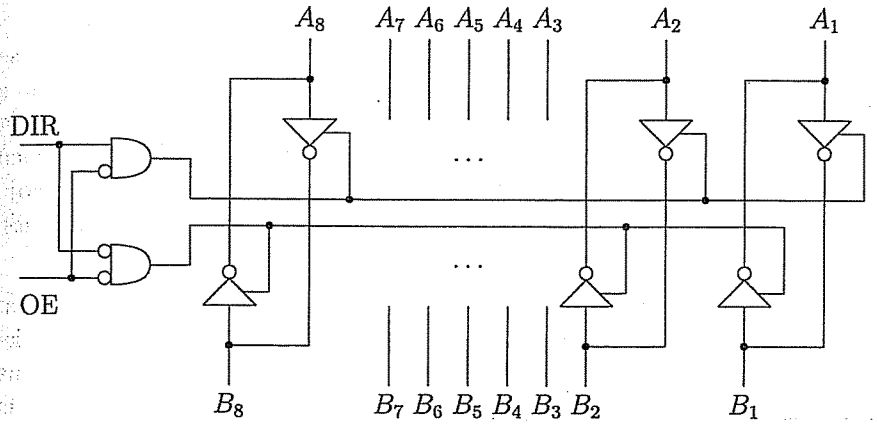
Dvosmjerna magistrala podataka upotrebljava se za prenos podataka između mikroprocesora P, radne memorije M i spoljašnjih uređaja (ulaznih i izlaznih uređaja).

Trostatički baferi vezani tako da propuštaju logičke signale u oba smjera nazivaju se *bidirekcionim trostatičkim baferima* ili *trostatičkim baferima koji su primopredajnici* (engl. *three state transciever*). Transciever je kovanica od *transmitter* i *receiver* - predajnik i prijemnik.

Trostatički baferi se proizvode kao integrisane komponente u više verzija. Mogu biti pakovani po 4–6 invertujućih ili neinvertujućih bafera sa odvojenim signalima dozvole, po 8 bafera sa zajedničkim signalom dozvole ili po 8 bidirekcionih bafera sa zajedničkim signalom dozvole.

Magistrale računarskih sistema najčešće sadrže osam linija ili multiple po osam linija, tako da su integrisani trostatički baferi najčešće pakovani sa osam bidirekcionih bafera u čipu. Baferi mogu da budu invertujući ili neinvertujući, a izlazni faktor grananja je najčešće između 25 i 50.

Na slici 7.18 prikazana je logička šema integrisanog kola 74 HC 640. Skraćenica DIR označava $DIRECTION$ (smjer) a OE označava $OUTPUT\ ENABLE$ (dozvola izlaza). Vidi se da je signal za dozvolu OE postavljen "obrnuto" od uobičajenog načina: dozvola je data ako i samo ako je $OE = 0$. Jasno, možemo smatrati da na slici izlazna linija iz bafera uopšte nije ni nacrtana kada baferu nije dat njegov signal dozvole.



Slika 7.18: Logička šema komponente 74 HC 640.

Mreža sadrži osam bidirekcionih bafera i logička kola za generisanje zajedničkih signala dozvole. Signal OE, kada je aktivan tj. kada je na niskom logičkom nivou, aktivira trostatičke bafere u smjeru od A prema B u slučaju da je signal DIR na visokom logičkom nivou, a u smjeru od B prema A kada je DIR = 0. Drugim riječima, ako je $OE = 1$, onda se ništa ne prenosi. Ako

je $OE = 0$ i $DIR = 0$, onda

$$(A_8, \ldots, A_1) \leftarrow (\overline{B_8}, \ldots, \overline{B_1}).$$

Konačno, ako je $OE = 0$ i $DIR = 1$, onda

$$(B_8, \ldots, B_1) \leftarrow (\overline{A_8}, \ldots, \overline{A_1}).$$

Pošto se ovakvi baferi najčešće koriste za spajanje dvije magistrale, dobili su naziv *bidirekcioni baferi magistrale* odnosno *primopredajnici magistrale*.

7.6 Pojam bistabilnog kola

U ovom naslovu ćemo izložiti princip rada memorijskog (*bistabilnog*) elementa i nabrojati neke vrste takvih elemenata.

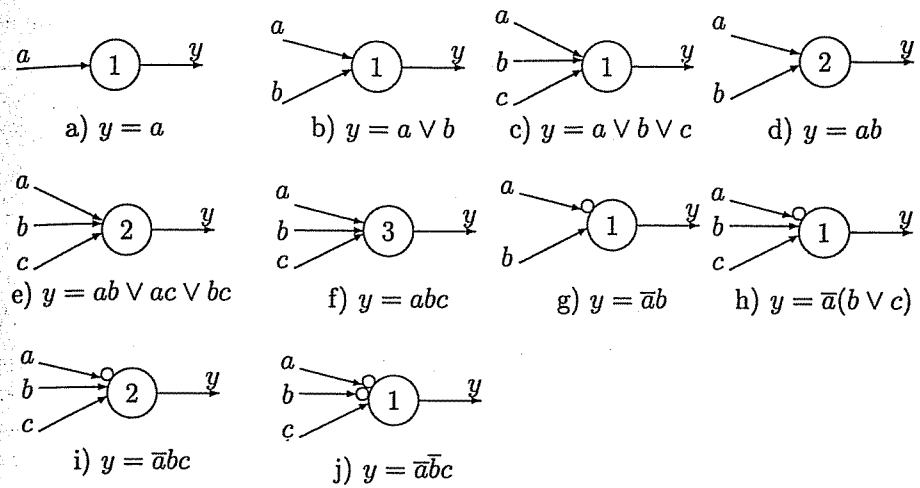
Da bismo izložili princip čuvanja podataka u računaru, poslužićemo se konceptom *E-elementa* ili *elementa* koji je još davno dao Fon Nojman u svom izvještaju za računar Edvac. Definišimo pojam elementa. Element ima više ulaza i jedan izlaz Y. Grafički ga predstavljamo kružićem. Ka kružiću vode strelice koje predstavljaju ulaze a, b, c, itd, a od kružića strelica koja predstavlja izlaz. Dalje, za element se definiše njegov prag (engl. *treshold*). To je broj t, koji može biti jednak $1, 2, 3$, i koji se upisuje u kružić.

Potrebno je još definisati na koji način izlaz zavisi od ulaza i od praga. Postoje dvije vrste ulaza u elemente. Ulazi prve vrste vode direktno ka kružiću i oni doprinose da postane $y = 1$. Ako manje od t tih ulaza ima vrijednost jedan, tada će biti $y = 0$. Ulazi druge vrste crtaju se tako da imaju mali krug na ulazu u element. Za njih kažemo da čine *prepreke* (da y postane 1).

Ako bar jedna prepreka ima vrijednost jedan, onda je $y = 0$, bez obzira na vrijednosti ostalih ulaza. Prepreka čija je vrijednost nula ne utiče na rad elementa.

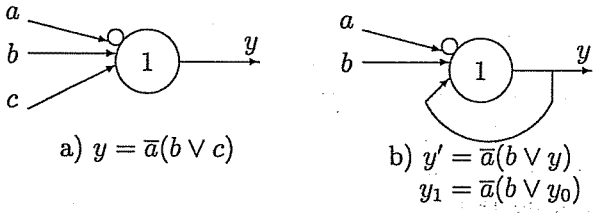
Na slici 7.19 dati su primjeri elemenata kao i njihove jednačine.

Razmotrimo sada elemente na slici 7.20. Element *a)* sa ove slike ima prag $t = 1$. On ima jednu prepreku a i dva ulaza b i c. Napravimo sada sljedeću izmjenu. Sa izlaza y dovedimo sada signal na ulaz c (slika 7.20.b)). Kažemo da smo napravili *petlju* ili da koristimo *princip povratne sprege*.



Slika 7.19: Elementi i njihove jednačine

Naponski nivo u petlji se brzo ustali na vrijednost $y = 0$ ili $y = 1$, zavisno od okolnosti, tj. zavisno od ulaza i prethodne vrijednosti izlaza. Napomenimo da postoje slučajevi povratne sprege gdje stabilna vrijednost izlaza neće biti obrazovana.



Slika 7.20: Konstruisanje memorijskog elementa

Analizirajmo kolo sa slike 7.20.b). Sa y označavamo staru vrijednost na izlazu, a sa y' novu vrijednost na izlazu. Zadavanjem $a = 1$ postiže se da postane $y' = 0$. Zadavanjem $b = 1$ postiže se da postane $y' = 1$, naravno, u

slučaju kada nema signala a, tj. kada je $a = 0$. Najzad, ako nema ni jednog ni drugog signala ($a = b = 0$), onda važi $y' = y$.

Nova vrijednost biće obrazovana poslije vremena potrebnog da se u petlji dostigne novi ravnotežni položaj. Potrebno je vrijeme od jednog takta. Često se umjesto y' i y piše y_1 i y_0, ili y_{n+1} i y_n. Vidimo da je $y' = y$ kada je $a = b = 0$. Ovo znači da kolo *pamti* kada nema pobude spolja. Dakle, konstrusali smo memorijski element. Osim pamćenja, ovo kolo ima i svojstvo da se lako postavlja da pamti jedinicu ili nulu. Tabela 7.3 predstavlja tabelu za bistabilno kolo realizovano pomoću $E-$elementa.

a	b	y_{n+1}
0	0	y_n
0	1	1
1	0	0
1	1	0

Tabela 7.3: Tabela za bistabilno kolo realizovano pomoću E-elementa

U savremenim računarima koriste se četiri vrste bistabilnih elemenata, i to SR kolo, JK kolo, D kolo i T kolo.

7.7 SR leč kolo

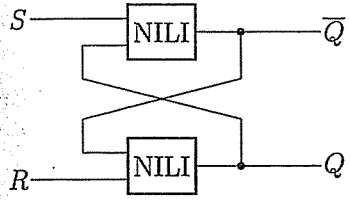
Izvršimo analizu mreže prikazane na slici 7.21 a). To radimo tako što pratimo logičke vrijednosti (nizak napon za nulu i visok napon za jedinicu) po njenim ulaznim i izlaznim linijama i u njenim karakterističnim tačkama, odnosno pratimo kako se te vrijednosti mijenjaju po vremenskoj osi. Vidimo da je u mreži prisutna povratna sprega. Kada kolo ima povratnu spregu, uopšte uzevši, može se desiti da se stabilne naponske vrijednosti uopšte ne uspostave, ili da se one uspostave na "slučajan" način, koji zavisi od okolnosti koje se ne mogu predvidjeti - to su tzv. nepravilne situacije.

Vidimo da kolo koristi NILI kolo, dakle kolo sa jednačinom $y = \overline{a \vee b}$. Tokom pravilnog rada kola, dvije izlazne linije uvijek saopštavaju dvije međusobno komplementarne vrijednosti. Zato su te dvije izlazne linije i označene kao Q i $\overline{Q}$.

Bistabilno kolo na slici 7.21 a). naziva se *SR leč kolom*. Slobodni ulazi logičkih kola označeni su sa S i R, S znači *set* – postavi – upiši jedinicu, a R znači *reset* – vrati se – upiši nulu. Izlazi su označeni sa Q i $\overline{Q}$; oni moraju da budu komplementarni. Kada su izlazni nivoi $Q = 1$ i $\overline{Q} = 0$ kaže se da je leč kolo *setovano*, dok se za slučaj kada je $Q = 0$ i $\overline{Q} = 1$ kaže da je leč kolo *resetovano*.

Analiza rada kola sastoji se od četiri slučaja.

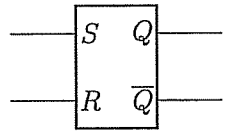
Prvi slučaj je $S = 0$ i $R = 0$, odnosno kada se po jednoj i drugoj ulaznoj liniji dovodi logička nula. Tada na ulaz gornjeg NILI–kola pristižu $S = 0$ i Q, tj. Q zatečeno, tako da će izlaz iz tog NILI–kola biti jednak $y = \overline{0 \vee Q} = \overline{Q}$. Na ulaz donjeg NILI–kola pristižu $R = 0$ i $\overline{Q}$ zatečeno, pa je njegov izlaz jednak $y = \overline{0 \vee \overline{Q}} = Q$. Zaključujemo da, nezavisno od toga da li je Q zatečeno jednako nuli ili jedinici, ništa se neće promijeniti na izlazima Q i $\overline{Q}$. Dakle, kada je $S = 0$ i $R = 0$, kolo *pamti* jedan bit. Ono po liniji Q saopštava vrijednost koju pamti, a po liniji $\overline{Q}$ saopštava još i komplementarnu vrijednost.



a) Šema SR leč kola

S	R	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	Q_n	$\overline{Q_n}$
0	1	0	1
1	0	1	0
1	1	0	0

b) Tabela



c) Simbol

Slika 7.21: SR leč kolo

Slično, znajući definiciju NILI kola, vidi se da se dovođenjem kombinacije $S = 1$, $R = 0$ na ulaze kola dešava da se izlazi postave u stanje $Q = 1$, $\overline{Q} = 0$; kolo je sada setovano. Dovođenjem kombinacije $S = 0$, $R = 1$ izlazi će se postaviti u novo stanje $Q = 0$, $\overline{Q} = 1$, odnosno kolo će sada biti resetovano. Iz ova dva slučaja vidimo da se postavljanje kola u željeno stanje vrši dovođenjem logičke jedinice na odgovarajući ulaz.

Konačno, pogledajmo slučaj $S = R = 1$. Ako se na ulazu pojavi $S = 1$, $R = 1$ onda će se oba izlaza dovesti u stanje logičke nule i (znači) izlazi će

prestati da budu komplementarni. U okviru ovog, poslije prelaska pobude iz stanja $S = 1$, $R = 1$ u stanje $S = 0$, $R = 0$ ne može da bude predviđeno stanje na izlazima; stanje na izlazima zavisi od toga koji se ulazni signal (S ili R) prvi promijenio. Zato se kombinacija $S = R = 1$ naziva *zabranjenim stanjem* ili *nedozvoljenim stanjem na ulazu*. Ako se SR leč kolo pojavljuje kao dio u nekoj većoj mreži onda je zadatak onoga koji projektuje mrežu da osigura da na ulazima kola nikad neće biti zabranjeno stanje.

Na osnovu dosadašnjeg razmatranja rada SR leč kola, lako formiramo njegovu *funkcionalnu tabelu*, datu na slici 7.21 b). Tabela daje stanja na izlazima za sve moguće kombinacije stanja na ulazima. Funkcionalna tabela naziva se često i *karakterističnom tabelom* kola. U tabeli, Q_n označava trenutno stanje izlaza Q, dok Q_{n+1} označava naredno stanje izlaza Q, odnosno stanje izlaza Q poslije promjene ulaznih signala. Na slici 7.21 c) prikazan je grafički simbol za SR leč kolo.

Prilikom sinteze složenih sekvencijalnih mreža, osim funkcionalne tabele, često se koristi i *eksitaciona tabela* ili *tabela pobude*. Eksitaciona tabela može se izvesti iz funkcionalne tabele, a definiše vrijednosti ulaznih signala pomoću kojih se kolo prevodi u željeno stanje. Tako eksitaciona tabela SR leč kola govori sljedeće: ako se kolo želi prebaciti iz stanja $Q_n = \ldots$ u stanje $Q_{n+1} = \ldots$ onda treba zadati $S = \ldots$ i $R = \ldots$.

Rad leč kola može da bude opisan i pomoću njegove tzv. *logičke jednačine* (njegove *formule*, *funkcionalne jednačine* ili *karakteristične jednačine*). Logička jednačina neposredno se dobija iz funkcionalne tabele. Ona izražava novo stanje izlaza Q_{n+1} kao funkciju od dosadašnjeg stanja izlaza Q_n i dva ulaza S i R; $Q_{n+1} = Q_{n+1}(Q_n, S, R)$.

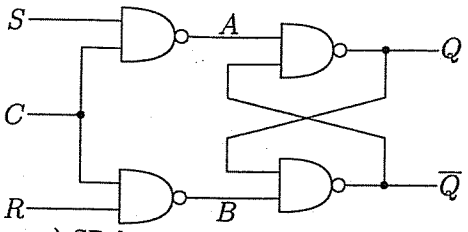
Za SR leč kolo logička jednačina glasi

$$Q_{n+1} = S\overline{R} \vee \overline{S}\,\overline{R}Q_n.$$

Ako se koristi činjenica da za dozvoljena stanja na izlazima nije istovremeno $S = 1$ i $R = 1$ onda se ova logička jednačina svodi na jednostavniji oblik. Na primjer, korišćenjem zakona apsorpcije, jednačina se svodi na

$$Q_{n+1} = S \vee \overline{R}Q_n.$$

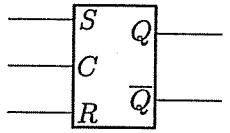
Rad SR leč kola može da bude opisan i pomoću njegovih tzv. vremenskih dijagrama. U katalozima integrisanih kola, rad kola često se opisuje na takav



a) SR leč sa signalom dozvole

S	R	C	Q_{n+1}	$\overline{Q_{n+1}}$	
0	0	1	Q_n	$\overline{Q_n}$	$A = 1,\ B = 1$
0	1	1	0	1	$A = 1,\ B = 0$
1	0	1	1	0	$A = 0,\ B = 1$
1	1	1	1	1	$A = 0,\ B = 0$
x	x	0	Q_n	$\overline{Q_n}$	$A = 1,\ B = 1$

b) Tabela



c) Simbol

Slika 7.22: SR leč kolo sa dozvolom

način. Vremenski dijagrami izražavaju $Q = Q(t)$ i $\overline{Q} = \overline{Q}(t)$ u zavisnosti od $S = S(t)$ i $R = R(t)$; t – vrijeme. Drugim riječima, na vremenskim dijagramima je prikazano kako se tokom vremena mijenjaju četiri funkcije, gdje su očito prve dvije $Q = Q(t)$ i $\overline{Q} = \overline{Q}(t)$ uslovljene sa druge dvije $S = S(t)$ i $R = R(t)$. Obuhvaćeni su razni slučajevi, kada te funkcije tokom vremena mijenjaju vrijednosti.

Na slici 7.21.a) prikazan je jedan mogući način da se sastavi SR leč kolo. Postoje i drugi načini (druge šeme) koji takođe realizuju SR leč kolo. U tim šemama koriste se logički elementi ILI, NILI, I, NI i NE. Tada ostaju da važe dosadašnja funkcionalna tabela, eksitaciona tabela, jednačina i vremenski dijagrami, a takođe naravno i grafički simbol kola.

Dosad razmatrano SR leč kolo reaguje na promjenu ulaznih signala u bilo kom trenutku. Međutim, često se ukazuje potreba za leč kolom koje može da mijenja svoje stanje samo u određenim vremenskim intervalima, samo u vremenskim intervalima kada je aktivan taktni signal C. U literaturi se za C sreću i oznake CLK, EN, $ENABLE$ i G. Takvo leč kolo naziva se *SR leč kolom sa signalom dozvole*. Prikazano je na slici 7.22.a).

Vidi se da je realizovano pomoću NI kola. Funkcionalna tabela ovog leč kola prikazana je na slici 7.22.b). Na osnovu S, R i C obrazuju se međuvrijednosti A i B. A i B su ulazi u petlju povratne sprege u kojoj učestvuju još i Q i $\overline{Q}$. Dakle, na osnovu A, B, Q staro i $\overline{Q}$ staro formiraju se Q novo i $\overline{Q}$ novo. Za razne slučajeve koji mogu da nastupe, možemo proučiti kako će se formirati naponski nivoi u raznim čvorovima koji su obuhvaćeni petljom (da li će se stabilni naponski nivoi formirati), čime dobijamo tabelu. Slika 7.22. c) prikazuje grafički simbol za SR leč kolo sa signalom dozvole.

Logička jednačina SR leč kola sa dozvolom glasi

$$Q_{n+1} = CS \vee \overline{C}RQ_n.$$

Ako je kontrolni signal C periodični signal takta onda se dobija *taktovano SR kolo* ili *sinhrono SR kolo*.

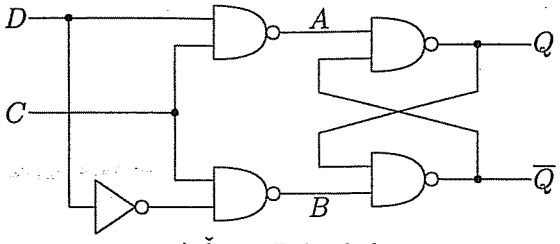
7.8 D leč kolo

Vidjeli smo da SR leč kolo ima razdvojene ulaze za setovanje odnosno za resetovanje. Razdvojeni ulazi pogodni su za primjene u kontrolnim sistemima. Međutim, za primjene u sistemima za pamćenje informacija pogodnije je imati samo jedan ulaz u leč kolo. Taj ulaz će određivati stanje na izlazu. Kolo, po pravilu, ima još i kontrolni ulaz. Takvu funkciju obavlja D leč kolo.

Šema D leč kola prikazana je na slici 7.23 a). Sa slike se vidi da osnovu za šemu D leč kola čini SR leč kolo sa dozvolom. Jedina razlika je dodatni invertor na ulazu koji uklanja mogućnost da na ulaz bude dovedena nedozvoljena kombinacija signala.

Razmotrimo šemu i objasnimo funkcionisanje kola. Neka je $C = 1$. Kada je na ulazu $D = 1$ tada je $A = 0$ i $B = 1$ pa se kolo setuje ($Q \leftarrow 1$). Suprotno tome, kada je na ulazu $D = 0$, tada je $A = 1$ i $B = 0$, pa će se kolo resetovati.

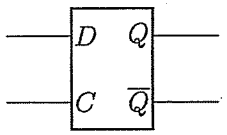
Dakle, na izlazu se uvijek pojavljuje isti signal koji je i na ulazu, naravno poslije vremena kašnjenja kroz logičke elemente i uspostavljanja ravnotežnog stanja u petlji povratne sprege. Kada se kontrolni signal C vrati na stanje logičke nule onda stanje na izlazu biva zamrznuto.



a) Šema D leč kola

D	C	Q_{n+1}	$\overline{Q_{n+1}}$
0	1	0	1
1	1	1	0
x	0	Q_n	$\overline{Q_n}$

b) Tabela



c) Simbol

Slika 7.23: D leč kolo

U našoj konfiguraciji D leč kola, ulazni signal dozvole C je aktivan kada je na visokom logičkom nivou (kada je jednak jedan).

Na slici 7.23.b) prikazana je funkcionalna tabela D leč kola. U tabeli se na jednom mjestu pojavljuje znak x. To znači da vrijednost ulaza D u tom redu tabele ne utiče na rad kola. Svejedno je da li je ta vrijednost nula ili jedan; isto će se desiti na izlazima kola.

Na slici 7.23.c) prikazan je grafički simbol D leč kola.

Osnovni vremenski parametri D leč kola su t_{su} – setup time, vrijeme postavljanja i t_h – hold time, vrijeme držanja.

Za pravilan rad kola, signal na ulazu D treba da bude stabilan tokom najmanje t_{su} ns prije opadanja signala dozvole sa $C = 1$ na $C = 0$. Isto tako, zahtijeva se da signal na ulazu D bude stabilan za vrijeme najmanje t_h poslije opadanja signala dozvole sa $C = 1$ na $C = 0$. Vrijeme t_{su} se posmatra zato što je potrebno određeno vrijeme da struja prođe kroz logičke elemente, tj. da se uspostave vrijednosti na izlazima kola. Vrijeme t_h je važno zato što signal D ne dolazi u istom trenutku na element čiji je izlaz A i na element čiji je izlaz B, jer je prisutan invertor, koji ima svoje vrijeme kašnjenja. Ako bi se signal D mijenjao još dok je $C = 1$ onda bi moglo da se desi da postane

i $A = 0$ i $B = 0$, a to je nedozvoljena kombinacija.

7.9 Sinhroni flipflopovi SR i D

Često se koriste bistabilni elementi kod kojih se promjena stanja može vršiti samo u tačno određenim trenucima vremena koji su određeni taktom sistema. Takvi bistabilni elementi nazivaju se *flipflopovima.* Promjena stanja može se vršiti prilikom promjene logičke vrijednosti ulaza na koji se dovodi takt.

U praksi se sreću dva načina okidanja flipflopa: *impulsni način*, gdje se okidanje vrši cijelim jednim takt impulsom, i *ivični način*, gdje se okidanje vrši sinhrono sa ivicom takta (rastućom ili opadajućom).



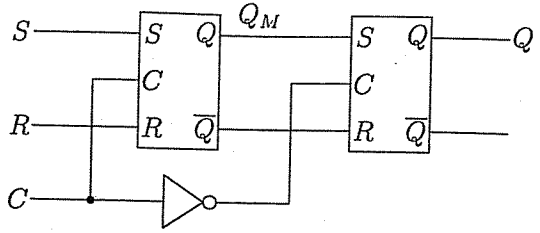
Slika 7.24: Vremenski dijagrami signala takta C

U prvom slučaju se radi o *flipflopu sa impulsnim okidanjem* ili *MS flipflopu* (engl. master–slave). Realizacija ove ideje može se izvršiti korišćenjem dva leč kola (slika 7.25.a)). U prvo leč kolo (*master kolo*) informacija sa ulaza upisuje se poslije uzlazne ivice takta, a u drugo leč kolo (*slave kolo*) informacija sa izlaza prvog leč kola upisuje se poslije silazne ivice takta.

Na osnovu poznavanja rada SR leč kola i opisanog načina rada MS SR flipflopa sa slike 7.25.a), lako se izvodi njegova funkcionalna tabela – slika 7.25.b). Na slici 7.25.c) prikazan je odgovarajući grafički simbol.

Mogu se prikazati i vremenski dijagrami razmatranog flipflopa. Pobudni signal (S ili R) treba da postane aktivan odmah poslije uzlazne ivice takta C. Ne traži se da on ostane aktivan sve do silazne ivice, već samo dok se uspostave izlazi master kola, jer onda ti izlazi pobuđuju slave kolo. Ukupno, tokom jedne periode takta može jednom da dođe do promjene stanja izlaznog signala flipflopa (do promjene vrijednosti koju flipflop pamti).

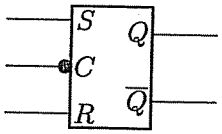
Za pouzdan rad kola treba dobro usaglasiti ulazne signale sa oblikom signala dozvole i sa vremenima kašnjenja logičkih elemenata kola. Za pojedini ulazni signal propisuje se kada će stabilan nivo signala biti dostignut i koliko će onda takav nivo trajati.



a) Šema SR flipflopa sa impulsnim okidanjem

S	R	C	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	⊓	Q_n	$\overline{Q_n}$
0	1	⊓	0	1
1	0	⊓	1	0
1	1	⊓	?	?

b) Tabela



c) Simbol

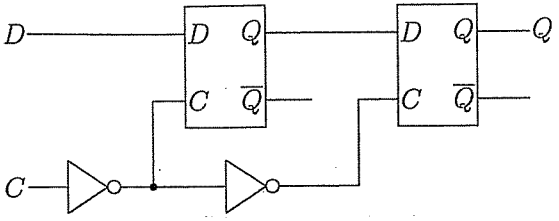
Slika 7.25: SR flipflop sa impulsnim okidanjem

Na slici 7.26.a) prikazana je šema D flipflopa sa ivičnim okidanjem, sa okidanjem (prebacivanjem) na rastućoj ivici takta. Slično se definiše i D flipflop koji se okida opadajućom ivicom takta.

Vidimo da u sastavu flipflopa učestvuju dva leč kola. Za prvo D leč kolo kaže se da je master kolo a za drugo D leč kolo kaže se da je izvršno kolo. Kada je signal takta na niskom nivou onda je moguće upisivati informaciju u master kolo. Dovođenjem rastuće ivice takta, signal dozvole master kola ide na nivo nule i tako se stanje master kola zamrzava. Istovremeno, ako se zanemari kašnjenje kroz invertor, signal dozvole izvršnog kola ide na nivo jedinice. Drugim riječima, sada se aktivira drugo D leč kolo (izvršno). Sada se stanje prvog D leč kola upisuje u drugo D leč kolo. Time se završava proces upisivanja u flipflop.

Na slici 7.26.b) prikazana je odgovarajuća funkcionalna tabela, a na slici 7.26.c) odgovarajući grafički simbol. Uočimo trougao na CLK ulazu; taj ulaz često se označava i kao C ili CP. Taj trougao govori da je okidanje ivično ili dinamičko. Ivično okidanje razlikuje se od impulsnog okidanja po tome što u

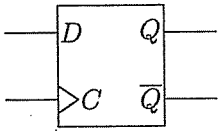
slučaju ivičnig okidanja pobuda D nastupa u "okolini" ivice takta (uzlazne ivice takta); pobuda počinje malo prije ivice a traje do malo poslije nje.



a) D flipflop sa ivičnim okidanjem

D	C	Q_{n+1}	$\overline{Q_{n+1}}$
0	⌐¬	0	1
1	⌐¬	1	0

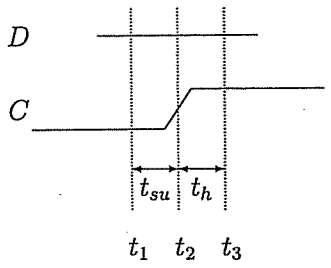
b) Tabela



c) Simbol

Slika 7.26: D flipflop sa ivičnim okidanjem

Za pravilan rad ovoga kola propisuju se parametri t_{su} i t_h (slika 7.27). U trenutku $t = t_1$ ulaz D dostigao je stabilan nivo i neće se mijenjati do trenutka $t = t_3$. Trenutak $t = t_2$ odnosi se na rastuću ivicu takta (ulaz C). Ovdje je $t_1 < t_2 < t_3$. Treba da bude $t_2 - t_1 > t_{su}$ i $t_3 - t_2 > t_h$. Na vremenskim dijagramima nacrtani su jedan iznad drugog grafici dviju funkcija: $D = D(t)$ i $C = C(t)$.



Slika 7.27: Vremenski dijagrami D flipflopa sa ivičnim okidanjem

Sinhroni flipflopovi mogu dodatno imati i asinhrone odnosno direktne priključke oblika S i R, koji služe za direktno setovanje i resetovanje. Dovođenjem signala na asinhrone ulaze, mijenja se stanje flipflopa, nezavisno od stanja na C i D ulazima. Znači, ti signali će djelovati na stanje flipflopa nezavisno od toga u kom trenutku oni nastupaju (nezavisno od stanja ostalih ulaznih signala). Ulaz za asinhrono setovanje obično se obilježava kao S_D ili PR (engl. preset). Ulaz za asinhrono resetovanje obično se obilježava kao R_D ili CLR (engl. clear), D je od direct. Asinhroni ulazi obično se koriste za jednostavne operacije. Na primjer, koriste se za postavljanje flipflopa u određeno stanje prilikom inicijalizacije sistema.

7.10 Flipflopovi JK i T

Kada su S i R ulazi SR leč kola istovremeno aktivni, onda je stanje na izlazu neodređeno. Ovaj problem rješava se uvođenjem novog tipa bistabilnog kola, uvođenjem JK flipflopa.

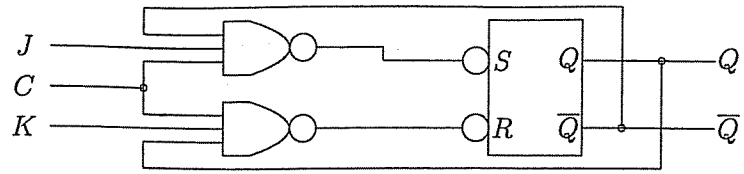
Postoje brojne realizacije JK flipflopova u TTL i CMOS tehnologijama. Postoje razna kola za koja se može reći da predstavljaju JK flipflop. Jedno takvo kolo predstavljeno je na slici 7.28.a).

Vidimo da je kolo formirano korišćenjem SR leč kola i dva NI kola. Preko NI kola dovode se nove povratne sprege sa izlaza na ulaze. Sa slike možemo zaključiti da se promjena stanja može vršiti samo kada je takt C na visokom nivou. Takođe, vidimo da ulaz J služi za setovanje flipflopa a K za njegovo resetovanje. Posebno je interesantno šta se dešava kada se na ulaze dovede $J = K = 1$. U tom slučaju, zbog dejstva povratnih sprega, flipflop mijenja stanje - stanje se komplementira. Na primjer, ako je $Q_n = 1$, $\overline{Q}_n = 0$, onda NI kolo koje se dovodi na ulaz K daje na izlazu logičku nulu. Zato se leč kolo resetuje, odnosno flipflop dolazi u stanje $Q_{n+1} = 0$, $\overline{Q}_{n+1} = 1$. Sličan je slučaj ako je bilo $Q_n = 0$, $\overline{Q}_n = 1$. Za ostale slučajeve lako je utvrditi stanje izlaza. Tako dobijamo funkcionalnu tabelu prikazanu na slici 7.28.b). Slika 7.28.c) sadrži grafički simbol JK flipflopa.

Karakteristična jednačina JK flipflopa glasi

$$Q_{n+1} = J\overline{Q_n} \vee \overline{K}Q_n.$$

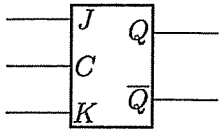
Posmatrajmo rad kola sa slike 7.28.a) u slučaju kada je $C = 1$ i $J = K = 1$. Tada se komplementira izlaz Q a time i $\overline{Q}$. Izlazi se dovode na ulaze. Ako



a) JK flipflop realizovan NI kolima

J	K	C	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	1	Q_n	$\overline{Q_n}$
0	1	1	0	1
1	0	1	1	0
1	1	1	$\overline{Q_n}$	Q_n
x	x	0	Q_n	$\overline{Q_n}$

b) Tabela



c) Simbol

Slika 7.28: JK flipflop

je takt impuls još uvijek aktivan , tj. ako je još uvijek $C = 1$, onda će se izlazi još jednom komplementirati, zato što su ulazi u stanju $J = K = 1$. Ovo, naravno, nije poželjno i postavlja se pitanje kako da se prevaziđe ovaj nedostatak kola. Kolo će raditi ispravno samo ako je takt impuls vrlo kratak. Bolje rečeno, takt treba da bude kraći od vremena kašnjenja kroz dva NI kola i SR leč kolo. Vrijeme kašnjenja podložno je velikim varijacijama usljed proizvodnih tolerancija i promjena temperature ambijenta. Nije pogodno da se oslonimo na to da će vrijeme kašnjenja biti veće ili jednako od neke vrijednosti. Rad ovdje opisanog flipflopa može da bude nepouzdan. Zato je potrebno u njegovoj konfiguraciji izvršiti neke izmjene i dogradnje. JK flipflopovi uvijek se realizuju pomoću složenijih kombinacija sa MS okidanjem ili sa okidanjem na rastuću ili opadajuću ivicu.

T flipflop je ivični flipflop koji mijenja stanje na svaku rastuću ivicu takta. Može da bude realizovan, recimo, pomoću D flipflopa ili pomoću JK flipflopa. Karakteristična jednačina T flipflopa je vrlo prosta i glasi

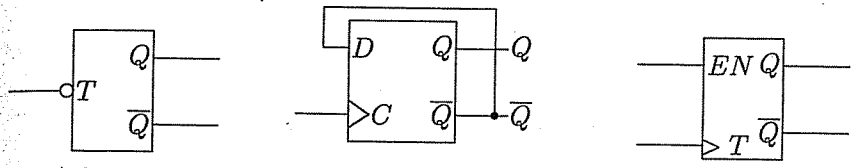
$$Q_{n+1} = \overline{Q_n}.$$

Izlazni signal iz T flipflopa ima tačno dva puta manju učestanost od takta,

pa je zato glavna primjena ove vrste flipflopa u djeliteljima učestanosti.

U nekim primjenama potrebno je zabraniti okidanje T flipflopa. Takva funkcija realizuje se pomoću T flipflopa sa dozvolom, koji može da bude realizovan, na primjer, pomoću D flipflopa ili pomoću JK flipflopa. Za ulaz dozvole propisuje se vrijeme postavljanja t_{su} i vrijeme držanja t_h u odnosu na okidnu ivicu takta. Karakteristična jednačina T flipflopa sa dozvolom glasi

$$Q_{n+1} = \overline{EN}Q_n \vee EN\overline{Q_n}.$$

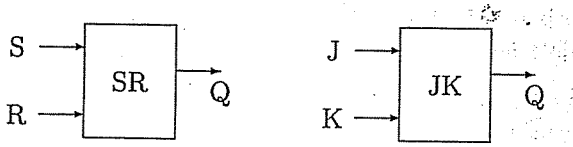


a) Simbol b) Realizacija sa D flipflopom c) T flipflop sa dozvolom

Slika 7.29: T flipflop

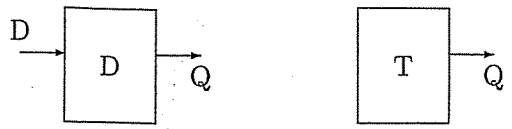
7.11 Rezime o flipflopovima

Nakon što smo opisali sve vrste flipoflopova, daćemo na jednom mjestu (slike 7.30 i 7.31) pregled njihovih uprošćenih simbola i jednačina.



$$\begin{cases} S=0, R=0 \Rightarrow Q_{n+1}=Q_n \\ S=0, R=1 \Rightarrow Q_{n+1}=0 \\ S=1, R=0 \Rightarrow Q_{n+1}=1 \\ S=1, R=1 \Rightarrow Q_{n+1}=? \end{cases} \qquad \begin{cases} J=0, K=0 \Rightarrow Q_{n+1}=Q_n \\ J=0, K=1 \Rightarrow Q_{n+1}=0 \\ J=1, K=0 \Rightarrow Q_{n+1}=1 \\ J=1, K=1 \Rightarrow Q_{n+1}=\overline{Q_n} \end{cases}$$

Slika 7.30: Pregled flipflopova (1)



$$\begin{cases} D=0 \Rightarrow Q_{n+1}=0 \\ D=1 \Rightarrow Q_{n+1}=1 \end{cases} \qquad \{\ Q_{n+1}=\overline{Q_n}$$

Slika 7.31: Pregled flipflopova (2)

7.12 Stacionarni registri

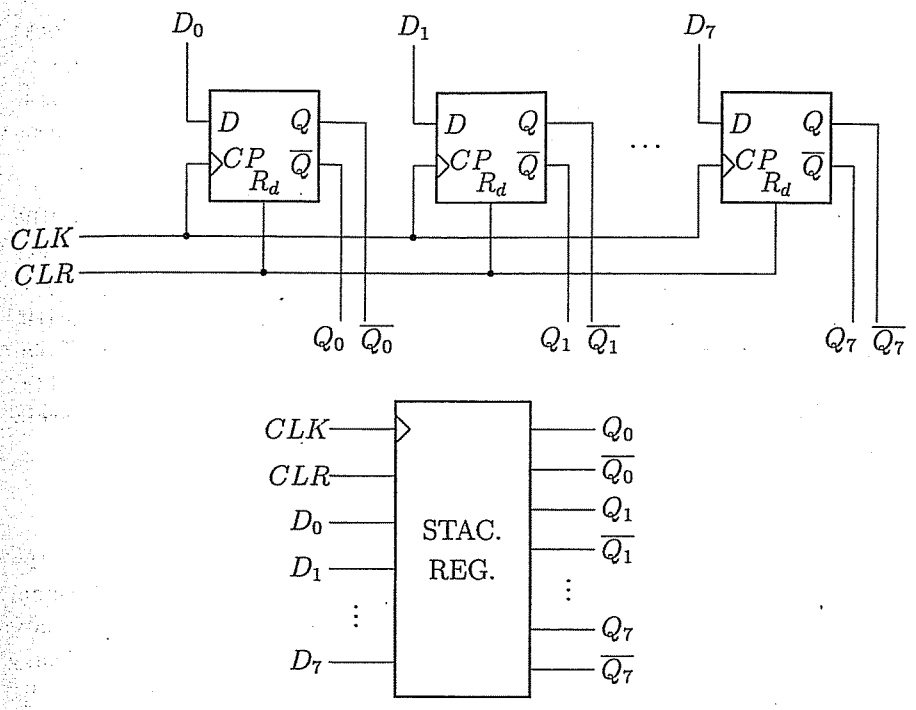
Registar ili stacionarni registar je sekvencijalna mreža koja se koristi za privremeno memorisanje digitalnih informacija. Kao memorijski elementi obično se koriste D flipflopovi, a takođe i D ili SR leč kola. Da bi dva ili više bistabilnih kola predstavljali registar, uslov je da oni imaju zajednički taktni impuls.

Stacionarni registar može da se koristi i za privremeno memorisanje digitalnih podataka prilikom asinhrone razmjene informacija, odnosno između digitalnih uređaja različitih brzina. U takvoj ulozi registar se naziva baferom, po engleskoj riječi *buffer*.

Stacionarni registri koji se najčešće upotrebljavaju koriste ivične D flipflopove kao svoje memorijske elemente. Na slici 7.32 prikazana je logička šema stacionarnog registra u kome se flipflopovi aktiviraju uzlaznom ivicom taktnog impulsa, tj. ulazna informacija upisuje se u registar na uzlaznu ivicu taktnog impulsa. Registar sa slike 7.32 ima i priključak za istovremeno resetovanje svih flipflopova. Signal za reset dovodi se na asinhrone R_D ulaze flipflopova.

Kao integrisane komponente, registri se obično prave sa $n = 2$, $n = 4$ ili $n = 8$ flipflopova. Mogućnost reseta nije uvijek neophodna, tako da se proizvode i stacionarni registri bez priključaka za reset.

Za međusobni vremenski odnos i trajanje ulaznih signala registra važi sve



Slika 7.32: Logička šema i simbol stacionarnog registra ($n = 8$)

što je ranije rečeno za flipflopove. Izlazi flipflopova zauzeće stabilno stanje nakon vremena propagacije t_d od aktivne ivice taktnog impulsa. Za vrijeme dok je signal za reset CLR aktivan u registar se ne može upisati nikakav sadržaj. U praksi je signal za reset CLR kod većine registara aktivan na niskom nivou.

Smisao stacionarnog registra: ako takt onda $Q_i \leftarrow D_i$ za $i = 0, 1, \ldots, 7$, tj. pod uslovom CLK biće urađeno $Q_0Q_1 \ldots Q_7 \leftarrow D_0D_1 \ldots D_7$. Drugim riječima, ulazni podaci D_i upisuju se u flipflopove koji čine registar. Upisane binarne vrijednosti saopštavaju se po linijama Q_i.

Uobičajeno je da se ulazne promjenljive obilježavaju sa D_i ili DAT_i (Data)

ili slovima A, B, C, itd. Taktni impuls obilježava se sa CLK ili C (clock) ili sa PE (Parallel Entry). Signal koji resetuje sve flipflopove u registru obilježava se obično sa CLR (Clear) ili MR (Master Reset). Najzad, izlazi iz registra obilježavaju se sa Q_i; to su stanja flipflopova odnosno to su binarne vrijednosti koje se memorišu.

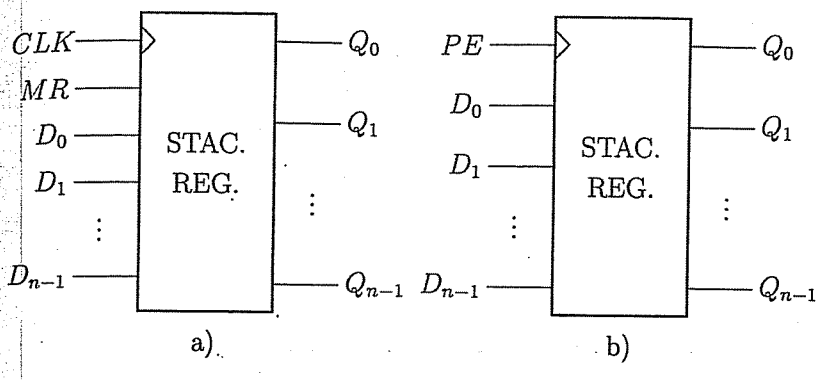
Na slikama 7.32 i 7.33 prikazani su standardni simboli za stacionarne registre. Tako je na slici 7.32 prikazan simbol za registar čija je šema data na toj slici. Na slici 7.33.a) prikazan je simbol za registar koji memoriše ulaznu informaciju nakon uzlazne ivice taktnog impulsa. Na tom registru, komplementarni izlazi flipflopova nisu izvedeni na nožice integrisanog kola. Najzad, registar sa slike 7.33.b) takođe je aktivan na uzlaznu ivicu takta. Ovaj registar nema mogućnost asihronog reseta i nema komplementarnih izlaza flipflopova.

Postoje brojne vrste stacioniranih registara.

- Stacionarni registar čiji su izlazi Q_i trostatički. Za njih je karakteristično prisustvo ulaznog priključka OE (Output Enable). Kada je signal OE neaktivan, onda su izlazi registra u stanju visoke impedanse (praktično, izlazne vrijednosti se ne saopštavaju). Kada je $OE = 1$ registar radi kako je uobičajeno. Takvi registri primjenjuju se u sistemima gdje više digitalnih uređaja razmjenjuje informacije preko zajedničke magistrale.

- Bidirekcioni stacionarni registar. U slučaju kad digitalni uređaj treba i da prima informacije sa magistrale i da šalje informacije na magistralu (da se ne bi koristila dva registra) koristi se samo jedan bidirekcioni stacionarni registar.

- Bit-adresibilni stacionarni registar ili, skraćeno, adresibilni registar je registar kod koga informacija biva upisana u samo jedan njegov bit (u adresirani bit). Ulaz DAT donosi informaciju koja će biti upisana. Uzmimo da je $n = 8$, tj. uzmimo da ima $n = 8$ izlaza $Q_0, Q_1 \ldots, Q_7$. Ulazi A_0, A_1 i A_2 prolaze kroz dekoder oblika 3/8. Tri ulazne vrijednosti A_0, A_1 i A_2 određuju jedan broj i u granicama $0 \leq i \leq 7$. Koju funkciju obavlja takav registar? Pod uslovom ulaza za takt CLK, izvršiće se pridruživanje $Q_i \leftarrow DAT$. Dakle, ulazna infotmacija DAT upisaće se u adresirani, odnosno odabrani flipflop.



Slika 7.33: Simboli za stacionarne registre

7.13 Pomjerački registri

Pomjerački registar ili *šift* registar (engl. *shift register*) je registar u kome se zapamćena informacija pomjera za jedno mjesto, ako je prisutan taktni implus.

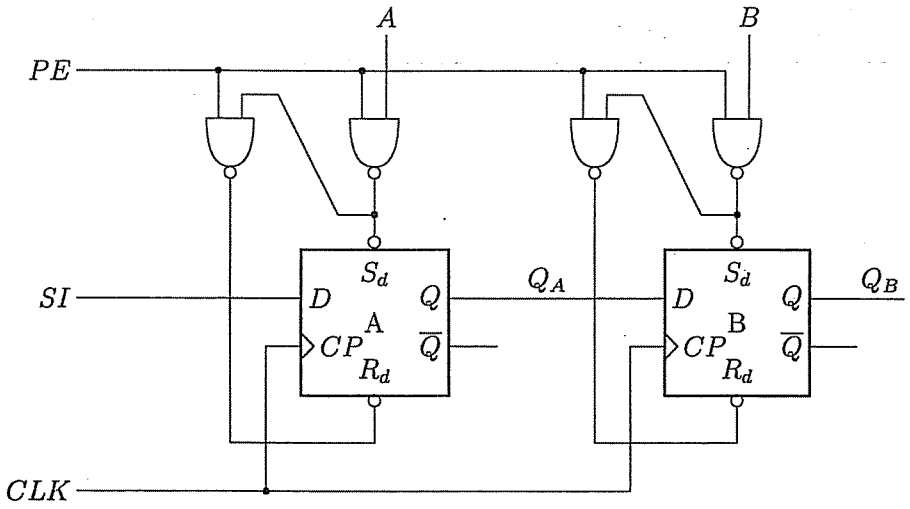
U pomjeračkim registrima mogu se koristiti D ili JK flipflopovi, ivičnog ili MS tipa. Na slici 7.34 prikazana je logička šema pomjeračkog registra čija je veličina dva bita, koji ima još i mogućnost paralelnog upisivanja. Realizovan je pomoću ivičnih D flipflopova.

Razmotrimo funkcionisanje ovog registra. Informacija se upisuje u prikazani pomjerački registar preko asinhronih ulaza S_D i R_D, na način kako je to objašnjeno za stacionarne registre. Ova funkcija je funkcija upisa ili *load* funkcija. Pomjerački registar ima još jednu funkciju - *shift* ili *pomjeranje*.

Ako su ulazni signali A i B bili u stanju $A = 1$ i $B = 0$ za vrijeme dok je PE bio aktivan, onda se u flipflopove Q_A i Q_B upisalo redom 1 i 0. Dakle, sada je $Q_A = 1$ i $Q_B = 0$. Pogledajmo šta se dešava tokom prve periode vremena (tokom jedne periode takta). Za vrijeme aktivne (uzlazne) ivice prvog CLK impulsa ulaz D_A je na logičkom nivou SI, a ulaz D_B je na logičkom nivou Q_A. Nakon vremena propagacije kroz flipflopove postaće $Q_A = 0$ i $Q_B = 1$, pod pretpostavkom da je $SI = 0$. Tako se sadržaj flipflopa A pomjerio u flipflop B.

Možemo razmotriti i šta se dešava tokom druge periode vremena. Sljedeća aktivna ivica taktnog impulsa ponovo pomjeri sadržaj u registru, tako da onda u registru ostaju nule na oba mjesta. Prema tome, za registar sa slike 7.34 važe sljedeće dvije formule:

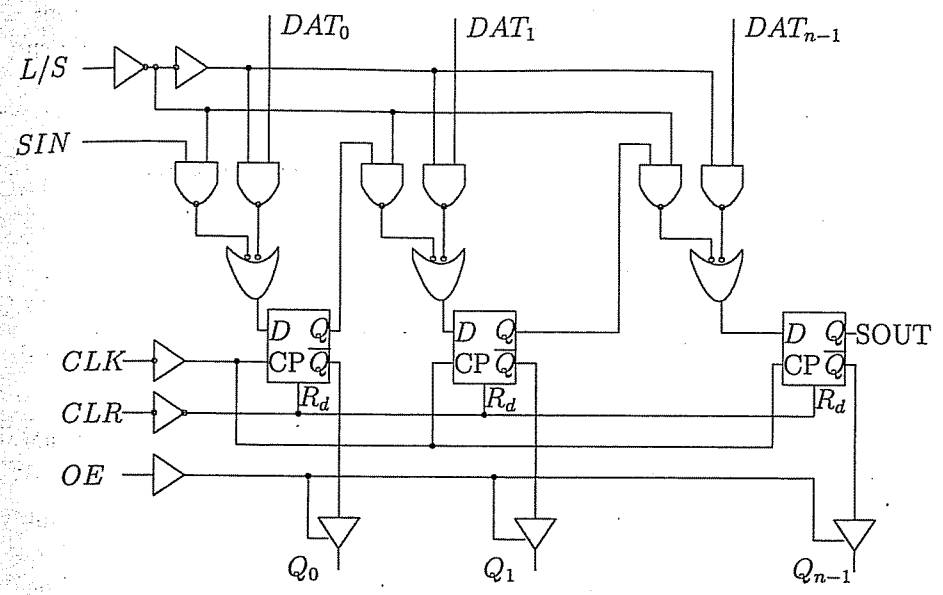
$$PE : Q_A \leftarrow A,\ Q_B \leftarrow B \quad \text{i} \quad CLK : Q_B \leftarrow Q_A,\ Q_A \leftarrow SI$$



Slika 7.34: Logička šema dvobitnog pomjeračkog registra

U slučaju kada se za pomjerački registar koriste JK flipflopovi, Q izlaz prethodnog flipflopa vezuje se za J ulaz narednog, a $\overline{Q}$ izlaz prethodnog na K ulaz narednog. Paralelni upis može se obavljati preko asinhronih ulaza S_D i R_D, na isti način kao što je to prikazano na slici 7.34.

Postoje brojne realizacije integrisanih pomjeračkih registara. Uglavnom se koriste ivični D flipflopovi. Paralelni upis podataka može se obavljati na način kako je to prikazano na slici 7.34 (asinhrono). Međutim, većina registara koristi taktovani odnosno sinhroni paralelni upis. Šema takvog jednog registra data je na slici 7.35, a njegov simbol na slici 7.36.a).



Slika 7.35: Integrisani pomjerački registar

Slika 7.35. prikazuje šemu pomjeračkog registra koji ima i mogućnost paralelnog upisa, a pored toga i mogućnost paralelnog čitanja. Kada je signal L/S u stanju $L/S = 1$, onda se na D ulazima flipflopova nalazi prisutan logički nivo ulazne informacije DAT_i. Tako da se onda generisanjem takta informacija upisana u registru pomjera za jedno mjesto udesno, a u Q_0 se (aktivnom ivicom takta) upisuje logički nivo ulazne promjenljive SIN (*serial input*). Flipflopovi se prebacuju silaznom ivicom takta CLK.

L/S ovdje znači Load/Shift ili, u prevodu, unesi ili pomjeri. Vrijednost signala L/S razvodi se na n dvoulaznih multipleksera (realizovanih pomoću NI kola), čime se postižu dvije različite funkcije - jedna kada je $L/S = 1$, a druga kada je $L/S = 0$.

Registar može da se resetuje asinhrono pomoću signala CLR, čiji je aktivni nivo nula. Na izlaznim linijama registra prisutna su trostatička kola koja se aktiviraju pomoću signala OE.

U sljedećoj tabeli (7.4) prikazana su stanja pomjeračkog registra koji razmatramo, u zavisnosti od ulaznih promjenljivih. Neka je $n = 8$. U tabeli

se koriste sljedeće oznake: $Q_0, Q_1, \ldots, Q_7$ su stare vrijednosti izlaznih promjenljivih, a $Q'_0, Q'_1, \ldots, Q'_7$ nove. U tabeli se pomoću slova Z označava da su izlazi registra u stanju visoke impedanse.

L/S	CLR	OE	Q'_0	Q'_1	Q'_2	...	Q'_7
x	x	0	Z	Z	Z	...	Z
x	0	1	0	0	0	...	0
1	1	1	DAT_0	DAT_1	DAT_2	...	DAT_7
0	1	1	SIN	Q_0	Q_1	...	Q_6

Tabela 7.4: Tabela stanja integrisanog pomjeračkog registra

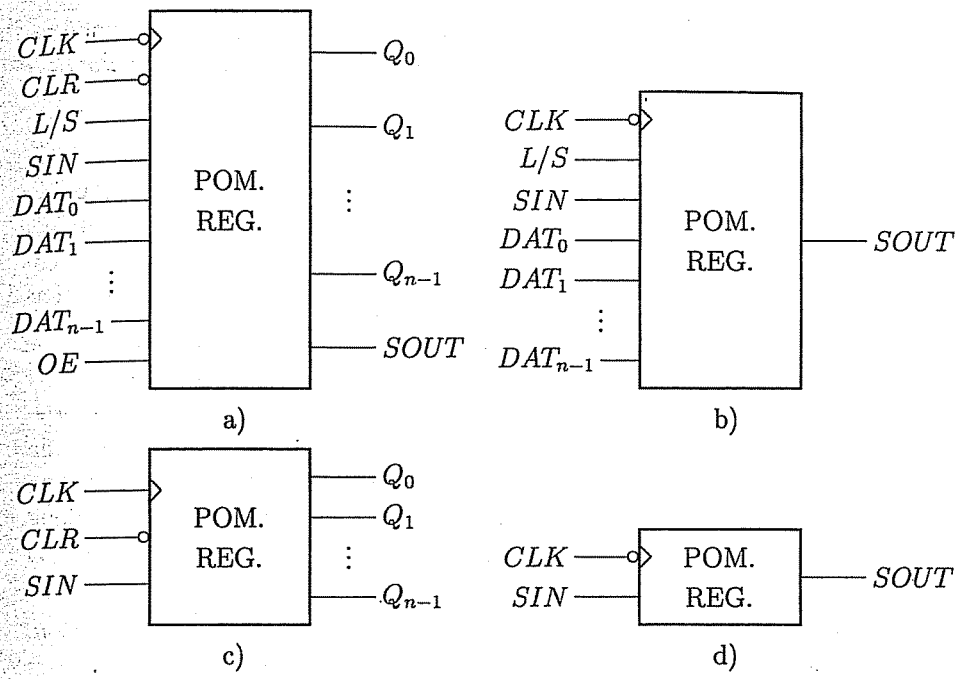
Možemo još definisati u posljednjem redu $SOUT = Q_7$. $SOUT$ znači serijski izlazni priključak (serial-out).

Ovakav integrisani pomjerački registar proizvodi se kao integrisana komponenta sa četiri ili osam flipflopova. Kaskadnim vezivanjem više četvorobitnih ili osmobitnih registara može se formirati pomjerački registar proizvoljne dužine. Kaskadno vezivanje obavlja se tako što se izlazni serijski priključak $SOUT$ priključi na SIN priključak narednog pomjeračkog registra.

Integrisani pomjerački registri nemaju uvijek sve mogućnosti prethodno opisanog registra. Postoje registri bez mogućnosti reseta ili sa izlazima koji nisu trostatički. Takođe, postoje integrisani pomjerački registri kod kojih izlazi flipflopova nisu svi izvedeni na priključke, već je izveden samo izlaz posljednjeg flipflopa. Takav registar naziva se registrom sa paralelnim ulazom i serijskim izlazom (parallel-in, serial-out). Njegov grafički simbol dat je na slici 7.36.b).

Slika 7.36.c) odgovara pomjeračkom registru sa serijskim ulazom i paralelnim izlazom. U takav registar informacija može da se upiše samo serijski i on ne sadrži multipleksere na D ulazima flipflpova. Registar ima mogućnost reseta, a izlazi nisu trostatički. Za registar sa slike 7.36.c) važi sljedeće: ako je njegovo staro stanje bilo $q_0q_1 \ldots q_7$, onda će novo stanje biti $q'_0q'_1 \ldots q'_7 = 0q_0 \ldots q_6$, pod uslovom da je $SIN = 0$. Recimo, ako je staro stanje bilo 11111000 novo stanje će biti 01111100. Dakle, kao da smo binarni broj podijelili sa dva.

Slika 7.36.d) odgovara pomjeračkom registru sa serijskim ulazom i serijskim izlazom, bez mogućnosti reseta.



Slika 7.36: Simboli integrisanih pomjeračkih registara

Ako se izlazni priključak $SOUT$ jednog pomjeračkog registra (čija je veličina n bita) veže na ulazni priključak SIN drugog registra (iste veličine), onda će sadržaj prvog da bude upisan u drugi kada prođe vrijeme od n taktova, zato što pomjeračkom registru odgovara jednačina

$$CLK : SOUT \leftarrow Q_7,\ Q_7 \leftarrow Q_6,\ Q_6 \leftarrow Q_5, \ldots,\ Q_1 \leftarrow Q_0,\ Q_0 \leftarrow SIN.$$

Na primjer, neka je stanje prvog registra u trenutku $t = 0$ jednako $a_1a_2a_3a_4$ a drugog $b_1b_2b_3b_4$. U trenutku $t = 1$ stanje ovih registara je $0a_1a_2a_3\ \ a_4b_1b_2b_3$, a u trenutku $t = 2$ imamo $00a_1a_2\ \ a_3a_4b_1b_2$. U trenutku $t = 3$ stanje je $000a_1\ \ a_2a_3a_4b_1$ i, konačno, u trenutku $t = 4$ imamo $0000\ \ a_1a_2a_3a_4$.

Postoje i brojne druge vrste pomjeračkih registara.

- Kod *bidirekcionog pomjeračkog registara* upisana informacija pomjera se udesno ili ulijevo, zavisno od logičkog nivoa kontrolnog signala L/R (*Left/Right* - Lijevo/Desno).

- *Kružni brojač* (ring counter) je pomjerački registar kod koga je izlaz posljednjeg flipflopa spojen sa serijskim ulazam, tj. $SOUT$ je spojen sa SIN. Informacija zapisana u kružnom brojaču će se, bit po bit, ponovo upisivati u brojač.

7.14 Serijski sabirač

Sada ćemo konstruisati mrežu za sabiranje dva binarno prikazana broja $x = (x_n \dots x_2 x_1)_2$ i $y = (y_n \dots y_2 y_1)_2$ od po n bita. Za tu mrežu kaže se da predstavlja *serijski sabirač* budući da je potrebno n taktova da se izračuna rezultat $z = x + y = (z_n \dots z_2 z_1)_2$. Na ulaz mreže redom pristižu cifre jednog i drugog sabirka x_k i y_k, a isto tako se postepeno obrazuju i cifre zbira z_k, gdje je $k = 1, 2, \dots, n$. Tri broja x, y i z drže se u tri pomjeračka registra.

Neka je $x = 2^{n-1}x_n + \dots + 2x_2 + x_1$, gdje $x_k \in \{0, 1\}$ za $1 \leq k \leq n$, i slično za y i z. Ovdje je $n \geq 1$, a u primjeru kasnije biće izabrano $n = 8$. Dobro su poznate sljedeće relacije koje odražavaju zakon sabiranja u binarnom brojnom sistemu:

$$\begin{cases} z_k = x_k \oplus y_k \oplus q_{k-1} \\ q_k = x_k y_k \vee x_k q_{k-1} \vee y_k q_{k-1} \end{cases}$$

Pomoćna promjenljiva q_{k-1} ima smisao prenosa koji se formira prilikom sabiranja na $(k-1)$–voj koordinati, a zatim utiče na formiranje k–te cifre zbira. Upravo, stari prenos q_{k-1} utiče na formiranje tekuće cifre zbira z_k i novog prenosa q_k. Treba staviti $q_0 = 0$.

Šema serijskog sabirača prikazana je na slici 7.37. Serijski sabirač je jedna sekvencijalna mreža: sastavljen je od logičkih elemenata, koji čine logičku mrežu na slici, i memorijskog elementa – jednog D flipflopa. Znamo da je izlazni signal Q tog flipflopa jednak njegovom ulaznom signalu D, ali iz prethodnog takta, što opisujemo formulom $Q(t) = D(t-1)$, gdje je t broj takta ili vrijeme (koliko μsec). Za $Q(t)$ se kaže da je tekuće stanje flipflopa. Vidimo da je na slici povezano tako da važi $q(t) = D(t)$ za svako t i $Q(t) = q'(t)$ za svako t. Sliku bi trebalo upotpuniti: prikazati i upravljačke

signale koji zadaju start i stop za cirkulaciju podataka u ukupnoj mreži i koji još zadaju da se na samom početku flipflop resetuje $Q(0) \leftarrow 0$.

Na ulaze serijskog sabirača pristižu iz takta u takt po jedna cifra prvog i drugog sabirka, a na izlazu se saopštava jedna cifra rezultata. Takođe, iz takta u takt mijenja se stanje flipflopa. Upravljajući se po šemi, analizirati rad predložene mreže odnosno pratiti kako se tokom vremena mijenjaju vrijednosti u karakterističnim tačkama mreže. Pogodno je da se zapiše u obliku tabele 7.5:

$t = \dots$ (μsec)	0	1	2		n
ulaz $x =$		x_1	x_2		x_n
ulaz $y =$		y_1	y_2		y_n
stanje $q =$	0	q_1	q_2		q_n
izlaz $z =$		z_1	z_2		z_n

Tabela 7.5: Rad serijskog sabirača

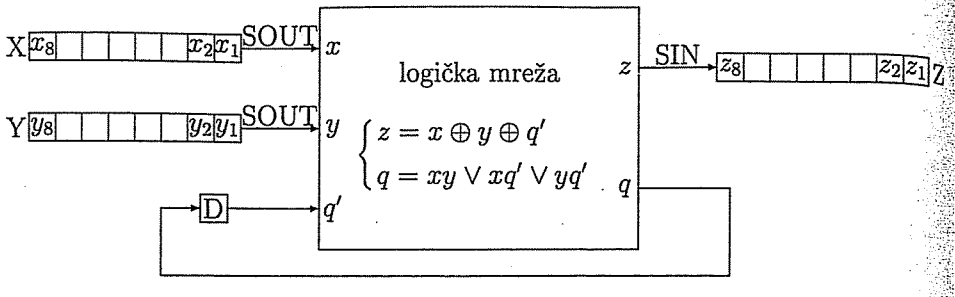
Pogledajmo primjer 00000110 + 00000011 = 00001001 (binarno) ili 6 + 3 = 9 (dekadno). U razmatranoj mreži važi $q'(t) = q(t-1)$ (tabela 7.6).

$t =$	0	1	2	3	4	5	6	7	8
$x =$		0	1	1	0	0	0	0	0
$y =$		1	1	0	0	0	0	0	0
$q =$	0	0	1	1	0	0	0	0	0
$z =$		1	0	0	1	0	0	0	0

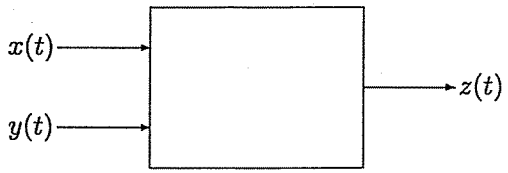
Tabela 7.6: Primjer rada serijskog sabirača

Svaka sekvencijalna mreža može da bude realizovana ako su na raspolaganje stavljeni logički elementi iz nekog kompletnog sistema plus elementi jediničnog kašnjenja D čija jednačina glasi izlaz(t) = ulaz$(t-1)$.

SIN je serijski ulazni priključak pomjeračkog registra (šift registra), a SOUT je njegov serijski izlazni priključak.



Slika 7.37: Serijski sabirač; flipflop D: izlaz(t) = ulaz$(t-1)$.



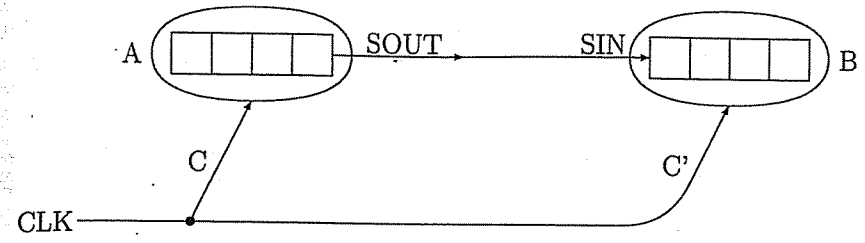
Slika 7.38: Serijski sabirač kao cjelina

7.15 Serijski i paralelni prenos digitalnih informacija

Na slici 7.39 prikazan je serijski prenos digitalnih informacija. Mreža se sastoji od dva četvorobitna pomjeračka registra A i B. Predajni registar A ima serijski izlaz SOUT a prijemni registar B ima serijski ulaz SIN. SOUT se veže na SIN. Signal takta CLK je zajednički a kod A i B označen je kao C odnosno C', i uvodi se na ulaze za dozvolu flipflopova. Vidi se da će nakon generisanja četiri taktna impulsa biti postignuto B $\leftarrow$ A.

Po taktovima t u dva registra A i B imamo stanje kao na slici 7.40 (lijevo je A, desno B).

Registar A može biti lociran u digitalnom uređaju koji šalje podatke, a registar B može biti fizički dislociran kao sastavni dio uređaja kome se podaci šalju. Vidimo da se informacija prenosi na fizički udaljeno mjesto korišćenjem samo dvije električne veze: informacione linije SOUT – SIN i



Slika 7.39: Serijski prenos

$t=0$	a_1	a_2	a_3	a_4	b_1	b_2	b_3	b_4
$t=1$	0	a_1	a_2	a_3	a_4	b_1	b_2	b_3
$t=2$	0	0	a_1	a_2	a_3	a_4	b_1	b_2
$t=3$	0	0	0	a_1	a_2	a_3	a_4	b_1
$t=4$	0	0	0	0	a_1	a_2	a_3	a_4

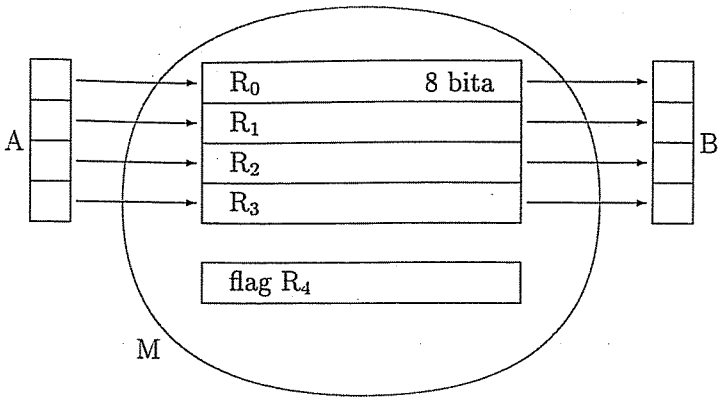
Slika 7.40: Stanje u registrima

linije takta CLK – C – C'. Kaže se da se vrši *sinhroni* serijski prenos.

Kod *asinhronog* serijskog prenosa šalje se samo informacija, dok se takt prijemnog pomjeračkog registra generiše lokalno. To omogućava da se informacije prenose samo po jednoj liniji, što je naročito pogodno u sistemima gdje se za prenos koriste standardni telekomunikacioni sistemi.

Slika 7.41 ilustruje paralelni prenos digitalnih informacija. Posmatrajmo predajni registar A centralnog procesora i prijemni registar B štampača. Štampač radi puno sporije, pa želimo da centralni procesor šalje podatke po paketima. Između A i B postavi se mreža M za koju se kaže da predstavlja *bafer*. Mreža M sastoji se od $m+1$ pomjeračkih registara R_0, R_1, ..., R_{m-1}, R_m, za koje kažemo da su *informacioni registri*.

Registar R_m ima kontrolnu ulogu i naziva se *flag registrom*. Uzeli smo u našem primjeru da je $m = 4$. Svi pomjerački registri R_0, R_1, ..., R_{m-1}, R_m su jedne te iste veličine, po n bita. U primjeru je $n = 8$. S druge strane, dva



Slika 7.41: Paralelni prenos

registra A i B su veličine po m bita.

Ukupna mreža A – M – B funkcioniše na sljedeći način. Tokom prvog takta $t = 1$ iz A se pošalje po jedan bit informacije svakom informacionom pomeračkom registru R_0, R_1, ..., R_{m-1}, tokom $t = 2$ pošalje im se po još jedan bit, itd. Poslije $t = n$ čitav paket je poslat i bafer je pun. Sljedeći koraci vršiće se po drugom taktu, znatno sporijem taktu, koji odgovara registru B (dakle, odgovara štampaču). Informacije koje su tokom $t = 1$ bile poslate iz A u M sada se nalaze u R_0, R_1, ..., R_{m-1} u krajnjim desnim flipflopovima i one će tokom sljedećeg takta biti poslate registru B. Znači, u nastavku se (tokom n koraka) informacije iz M postepeno šalju prijemnom digitalnom uređaju.

U ukupnoj mreži A – M – B, za regulisanje komunikacije, pored flag registra, služe i dodatne linije veze.

Dok se sadržaj bafera bude štampao, centralni procesor će obavljati druge radnje. Kada se štampanje jednog paketa podataka završi, uređaj A dobija obavještenje o tome, tako da uređaj A sada može da pošalje novi paket podataka za štampanje.

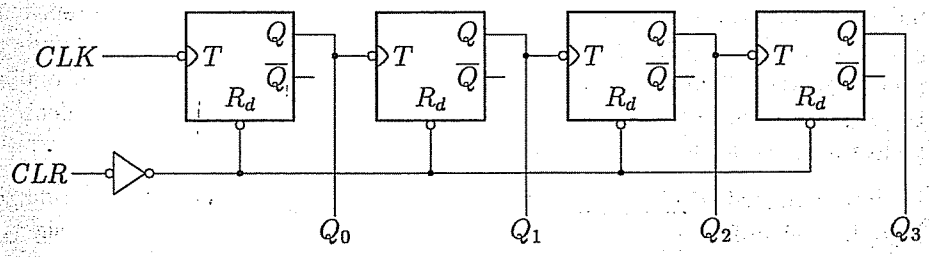
7.16 Brojači

Govoreći uopšteno, brojač se može opisati na sljedeći način. Uzmimo da brojač ima $n = 8$ bita. Neka je njegovo prethodno stanje označeno kao $q = q_7q_6 \ldots q_0$, a novo stanje kao $q' = q'_7q'_6 \ldots q'_0$. Tada važi jednakost $q' = q + 1$. Na primjer, ako je staro stanje bilo $q = 10000011$ onda će novo stanje biti $q' = 10000100$.

Brojač je sekvencijalna mreža kod koje se upisana vrijednost iz takta u takt povećava za jedan. Upisane vrijednosti tokom vremena obrazuju ciklus koji se ponavlja. Broj različitih stanja u ciklusu naziva se *osnovom brojača* (*modulom brojača*).

Kao memorijski elementi u brojačima se koriste flipflopovi. Razmotrimo brojač koji se sastoji od n flipflopova. Uzmimo da brojač ima $m = 2^n$ stanja i da se stanja smjenjuju u sekvenci binarnih brojeva. Tada se kaže da je to jedan *n-bitni binarni brojač* ili da je to binarni brojač čiji je moduo brojanja jednak $m = 2^n$.

Ako se svi flipflopovi u brojaču taktuju zajedničkim taktnim impulsom, onda se brojač naziva *sinhronim*. Ako taktni impuls nije zajednički za sve flipflopove, brojač je *asihroni*.



Slika 7.42: Kaskadna veza T flipflopova

Kada se na prvi u lancu ivičnih flipflopova vrste T prikazanih na slici 7.42 priključi povorka taktnih impulsa CLK onda će se na izlazima flipflopova dobiti talasni oblici prikazani na slici 7.43; naravno, umjesto cifara 1 i 0 možemo nacrtati visoki odnosno niski naponski nivo. Vidimo da na slici 7.42 učestvuju T flipflopovi, koji se prebacuju silaznom ivicom impulsa priključenog na T ulaz, i da se svaki naredni flipflop prebacuje silaznom ivicom izlaznog signala

prethodnog flipflopa. Prebacivanje se može obavljati kada su R_d ulazi na neaktivnom logičkom nivou. Razmatrani brojač može da bude postavljen u stanje $Q_3Q_2Q_1Q_0 = 0000$ dovođenjem signala za reset CLR na aktivni logički nivo (na niski nivo).

Sa slike 7.43 vidi se da stanja četiri flipflopa Q_0 do Q_3 u vremenskim trenucima označenim kao $k = 0, \ldots, k = 15$ kodiraju binarni broj koji izražava broj taktnih impulsa koji su do posmatranog trenutka djelovali na ulaz flipflopa Q_0. Prema tome, mreža prikazana na slici 7.42 obavlja funkciju brojanja taktnih impulsa, gdje se broj prikazuje u prirodnom binarnom kodu. Flipflopovi u mreži ne prebacuju se zajedničkim taktnim impulsom. Za takvu mrežu kaže se da predstavlja *asinhroni binarni brojač*. Vidimo da se prebacivanje flipflopova izvodi serijski. Zato se takav brojač često naziva i *serijskim* ili *rednim* brojačem, a u engleskoj literaturi *ripple counter*, što se prevodi kao brojač sa prenošenjem talasa.

Ista mreža može da bude realizovana i korišćenjem JK flipflopova. J i K ulazi svih flipflopova u toj mreži vezani su za logički nivo 1. Q izlaz svakog flipflopa u lancu priključen je na taktni ulaz narednog flipflopa (isto kao na slici 7.42).

Asinhroni brojači imaju ograničenu primjenu upravo zbog asinhronog načina prebacivanja flipflopova u lancu. Uzmajući u obzir vrijeme propagacije kroz pojedini flipflop i uzimajući u obzir da se svaki naredni flipflop prebacuje tek nakon isteka vremena propagacije prethodnog, onda tokom trajanja prelaznih stanja brojač kodira pogrešan sadržaj. Tako da navedeni nedostatak asinhronog brojača dovodi do pojave neregularnih stanja.

Pored toga, taj nedostatak ograničava maksimalnu učestanost takta na kojoj brojač može ispravno da radi. Naime, sljedeća aktivna ivica taktnog signala ne smije da se pojavi prije nego što se završi prethodno prelazno stanje. Asinhroni brojači upotrebljavaju se uglavnom kao djelitelji učestanosti. Oni

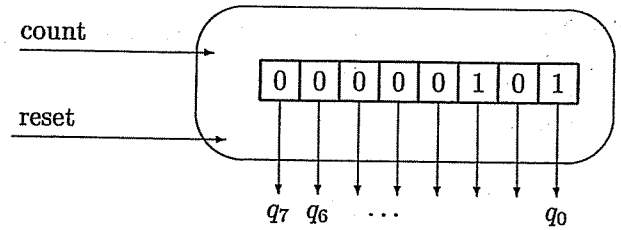
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
Q_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	
Q_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	
Q_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	
Q_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	

Slika 7.43: Vremenski dijagrami mreže sa slike 7.42.

se koriste u mrežama gdje nije potrebno dekodiranje stanja brojača (čitanje upisane vrijednosti). Koristi se jedino izlaz posljednjeg flipflopa u lancu, to je $Q_{n-1} = Q_3$ na slici 7.42. Ovaj signal koristi se kao signal čija je učestanost 2^n puta niža od učestanosti ulaznog takta.

Postoje razne vrste sekvencijalnih mreža za koje se kaže da predstavljaju brojače:

- Asinhroni brojači. Njih smo upravo opisali.
- Sinhroni brojači. Za razliku od asnihronih, kod sinhronih brojača svi flipflopovi prebacuju se sinhrono zajedničkim taktnim impulsom. Uslov brojanja može da se prenosi serijski ili paralelno.
- Brojači unazad (*down counter*). To su brojači koji koduju opadajući niz binarnih brojeva.
- Obostrani brojači (*up-down counter*). Ova vrsta brojača, zavisno od kontrolnog signala, može da broji unaprijed ili unazad.
- Brojači sa mogućnošću paralelnog upisa. Ovakvi brojači sastavljaju se od flipflopova koji posjeduju asinhrone S_D i R_D ulaze. Prije početka brojanja, upiše se inicijalni sadržaj.
- Brojač čiji je moduo 10. Ovaj brojač pripada klasi brojača čija osnova nije stepen dvojke. Brojač čiji je moduo 10 može da bude sastavljen od 4 flipflopa.



Slika 7.44: Brojač čiji je trenutni sadržaj $(00000101)_2 = 5$.

Generalno, brojač možemo predstaviti kao na slici 7.44. Po liniji *count* šalje se uslov brojanja, tako da važi relacija *count* : $q \leftarrow q+1$. Po liniji *reset* zadaje se svođenje na nulu; *reset* : $q \leftarrow 0$; ako reset onda q dobija vrijednost nula. Ovdje je $q = (q_7 q_6 \dots q_0)_2$.

7.17 Zadaci za vježbu

1. Logička mreža za promjenu predznaka broja koji je predstavljen u obliku potpunog komplementa. Neka je $n = 8$, $x = (x_8 x_7 \dots x_1)_{\text{PK}}$, $y = (y_8 y_7 \dots y_1)_{\text{PK}}$ i $y = -x$ i neka su $x_8, x_7, \dots, x_1$ ulazne veličine a $y_8, y_7, \dots, y_1$ izlazne. Sastaviti odgovarajuće logičke jednačine i nacrtati šemu mreže.

2. Logička mreža za upoređivanje dva broja. Neka je $n = 8$ ili uopšte $n \geq 1$ i razmotrimo funkciju algebre logike y koja zavisi od $2n$ promjenljivih $x_{11}, \dots, x_{1n}, x_{21}, \dots, x_{2n}$ a definisana je uslovom: $y = 1$ ako i samo ako je $x_{1i} = x_{2i}$ za svako $i \in \{1, \dots, n\}$. Sastaviti izraz za y. Neka u izrazu učestvuju samo I, ILI i NE. Za razmatrano digitalno kolo kaže se da predstavlja komparator, jer ispituje da li je $x_1 = x_2$.

3. Logička mreža za sabiranje u BCD kodu u slučaju "višak 3". Radi jednostavnosti neka je prvi sabirak x dvocifren, a takođe i drugi sabirak y kao i rezultat $z = x + y$. Neka je $x = 10x_1 + x_2$, $0 \leq x_1 \leq 9$, $0 \leq x_2 \leq 9$ i slično $y = 10y_1 + y_2$ i $z = 10z_1 + z_2$. Neka se x_1 kodira pomoću četiri binarne promjenljive $x_{11}, x_{12}, x_{13}, x_{14}$, a x_2 pomoću $x_{21}, x_{22}, x_{23}, x_{24}$, itd. Vidi se da ulazne veličine treba da budu $x_{11}, \dots, y_{24}$ a izlazne $z_{11}, \dots, z_{24}$. Kakva je funkcionalna zavisnost izlaznih veličina od ulaznih veličina?

4. Sekvencijalna mreža za množenje dva pozitivna broja. Proizvod $X = A \cdot B$ prikazuje se pomoću dvije riječi. Neka je $0 \leq A \leq 2^{32} - 1$, $0 \leq B \leq 2^{32} - 1$, $0 \leq X \leq 2^{64} - 1$ i $A = (a_{32} \dots a_1)_2$, sa vodećim nulama. Sastaviti odgovarajuću mrežu. Pored tri vrste logičkih elemenata I, ILI i NE neka se i jedan memorijski element D flipflop pojavljuje kao komponenta. Dovoljno je da se sastavi grubi plan mreže. Uputstvo.

$$
\begin{aligned}
X_1 &= a_{32}B, \\
X_2 &= 2X_1 + a_{31}B, \\
X_3 &= 2X_2 + a_{30}B, \quad \dots, \\
X_k &= 2X_{k-1} + a_{33-k}B, \quad \dots, \\
X &= X_{32} = 2X_{31} + a_1 B.
\end{aligned}
$$

5. Sekvencijalna mreža za množenje dva cijela broja koji su predstavljeni u obliku potpunog komplementa. Traženu mrežu dobiti dogradnjom mreže množača iz prethodnog zadatka. Dovoljno je da se izanalizira korekcija Δ. Uputstvo. Pogledajmo primjer $a = 6$ i $b = -5$. Tako da je $a^* = 6$ i $b^* = 2^{32} - 5$. Množač će saopštiti $6(2^{32} - 5)$, što treba uvećati za $\Delta = 2^{32}(2^{32} - 6)$ pomoću sabirača za riječi dvostruke dužine (znamo da pomnožiti sa 2^{32} znači izvršiti šiftovanje). Time se dobija tačan rezultat $(a \cdot b)^* = 2^{64} - 30$ dvostruka riječ. Pogledajmo i drugi primjer $a = -6$ i $b = -5$. U ovom slučaju množač će produkovati $(2^{32} - 6) \cdot (2^{32} - 5)$, na šta treba dodati $2^{32} \cdot 6$ i $2^{32} \cdot 5$ pomoću sabirača za riječi dvostruke dužine, čime će biti dobijen tačan rezultat 30. Zato što će prekobrojno 2^{64} biti poništeno kao prenos.

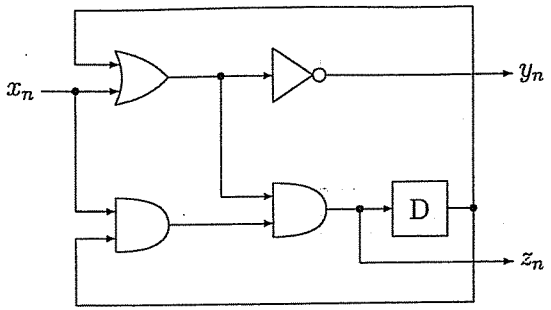
6. Sinteza sekvencijalne mreže digitalni časovnik, u najgrubljim mogućim crtama. Iz koraka u korak neka se na izlazu redom saopštava 00:00, 00:01, ..., 11:59, a zatim sve iznova. Dekadna cifra kodira se pomoću četvorke binarnih cifara.

7. Sinteza sekvencijalne mreže elektronska brava, u osnovnim crtama. Ako se zada $a = 1$ onda neka se brava zaključa; closed $\leftarrow 1$. Ako se ulaznim veličinama $b_1, \dots, b_8$ zadaju u istom taktu specifične vrijednosti $b_{01}, \dots, b_{08}$ (predstavljaju šifru) onda neka se brava otključa; open $\leftarrow 1$. Pored toga, ako se tri puta uzastopno pokuša sa pogrešnom šifrom onda nakon toga treba da se brava definitivno zaključa; izlazna veličina lock $\leftarrow 1$.

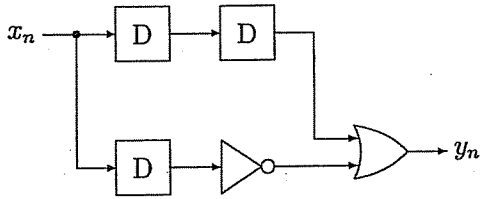
8. Sastaviti logičku mrežu za funkciju $f(x_1, x_2, x_3, x_4, x_5, x_6)$ koja je jednaka jedinici ako i samo ako je $(x_1, x_2, x_3) \preceq (x_4, x_5, x_6)$.

9. Sastaviti logičku mrežu čije su ulazne veličine x_1, x_2 i x_3, a koja ispunjava sljedeće uslove: i) Na izlazu mreže saopštavaju se $\overline{x_1}$, $\overline{x_2}$ i $\overline{x_3}$. ii) Koriste se jedino logički elementi vrsta I, ILI i NE. iii) U mreži, NE elemenata ima najviše dva. Elemenata I i ILI može biti proizvoljno.

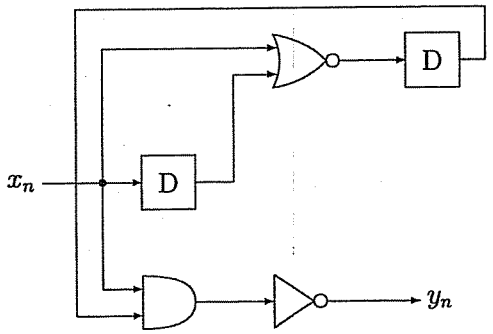
10. Sastaviti jednačine i tablicu za mrežu prikazanu na donjoj slici. Na slici, sa D je označen D flipflop čija jednačina (kao što znamo) glasi $Q_n = D_{n-1}$ ili svejedno element jedinične zadrške D (engl. delay) čija jednačina glasi $\text{output}(t) = \text{input}(t-1)$.



11. Sastaviti jednačine i tablicu za mrežu prikazanu na slici.



12. Sastaviti jednačine i tablicu za mrežu prikazanu na slici

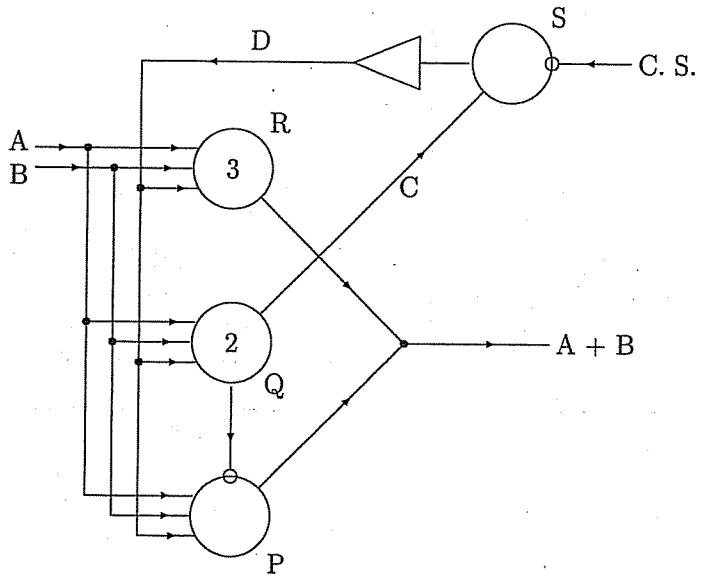


13. Fon Nojmanov sabirač. To je serijski sabirač sastavljen od E–elemenata. Na ulaze input_1 i input_2 dovode se redom jedna po jedna cifra prvog odnosno

drugog sabirka, dok se po izlaznoj liniji output redom iz takta u takt saopštava cifra za cifrom zbira. Naravno, dvije ulazne sekvence cifara međusobno su vremenski usaglašene; dvije slične cifre dolaze simultano. Računa se i prenos za naredno mjesto (carry over). Ovaj primjer navodi se zbog značaja koji ima u istoriji računara. Preuzet je iz "First Draft of a Report to the EDVAC", Philadelphia, Pennsylvania, USA, 1945. godine. Rješenje vidimo na slici. Analizom se uvjeriti da predložena mreža zaista ostvaruje funkciju sabiranja.

Na slici, linije input_1, input_2 i output označene su kao A, B i A+B redom. Po liniji C. S. dolazi tzv. carry suppression signal (poništavanje prenosa). Jednak je nuli 31 puta, a onda je tokom jednog takta jednak 1. Trougao je simbol za element kašnjenja (element jedinične zadrške).

output = P ∨ R; nije istovremeno P = 1 i R = 1.



Glava 8

Komponente računarskog sistema

Računar se sastoji iz pet glavnih komponenti: ulazne, izlazne, memorijske, aritmetičko-logičke i upravljačke jedinice. U ovom poglavlju detaljnije ćemo opisati komponente i principe funkcionisanja računarskog sistema.

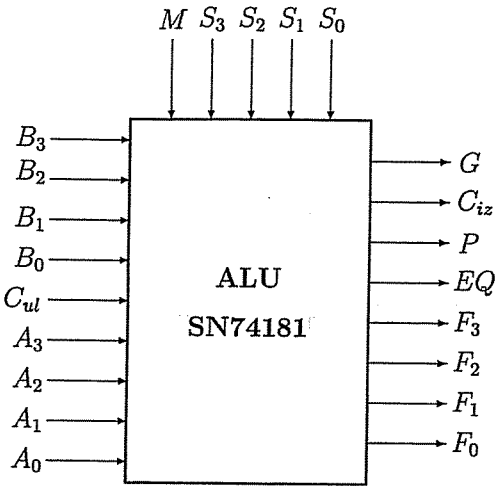
8.1 Aritmetičko–logička jedinica

Aritmetička kola u digitalnim sistemima obavljaju osnovne aritmetičke operacije: sabiranje, oduzimanje, množenje i dijeljenje. U klasu aritmetičkih kola spadaju i mreže za komplementiranje brojeva, mreže za poređenje brojeva i *aritmetičko–logičke jedinice (ALU)*, koje mogu da obavljaju aritmetičke i logičke operacije.

Osnovna aritmetička kola spadaju u klasu kombinacionih mreža. Međutim, aritmetička kola najčešće se koriste u mrežama koje pored kombinacionih elemenata koriste i sekvencijalne (memorijske) elemente, kako bi se pamtili rezultati složenijih aritmetičkih operacija. Kao integrisane komponente, aritmetička kola se izrađuju za operacije sa brojevima izraženim u binarnom brojnom sistemu i (rijetko) sa brojevima izraženim u BCD kodu.

Aritmetičko–logičke jedinice, engl. arithmetic-logic unit, ALU, su višefunkcionalne mreže koje zavisno od kontrolnih ulaza mogu da obavljaju različite aritmetičke ili logičke operacije nad dva broja od po n cifara.

Razmotrimo jedan primjer ALU - komponentu $SN74181$, čiji je simbol prikazan na slici 8.1. Ova integrisana aritmetičko-logička jedinica izrađena je za operacije nad četvorobitnim brojevima i ima 5 kontrolnih ulaza, tako da



Slika 8.1: Logička šema ALU $SN74181$.

može da obavlja do $2^5 = 32$ različitih funkcija. Komponenta je sastavljena od velikog broja logičkih elemenata, pa nećemo detaljnije razmatrati njenu unutrašnju strukturu, već ćemo opisati princip njenog rada.

Kao prvo, opišimo smisao spoljašnjih priključaka komponente. Kontrolnim ulazom M bira se skup logičkih operacija ($M = 1$) ili skup aritmetičkih operacija ($M = 0$). Selekcionim signalima S_3, S_2, S_1 i S_0 bira se određena funkcija iz skupa funkcija pridruženog datoj vrijednosti M. Dakle, ukupno ima 5 kontrolnih pinova.

Što se tiče linija za podatke, po ulazima A_3 do A_0 i B_3 do B_0 zadaju se dva broja nad kojima će biti izvedena operacija, a rezultat operacije će biti saopšten preko izlaznih linija F_3 do F_0. U slučaju aritmetičkih funkcija, koriste se i signali C_{ul} i C_{iz} – ulazni i izlazni signal prenosa.

U tabeli 8.1 prikazane su sve logičke i aritmetičke funkcije koje komponenta obavlja, zavisno od kombinacije kontrolnih promjenljivih. U tabeli je $A = A_3A_2A_1A_0$ i $B = B_3B_2B_1B_0$. Oznaka $F = \overline{A}$ ima smisao $F_i \leftarrow \overline{A_i}$ za $i = 3, 2, 1, 0$, dok izraz $F = A \,\&\, B$ označava $F_i \leftarrow A_i \,\&\, B_i$ za $i = 3, 2, 1, 0$.

Selekcija				Logičke funkcije	Aritmetičke funkcije
S_3	S_2	S_1	S_0	$M = 1$	$M = 0$
0	0	0	0	$F = \overline{A}$	$F = A + C_{ul}$
0	0	0	1	$F = \overline{A \vee B}$	$F = (A \vee B) + C_{ul}$
0	0	1	0	$F = \overline{A} \,\&\, B$	$F = (A \vee \overline{B}) + C_{ul}$
0	0	1	1	$F = 0000$	$F = -\overline{C_{ul}}$
0	1	0	0	$F = \overline{A \,\&\, B}$	$F = A + (A \,\&\, \overline{B}) + C_{ul}$
0	1	0	1	$F = \overline{B}$	$F = (A \vee B) + (A \,\&\, \overline{B}) + C_{ul}$
0	1	1	0	$F = A \oplus B$	$F = A - B - \overline{C_{ul}}$
0	1	1	1	$F = A \,\&\, \overline{B}$	$F = (A \,\&\, \overline{B}) - \overline{C_{ul}}$
1	0	0	0	$F = \overline{A} \vee B$	$F = A + (A \,\&\, B) + C_{ul}$
1	0	0	1	$F = \overline{A \oplus B}$	$F = A + B + C_{ul}$
1	0	1	0	$F = B$	$F = (A \vee \overline{B}) + (A \,\&\, B) + C_{ul}$
1	0	1	1	$F = A \,\&\, B$	$F = (A \,\&\, B) - \overline{C_{ul}}$
1	1	0	0	$F = 1111$	$F = A + A + C_{ul}$
1	1	0	1	$F = A \vee \overline{B}$	$F = (A \vee B) + A + C_{ul}$
1	1	1	0	$F = A \vee B$	$F = (A \vee \overline{B}) + A + C_{ul}$
1	1	1	1	$F = A$	$F = A - \overline{C_{ul}}$

Tabela 8.1: Funkcije ALU SN74181

Slično važi i za ostale oznake.

Razmotrimo rad komponente na primjeru. Recimo, u desetom redu tabele imamo $S_3S_2S_1S_0 = 1001$. Neka je $M = 0$, tj. radi se o aritmetičkoj funkciji. U slučaju ove kombinacije upravljačkih signala, razmatrana komponenta obavlja funkciju sabiranja dva četvorobitna broja, uz uračunavanje u zbir i prenosa C_{ul} i uz još i saopštavanje prenosa C_{iz} koji nastaje na mjestu najveće težine (na bitu broj 3). Izraženo formulom,

$$F \leftarrow A + B + C_{ul} \text{ i } C_{iz} \leftarrow 1,$$

ako je došlo do prekoračenja prilikom sabiranja.

Na primjer, ako se zadaju ulazi $A_3A_2A_1A_0 = 0011$, $B_3B_2B_1B_0 = 0011$ i

$C_{ul} = 1$ onda će se obrazovati sljedeće izlazne vrijednosti: $F_3F_2F_1F_0 = 0111$ i $C_{iz} = 0$. Prevedeno u dekadni oblik: $3+3+1 = 7$. Vidimo da nema prenosa (prekoračenja), jer je saopšteno $C_{iz} = 0$. Važi ekvivalencija: $C_{iz} = 1$ (prenos se generiše) $\Leftrightarrow A + B + C_{ul} \geq 16$.

Za izlaznu liniju EQ važi: $EQ = 1$ ako i samo ako $A_0 = B_0$, $A_1 = B_1$, $A_2 = B_2$ i $A_3 = B_3$. O izlaznim linijama G i P biće riječi kasnije. Treba reći da komponenta ima još dva pina koji nisu prikazani na slici. To su V_{CC} preko koga se uvodi napon napajanja i GND - uzemljenje. Razmatrana komponenta ukupno ima tačno 24 pina.

Mreža za izvođenje aritmetičkih i logičkih operacija nad dva broja od po 16 bita dobija se jednostavnim kaskadnim povezivanjem četiri razmatrane komponente $K(1)$, $K(2)$, $K(3)$ i $K(4)$. Po četiri cifre ulaznih brojeva, koje su najmanje težine, uvode se u $K(1)$, sljedeće četiri u $K(2)$, ... , četiri cifre najveće težine jednog i drugog broja uvode se u $K(4)$. Izlaz C_{iz} iz $K(1)$ sprovodi se na ulaz C_{ul} komponente $K(2)$, i slično $C_{ul}(3) \leftarrow C_{iz}(2)$ i $C_{ul}(4) \leftarrow C_{iz}(3)$.

Razmotrimo jedan način da se aritmetička operacija, recimo sabiranje, nad dva 16-bitna broja ubrza, odnosno obavi za kraće vrijeme. Taj način predviđa izvjesnu hardversku nadogradnju. Naime, zapaža se da prenos za jednu komponentu $K(j)$ može da se izračuna brže nego rezultat $F = F_3F_2F_1F_0$ (rezultat je, recimo, zbir). Izlazi G i P služe da omoguće ubrzano (unaprijed) saznavanje i predaju prenosa, engl. fast look-ahead carry, prilikom kaskadnog vezivanja komponenti. Opišimo u kratkim crtama G i P, mada nećemo sastavljati njihove izraze. Važi da je $G = 1$ kada je $A+B \geq 16$ (prenos se generiše) a $P = 1$ kada je $A + B \geq 15$ (prenos se generiše ili ne generiše zavisno od C_{ul}). Recimo, ako je $A_3 = 1$, odnosno $A \geq 8$, i $B_3 = 1$, odnosno $B \geq 8$, slijedi da je $A + B \geq 16$, pa zaključujemo da je $P = G = 1$. Slično sa razmatraju ostali slučajevi.

Uzimajući u obzir ovaj princip, prilikom kaskadnog vezivanja četiri razmatrane komponente $SN74181$, u praksi se po pravilu još dodaje i jedna komponenta $SN74182$. Time se smanjuje ukupno kašnjenje. Pomoćna mreža $SN74182$ prostija je od mreže $SN74181$. Govoreći uprošćeno, ulazi u mrežu $SN74182$ su $C_{ul}(1)$, $G(1)$, $P(1)$, $G(2)$, $P(2)$, $G(3)$ i $P(3)$, a njeni izlazi sprovode se na $C_{ul}(2)$, $C_{ul}(3)$ i $C_{ul}(4)$. Ovdje je sa $C_{ul}(1)$ označen ulaz C_{ul} prve mreže $SN74181$, tj. mreže $K(1)$, itd.

8.2 Memorija

Memorije u digitalnim sistemima predstavljaju sklopove u koje se može upisati i iz kojih se može pročitati informacija. Zavisno od medijuma na kome se informacija pamti, najčešće se koriste poluprovodničke, magnetne i optičke memorije.

Memorije sa kojima raspolaže računarski sistem dijele se u dvije vrste:

- Memorija ili unutrašnja memorija ili glavna memorija ili radna memorija ili operativna memorija ili RAM.
- Spoljašnja memorija ili pomoćna memorija ili periferna memorija ili velika memorija ili jedinica diska ili skladište.

Memorijski uređaj koji neposredno komunicira sa CPU naziva se unutrašnjom memorijom. Svi ostali memorijski uređaji odgovaraju spoljašnjim memorijama. U unutrašnjoj memoriji nalaze se samo oni programi i podaci koje procesor u datom trenutku koristi, odnosno koji su potrebni programu čije izvršavanje je u toku. Sve ostale informacije smještene su u spoljašnjoj memoriji, odakle mogu da budu po zahtjevu prebačene u unutrašnju memoriju.

Magnetne i optičke memorije uglavnom se koriste za memorisanje velikog broja digitalnih informacija. Vrijeme upisivanja i čitanja informacija je relativno dugo, zbog neophodnih mehaničkih pomjeranja diska ili trake. Informacija u ovim memorijama ostaje zapamćena i kada je isključeno električno napajanje, tako da ove memorije spadaju u klasu *postojanih memorija*, engl. nonvolatile memory.

Unutrašnja memorija praktično uvijek je poluprovodnička. Poluprovodnička unutrašnja memorija u koju se može upisati ili pročitati informacija u proizvoljnom trenutku naziva se *RAM memorijom*, za razliku od *ROM memorije*, kod koje je, fizički i vremenski, proces upisivanja različit od procesa čitanja sadržaja. Naziv RAM dolazi od engleskog naziva random access memory - memorija sa slučajnim pristupom, što znači da je vrijeme za upisivanje ili čitanje nezavisno od adrese sa koje se čitanje ili upisivanje obavlja. Poluprovodničke RAM memorije po pravilu gube sadržaj kada se isključi napon napajanja, tako da spadaju u klasu nepostojanih memorija, engl. volatile memory.

Napomenimo da su se do početka sedamdesetih godina kao RAM memorije koristile i memorije od magnetnih jezgara. Danas se magnetne RAM memorije više ne koriste, zbog velike potrošnje i visoke cijene. Magnetne memorije su postojane memorije.

U skorije vrijeme svjedoci smo pojave efikasnih poluprovodničkih spoljašnjih memorija, koje su, dakle, postojane. One su još relativno spore, sporije su i od magnetnih memorija, ali može se očekivati da će se sve više poboljšavati.

Poluprovodničke memorije mogu biti *statičke* i *dinamičke*. Informacija upisana u statičkoj memoriji ostaje zapamćena sve dok je memorija priključena na napon napajanja. Da bi informacija ostala zapamćena u dinamičkoj memoriji neophodno je periodično obavljati "osvježavanje" memorije, inače se informacija gubi.

Ovdje ćemo razmotriti osnovne principe rada unutrašnje memorije, koju prosto nazivamo memorijom, dok ćemo spoljašnju memoriju obraditi u dijelu u kome se bavimo ulazno-izlaznim uređajima.

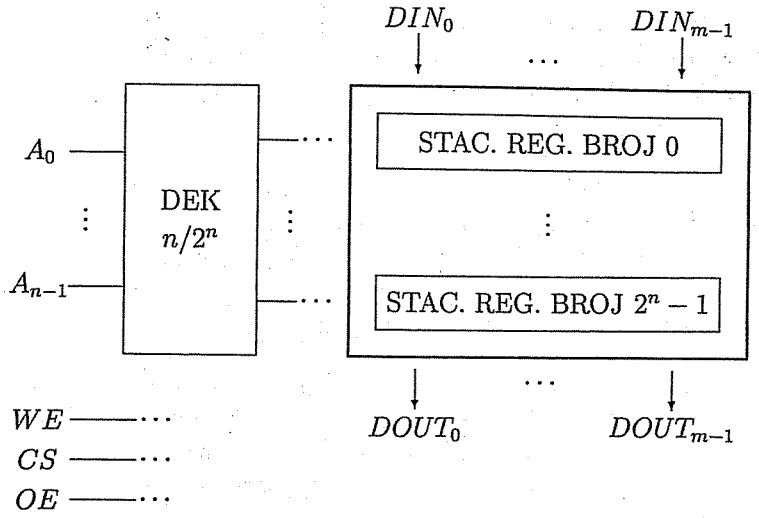
8.3 RAM memorije

8.3.1 Statičke poluprovodničke memorije

Statička RAM memorija (SRAM) predstavlja skup stacionarnih registara sa zajedničkim ulaznim i izlaznim priključcima. Selekcija registra u koji će se informacija upisati ili iz koga će se informacija pročitati obavlja se *adresnim dekoderom*. Logička šema statičke RAM memorije sa jednodimenzionim dekodiranjem prikazana je na slici 8.2.

Na slici 8.2 pojedini stacionarni registar sastoji se od m flipflopova vrste D sa ivičnim prebacivanjem. DIN znači data in, a $DOUT$ znači data out. DIN_0 do DIN_{m-1} su ulazne linije podataka, a $DOUT_0$ do $DOUT_{m-1}$ su izlazne linije podataka.

U svaki od 2^n stacionarnih registara može se upisati po jedna digitalna riječ od m bita. Adresnim ulazima $A_0, A_1, \ldots, A_{n-1}$ kodira se lokacija k–tog memorijskog registra. Time se k–ta izlazna linija dekodera (k–ta adresna linija) postavlja na logički nivo jedan. Postavljanjem $CS = 1$ selektuje se taj memorijski čip i onda se dovođenjem impulsa na kontrolni ulaz WE (write



Slika 8.2: Uprošćena logička šema statičke $2^n \times m$ RAM memorije

enable) u selektovani registar upisuje sadržaj prisutan na ulaznim linijama podataka $DIN_0, DIN_1, \ldots, DIN_{m-1}$. Sadržaj ostaje upisan (memorisan) sve dok se istim postupkom ne promijeni ili dok se ne isključi napon napajanja.

Čitanje upisane digitalne informacije obavlja se adresiranjem k–tog registra, čime se izlazne linije selektovanog registra priključuju na izlazne linije podataka. Postavljanjem $CS = 1$ i $OE = 1$, podaci upisani u k–tom memorijskom registru postaju pristupačni na izlaznim priključcima $DOUT_0, DOUT_1, \ldots, DOUT_{m-1}$.

U cilju smanjenja broja dekoderskih kola i u cilju formiranja kvadratne matrice memorijskih ćelija, RAM memorija se po pravilu izrađuje sa dvodimenzionim dekodiranjem. Govoreći uprošćeno, kod jednodimenzionog dekodiranja u jednoj vrsti memorijske matrice nalazi se jedan registar, tako da je m dužina vrste, a 2^n je broj vrsta. Kod dvodimenzionog dekodiranja, u jednoj vrsti nalazi se više registara. Napomenimo da iste okolnosti važe i za dekodiranje u slučaju ROM memorije.

Statičke memorije se u CMOS tehnologiji izrađuju sa raznim širinama, odnosno brojem bita, a vrijeme pristupa varira zavisno od tipa. Bipolarne memorije su po pravilu brže, ali manjeg kapaciteta.

8.3.2 Dinamičke poluprovodničke memorije

Svaka memorijska ćelija u statičkoj RAM memoriji sastoji se od najmanje četiri do šest tranzistora. Da bi se realizovala memorija sa većom gustinom pakovanja, konstruisana je memorija sa samo jednim tranzistorom i jednim kondenzatorom po memorijskoj ćeliji. Ovakva memorija bazira pamćenje informacije na električnom punjenju kondenzatora.

Da bi se postigla velika gustina pakovanja, kondenzator u memorijskoj ćeliji veoma je malih dimenzija, pa je i kapacitivnost kondenzatora veoma mala. Kada bi otpornost MOS tranzistora (za vrijeme dok je neprovodan) bila beskonačno velika, onda bi napon na kondenzatoru ostao nepromijenjen. Zbog konačne otpornosti neprovodnog MOS tranzistora, a i zbog male kapacitivnosti kondenzatora, zapamćeni napon na kondenzatoru (kada je zapamćena logička jedinica) postepeno opada i nakon nekoliko milisekundi zapamćena informacija bi se izgubila. Da se ovo ne bi dogodilo, svakih 2 do $4ms$ treba ponovo upisivati informaciju u memorijsku ćeliju. Repetitivni upis (ponovni upis) naziva se osvježavanjem sadržaja, a RAM memorija koja sadrži ćelije kojima je neophodno periodično osvježavanje naziva se *dinamičkom RAM memorijom* ili skraćeno DRAM.

U okviru integrisanog kola nalazi se i kontroler osvježavanja, tako da korisnik ne mora da vodi računa o redosljedu generisanja adresa za osvježavanje (za razliku od starijih tipova dinamičkih memorija). Da bi memorija bila raspoloživa za upisivanje i čitanje sadržaja, proces osvježavanja memorije treba da se obavi u što kraćem vremenu. U tom cilju, dinamičke memorije uvijek koriste dvodimenziono dekodiranje, a osvježavanje se obavlja istovremeno u svim ćelijama u jednoj vrsti. Takođe, u cilju smanjenja cijene memorijskog čipa, da bi broj spoljnih priključaka bio što manji, memorije se izrađuju sa zajedničkim (preklopljenim) adresnim ulazima za selekciju vrste i selekciju kolone. Po tim ulazima prvo se memoriji pošalje adresa vrste, a nakon toga se pošalje i adresa kolone.

8.3.3 Primjena RAM memorija

Poluprovodničke RAM memorije sastavni su dio svakog računarskog sistema. Zavisno od veličine računara, potrebni kapacitet RAM memorije kreće se od nekoliko desetina KB (kilo-bajta) za specijalizovane mikro-računarske sisteme pa do više GB za veće računare. Kapacitet RAM memorije personalnih računara je nekoliko stotina MB. Često je za realizaciju RAM memorije računarskog sistema neophodno koristiti više čipova.

Statičke memorije su manjeg kapaciteta po čipu, a koriste se u sistemima gdje se zahtijeva veća brzina pristupa memoriji i manja potrošnja struje iz izvora za napajanje. Takođe je vjerovatnoća greške kod statičkih memorija manja nego kod dinamičkih, tako da se koriste u sistemima gdje se zahtijeva veća pouzdanost.

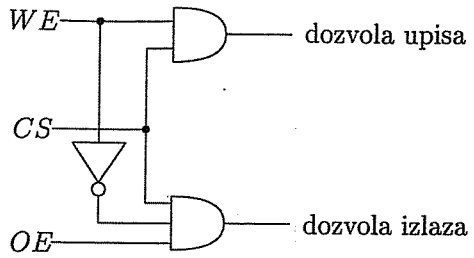
Dinamičke memorije, zbog daleko veće gustine pakovanja, zahtijevaju manji broj integrisanih kola nego statičke, za isti kapacitet. Potrošnja struje dinamičkih memorija je znatno veća nego statičkih, a prilikom svake selekcije vrste struja napajanja se impulsno poveća za nekoliko desetina mA po čipu. Ove impulsne promjene struje mogu da generišu električne smetnje, tako da prilikom korišćenja DRAM memorija treba posebno voditi računa o filtraciji napona napajanja. Takođe, dinamička memorija zahtijeva periodični impuls za osvježavanje svakih nekoliko ms.

I pored ovih nedostataka, kada je potreban veći kapacitet, DRAM memorije se češće koriste, s obzirom da imaju veću gustinu pakovanja a time zauzimaju manje prostora na štampanoj ploči. Zbog manjeg broja čipova i jednostavnije štampane ploče, cijena DRAM memorije obično je niža od SRAM memorije istog kapaciteta.

8.3.4 Organizacija statičke memorije

U računarskim sistemima razmjena informacija sa memorijom uglavnom se obavlja preko sistemske magistrale koja sadrži skup adresnih linija, skup linija podataka i skup kontrolnih linija. Na slici 8.3 prikazana je logička šema kontrolne mreže memorijskog čipa.

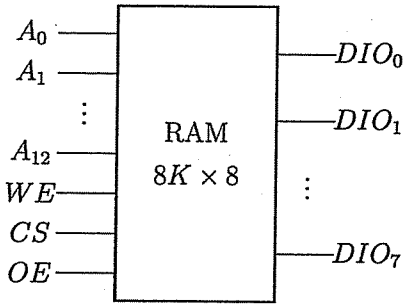
Na slici 8.3 prisutni su signali WE (engl. write enable) i OE (engl. output enable). Obezbijeđen je i ulaz za selekciju čipa CS (engl. chip select). Treba reći da su u praksi po pravilu prisutna dva ulaza za selekciju čipa, od kojih je jedan aktivan na visokom logičkom nivou a drugi na nultom nivou; čip



Slika 8.3: Logička šema kontrolne mreže

je selektovan ako su oba aktivna. Osim toga, u praksi su dva signala OE i WE najčešće aktivni na niskom logičkom nivou, zato što su na magistralama kontrolni signali po pravilu aktivni na niskom nivou.

Pošto se upis u memoriju i čitanje iz memorije nikad ne obavlja istovremeno, izrađeni su zajednički bidirekcioni priključci za podatke.



Slika 8.4: Simbol za statičku $8K \times 8$ RAM memoriju

Na slici 8.4 prikazan je simbol statičke RAM memorije oblka $8K \times 8$. Na slici, A znači address (adresa), a DIO znači data input output (ulaz izlaz podataka). Vidimo da je $2^n = 2^{13} = 8K$ i da je $m = 8$.

Memorija funkcioniše na sljedeći način: ako je aktivan signal WE, tada

$$(DIO_0, \ldots, DIO_{m-1}) \rightarrow \text{stacionarni registar broj } k,$$

a ako je aktivan signal OE, tada

$$\text{stacionarni registar broj } k \rightarrow (DIO_0, \ldots, DIO_{m-1}),$$

pri čemu je $0 \leq k \leq 2^n - 1$, broj k zadaje se od strane $A_0, A_1, \ldots, A_{n-1}$.

Vidimo da memorija ima $2^{13} \times 2^3 = 2^{16}$ bita. Ako se vrši dvodimenziono dekodiranje, tj. ako se želi napraviti kvadratna matrica memorijskih ćelija onda očito treba da bude 2^8 vrsta, i po 2^8 ćelija u jednoj vrsti.

Ako je potrebno na primjer da se realizuje memorija od 64KB ($64K \times 8$) a na raspolaganju su memorijski čipovi kapaciteta $8K \times 8$, onda je za formiranje takve memorije potrebno 8 čipova.

8.4 ROM memorije

8.4.1 Pojam ROM memorije

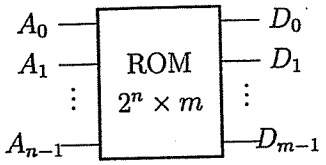
Memorija sa konstantnim sadržajem je integrisano kolo u koje se posebnim postupkom upisuje željeni sadržaj, a kada je sadržaj upisan onda memorija može samo da se čita. Takva memorija naziva se engl. read only memory odnosno ROM memorijom.

Na slici 8.5 prikazan je logički simbol ROM memorije oblika $2^n \times m$. To znači da sadrži 2^n riječi (memorijskih riječi) i da pojedina riječ ima dužinu m bita. Ulazni signali obilježeni su kao $A_0, A_1, \ldots, A_{n-1}$ i nazivaju se *adresnim ulazima.* Izlazni signali obilježeni su kao $D_0, D_1, \ldots, D_{m-1}$ i nazivaju se *izlazima podataka.*

Veličine A_i zadaju broj k u granicama $0 \leq k \leq 2^n - 1$ na način koji je za dekoder uobičajen. Po izlaznim linijama podataka D_0 do D_{m-1} saopštava se sadržaj k–te memorijske riječi.

8.4.2 ROM memorija sa fiksiranim sadržajem (mask-ROM)

Logička šema integrisane ROM memorije prikazana je na slici 8.6. Memorija je realizovana pomoću dioda, a koristi se i jedan dekoder. Ako je na jednom



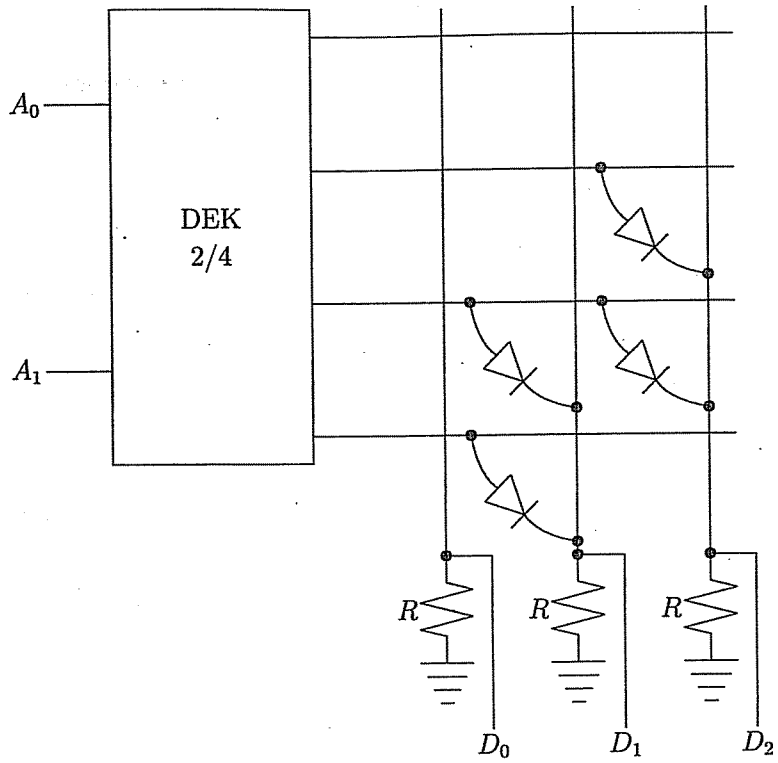
Slika 8.5: Simbol ROM memorije

mjestu dioda prisutna onda se na tom mjestu (na tom bitu) pamti logička jedinica (logički nivo je visok), a ako je dioda izostavljena onda je na tom mjestu upisana logička nula.

Da bi slika bila preglednija, uzeto je $n = 2$ i $m = 3$. Dakle, kapacitet memorije je 4 riječi, a pojedina riječ je trobitna. Označimo izlaze dekodera kao W_0, W_1, W_2 i W_3 (izlazne adresne linije). Ako je $(A_0, A_1) = (0, 0)$ onda je $W_0 = 1$, a ostali izlazi su $W_j = 0$. Ako je $(A_0, A_1) = (0, 1)$ onda je $W_1 = 1$, i sl. Dakle, dekoder selektuje jednu od 4 horizontalne linije.

Vidi se da na slici učestvuju četiri diode od dvanaest mogućih, tako da se pamte četiri jedinice. Prva memorijska riječ $(W_0 = 1)$ glasi $(D_0, D_1, D_2) = (0, 0, 0)$, jer na prvoj adresnoj liniji W_0 nije priključena nijedna dioda. Za $W_1 = 1$ izlazni kod je 001. Za $W_2 = 1$ izlazni kod ROM-a će biti 011. I četvrto, ako je $W_3 = 1$ onda se saopštava trobitna riječ 010.

Umjesto dioda, za definisanje sadržaja ROM memorije mogu se koristiti tranzistori. Programiranje opisanih ROM memorija obavlja se prilikom izrade integrisanog kola. Korisnik od proizvođača naruči sadržaj memorije, proizvođač na osnovu zahtijevanog sadržaja formira masku za izradu integrisanog kola sa priključenim diodama na mjestima gdje sadržaj memorije treba da bude 1 ili masku sa priključenim tranzistorima na mjestima gdje sadržaj treba da bude 0. Zbog načina programiranja (načina upisivanja sadržaja) ove memorije se nazivaju mask-ROM memorijama. Izrada posebne maske za svaki različit sadržaj ROM memorije je skup proces u odnosu na cijenu koštanja integrisanog memorijskog kola ROM memorije (nekoliko dolara). Zato se mask-ROM koristi tamo gdje postoji potreba za velikim brojem takvih memorija.

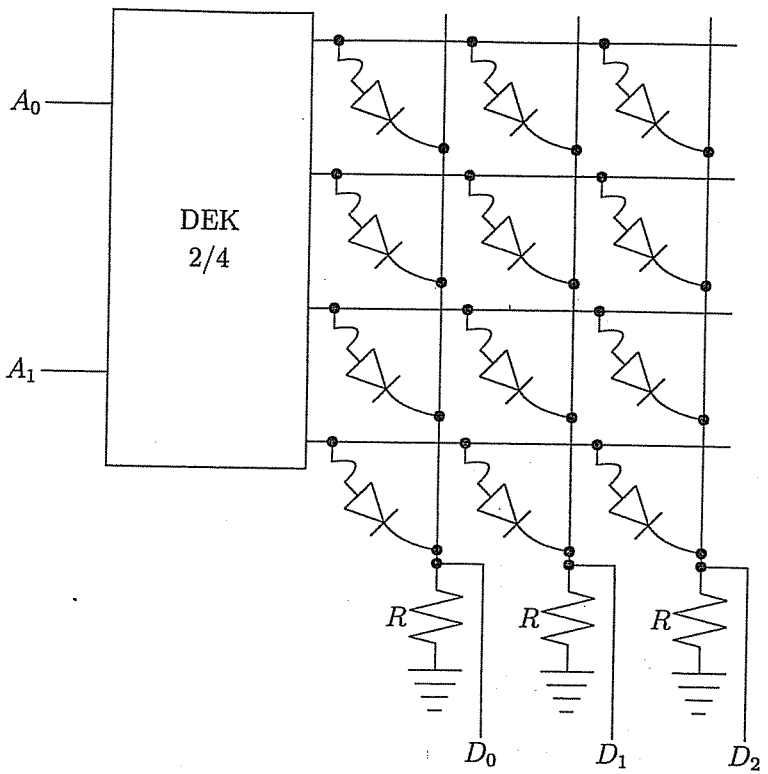


Slika 8.6: Diodna ROM memorija

8.4.3 Programabilna ROM memorija (PROM)

Za digitalne uređaje gdje nisu potrebne velike serije ROM memorija sa jednakim sadržajem koristi se programabilna ROM memorija ili skraćeno PROM memorija. Diodne PROM memorije se proizvode sa ugrađenim svim diodama a redno sa svakom diodom ugrađen je topljivi osigurač, kao što je prikazano na slici 8.7.

Kada memorija nije programirana, s obzirom da su sve diode priključene, na svim adresama memorije su "upisane" sve jedinice. Korisnik sam programira memoriju time što izazove pregorijevanje osigurača na mjestima gdje



Slika 8.7: Programabilna diodna ROM memorija

želi da sadržaj memorije bude 0.

Pregorijevanje osigurača izaziva se tako što se adresira riječ po riječ, a na linije podataka (gdje treba odstraniti diodu) dovodi se negativni impuls. Za vrijeme trajanja impulsa teći će struja kroz sve diode na toj liniji podataka. Međutim, kroz diodu koja je priključena na adresiranu adresnu liniju struja će biti većeg intenziteta, s obzirom da je anoda adresirane diode na visokom logičkom odnosno naponskom nivou, dok su anode svih ostalih dioda na naponu logičke nule. Struja kroz adresiranu diodu izaziva pregorijevanje osigurača, čime će na izabranoj adresi i izabranoj poziciji biti upisana logička nula.

Za unikatne ili manje serije digitalnih uređaja, programabilne ROM memorije su daleko ekonomičnije od mask-ROM memorija. Programiranje se obavlja korišćenjem specijalnog uređaja - *PROM programatora*. U PROM programator se pomoću računara upiše željeni sadržaj memorije. Programator sukcesivno adresira sve adrese priključene PROM memorije i dovodi odgovarajuće impulse na linije podataka na mjestima gdje treba promijeniti inicijalni sadržaj.

Glavni nedostatak PROM memorije je što se jedanput upisani sadržaj ne može promijeniti. Potreba za promjenom sadržaja veoma je česta, naročito prilikom razvoja novih uređaja ili promjene načina funkcionisanja već razvijenih. Za takve primjene, pogodnije je da se koriste programabilne ROM memorije sa mogućnošću brisanja (EPROM memorije).

8.4.4 Programabilna ROM memorija sa mogućnošću brisanja (EPROM)

EPROM memorije (engl. erasable programmable read only memory) kao memorijske elemente koriste MOS tranzistore sa izolovanim gejtom. Svaki tranzistor u memorijskoj matrici ima dva gejta. Izolovani gejt okružen je praktično idealnim izolacionim materijalom (SiO_2) i sa neizolovanim gejtom predstavlja kapacitivni razdjelnik napona. Kada EPROM nije programiran, napon logičke jedinice na adresnoj liniji dovoljan je da preko kapacitivnog razdjelnika formira kanal MOS tranzistora, tako da je sadržaj svih lokacija u memoriji nula.

Da bi se na određenoj lokaciji upisala logička jedinica, željena linija podataka i adresna linija priključuju se na visok napon (oko $25V$). U tako selektovanom MOS tranzistoru dolazi do relativno velike struje drejna, elektroni dobijaju veliko ubrzanje, tako da dio elektrona nedestruktivno probija izolaciju i akumulira se na izolovanom gejtu. Kada se isključi visoki napon, izolovani gejt ostaje negativno naelektrisan (na oko $-5V$), tako da pri radnom naponu (tipično $5V$) napon logičke jedinice na neizolovanom gejtu nije dovoljan da formira kanal. I kada je adresiran, MOS tranzistor ostaje neprovodan, odnosno na tom mjestu je upisana logička jedinica, ekvivalentno pregorijevanju osigurača kod PROM memorije. Kvalitet izolacije izolovanog gejta obezbjeđuje da elektronski tovar ostaje na gejtu više od deset godina, čak i kada se EPROM nalazi na temperaturi od $125°C$. Međutim, ako se izolacioni materijal izloži ultraljubičastoj svjetlosti onda SiO_2 postaje slabo

provodan i elektroni napuštaju izolovani gejt, odnosno sadržaj svih lokacija ponovo postaje nula. Iz tog razloga, EPROM memorije se izrađuju u kućištima sa providnim prozorom od kvarcnog stakla, kako bi mogle da bude izložene dejstvu ultraljubičastog svijetla u cilju brisanja.

8.4.5 EEPROM

Programabilna ROM memorija sa mogućnošću elektronskog brisanja (engl. electrically erasable programmable read only memory, *EEPROM*) takođe kao memorijske ćelije koristi MOS tranzistore sa izolovanim gejtom.

Šema memorije ista je kao za EPROM, s tim da je izolacija između izolovanog gejta i kanala svedena na svega $100nm$. Upis logičke jedinice u ćeliju EEPROM–a obavlja se slično kao kod EPROM–a, samo što je sada napon proboja izolacije sveden na oko $10V$. Brisanje EEPROM–a obavlja se električno, tako što se na gejt priključuje napon suprotnog polariteta od napona za upis.

Osnovna razlika između EPROM-a i EEPROM-a sastoji se u tome što se prilikom brisanja EPROM-a briše kompletan sadržaj cijele memorije, brisanje traje 10 do 20 minuta i to u komori sa ultraljubičastom svjetlošću. Brisanje EEPROM memorije obavlja se selektivno, samo adresirana riječ, priključivanjem negativnog napona na selektovane memorijske ćelije za vrijeme od oko $10ms$ po riječi.

Savremene EEPROM memorije imaju u istom integrisanom kolu ugrađen pretvarač $5V$ u $\pm 10V$, tako da se upis i brisanje EEPROM-a može obavljati bez isključivanja digitalnog uređaja u kome je ugrađena EEPROM komponenta. Dovoljno je dovesti odgovarajući kontrolni signal na priključak za upis odnosno za brisanje.

8.5 Dvodimenziono dekodiranje adresa memorije

Znamo da izraz ROM memorija oblika $2^n \times m$ znači da memorija (ili memorijska matrica) ima n adresnih ulaza i onda, naravno, da sadrži 2^n podataka (memorijskih riječi). Veličina pojedinog podatka jednaka je m bita.

Vidjeli smo da je šema ROM memorije malog kapaciteta oblika dekoder – matrica. Dakle, svi adresni ulazi usmjeravaju se pravo na dekoder, zato kažemo da se radi o jednodimenzionom dekodiranju.

Postoji i dvodimenziono dekodiranje memorije, tj. dvodimenzioni način da se zada koji podatak treba da bude pročitan. Kada se radi o memoriji većeg kapaciteta onda je pogodno da prilikom izbora podatka koji se čita dekoderu pomaže multiplekser (ili multiplekseri). Moguće je da se jedan dio adresnih linija usmjerava prema dekoderu, a drugi dio prema multiplekseru. Tako da sada šema postaje: dekoder – matrica – multiplekser. Multipleksera ima m, tj. broj multipleksera u mreži zavisi od veličine podatka.

Istaknimo da važe iste okolnosti u pogledu mogućnosti jednodimenzionog ili dvodimenzionog dekodiranja adrese podatka i za RAM i za ROM memorije.

Dekodiranje memorijskih adresa na jednodimenzioni način primjenjuje se samo za memorije manjeg kapaciteta. Realizacija memorije većeg kapaciteta na takav način zahtijevala bi veliki broj logičkih elemenata (za dekoder), a i sama memorijska matrica bila bi nepogodna za izradu u integrisanoj tehnologiji, s obzirom da bi matrica zauzimala prostor veoma izduženog pravougaonika sa velikim brojem vrsta (redova) i malim brojem kolona. Na primjer, memorijska matrica ROM memorije oblika $1K \times 4$ imala bi 1024 reda i samo 4 kolone. Izrada integrisanih kola najekonomičnija je kada se postiže najveća gustina pakovanja, a to je kada su komponente fizički raspoređene unutar kvadrata. Iz tog razloga, za izradu memorije većeg kapaciteta primjenjuje se dvodimenziono dekodiranje, tako da se smanjuje broj logičkih elemenata i istovremeno se fizički raspored komponenata memorije približava kvadratnom obliku.

Princip dvodimenzionog dekodiranja opišimo na primjeru sinteze ROM memorije 32×2. U memorijskoj matrici upisane su 64 binarne vrijednosti, organizovane kao 32 dvobitne riječi. Pomoću ulaza A_0 do A_4 definiše se jedan broj k u granicama $0 \leq k \leq 31$. Na izlazu iz mreže (izlazni priključci D_0 i D_1) saopštava se memorijska riječ čiji je redni broj k.

Kada bi se koristilo jednodimenziono dekodiranje, memorija bi sadržala dekoder 5/32 i memorijsku matricu sa 32 reda i dvije kolone. Dvodimenzionim dekodiranjem ova memorija se realizuje korišćenjem jednog dekodera 3/8, dva multipleksera 4/1 i kvadratne matrice memorijskih ćelija sa 8 redova i 8 kolona.

Tri od ukupno pet adresnih linija, recimo A_0, A_1 i A_2, uvode se u 3/8 dekoder koji selektuje osam adresnih linija, odnosno bira jedan red matrice. Na svakoj adresnoj liniji nalaze se memorijske ćelije četiri susjedne dvobitne memorijske riječi. Raspoređene su tako da prvo idu prvi bitovi tih riječi, a zatim drugi. Recimo, ako je sadržaj prve četiri (od 32) memorijske riječi 01, 11, 01, 00, tada je u prvom redu (diodama) zapisano 01001110.

U mreži su prisutna dva multipleksera. Preostali adresni ulazni signali (A_3 i A_4) sprovode se na ulaze za selekciju jednog i drugog multipleksera. Izlazi multipleksera su izlazni priključci memorijske mreže, D_0 i D_1.

Memorijska matrica ima svojih osam izlaznih linija podataka (vertikalne linije), s tim da se prve četiri linije uvode na ulaze za podatke u prvi multiplekser, a druge četiri linije u drugi multiplekser. Znači, u prvi multiplekser uvode se linije koje predstavljaju prve bite podataka u jednom redu. Slično, u drugi multiplekser se uvode linije koje predstavljaju druge bite podataka u jednom redu.

Zadatak multipleksera je da, zavisno od kontrolnih bita, na izlaz propusti jedan od četiri moguća bita koji stižu kroz vertikalne linije iz dekoderom selektovanog reda. Prvi multiplekser na izlazu daje prvi bit, a drugi multiplekser - drugi bit zapisanog podataka.

Dakle, svaka kombinacija ulaznih signala A_0, A_1 i A_2 bira tačno jedan red matice, a svaka kombinacija ulaznih signala A_3 i A_4 propušta na izlaze multipleksera jedan od četiri para bita u tom redu.

Kao još jedan primjer, razmotrimo ROM memoriju 256×1 sa dvodimenzionim dekodiranjem. Tada imamo sljedeću šemu: dekoder 4/16 – memorijska matrica sa 16 redova i 16 kolona – jedan multiplekser 16/1.

8.6 Organizacija centralnog procesora

Centralni procesor najvažniji je dio računarskog sistema. On kontroliše rad cijelog sistema. Ova komponenta računarskog sistema tumači program i izdaje naredbe kako da se njegove naredbe ostvare. Zadatak procesora je da organizuje i koordinira djelatnost svih ostalih jedinica u digitalnom sistemu. Često se procesor posmatra kroz dvije cjeline: upravljački dio procesora, *kontrolna jedinica* (CU, control unit) i aritmetičko-logička jedinica.

Digitalni sistem procesora sastavljen je od hardverskih komponenti čiji je zadatak da obavljaju određene obrade nad podacima. Primjeri komponenti su registri, dekoderi, aritmetička kola i upravljačka logička kola.

Važni primjeri registara koji se često sreću u procesorima računara su

- akumulator (Accumulator, AC),
- adresni registar (Memory Address Register, MAR),
- prihvatni memorijski registar (Memory Buffer Register, MBR).

Akumulator se koristi prilikom izvođenja aritmetičkih i logičkih operacija. Adresni registar čuva adresu memorijske lokacije nad kojom se radnja vrši. Pomoću memorijske magistrale, MAR je povezan sa memorijskom jedinicom. Prihvatni memorijski registar služi da prihvati sadržaj memorijske lokacije nad kojom se vrši operacija, koja može biti čitanje lokacije ili upis u lokaciju. Pomoću memorijske magistrale, MBR je povezan sa memorijskom jedinicom.

Procesor izvršava naredbe zapisane u memoriji, na taj način što ih prvo dovlači u svoje registre, zatim analizira i izvršava. Dakle, izvršavanje jedne naredbe sastoji se od nekoliko manjih koraka, koje nazivamo *mikro-operacijama.* Detaljan opis rada jednog konkretnog procesora dajemo u sljedećem poglavlju, a ovdje ćemo definisati pojam mikro-operacije, i razmotriti njihove vrste.

8.6.1 Pojam mikro–operacije i registarskog prenosa

Za radnju koja se nad podacima upisanim u jednom ili u više registara obavi tokom jednog otkucaja časovnika (tokom jednog takta) kaže se da predstavlja jednu *mikro–operaciju.* Primjeri mikro–operacija su pomjeranje, brojanje, brisanje i upisivanje. Za opis organizacije digitalnog računara potrebno je opisati:

- skup registara koje računar sadrži i moguće radnje koje se odnose na registre.
- niz mikro–operacija koje se obavljaju nad podacima sadržanim u registrima.
- upravljačke radnje, tj. kontrolni signali koji naređuju da počne niz mikro–operacija.

Generalno, postoje četiri grupe mikro–operacija: 1) one koje služe da se ostvari prenos između registara; 2) aritmetičke; 3) logičke i 4) za pomjeranje. U nastavku će biti riječi posebno o svakoj od četiri vrste.

Znamo da se taktovanje u digitalnom sistemu zadaje od strane glavnog generatora otkucaja, čiji taktni impulsi se primjenjuju na sve flipflopove u sistemu (na sve flipflopove koji rade sinhrono). Primjenjivanje taktnog impulsa samo po sebi ne izaziva promjenu sadržaja registra - potrebno je da bude primijenjen i odgovarajući upravljački signal.

8.6.2 Mikro–operacije za prenos između registara

Kod ove grupe mikro–operacija podaci iz jednog registra prenose se u drugi registar (kažemo da se podaci prenose iz izvora na odredište). Postoji nekoliko oblika prenosa između registara.

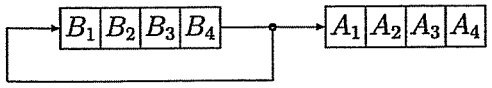
U slučaju *paralelnog prenosa*, svi bitovi izvora jednovremeno se prenose na odredište. Dakle, ova radnja se obavlja tokom jednog otkucaja časovnika i stoga ona predstavlja jednu mikro–operaciju. Ta mikro–operacija se označava kao

$$A \leftarrow B.$$

Naravno, ovdje se pretpostavlja da postoji mreža koja sa izlaza flipflopova registra B vodi prema ulazima registra A. Po pravilu, mi želimo da se prenos obavi samo ako je ispunjen neki unaprijed zadati uslov. *Uslov* ili *upravljačka radnja* može da bude označen kao P. Tada ovu operaciju možemo označiti sa

$$P: A \leftarrow B.$$

Prepisivanje sadržaja procesorovog registra B u procesorov registar A može da bude ostvareno na paralelni način ili na serijski način.



Slika 8.8: Serijski prenos između registara

Pogledajmo slučaj *serijskog prenosa*. Neka su A i B pomjerački registri i neka i jedan i drugi imaju po četiri flipflopa (i jedan i drugi su veličine četiri

bita), $A = A_1A_2A_3A_4$ i $B = B_1B_2B_3B_4$. Radnja prenosa sadržaja može da bude ostvarena pomoću sljedećeg niza mikro–operacija:

$$(A_4, A_3, A_2, A_1) \leftarrow (A_3, A_2, A_1, B_4),$$

$$(B_4, B_3, B_2, B_1) \leftarrow (B_3, B_2, B_1, B_4),$$

gdje navedeni niz treba da bude ponovljen četiri puta. Drugim riječima, izvršavanje radnje $A \leftarrow B$ traje četiri takta (v. sliku 8.8). Vidimo da se u registru B odvija kružno pomjeranje udesno, dok se u registru A odvija pomjeranje udesno. Na kraju će se sadržaj registra B vratiti na početnu vrijednost.

U slučaju *prenosa na magistralu* izvrši se radnja recimo BUS $\leftarrow$ C ili R1 $\leftarrow$ BUS. Ovdje su C i R1 registri u procesoru, a BUS je registar na magistrali.

Razmotrimo *prenos koji se odnosi na memoriju*. Prenošenje podataka iz memorije njenom okruženju naziva se *operacijom čitanja*. Prenošenje nove informacije memoriji naziva se *operacijom upisivanja*. Za jednu i drugu operaciju, treba da bude naznačena odgovarajuća memorijska riječ M, odnosno *adresa* riječi M.

Ako se u izjavi o prenosu prosto pojavljuje samo slovo M, onda se time kaže da je adresa izabrane memorijske lokacije jednaka tekućem sadržaju registra MAR. Dakle, mikro–operacija čitanja predstavlja prenošenje iz izabrane memorijske riječi M u registar MBR. Označava se kao

$$MBR \leftarrow M$$

ili kao

$$MBR \leftarrow M[MAR].$$

Mikro–operacija upisivanja predstavlja prenošenje iz MBR u zadatu riječ M. Bilježi se kao

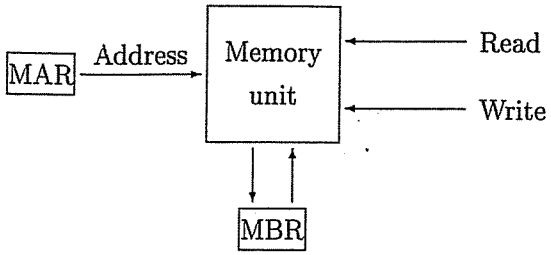
$$M \leftarrow MBR$$

odnosno

$$M[MAR] \leftarrow MBR.$$

Na slici 8.9 prikazana je šema komunikacije sa memorijom. I read i write mikro–operacija obaviće se pod uslovom prisustva odgovarajućeg upravljačkog signala.

Ukupno smo za ovu grupu mikro-operacija nabrojali sljedeće mogućnosti: paralelno, serijski, sa magistralom i sa memorijom.



Slika 8.9: Komunikacija sa memorijom

8.6.3 Aritmetičke mikro–operacije

Kao što im i ime kaže, aritmetičke mikro–operacije obavljaju aritmetičke operacije nad registrima.

Primjer aritmetičke mikro–operacije jeste mikro–operacija sabiranja

$$R2 \leftarrow R1 + R2,$$

gdje su R1 i R2 dva registra unutar procesora. Ovdje se podrazumijeva postojanje logičkih kola koja su u stanju da obave aritmetičku radnju sabiranja.

Sličan primjer aritmetičke mikro–operacije je

$$P: A \leftarrow A + B,$$

a njen smisao je: sabiranje će se izvršiti samo ako je $P = 1$.

Primjer aritmetičke mikro–operacije je i

$$EA \leftarrow A + B,$$

gdje su A i B registri, a E je flipflop. Flipflop E namijenjen je da čuva prenos koji prilikom sabiranja nastaje na mjestu najveće težine. Ova mikro–operacija ima smisao sabiranja sa pamćenjem posljednjeg prenosa.

Navedimo i druge obične primjere aritmetičkih mikro–operacija. Radnja brojanja ili inkrementiranja (povećavanja) za jedan sadržaja registra A zapisuje se kao

$$A \leftarrow A + 1.$$

Odgovarajuća hardverska komponenta za tu operaciju jeste očito brojač.

Radnja komplementiranja bit–po–bit sadržaja registra A zapisuje se kao

$$A \leftarrow \overline{A}.$$

Na primjer, ako je bilo $A = 0010$ onda će se dobiti $A = 1101$. Ako je registar A sastavljen od JK flipflopova, mikro–operacija komplementiranja može da se ostvari dovođenjem upravljačkog signala P na sve J i K ulaze flipflopova.

Operacija

$$A \leftarrow \overline{A} + 1$$

odgovara promjeni predznaka broja, ako se sadržaj registra tumači kao zapis broja u obliku potpunog komplementa (u obliku PK). Na primjer, ako je staro stanje bilo $A = 0010$ nakon izvršenja mikro–operacije $A \leftarrow \overline{A} + 1$ će biti $A = 1110$. Prevedeno na dekadni sistem, od $A = 2$ postaće $A = -2$.

Mikro–operacija brisanja

$$P: A \leftarrow 0$$

može da se ostvari za registar A sastavljen od JK flipflopova, tako što će na sve K ulaze flipflopova biti direktno primijenjena upravljačka radnja P.

8.6.4 Logičke mikro–operacije

Logičke mikro–operacije izvode logičke operacije na registrima, i to bit-po-bit.

Primjer logičke mikro–operacije jeste

$$A \leftarrow A \oplus B.$$

Ako se radi o registrima od po četiri bita tj. $A = A_1A_2A_3A_4$ i $B = B_1B_2B_3B_4$ onda navedena radnja ima smisao $A_i \leftarrow A_i \oplus B_i$ za $i = 1, 2, 3, 4$.

U ovu grupu mikro–operacija spadaju sve Bulove funkcije, tipično sa dva argumenta. Umjesto $p \vee q$ ponekad se piše $p + q$ (disjunkcija, logičko sabiranje). Takođe, umjesto $\overline{p}$ ponekad se piše p' (negacija).

8.6.5 Pomjeračke mikro–operacije

Ova grupa mikro–operacija koristi brojne vrste pomjeračkih registara. Navedimo samo četiri osnovne operacije pomjeranja (šiftovanja): pomjeranje ulijevo, pomjeranje udesno, kružno pomjeranje ulijevo i kružno pomjeranje udesno. Ako se radi o četvorobitnom registru $A = A_1A_2A_3A_4$ onda formule za te četiri osnovne operacije glase redom:

$$A_1A_2A_3A_4 \leftarrow A_2A_3A_4 0,$$

$$A_1A_2A_3A_4 \leftarrow 0A_1A_2A_3,$$

$$A_1A_2A_3A_4 \leftarrow A_2A_3A_4A_1,$$

$$A_1A_2A_3A_4 \leftarrow A_4A_1A_2A_3.$$

8.7 O registrima i izvođenju operacija

Znamo da više povezanih flipflopova čini registar. Ako se registar nalazi u CPU onda se naravno kaže da je to jedan procesorov registar. Ako se registar nalazi u memoriji, umjesto *memorijski registar*, češće se kaže *memorijska riječ*. Znamo da memorijska riječ ima svoju adresu, tj. da su memorijske riječi numerisane.

Računari čiji CPU ima mali broj registara koriste jedan akumulator za izvođenje aritmetičkih i sličnih radnji. Sa pojavom integrisanih kola veće složenosti (većeg stepena integracije) povećao se i broj registara CPU. Drugim riječima, napredak tehnologije omogućio je da jedno integrisano kolo može da sadrži veliki broj komponenti, tako da umjesto jednog akumulatora ima više njih, recimo 4 ili 16.

Bilo bi korisno da se rezultat neke operacije (recimo neke aritmetičke operacije) ne šalje u memoriju, pa zatim poslije kratkog vremena čita iz memorije. Drugim riječima, pogodno je da neki međurezultati ostanu u CPU. Isto tako, pogodno je da u CPU ostanu brojači, indeksi i povratne adrese.

Jasno je da je organizacija CPU koji ima znatan broj registara složenija od organizacije računara koji ih ima nekoliko. Skup naredbi, odnosno *mašinski jezik* takvog računara složeniji su od mašinskog jezika računara sa par registara. Takođe je i prevodilac za, recimo, Paskal takvog računara složeniji. Naravno, jezik Paskal je jedan te isti u jednom i drugom slučaju.

Mogu da budu razvijeni elementi paralelizma u radu CPU. Na primjer, dok ALU izvršava jednu naredbu, iz memorije može da se čita sljedeća naredba.

Ako procesor ima više registara, mogući su razni metodi za izvršavanje naredbi. Razmotrimo, na primjer, aritmetičko–logičku jedinicu (ALU) koja ima nekoliko akumulatora. Vidjeli smo da aritmetičko–logička jedinica predstavlja dio CPU koji služi za izvođenje atirmetičkih i logičkih operacija.

Tada je moguća naredba oblika R3 $\leftarrow$ R1 + R2, gdje su R1, R2 i R3 – neka tri akumulatora.

Postavlja se pitanje, kako se vrši množenje dva broja? Prva mogućnost je da se to množenje izvodi pomoću jedne velike kombinacione mreže, što može biti jako teško za realizaciju i skupo. Druga mogućnost je da se to izvede preko šiftovanja i sabiranja, uz upotrebu memorijskih komponenti (registara).

Razmotrimo detaljnije drugu mogućnost. Da bi čitavo množenje moglo da se obavi unutar procesora tj. bez obraćanja memoriji, moramo imati na raspolaganju četiri–pet registara. Tokom izvršavanja množenja, svaki od tih registara pamti jedan broj (jedan međurezultat), kao što su: prvi množitelj, drugi množitelj, šiftovani oblik drugog množitelja i djelimični proizvod; na kraju će djelimični proizvod postati jednak proizvodu (rezultatu operacije množenja). Eventualno je potreban i jedan brojač (vidi primjer 1 niže).

U savremenim računarima cijeli brojevi se praktično uvijek zapisuju u obliku potpunog komplementa (PK), a realni brojevi u obliku pokretnog zareza. Osnovne aritmetičke operacije zasnovane su na sabiranju. Sabiranje se izvodi korišćenjem sabirača, a slično i oduzimanje. Množenje se realizuje preko šiftovanja i sabiranja. Slično se izvodi i dijeljenje.

Navešćemo par primjera koji ilustruju postupak za sabiranje odnosno množenje.

Primjer 1. Neka je $a, b, c, d, e, f \in \{0, 1\}$. Dakle, imamo binarni broj $abcdef$. Množenje ovog broja sa brojem 1101 izvodi se na sljedeći način:

$$abcdef \cdot 1101 = abcdef000 + abcdef00 + abcdef = \ldots$$

Međurezultati $abcdef000$ i $abcdef00$ dobijeni su šiftovanjem. Treba još izvršiti dva naznačena sabiranja.

Primjer 2. Znamo da se u pokretnom zarezu broj x prikazuje preko para brojeva m i e, po propisu $x = m \cdot 16^e$, $\frac{1}{16} \leq m < 1$; ako je $x > 0$. Mi ovdje želimo samo da izložimo princip sabiranja, pa nam je pogodno da 16 zamijenimo sa 10. Dakle, dopustimo da prikazivanje ima oblik $x = m \cdot 10^e$, $\frac{1}{10} \leq m < 1$. Znamo da se za mantisu kaže da je normalizovana ako se njena vrijednost kreće u okviru $\frac{1}{16} \leq m < 1$ (odnosno $\frac{1}{10} \leq m < 1$).

U tabeli 8.2 prikazano je sabiranje $9.500.000 + 850.000 = 10.350.000$

Vidimo da se sabiranje zasniva na jednakosti

$$m_1 \cdot 10^e + m_2 \cdot 10^e = (m_1 + m_2) \cdot 10^e.$$

I sabirak		II sabirak		zbir		
m	e	m	e	m	e	
0,95	7	0,85	6			
		0,085	7			poravnavanje eksponenata
				1,035	7	sabiranje mantisa
				0,1035	8	normalizacija mantise zbira

Tabela 8.2: Sabiranje u pokretnom zarezu

Primjer 3. Razmotrimo množenje brojeva u pokretnom zarezu. Primjer opisan u tabeli 8.3 jeste očito $20 \cdot 3.000 = 60.000$.

Osnovni princip koji se koristi je:

$$(m_1 \cdot 10^{e_1}) \cdot (m_2 \cdot 10^{e_2}) = (m_1 \cdot m_2) \cdot 10^{e_1+e_2}$$

I množ.		II množ.		proizvod		
m	e	m	e	m	e	
0,2	2	0,3	4			
				0,06	6	množenje mantisa, sabiranje eksponenata
				0,6	5	normalizacija mantise proizvoda

Tabela 8.3: Množenje u pokretnom zarezu

8.8 Organizacija ulazno–izlaznih jedinica

Jedan računar povezan je sa nekoliko ulaznih i izlaznih uređaja, koje nazivamo *perifernim jedinicama.* Kao što smo vidjeli ranije, centralni procesor i unutrašnja memorija su elektronski uređaji. S druge strane, periferije su najčešće elektro–mehanički uređaji. Zato su centralni procesor i periferni uređaji povezani posredstvom izvjesnog hardvera koji omogućava skladno funkcionisanje. Taj hardver je tzv. *interfejs* ili *kontroler.*

Kada se podaci iz centralnog procesora šalju nekom perifernom uređaju onda odgovarajući interfejs prihvata signale koji mu pristižu, prerađuje te signale u oblik koji odgovara uređaju i predaje signale samom uređaju. Onda periferni uređaj (recimo da je to štampač) izvrši odgovarajuću radnju (odštampa slovo). Interfejs štampača određuje koje slovo će biti odštampano, reguliše vremenske intervale štampanja, čini da se papir pomjera, i slično. I kada se podaci prenose u suprotnom smjeru, od perifernog uređaja prema procesoru, imamo slične okolnosti.

Magistrala služi da CPU komunicira sa više perifernih uređaja. Tipična veza koja služi za komuniciranje prikazana je na slici 8.10. Vidimo da su na slici prikazana četiri periferna uređaja: tastatura (ulazni uređaj), štampač (izlazni uređaj), ekran (izlazni uređaj) i magnetni disk (ulazno–izlazni uređaj) (jedinica spoljašnje memorije).

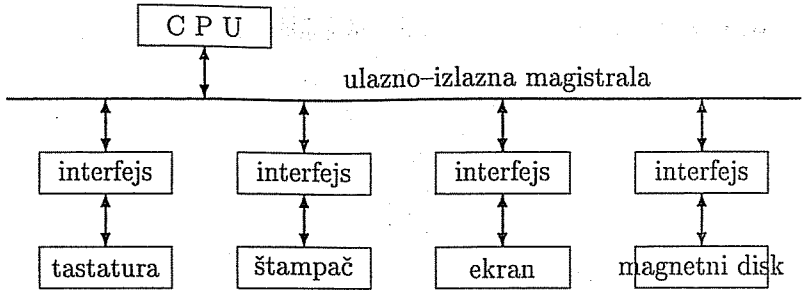
U najprostijem slučaju interfejs se svodi na tzv. ulazno–izlazna vrata, kakva su, na primjer, ulazna vrata za tastaturu. To je praktično jedan prihvatni registar. Nasuprot tome, neki interfejsi predstavljaju složena integrisana kola koja manipulišu sa podacima. Za takav interfejs kaže se da predstavlja *kontroler.* Takvi su, na primjer, kontroler ekrana i kontroler magnetnog diska.

Razmotrimo, u osnovnim crtama, neke periferne uređaje koje često srećemo.

8.8.1 Spoljašnji memorijski uređaji

Obični spoljašnji memorijski uređaji su magnetni disk, magnetna traka i magnetni doboš. Nešto više ćemo reći o magnetnom disku. Magnetnih diskova opet ima dvije vrste:

a) Tvrdi disk ili hard disk ili veliki disk ili stalni disk.



Slika 8.10: Komunikacija CPU sa periferijama

b) Savitljivi disk ili floppy disk ili disketa.

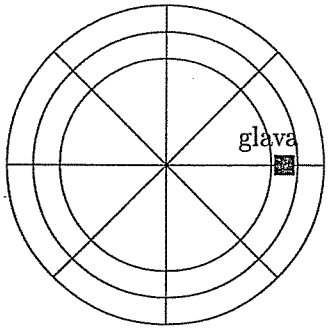
Tvrdi disk (*hard disk*) je veoma važan ulazno–izlazni uređaj. On, za razliku od RAM-a, ne gubi sadržaj nakon prestanka napajanja. Zato se on koristi za smještanje podataka i programa. Podaci i programi se smještaju u vidu *fajlova*.

Memorija sa magnetnim diskovima predstavlja sistem sa glavama i diskovima. Disk je metalna ili plastična ploča kružnog oblika čije su obije strane presvučene magnetnim materijalom. Ploča je podijeljena na koncentrične krugove (trake) koje sadrže memorijske ćelije (bitove). Trake su podijeljene na sektore jednakog kapaciteta. Jedan sektor obično sadrži 512 bajta. Na vertikalnoj osovini smješteno je više ploča. Ploče rotiraju konstantnom brzinom.

Za zapisivanje i očitavanje podataka koriste se magnetne glave. Magnetna glava lebdi iznad ploče na vrlo malom rastojanju. Za zapisivanje, propusti se struja kroz zapisne namotaje glave. Struja stvara magnetno polje koje prodire u magnetni materijal nanesen na ploču. Tako da se u elementu medijuma (materijala) stvori namagnetisanje u jednom ili drugom smjeru (zavisno od smjera zapisne struje), a odgovara logičkoj nuli ili logičkoj jedinici. Slično se dešava kada glava vrši očitavanje stanja elementa (ćelije). Upisno–čitajuća glava može da se kreće linearno prema vertikalnoj osovini ili od vertikalne osovine, a široka je toliko da može upisivati ili čitati podatke sa tačno jedne trake. Obično je mehanička koncepcija sistema izvedena tako da svakoj ploči pripada jedan par magnetnih glava (iznad i ispod ploče).

Ako želimo da izračunamo vrijeme pristupa lokaciji, radi upisivanja ili čitanja podataka, trebamo uračunati: vrijeme potrebno za linearno pozicioniranje glave, vrijeme potrebno da se rotacijom diska odgovarajući sektor postavi ispod glave i vrijeme potrebno za sami prenos podataka.

Na slici 8.11 je prikazan jedan magnetni disk (ploča). Kapacitet tvrdog diska personalnog računara kreće se od 20 MB do nekoliko desetina GB. Adresa riječi upisane na disku određuje se obično pomoću: broja ploče, brojeva trake i sektora i broja riječi.



Slika 8.11: Magnetni disk

Potreba za distribucijom softvera dovela je do pojave savitljivih diskova (disketa). Disketa se može odvojiti od jedinice diska i na taj način se može prenositi softver i podaci. Ime su dobile po fizičkoj savitljivosti diskova. Postoji više vrsta disketa i disketnih jedinica za diskete. Dvije vrste disketnih jedinica su tipične: jedna za diskete prečnika 5,25 inča (koja se danas praktično ne koristi) i druga za diskete prečnika 3,5 inča. Treba napomemuti da su diskete prečnika 3,5 inča zaštićene čvrstim omotom i da ustvari i nisu savitljive. Standardni kapacitet jednog sektora diskete je 512 bajta. Na primjer, uzmimo jednu disketu koja je dvostrana, ima 40 traka i ima 9 sektora po traci. Ukupni kapacitet te diskete iznosi očito:

$$40 \text{ traka} \times 9 \text{ sektora} \times 512 \text{ bajta} \times 2 \text{ strane} = 360\,\text{KB}.$$

Dalje, kod nekih računara prisutna je i tzv. prikrivena memorija ili *keš memorija* (cache). To je jedna vrsta unutrašnje memorije. Karakteriše se velikom brzinom i malim kapacitetom. Upotrebljava se za smještanje podataka koje tekući program često koristi i smještanje onih djelova teksta tekućeg programa koji se često koriste. Kod personalnih računara, keš memorija je smještena na jednom čipu.

Na primjer, memorija personalnog računara može da se sastoji od sljedeća tri dijela, s tim da kapaciteti pojedinih djelova iznose redom: keš 256 KB, unutrašnja 512 MB i spoljašnja 80 GB.

Keš memorija je brža tj. ima kraće vrijeme pristupa od unutrašnje memorije. Slično je i unutrašnja memorija brža od spoljašnje memorije. Keš memorija je tipično jedan red veličine brža od unutrašnje memorije (10 puta brža), a unutrašnja memorija može da bude tri reda veličine brža od spoljašnje memorije (to znači 1000 puta brža).

Važi princip: što je memorijski uređaj sporiji to je cijena po jedinici informacije manja. Zato se za razne vrste memorija u računarskom sistemu često kaže da čine *hijerarhiju memorija*. Na vrhu hijerarhije su memorije malog kapaciteta a skupe, a na dnu one velikog kapaciteta i jeftine.

Organizacija fajlova

Postoje razni načini ogranizacije fajlova na disku. Na primjer, na PC računarima koji koriste operativni sistem DOS ili neki od njegovih naslednika, na tvrdom disku postoje dva važna područja: a) boot record ili boot sector i b) FAT (File Allocation Table) i Korijenski direktorijum (Root Directory). Boot record zauzima jedan sektor (512 bajta), i to prvi po redu sektor na disku. FAT + Root Directory zauzima sljedeće po redu sektore na disku, a potroši približno 1% ukupnog prostora na disku.

Boot record ima značajnu ulogu prilikom pokretanja računara. Na početku pokretanja, taj sektor se učitava na određeno mjesto u unutrašnjoj memoriji i njemu se predaje upravljanje. Izvršavanjem naredbi iz tog dijela unutrašnje memorije čine se dalji koraci u okviru pokretanja sistema. Najveći dio naredbi se odnosi na učitavanje novih programa i podataka (zapisanih u određenim fajlovima). U boot recordu, odnosno u određenom dijelu unutrašnje memorije, nalaze se i neki podaci o disku, kao što su labela ili oznaka diska, veličina jednog sektora u bajtima, broj sektora na jednoj traci, ukupan broj traka koliko ih sadrži čitavi disk, itd.

Fajlovi su ogranizovani u grupe, koje nazivamo *direktorijumima* ili *folderima*. Direktorijumi mogu imati svoje poddirektorijume a postoji tačno jedan korijenski direktorijum. Direktorijum je, u suštini, specijalna vrsta fajla. FAT + (korijenski) Directory sadrži spisak naziva fajlova koji se čuvaju na disku i podatke o rasporedu fajlova. Directory sadrži za svaki fajl: naziv fajla, da li je fajl ili je poddirektorijum, datum i vrijeme, veličina u bajtima, gdje počinje u smislu broj sektora, i slično. Fajl počinje u jednom sektoru, zatim se nastavlja u nekom drugom sektoru koji nije obavezno susjedni, zatim zauzima još jedan sektor, itd, sve do svog "završnog" sektora. Tako da svakom fajlu odgovara jedan "lanac" sektora. Približno govoreći, FAT sadrži za svaki fajl podatke o lancu (podatke o lancu sektora) tog fajla.

8.8.2 Tastatura

Tastatura je ulazni uređaj i služi za prenošenje podataka na način riječ po riječ (karakter po karakter, bajt po bajt, jedna po jedna memorijska riječ). Obično je organizovano tako da podaci prvo dospijevaju u dio unutrašnje memorije koji je za to namijenjen (bafer veličine 20 bajta), a odatle ih program preuzima.

8.8.3 Štampači

Štampači su izlazni uređaji. Oni se izrađuju u raznim tehnologijama i imaju različite principe rada.

Matrični štampač spada u najjednostavnije tehnologije štampača. Procesor takvom štampaču šalje tekst znak po znak, tipično u ASCII kodu. Uz tekst idu i specijalni ASCII znakovi koji označavaju podebljano, kurziv, podvučeno, uvećano, umanjeno i sl. Znak se opisuje pomoću iglica (pin, dot), koje su organizovane u pravougaoni oblik - matricu. Iglicama se formira određeni oblik. Nakon toga, pritiskom iglica preko trake sa mastilom (*ribona*), iscrtava se znak na papir. Zatim se prelazi na sljedeći znak. Štampa se izvodi liniju po liniju, obično veoma bučno i sporo. Ova štampa je prilično slabog kvaliteta.

Laserski štampač sadrži puno više integrisanih kola i njegova saradnja sa procesorom znatno je unaprijeđena. On predstavlja jedan podsistem ukupnog računarskog sistema ili, kako se to često kaže, jedan računar specijalne namjene. Procesor mu šalje podatke u velikim paketima, koji, na primjer, odgovaraju štampanju jedne stranice. Komunikacija naprednijih štampača sa računarom izvodi se pomoću posebnih jezika kojima se opisuje ono što se želi odštampati. Ti jezici se nazivaju *jezicima za opis stranice* (page description language). Primjer takvog jezika je POSTSCRIPT. Ovim jezicima se strana zapisuje u vektorskoj formi, koju zatim sam uređaj prevodi u tačke.

Srce laserskog štampača je rotirajući bubanj, prekriven materijalom koji može da drži naelektrisanje. U početku je čitav bubanj pozitivno naelektrisan. Laserski zrak, korišćenjem sistema ogledala, prelazi preko površine bubnja selektivno postavljajući negativno naelektrisanje na one tačke površine bubnja koje odgovaraju pozicijama na papiru koje treba odštampati. Bubanj rotira tako da sledeća linija bude laserski obrađena, dok tekuća linija nailazi na *toner*. Toner je izuzetno fin crni prah, koji je pozitivno naelektrisan, pa ga privlače negativno naelektrisane tačke bubnja. Tako nakon pune rotacije bubnja površina bubnja sadrži željenu sliku, sa zrncima tonera na negativno naelektrisanim tačkama. Pozitivno naelektrisane tačke ne privlače toner.

U međuvremenu, papir se povlači kroz naelektrisanu žicu koja postavlja negativno naelektrisanje na njega. Papir nakon toga dolazi u kontakt sa bubnjem. Kako je naelektrisanje papira jače od onoga na bubnju, papir magnetski privlači toner na papir, čime se slika prenosi sa bubnja na papir. Na kraju, papir sa nanešenim tonerom prolazi kroz dio štampača koji na njega primjenjuje temperaturu i pritisak. Kako toner sadrži vosak kao komponentu, on se ovim postupkom definitivno vezuje za papir.

Zbog svih ovih naprednih tehnologija, laserski štampači su i danas relativno skupi.

Ink-jet štampači su posebna kategorija štampača koji rasprskavaju mastilo po papiru. Danas su dosta popularni zbog cijene, koja je znatno niža od laserskih štampača. Ovi štampači su bolji od matričnih, ali daju znatno slabije rezultate od laserskih.

8.8.4 Monitori

Monitori ili ekrani su izlazni uređaji. Najčešće se izrađuju u CRT (cathode-ray tube, cijev sa katodnim zracima) tehnologiji, mada u današnje vrijeme sve više mjesta na tržištu zauzimaju monitori zasnovani na tehnologijama tečnih kristala (LCD, liquid color display).

CRT monitori sastoje se od staklene posude bez vazduha, koja počinje uskom cijevi a završava se površinom koju vidimo kad gledamo monitor. Ova površina pokrivena je iznutra hiljadama malih fosfornih grupa (tačaka). Fosfor je hemikalija koja emituje svjetlost kada je pobuđena zrakom elektrona, pri čemu različiti fosfori emituju svjetlost razne boje. Svaka tačka sastoji se od tri fosforna dijela: crvenog, zelenog i plavog. Grupe od tri fosfora čine jednu tačku na ekranu ili *piksel*.

Cijev, kojom počinje staklena posuda sa zadnje strane, sadrži elektronski "top", koji se sastoji od katode, izvora toplote i elemenata za fokusiranje. Kolor monitori imaju tri takva topa, po jedan za svaku osnovnu boju. Kada izvor toplote dostigne određenu temperaturu on počinje da oslobađa elektrone (negativno naelektrisanje) sa katode. Fokusirajući elementi usmjeravaju taj zrak elektrona na određenu fosfornu tačku, koja počinje da emituje svjetlost.

Zrak se usmjerava u svaku tačku ekrana pojedinačno, jednu za drugom. Brzina tog procesa, tzv. *frekvencija osvježavanja,* mjeri se u hercima (Hz). Monitori kod kojih je ona niska, obično je to ispod 50 Hz, trepću i izazivaju zamor oka.

Važna karakteristika monitora je *rezolucija*, odnosno broj tačaka koje mogu biti prikazane na ekranu.

Računarski sistem sadrži poseban kontroler, dakle jednu hardversku komponentu, tzv. *grafičku kartica,* koja kontroliše rad monitora.

8.8.5 Prekid. DMA. Drajver

Procesor i periferni uređaj razmjenjuju kontrolne informacije i obavještenja u užem smislu (podatke). Tako, recimo, procesor naređuje štampaču šta da štampa. Slično, štampač šalje razna obavještenja procesoru - završeno štampanje, nema papira, kvar, itd. Ova obavještenja može procesor da čeka, ne radeći ništa - što očigledno nije efikasno. Drugi način je da se uvede mogućnost *prekida* ili *interapta* (engl. interrupt), tj. uvede mogućnost da dok procesor radi neki drugi posao, neki uređaj u sistemu, na primjer štampač, hardverskim putem (dakle, slanjem signala) izazove prekid te trenutne aktivnosti. Nakon *obrade prekida*, tj. reagovanja na zahtjev tog štampača, procesor nastavlja tamo gdje je stao. Prekidima ćemo se baviti detaljnije u narednim poglavljima.

Čak i mehanizam prekida ponekad nije dovoljno dobar. Neka procesor treba da pošalje perifernom uređaju jednu veliku količinu podataka, paket (napomenimo da slične okolnosti važe i kada se vrši prenos podataka u suprotnom smjeru). Podaci odnosno paket nalaze se u unutrašnjoj memoriji (u

RAM–u) a treba da budu proslijeđeni perifernom uređaju. Svako prebacivanje tih podataka uzima jako puno vremena, što zavisi od brzine perifernog uređaja. Dakle, procesor gubi jako puno vremena čekajući da periferni uređaj završi posao. Ako se čak koristi i sistem prekida, sama obrada puno prekida može uzeti dosta vremena. Zato u savremenim sistemima postoji poseban čip *DMA* (direct memory access) kontroler koji omogućava da se prenos obavi bez opterećenja procesora. Ako se koristi DMA onda put kojim idu podaci više nije memorija – procesor – periferija nego je sada memorija – DMA – periferija. Dakle, ako procesor to želi, on može da se obrati za uslugu DMA integrisanom kolu, koje radi nezavisno od njega.

Kod funkcionisanja DMA integrisanog kola organizovanog na ovaj način, kaže se da periferni uređaj DMA pozajmljuje ili krade cikluse.

Kontrola perifernih uređaja ima i softverskih, dakle programskih elemenata. (Softverski) *drajver* perifernog uređaja je program koji je zadužen za taj periferni uređaj. Recimo, kada neki program želi nešto da odštampa onda se on obraća drajveru, a drajver se obraća štampaču. Slično se štampač obraća svom drajveru a drajver eventualno programu. Pojam drajvera detaljnije ćemo obraditi u poglavlju o operativnim sistemima.

8.9 Pitanja i zadaci

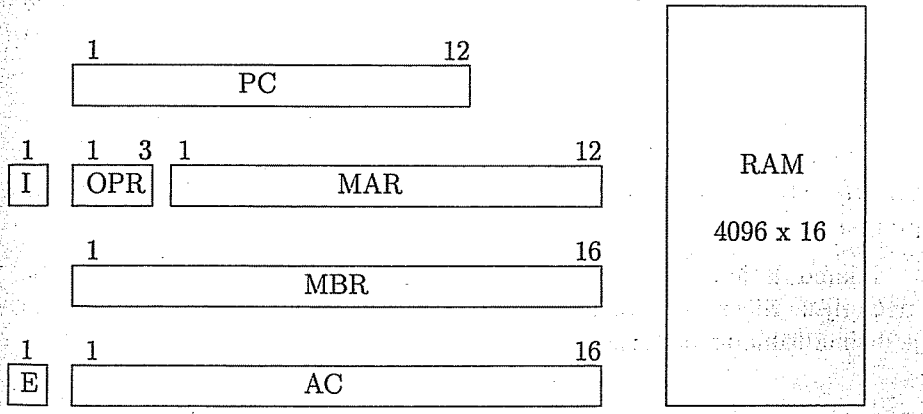
1. Koje su komponente računarskog sistema?
2. Šta je zadatak aritmetičko–logičke jedinice?
3. Koje su funkcije memorije?
4. Kakve memorije postoje u računarskom sistemu?
5. Koje se tehnologije koriste pri izradi memorija?
6. Koji su tipovi RAM memorija, a koji ROM memorija? Navedite karakteristike svakog od njih.
7. Objasnite pojam mikro–operacije.
8. Navedite vrste mikro–operacija.
9. Šta je to kontroler ulazno–izlaznog uređaja?
10. Opišite ulazno–izlazne uređaje na vašem računaru.

11. Koje su vrste štampača? Opišite principe rada za svaku od njih.
12. Šta je to fajl, a šta direktorijum? Šta je korijenski direktorijum?
13. Koje su važnije karakteristike monitora?
14. Čemu služi grafička kartica?
15. Šta je to drajver?
16. Objasnite pojam DMA prenosa.

Glava 9

Arhitektura osnovnog računara

Principe organizacije računara izložićemo detaljnije na primjeru organizacije jednog konkretnog računara. Za njega kažemo da je *osnovni računar* ili *prosti računar* (engl. basic computer). Model koji ovdje razmatramo zasnovan je na $PDP-8$ računaru. Iako su savremeni računari znatno složeniji, ovaj primjer omogućuje razumijevanje osnovnih principa arhitekture računara.



Slika 9.1: Arhitektura osnovnog računara

Na slici 9.1 prikazane su glavne komponente osnovnog računara.

Razmotrimo prvo procesorske registre. Akumulator AC ima 16 bita. Za komuniciranje sa memorijom postoje dva registra: MBR (*memory buffer*

register, prihvatni memorijski registar), od 16 bita, i MAR (*memory address register*, memorijski adresni registar), koji ima 12 bita. Flipflop E je pridružen akumulatoru. Prisutan je i I flipflop za koga kažemo da je mode bit ili bit režima. Pomoću njega se razlikuju neposredno i posredno adresiranje argumenata. Procesor sadrži i OPR registar (operation register) koji sadrži kod naredbe u užem smislu. Kažemo da I i OPR registar zajedno sadrže kod naredbe u širem smislu. Slično, možemo reći da I, OPR i MAR zajedno predstavljaju *kontrolni registar* (control register). Na kraju, tu je i PC registar ili *programski brojač* (program counter). On ima 12 bita, tj. sastoji se od 12 flipflopova. Sadržaj ovog brojača određuje koja naredba programa u datom trenutku treba da bude izvršena.

Memorijska jedinica osnovnog računara sastoji se od 4096 riječi, tj. registara. Pojedina riječ ima 16 bita. Radi se o RAM memoriji, dakle u svakom trenutku može se pročitati sadržaj bilo kog registra ili nešto upisati u registar. Ukratko, memorija predstavlja RAM oblika 4096x16.

Sadržaj memorije podijeljen je na dva dijela. Prvi dio sadrži program napisan na mašinskom jeziku, dakle pomoću 0 i 1. Pojedina naredba ima 16 bita, tako da ona zauzima jednu memorijsku riječ. Drugi dio čine podaci koje se obrađuju. Podaci mogu da budu ulazni podaci, međurezultati ili rezultati rada programa.

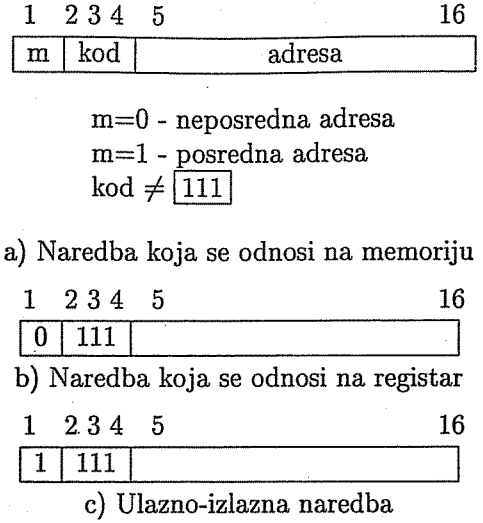
Prije početka izvršavanja programa potrebno je učitati program u memoriju. Za ovaj proces se upotrebljava riječ *loadovanje* (od engl. load, utovariti, učitati). Ova radnja, po pravilu, neposredno prethodi početku izvršavanja programa.

Imamo da je $2^{12} = 4096$. Zato je potrebno 12 bita za adresiranje jedne memorijske riječi. To je razlog zašto registri MAR i PC imaju 12 bita. Kako je dužina jedne memorijske riječi 16 bita, MBR registar mora imati 16 bita.

9.1 Oblik naredbe osnovnog računara

Potrebno je precizno definisati kakav je oblik naredbe osnovnog računara. Kazali smo da naredba osnovnog računara zauzima jednu memorijsku riječ, tj. da ima 16 bita.

Naredba osnovnog računara sastoji se od dva osnovna dijela - prva četiri bita (bitovi 1-4) i sljedećih 12 bita (bitovi 5-16).



Slika 9.2: Oblik naredbe osnovnog računara

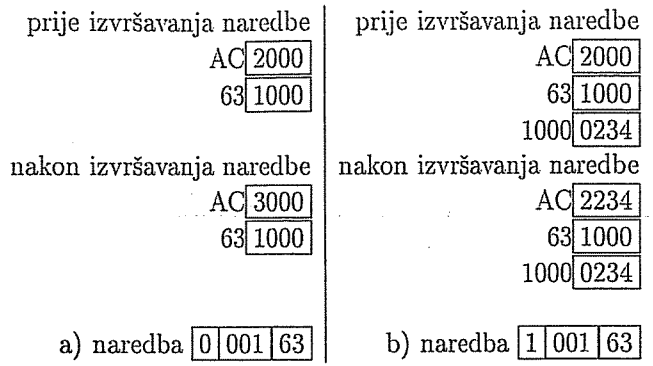
Postoje tri vrste naredbi (slika 9.2):

- naredbe koje se odnose na memoriju. Ove naredbe karakterišu se po tome što su kod njih bitovi 2-4 različiti od 111. Prvi bit može biti 0 ili 1, što označava neposredno odnosno posredno adresiranje, respektivno. Preostalih 12 bitova (5-16) sadrže memorijsku adresu. Ovih naredbi ima onoliko koliko ima kombinacija bita 2-4 različitih od 111, dakle 7.
- naredbe koje se odnose na registre. Naredbe ove klase razlikuju se od ostalih po tome što počinju sa 0111. Biti od 5-16 su svi 0 osim jednog, tako da ovih naredbi ukupno ima 12.
- ulazno-izlazne naredbe. Prva četiri bita ovih naredbi su 1111. Osnovni računar ima ukupno 6 naredbi iz ove grupe.

Dakle, osnovni računar ima ukupno 25 naredbi.

Kao što ćemo vidjeti kasnije, pripremni korak za izvršavanje naredbe jeste njeno prepisivanje iz memorije u procesor. U suštini, prvi bit naredbe prepisuje se u I, bitovi 2-4 u OPR, a bitovi 5-16 u MAR. Iako to nije uvijek

korektno (osim u slučaju naredbi koje se odnose na memoriju) bit 1 naziva se *mode bitom*, bitovi 2-4 su *kod operacije* (govore koja naredba treba da se izvrši), a bitovi 5-16 čine *adresu*.



Slika 9.3: Neposredno i posredno adresiranje

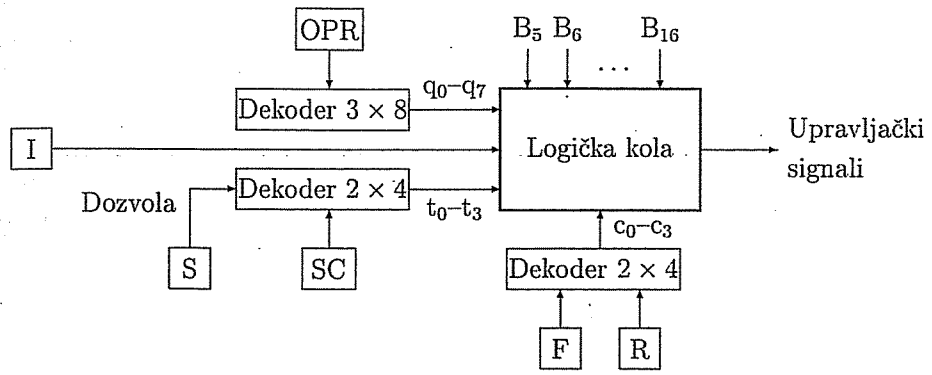
Objasnimo još pojmove neposrednog i posrednog (ili *indirektnog*) adresiranja. Ovi pojmovi imaju smisla samo kod naredbi koje se odnose na memoriju.

Kada se radi o direktnom ili neposrednom adresiranju (kada je I=0), dio 5-16 naredbe sadrži *adresu argumenta* naredbe. Na primjer, vidjećemo da naredba koja počinje sa 0001 označava sabiranje sa neposrednim adresiranjem. Ova operacija uzima vrijednost sa naznačene lokacije u memoriji i dodaje ga na vrijednost akumulatora. Recimo, ako se na adresi 63 (dekadno) nalazi broj 1000, naredba 0 001 63 označava 'dodaj na vrijednost akumulatora sadržaj lokacije 63' odnosno 'dodaj 1000 na vrijednost akumulatora' (slika 9.3.a)).

Kod posrednog ili indirektnog adresiranja, u naredbi naznačena adresa ne predstavlja adresu argumenta već *adresu na kojoj se nalazi adresa argumenta*. Tako naredba 1 001 63 označava 'dodaj na vrijednost akumulatora sadržaj memorije sa lokacije čija je adresa na lokaciji 63' što u konkretnoj situaciji znači 'dodaj na vrijednost akumulatora sadržaj lokacije 1000' odnosno, ako je na lokaciji 1000 upisana vrijednost 234, 'dodaj 234 na vrijednost akumulatora' (slika 9.3.b)). Dakle, u slučaju indirektnog adresiranja ne znamo direktno adresu argumenta već je potrebno dodatno čitanje da to saznamo.

9.2 Kontrolna jedinica i vremenski ciklusi

Procesor (CPU, central processor unit, centralna procesorska jedinica, ili jednostavno procesor) se sastoji od upravljačke jedinice (CU control unit) i aritmetičko–logičke jedinica (ALU, arithmetic–logic unit).



Slika 9.4: Blok–dijagram kontrolne jedinice

Na slici 9.4 prikazan je blok–dijagram kontrolne jedinice. Vidi se da glavni dio kontrolne jedinice čine "logička kola". To je jedan splet upravljačkih logičkih kola – izlazi iz spleta su *upravljački signali*. Ti signali definišu mikro–operacije.

Dakle, glavni zadatak kontrolne jedinice je generisanje upravljačkih signala. Razmotrimo način njenog rada. Sa slike se vidi da ima pet vrsta ulaza, to su: t_0–t_3, c_0–c_3, I, q_0–q_7 i B_5–B_{16}. Opišimo ih redom.

Registar SC (sequence counter, brojač taktova) ima dva bita. Taj registar definiše vremensko vođenje rada računara, odnosno on predstavlja generator taktova (izvor otkucaja). Sadržaj dvobitnog brojača SC daje se na ulaz tzv. vremenskog dekodera koji ima oblik 2 puta 4. Izlazi dekodera označeni su kao t_0, t_1, t_2 i t_3. Brojač SC radi stalno na isti način. Njegov rad neposredno se odražava na vrijednosti promjenljivih t_0 do t_3, tako da se i te četiri vrijednosti (tokom cijelog vremena rada računara) mijenjaju stalno po jednom te istom nepromjenljivom šablonu. Iz takta u takt imamo:

$$t_0 = 1,\ t_1 = 0,\ t_2 = 0,\ t_3 = 0$$

$$t_0 = 0, t_1 = 1, t_2 = 0, t_3 = 0$$
$$t_0 = 0, t_1 = 0, t_2 = 1, t_3 = 0$$
$$t_0 = 0, t_1 = 0, t_2 = 0, t_3 = 1.$$

U jednom redu prikazan je jedan takt. Dalje se ponavlja. Postoji samo jedna mogućnost da se ovaj šablon poremeti. Flipflop S ima ulogu prekidača za start–stop. Taj flipflop djeluje na ulaz za dozvolu (Enable) vremenskog dekodera, pa se vremenski signali t_i generišu samo kada je $S = 1$. Ako je $S = 0$ onda će biti $t_0 = t_1 = t_2 = t_3 = 0$ i tada nijedan upravljački signal više neće biti izdat, što znači da će se računar zaustaviti.

Digitalni računar radi u diskretnim koracima. Mikro–operacije se izvršavaju tokom svakog koraka. Naredbe se čitaju iz memorije i izvršavaju u registrima CPU. One se izvršavaju preko mikro–operacija.

Kada se jednom aktivira startni prekidač, onda računar stalno postupa po svom šablonu. Iz memorije se pročita naredba čija se adresa nalazi u registru PC; naredba se upisuje u registar MBR. Mode bit naredbe prepiše se u flipflop I, a kod naredbe u užem smislu prepiše se u registar OPR (tri bita). Sadržaj registra OPR dekodira se u upravljačkoj jedinici. Ako se vidi da se naredba odnosi na memoriju (pa prema tome zahtijeva argument iz memorije) onda kontrola gleda sadržaj flipflopa I. Ako je $I = 1$ onda se kontrola obraća memoriji da bi se pročitala adresa argumenta, pa se zatim ponovo obraća memoriji da bi se i sam argument pročitao. Ako je $I = 0$ onda se memoriji pristupa radi čitanja argumenta.

Prema tome, riječ koja se nalazi u registru MBR je jednog od sljedeća tri tipa: to je naredba ili adresa argumenta ili argument. Kada se iz memorije čita naredba onda se kaže da se računar nalazi u ciklusu dohvatanja naredbe (u ciklusu primanja naredbe, u prijemnom ciklusu, u fetch ciklusu). Tokom vremena čitanja (iz memorije) adrese argumenta računar se nalazi u indirektnom ciklusu (indirect cycle). A tokom izvršnog ciklusa (execute cycle) obavi se, pored ostalog, i čitanje argumenta. Kasnije ćemo vidjeti da postoji i četvrti vremenski ciklus – interrupt cycle (prekidni ciklus).

Dakle, postoje četiri osnovna *ciklusa* u radu računara, to su fetch (dohvatanje), indirect (indirektni), execute (izvršni) i interrupt (prekidni) ciklus. Zadatak kontrolne jedinice je da reguliše smjenjivanje vremenskih ciklusa navedena četiri tipa. U kontrolnoj jedinici dva flipflopa služe da se ciklusi razlikuju. To su flipflopovi F i R. Sa slike se vidi da se izlazi flipflopova F

i R dovode na ulaze tzv. *ciklusnog dekodera* koji ima oblik 2 puta 4. Izlazi dekodera su c_0 do c_3. Očito je da se dodjeljivanjem vrijednosti flipflopovima F i R definiše jedan od četiri moguća ciklusa. Nakon toga, veličine c_0 do c_3 učestvuju u definisanju upravljačkih signala.

Ako je

$F = 0, R = 0$ slijedi $c_0 = 1, c_1 = 0, c_2 = 0, c_3 = 0$, ovo je Fetch cycle;
$F = 0, R = 1$ slijedi $c_0 = 0, c_1 = 1, c_2 = 0, c_3 = 0$, ovo je Indirect cycle;
$F = 1, R = 0$ slijedi $c_0 = 0, c_1 = 0, c_2 = 1, c_3 = 0$, ovo je Execute cycle;
$F = 1, R = 1$ slijedi $c_0 = 0, c_1 = 0, c_2 = 0, c_3 = 1$, ovo je Interrupt cycle.

Izvršavanje jednog ciklusa traje tačno četiri takta. Već je rečeno da se veličine t_0 do t_3 mijenjaju monotono: iz takta u takt je $t_0 = 1$ pa $t_1 = 1$ pa $t_2 = 1$ pa $t_3 = 1$, a dalje periodično (kružno). Za promjenljive c_0 do c_3 možemo reći samo da se mijenjaju u osnovi monotono (obično poslije $c_0 = 1$ ide $c_1 = 1$, zatim $c_2 = 1$, onda $c_3 = 1$, itd). Ako se ne koristi mogućnost prekida onda će biti preskočen prekidni ciklus $c_3 = 1$. Drugim riječima, poslije $c_2 = 1$ postaće $c_0 = 1$. Slično, tokom izvršavanja naredbe koja se odnosi na memoriju a čija je adresa zadata neposredno ($I = 0$) biće preskočen indirektni ciklus $c_1 = 1$. Drugim riječima, poslije $c_0 = 1$ postaće $c_2 = 1$.

Prekidni ciklus kod ovog računara koristi se samo prilikom ulazno–izlazne komunikacije. Kasnije ćemo vidjeti da ulazno–izlazna komunikacija može da bude ostvarena bez upotrebe ili sa upotrebom pogodnosti prekida. Pogodnost prekida pruža mogućnost programeru da komunikaciju sa ulaznom i izlaznom jedinicom organizuje na efikasan način.

Za ilustraciju, uzmimo da se izvršava dio mašinskog programa sastavljen od niza naredbi koje se odnose na memoriju, gdje svaka naredba ima neposrednu adresu. Dodatno, uzmimo da izvršavanje tog dijela programa nije spolja ometano (prekidano), tj. da nema spoljašnjih događaja koji mogu da nastupe asinhrono, a koji bi uticali na izvršavanje. Drugim riječima, uzmimo da se ne zalazi u prekidni ciklus. Tada će se vrijednosti promjenljivih c_0 do c_3 (gledano po vremenskoj osi) smjenjivati ovako:

$$c_0 = 1, c_1 = 0, c_2 = 0, c_3 = 0, \text{ tokom četiri takta}$$
$$c_0 = 0, c_1 = 0, c_2 = 1, c_3 = 0, \text{ tokom četiri takta}$$
(dalje se periodično ponavlja)

U slučaju da svaka naredba ima posrednu (indirektnu) adresu redosljed vremenskih ciklusa je sljedeći:

$c_0 = 1, c_1 = 0, c_2 = 0, c_3 = 0$, tokom četiri takta
$c_0 = 0, c_1 = 1, c_2 = 0, c_3 = 0$, tokom četiri takta
$c_0 = 0, c_1 = 0, c_2 = 1, c_3 = 0$, tokom četiri takta
(dalje se periodično ponavlja).

Izvršavanje jedne naredbe traje četiri ciklusa ili manje. To znači da izvršavanje bilo koje naredbe traje 16 taktova ili manje. Kasnije ćemo vidjeti da izvršavanje pojedine naredbe traje, zavisno od okolnosti, dva ili tri ili četiri ciklusa, tj. 8 ili 12 ili 16 taktova.

Vrijednost flipflopa I takođe utiče na definisanje kontrolnih signala. Toj vrijednosti nije potreban dekoder. Sa slike se vidi da se izlaz flipflopa I sprovodi pravo na upravljačku kombinacionu mrežu.

Sa slike se vidi da se sadržaj registra OPR uvodi u jedan dekoder oblika 3 puta 8 (to je tzv. *dekoder stanja*), čiji su izlazi označeni redom kao q_0 do q_7. Sa svoje strane, ti izlazi dovode se na upravljačku kombinacionu mrežu. Ako je $q_0 = 1$ ili itd. ili $q_6 = 1$ (sedam mogućnosti) onda se radi o naredbi prve vrste (ima sedam naredbi prve vrste). Ako je $I = 0$ i $q_7 = 1$ onda se radi o naredbi druge vrste. A ako je $I = 1$ i $q_7 = 1$ onda se radi o naredbi treće vrste.

U kombinacionu upravljačku mrežu uvode se i signali B_5 do B_{16}. Uvedena je oznaka $B_i = MBR(i)$, za $1 \leq i \leq 16$. Vidjećemo da se ti signali koriste kada se radi o naredbama druge vrste (naredbe koje se odnose na registre) i naredbama treće vrste (ulazno–izlazne naredbe). Kao što znamo, za te naredbe karakteristično je da one ne traže argument iz memorije. Zato se adresno polje koristi da se zada vrsta naredbe. Drugim riječima, pomoću B_i saznaje se vrsta naredbe, odnosno o kojoj vrsti radnje je riječ. Na primjer, ima 12 naredbi druge vrste. Svakoj od tih 12 naredbi odgovara jedan određeni raspored vrijednosti B_5 do B_{16}. U svakom rasporedu, samo jedna od tih vrijednosti je jednaka 1, a sve ostale su jednake 0. Slične okolnosti važe i kada se radi o naredbama treće vrste, samo što naredbi treće vrste ima manje od 12, tako da nisu iskorišćene sve mogućnosti. Ponovimo da mogućnosti ima ukupno 12: raspored u kome je $B_5 = 1$, ..., raspored u kome je $B_{16} = 1$.

9.2.1 Vremenski ciklusi

Prijemni ciklus (Fetch cycle)

Prijemni ciklus sastoji se od sljedećih mikro–operacija:

c_0t_0: MAR $\leftarrow$ PC	prenošenje adrese naredbe
c_0t_1: MBR $\leftarrow$ M, PC $\leftarrow$ PC + 1	naredba se donosi, PC mijenja
c_0t_2: I $\leftarrow$ MBR(1), OPR $\leftarrow$ MBR(2 – 4)	djelovi naredbe se raspoređuju
c_0t_3: if $q_7'I = 1$ then R $\leftarrow$ 1 else F $\leftarrow$ 1	podešavanje vrijednosti F i R

Izvršavanje prijemnog ciklusa traje četiri takta. Prijemni ciklus poznaje se po promjenljivoj c_0. Način izvršavanja prijemnog ciklusa je svaki put jedan te isti, ne zavisi od vrste naredbe, ne zavisi od naredbe. Kao što znamo, zadatak tog ciklusa je da se iz memorije donese naredba i da se pripremi teren za naredni ciklus. Opišimo detaljnije mikro–operacije uključene u ovaj ciklus.

Dok je $t_0 = 1$ sadržaj programskog brojača PC prepisuje se u adresni registar MAR. To je zato što PC sadrži adresu naredbe koja je stigla na red za izvršavanje, a MAR (u saradnji sa MBR) učestvuje u ostvarivanju komunikacije sa memorijom. U toku $t_1 = 1$, u MBR se donosi sadržaj memorijske lokacije M (radnja čitanja memorije). Tokom tog istog vremena $t_1 = 1$ PC se uveća za 1, čime je PC već u tom trenutku postao pripremljen za iduće čitanje memorije, za izvršavanje iduće mašinske naredbe (što će uslijediti kasnije).

U periodu $t_2 = 1$ izvršiće se dvije radnje. Prvi bit registra MBR prepisuje se u flipflop I. Bitovi 2–4 registra MBR prepisuju se u registar OPR. Zapazimo da je adresni dio naredbe 5–16 ostao u MBR (čitava naredba je ostala u 1–16).

Vrijeme $t_3 = 1$ posvećeno je definisanju radnji koje treba da uslijede. Ako je $B_1B_2B_3B_4 = 0111$ (naredba druge vrste) ili $B_1B_2B_3B_4 = 1111$ (naredba treće vrste), onda treba preći u izvršni ciklus. Ako je $B_2B_3B_4 \neq 111$ i $B_1 = 0$ (naredba prve vrste sa direktnom adresom), onda takođe treba preći u izvršni ciklus. Jedino ako je $B_2B_3B_4 \neq 111$ i $B_1 = 1$ (naredba prve vrste sa indirektnom adresom), treba preći u indirektni ciklus. Veličina B_1 vidi se po I. Veličine B_2, B_3 i B_4 vide se (posredstvom OPR) po $q_0, \ldots, q_7$.

Da bi se razumjela dešavanja tokom $t_3 = 1$ treba znati da je prilikom ulaska u prijemni ciklus bilo $F = 0$ i $R = 0$. S druge strane, ako na završetku prijemnog ciklusa bude $F = 0$ i $R = 1$, onda će se preći u indirektni ciklus, a ako bude $F = 1$ i $R = 0$, onda u izvršni. Znamo da F i R determinišu

vrijednosti promjenljivih $c_0, \ldots, c_3$. Poznata je jednakost $(q_7'I)' = q_7 + I'$. Očito je da $q_7'I = 1$ znači $q_7 = 0$ i $I = 1$. Isto tako, $q_7'I = 0$ znači $q_7 = 1$ ili $I = 0$.

U drugom redu (kada je $c_0 = 1$ i $t_1 = 1$) piše MBR $\leftarrow$ M, PC $\leftarrow$ PC + 1. Te dvije radnje MBR $\leftarrow$ M i PC $\leftarrow$ PC + 1 možemo smatrati jednom mikro–operacijom, jer se one izvedu tokom jednog takta.

Naš opis rada osnovnog računara predviđa da čitanje memorije (ili upis u memoriju) može da bude obavljeno tokom jednog takta. Mi uzimamo ovu pretpostavku da bi se izlaganje pojednostavilo. Kod mnogih računara za čitanje ili upisivanje potrebno je vrijeme koje je duže od jednog takta.

Indirektni ciklus (Indirect cycle)

U toku indirektnog ciklusa izvršavaju se sljedeće mikro–operacije:

c_1t_0:	MAR $\leftarrow$ MBR(5 − 16)	adresni dio naredbe ide u adresni registar
c_1t_1:	MBR $\leftarrow$ M	mikro–operacija čitanja memorije
c_1t_2:		ništa se ne dešava
c_1t_3:	F $\leftarrow$ 1, R $\leftarrow$ 0	determiniše se prelazak u izvršni ciklus

Indirektni ciklus poznaje se po promjenljivoj c_1. Tokom indirektnog ciklusa vrši se čitanje memorijske lokacije koja je zapisana u MBR registru.

Prilikom $t_0 = 1$ izvrši se radnja registarskog prenosa kojom se pripremi adresni registar MAR. Tokom $t_1 = 1$ izvrši se radnja čitanja memorijske riječi M odnosno M[MAR]. Riječ M ima 16 bita, iako u konkretnom slučaju M ima smisao adrese; znamo da za definisanje adrese treba 12 bita. Tokom $t_2 = 1$ imamo prazan hod. To je zato što se prilagođavanje vrijednosti dva flipflopa F i R uvijek vrši tokom vremena $t_3 = 1$, kako bi idući ciklus počeo u situaciji kada je $t_0 = 1$. Tokom $t_3 = 1$ setuje se (postavlja) F i resetuje (briše) se R. Iz indirektnog ciklusa uvijek se prelazi u izvršni ciklus.

Izvršni ciklus (Execute cycle)

Tokom izvršnog ciklusa dolazi do izvršavanja (u užem smislu) jedne mašinske naredbe. Tako da četiri mikro–operacije koje se izvedu tokom tog ciklusa zavise od vrste mašinske naredbe. Budući da ima 25 vrsta naredbi, da bi potpuno definisali računarov izvršni ciklus, trebamo napisati 25 tabela, od kojih svaka ima četiri reda. Umjesto toga, ilustrovaćemo izvršni ciklus na

jednom primjeru: razmotrićemo naredbu za uvećavanje akumulatora ADD. Toj naredbi odgovara $B_2B_3B_4 = 001$, tj. OPR = 001, odnosno $q_1 = 1$.

ADD operacija

$q_1c_2t_0$:	MAR $\leftarrow$ MBR(5 − 16)	adresa se donosi u adresni registar
$q_1c_2t_1$:	MBR $\leftarrow$ M	mikro–operacija čitanja memorije
$q_1c_2t_2$:	EAC $\leftarrow$ AC + MBR	izvršava se sabiranje i sačuva prenos
$q_1c_2t_3$:	F $\leftarrow$ 0	prelazak u prijemni ciklus

Izvršni ciklus poznaje se po tome što je $c_2 = 1$. Mašinska naredba ADD poznaje se po tome što je $q_1 = 1$. Naredba ADD ostvaruje jedno sabiranje. Flipflop E pomaže prilikom sabiranja, time što prihvata prenos sa mjesta najveće težine. Na E i AC ovdje se gleda kao na jednu cjelinu.

Može da bude I = 0 ili I = 1, što za odvijanje izvršnog ciklusa nema značaja. Zaista, ako je I = 1 onda je računar maločas prošao kroz indirektni ciklus. Dakle, u trenutku kada je postalo $q_1 = c_2 = t_0 = 1$ imamo sigurno da dio 5–16 registra MBR sadrži m, gdje je m adresa argumenta. Mi želimo da u MBR dospije argument M, jer M ima smisao drugog sabirka y (prvi sabirak je u akumulatoru).

Tokom $t_0 = 1$ pripremi se adresni registar MAR. Kada je $t_1 = 1$, pročita se argument. Dok je $t_2 = 1$, izvrši se samo sabiranje. Tako da učinak izvršnog ciklusa naredbe ADD možemo da izrazimo kao EAC $\leftarrow$ AC + M. Zapis EAC $\leftarrow$ AC + M odražava jednu veliku operaciju ili jednu *makro–operaciju*. Tokom $t_3 = 1$ učini se da postane F = 0 i R = 0, što će očito izazvati prelazak u novi prijemni ciklus. Preciznije, uradi se F $\leftarrow$ 0, jer je i inače bilo R = 0.

Naglasimo da je u prethodnoj tabeli radnja tokom $t_3 = 1$ prikazana uprošćeno. U realnoj situaciji se tokom $t_3 = 1$ vrši podešavanje vrijednosti dva flipflopa F i R. Postoje dva moguća ishoda podešavanja: da se pređe u prekidni ciklus ili da se pređe u novi prijemni ciklus. Mi smo zasad prikazali kao da se svaki put prelazi u novi prijemni ciklus, a kasnije će biti prikazano kako treba.

Prekidni ciklus (Interrupt cycle)

O prekidnom ciklusu biće riječi kasnije, kada budemo govorili o ulazno–izlaznim naredbama. Ovdje ćemo ukratko dati nekoliko napomena o pojmu prekida ili interapta. Računari koriste tzv. pogodnost prekida ili sistem

prekida. Sistem prekida stavlja na raspolaganje mogućnost da program P, čije izvršavanje je u toku, može da prepusti upravljanje računarom drugom programu Q. Pomoćni program Q (*rutina* Q) služi po pravilu da obavi neku kratku i rutinsku obradu, najčešće neko ulazno ili izlazno prenošenje. Kada se ta obrada obavi, onda će se upravljanje vratiti programu P. Rad programa P biće obnovljen, izvršavanje programa P biće nastavljeno kao da P nije ni bio prekidan.

Odluka o eventualnom prepuštanju upravljanja zavisi od samog programa P, ali i od spoljašnjih okolnosti, odnosno od spoljašnjeg signala. Spoljašnji signal nastupa asinhrono u odnosu na rad programa P.

Ako su se stekli uslovi onda će se tokom $c_3 = 1$ upravljanje računarom prepustiti rutini Q. Drugim riječima, tada će se izvršiti skok na rutinu Q.

9.3 Naredbe koje se odnose na memoriju

U ovom naslovu definišu se naredbe prve vrste osnovnog računara. Za ove naredbe karakteristično je da se one obraćaju memoriji. Pregled tih naredbi, uz kratak opis, dat je u tabeli 9.1.

Kod operacije (3 bita) opisan je dijelom 2–4 naredbe. Memorijska riječ sa kojom se radi definisana je adresnim dijelom 5–16 same naredbe. Osim toga, u definisanju memorijske riječi učestvuje i prvi bit naredbe (koji se smješta u flipflop I).

Pomoću naredbe AND izvrši se konjunkcija bit–po–bit između memorijske riječi i akumulatora, a rezultat te radnje ostaje upisan u akumulatoru. Naredba ADD vrši obično sabiranje brojeva (brojevi su prikazani binarno). Pomoću LDA vrši se prenošenje iz memorije u procesor. Pomoću STA vrši se prenošenje u suprotnom smjeru.

Sada je potrebno preciznije definisati naredbe. Tabela 9.2 definiše učinak izvršnog ciklusa pojedine naredbe na jeziku velikih operacija (na jeziku makro–operacija).

U tabeli, m označava tekući sadržaj dijela 5–16 registra MBR (m označava sadržaj dijela 5-16 registra MBR u trenutku ulaska u izvršni ciklus). Dakle, m je *adresa* ili *efektivna adresa*. Dalje, M označava sadržaj odgovarajuće memorijske lokacije. Dakle, M je *memorijska riječ*. Ovo možemo iskazati formulom: M = c(m), gdje c znači contents (sadržaj).

	oznaka	2–4	smisao
1	AND	000	izvrši AND između memorijske riječi i akumulatora
	and – i, konjunkcija		
2	ADD	001	uvećaj akumulator za memorijsku riječ
	add – dodati		
3	LDA	010	donesi iz memorije u akumulator
	load to accumulator – učitaj u akumulator		
4	STA	011	pošalji iz akumulatora u memoriju
	store from accumulator – pohrani iz akumulatora		
5	BUN	100	bezuslovni skok (bezuslovno grananje)
	branch unconditionally – granaj bezuslovno		
6	BSA	101	skok i spašavanje povratne adrese
	branch and save return address – granaj i sačuvaj povratnu adresu		
7	ISZ	110	povećaj za jedan i preskoči ako nula
	increment and skip–if–zero – inkrementiraj i preskoči–ako–nula		

Tabela 9.1: Naredbe koje se odnose na memoriju

Razmotrimo posebno slučajeve I = 0 i I = 1. Ako je I = 0 onda je m jednako dijelu 5–16 same naredbe, a M je sadržaj odgovarajuće memorijske lokacije. Možemo reći da se, kada je I = 0, učinak čitave naredbe poklapa sa učinkom njenog izvršnog ciklusa. Međutim, ako je I = 1 onda je izvršnom ciklusu prethodio indirektni ciklus. Tako da m i M imaju drugi smisao; sada je m sadržaj memorijske lokacije koju definiše dio 5–16 same naredbe.

Ilustrujmo ovo primjerom. Neka treba da se izvrši naredba čiji dio 2–4 glasi 001 (naredba ADD). Neka je u datom trenutku vrijednost akumulatora 2000, i neka je u memorijskoj riječi na adresi 63 upisan podatak 3000, a u memorijskoj riječi na adresi 3000 - podatak 14500. Ovo možemo predstaviti na sljedeći način:

AC [2000], 63 [3000] i 3000 [14500].

U slučaju naredbe [0001 | 63] važi m = 63 i M = 3000, a u slučaju naredbe [1001 | 63] imamo m = 3000 i M = 14500.

Za potpunu definiciju, trebalo bi napisati mikro–operacije za svaku od sedam naredbi, što nećemo raditi zbog prostora. Za naredbu ADD to je urađeno u prethodnom naslovu.

1	AND	$AC \leftarrow AC \wedge M$	bit–po–bit
2	ADD	$EAC \leftarrow AC + M$	$AC \leftarrow AC + M$, prenos najveće težine ide u E
3	LDA	$AC \leftarrow M$	iz memorije u akumulator
4	STA	$M \leftarrow AC$	iz akumulatora u memoriju
5	BUN	$PC \leftarrow m$	u programski brojač upisuje se određeni sadržaj
6	BSA	$M \leftarrow PC$, $PC \leftarrow m + 1$	ostavi dosadašnji sadržaj, a zatim upiši novi sadržaj
7	ISZ	$M \leftarrow M + 1$, if $M + 1 = 0$ then $PC \leftarrow PC + 1$	povećaj za 1 sadržaj lokacije; ako je novi sadržaj lokacije = 0 onda povećaj za 1 programski brojač

Tabela 9.2: Naredbe koje se odnose na memoriju

Sada su sve naredbe prve vrste definisane. U nastavku ćemo navesti nekoliko primjera i komentara vezano za upotrebu naredbi.

Naredba AND. Na primjer, neka prije izvršnog ciklusa naredbe AND imamo $AC = 0110\ldots0$ i $M = 1100\ldots0$. Tada će poslije izvršavanja naredbe biti $AC = 0100\ldots0$.

Naredba ADD. Pogledajmo na primjeru izvršavanja naredbe 0|001|000000011111. Uzmimo da je prije njenog izvršavanja AC bio jednak 0...00101 i da je sadržaj lokacije 000000011111 bio jednak 0...00100. Tada će njenim izvršavanjem da postane $AC = 0\ldots01001$ i $E = 0$. Znači da je (govoreći dekadno) $5 + 4 = 9$ i da nije došlo do prekoračenja.

Naredbe LDA i STA služe za komunikaciju između akumulatora i memorije, odnosno služe za komunikaciju između procesora i dijela RAM–a koji je namijenjen da čuva podatke. Pomoću naredbe LDA iz memorije se donosi neki podatak. Pomoću naredbe STA neki podatak se smješta u memoriju. Prevodilac programa sa višeg programskog jezika na mašinski jezik dodjeljuje svakoj promjenljivoj koja se u programu pojavljuje jednu lokaciju u memoriji, tj. prevodilac definiše "fizičku adresu" promjenljive.

Naredba BUN. Kazali smo da sadržaj programskog brojača PC definiše redosljed izvršavanja naredbi. U običnom slučaju, naredbe se izvršavaju po redu. Tome odgovara inkrementacija registra PC urađena tokom prijemnog ciklusa. Međutim, ponekad treba da se odstupi od običnog redosljeda. Pomoću naredbe BUN u registar PC se prepiše dio 5–16 registra MBR. Znamo da će na početku sljedećeg prijemnog ciklusa (kada bude $c_0 = t_0 = 1$) to biti prepisano u adresni registar MAR. Onda će u idućem taktu biti donesena odgovarajuća naredba.

Naredba BSA odgovara naredbi za prelazak iz glavnog programa u potprogram u slučaju viših programskih jezika (CALL). Ta naredba upisuje vrijednost $m + 1$ u programski brojač PC, čime se očito izvrši bezuslovni skok. Osim toga, ona prethodno zapamti (sačuva) staru (zatečenu) vrijednost programskog brojača. Zatečena vrijednost šalje se na adresu m.

Zatečena vrijednost programskog brojača ima smisao povratne adrese. Spašavanjem povratne adrese stvaraju se preduslovi da kasnije (kada potprogram uradi svoje) bude obnovljeno izvršavanje glavnog programa. Od potprograma se očekuje da na kraju svog rada skoči na povratnu adresu.

Naredba ISZ. Lako se vidi da je mašinska naredba ISZ slična naredbi IF u višim programskim jezicima. Vidi se da se izvršavanje naredbe ISZ odvija u dvije faze. U prvoj fazi inkrementira se vrijednost promjenljive koja se drži na lokaciji m. Ta promjenljiva ima smisao brojača. U drugoj fazi, u zavisnosti od toga da li je nova vrijednost našeg brojača $\neq 0$ ili $= 0$, ne inkrementira se ili se inkrementira programski brojač PC. Ukupno, na kraju će biti $PC = k + 1$ ili $PC = k + 2$, gdje sama naredba ISZ zauzima lokaciju k.

Programer obično postavlja na lokacije $k + 1$ i $k + 2$ dvije naredbe za bezuslovni skok.

9.4 Naredbe koje se odnose na registre

U ovoj sekciji govorimo o naredbama druge vrste, o naredbama koje se odnose na registre unutar procesora. Kontrola poznaje naredbu iz ove grupe po tome što naredba ima oblik |0111.....| ili kodirano heksadekadno |7...|. Karakteristika ove grupe naredbi je i da u bitovima 5-16 ima samo jedna jedinica. Ove naredbe prikazane su u tabelama 9.3 i 9.4.

Vidimo da se radi o relativno jedostavnim naredbama. Svaka od njih može da se izvrši za jednu mikro-operaciju. Konkretno, naredba se izvršava sa vremenskom promjenljivom t_3. Vremenske promjenljive t_0, t_1, t_2 se ne upotrebljavaju.

Kod naredbe CIR vrši se desni kružni šift za jedno mjesto nad EAC, tj. za ovu naredbu treba flipflop E i registar AC smatrati cjelinom. Slično važi i za naredbu CIL.

Cijeli brojevi se zapisuju u potpunom komplementu. Zato znak zapisanog broja možemo saznati na osnovu prvog bita slijeva, bita a_1. Ako je $a_1 = 0$ broj je pozitivan, a ako $a_1 = 1$ broj je negativan.

Ozn.	Opis naredbe
CLA	Clear AC; izbriši AC
CLE	Clear E; izbriši E
CMA	Complement AC; komplementiraj AC
CME	Complement E; komplementiraj E
CIR	Circulate right E and AC; kružni desni šift nad E i AC
CIL	Circulate left E and AC; kružni lijevi šift nad E i AC
INC	Increment AC; inkrementiraj akumulator
SPA	Skip if AC is positive; preskoči jednu naredbu ako je AC>0
SNA	Skip if AC is negative; preskoči jednu naredbu ako je AC<0
SZA	Skip if AC is zero; preskoči jednu naredbu ako je AC=0
SZE	Skip if E is zero; preskoči jednu naredbu ako je E=0
HLT	Halt computer; zaustavi računar

Tabela 9.3: Naredbe koje se odnose na registre

Naredbe SPA, SNA, SZA, SZE odgovaraju IF (uslovnoj) naredbi u višem programskom jeziku.

Pomoću naredbe HLT resetuje se flipflop S, čime se okončava vremenska serija.

9.5 Ulazno-izlazna jedinica i mogućnost prekida

Komercijalni računari sadrže više različitih ulaznih i izlaznih uređaja (jedinica). Kako ovdje želimo samo da ilustrujemo glavne elemente ulazno-izlazne komunikacije, pretpostavićemo da osnovni računar ima jednu jedinicu koja je istovremeno i ulazni i izlazni uređaj. Neka je to teleprinter, koji ima

Br.	Ozn.	Naredba	Hex	Operacija
1	CLA	0111100000000000	7800	$AC \leftarrow 0$
2	CLE	0111010000000000	7400	$E \leftarrow 0$
3	CMA	0111001000000000	7200	$AC \leftarrow \overline{AC}$
4	CME	0111000100000000	7100	$E \leftarrow \overline{E}$
5	CIR	0111000010000000	7080	$cirEAC$
6	CIL	0111000001000000	7040	$cilEAC$
7	INC	0111000000100000	7020	$EAC \leftarrow AC + 1$
8	SPA	0111000000010000	7010	if $AC(1) = 0$ then $PC \leftarrow PC + 1$
9	SNA	0111000000001000	7008	if $AC(1) = 1$ then $PC \leftarrow PC + 1$
10	SZA	0111000000000100	7004	if $AC = 0$ then $PC \leftarrow PC + 1$
11	SZE	0111000000000010	7002	if $E = 0$ then $PC \leftarrow PC + 1$
12	HLT	0111000000000001	7001	$S \leftarrow 0$

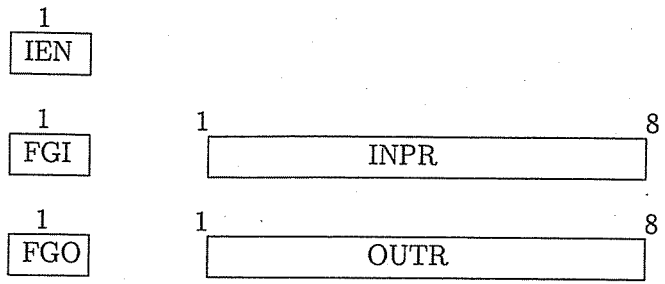
Tabela 9.4: Naredbe koje se odnose na registre

tastaturu (ulazni uređaj) i štampač (izlazni uređaj). Ako se pritisne taster, uređaj je u stanju da proizvede jedan niz impulsa koji je ekvivalentan binarnom kodu karaktera koji odgovara tom tasteru. Taj niz biva proslijeđen u pomjerački registar i tako čini ulazni karakter. S druge strane, jedna serija impulsa alfanumeričkog koda karaktera može da bude poslata štampaču. Tamo ona biva dekodirana, da bi se odredio karakter koji treba da bude odštampan.

Ovaj uređaj šalje i prima podatke serijski. Serijska informacija sa tastature upisuje se (pomjeranjem) u *ulazni registar* INPR a serijska informacija za štampač u *izlazni registar* OUTR (slika 9.5). Dakle, ova dva registra komuniciraju sa uređajem serijski, a oba komuniciraju sa akumulatorom paralelno.

Ulazni registar INPR sastoji se od 8 bita i on čuva ulaznu informaciju. FGI je jedan upravljački flipflop, kažemo da je on jednobitni flag ili zastavica. Kad je nova informacija na raspolaganju, onda je taj flag setovan. Ovaj flag se resetuje kada računar prihvati informaciju. Postoji velika razlika u brzini rada između ulaznog uređaja i računara - računar je brži bar četiri reda veličine. Flag FGI upravo služi da uskladi tu razliku.

Na početku rada računara FGI je nula. Kada se pritisne taster, u INPR



Slika 9.5: Registri za ulaz-izlaz osnovnog računara

se upisuje kod dužine 8 bita i flag se postavlja, tj. dobija vrijednost 1. Sve dok je flag postavljen pritisak na drugi taster ne može da promijeni sadržaj INPR registra. Računar može da provjerava vrijednost flaga. Ako je FGI=1 onda informacija iz INPR može da bude prenesena na paralelni način u akumulator. Nakon toga, FGI se briše, odnosno postavlja na nulu. Nakon toga, nova informacija može da bude proslijeđena u INPR.

Izlazni registar OUTR funkcioniše na sličan način samo što je sada preokrenut smjer kretanja informacije. Dakle, postoji izlazni flag - FGO. Ako je FGO=1, računar može da prenese informaciju iz AC u OUTR. U okviru tog prenošenja računar još i izbriše flag ($FGO \leftarrow 0$). Nakon toga izlazni uređaj prihvata kodiranu informaciju i štampa odgovarajući karakter. Po završetku štampanja flag se setuje ($FGO \leftarrow 1$), što označava da štampač može da prihvati novu informaciju.

Za ovakav način ulazno-izlazne komunikacije kažemo da je *programski upravljan*. To označava da program upravlja prenosom i da se prenos izvodi samo u slučaju pripravnosti spoljašnjeg uređaja. Za ovaj način komunikacije karakteristično je prisustvo programske *petlje*. Petlja se tipično sastoji od dvije naredbe: prve, koja ispituje flag, odnosno pripravnost uređaja, i druge, koja započinje prenos ako je uređaj pripravan ili vraća računar na izvršavanje prve naredbe, ako uređaj nije pripravan. Tako je moguće da računar u cilju, recimo, štampanja jednog karaktera izvrši nekoliko hiljada puta radnju čitanja i provjere vrijednosti flaga FGO. Ovo, očigledno, nije efikasno. Alternativa je moguća samo ako postoji neki način da procesor naznači da treba odštampati neki karakter, koji je smješten na određenom mjestu, a da nakon toga uređaj sam štampa. Uređaj bi tada trebao da o završetku štampanja

obavijesti procesor, koji u međuvremenu izvršava neke druge naredbe.

Realizacija ove ideje moguća je uz određenu dogradnju. Dogradnja je hardverska – uvođenje prekidnog ciklusa i flipflopa IEN, ali i softverska – uvođenje prekidne rutine. Ovo izmjene omogućuju uvođenje mogućnosti *prekida*. Program upravlja ulazno-izlaznom komunikacijom zajedno sa hardverom periferne jedinice. Spoljašnji uređaj prekida trenutnu aktivnost računara i obavještava ga da je spreman za prenošenje nove informacije.

IEN flipflop je flag koji označava da li je dozvoljen prekid (engl. interrupt enable, IEN). Ako je IEN=0 ne može doći do prekida. Računar postavlja IEN na nulu kada ne želi da bude prekidan, a na jedinicu kada želi da dozvoli mogućenost prekida. U konkretnom slučaju, kada je IEN=1, tj. kada je prekid dozvoljen, ako je bar jedan od flagova FGI i FGO setovan, doći će do prekida. Bez obzira do kojeg mjesta u programu se došlo, izvršavanje programa se privremeno *prekida* i počinje izvršavanje jednog drugog programa - *prekidne rutine*. Zadatak prekidne rutine je da obavi radnju koja je unaprijed pripremljena, čim se steknu preduslovi. Nakon završetka njenog rada biće nastavljeno izvršavanje izvornog programa od mjesta na kom je bio prekinut.

Uvođenje prekida stvara mnoga pitanja. Vidimo da kod ovog pristupa moramo sastaviti prekidne rutine, a trebamo izmijeniti i izvorni program da bi on koristio mogućnost prekida, tako da ova dva programa budu usaglašena. Kao prvo, upotreba memorije treba da bude usaglašena. Prilikom prelaska na izvršavanje servisne rutine potrebno je zapamtiti adresu naredbe izvornog programa koja treba da se izvrši nakon prekidne rutine. Ta adresa smješta se u memorijsku lokaciju 0 (podsjetimo se da memorija osnovnog računara ima 4096 lokacija sa adresama od 0 do 4095). Takođe, u memorijsku lokaciju 1 treba upisati naredbu za skok na prvu naredbu prekidne rutine. Stoga adrese 0 i 1 ne smijemo koristiti u programima, one imaju specijalnu namjenu.

$a = 0$	return address
$a = 1$	BUN i

Slika 9.6: Memorijske lokacije za interapte

9.6 Ulazno-izlazne naredbe

Tabela 9.5 sadrži pregled ulazno-izlaznih naredbi osnovnog računara.

Ozn.	Hex zapis	Opis naredbe
INP	F800	Input character to AC (upiši znak u AC) $AC(9-16) \leftarrow INPR,\ FGI \leftarrow 0$
OUT	F400	Output character from AC (izdaj znak iz AC) $OUTR \leftarrow AC(9-16),\ FGO \leftarrow 0$
SKI	F200	Skip on input flag (preskoči ako ulazni flag) ako je $FGI = 1$ onda $PC \leftarrow PC + 1$
SKO	F100	Skip on output flag (preskoči ako izlazni flag) ako je $FGO = 1$ onda $PC \leftarrow PC + 1$
ION	F080	Interrupt on (uključi mogućnost prekida) $IEN \leftarrow 1$
IOF	F040	Interrupt off (isključi mogućnost prekida) $IEN \leftarrow 0$

Tabela 9.5: Ulazno-izlazne naredbe

Naredbe INP i OUT odnose se na nižih osam bitova akumulatora. Dvije uzastopne ulazne informacije čine jednu riječ.

Kod naredbe SKI provjerava se vrijednost flaga FGI. Ako je FGI=1 onda se jedna naredba preskoči i, po pravilu, izvrši naredba za učitavanje (INP). Ako je FGI=0 onda se izvršava sljedeća naredba (ona koja se preskače u slučaju FGI=1), a to je, po pravilu (odluka je, naravno, na programeru), naredba za skok koja vraća računar na ponovno provjeravanje vrijednosti flaga FGI. Slično važi i za naredbu SKO.

Kao što pokazuje tabela, u rezultatu izvršavanja naredbe ION setovaće se flag za dozvolu prekida IEN. Napomenimo da se u realnosti ovo setovanje malo odlaže, tj. ne izvršava se u izvršnom ciklusu – ono će biti izvršeno tek pošto počne izvršavanje iduće naredbe. Zakašnjenje je potrebno jer bi se, kao što ćemo vidjeti, ako smo u obradi interapta i ako je $FGI \vee FGO = 1$, ušlo u novi prekidni ciklus, čime bi se izgubila povratna adresa.

Prekidni ciklus

Sada možemo detaljno opisati prekidni ciklus. U prekidni ciklus se može ući na kraju izvršavanja neke naredbe, tj. kada je $c_2 = 1$ i $t_3 = 1$. Tada se u izvršnom ciklusu odlučuje koji je ciklus sljedeći. Kod definicije izvršnog ciklusa ADD naredbe napisali smo kao posljednju mikro-operaciju $F \leftarrow 0$, što znači prelazak u prijemni ciklus. Međutim, ukoliko uvodimo mogućnost interapta, ovaj red treba definisati na sljedeći način:

c_2t_3 : ako je $IEN \wedge (FGI \vee FGO) = 1$ onda $R \leftarrow 1$, inače $F \leftarrow 0$.

Smisao je sljedeći: ako je uključena mogućnost prekida (IEN=1) i ako je bar jedan od dva uređaja (ulazni ili izlazni) slobodan ($FGI \vee FGO = 1$) onda pređi u prekidni ciklus, inače pređi u prijemni ciklus. Mikro-operacije prekidnog ciklusa prikazane su u tabeli 9.6.

c_3t_0: MBR(5 – 16) ← PC, PC ← 0	PC se prenosi u MBR i briše
c_3t_1: MAR ← PC, PC ← PC + 1	u MAR upisuje se 0, u PC 1
c_3t_2: M ← MBR, IEN ← 0	MBR na lokaciju 0, IEN se briše
c_3t_3: F ← 0, R ← 0	prelazak u prijemni ciklus

Tabela 9.6: Mikro-operacije prekidnog ciklusa

Tokom takta $t_0 = 1$ tekuća vrijednost programskog brojača PC upisuje se u prihvatni registar MBR. Čim se neki sadržaj upisuje u prihvatni registar, znamo da će taj sadržaj biti poslat u memoriju. U ovom taktu se takođe anulira programski brojač. Tokom takta $t_1 = 1$ anulira se adresni registar MAR. Znamo da adresni registar determiniše adresu, kada se nešto šalje u memoriju, ili se iz memorije dobija. Takođe, PC se povećava za jedan. Tokom takta $t_2 = 1$ izvrši se radnja slanja u memoriju i još se resetuje flipflop za dozvolu prekida IEN. Najzad, tokom takta $t_3 = 1$ resetuju se dva flipflopa F i R, što će izazvati prelazak u novi prijemni ciklus. Da rezimiramo, u ovom smo ciklusu zatečenu vrijednost programskog brojača plasirali na adresu $a = 0$, i podesili da je sada PC=1.

Napomenimo još jednom da memorijske lokacije 0 i 1 imaju posebnu namjenu. Lokacija 0 čuva povratnu adresu, koja će biti iskorišćena na kraju prekidne rutine, za nastavak izvršavanja izvornog programa. Prekidni ciklus određuje da se pređe u prijemni ciklus naredbe na lokaciji 1. Zato tamo treba prethodno da bude smještena naredba za bezuslovni skok na uslužni

program. Osim toga, u prekidnom ciklusu isključuju se interapti ($IEN \leftarrow 0$), čime unaprijed bivaju odbačeni (barem u datom trenutku) novi zahtjevi za prekidom.

9.7 O sistemu prekida osnovnog računara

Program koji koristi ulaz ili izlaz može se pisati na nekoliko načina. Na primjer, program može uopšte da ne koristi mogućnost prekida. Ovo je razumno kada ima malo ulaznih podataka i rezultata ili kada program ne može da vrši obradu ako nije saznao ulazne podatke koji su mu potrebni. Program, dalje, može da neka učitavanja i štampanja obavlja bez mogućnosti prekida a neka druga sa prekidima. Konačno, program može kompletnu ulazno-izlaznu komunikaciju riješiti preko sistema prekida, naročito ako je komunikacija obimna, a ima se za cilj i da ulazni i izlazni uređaji budu što je moguće više iskorišćeni.

Izgled prekidne rutine zavisi, naravno, od njene namjene i programera. Mi ćemo ovdje opisati neke tipične faze. To su:

- Da bi se kasnije nastavilo izvršavanje prekinutog programa, zatečene vrijednosti AC i E treba da budu sačuvane, pa ih treba upisati negdje u memoriju.
- Nakon toga može se obaviti zadatak prekidne rutine u užem smislu. Recimo, rutina može prenijeti sadržaj INPR na određeno mjesto.
- Sačuvane vrijednosti AC i E treba vratiti u procesorove registre AC i E.
- Uključiti prekide.
- Izvršiti bezuslovni skok na naredbu čija se adresa čuva u lokaciji 0.

Arhitektura prekida osnovnog računara mogla je biti i drugačija. Na primjer, mogli smo razdvojiti prekide za ulaz i izlaz. To bi zahtijevalo razne dorade i unijelo veću kompleksnost: dva flipflopa, dvije rutine, složeniju logiku,.... Postavlja se zatim i pitanje, šta raditi ukoliko se istovremeno pojave dva zahtjeva za prekid?

Mogućnost prekida otvara još jedan problem. Naime, slično kao za memorijske lokacije 0 i 1, čija je namjena precizno definisana, mora se precizno definisati koje djelove memorije izvorni program koristi za svoje naredbe i podatke, a koje djelove koristi prekidna rutina.

Savremeni računari imaju složene sisteme prikida. Prekidi se obično dijele na *hardverske* i *softverske*. Hardverski prekid generiše se u hardveru i može da dođe od perifernog uređaja (recimo, da označi uspješan završetak nekog postupka ili grešku, kvar uređaja), ukoliko program pokuša da izvrši neku nedozvoljenu radnju (da pristupi zabranjenim djelovima memorije, izvrši neku za njega zabranjenu instrukciju) .... Softverski prekid generiše se od strane samog programa čije izvršavanje je u toku.

Prekidi često imaju prioritet, tj. nisu svi prekidi iste važnosti. Ukoliko se istovremeno pojave dva zahtjeva za prekid, prednost ima onaj koji ima veći prioritet. Kod arhitektura sa manje prekida, o tome koji prekid prihvatiti odlučuje sam procesor. Međutim, u računarskim sistemima često postoji hardverska komponenta kojoj dolaze zahtjevi za prekid iz raznih djelova sistema. Takva komponenta obično se naziva *kontrolerom prekida ili interapt kontrolerom* (engl. *interrupt contoller*). Interapt kontroler upućuje odabrani zahtjev procesoru, zajedno sa tzv. brojem prekida, tj. signalima koji određuju o kom prekidu se radi.

9.8 Osnovni računar naspram PDP-8

Projektovanje hardvera je komplikovan posao. Gleda se da računarski sistem bude efikasan, ali isto tako i da ne bude previše komponenti, kako bi cijena bila manja. Naš model osnovnog računara odgovara stvarnom računaru DEC PDP–8. Iako su ovi računari konceptualno slični, između projektovanja jednog i drugog lako može da se stvori jaz, sa izvjesnim negativnim posljedicama. Da bi se jaz smanjio ili da ga ne bi bilo, pogledajmo šta bi trebalo da se izmijeni ili doda osnovnom računaru.

a) Upravljačka tabla računara.

Na upravljačkoj tabli postoje tri dugmeta. Ako se pritisne na dugme CLEAR onda će se čitava memorija računara izbrisati (svih 4096 adresa). Ako se pritisne na dugme START onda računar počinje da radi. Drugim riječima, počinje da se izvršava program koji je upisan u memoriji, i to od adrese koja je jednaka trenutnom sadržaju programskog brojača PC. Ako se

pritisne na dugme STOP onda će računar prestati da radi. Pored dugmadi, prisutna je i lampa (svjetlosni indikator) RUN: lampa je upaljena ako i samo ako računar radi, tj. ako se izvršavaju naredbe programa. Još, kada se u programu izvrši naredba za zaustavljanje (halt, HLT) onda zazvoni zvonce BELL.

Preporučuje se da se CLEAR pritiska dok je lampa RUN izgašena. Djelovanje dugmeta CLEAR u slučaju RUN = 1 nije definisano. Između dva programa treba pritisnuti CLEAR, osim ako obrada koju treba da izvrši drugi program predstavlja nastavak onoga što je prvi uradio.

b) Podrška za ručno loadovanje.

Postoji i niz od 16 prekidača (svičeva). Prekidač u gornjoj poziciji znači jedinicu, a u donjoj znači nulu. Svi prekidači zajedno čine tzv. SWITCH REGISTER. SR sarađuje sa tri dugmeta LOAD PC, LOAD MAR i DEPOSIT. Ako se pritisne na LOAD PC onda će se desiti PC $\leftarrow$ SR(5–16), tj. sadržaj svič registra preslikaće se u programski brojač. Ako se pritisne na LOAD MAR onda će se desiti MAR $\leftarrow$ SR(5–16). Ako se pritisne na DEPOSIT onda će se desiti M $\leftarrow$ SR, odnosno sadržaj će biti upisan na odgovarajuću lokaciju u memoriji.

Jasno, SR služi prilikom pokretanja računarskog sistema (bootstrap).

c) Izmjene na ulazno–izlaznoj jedinici.

Na ulazno–izlaznoj jedinici (teleprinteru) postoje tri svjetlosna indikatora koji pokazuju kakvo je trenutno stanje flipflopova FGI, FGO i IEN. Pored toga, nalaze se i dva prekidača. Ako je prekidač na kome piše "FGI = 0" u aktivnoj poziciji onda će (bez obzira na druge okolnosti) biti FGI = 0. A ako je taj prekidač u drugoj mogućoj poziciji (neaktivmoj) onda on ne utiče na bilo šta. Dakle, sve dok je prekidač u aktivnoj poziciji, računar će smatrati da ulazni uređaj nije pripravan, da nema šta da saopšti, da nijedan taster nije pritisnut – kao da ulaznog uređaja nema, tj. kao da je otkačen. Slično, ako je prekidač na kome piše "FGO = 0" u aktivnoj poziciji onda će (bez obzira na druge okolnosti) biti FGO = 0. Dakle, sve dok je ovaj prekidač u aktivnoj poziciji, računar će smatrati da je izlazni uređaj zauzet, da je u toku štampanje karaktera.

Napomenimo još da se prilikom startovanja resetuje flipflop za dozvolu prekida IEN.

Dva dugmeta "FGI = 0" i "FGO = 0" služe očito da se ograniče nepotrebna ili neželjena prekidanja programa. Na primjer, program izvršava jedan veliki niz aritmetičkih operacija, a paralelno sa tim želi i da učita puno karaktera preko tastature, koji će mu kasnije trebati. Učitavanje se odvija uz korišćenje mogućnosti prekida. Dok se sve to dešava, štampač je slobodan. Kako štampač nema šta da radi, stalno se javlja, odnosno bespotrebno prekida proces.

d) Izmjene u hardveru.

Umjesto tastature, može kao ulazni uređaj da služi čitač bušene papirne trake. Dobro bi bilo da se doda jedinica spoljašnje memorije, u obliku uređaja za čitanje magnetne trake i pisanje po magnetnoj traci. Trebalo bi dodati i DMA kontroler (direct memory access), radi bolje komunikacije sa magnetnom trakom.

Sa izmjenama i dogradnjama koje su navedene, osnovni računar bi se pretvorio u jedan realan računarski sistem, iako sa današnjeg stanovišta prost. Mini–računari iz šezdesetih i sedamdesetih godina tipično su imali čitač bušene papirne trake kao ulazni uređaj, jedinicu za rad sa magnetnom trakom i još su imali i DMA uređaj.

e) Tehnički parametri računara.

Analizirajući realni računar, možemo pretpostaviti da osnovnom računaru odgovaraju sljedeće tehničke karakteristike. Tempo rada ulaznog uređaja: može da se učita 10 karaktera za jednu sekundu; tempo rada štampača: štampač može za jednu sekundu vremena da odštampa 10 karaktera. Učestanost takta jednaka je 1 MHz; to znači milion otkucaja u sekundi. Sa magnetne trake može da se pročita 15.000 karaktera (bajta) za jednu sekundu, ili da se isto toliko karaktera zapiše.

9.9 Zadaci za vježbu

1. Sastaviti program P_1 na Paskalu koji simulira izvršavanje nekog programa P za osnovni računar. Ulazni podatak programa P_1 jeste sadržaj memorije osnovnog računara (od adrese $a = 0$ do adrese $a = 4095$) u trenutku kada počinje izvršavanje programa P i broj e – ulazna tačka programa P. Tokom svog rada, program P_1 treba da saopštava neke od karakterističnih informacija, kakve su recimo: trenutna vrijednost programskog brojača, kako glasi

tekuća naredba, trenutni sadržaj akumulatora, i slično. Na kraju, P_1 treba da saopšti završnu memorijsku sliku osnovnog računara (kada naiđe "halt" u P). To znači da saopšti sadržaj svih adresa od $a = 0$ do $a = 4095$. Ili (bolje) sadržaj specifikovanog područja, i to svaku specifikovanu adresu u formatu brojne vrijednosti.

Radi jednostavnosti, neka program P ispunjava uslov da u njemu nema naredbi za učitavanje (`inp`), a ni naredbi za štampanje (`out`). Takođe, pretpostavimo da se tokom rada osnovnog računara ne ispostavljaju zahtjevi za prekidom.

2. Sastaviti program P_2 na Paskalu koji simulira izvršavanje nekog programa P za osnovni računar, koji može da sadrži ulazne i izlazne naredbe.

Ulazni uređaj osnovnog računara (tastatura) treba da se modelira kao tastatura (naredba `read` u Paskalu). Izlazni uređaj osnovnog računara (štampač) treba da bude modeliran pomoću ekrana (naredba `write` u Paskalu).

Pretpostavlja se da se osnovnom računaru, tokom njegovog rada, ne ispostavljaju zahtjevi za prekidom.

Glava 10

Softver za osnovni računar

Za funkcionisanje računarskog sistema osim hardvera, potreban je i softver, odnosno programi koje hardver izvršava. U ovom poglavlju razmotrićemo prvo osnovne pojmove o softveru, a zatim navesti primjere programa za osnovni računar.

10.1 Računarski softver

Računarski sistem sastoji se od svog hardverskog dijela i svog softverskog dijela. Hardver čine fizičke komponente i oprema koja je sa njima povezana. Softver čine programi koji su napisani za računar.

Na primjeru osnovnog računara vidjeli smo da računar ima svoj jedan jedini *mašinski jezik*. U mašinskom jeziku sve je izraženo preko nule i jedinice i to je jedini jezik koji računar razumije.

Kao što se može naslutiti iz opisa naredbi osnovnog računara, mašinski jezik teško je razumljiv za čovjeka. Da bi se olakšalo programiranje razvijeni su drugi jezici, raznih nivoa, koji su bliži čovjeku od mašinskog, na kojima je moguće zapisivati programe. Za svaki nemašinski programski jezik, bilo kog nivoa, mora postojati poseban program, *prevodilac*, koji prevodi programe sa tog jezika na mašinski.

Jezik asemblera ili *asemblerski jezik* ili *simbolički jezik* se po svom nivou nalazi između mašinskog jezika i višeg programskog jezika, a znatno je bliži mašinskom jeziku. On se od mašinskog jezika razlikuje uglavnom po dvije stvari. Kao prvo, naredbe se ne zapisuju binarno nego pomoću *skraćenica*.

Tako, recimo, umjesto 0111 0000 0000 0001 piše se HLT (ovo smo detaljno prikazali kod opisa osnovnog računara). Kao drugo, adrese podataka ne zapisuju se binarno nego kao "imena promjenljivih". Na primjer, umjesto konkretne serije od 12 bita piše se A ili B ili slično.

Poseban sistemski program (zvani *asembler*) prevodi bilo koji program napisan na jeziku asemblera na odgovarajući program napisan na mašinskom jeziku.

Viši programski jezici služe da zadamo obradu koju računar treba da ostvari na način koji je blizak čovjeku. Primjeri viših programskih jezika su Fortran, Paskal, C, C++, Java, ...

Iako je i asembler u suštini prevodilac, *prevodiocem* ili *programskim prevodiocem* ili *kompajlerom* naziva se poseban sistemski program koji prevodi (pretvara) tekst našeg programa napisanog na višem programskom jeziku, recimo na Paskalu, u ekvivalentni program na mašinskom jeziku. Ponovimo da je prevođenje potrebno zato što računar može neposredno da izvršava samo program napisan na njegovom mašinskom jeziku.

Dakle, ulazni podatak za prevodilac P jeste tekst nekog programa P_1 (kažemo da je to *izvorni kod* ili source code) na jednom jeziku, a njegov izlazni podatak, tj. rezultat njegovog rada – jeste tekst programa P_2 (*ciljni kod* ili object code) na nekom drugom jeziku, čiji je nivo niži.

Sada ćemo da ilustrujemo ova tri nivoa (viši programski jezik, asembler, mašinski jezik) pomoću jednog primjera. Neka treba da se sastavi program za sabiranje dva cijela broja. Na slici 10.1. prikazano je kako taj program glasi: a) na mašinskom jeziku osnovnog računara; b) na jeziku asemblera osnovnog računara; c) na višem programskom jeziku (na Paskalu).

Pojasnimo prvo sliku 10.1.a). Prva kolona pokazuje memorijsku adresu tj. redni broj memorijske lokacije. Drugim riječima, prva kolona definiše položaj u memoriji svake naredbe i podatka; položaj zapisujemo binarno. Ovdje smo, zbog preglednosti, adrese zapisali bez vodećih nula, recimo adresa 110 trebala bi da bude zapisana kao 0000 0000 0110, i slično. Druga kolona pokazuje sadržaj te memorijske lokacije (memorijske riječi). Naravno, i sadržaj se izražava binarno.

Prva četiri reda odnose se na naredbe, a posljednja tri na podatke.

Očigledno, prva naredba je LDA, jer su prva četiri bita 0010. Adresa na koju se odnosi ova naredba, odnosno adresa podatka kojeg treba prenijeti u

a)

adresa	sadržaj
0	0010 0000 0000 0100
1	0001 0000 0000 0101
10	0011 0000 0000 0110
11	0111 0000 0000 0001
100	0000 0000 0101 0011
101	1111 1111 1110 1001
110	0000 0000 0000 0000

b)

```
      LDA A    donesi u akumulator broj sa adrese A
      ADD B    uvećaj akumulator za broj sa adrese B
      STA C    broj iz akumulatora pošalji na adresu C
      HLT      zaustavi računar
A,    DEC 83
B,    DEC -23
C,    DEC 0
```

c)

```
program progra;
var a,b,c: integer;
begin
a:=83; b:=-23;
c:=a+b
end.
```

Slika 10.1: Mašinski, asembler i Paskal program

akumulator je 0000 0000 0100, ili, kraće zapisano, 100.

Zaključujemo da u prva četiri reda imamo redom sljedeće naredbe (dat je kod naredbe i smisao koda naredbe):

0010 LDA $AC \leftarrow M$
0001 ADD $AC \leftarrow AC + M$
0011 STA $M \leftarrow AC$
0111 0000 0000 0001 HLT zaustavljanje

Posljednja tri reda sadrže podatke - prvi sabirak nalazi se na adresi 100, drugi na adresi 101, a rezultat rada programa (zbir) biće smješten na adresi 110. Vrijednost prvog sabirka jednaka je 83 a drugog -23, budući da se brojevi prikazuju u obliku potpunog komplementa (PK).

Program na mašinskom jeziku sa slike 10.1.a) može da bude unesen u naznačeni dio memorije. On zatim može da bude izvršen startovanjem računara od adrese 0. Računarski hardver će izvršiti ove naredbe i time obaviti predviđeni posao. Ipak, kada se gleda tekst programa, teško se razumije šta će se dešavati tokom izvršavanja.

Slika 10.1.b) sadrži isti program na asemblerskom jeziku. Sada adrese više ne izražavamo u binarnom obliku kakav je recimo 0000 0000 0110 već ih izražavamo simbolički kao recimo C. Oznaka DEC znači dekadno - vidimo da brojeve sada možemo zapisati dekadno, a ne u potpunom komplementu.

Slika 10.1.c) prikazuje ekvivalentan program sastavljen na programskom jeziku Paskal. Paskalov prevodilac pretvara tekst c) u tekst a).

Da se primjer ne bi uveličavao, u programima su izostavljene ulazno–izlazne naredbe. Naime, u slučaju a), učitavao bi se karakter–po–karakter (cifra–po–cifra) ulaznog podatka, tako da bi trebalo sastaviti i dio programa koji od učitanih karaktera stvara prvi sabirak, odnosno drugi sabirak. Slično važi i za štampanje rezultata.

Sistemski i aplikativni softver

Računarski softver sastoji se iz dva dijela – sistemskog i aplikativnog.

Sistemski softver sastoji se od niza izvršnih programa koji treba da učine da korišćenje računara bude lako i efikasno. Tako prevodilac za Paskal pripada sistemskom softveru. I asembleri pripadaju sistemskom softveru. Za program koji pripada sistemskom softveru kaže se da je – sistemski program. Možemo reći da hardver zajedno sa sistemskim softverom čini komercijalni računar.

Aplikativni softver sastoji se od programa koji su napravljeni za neke konkretne ciljeve korisnika računara, i koji se zato nazivaju aplikativnim. Takvi su, na primjer, program za rješavanje sistema linearnih jednačina, jedan ili više programa za rješavanje nekih zadataka iz statistike. Kada imamo više programa koji obavljaju neku složenu funkciju onda kažemo da je to *paket programa.*

Podjela između sistemskog i aplikativnog sosftvera nije stroga, jer često neki programi (ili programski paketi) mogu imati i sistemske i aplikativne komponente.

Među sistemskim softverom posebno mjesto zauzima *operativni sistem.* Operativni sistem jeste skup programa koji omogućavaju korišćenje hardvera. To je posrednik između računarskog hardvera i korisnika računara. Operativni sistem je softver koji je najbliži hardveru. Može se reći da posredstvom operativnog sistema svi ostali programi pristupaju hardveru.

Operativni sistemi dijele se na jednokorisničke i višekorisničke. Kod jednokorisničkih sistema, kao što i sama riječ kaže, samo jedan korisnik može da upotrebljava računar u datom vremenu. Takav sistem je, recimo, DOS, za personalne računare. Primjer višekorisničkog sistema je Unix, za velike računare. Moderni operativni sistemi imaju znatne mogućnosti za rad u mreži. Tako, na primjer, operativni sistemi Windows služe za personalne računare i za mreže personalnih računara.

10.2 Primjeri programa za osnovni računar

Kao primjere programa za osnovni računar navešćemo programe koji koriste ulazne i izlazne uređaje (sa i bez mogućnosti prekida), progam koji izvodi artimetičke operacije i program koji radi sa potprogramima.

10.2.1 Primjeri programa za ulaz i izlaz

U ovoj sekciji navešćemo primjere ulaznog i izlaznog prenošenja kod osnovnog računara.

Generalno, postoje dva načina rada sa ulazno/izlaznim uređajima: programsko prenošenje i prenošenje pomoću sistema prekida. Razmotrimo prvo primjer programskog prenošenja, kod koga računar čeka dok se ne završi ulazna ili izlazna komunikacija.

Dio programa koji realizuje programski kontrolisano učitavanje jednog karaktera može da izgleda ovako (prva kolona je adresa, druga naredba, a treća objašnjenje):

```
240  SKI      preskoči jednu naredbu ako je INPR pun
241  BUN 240  bezuslovni skok na prethodnu naredbu
242  INP      prepiši iz INPR u AC.
```

Slično, programski kontrolisano štampanje:

```
260  SKO      preskoči jednu naredbu ako je OUTR slobodan
261  BUN 260  bezuslovni skok na prethodnu naredbu
262  OUT      prepiši iz AC u OUTR (štampaj karakter).
```

Napišimo sada program koji štampa sadržaj zadate memorijske lokacije (298), a koji ne sadrži mogućnost prekida. Ta lokacija sadrži ukupno dva bajta, odnosno karaktera. Prvo ćemo odštampati prvih osam bita sa desne

strane lokacije 298 a zatim narednih osam. Smatramo da izlazni uređaj (štampač) koristi ASCII kod. Isto važi i za ulazni uređaj (tastaturu).

```
200  IOF       IEN← 0 interrupt off
201  LDA 298   AC ← c(298)
202  OUT       OUTR ← AC(9-16) (štampaj karakter)
203  CIR       desni kružni pomjeraj
204  CIR       desni kružni pomjeraj
205  CIR       desni kružni pomjeraj
206  CIR       desni kružni pomjeraj
207  CIR       desni kružni pomjeraj
208  CIR       desni kružni pomjeraj
209  CIR       desni kružni pomjeraj
210  CIR       desni kružni pomjeraj
211  SKO       prekoči ako izlazni fleg
212  BUN 211   bezuslovni skok na 211
213  OUT       štampaj karakter
214  HLT       zaustavi računar
```

Razmotrimo sada slučaj ulazno-izlaznog prenošenja koje koristi sistem prekida. Sistem prekida računara temelji se na hardveru (koji smo opisali ranije) i na softveru, kojeg ćemo sada definisati. Sporost perifernih uređaja predstavlja problem.

Pretpostavljamo da se karakteri koji stižu sa tastature odmah (dakle, bez čekanja) upisuju u dio memorije koji je za to namijenjen. Za taj dio memorije kažemo da je *ulazni bafer*. Vidjećemo da će ulazni bafer biti ogranizovan kao kružni bafer, odnosno kružni red. Naš program će preuzimati podatke iz bafera.

Slične okolnosti važe za izlazno prenošenje. Program upisuje podatke u izlazni bafer, a iz njega se oni šalju štampaču, po mjeri spremnosti štampača, odnosno tempom koji štampač može da prati.

Ključnu ulogu ima sistemska rutina R, rutina za obradu prekida (engl. interrupt service routine, ISR). Neka program zauzima adrese a u granicama $100 \leq a \leq 199$ za svoje naredbe i podatke. Pri tome adrese u opsezima $100 \leq a \leq 109$ i $110 \leq a \leq 119$ rezervisane su za ulazni, odnosno izlazni bafer, respektivno.

Neka $a = 120$ bude ulazna tačka rutine R, tj. adresa od koje treba da počinje njeno izvršavanje. Adresa $a = 0$ čuva vrijednost koju je programski

brojač PC imao u trenutku skoka na rutinu. Dakle, $a = 0$ čuva *povratnu adresu* a rutina R treba da na kraju svog rada obezbijedi da se izvršavanje programa P nastavi upravo od te adrese. Adresa $a = 1$ sadrži naredbu skoka na rutinu. Kao što smo ranije istakli, mjesta $a = 0$ i $a = 1$ su namjenska mjesta, koja smo definisali pri definiciji hardvera. Da asihroni događaj (skok na prekidnu rutinu) ne bi poremetio rad programa, osim programskog brojača i vrijednosti registara AC i E moraju biti sačuvane. Njih ćemo, prilikom skoka na prekidnu rutinu, sačuvati na lokacijama $a = 198$ i $a = 199$. Prilikom povratka iz rutine u registre AC i E biće upisane sačuvane vrijednosti.

Za ulazni bafer važi sljedeće. Adresa $a = 100$ sadrži mjesto gdje se nalazi prvi (po logičkom redosljedu) član bafera, *ib* ili input begin. Na adresi $a = 101$ nalazi se input number, *in*, odnosno broj članova bafera. Dakle, $c(100) = ib$, $c(101) = in$. Sami članovi bafera su na 8 lokacija, od $a = 102$ do $a = 109$. To znači da je $0 \leq in \leq 8$. Ako je $in = 0$, bafer je prazan, a ako je $in = 8$ - bafer je pun. Uvijek je $102 \leq ib \leq 109$, osim kada je bafer prazan. Podaci se u bafer unose po redu, nakon upisa u lokaciju 105, upisuje se u 106. Bafer je kružan, što znači da se nakon 109 upisuje u 102.

Prvi član koji je upisan u bafer će prvi iz njega i izaći. Ako je bafer prazan, neka bude $ib = 0$. Prekidna rutina R puni bafer, dakle stavlja elemente u njega, a program P ga prazni (uzima elemente iz njega). Ako je, recimo, $ib = 102$ i $in = 4$, to znači da u baferu imamo četiri člana i da su oni, redom, na adresama 102, 103, 104 i 105. Slično, ako je $ib = 108$ i $in = 4$, imamo četiri člana na adresama 108, 109, 102 i 103. Prilikom svakog upisivanja jednog člana u bafer treba ažurirati vrijednosti *ib* i *in*. Slično važi i prilikom uzimanja člana iz bafera. Bafer je organizovan kružno da bi se njegov raspoloživi prostor (8 lokacija) maksimalno koristio a da ne moramo premještati elemente prilikom dodavanja ili oduzimanja članova.

Za izlazni bafer važe slične okolnosti. Njemu je dato 10 lokacija $110 \leq a \leq 119$, od čega prve dvije služe za evidenciju: *ob* – output begin (adresa prvog člana) i output number, *on* (tekući broj članova). Program P stavlja u izlazni bafer karaktere koji treba da budu odšampani, a rutina R uzima iz bafera i šalje štampaču.

Sada možemo detaljnije opisati radnje koje rutina R obavlja, mada je nećemo pisati kompletno, kao program. Možemo razdvojiti šest koraka a)-f) rutine R.

a) Zatečene vrijednosti AC i E treba da budu sačuvane, da bi se kasnije

izvršavanje programa P obnovilo u okolnostima koje se nisu izmijenile. Dakle, zapisujemo vrijednosti AC i E negdje u memoriju.

STA 198	c(198) ← AC
CLA	AC ← 0, izbriši AC
CIL	lijevi kružni pomjeraj E i AC, tako da a_{16} ←E
STA 199	c(199) ← AC, flipflop E je sačuvan

Koraci b) i c) predstavljaju zadatak rutine R u užem smislu riječi. Napomenimo da se znak * koristi da označi da se koristi indirektno adresiranje.

b) Ako je FGI=0 onda pređi na c). Ako je $in = 8$, ulazni bafer je pun, pa izvrši naredbu za zvučni signal i pređi na c). Vidimo da ovdje uvodimo novu mašinsku naredbu - naredbu za zvučni signal. Obično se ona naziva "BEEP".

Ako je FGI=1, onda ulazni uređaj ima nešto da kaže i ako je još $in < 8$ (ulazni bafer nije pun), tada procedura R preuzima jedan karakter. Nećemo opisivati sve naredbe ove faze, već ćemo razmotriti dva posebna slučaja. Ako je bilo $in = 0$ (bafer je prazan) onda, pod pretpostavkom da se na adresi 197 nalazi odranije upisana brojna vrijednost 102, treba uraditi:

LDA 197	AC← c(197) =102
STA 100	c(100) =AC, *ib* je podešen
CLA	AC← 0
INC	AC← AC+1
STA 101	c(101) =AC, *in* je podešen
INP	AC ← INPR, učitavanje karaktera u AC
STA 102	c(102) ← AC, karakter je stavljen u bafer,

Radi ilustracije, razmotrimo još slučaj kada je $in > 0$, dakle, bafer nije prazan a $ib + in < 109$. Dakle, nećemo pisati program koji koristi kružno svojstvo.

LDA 101	AC← c(101), AC ← *in*
INC	AC← AC+1
STA 101	c(101) =AC, *in* je podešen
ADD 100	AC← AC+c(100)
STA 196	c(196)← AC
INP	AC ← INPR, učitavanje karaktera u AC
STA* 196	c(c(196)) ← AC, karakter je stavljen u bafer.

c) Ako je FGO=0 onda pređi na d).

Ako je FGO=1 (štampač je spreman) i $on = 0$ (izlazni bafer je prazan) onda pređi na d).

Ako je FGO=1 i $on > 0$, onda pošalji odgovarajući karakter štampaču (naredbom OUT), zatim ažuriraj *ob* i *on* i pređi na korak d).

d) Vratiti dvije spašene vrijednosti (iz 198 i 199) u AC i E. Prvo se vraća E a onda AC.

e) Naredba ION. Efekat ove naredbe je IEN← 1, da bi ostale iste okolnosti jer je u trenutku prelaska na rutinu, očigledno, bilo IEN=1.

f) Naredba BUN* 0. Efekat ove naredbe je PC← c(0), čime se obnavlja izvršavanje programa P.

Uočimo da program P nema naredbu za ulaz INP, nego umjesto nje koristi naredbu LDA za preuzimanje iz ulaznog bafera u AC (naravno, ako ima šta da preuzme). Slično, P nema ni naredbu za izlaz OUT, nego koristi naredbu STA za stavljanje u izlazni bafer podataka koje će rutina odštampati.

Sada možemo da napišemo program koji koristi mogućnost prekida, odnosno navedenu prekidnu rutinu.

Program treba da odštampa desnih (donjih, nižih) 8 bita lokacija 296 do 299 (ukupno četiri lokacije), a zatim izvrši veći niz aritmetičkih operacija.

Umjesto da sam štampa i čeka završetak štampanja za svaki karakter (što je jako sporo), program će koristiti mogućnost prekida, upisaće podatke u izlazni bafer i uposliti prekidnu rutinu da štampa.

Naravno, za izvršavanje ovog programa potrebno je naznačene naredbe učitati na naznačene adrese. Posebno, u lokacije 294 i 295 treba upisati brojne vrijednosti, dekadno 112 i 4. Osim toga, prekidna rutina mora biti učitana (ona zauzima lokacije $100 \leq a \leq 199$). Program treba startovati sa adrese 200.

1	BUN 120	skok na prekidnu rutinu
200	IOF	IEN ← 0, interrupt off
201	CLA	AC ← 0, clear AC
202	STA 100	c(100) ← AC, ib← 0
203	STA 101	c(101) ← AC, in← 0 (ulazni bafer je prazan)
204	LDA 296	AC← c(296), počinje upisivanje u izlazni bafer
205	STA 112	c(112) ← AC
206	LDA 297	AC← c(297)
207	STA 113	c(113) ← AC
208	LDA 298	AC← c(298)
209	STA 114	c(114) ← AC
210	LDA 299	AC← c(299)
211	STA 115	c(115) ← AC, četiri vrijednosti su upisane u bafer
212	LDA 294	AC← c(294)=112
213	STA 110	c(110) ← AC, *ob* ← 112, ažuriranje
214	LDA 295	AC← c(295)=4
215	STA 111	c(111) ← AC, *on* ← 4, ažuriranje
216	ION	IEN ← 1, interrupt on
217	ADD 254	počinju aritmetičke i logičke operacije
218	ADD 253	
219	AND 252	
220	AND 251	
...	...	...
294	DEC 112	ova lokacija sadrži brojnu vrijednost, dekadno 112
295	DEC 4	ova lokacija sadrži brojnu vrijednost, dekadno 4.

Naš osnovni računar ima dva periferna uređaja i za svaki od njih po jedan flag - FGI i FGO. Ukoliko bi bilo više uređaja, bilo bi i više flagova. Tokom prekidnog ciklusa neke naredbe izvršiće se skok na R, ako je makar jedan flag podignut i ako su interapti dozvoljeni (IEN=1).

Flagovi informišu procesor koji uređaj želi prekid. Kad ima više izvorišta prekida u sistemu moguće je da u istom trenutku ima više zahtjeva. Zbog toga, prekidna rutina mora u tim situacijama da izdvoji jedan zahtjev i opslući samo njega.

Tokom rada prekidne rutine mi smo isključivali interapte (postavljali smo IEN na nulu). Kod nekih računara moguće je i da prekidna rutina bude prekinuta. Prilikom prekidanja opet je potrebno sačuvati procesorove registre i povratnu adresu. Sada ove podatke nije moguće sačuvati na fiksiranim

lokacijama, kao što je urađeno u slučaju osnovnog računara, jer su te lokacije zauzete. U takvim arhitekturama podaci koji se čuvaju smještaju se na sistemski stek, a kod povratka skidaju sa njega. Ukupna šema tada je nešto složenija od ovdje izložene.

10.2.2 Primjer aritmetičkog programa

Sastavimo program za osnovni računar koji računa vrijednosti izraza $y = 2a+b+1$. Pretpostavimo da se program učitava u memoriju počev od adrese 200, do adrese 205. Izvršavanje počinje od lokacije 200. Na lokacijama 210-212 su podaci - a, b i y.

200	LDA 210	20D2	AC← c(210)=a
201	CIL	7040	left shift AC
202	ADD 211	10D3	AC← AC+c(211)=AC+b
203	INC	7020	AC← AC+1 (increment AC)
204	STA 212	30D4	$y \leftarrow$ AC
205	HLT	7001	halt, stop the computer
210	DEC 300	012C	$a = 300$
211	DEC 44	002C	$b = 44$
212	DEC 0	0000	$y = 0$

Vidimo da je $a = 300$ i $b = 44$. Na kraju će biti $y = 645$, odnosno sadržaj lokacije 212 biće 0285 u heksadekadnom zapisu, tj. 0000001010000101 u binarnom. Program smo prikazali u simboličkom jeziku, gdje DEC x znači 'sadržaj je dekadno x.' Isti program prikazan je i u heksadekadnom obliku, iz kojeg se lako vidi stvarni binarni sadržaj lokacija, odnosno program na mašinskom jeziku.

Za vježbu možete uraditi nekoliko primjera navedenih niže. Potrebno je prvo napraviti raspored za memoriju, tj. napraviti plan upotrebe memorije - odrediti koje lokacije služe za naredbe a koje za promjenljive. Nakon toga treba napisati program i naznačiti lokaciju sa koje počinje izvršavanje.

Primjer 1. Napisati program koji računa vrijednost izraza:

$$y = \begin{cases} a+b+10, & \text{ako je } c < 0 \\ a^2+b^2+100, & \text{ako je } c \geq 0. \end{cases}$$

U ovom programu potrebno je koristiti naredbu skoka.

Primjer 2. Napisati program koji računa vrijednost izraza:

$$y = \sum_{k=a}^{b} k,$$

gdje su a i b ulazni podaci.

Primjer 3. Sastaviti program za osnovni računar koji ilustruje rad sa nizom brojnih vrijednosti. Recimo, sa nizom od 10 brojeva $a_1, a_2, \ldots, a_{10}$ koji zauzimaju redom adrese 1001, 1002, ..., 1010. Neka program učitava vrijednosti članova niza u memoriju, na lokacije 1001-1010. Program, na primjer, treba nakon toga da izračuna sumu

$$y = \sum_{k=1}^{10} a_k.$$

Uputstvo. Treba koristiti naredbe STA sa indirektnom adresom. Na simboličkom jeziku to je naredba STA*. Na primjer, može se koristiti STA* 1000, gdje kroz adresu 1000 prolaze vrijednosti 1001, 1002, ... 1010. Isto tako, treba koristiti i naredbu LDA sa indirektnom adresom.

10.2.3 Primjer rada sa potprogramom

Sastavimo program za osnovni računar koji računa vrijednost izraza

$$y = (a/16 + b)/16,$$

korišćenjem potprograma. Sam program i podaci su na lokacijama 200-208. Izvršavanje počinje od adrese 200. Pretpostavimo da se od adrese 250 do adrese 255 nalazi potprogram, koji vrši radnju AC← AC/16.

Povratna adresa, tj. adresa naredbe koju treba izvršiti nakon završetka potprograma, nalazi se na lokaciji 250. Zato je posljednja naredba potprograma BUN* 250. Za pozivanje potprograma (što se obavlja dva puta) koristi se naredba BSA 250. Shodno njenoj definiciji, ona smješta sadržaj programskog brojača uvećanog za 1 na naznačenu lokaciju (smješta adresu naredbe koju treba izvršiti nakon završetka potprograma na lokaciju 250) i izvodi skok na lokaciju 251.

200	LDA 206	AC← c(206)=a
201	BSA 250	call, c(250)←202 and PC←251
202	ADD 207	AC← AC+c(207)=AC+b
203	BSA 250	call, c(250)←204 and PC←251
204	STA 208	c(208)← AC (y←AC)
205	HLT	halt, stop the computer
206	DEC 900	$a = 900$
207	DEC 800	$b = 800$
208	DEC 0	$y = 0$ (na kraju y=53)
250	DEC 0	mjesto za povratnu adresu
251	CIR	desno pomjeranje (dijeljenje sa 2)
252	CIR	desno pomjeranje (dijeljenje sa 2)
253	CIR	desno pomjeranje (dijeljenje sa 2)
254	CIR	desno pomjeranje (dijeljenje sa 2)
255	BUN* 250	return, PC← c(250), skok na c(250)

U opštem slučaju, i potprogram ima svoj radni prostor. U našem primjeru program ima svoj radni prostor 206-208, a potprogram ga nema (osim 250 za povratnu adresu).

Napomenimo i da razmjena parametara između glavnog programa i potprograma ne mora da se vrši kao u našem primjeru, preko akumulatora. Razmjenu je moguće obaviti tako što se izdvoji nekoliko adresa u okviru potprograma ili putem nekoliko fiksiranih adresa (neke zajedničke zone, čije su adrese apsolutne).

10.3 Loader

Loader je program koji učitava druge programe u memoriju. U zavisnosti od toga kako je program napisan, možemo posmatrati apsolutne loadere i loadere sa relokacijom.

10.3.1 Apsolutni loader

Ukoliko u trenutku prevođenja programa znamo fizičke (apsolutne) adrese tada koristimo *apsolutni loader* za učitavanje programa u memoriju. Ovaj loader, u suštini, bukvalno kopira program sa ulaza (ili sa nekog spoljašnjeg uređaja, recimo diska) u memoriju. Kako su u programu naznačene sve ap-

solutne adrese, loader ne mijenja program. Nakon učitavanja, loader predaje upravljanje učitanom programu, tj. počinje izvršavanje učitanog programa.

Pretpostavimo da loader zauzima lokacije od 0-99, i da je taj prostor već popunjen u trenutku startovanja računara, odnosno da učitavanje loadera nije programsko. Vidimo da loader koristi i posebne lokacije 0 i 1. Tokom njegovog rada nećemo koristiti mogućnost prekida, dakle biće IEN=0 u trenutku uključivanja računara. Pretpostavimo još da je PC=0 u trenutku uključivanja računara. Poslužićemo se primjerom aritmetičkog programa iz prethodne sekcije, da opišemo rad apsolutnog loadera. Za loadovanje (učitavanje) aritmetičkog programa ulazni podaci loadera su adrese njegove dvije cjeline (naredbe i podaci), kao i početna adresa od koje se izvršava program. Dakle, ulazni podaci loadera su: $200 = (C8)_{16}$, $210 = (D2)_{16}$, $200 = (C8)_{16}$. Svi potrebni podaci za rad loadera mogu se definisati pomoću sljedećeg niza karaktera:

00C8 20D2 7040 10D3 7020 30D4 7001 7777 00D2 012C 002C 0000

7777 FFFF 00C8.

Vidimo da loader redom čita cjeline, on čita u grupama od po 4 karaktera. Prva cjelina definiše mjesto gdje počinje popunjavanje, a iduće cjeline definišu sami sadržaj koji treba da bude upisan u memoriju. Takve cjeline se ponavljaju sve dok se ne pročita cjelina 7777. Ova cjelina označava da je završeno popunjavanje jednog odsječka u memoriji. Na primjer, prvi odsječak počinje na adresi $200 = (00C8)_{16}$, i loader redom popunjava memorijske lokacije počevši od 200: na adresu 200 upisuje 20D2, na adresu 201 upisuje 7040, ..., na adresu 205 upisuje 7001, čime završava učitavanje prvog odsječka. Naredni odsječak učitava se počev od adrese $210 = (00D2)_{16}$.

Vidimo da loader može da učitava više odsječaka. Cjelina FFFF označava kraj, tj. da nema više odsječaka koje treba učitati. Jedino još preostaje da se ukaže odakle treba da počne izvršavanje, odnosno da se specificira *ulazna tačka* aritmetičkog programa. To radi posljednja četvorka, ona nakon FFFF - u našem slučaju to je 00C8. Dakle, loader na kraju svog rada izvršava jedan bezuslovni skok.

Opišimo, na neformalnom jeziku bliskom Paskalu, kako izgleda loader. Ovdje l znači *load point* (tačka punjenja, mjesto na koje treba učitati naredbu).

1. *read four*; *if four* $=$ *FFFF then goto* 3; $l \leftarrow$ *four*

2. *read four*; *if four* $=$ 7777 *then goto* 1; $c(l) \leftarrow$ *four*; $l \leftarrow l+1$; *goto*2;

3. *read four*; $PC \leftarrow$ *four*;

Naravno, umjesto neformalnog programa možemo napisati i realni prgram. Znamo da učitati karakter preko ulaznog uređaja znači upisati odgovarajuću brojnu vrijednost u ulazni registar INPR, imajući u vidu ASCII korespodenciju između karaktera i brojne vrijednosti.

Uočimo da se, nakon završetka rada loadera, prostor 0-99 može slobodno koristiti, tj. može se prebrisati.

10.3.2 Loader koji vrši relokaciju

Za korišćenje apsolutnog loadera potrebno je unaprijed, u vrijeme pisanja programa, znati lokaciju u memoriji na kojoj će se program nalaziti. Ova činjenica predstavlja veliko ograničenje. Savremeni računari omogućuju da se u memoriji nalazi više od jednog programa i u takvim arhitekturama ne može se unaprijed znati dio memorije u koji se smješta program. Konkretni detalji zavise od načina organizacije memorije, od broja i veličine već učitanih programa, itd.

U takvim situacijama, program se piše kao da počinje sa adrese 0. Kažemo da je program napisan u *relativnim adresama*. U trenutku učitavanja svaka adresa koja se pominje u programu povećava se za iznos pomjeranja Δ, koji se određuje prije početka učitavanja. Kažemo da se time izvodi *relokacija*, a loader pripada klasi loadera koji vrše relokaciju.

U našem primjeru osnovnog računara, relokacija se izvodi putem uvećavanja za Δ adresnog polja 5-16 svake naredbe prve vrste. Konkretno, dodati Δ naredbi *ADD a* znači podesiti naredbu tako da se drugi sabirak ne uzima sa lokacije a već $\Delta + a$. Slično, dodati Δ naredbi *BUNa* znači podesiti naredbu tako da se vrši bezuslovni skok na $\Delta + a$, a ne na a.

Posebno korisna upotreba relokacije je kod potprograma. Kad pišemo potprogram teško je znati koliki je glavni program, koliko ima potprograma i koliki su, kolika je eventualna zajednička zona memorije, ... Dakle, više stvari utiče na to koji će dio memorije pripasti potprogramu. Jedino loader sa relokacijom može pravilno podesiti adrese tako da potprogram obavi posao za koji je namijenjen.

Kako treba da budu pripremljeni ulazni podaci za loader sa relokacijom? Za apsolutni loader, pojedini odsječak zadavao se pomoću sljedećeg niza cjelina:

$$l\ four\ four \ldots four\ 7777.$$

Za relativni loader niz cjelina ima jednu cjelinu više. Za jedan odsječak ulaz glasi:

$$l\ \Delta\ four\ four \ldots four\ 7777.$$

Neka kao primjer opet posluži raniji aritmetički program i neka su Δ za njegove cjeline jednaki $\Delta = (0010)_{16} = 16$ i $\Delta = (0000)_{16} = 0$. Takođe, neka ulazni podaci za relativni loader glase

00C8 0010 20D2 ... 7001 7777 00D2 0000 012C 002C 0000 7777

FFFF 00C8.

Razlika u odnosu na prethodni slučaj loadovanja je sljedeća: na mjesta 200-205 upisaće se 20E2... 7001. Preciznije, umjesto ranijeg:

```
200  LDA 210
201  CIL
202  ADD 211
203  INC
204  STA 212
205  HLT
```

biće upisano

```
200  LDA 226
201  CIL
202  ADD 227
203  INC
204  STA 228
205  HLT.
```

Na nestrogom jeziku, jedan dio programa relativni loader glasi kako slijedi (ovdje je $four = x_1x_2x_3x_4$):

$$\text{if } x_1 \neq 7 \ \&\ x_1 \neq F \text{ then } c(l) \leftarrow four + \Delta \text{ else } c(l) \leftarrow four.$$

Naravno, korisno je i da se kompletan program napiše na jeziku osnovnog računara.

10.3.3 Bootstrap looder

U ovoj sekciji kazaćemo nekoliko riječi o programu koji pokreće računar. Takav program naziva se *startnim* ili *inicijalnim* loaderom (engl. bootstrap loader). Ovaj program djeluje kada je memorija prazna, tj. kada je u njoj samo on prisutan. Bootstrap loader se čuva u ROM-u i počinje izvršavanje kada se pritisne dugme za uključivanje računara. On, najčešće, samo učitava jedan veći program i započinje njegovo izvršavanje. Taj drugi program provjerava periferne uređaje, inicijalizuje registre, vrši određena izračunavanja, loaduje, itd.

Vidimo da se startovanje računara izvodi u nekoliko faza. U slučaju DOS, bootstrap loader učitava tzv. boot block (prvi sektor na disku) i započinje njegovo izvršavanje.

10.4 Asembler

Sa sintaksom asemblerskog jezika osnovnog računara upoznali smo se kroz prethodne primjere. Ovdje ćemo razmotriti neke dodatne elemente važnog sistemskog programa za osnovni računar - asemblerskog prevodioca, koji se, kao što smo istakli, i sam naziva asemblerom. Zadatak asemblera jeste da generiše mašinski kod (program na mašinskom jeziku), označimo ga sa P_2, koji odgovara ulaznom podatku P_1. Program P_1 izražen je na jeziku asemblera.

Počinjemo sa primjerom programa P_1. Sastavimo program na jeziku asemblera osnovnog računara za računanje vrijednosti izraza $y = 2a + b + 1$.

```
     ORG 200
     LDA A
     CIL
     ADD B
     INC
     STA C
     HLT
A,   DEC 300
B,   DEC 44
C,   DEC 0
     END
```

Pojedina naredba jezika asemblera prikazuje se u posebnom redu. Naredba može da bude pseudo-naredba (asemblerska direktiva) ili obična naredba.

Postoje četiri vrste pseudo-naredbi ORG, END, DEC i HEX. ORG n govori da program P_2 treba da zauzima adrese $n, n+1$, itd; ORG je skraćenica od origin (izvor, početak). END prosto označava kraj teksta programa P_1. Naredba DEC n govori da na odgovarajućem mjestu u P_2 treba da piše sadržaj koji odgovara $(n)_{10}$. HEX ima slično značenje, samo što se sada upisuje $(n)_{16}$.

Običnih naredbi ima 25 vrsta, koliko ima i vrsta mašinskih naredbi. Pojedina naredba jezika asemblera sastoji se od tri polja:

label, opcode address.

U primjeru, u naredbi A, DEC 300 (čiji je smisao $a \leftarrow 300$) A je *labela*; odnosno *simboličko ime* adrese 206. Ako se u programu P_1 pojavljuje naredba, recimo, BUN MJESTO (bezuslovni skok), onda će se u P_1 pojaviti i naredba čija je labela MJESTO. Dio naredbe label može i da odsustvuje.

Drugo polje, opcode, označava *kod operacije*. Postoje mogućnosti AND, AND*, ADD, ADD*, ..., CLA, CLE, ..., INP, OUT, ... U slučajevima CLA, CLE, ... (naredbi druge vrste) i INP, OUT, ... (naredbi treće vrste) odsustvuje treće polje address. Primjeri naredbi koje imaju treće polje su AND A, AND* A, BUN MJESTO, BUN* MJESTO. Ovdje su A i MJESTO mogući primjeri tzv. *simboličkih adresa.*

Kao rezultat rada asemblera dobijamo sledeći program, koji počinje od adrese 200 (odnosno $(0C8)_{16}$), zapisan u heksadekadnom i u binarnom obliku:

adresa	hex	bin
0C8	20CE	0010000011001110
0C9	7040	0111000001000000
0CA	10CF	0001000011001111
0CB	7020	0111000000100000
0CC	30D0	0011000011010000
0CD	7001	0111000000000001
0CE	012C	0000000100101100
0CF	002C	0000000000101100
0DO	0000	0000000000000000

Ulazni podatak asemblera je tekst programa P_1. Asembler može preuzeti ulaz sa ulaznog uređaja (tastature) ili sa spoljašnje memorije, na primjer iz fajla primjer.asm. Izlaz može da saopštava preko izlaznog uređaja ili da ga upisuje na spoljašnju memoriju, recimo u fajl primjer.exe.

Ovakvi asembleri se konstruišu tako da rade kroz *dva prolaza.* Prvi prolaz analizira tekst asemblerskog programa i pravi tabelu simbola koji se pojavljuju u tekstu, dok drugi generiše mašinski program, koristeći tabelu simbola.

10.4.1 Sistemski softver osnovnog računara

Da rezimiramo, sačinimo spisak sistemskih programa koji su napisani za osnovni računar ili bi trebalo da budu napisani. Neki od njih su apsolutno neophodni za funkcionisanje samog računara, a drugi su potrebni zbog njegovog lakšeg korišćenja i programiranja.

a) Bootstrap loader. Znamo da bez ovog programa ne može da se počne. Jedna mogućnost za unos ovog programa u računar je da se on drži negdje zapisan, pa da se unosi ručno preko svičeva. Zato je bolje da program bude što je moguće kraći. Druga mogućnost: program se drži na posebnom ulazu, recimo specijalnoj bušenoj papirnoj traci. Predviđen je mehanizam da se na polu–automatski sadržaj trake preslika u odgovarajuće područje u memoriji. Treća mogućnost bila bi da se drži u ROM–u, ili nekom malom RAM-u (kao što je to slučaj kod savremenih računara).

b) Apsolutni loader zauzima područje 2–99. Predviđeno je da stalno bude prisutan u memoriji. Zato se kaže da ovaj program pripada tzv. rezidentnom dijelu operativnog sistema.

c) Loader koji vrši relokaciju.

d) Rutina za obradu prekida zauzima područje 100–199. Predviđeno je da ona bude stalno prisutna u memoriji. Zato se kaže da i ona pripada rezidentnom dijelu operativnog sistema.

e) Asembler.

f) Program koji vrši interpretaciju drugih programa. On može zauzimati, recimo, lokacije od 3000–4095. Znamo da nije lako napisati program da odmah radi kako treba. Interpreter polako pušta da se izvodi program u kome

ima grešaka, u cilju fiksiranja grešaka. On prati dešavanja i šalje obavještenja. Program koji analiziramo smije da zauzima samo adrese do 2999.

g) Potprogram za množenje dva broja. Znamo da operacija $y = a \cdot b$ nije hardverski data, u slučaju osnovnog računara. Ono što hardver nije riješio, prepušta se softveru. Lako je razumjeti da je operacija $y = a \cdot b$ često potrebna u mnogim zadacima.

h) Potprogrami za podršku ulazu i izlazu. Često treba da se učita niz dekadnih cifara i da se obrazuje brojna vrijednost. Slično, često treba da sadržaj zadate adrese bude odštampan u formatu brojne vrijednosti.

10.5 Zadaci za vježbu

1. Neka su ulazni podaci programa tri cijela broja a, b i c. Sastaviti program za osnovni računar u kome se računaju cijeli brojevi $y_1 = a + 3b + 5c$ i $y_2 = -a + b/2 + c/4$ i štampaju se rezultati y_1 i y_2. Predvidjeti na kojim mjestima u memoriji će se držati tri ulazna podatka. Može se pretpostaviti da se oni već nalaze na tim mjestima, u trenutku kada počinje rad programa. Isto tako, predvidjeti adrese za dva rezultata. Može se izostaviti njihovo štampanje. Takođe, sami definišite položaj u memoriji naredbi i promjenljivih, kao i ulaznu tačku programa.

2. Neka su ulazni podaci programa tri broja a, b i c. Sastaviti program u kome se računa i štampa y:

$$y = \begin{cases} |a| + |b|, & \text{ako je } c \text{ paran broj;} \\ |a+1| + |b+1|, & \text{ako je } c \text{ neparan broj.} \end{cases}$$

3. Sastaviti program za osnovni računar čiji su ulazni podaci a i b ($a \leq b$), a rezultat je $y = a + (a+1) + (a+2) + \ldots + b$. U programu, neka se y računa pomoću postepenog sabiranja, odnosno neka se ne koristi formula za zbir prvih n prirodnih brojeva.

4. Sastaviti program čiji su ulazni podaci a i b ($a \leq b$), a rezultat je $y = a + 2(a+1) + 4(a+2) + \ldots + 2^{b-a} \cdot b$.

5. Potprogram za množenje. Neka je $0 \leq n_1 \leq 2^{15} - 1$, $0 \leq n_2 \leq 2^{15} - 1$, $0 \leq n_3 \leq 2^{15} - 1$ i $n_3 = n_1 \cdot n_2$. Sastaviti potprogram čiji su ulazni parametri n_1 i n_2 i čiji je izlazni parametar $n_3 = n_1 \cdot n_2$. Realizovati množenje kao kombinaciju šiftovanja i sabiranja. Sastaviti i glavni program, u kome se nekoliko puta poziva potprogram.

6. Neka je $1 \leq n \leq 10$ i $y = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$ (skalarni proizvod dva niza). Neka se ulazni podatak a_k nalazi na adresi $200 + k$ i neka se ulazni podatak b_k nalazi na adresi $210 + k$, gdje je $1 \leq k \leq n$. Ulazni podatak n nalazi se na adresi 221. Rezultat y treba ostaviti na adresi 222. Sastaviti program za računanje y, s tim da zauzima (zajedno sa radnim prostorom) lokacije od 223 pa naprijed. Neka 223 bude ulazna tačka.

7. Razmotrimo niz brojeva $a_1, a_2, \ldots, a_n$, gdje je $1 \leq n \leq 95$. Izračunati: y_1 – najmanji član niza, y_2 – najveći član niza, y_3 – zbir svih članova niza i y_4 – broj pozitivnih članova. Predviđene su adrese 201–295 za članove i 296–300 za n, y_1, y_2, y_3 i y_4.

8. Neka je specifikovana jedna adresa a ($0 \leq a \leq 4095$). Sastaviti potprogram koji će da odštampa sadržaj te adrese na najprostiji mogući način, odnosno kao niz od 16 binarnih cifara. Uputstvo. Vidjeti u ASCII tabeli koji broj odgovara nuli odnosno jedinici, da bi se znalo koji broj da se šalje štampaču. Prije nego što se nešto pošalje štampaču, provjeriti da li je štampač pripravan (ispitati njegov flag).

9. Sastaviti potprogram koji će da odštampa sadržaj specifikovane adrese u formatu broja, i to u dekadnom brojnom sistemu. Uputstvo. Ako je $n < 0$ onda pošalji karakter "−" štampaču i $n \leftarrow -n$. Ponavljati $n \leftarrow n - 10000$ dok se ne odredi cifra desetina hiljada c_4. Poslati c_4 štampaču (osim ako je $c_4 = 0$). Slično odrediti cifru hiljada c_3 i odštampati c_3 (osim ako je $c_4 = c_3 = 0$). Još c_2, c_1 i c_0.

10. Potprogram za učitavanje trocifrenog pozitivnog broja. Kao ulazni podaci, preko tastature se zadaju jedna do tri dekadne cifre, a zatim karakter "blanko". Potprogram treba da obrazuje odgovarajući broj n ($0 \leq n \leq 999$) i onda da upiše n na adresu koja je specifikovana.

Glava 11

Procesor Intel 8086

11.1 Uvod

U prethodnim poglavljima opisali smo osnovne principe funkcionisanja računarskog sistema na primjeru osnovnog računara. U ovom poglavlju daćemo opis konkretnog procesora - Intel 8086. Ovaj procesor je jedan od najvažnijih u istoriji računarstva jer je, praktično, rodonačelnik danas najpopularnije familije procesora - Intelove 80x86, kojoj pripadaju procesori 8086, 80286, 80386, 80486, Pentium, Pentium II, Pentium III, Pentium IV, ... Kažemo da ovi procesori čine familiju jer svaki naredni može da izvršava programe pisane za prethodni model (kažemo da su procesori *kompatibilni*). Zahtjev za kompatibilnošću unosi dodatnu složenost i probleme pri konstrukciji procesora, ali istovremeno omogućuje da korisnici zadrže svoje programe pri prelasku na novi računar.

U ovom poglavlju imamo sljedeći grubi plan rada:

– izložiti osnovnu arhitekturu i skup mašinskih naredbi procesora Intel 8086,

– izložiti kako radi program debug.exe, da bismo mogli da vježbamo programiranje za Intel 8086 i

– izložiti kroz jedan realni asemblerski paket kako funkcioniše i kako se programira računar baziran na procesoru Intel 8086. Uz to ćemo izložiti i glavna svojstva sistemskog softvera koji je po pravilu prisutan; DOS–ove odnosno Windows–ove rutine.

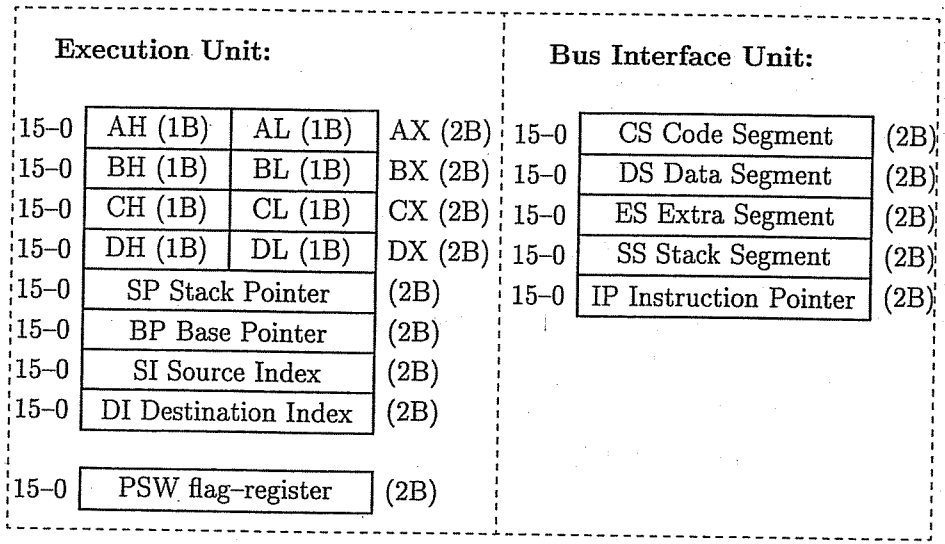
11.2 Osnovna arhitektura

Kao što se vidi na slici 8086 procesor ima dva glavna dijela:

- interfejs prema magistrali - ovaj dio dohvaća podatke i naredbe iz memorije;
- izvršna jedinica - dekodira i izvršava naredbe.

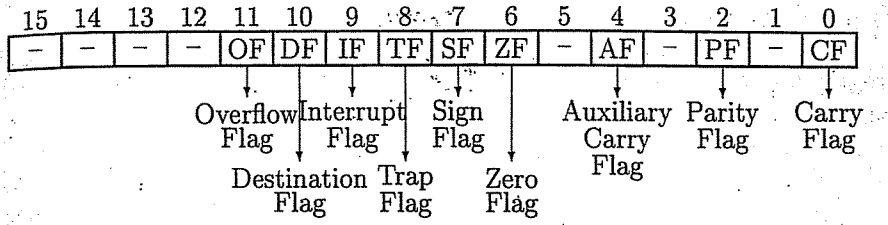
Ova dva dijela rade istovremeno, tj. ciklus izvršavanja poklapa se s ciklusom dohvata (paralelizam) što ubrzava rad mikroprocesora.

Na slici 11.1 prikazani su glavni registri procesora Intel 8086. Uobičajene su oznake AX = AH:AL i slično za BX, CX i DX. Registri opšte namjene su AX, AH, AL, BX, BH, BL, CX, CH, CL, DX, DH, DL, SP, BP, SI i DI (izdvojeni dio na slici).



Slika 11.1: Glavni registri procesora 8086

Na slici 11.2 prikazana je struktura važnog registra – registra flagova (Program Status Word).



Slika 11.2: Struktura registra flagova (Program Status Word)

11.3 Memorija

Magistala ima 20 bita, što znači da se može adresirati do $2^{20} = 1048576$ memorijskih lokacija, koje sadrže 8 bita (1 bajt), odakle slijedi da memorija ima kapacitet 1MB. Osnovne jedinice za rad s memorijom su bajt (8 bita) i riječ (16 bita).

Adresa svakog bajta u memoriji određena je sa 20 bita, ali kako su svi registri 16-bitni, cijela adresa ne može se smjestiti u jedan registar, pa se dijeli na dva dijela:

- adresa segmenta,
- adresa bajta u segmentu (tzv. offset ili pomjeraj).

Jedan segment ima 65536 (2^{16}) bajta. U heksadekadnoj notaciji adresa se zapisuje na sljedeći način:

$$SSSS : OOOO.$$

Na primjer, 0040:0002 je adresa trećeg bajta u segmentu 40h. Do 20–bitne adrese se dolazi tako što se adresa segmenta pomnoži sa 10h (dekadno 16) i sabere sa pomjerajem: 2A64:0042 => 2A64h * 10h + 42h = 2A640h+ 42h= 2A682h.

11.4 Registri

Procesor Intel 8086 posjeduje 14 registara. To su programski brojač, status registar, 4 registra opšte namjene, 2 pointera, 2 indeksna registra i 4 segmentna registra. Ovaj skup registara prisutan je i u svim narednim generacijama ove Intel-ove familije procesora.

11.4.1 Programski brojač

Programski brojač se kod ovog procesora zove *IP* (Instruction Pointer). On pokazuje na narednu instrukciju koju procesor treba da izvrši. Njegovu vrijednost uvećava sam procesor prilikom izvršenja instrukcije, osim u slučajevima skoka i poziva potprograma, kada mi zadajemo adresu na koju će se postaviti programski brojač. Osim toga, do promjena programskog brojača dolazi kada se desi prekid i kada procesor prelazi na izvršenje prekidne rutine.

11.4.2 Status registar

Status registar ili PSW (Program Status Word) je 16-bitni registar koji sadrži informacije o rezultatu posljednje izvršene instrukcije. Prazna mjesta označavaju rezervisane bite, čiji je sadržaj obično nula.

			O	D	I	T	S	Z		A		P		C

Bitovi PSW registra imaju sljedeće značenje:

O - Overflow. Ovaj bit označava da je došlo do prekoračenje opsega prilikom izvršenja aritmetičke operacije, koja se odnosi na rad sa označenim brojevima.

D - Direction. Ovaj bit se koristi u operacijama za rad sa stringovima i pokazuje smjer kretanja unutar stringa, 0 - ka višim, 1 - ka nižim memorijskim adresama.

I - Interrupt Enable. Ako je ovaj bit postavljen procesor će reagovati na spoljašnje prekide, a ako nije, reagovaće jedino na NMI (nemaskirajući prekid).

T - Trap. Ovaj bit se koristi za izvršavanje programa instrukciju po instrukciju. Ako je postavljen, poslije svake instrukcije će biti generisan prekid INT 3.

S - Sign. Ovaj bit sadrži znak rezultata prethodne instrukcije, zapravo kopiju najvišeg bita rezultata. Pretpostavka je da se radi o označenom broju.

Z - Zero. Ovaj bit nam govori da je rezultat posljednje operacije jednak nuli, odnosno, ako se radi o poređenju, da su operandi jednaki.

A - Auxiliary Carry. Ovaj bit predstavlja bit prenosa pri radu u BCD aritmetici.

P - Parity. Ovaj je bit pariteta, koji će biti postavljen u slučaju da je broj jedinica u rezultatu prethodne instrukcije paran.

C - Carry. Ovo je bit prenosa sa najvišeg mjesta poslije izvršavanja aritmetičkih operacija; On se koristi i kod operacija pomjeranja i rotiranja.

11.4.3 Registri opšte namjene

Registri opšte namjene mogu se generalno koristiti za smještanje podataka, ali svaki od njih ima i neke specifičnosti. Kao prvo, oni se mogu posmatrati ili kao 16-bitni ili kao par 8-bitnih registara. Na primjer, 16-bitni registar AX može se posmatrati kao par 8-bitnih registara AH i AL. H i L dolazi od high i low - viši i niži biti.

15 8	7 0	
AH	AL	AX
BH	BL	BX
CH	CL	CX
DH	DL	DX

AX i njegov niži dio **AL** često se nazivaju i akumulatorom. Akumulator predstavlja podrazumijevani argument nekih instrukcija i zato te instrukcije zauzimaju manji prostor u memoriji u odnosu na one koje rade sa drugim registrima.

BX (Base Register) se koristi kao bazni registar prilikom pristupa memoriji, naravno kada se radi o nekoj od baznih metoda adresiranja (vidi niže).

CX (Counter Register) se koristi kao brojač. Ovaj registar služi za smještanje trenutne vrijednosti brojača. Implicitno se umanjuje prilikom izvršavanja instrukcija za rad sa stringovima i petljama. Njegov niži dio **CL** koristi se u istrukcijama za rotiranje i pomjeranje - u njega smješta operand koji označava broj ponavljanja date operacije nad argumentom.

Registar **DX** implicitno se koristi u aritmetičkim operacijama množenja i dijeljenja, kao i za smještanje adrese porta kod ulazno-izlaznih instrukcija.

11.4.4 Pointerski registri

Pointerski registri služe za pristup podacima koji se nalaze u stek segmentu, a dozvoljena je i njihova upotreba u aritmetičko-logičkim operacijama.

SP (Stack Pointer) je pokazivač na vrh steka, i svaka instrukcija koja radi sa stekom ga modifikuje. Prilikom formiranja apsolutne adrese uz njega se podrazumijeva segmentni registar **SS** (Stack Segment). Treba biti vrlo pažljiv pri direktnim izmjenama ovog registra, naročito ako nismo sigurni šta se trenutno nalazi na steku.

BP (Base Pointer) se obično koristi za pristup podacima koji se nalaze na steku i to bez izmjene steka. Korišćenjem ovog registra možemo podatke prenijete potprogramu koristiti u proizvoljnom redosljedu, nezavisnom od redosljeda njihovog postavljanja na stek. Na steku se obično nalaze i lokalne promjenljive, koje pripadaju potprogramu i njima se pristupa na isti način.

11.4.5 Indeksni registri

Indeksni registri koriste se implicitno prilikom rada sa stringovima. Oni se mogu koristiti i kao registri opšte namjene u aritmetičkim i logičkim operacijama.

SI (Source Index) predstavlja indeks unutar izvornog stringa. Njegova vrijednost se mijenja u zavisnosti od flega smjera (Direction).

DI (Destination Index) igra sličnu ulogu i predstavlja indeks unutar odredišnog stringa. Njegova vrijednost se takođe implicitno mijenja u zavisnosti od flega smjera kao i **SI**.

11.4.6 Segmentni registri

Segmentni registri omogućavaju segmentnu organizaciju memorije procesora. Svaki od ovih segmentnih registara definiše blok od 64 KB memorije. Procesor 8086 ima 4 segmentna registra.

CS (Code Segment) registar govori u kom bloku memorije se nalazi izvršni kod programa. U toku faze prihvatanja instrukcije pristupa se isključivo lokacijama koje se nalaze u ovom segmentu. Sadržaj CS registra se smješta na stek prilikom poziva potprograma samo u slučaju kada se potprogram nalazi u drugom segmentu (**far**).

DS (Data Segment) registar pokazuje na blok memorije namijenjen smještanju podataka. Sve instrukcije, osim onih za rad sa stekom, pristupaju ovom segmentu (izuzev ako se to drugačije ne navede).

SS (Stack Segment) registar definiše stek segment i samim tim se koristi implicitno u svim operacijama za rad sa stekom, odnosno kada se adresa formira sa registrima **SP** i **BP**.

ES (Extra Segment) je dodatni segment podataka, i implicitno se koristi za rad sa stringovima.

11.5 Metode adresiranja

Procesor 8086 poznaje 8 načina adresiranja.

11.5.1 Neposredno adresiranje

Ovo je najjednostavnija metoda adresiranja - radi se o neposrednoj dodjeli konstante. Kada se instrukcija, koja koristi ovo adresiranje, prevede u mašinski oblik, neposredno iza koda operacije nalaziće se zadata konstanta.

```
mov ah,34h    (ima smisao ah ← 34h)
mov ax,4563h
```

Za 16-bitne konstante, važno je naglasiti da se u memoriju prvo smješta njen niži, a zatim viši bajt.

11.5.2 Registarsko adresiranje

Ova metoda adresiranja svodi se na preuzimanje sadržaja registra i njegovo smještanje u odredište. Primjeri :

```
mov ax,bx
mov si,dx
mov al,dl
mov ds,ax
mov ax,cs
```

11.5.3 Registarsko indirektno adresiranje

Kod ovog adresiranja imamo indirekciju sadržaja registra. Dakle, sadržaj registra, navedenog između uglastih zagrada, tumači se kao adresa na kojoj se nalaze podaci. Primjeri:

```
mov al,[bx]
mov al,[bp]
mov al,[si]
mov al,[di]
```

Prilikom upotrebe registara BX, SI i DI za računanje efektivne adrese koristi segmentni registar DS, dok se za BP koristi SS. Eksplicitnim navođenjem segmentnih registara moguće je uticati na računanje efektivne vrijednosti, kao u sljedećim primjerima:

```
mov al,cs:[bx]
mov al,ds:[bp]
mov al,es:[si]
mov al,ss:[di]
```

11.5.4 Direktno adresiranje

U ovoj metodi adresiranja direktno se, u zagradama, naznačava adresa na kojoj se nalazi vrijednost. Adresa je uvijek 16-bitna vrijednost. Primjeri:

```
mov al,[0234h]    (ima smisao al ← c(ds : 234))
mov [0045h],ax
```

Treba reći da se kod ove metode adresiranja kao segmentni registar podrazumijeva DS, ali je to moguće predefinisati kao u narednim primjerima.

```
mov al,es:[0234h]
mov ax,cs:[3452h]
```

11.5.5 Indeksirano adresiranje

Ova metoda adresiranja povezuje registarsko indirektno i direktno adresiranje. Efektivna adresa formira se sabiranjem vrijednosti registra i navedene konstante. Kod ove metode adresiranja indeks je u registrima SI i DI, a konstanta je u principu početna adresa bloka kome se pristupa.

```
mov al,20h[si] ili mov al,[si+20h]
```



```
mov dl,78[di] ili mov dl,[di+78]
```

Prilikom računanja efektivne adrese koristi se segmentni registar DS, što se može izmijeniti ako se eksplicitno navedu segmentni registri kao u sljedećim primjerima.

```
mov bh,es:[si+340]
mov cl,ss:[di+120]
```

11.5.6 Bazno adresiranje

Razlika ovog adresiranja i indeksiranog adresiranja je u tome da bazni registar predstavlja početak bloka, a konstanta, koja se uz njega navodi, relativan položaj elementa kome pristupamo. Kao bazni registri se mogu koristiti BX i BP.

```
mov al,20h[bx] ili mov al,[bx+20h]
mov dl,120[bp] ili mov dl,[bp+120]
```

Kada je bazni registar BX, za računanje efektivne adrese koristi se segmentni registar DS, dok se kod baznog registar BP koristi segmentni registar SS. Ovo se može izmijeniti ako se segmentni registri navedu eksplicitno :

```
mov bh,es:[bx+340]
mov cl,cs:[bp+120]
```

11.5.7 Bazno indeksirano adresiranje

Ovo je podvarijanta indeksiranog adresiranja, s tom razlikom da više ne postoji konstanta koja određuje početak strukture kojoj se pristupa, već tu ulogu preuzima bazni registar. Kao i kod prethodnih metoda, važi ista napomena vezana za segmentne registre.

```
mov al,ds:[bp][si] ili mov al,ds:[bp+si]
mov dl,es:[bx][di] ili mov dl,es:[bx+di]
```

11.5.8 Bazno indeksirano relativno adresiranje

Ovo je najkompleksnija metoda adresiranja koja kombinuje prethodne dvije vrste adresiranja, bazno i bazno indeksirano. U računanju efektivne adrese

pored jednog baznog i jednog indeksnog registra učestvuje i konstanta pomjeraja. Takođe važi napomena vezana za segmentne registre. Primjeri za ovu metodu slični su prethodnim.

```
mov al,ds:25h[bp][si] ili
mov al,ds:[bp+si+25h]
mov dl,es:120[bx][di] ili
mov dl,es:[bx+di+120]
```

11.6 Instrukcije

Sada ćemo se upoznati sa instrukcijama procesora Intel 8086. Instrukcije ćemo podijeliti u više logičkih grupa, po njihovoj funkciji.

11.6.1 Instrukcije za prenos podataka

Instrukcije za prenos podataka možemo podijeliti u četiri grupe.

Instrukcije prenosa opšte namjene

U ovoj grupi nalazi se instrukcija **MOV**. Ona posjeduje dva operanda, od kojih je prvi odredište, a drugi izvor podatka koji treba prenijeti. Moguće je vršiti prenos između registara, memorijskih lokacija, a i dodijeliti konstantu. Preciznije, moguće su varijante

```
MOV reg, {reg|mem|immed}
MOV mem, {reg|immed}
MOV {reg16|mem16}, {CS|DS|ES|SS}
MOV {DS|ES|SS}, {reg16|mem16}
```

gdje koristimo sljedeće oznake: reg je procesorov registar opšte namjene (8 bita ili 16 bita); mem je sadržaj memorijske lokacije; immed – immediate tj. neposredna vrijednost; reg16 je procesorov registar opšte namjene čija je veličina 16 bita, a mem16 je sadržaj memorijske lokacije čija je veličina 16 bita. Naznačeni segmenti označavaju da memorijske lokacije mogu biti samo u tim segmentima - ne može se upisivati u Code Segment.

Instrukcija **LEA** predstavlja specifičnu instrukciju prenosa. Ona služi za učitavanje efektivne adrese drugog operanda (Load Effective Address).

Prvi operand ove instrukcije mora biti neki od 16-bitnih registara izuzev segmentnih.

Učitavanje pointera se vrši uz pomoć instrukcija **LDS** i **LES**. Drugi operand predstavlja adresu na kojoj se nalazi 32-bitni pointer (adresa). Prvi operand predstavlja registar u koji će biti smješten ofset na koji pokazuje pointer, dok se segment smješta u jedan od segmentnih registara (**DS**, kod instrukcije **LDS**, odnosno u **ES**, kod **LES**). Dakle, iz memorije se donose 4 bajta. Dva niža bajta upisuju se u naznačeni registar, a dva viša bajta u registar DS (odnosno ES).

Instrukcije **LAHF** i **SAHF** obavljaju razmjenu sadržaja sa status registrom. Ove instrukcije nemaju operanada, podrazumijeva se da je **AH** prvi, a nižih 8-bita status registra - drugi operand. Instrukcija **LAHF** smješta sadržaj nižih osam bita status registra u **AH**, dok instrukcija **SAHF** ima suprotno dejstvo.

Instrukcija **XCHG** služi za zamjenu sadržaja dva operanda. Jedan od operanada može biti i memorijska lokacija, dok drugi obavezno mora biti registar. Ovom instrukcijom izbjegava se korišćenje treće, pomoćne, promjenljive prilikom izmjene sadržaja.

Navedimo par primjera (brojne vrijednosti date su u heksadekadnom sistemu).

Ako je stanje memorije:

FFFF:000F		ECA8:000A	ECA8:0009		0000:0000
	...	51	60	...	

a u procesoru:

AX | 3 | 4 |, BX | 9A | BC |

tada

mov ax, bx	ima smisao ax ← 9abc
mov ah, bh	ima smisao ah ← 9a
mov ax, [eca8:0009]	ima smisao ax ← 5160
mov [eca8:0009], ah	ima smisao c(eca8:0009) ← 3 (jedan bajt)

Takođe (uzima se iz memorije iz tekućeg segmenta podataka):

lds si, [5f]	ima smisao si ← c(5f) & ds ← c(61)
les si, [5f]	ima smisao si ← c(5f) & es ← c(61)

Instrukcije za rukovanje stekom

Stek je jedna od procesorski podržanih struktura podataka. Kao što znamo, radi se o LIFO (Last In First Out) organizaciji, što znači da prvo čitamo posljednji unijeti podatak. Instrukcije ove grupe, kao implicitni operand, imaju memorijsku adresu na koju pokazuje registarski par **SS:SP**, što predstavlja *vrh steka*. Stek se puni prema dolje, odnosno prema nižim adresama.

Instrukcija stavljanja podatka na stek je **PUSH**. Njen jedini operand određuje sadržaj koji treba staviti na stek. On može biti 16-bitni registar, uključujući i segmentne registre (osim **CS**), ili sadržaj neke memorijske lokacije.

Za skidanje sa steka koristi se instrukcija **POP**. Njen operand govori gdje će podatak biti smješten. To može biti 16-bitni registar ili memorijska lokacija.

Postoje i instrukcije **PUSHF** i **POPF**, za koje se podrazumijeva da je njihov operand status registar.

Stek se implicitno koristi i u nekim drugim instrukcijama, što ćemo vidjeti niže.

Instrukcije konverzije

Instrukcije ove grupe nemaju operande, a implicitno mijenjaju neke od registara.

Konverzija broja, odnosno prenošenje znaka, iz bajta (8 bita) u riječ (16 bita) se obavlja instrukcijom **CBW** (Convert byte to word). Instrukcija radi sa akumulatorom i podrazumijeva da se podatak nalazi u **AL**. Rezultat je kompletan registar **AX**.

Slično, imamo instrukciju **CWD**, koja služi za konverziju iz riječi u duplu riječ, pri čemu se podrazumijeva da je ulaz registar **AX**, a izlaz registarski par **DX:AX**.

Konverzija, uz upotrebu posebne tabele konverzije, može se izvršiti pomoću instrukcije **XLAT**. Tabela konverzije je tabela uzastopnih vrijednosti, čiji indeksi odgovaraju vrijednostima koje se konvertuju u vrijednosti navedene u tabeli. Ova instrukcija ima smisao AL $\leftarrow$ [BX + AL]. Podrazumijeva se da se operand konverzije nalazi u registru **AL**, dok **BX** sadrži adresu početka tabele konverzije. Instrukcija smješta u registar **AL** vrijednost pročitanu na lokaciji dobijenoj sabiranjem vrijednosti registara **BX** i **AL**.

Instrukcije za rukovanje periferijom

Pristup periferijama se vrši isključivo preko instrukcija iz ove grupe, osim ako nemamo računarski sistem sa memorijski preslikanim ulazom-izlazom, u kom slučaju se mogu koristiti i ostale instrukcije prenosa podataka.

Periferijama odgovaraju posebne memorijske lokacije - *portovi*. Za čitanje vrijednosti na nekom portu koristimo instrukciju **IN**. Vrijednost se smješta isključivo u akumulator, što je i prvi operand. On govori koja je širina podatka koji se prenosi, **AL** 8-bitna, **AX** 16-bitna. Drugi operand može biti 8-bitna konstanta, koja predstavlja adresu porta, ili registar **DX**, koji sadrži 16-bitnu adresu porta kojeg čitamo. Vidimo da se može adresirati 256 ili 65536 ulazno–izlaznih uređaja.

Slična je i instrukcija **OUT**, koja služi za slanje podatka na odgovarajući port. Operandi su isti, s tom razlikom da su im zamijenjena mjesta, tj. prvo je adresa porta, a zatim podatak.

11.6.2 Aritmetičke instrukcije

Aritmetičko-logička jedinica 8086 procesora je 16-bitna. Ona ima mogućnost rada u BCD aritmetici. BCD brojevi se predstavljaju u formi u kojoj svaka četiri bita predstavljaju binarni kod jedne decimalne cifre. Postoje tzv. otpakovani i spakovani format. Otpakovani format ima za pretpostavku da se u jednom 8-bitnom registru nalazi samo jedna cifra, što je uvedeno radi lakše implementacije aritmetičkih operacija, dok spakovani format podrazumijeva da se u 8-bitnom registru nalaze dvije cifre, što je opet zgodno za čuvanje takvih brojeva u memoriji, jer su oni i onako predugački. Nedostatak korišćenja brojeva u spakovanoj formi je u tome što se nad njima ne mogu direktno vršiti operacije množenja i dijeljenja.

Instrukcije za sabiranje

Instrukcije za sabiranje imaju dva operanda, koji predstavljaju sabirke, a rezultat sabiranja se smješta u prvi operand. Na isti način se vrši sabiranje i sa označenim i sa neoznačenim brojevima, tj. ne postoje posebne instrukcije.

Prva od instrukcija sabiranja je **ADD**, koja sabira dva operanda, a rezultat smješta u prvi. Operandi naravno moraju biti iste dužine, a dozvoljene su kombinacije registara, memorijskih lokacija. Drugi operand može biti i konstanta.

Instrukcijom **ADC** vršimo sabiranje sa prenosom. Ona je slična instrukciji **ADD**, s tim da se na zbir dodaje i vrijednost bita prenosa iz status registra.

Prilikom pisanja programa u asembleru često se javlja potreba za uvećavanjem vrijednosti nekog registra za jedan, što je najčešći slučaj u petljama. Za to služi instrukcija **INC**. Ona ima jedan operand koji može biti neki od 8-bitnih ili 16-bitnih registara (izuzimajući segmentne registre) ili memorijska lokacija.

Sabiranje brojeva u BCD otpakovanom formatu se takođe vrši sa instrukcijom **ADD**, ali je nakon njene upotrebe potrebno pozvati instrukciju **AAA** koja je zadužena da izvrši popravku rezultata i njegovo pretvaranje u BCD oblik. Ograničenje je u tome što se sabiranje mora vršiti kroz registar **AX**, jer je on podrazumijevani operand ove instrukcije.

Sabiranje se može vršiti na identičan način i u spakovanoj formi, s tom razlikom da se nakon sabiranja mora pozvati instrukcija **DAA**, koja ima dejstvo kao i prethodna instrukcija, ali se radi o spakovanoj formi.

Primjeri (prvi navedeni primjer ima generalni smisao):

add dest, src	ima smisao dest ← dest + src
add ax, bx	ima smisao ax ← ax +bx
add ax, 356a	ima smisao ax ← ax + $(356a)_{16}$
add ax, [bx]	ima smisao ax ← ax + c(bx)
add ax, [356a]	ima smisao ax ← ax + c(356a)
add al, [356a]	ima smisao al ← al + c(356a)
add [356a], al	ima smisao c(356a) ← c(356a) + al

Instrukcije za oduzimanje

Postoji više instrukcija za oduzimanje. Instrukcija **SUB** vrši oduzimanje drugog operanda od prvog, a rezultat smješta u prvi operand. Operandi moraju biti istih dužina da bi se dobio ispravan rezultat, bilo označen bilo neoznačen.

Instrukciju **SBB** vrši oduzimanje sa pozajmicom, koja se nalazi u bitu prenosa status registra.

Umanjivanje operanda za jediničnu vrijednost je takođe često potrebna funkcija, pa zato imamo instrukciju **DEC**, koja vrši umanjivanje vrijednosti jedinog svog operanda, koji može registar ili memorijska lokacija.

Oduzimanje sa BCD brojevima se vrši slično kao i sabiranje. Kada se koristi BCD aritmetika u otpakovanoj formi, nakon oduzimanja koristimo instrukciju **AAS**, da bismo ponovo vratili rezultat u odgovarajuću formu, a **DAS** kada radimo u spakovanoj formi. Slično kao kod sabiranja, i ovdje važi ograničenje da se sabiranje mora vršiti kroz akumulator, jer je on imlicitni operand prethodne dvije instrukcije.

Instrukcija za poređenje

Poređenje se zasniva na oduzimanju, s tom razlikom da se razlika zanemaruje, a postavljaju se samo bitovi status registra. Instrukcija **CMP** omogućuje poređenje. Ona ima dva operanda koje poredimo. Nakon ove instrukcije uvijek se izvršava neka od instrukcija uslovnog skoka.

Instrukcija za komplement

Nalaženje suprotne vrijednosti omogućuje instrukcija **NEG**. Ona posjeduje jedan operand koji predstavlja vrijednost čiju suprotnu (negativnu vrijednost) treba naći.

Instrukcije za množenje

Za množenje neoznačenih brojeva koristimo instrukciju **MUL**. Ona podrazumijeva da se u akumulatoru nalazi jedan operand i zahteva da se eksplicitno navede samo drugi operand. Ako je operand dužine bajta, tada se množi sa **AL**, a rezultat smješta u **AX** registar. Ako je operand riječ, tada se množi sa **AX**, a rezultat se smješta u duplu riječ **DX:AX**.

Slično ovome, za množenje označenih brojeva koristimo instrukciju **IMUL**, koja radi na sličan način kao i prethodna.

Kada se radi sa BCD brojevima, moramo koristiti raspakovani format da bismo bili u mogućnosti da radimo i operaciju množenja. Nakon izvršenog

množenja potrebno je popraviti rezultat instrukcijom **AAM**, koja nema operanada, a podrazumijevani je akumulator, što se poklapa sa izlaznim registrom instrukcije množenja.

Instrukcije za dijeljenje

Za dijeljenje neoznačenih brojeva koristimo instrukciju **DIV**. Ona podrazumijeva da se u akumulatoru nalazi dijeljenik, i zahteva da se kao eksplicitni operand navede samo djelilac. Ako je operand dužine bajta, tada se vrijednost registra **AX** dijeli sa operandom, rezultat smješta u **AL**, a ostatak dijeljenja u **AH**. Kada je operand dužine riječi, tada se **DX:AX** dijeli operandom, rezultat smješta u **AX**, a ostatak u **DX**.

Slično, postoji instrukcija za dijeljenje označenih brojeva, **IDIV**.

Dijeljenje BCD brojeva u otpakovanom formatu je takođe moguće, ali je nakon njega potrebno instrukcijom **AAD** prilagoditi rezultat dijeljenja. Implicitni operand ove instrukcije je akumulator, u kome se, ujedno, nalazi rezultat dijeljenja.

11.6.3 Logičke instrukcije i instrukcije pomjeranja i rotiranja

Logičke instrukcije

Instrukcijom **AND** izvršavamo logičku operaciju I nad operandima, a rezultat operacije se smješta u prvi od njih. Kao što znamo, rezultat ima jedinice na onim binarnim pozicijama na kojima ih istovremeno imaju i oba operanda, dok se na ostalim pozicijama nalaze nule. Dozvoljene kombinacije operanada su iste kao i kod aritmetičkih instrukcija.

Logičko ILI se postiže instrukcijom **OR**, uz ista ograničenja.

Operacija ekskluzivno ILI obavlja se instrukcijom **XOR**. Način upotrebe je identičan kao kod prethodnih instrukcija.

Logičko NE, odnosno negaciju, obavlja instrukcija **NOT** i, za razliku od prethodnih instrukcija, ima jedan operand, koji ujedno služi i kao izlaz operacije.

Instrukcija za rukovanje bitima

Često imamo potrebu da provjerimo vrijednost nekog bita, a da tom prilikom ne izmijenimo sadržaj registra ili memorijske lokacije u kojoj se on nalazi. Taj problem se rješava instrukcijom **TEST**, koja vrši operaciju logičko I između dva operanda i samo postavlja bitove status registra. Poslije ove instrukcije obično slijede instrukcije uslovnog skoka.

Instrukcije za pomjeranje

Postoje dvije vrste pomjeranja - logičko i aritmetičko. Razlika je u tome što instrukcije aritmetičkog pomjeranja čuvaju znak operanda, dok kod logičkog pomjeranja to nije slučaj.

Instrukcije logičkog i aritmetičkog pomjeranja ulijevo su identične. Njihovi nazivi su **SHL** i **SAL**, respektivno. Ove instrukcije kao prvi operand sadrže vrijednost koju treba pomjeriti, a drugi operand je broj 1, ili registar **CL**, koji sadrži podatak o zahtijevanom broju pomjeranja sadržaja prvog operanda. Prilikom pomjeranja ubacuje se nula na najnižu poziciju, a bit sa najviše pozicije odlazi u bit prenosa status registra.

Kod pomjeranja udesno razlikuju se operacije logičkog i aritmetičkog pomjeranja. Za operaciju logičkog pomjeranja udesno instrukcija je **SHR**. Prilikom pomjeranja bit sa najniže pozicije odlazi u bit prenosa status registra, dok na najvišu poziciju dolazi nula.

Instrukcija **SAR**, koja omogućuje aritmetičko pomjeranje udesno razlikuje se od instrukcije **SHR** u tome što se na najvišoj poziciji zadržava prethodna vrijednost, odnosno znak operanda.

Instrukcije za rotiranje

Rotacija (kružno pomjeranje) ulijevo se obavlja instrukcijom **RCL**, uz preuzimanje zatečenog bita prenosa, i instrukcijom **ROL**, bez preuzimanja zatečenog bita prenosa. Kod obije instrukcije se vrijednost sa najviše pozicije smješta u bit prenosa, s tom razlikom da se kod **RCL**, u najniži bit upisuje zatečena vrijednost iz bita prenosa, a kod **ROL** vrijednost sa najviše pozicije.

Rotacije udesno je slična. Instrukcija **RCR** omogućuje rotiranje udesno uz preuzimanje zatečenog bita prenosa, a **ROR** bez preuzimanja zatečenog bita prenosa. Prilikom rotacije udesno, bit sa najniže pozicije odlazi u bit

prenosa status registra, ali kod **RCR** instrukcije na najvišu poziciju dolazi zatečena vrijednost bita prenosa, dok kod **ROR** instrukcije na to mesto dolazi vrijednost sa najniže pozicije.

11.6.4 Upravljačke instrukcije

Upravljačke instrukcije omogućuju bezuslovnu i uslovnu izmjenu redosljeda izvršavanja instrukcija, obrazovanje programskih petlji i pozive potprograma.

Instrukcije bezuslovnog skoka

Osnovna instrukcija skoka je instrukcija bezuslovnog skoka **JMP**. Ona ima jedan operand koji predstavlja adresu na koju će se izvršiti skok. Razlikujemo tri različita oblika instrukcije bezuslovnog skoka, kratki skok (**short**) koji obuhvata raspon od -128 do 127 bajta, skok u blizini (**near**) koji se odnosi na lokacije iz istog segmenta, i daleki skok (**far**) koji dozvoljava skok van tekućeg segmenta. Za kratki skok raspon se računa u odnosu na instrukciju skoka. Prilikom prevođenja asembler sam otkriva da li je skok unutar istog segmenta ili između dva segmenta. Kada je riječ o kratkom skoku, to treba eksplicitno navesti stavljanjem ključne riječi **short** ispred imena labele.

Primjeri:

jmp 4	ima smisao cs $\leftarrow$ cs & ip $\leftarrow$ 4
jmp 5:6	ima smisao cs $\leftarrow$ 5 & ip $\leftarrow$ 6

Instrukcije uslovnog skoka

Instrukcije uslovnog skoka koriste se nakon instrukcija koje utiču na sadržaj status registra. Obično se radi o aritmetičkim instrukcijama, najčešće poređenju ili nekoj iz grupe logičkih instrukcija. Njihov jedini operand omogućuje kratki skok u rasponu -128 do 127 bajta.

Instrukcija **JE** omogućuje utvrđivanje važenja relacije jednako. Ona važi za poređenje i označenih i neoznačenih brojeva. Slično, instrukcija **JNE** omogućuje utvrđivanje relacije različito.

Instrukcija **JA**, odnosno **JNBE**, se odnosi na neoznačene brojeve. Ona omogućuje utvrđivanje relacije veći, odnosno nije manji ili jednak.

Instrukcija **JBE**, odnosno **JNA**, se odnosi na neoznačene brojeve. Ona omogućuje utvrđivanje važenja relacije manje ili jednako, odnosno nije veće.

Za utvrđivanje važenja relacije veće ili jednako, odnosno nije manje, koristimo instrukciju **JAE**, odnosno **JNB**. I ova instrukcija važi za neoznačene brojeve.

Za utvrđivanje važenja relacije manje, odnosno nije veće ili jednako služi instrukcija **JB**, odnosno instrukcija **JNAE**. I ova instrukcija važi za neoznačene brojeve.

Za utvrđivanje važenja relacije veće, odnosno nije manje i jednako, za označene brojeve služi instrukcija **JG**, odnosno **JNLE**.

Za utvrđivanje važenja relacije manje ili jednako, odnosno nije veće, za označene brojeve služe instrukcije **JLE**, odnosno **JNG**.

Za utvrđivanje važenja relacije veće ili jednako, služi instrukcija **JGE**, a relacije nije manje instrukcija **JNL**. Obije se odnose na označene brojeve.

Za utvrđivanje važenja relacije manje za označene brojeve služi instrukcija **JL**, odnosno **JNGE**.

Pošto se brojačka konstanta obično smješta u registar **CX**, procesor podržava i instrukcija skoka ako je registar **CX** jednak nuli. To je instrukcija **JCXZ**.

Preostale instrukcije uslovnog skoka omogućuju skok ako je odgovarajući bit status registra jednak nuli odnosno jedinici.

Instrukcija **JZ** omogućuje skok ako je bit **Z** postavljen, a instrukcija **JNZ** omogućuje skok ako ovaj bit nije postavljen. Slično, instrukcija **JC** omogućuje skok kada je bit **C** postavljen, a instrukcija **JNC** omogućuje skok ako ovaj bit nije postavljen.

Što se tiče bita pariteta, instrukcija **JP**, odnosno **JPE** omogućuje skok kada je bit postavljen, a instrukcija **JPO**, odnosno **JNP**, kada on nije postavljen.

Slično, za bit prekoračenja instrukcija **JO** omogućuje skok kada je bit **O** postavljen, a instrukcija **JNO** kada on nije postavljen.

Za bit predznaka instrukcija **JS** omogućuje skok kada je bit **S** postavljen, a instrukcija **JNS** kada on nije postavljen.

Navedimo još jednom ove instrukcije skoka.

Uslovni skokovi sa označenim brojevima.

JL/JNGE	Skok ako je manji. Ako je SF $\neq$ OF.
JLE/JNG	Skok ako je manji ili jednak. Ako je ZF = 1 ili SF $\neq$ OF.
JG/JNLE	Skok ako je veći. Ako je ZF = 0 i SF = OF.
JGE/JNL	Skok ako je veći ili jednak. Ako je SF = OF.

Uslovni skokovi sa neoznačenim brojevima.

JA/JNBE	Skok ako je veći. Ako je CF = 0 i ZF = 0.
JBE/JNA	Skok ako je manji ili jednak. Ako je CF = 1 ili ZF = 1.
JCXZ	Skok ako CX sadrži nulu. Skok ako je CX = 0.

Uslovni skokovi koji zavise od vrijednosti flagova.

JAE/JNB/JNC	Skok ako je veći ili jednak. Ako je CF = 0.
JB/JNAE/JC	Skok ako je manji. Ako je CF = 1.
JE/JZ	Skok ako je jednak. Ako je ZF = 1.
JNE/JNZ	Skok ako je različit. Ako je ZF = 0.
JNO	Skok ako nije prekoračenje. Ako je OF = 0.
JO	Skok ako je prekoračenje. Ako je OF = 1.
JNS	Skok ako nije predznak. Ako je SF = 0.
JS	Skok ako jeste predznak. Ako je SF = 1.
JP/JPE	Skok ako je parnost parna. Ako je PF = 1.
JNP/JPO	Skok ako je parnost neparna. Ako je PF = 0.

Instrukcije za rukovanje potprogramima

Pozivanje potprograma omogućuje instrukcija **CALL**, čiji operand sadrži adresu potprograma. Ova instrukcija implicitno koristi stek za smještanje povratne adrese.

Na primjer, **CALL n** ima smisao 'idi na potprogram koji počinje na mjestu n'. U suštini, ova naredba upisuje odgovarajuću povratnu adresu u stek i prenosi upravljanje na prvu naredbu potprograma. Potprogram n može da bude oblika NEAR ili oblika FAR. Ako je u pitanju potprogram oblika NEAR onda se u stek upisuje samo IP, a ako je oblika FAR onda se u stek upisuju CS i IP. Na primjer, **CALL 55** znači da staro IP ide u stek (tako da SP $\leftarrow$ SP $-$ 2) i IP $\leftarrow$ 55; CS ostaje po starom. A **CALL 77:99** znači da stari CS i IP idu u stek (tako da SP $\leftarrow$ SP $-$ 4), CS $\leftarrow$ 77 i IP $\leftarrow$ 99.

Povratak iz potprograma omogućuje instrukcija **RET** (Return, povratak). Ova instrukcija uzima sadržaj iz steka i prenosi upravljanje na

povratnu adresu. Drugim riječima, IP dobija vrijednost iz steka (tako da SP $\leftarrow$ SP + 2); CS ostaje po starom.

Ako ima direktni operand **RET immed**, tada se izvodi jedna radnja više u odnosu na RET. To je radnja SP $\leftarrow$ SP + immed, kojom se briše sa steka. Dakle, IP dobija vrijednost iz steka i SP $\leftarrow$ SP+2+immed; CS ostaje po starom.

Sintaksa **RETF** označava Return far. Sada i IP i CS dobijaju vrijednosti iz steka (tako da SP $\leftarrow$ SP + 4).

Instrukcije za obrazovanje programskih petlji

Instrukcije za obrazovanje programskih petlji podrazumijevaju da se brojačka konstanta nalazi u **CX** registru i da se njena vrijednost u svakoj iteraciji smanjuje za jedan. Izlazak iz petlje se dešava onda kada registar **CX** postane jednak nuli i on podrazumijeva obavljenje kratkog skoka.

Instrukcija **LOOP** je osnovna iz ove grupe, a tu su i **LOOPE** i **LOOPZ**, kao i **LOOPNE** i **LOOPNZ**, kod kojih dolazi do izlaska iz petlje i ako je postavljen bit **Z** status registra, odnosno ako on nije postavljen.

Preciznije, **LOOP operand** ima smisao CX $\leftarrow$ CX$-$1 i skoči na operand ako je CX $\neq$ 0. Dakle, sve dok je CX $\neq$ 0, CX se smanjuje za 1 i ponavljaju se naredbe između operanda i naredbe LOOP.

Sintaksa **LOOPE/LOOPZ operand** ima smisao CX $\leftarrow$ CX $-$ 1 i skoči na operand ako je ZF = 1 i CX $\neq$ 0. Sintaksa **LOOPNE/LOOPNZ operand** znači CX $\leftarrow$ CX $-$ 1 i skoči na etiketu ako je ZF = 0 i CX $\neq$ 0.

Već je rečeno da skok ne može biti dalji od -128 bajta ili $+127$ bajta od mjesta gdje se nalazi naredba prelaska u kojoj se operand pojavljuje.

Instrukcije za rukovanje prekidima

Prekidi se mogu softverski izazivati pomoću instrukcije **INT**, čiji je jedini operand redni broj interapta, tj. indeks u vektoru prekida. Prilikom skoka na prekidnu rutinu CS, IP i PSW se smještaju na stek. CS i IP dobijaju odgovarajuće vrijednosti: CS $\leftarrow$ [0 : 4 $\cdot$ operand + 2], IP $\leftarrow$ [0 : 4 $\cdot$ operand].

Postoji i instrukcija izazivanja prekida prekoračenja **INTO**, koja nema operanda, i izaziva prekid 4 ako je bit **O** status registra postavljen. Prilikom

softverskog izazivanja prekida na steku se sačuvaju povratna adresa kao i zatečeni sadržaj status registra.

Povratak iz obrade prekida omogućuje instrukcija **IRET** koja skida sa steka povratnu adresu i sadržaj status registra, i podešava IP, CS i PSW, pa se tako procesor vraća na prekinuti program.

11.6.5 Instrukcije za rukovanje nizovima

Instrukcije za rukovanje nizovima, kao implicitni operand koriste indeksne registre **SI** i **DI**, čija vrijednost se nakon izvršenja instrukcije mijenja zavisno od sadržaja bita **D** status registra. Pritom je **SI** registar vezan za **DS** segmentni registar, a **DI** registar je vezan za **ES** segmentni registar. Sve instrukcije za rukovanje nizovima imaju dva oblika: jedan za rad sa nizovima bajta, sa postfiksom **B**, i drugi za rad sa nizovima riječi, sa postfiksom **W**.

Instrukcija **LODS** (**LODSB** za bajte i **LODSW** za riječi) služi za učitavanja elementa niza u akumulator. Ovdje se implicitno mijenja samo **SI** registar, koji pokazuje na elemente niza iz koga se čitaju podaci.

Smještanje u niz se vrši instrukcijom **STOS** (**STOSB** za bajte i **STOSW** za riječi). Vrijednost se uzima iz akumulatora i smješta u element na koji pokazuje **DI** indeksni registar.

Za kopiranje elemenata nizova koristi se instrukcija **MOVS** (**MOVSB** za bajte, odnosno **MOVSW** za riječi). Kopiranje se vrši iz elementa na koji pokazuje **SI**, u element na koji pokazuje **DI**.

Instrukcija **SCAS** (**SCASB** za bajte i **SCASW** za riječi) omogućuje poređenje akumulatora sa elementima niza. Na elemente niza pokazuje **DI** registar.

Međusobno poređenje elemenata nizova omogućuje instrukcija **CMPS** (**CMPSB** za bajte, i **CMPSW** za riječi). Na prvi operand pokazuje **SI** registar, a na drugi **DI** registar.

Dodavanje prefiksa **REP** ispred imena instrukcije za rukovanje nizovima označava ponavljanje odgovarajuće instrukcije. Pri tome broj ponavljanja smještamo prethodno u **CX** registar. Nakon svakog ponavljanja vrijednost **CX** registra se umanjuje, a ponavljanje završi kada vrijednost ovog registra padne na nulu. Kada prefiksi **REP**, **REPE** ili **REPZ** stoje ispred instrukcija **SCAS** ili **CMPS**, ponavljanje se završava i kada bit **Z** status registra nije

postavljen. U slučaju prefiksa **REPNE** i **REPNZ** ponavljanje se završava i kada je bit **Z** status registra postavljen.

11.6.6 Instrukcije za rukovanje status registrom

Postavljenje bita **C** status registra omogućuje instrukcija **STC**, njegovo poništenje instrukcija **CLC**. Izmjenu vrijednosti ovog bita, odnosno njegovo komplementiranje, omogućuje instrukcija **CMC**.

Postavljanje bita **D** status registra omogućuje instrukcija **STD**, a njegovo poništavanje instrukcija **CLD**.

Instrukcija **STI** postavlja bit **I** status registra, a instrukcija **CLI** poništava ovaj bit.

11.6.7 Razne instrukcije

Instrukcija **NOP** ne radi ništa osim što troši procesorsko vreme, pa omogućuje generisanje kratkih pauza, reda desetina taktova procesora.

Instrukcija **WAIT** omogućuje usklađivanje rada procesora i numeričkog koprocesora. Naime, ovom naredbom procesor se dovodi u stanje mirovanja. Svakih 5 taktova on gleda da li je aktivna njegova ulazna linija (njegov pin) TEST. Kada ta linija postane aktivna, procesor prelazi na sljedeću naredbu iza naredbe WAIT.

Instrukcija **HLT** omogućuje zaustavljenje rada procesora.

Ako ispred neke naredbe stoji naredba **LOCK** (stoji prefiks LOCK) onda to znači da je onemogućeno da drugi procesori koriste magistrale, sve dok se ne završi izvršavanje te naredbe. Drugi procesori su koprocesor, DMA kontroler ili slično. Naredba LOCK može da prethodi bilo kojoj naredbi i njome se aktivira pin LOCK.

Na kraju ovog pregleda instrukcija, daćemo još nekoliko napomena. Naime, u našem izlaganju, nastojali smo da izbjegnemo zamorne tehničke detalje o procesoru Intel 8086. Ipak, potrebno je kazati bar nekoliko rečenica.

Procesor ima 40 pinova (nožica, spoljašnjih izvoda) koji služe za njegovu komunikaciju sa ostalim djelovima računarskog sistema.

Kod ovog procesora postoji devet vrsta ciklusa: FETCH donošenje naredbe, MEMORY READ čitanje iz memorije, MEMORY WRITE upisivanje u memoriju, STACK READ čitanje iz steka, STACK WRITE zapis u stek, INPUT ulaz iz perifernog uređaja, OUTPUT izlaz na periferni uređaj, INTERRUPT prekidni ciklus i HALT zaustavljanje.

Koliko taktova traje izvršavanje pojedine naredbe zavisi od vrste naredbe, a može da zavisi (za jednu te istu vrstu naredbe) i od okolnosti– metode adresiranja ili drugog. U prosjeku, može se reći da se često potroši 10 taktova približno. Mada su odstupanja velika– neke naredbe se izvrše za dva–tri takta, a neke zahtijevaju i puno više od 10 taktova.

Veličina naredbe je u granicama od jednog do pet bajta. Na primjer, uzmimo naredbu koja zauzima tri bajta u memoriji i to (recimo) bajtove čiji su redni brojevi 108, 109 i 10A. U programu se prosto kaže da se ta naredba nalazi na adresi 108.

Prvi Intelov mikroprocesor bio je Intel 4004; četvoro–bitni; 1971. godina. Osmo–bitni mikroprocesor Intel 8080 pojavio se na tržištu 1974. godine. Intel 8086 je prvi 16–bitni mikroprocesor proizveden od strane firme Intel; 1978. godina.

11.7 Upotreba programa debug.exe

Možemo koristiti program debug.exe za propuštanje programa na jeziku asemblera. On se isporučuje uz operativni sistem Dos/Windows.

Nabrojmo neke od mogućnosti koje program pruža. Pomoću naredbe **A** (assemble) u memoriju mogu direktno da se upisuju naredbe ili podaci. Naredba **D** (dump) daje na ekranu sadržaj jedne oblasti memorije. Naredba **R** prikazuje sadržaj registara i daje mogućnost postavljanja svakog od njih. Na primjer, u programski brojač CS:IP može da se upiše određena vrijednost. Računar izvršava program počev od trenutne vrijednosti programski brojača pomoću naredbe **G** (go). Program može da se izvršava korak po korak, s tim da se poslije svakog izvršenog koraka, tj. naredbe, na ekranu prikaže tekući sadržaj procesorovih registara. Ovo se ostvaruje pomoću njegove naredbe **T** (trace).

Dakle, program debug.exe omogućuje dobar uvid u dešavanja u računaru odnosno otklanjanje grešaka. On je, u suštini, jedan asembler i *dibager*, tj.

ispravljač grešaka. Pomoću ovog programa mogu da se proizvode izvršni programi čija je ekstenzija .com, kao recimo primjera.com. U DOS–u postoje dvije ekstenzije koje odgovaraju izvršnim programima i to .com i .exe. Programi vrste .com jednostavniji su od onih .exe. Razlika je u tome što u slučaju .com program čitav stane u jedan segment memorije koji ima 64 KB. Pored toga, izvršavanje će početi na naredbi čiji je ofset (Hex) 0100. U slučaju .exe, program, generalno, zauzima nekoliko segmenata - ima posebne segmente za naredbe, za podatke i za stek; startna adresa treba da bude definisana; itd. Ako je već generisan fajl primjera.com onda se naravno može izaći iz dibagera i onda pozvati izvršni program primjera.com (iz operativnog sistema).

Postoje i drugi programi za propuštanje i popravljanje–dibagiranje programa sastavljenih na jeziku asemblera. Oni pored ostalog omogućavaju da se proizvede izvršni program čija je ekstenzija .exe. Primjeri takvih programa su symdeb, masm firme Microsoft, tasm firme Borland, nasm i a86.

Kazali smo da su Intelovi procesori familije x86, uključujući i Pentium, međusobno (hardverski) kompatibilni naviše: 80286 razumije program koji je nekad ranije bio sastavljen za 8086, i slično. Isto tako, razne verzije operativnih sistema DOS i Windows su sve zajedno međusobno (softverski) kompatibilne naviše.

Program debug.exe poziva se iz DOS–a odnosno, u slučaju Windowsa, iz MS-DOS prompta na običan način: iz bilo kog foldera otkucati debug (enter). Tada će se na ekranu pojaviti dibagerov prompt, znak '-', (minus). Tokom našeg rada, ekran je u jednostavnom tekstualnom modu. U prompt unosimo naredbu, recimo unesemo A 0100 (enter). Jedno slovo čini ime jedne naredbe, a ukupno ima oko 24 vrste naredbi.

Objasnimo detaljnije naredbe. Napomenimo prije toga da u programu debug.exe važi heksadekadni brojni sistem.

Sintaksa A [address] znači da počinje unošenje naredbi i podataka u memoriju, i to od naznačene adrese. Naredba ili svejedno podatak, čim ga mi ukucamo, pretvara se iz simboličkog, tj. asemblerskog oblika, u binarni (mašinski) oblik i unosi se (loaduje) na odgovarajuće mjesto u memoriji. Primjer, A 100. Na ekranu će biti prikazana adresa (fizička adresa) na koju će naredba da bude unesena. Tokom unošenja niza naredbi ta adresa se postepeno povećava, odnosno ona se sama podešava. Recimo, mi unosimo naredbu MOV AX, [200] (pritisnuti enter). Ili unosimo podatak DB FF. I

slično. Iz naredbe A izlazi se tako što se otkuca jedna prazna linija (pritisne se jedno enter više).

Uobičajeno je da loadovanje počinje od adrese 100. Obavezni smo da tako radimo jedino ako želimo da proizvedemo izvršni fajl vrste .com. Naime, za takve fajlove, DOS traži da se njemu ostavi prvih 256 bajta u segmentu, bajti od Hex 0 do Hex FF, jedna i druga granica su uključene. DOS koristi taj prostor za svoje potrebe. Taj prostor naziva se Program Segment Prefix ili skraćeno PSP. Možemo vidjeti da prva naredba u tom prostoru, na adresi 0, jeste INT 20. Sistemski prekid broj 20 okončava rad po izvršnom programu i vraća upravljanje operativnom sistemu.

Naredba G [=address] znači da će početi izvršavanje od naznačene adrese. Na primjer, primjer G =100. Sintaksa G [=address] [addresses] znači da je još zadato i nekoliko (od 1 do 10) tzv. tačaka prekida (breakpoints). Kada izvršavanje stigne do tačke prekida prikazaće se sadržaj procesorovih registara (kao u slučaju naredbe T) i onda će čekati da se pritisne enter.

Naredba D [range] prikazuje sadržaj naznačene oblasti memorije. Ako se otkuca samo jedna adresa onda se ona tumači kao prva adresa i onda se prikaže sadržaj 128 bajta. Sadržaj vidimo na ekranu u heksadekadnom obliku i u ASCII obliku. Na primjer, D 0 prikazaće 128 bajta od početka tekućeg segmenta. D 100 prikazaće sadržaj memorije od mjesta gdje je po pravilu počelo loadovanje naredbi i podataka. Slično, D 0F6E:0100 prikazaće od naznačenog mjesta u memoriji. Konačno, D 0:0 prikazaće sadržaj samog početka memorije.

Posljednji primjer D 0:0 je zanimljiv. Mi vidimo tabelu vektora prekida. Redom četiri po četiri vrijednosti čine jednu punu adresu. Prva četvorka odnosi se na INT 0, sljedeća na INT 1, itd. To je adresa na kojoj počinje rutina prekida. Ako četvorku čine vrijednosti a, b, c i d onda adresa glasi CS:IP = dc:ba. Jedna vrijednost su dva heksadekadna znaka.

Sintaksa T [=address] [number] upućuje debug.exe da izvrši naredbu koja stoji na naznačenoj adresi, odnosno da izvrši onoliko naredbi koliko kazuje number, i zaustavi se. Nakon toga, prikazuje se na ekranu: sadržaj svih procesorovih registara, posebno i sadržaje flagova, tekuću naredbu (naredbu koja je došla na red za izvršavanje) i još – ako se ta naredba obraća memoriji – i sadržaj odgovarajuće memorijske lokacije. Recimo, T =0123:89AB 40 izvršiće 40 naredbi a T =0123:89AB jednu, isto kao i T =140. Naravno, kako kod T=140 nije naznačen segment, izvršiće naredbu u tekućem segmentu čiji

je ofset (pomjeraj, relativna adresa) 140. Na kraju, prosto T izvršiće tekuću naredbu i prikazati sadržaje procesorovih registara.

Naredba U [range] ima smisao Unassemble, dizasembliraj, uradi suprotno od asembliranja. Ova naredba dizasemblira sadržaj naznačenog dijela memorije odnosno prevodi taj sadržaj iz binarnog oblika u simbolički oblik i prikazuje simbolički oblik na ekranu. Simbolički oblik je čitljiv za programera. Na primjer, U 100 110 prikazaće sadržaj 16 bajta memorije, od adrese 0100 u tekućem segmentu CS, do adrese 0110 u istom segmentu. Ako se izostavi druga adresa (bila je 110 maločas) onda će dizasemblirati 16 bajta (recimo, U 100). Slično, U 0123:89AB. Kao i ranije, ako se izostavi i prva adresa (na primjer U) onda se podrazumijeva da je ona jednaka tekućoj vrijednosti CS:IP.

Pomoću U možemo da vidimo šta smo sami ukucali u memoriju (recimo U 100). Pomoću U možemo da pročitamo kako glasi neki drugi program, samo treba da znamo njegovu prvu adresu. Pomoću U možemo da pročitamo kako glasi neka rutina za obradu prekida (recimo U 00C9:0F9E), jer smo prethodno saznali njihove početne adrese iz vektora prekida pomoću D 0:0 i D 0:80 i slično.

Naredbom Q koja ima smisao Quit izlazi se iz dibagera. Naredba ? na ekranu prikazuje moguće naredbe dibagera kao i njihovu sintaksa (Help).

Naredba R [register] prikazuje ili mijenja sadržaj jednog ili više registara. Naredba N (od Name) specificira ime fajla koje će biti iskorišćeno u L ili W naredbi.

Naredba W ima smisao Write. Ona piše fajl ili određeni broj sektora na određenom disku. Analogno, naredba L ima smisao Load, učitavanje.

Prvi primjer. Obični način rada sa programom debug.exe. Pozvati program. Otkucati A 100 (enter). Redom otkucati naredbe programa MOV AX, 150 MOV BX, 80 ADD AX, BX itd. INT 20 (enter) Otkucati enter (da se izađe iz A). Otkucati G (enter). Dobiće se rezultati i poruka Program terminated normally (naredba INT 20 poziva interapt za završetak rada programa). Otkucati Q (enter). Izašli smo iz programa.

Drugi primjer. Drugi način rada sa programom debug.exe je kada se proizvodi izvršni fajl. Poslužimo se prethodnim primjerom. Kada smo izašli iz A (Assemble) otkucajmo:

```
RCX
45
N PRIMJERX.COM
W
Q (enter)
```

Vrijednost 45 vezana je za konkretne okolnosti. Počeli smo da unosimo na adresu 100. Kada smo unijeli posljednju naredbu (to je bila naredba INT 20) onda je program spreman da iduću naredbu unese na adresu 145; to vidimo na ekranu. $145 - 100 = 45$. Pomoću R se registru CX dodjeljuje pravilna vrijednost. Ta vrijednost interveniše prilikom proizvodnje .com fajla. Vidimo da smo između "W" i "Q" na ekranu dobili poruku Writing 00045 bytes.

Sada se fajl primjerx.com nalazi zapisan na hard disku.

Pomenimo još da program debug.exe pored naredbi prihvata i dvije direktive: DB – define byte i DW – define word. Recimo, ako se na mjestu 1AA loaduje DB 7F onda će efekat tokom izvršavanja biti c(1AA) $\leftarrow$ 7F; jedan bajt. Slično, DW 6E7F ima efekat c(1AA) $\leftarrow$ 6E7F; dva bajta; opet je 1AA mjesto direktive.

11.7.1 Primjeri za debug.exe

Zadatak 1. Sastaviti program za računanje izraza $y = 2a + b + 1$. Izabrati ulazne podatke $a = 500$ i $b = 466$, tako da treba da se dobije rezultat $y = 1467$, dekadno. Heksadekadno: $a =$ 1F4, $b =$ 1D2, $y =$ 5BB.

Rješenje. U nastavku su tačno prikazane radnje sa uključenim računarom. Ono što računar sam prikazuje na ekranu dato je u fontu `typewriter`, a ono što mi unosimo preko tastature – dato je pomoću običnog fonta.

Opredijelili smo se da naredbe zauzimaju adrese od 100 pa naprijed. Promjenljive a, b i y (po dva bajta) su na mjestima 110, 112 i 114.

```
C:\WINDOWS>  debug (enter)
-  A (enter)
1368:0100  MOV AX,[110] (enter)
1368:0103  SHL AX,1 (enter)
1368:0105  ADD AX,[112] (enter)
1368:0109  INC AX (enter)
1368:010A  MOV [114],AX (enter)
```



```
1368:010D  INT 20 (enter)
1368:010F  (enter)
-  A 110 (enter)
1368:0110  DW 1F4 (enter)
1368:0112  DW 1D2 (enter)
1368:0114  (enter)
-  ♠ T =100 (enter)
AX=01F4 CS=1368 IP=0103  SHL AX,1  -  T (enter)
AX=03E8 CS=1368 IP=0105  ADD AX,[0112]  -  T (enter)
AX=05BA CS=1368 IP=0109  INC AX  -  T (enter)
AX=05BB CS=1368 IP=010A  MOV [0114],AX  -  T (enter)
AX=05BB CS=1368 IP=010D  INT 20  -  T (enter)
AX=05BB CS=00C9 IP=0FA8  NOP  - ♠  D 110 115 (enter)
1368:0110 F4 01 D2 01 BB 05 -  Q (enter)
C:\WINDOWS>
```

Dio ♠ ... ♠ može da bude zamijenjen sljedećim:

```
G =100 (enter)
Program terminated normally -
```

Objasnimo ukratko. Prvo se pomoću A unesu naredbe; Assemble. Zatim se pomoću A 110 unesu dva ulazna podatka a i b. Nakon toga se pomoću T =100, T, T, ... program izvršava korak po korak; Trace. Kada je i posljednja naredba izvršena, pogledajmo šta piše u memoriji, budući da nije predviđeno da se y odštampa. Zato kažemo D 110 115; Dump šest bajta. Kada se i to uradi, izlazimo iz debug.exe sa Q; Quit.

Izvršavanje programa korak po korak može da bude zamijenjeno običnim izvršavanjem. Tada treba upotrebiti komandu G =100; Go.

U nastavku je program ponovo napisan, nešto preglednije.

adresa	naredba	smisao	komentar
100–102	MOV AX,[110]	AX ← $c(110)$	AX ← a
103–104	SHL AX,1	shift left AX 1 place	AX ← $2 \cdot$ AX
105–108	ADD AX,[112]	AX ← AX + $c(112)$	AX ← AX + b
109	INC	increnent AX	AX ← AX + 1
10A–10C	MOV [114],AX	$c(114)$ ← AX	y ← AX
10D–10E	INT 20	interrupt # 20	stop
110–111	DW 1F4	$c(110)$ ← 1F4	$a = (1F4)_{16}$
112–113	DW 1D2	$c(112)$ ← 1D2	$b = (1D2)_{16}$
114–115	nema	nema	mjesto za y

Zadatak 2. U operativnim sistemima, koriste se tzv. parametri komandne linije. To je jedan važan mehanizam koji omogućava korisniku da, prilikom pozivanja izvršnog programa, definiše neke vrijednosti i da time utiče na obradu koju će program da ostvari. Na primjer, ako u MS–DOS–u otkucamo komandu dir *.tex onda je dir naziv komande a *.tex je njen parametar. Razmotrimo izvršni fajl čija je ekstenzija .com, kakav je recimo fajl vjezba.com u ovom zadatku. Tada se dio prostora PSP–a (kao što smo ranije naveli, Program Segment Prefix) koristi da prihvati parametre komandne linije. Preciznije, to je druga polovina PSP–a odnosno to je područje 80–FF (128 bajta). Dakle, prilikom pozivanja programa, parametri komandne linije prihvataju se tj. zapisuju se u to područje.

Primjer koji slijedi služi i da se ilustruje mogućnost, koju pruža debug.exe, kreiranja izvršnog fajla čija je ekstenzija .com.

Sastavimo program za množenje dva broja $y = a \cdot b$, gdje su a i b dvocifreni pozitivni brojevi.

Rješenje. Ulazne veličine a i b saopštiće se programu kao parametri komandne linije, jedna i druga veličina kao dvije dekadne cifre. Tako da će zauzeti adrese 82–85 u tekućem segmentu. Za rezultat y predviđena su sljedeća četiri bajta 86–89. Program će odštampati četiri dekadne cifre rezultata. U prvoj fazi, sastave se a i b i onda se pomnože. U drugoj fazi se y rastavi na cifre. U trećoj fazi se štampaju cifre y–a. Koriste se sljedeće formule.

ord('0')=48, ord('1')=49, ..., ord('9')=57; ASCII

$a = 10a_1 + a_2$, $b = 10b_1 + b_2$, $0 \le a_1, a_2, b_1, b_2 \le 9$

$y = 1000y_1 + 100y_2 + 10y_3 + y_4$, $0 \le y_1, y_2, y_3, y_4 \le 9$

$y = (y_1y_2y_3y_4)_{10}$, $a = (a_1a_2)_{10}$, $b = (b_1b_2)_{10}$, $y = a \cdot b$

y : 1000, brojilac = y, imenilac = 1000 ⇒ količnik = y_1, ostatak = r = $100y_2 + 10y_3 + y_4$

r : 100, brojilac = r, imenilac = 100 ⇒ količnik = y_2, ostatak = $10y_3 + y_4$

Plan za upotrebu memorije:

82	83	84	85	86	87	88	89	8A
a_1	a_2	b_1	b_2	y_1	y_2	y_3	y_4	a

Treba pozvati program debug.exe i treba otkucati naredbe, kao što je dato u nastavku. Napisani su i komentari. Pojasnimo još da **podebljani** font označava ono što dibager daje, `typewriter` – ono što mi unosimo, a obični font – komentare.

```
– A  Assemble
1368:0100 MOV BL,A       BL ← (A)16 = (10)10
1368:0102 MOV AL,[82]    AL ← c(82);  uzimam a1
SUB AL,30      AL ← AL – (48)10;  zbog ASCII
MUL BL         AX ← AL · BL
ADD AL,[83]    AL ← AL + c(83);  dodajem a2
SUB AL,30      AL ← AL – (48)10;  sada je AL = a
MOV [8A],AL    c(8A) ← AL;  ostavljam a
MOV AL,[84]    AL ← c(84);  uzimam b1
SUB AL,30      AL ← AL – (48)10
MUL BL         AX ← AL · BL
ADD AL,[85]    AL ← AL + c(85);  dodajem b2
SUB AL,30      AL ← AL – (48)10;  sada je AL = b
MOV BL,[8A]    BL ← c(8A);  sada je BL = a
MUL BL         AX ← AL · BL;  sada je AX = y
MOV BX,3E8     BX ← (1000)10
XOR DX,DX      DX ← 0
DIV BX         b = DX AX, i = BX, k → AX, o → DX
ADD AL,30      AL ← AL + (48)10;  sada je AL = y1
MOV [86],AL    c(86) ← AL;  ostavljam y1
MOV AX,DX      AX ← DX;  sada je AX = r
MOV BX,64      BX ← (100)10
XOR DX,DX      DX ← 0
```

```
DIV BX          b = DX AX, i = BX, k → AX, o → DX
ADD AL,30       AL ← AL + (48)10;  sada je AL = y2
MOV [87],AL     c(87) ← AL;  ostavljam y2
MOV AX,DX       AX ← DX;  sada je AX = 10y3 + y4
MOV BX,A        BX ← (10)10
XOR DX,DX       DX ← 0
DIV BX          b = DX AX, i = BX, k → AX, o → DX
ADD AL,30       AL ← AL + (48)10;  sada je AL = y3
MOV [88],AL     c(88) ← AL;  ostavljam y3
MOV AX,DX       AX ← DX;  sada je AX = y4
ADD AL,30       AL ← AL + (48)10;  zbog ASCII
MOV [89],AL     c(89) ← AL;  ostavljam y4
MOV AH,2        AH ← 2;  funkcija = štampanje karaktera
MOV DL,[86]     DL ← c(86);  y1 = karakter koji se štampa
INT 21          interrupt # 21 (input-output funkcije)
MOV DL,[87]     DL ← c(87)
INT 21          interrupt # 21
MOV DL,[88]     DL ← c(88)
INT 21          interrupt # 21
MOV DL,[89]     DL ← c(89)
INT 21          interrupt # 21
1368:016E INT 20        interrupt # 20 (return to DOS)
1368:0170               (preskoči jedan red)
- R CX  Register; da se CX prikaže i promijeni
CX 0000  vidimo da je CX = 0
: 70  mijenjamo CX; CX ← (70)16; biće zapisano CX bajta
- N VJEZBA.COM  Name; želimo da se fajl zove vjezba.com
- W  Write; da se zapiše na disk
Writing 00070 bytes  sada računar vrši zapisivanje fajla
- Q  Quit
```

Sada smo izašli iz dibagera. Naredba Write služi da se fajl vjezba.com zapiše na disk. Prilikom radnje zapisivanja fajla, dibager se upravlja po tekućoj vrijednosti registra CX. Veličina fajla biće jednaka tekućoj vrijednosti registra CX.

Osvjedočiti se da je sada na hard-disku (u tekućem direktorijumu - folderu) prisutan novi fajl vjezba.com i da zauzima 112 bajta, govoreći dekadno.

Za udobniji rad sa dibagerom, možemo koristiti DOS-ovu pogodnost preusmjeravanja ulaza i izlaza. Prvo, ono što mi treba da unesemo, saglasno prethodnom rješenju, zasebno otkucati da čini fajl čiji je naziv (recimo) vjez. Zatim u ms-dos promptu zadati: debug < vjez > vjezb (enter). Na kraju, editovati fajl vjezb, da se vidi šta je debug uradio. Ovim smo dibageru kazali da preuzme ulaz iz fajla vjez (a ne sa standardnog ulaza, dakle -preusmjerili smo ulaz) i da ga pošalje u fajl vjezb (a ne na standardni izlaz).

Vidimo da se u predloženom rješenju koriste dva sistemska poziva INT 20 i INT 21. Njihove precizne definicije biće date kasnije, u ovom poglavlju. Najkraće, INT 21 znači output DL, ako je u datom trenutku AH = 2.

Posao oko rješavanja zadatka još nije završen. Vidimo da dosad program nije izvođen, već da smo samo govorili o dešavanjima tokom njegovog izvođenja. Sada program udobno izvršavamo iz ms-dos prompta, koliko god puta hoćemo, zadajući ulazne veličine a i b kao parametre komandne linije. Ako otkucamo

vjezba 1234 (enter)

onda će se na ekranu odštampati 0408. Ako zadamo

vjezba 1017 (enter)

onda će računar saopštiti 0170. Navedimo još nekoliko slučajeva:

1307 ⇒ 0091, 2127 ⇒ 0567,
3771 ⇒ 2627, 6982 ⇒ 5658,
9999 ⇒ 9801, 0A0B ⇒ 0306,
1A0B ⇒ 0486, 0a0B ⇒ 0882.

Na kraju, vidimo da programiranje pomoću programa debug.exe nije udobno i da uzima puno vremena. Bolje je da potražimo malo jači program, koji će nas osloboditi adresne računice. Primjer takvog programa je Borland tasm.

11.8 Asemblerski jezik 8086

U ovoj sekciji razmotrićemo elemente asemblerskog jezika 8086 procesora. Asembler koji se ovdje razmatra jeste turbo-asembler (skraćeno tasm) firme Borland.

11.8.1 Format asemblerske linije

Svaka asemblerska instrukcija ili direktiva zauzima jednu asemblersku liniju. I instrukciji i direktivi može da prethodi labela. Labelu od instrukcije, pored razmaka, razdvaja i dvotačka. U asemblerskoj liniji se može nalaziti i komentar, koga najavljuje tačka zarez.

11.8.2 Segmenti

Za definisanje segmenata uvedene su posebne pretprocesorske direktive.

Programski modeli

Definicija programskog modela saopštava asembleru veličinu i broj segmenata koji će biti korišteni. Model definiše direktiva **.MODEL** koja uvodi naziv modela :

TINY model sadrži jedan segment od 64 Kb u kome su i podaci i kod.

SMALL model sadrži dva segmenta od po 64 Kb, jedan za podatke, a drugi za kod.

COMPACT model sadrži dva segmenta. Jedan od 1 Mb je namenjen za podatke, a drugi od 64 Kb je namenjen za kod.

MEDIUM model sadrži dva segmenta. Jedan od 64 Kb je namenjen za podatke, a drugi od 1 Mb za kod.

LARGE model sadrži dva segmenta od po 1 Mb, jedan za podatke, a drugi za kod.

HUGE model se razlikuje od **LARGE** modela po tome što dozvoljava definisanje nizova dužih od 64Kb.

U primjerima u ovom poglavlju će se koristiti **SMALL** model.

Kraj programa

Kraj svakog programa označava direktiva **END**. Nakon ove direktive može slijediti labela koja ukazuje na lokaciju od koje započinje izvršavanje programa.

Asembler ignoriše tekst koji slijedi iza ove direktive.

Segment koda

Direktiva **.CODE** označava početak segmenta koda, u kome se nalazi izvršni kod našeg programa. Nakon ove direktive slijede instrukcije asemblerskog jezika.

Segment podataka

Postoji više vrsta segmenata podataka. Svaki od njih najavljuje posebna direktiva:

.DATA - označava početak segmenta podataka sa inicijalizovanim podacima, odnosno onim podacima čija početna vrijednost je prilikom prevođenja poznata.

.DATA? - označava početak segmenta podataka sa neinicijalizovanim podacima. Za ove podatke znamo samo koliku memoriju zauzimaju, a početne vrijednosti nisu bitne.

.CONST - definiše početak segmenta konstantnih podataka.

Segmenti **.DATA**, **.DATA?**, **.CONST** spadaju u grupu **DGROUP**, odnosno grupu segmenata podataka.

Primjer korišćenja direktiva:

```
.model small
.data
.code
mov ax,@data
mov ds,ax
end
```

Jedine dvije instrukcije iz ovog primjera pune adresu segmenta podataka, koju označava ime **@data**, u **DS** registar.

Ime **@data** označava segment koji čine članovi **DGROUP**. Adresa nije smještena direktno u segmentni registar, zbog ograničenja procesora pa se dodjeljivanje vrši preko akumulatora.

Stek segment

Direktiva **.STACK** rezerviše 1 Kb memorije namenjene steku. Ukoliko je potrebno da stek bude duži, njegova dužina se navodi nakon direktive.

Ovako definisani stek takođe spada u grupu **DGROUP** segmenata podataka.

Registri SS i SP se postavljaju automatski prilikom prevođenja, i to SS na početak stek segmenta, a SP na dužinu stek segmenta.

11.8.3 Konstante

Uz instrukciju se kao parametar mogu navesti kako registri i simboli tako i neposredne vrijednosti, odnosno konstante. Njih možemo svrstati u više grupa koje se odlikuju svojom specifičnom upotrebom.

Numeričke i znakovne konstante

Dekadne konstante nastajuređanjem dekadnih cifara, koje se mogu završiti i znakom **d**.

Heksadecimalne konstante označava postfiks **h**, odnosno **H**. Pri tome, konstanta započinje cifrom od **0** do **9**, dok ostale cifre mogu biti od **0** do **F**.

Oktalne konstante označava postifks **q**, odnosno **o**. Dozvoljene cifare su od **0** do **7**.

Binarne konstante se sastoje od cifara **0** i **1**, iza kojih sljedi znak **b**.

Stringovi (znakovne konstante) se navode između jednostrukih ili dvostrukih znakova navoda.

Direktiva **EQU** omogućuje davanje imena konstantama. Ispred direktive se navodi ime konstante, a nakon nje konstanta (broj ili string).

Predefinisane konstante

Pored imena **@data** za adresu segmenta podataka predefinisane su i sljedeće konstante:

$ - adresa tekuće linije koda
@code - adresa segmenta koda
@curseg - adresa aktuelnog segmenta
??date - tekući datum
@FileName - ili **??filename** string sa nazivom asemblirane datoteke.
@Startup - adresa početka koda.
??time - string tekućeg vremena.

11.8.4 Zauzimanje memorije za promjenljive

Skalarne promjenljive

Direktiva **DB** omogućuje zauzimanje jednog bajta za promjenljivu čije ime se navodi ispred direktive, a iza nje se po potrebi navodi početna vrijednost promjenljive. Ako se početna vrijednost ne navodi, iza direktive slijedi znak pitanja. Primjeri direktive **DB**:

```
....
.data
x db 5
y db 2
z db ?
....
```

U prethodnom primjeru se u segmentu podataka zauzimaju tri uzastopna bajta, u kojima će se nalaziti naše promjenljive (x, y i z). Imena x, y i z predstavljaju adrese promjenljivih.

Direktiva **DW** omogućuje zauzimanje riječi, tj. dva bajta, a direktiva **DD** omogućuje zauzimanje 4 bajta.

Pokazivači

Zauzeta memorijska lokacija može da sadrži pokazivač. To se opisuje oznakom **PTR** iza koje slijedi opis pokazivane lokacije. Tako **x dw ptr word** označava zauzimanje memorijske lokacije od dva bajta koja sadrži adresu lokacije od dva bajta.

Nizovi

Do zauzimanja niza lokacija dolazi ako se iza direktive za zauzimanje jedne navede broj elemenata niza, a zatim riječ **DUP** iza koje u zagradi sljedi vrijednost na koju se inicijalizuju elementi niza :

```
....
.data
niz db 100 dup(?)
....
```

U prethodnom primjeru nam je definisan jedan niz od 100 bajta, bez inicijalizacije.

Pristupanje elementu niza se označava kao **ime_niza[indeks]**.

11.8.5 Uslovno asembliranje

U nastavku su navedene direktive za uslovno asembliranje:

IF - programski blok se asemblira u slučaju da je navedeni uslov ispunjen.

IFDEF - programski blok se asemblira u slučaju da je navedeni simbol definisan.

IFNDEF - asembliranje programskog bloka se vrši ako vrijednost simbola nije definisana.

Nakon navođenja programskog bloka koji će se asemblirati ako je neki od uslova ispunjen, može slijediti i programski blok, koji se asemblira u suprotnom slučaju. To označava direktiva **ELSE**.

Svaki programski blok za uslovno asembliranje se završava direktivom **ENDIF** :

```
.data
x dw 2
y dw 3
z dw ?
.code
ifdef saberi
mov ax,x
add ax,y
mov z,ax
else
mov z,0
endif
```

U prethodnom primjeru biće asembliran prvi programski blok (u kome se zbir promjenljivih **x** i **y** smješta u **z**), ako je definisan simbol **saberi**. Inače će biti asembliran drugi programski blok (u kome se anulira **z**).

11.8.6 Makroi

Makro definicija se nalazi između direktiva **MACRO** i **ENDM**. Prije prve direktive je potrebno navesti jedinstveni naziv makroa, a nakon nje listu

parametara koji će se u samom makrou koristiti. Druga direktiva označava kraj makro definicije i ograničava samo tijelo makroa. Slijedi primjer korišćenja ovih direktiva:

```
.data
x dw 2
y dw 3
z dw ?
.code
saberi macro a,b,c
mov ax,a
add ax,b
mov c,ax
endm
...
saberi x,y,z
```

Nakon asembliranja prethodnog primjera poziv makroa **saberi x,y,z** će biti zamenjen instrukcijama:

```
mov ax,x
add ax,y
mov z,ax
```

Znači, poziv makroa je zamijenjen tijelom makroa u kome su parametri zamijenjeni argumentima.

Direktiva **LOCAL** omogućuje uvođenje lokalnih labela u tijelu makroa. Jedan primjer njene upotrebe je dat u sljedećem primjeru:

```
pauza macro a
local umanji
mov cx,a
umanji:
dec cx
jnz umanji
endm
...
pauza 100
pauza 20
```

U prethodnom primjeru labela **umanji** je proglašena lokalnom, pa višestruki pozivi makroa **pauza** ne izazivaju probleme u asembliranju.

11.8.7 Potprogrami

Definicija prtprograma otpočinje direktivom **PROC** prije koje slijedi njeno ime. Nakon ove direktive može da slijedi lista parametara.

Potprogrami se pozivaju instrukcijom **CALL**.

Listu parametara najavljuje direktiva **ARG**. Nakon nje slijede imena parametara. Za njih se podrazumijeva da zauzimaju dva bajta. Iza parametra se može navesti njegov opis, koga najavljuje dvotačka.

Definicija potprograma se završava direktivom **ENDP**, kojoj prethodi naziv potprograma čiji kraj označava ova direktiva.

U nastavku slijedi potpun primjer asemblerskog programa (za $z \leftarrow x+y$) sa definicijom i pozivom potprograma **saberi**:

```
        .model small        64 KB za naredbe i 64KB za podatke
        .stack              1KB za stek
        .data               od pqrs:0000 do pqrs:ffff su podaci
        x dw 2              2B za x i x ← 2
        y dw 3              2B za y i y ← 3
        z dw ?              2B za z
        .code               sada dolaze naredbe
        saberi PROC NEAR    zaglavlje potprograma
                            parametri potprograma
        ARG a,b,c : NEAR    PTR WORD = ARGLEN
        push bp             spašavanje vrijednosti
        mov bp,sp           za argumente, bazno adresiranje
        mov ax,a            donesi prvi arg. u akumulator
        add ax,b            uvećaj akumulator za drugi arg.
        mov bx,c            donesi treći argument
        mov [bx],ax         iz akumulatora u memoriju
        pop bp              vraćanje spašene vrijednosti
        ret ARGLEN          SP ← SP + 6 i povratak
        saberi ENDP         kraj potprograma
start:                      ovo je labela
        mov ax,@data        ax ← pqrs
        mov ds,ax           ds ← ax
        lea ax,z            ax ← adresa(z)
        push ax             stavi na stek i SP ← SP − 2
        mov ax,y            ax ← y
        push ax             stavi na stek i SP ← SP − 2
        mov ax,x            ax ← x
        push ax             stavi na stek i SP ← SP − 2
        call saberi         poziva se potprogram
        mov ax,4c00h        AH=4C funkc. prekida=Terminate process
        int 21h             prekid broj 21
        end start           kraj programa i start je ulazna tačka.
```

U potprogramu **saberi** se podrazumijeva da se argumenti prenose posredstvom steka. Zbog toga je potrebno korišćenje **BP** registra. Za sva tri parametra se podrazumijeva da se odnose na lokacije steka veličine po dva bajta. Za treći parametar navedeno je da sadrži adresu lokacije veličine dva bajta. **ARGLEN** označava broj bajta potreban za smještanje argumenata na stek. On se koristi kao operand instrukcije **RET** da bi se stek ispraznio za pomenuti broj bajta. Posljednje dvije instrukcije u asemblerskom programu omogućuju da se, po završetku izvršavanja asemblerskog programa, nastavi izvršavanje operativnog sistema.

11.9 Primjeri programa

11.9.1 Dodjela vrijednosti

Prilikom računanja matematičkih izraza za privremeno čuvanje vrijednosti obično koristimo neki od registara opšte namjene, a najčešće akumulator. Da bi se vrijednost registra inicijalizovala, odnosno postavila na nulu, koristi se sljedeći izraz, konkretno za akumulator: mov ax,0 .

Međutim, to i nije najsrećnije rešenje. Eksplicitno dodjeljivanje brojne vrijednosti nekom registru zahtijeva postojanje neposrednog operanda u okviru mašinske riječi. Kod dodjeljivanja vrijednosti različite od nula, to je neizbježno, međutim, ako je vrijednost jednaka nuli to se može izvesti i upotrebom operacije "ekskluzivno ili" nad registrom, dok je i drugi parametar isti taj registar, na primjer xor ax,ax. Kao što se može odmah primijetiti, ova instrukcija ne zahtijeva nikakvu brojnu vrijednost, te je samim tim njen mašinski kod kraći. Ona se zbog svojih osobina mnogo češće sreće nego instrukcija prije nje.

Dodjelu vrijednosti svim elementima niza opisuje sljedeći asemblerski program :

```
        .model small
        .stack
        .data
        niz db 100 dup(?)
        .code
start:
        mov ax,@data
        mov ds,ax
        mov al,5
        xor bx,bx
sljedeci:
        mov niz[bx],al
        inc bx
        cmp bx,LENGTH niz
        jne sljedeci
        mov ax,4c00h
        int 21h
        end start
```

U primjeru smo sve elemente niza od sto elemenata inicijalizovali na vrijednost 5, koja se nalazi u akumulatoru. Kao što se može primijetiti, pristup svakom elementu je zahtijevao računanje nove vrijednosti indeksa niza, smještenog u registru BX. Da smo imali elemente dužine dva bajta, registar BX bi morali uvećavati za dva. Oznaku **LENGTH** prije imena niza u asembliranju zamenjuje dužina niza.

Naravno, ovo nije jedini način za inicijalizaciju niza. Postoji i posebna grupa instrukcija za rad sa nizovima, koju smo imali prilike ranije da upoznamo, a naredni primjer predstavlja ekvivalent prethodnog.

```
        .model small
        .stack
        .data
        niz db 100 dup(?)
        .code
start:
        mov ax,@data
        mov ds,ax
        mov es,ax
```



```
        mov al,5
        cld
        mov cx,LENGTH niz
        mov di,offset niz
        rep stosb
        mov ax,4c00h
        int 21h
        end start
```

Pošto instrukcija **STOS** piše sadržaj akumulatora na poziciju označenu registarskim parom ES:DI morali smo prije početka inicijalizacije postaviti vrijednost datog segmentnog registra da pokazuje na naš segment podataka. Korišćenje prefiksa **REP** prije instrukcije **STOS** uzrokuje nekoliko stvari. Registar CX koristimo za brojanje, jer je on implicitni parametar operacije repeticije. Osim što se vrijednost brojačkog registra smanjuje, mijenja se i vrijednost indeksnog registra odredišta, tj. DI, i to u onom smjeru koji je određen stanjem bita smjera status registra. Da bi brojanje bilo ka višim adresama morali smo postaviti taj bit na nulu. Pri tome se podrazumijeva da se za instrukciju **STOSB** vrijednost **DI** registra umanjuje za jedan.

11.9.2 Kopiranje vrijednosti

Kopiranje vrijednosti se vrši u sljedećoj formi:

```
mov ax, ime_prom1 ili
mov ime_prom2, ax
```

To naravno važi za slučaj kada se radi o skalaru dužine dva bajta, međutim, kada je u pitanju bajt potrebno je koristi neki od 8-bitnih registara kao pomoćni.

Moguće je iskopirati i vrijednost promjenljive manje dužine u onu veće dužine. Da bi se to ostvarilo moramo koristiti i neku od instrukcija za konverziju, kao što je to pokazano u sljedećem primjeru.

```
.model small      (program izvodi b ← a i c_visi:c_nizi ← b)
.stack
.data
a db 0
b dw ?
c_nizi dw ?
```

```
        c_visi dw ?
        .code
start:
        mov ax,@data
        mov ds,ax
        mov al,a
        cbw
        mov b,ax
        cwd
        mov c_nizi,ax
        mov c_visi,dx
        mov ax,4c00h
        int 21h
        end start
```

Ovaj primjer nam pokazuje kako se vrijednost dužine bajta upisuje u promjenljive dužine dva ili četiri bajta. Treba naglasiti da je to ovdje učinjeno za promjenljive sa označenim vrijednostima, jer instrukcije CBW i CWD proširuju vrijednost na veću širinu obraćajući pažnju na znak.

Za kopiranje nizova može se koristiti i instrukcija **MOVS**. Pri tome se podrazumijeva da registarski par DS:SI sadrži adresu početka niza koji kopiramo, dok registarski par ES:DI sadrži adresu odredišnog niza.

```
        .model small      (program izvodi niz2[k] ← niz1[k] za 1 ≤ k ≤ 10)
        .stack
        .data
        niz1 db 34,3,46,2,3,6,3,6,5,2
        niz2 db 10 dup(0)
        .code
start:
        mov ax,@data
        mov ds,ax
        mov es,ax
        cld
        mov cx,LENGTH niz2
        mov si,offset niz1
        mov di,offset niz2
        rep movsb
```



```
        mov ax,4c00h
        int 21h
        end start
```

Kada se na ovaj način radi kopiranje nizova treba obratiti da će operatori **LENGTH**, a i **SIZE**, vratiti ispravnu dužinu jedino kod nizova koji su definisani pomoću **DUP** direktive. Nizovi definisani nabrajanjem elemenata se tumače tako da jedino prvom polju odgovara ime niza, a ostali se računaju kao neimenovani, pa iz tog razloga prevodilac vraća dužinu samo prvog elementa.

11.9.3 Asemblerski oblik IF-THEN-ELSE iskaza

Za sljedeći program na Pascal-u:

```
if (((x+4)<> (y*2)) and (x < 10)) then z := x + y
else z := 0;
```

ekvivalentan asemblerski program može da izgleda ovako:

```
        .model small        64 KB za naredbe i 64KB za podatke
        .stack              1KB za stek
        .data               od pqrs:0000 do pqrs:ffff su podaci
        x dw 2              2B za x i x ← 2
        y dw 6              2B za y i y ← 6
        z dw ?              2B za z
        .code               sada dolaze naredbe
start:                      ovo je labela
        mov ax,@data        donesi adresu segmenta za podatke
        mov ds,ax           smjesti u ds
        mov ax,x            donesi x u akumulator
        add ax,4            uvećaj akumulator za 4
        mov bx,y            donesi y
        shl bx,1            lijevi šift za jedno mjesto
        cmp ax,bx           upoređivanje da se podese flagovi
        je netacno          ako je ax = bx onda skoči na netačno
        mov ax,x            donesi x u akumulator
        cmp ax,10           upoređivanje da se podese flagovi
        jge netacno         ako je ax ≥ 10 onda skoči na netačno
```

```
tacno:                  ovo je labela
        mov ax,x        donesi x u akumulator
        add ax,y        uvećaj akumulator za y
        mov z,ax        smjesti akumulator u z
        jmp short kraj  bezuslovni kratki skok na kraj
netacno:                ovo je labela
        mov z,0         z ← 0
  kraj:                 ovo je labela
        mov ax,4c00h    AH=4C funkc. prekida= Terminate process
        int 21h         prekid broj 21
        end start       kraj programa i start je ulazna tačka.
```

11.9.4 Asemblerski oblik FOR iskaza

Za sljedeću sekvencu u Pascal-u:

```
a:=0;
for a:=0 to 9 do
begin
  z:=x;
  x:=x+1;
end;
```

ekvivalentni asemblerski program može da izgleda:

```
        .model small
        .stack
        .data
        x dw 3
        z dw ?
        .code
start:
        mov ax,@data
        mov ds,ax
        mov cx,0
        jmp short provjera
tijelo:
        mov ax,x
        mov z,ax
        inc x
        inc cx
provjera:
        cmp cx,10
        jl tijelo
        mov ax,4c00h
        int 21h
        end start
```


Prethodni asemblerski program može se skratiti ako se koristi instrukcija **LOOP** (koja kao brojač iteracija koristi registar CX, a čiji operand je labela početka tijela petlje):

```
        .model small
        .stack
        .data
        x dw 3
        .code
start:
        mov ax,@data
        mov ds,ax
        mov cx,10
tijelo:
        inc x
        loop tijelo
        mov ax,4c00h
        int 21h
        end start
```

Ako se brojač iteracija ne smjesti u registar, prethodni asemblerski program može da izgleda:

```
        .model small
        .stack
        .data
        x dw 3
        z dw ?
        a dw ?
        .code
        start:
        mov ax,@data
```

```
        mov ds,ax
        mov a,0
        jmp short provjera
tijelo: mov ax,x
        mov z,ax
        inc x
        inc a
provjera:
        mov ax,a
        cmp ax,10
        jl tijelo
        mov ax,4c00h
        int 21h
        end start
```

11.9.5 Prenos argumenata preko registara

Argumenti se mogu slati preko registara u potprogram samo ako ima dovoljno registara. U primjeru koji slijedi argumenti potprograma se prosleđuju preko registara (pauza dužine 1000):

```
        .model small
        .stack
        .code
        pauza PROC
p :
        dec cx
        jnz p
        ret
        pauza ENDP
start:
        mov cx,1000
        call pauza
        mov ax,4c00h
        int 21h
        end start
```


11.9.6 Vraćanje rezultata potprograma preko registara

U primjeru asemblerskog programa koji slijedi naveden je potprogram **saberi** koji u akumulatoru vraća sumu (koja je rezultat izvršavanja ovog potprograma). Parametri potprograma su dva sabirka.

```
        .model small
        .stack
        .data
        x dw 2
        y dw 3
        z dw ?
        .code
        saberi PROC NEAR
        ARG a : WORD,b : WORD = ARGLEN
        push bp
        mov bp,sp
        mov ax,a
        add ax,b
        pop bp
        ret
        saberi ENDP
start:
        mov ax,@data
        mov ds,ax
        mov ax,y
        push ax
        mov ax,x
        push ax
        call saberi
        mov z,ax
        mov ax,4c00h
        int 21h
        end start
```

11.10 Prekidne rutine

Kao što znamo, prekidne rutine predstavljaju jednu posebnu vrstu potprograma. Postoje dvije vrste prekidnih rutina.

Prva, po kojoj su i dobile ime, predstavlja rutine koje se pozivaju u slučaju nekog hardverskog događaja. Takve rutine se nazivaju obrađivačima prekida.

Drugu grupu čine tzv. servisi, odnosno sistemski potprogrami, koji su napisani kako bi se olakšala komunikacija između programera i operativnog sistema. Kod njih se komunikacija odvija preko registara, koji sadrže ili vrijednosti potrebnih argumenata, ili adrese memorijskih zona gdje bi servis trebao da vrati rezultat. Ove rutine se ne pozivaju po imenu, već po rednom broju, za što služi instrukcija INT, nakon koje se navodi broj prekidne rutine. Svaki servis može posjedovati više različitih funkcija, a one se biraju, najčešće, na osnovu sadržaja višeg dijela akumulatora (AH).

Na samom početku adresnog prostora ovog procesora se nalazi tabela prekida sa adresama prekidnih rutina. Svaka adresa zauzima 4 bajta, dva za ofset, a dva za segment.

11.11 Sistemski pozivi i obrada izuzetaka

11.11.1 Unutrašnji prekidi

Prekide iz ove grupe izaziva procesor. To se dešava kao posljedica nekog internog događaja. Obično ih obrađuje operativni sistem.

Prekidi koje ćemo nabrojati su vezani za procesor 8086/8088 (kod procesora novijih generacija uvođeni su i dodatni prekidi).

INT 00h - Divide by Zero Ovaj prekid izazivaju instrukcije za dijeljenje, ukoliko je djelilac jednak nuli. Zadatak operativnog sistema u tom slučaju je da prekine izvršenje programa, i da do znanja korisniku da je došlo do dijeljenja nulom. Međutim, ovaj prekid može biti preuzet i od strane korisničkog programa, tako da on sam može da prijavi grešku i da se od nje oporavi u toku svog izvršenja.

INT 01h - Single Step Ukoliko je postavljen trap bit status registra, doći će do izazivanja ovog prekida nakon svake izvršene instrukcije. Na ovaj način je omogućen rad dibagera, i izvršavanje instrukciju po instrukciju unutar njih.

INT 02h - NMI Ovaj prekid je tzv. nemaskirajući te je nemoguće programski zabraniti njegovu pojavu. Zato se bit zabrane prekida status registra ne odnosi na njega. On se koristi za regulisanje kritičnih događaja za sam rad računara na hardverskom nivou.

Na samom procesoru postoji posebna nožica na koju se dovodi signal usljed nekog kritičnog događaja. Kod PC računara ovaj prekid se generiše ukoliko dođe do problema sa memorijskim modulima, greška pariteta, ili kao posljedica neispravnog rada koprocesora, ukoliko on postoji.

INT 03h - Breakpoint Ovaj prekid nam takođe služi za rad samog dibagera. Njega izaziva posebna naredba i tako prekida izvršavanje programa i predaje kontrolu dibageru.

INT 04h - Overflow Izaziva ga instrukcija **INTO**, ukoliko je bit prekoračenja opsega status registra postavljen. On nam služi za kontrolu grešaka, nakon bilo kakvog niza aritmetičkih operacija.

11.11.2 Spoljašnji prekidi

U ovoj grupi su opisani prekidi tipični za PC računare.

INT 08h - System timer Ovaj prekid je vezan na IRQ0 nožicu kontrolera prekida, i na nju se dovodi signal sa tajmera. Ovaj prekid se dešava svakih 55ms, odnosno 18.2 puta u sekundi.

Kao rezultat svog rada postavlja u memoriji vrijednost tajmera u otkucajima, koji se nalazi na adresu 0040h:006Ch i dužine je duple riječi. On još poziva i prekid 1Ch koji je korisnički prekid i koji slobodno možemo mijenjati, dok se ne preporučuje mijenjanje tijela obrađivača ovog prekida.

Od svih harderskih prekida ovaj ima najviši prioritet.

INT 09h - Keyboard data ready Ovaj prekid se generiše prilikom pritiskanja ili otpuštanja tastera. Vrijednosti pritisnutih tastera smješta u bafer. Ukoliko je došlo do pritiskanja kombinacije tastera Ctrl-Alt-Del tada dolazi do resetovanja računara. Ukoliko je pritisnut taster Print Screen izaziva se prekid 05h, a ukoliko je pritisnuta kombinacija Ctrl-Break izaziva prekid 1Bh

INT 0Bh - Serial port COM2 / COM4 Ovaj prekid se dešava kada dođe do prijema, odnosno slanja znaka preko serijskog porta COM2/COM4.

INT 0Ch - Serial port COM1 / COM3 Ovaj prekid se dešava kada dođe do prijema, odnosno slanja znaka preko serijskog porta COM1/COM3.

11.11.3 BIOS servisi

Kao sastavni dio svakog PC računara nalazi se i BIOS (Basic Input Output System), u okviru koga se nalazi inicijalizacioni program, kao i program namenjen omogućavanju podizanja operativnog sistema. Osim ovih rutina, BIOS preuzima kontrolu nad jednim brojem prekida, kako bi bio u stanju da pruža neke od osnovnih servisa za pristup hardveru računara.

Bez upotrebe ovih servisa rad sa računarom bi bio veoma komplikovan i mukotrpan, a programi razvijeni za jednu konfiguraciju računara se ne bi mogli izvršavati na drugoj. Ovom prilikom ćemo navesti samo one prekide koji se najčešće koriste i postoje u svim varijantama BIOS-a. Naravno, kako se tehnologija mijenja tako se mijenja i sadržaj BIOS-a, i dodaju se dodatni servisi za podržavanje novih mogućnosti.

Sistemski servisi

INT 19h - Bootstrap loader Nakon što se izvrši testiranje memorije i kompletne opreme računara, i utvrdi da je sve u redu, vrši se pozivanje ovog servisa. Njegov zadatak je da u memoriju računara sa diskete ili fiksnog diska učita kod koji služi za učitavanje operativnog sistema, i pokrene ga.

INT 11h - Equipment check Ovaj servis iz memorijske zone dodjeljene BIOS-u čita podatak o tekućoj opremi računara i smješta je u AX registar. Svaki bit ima posebno značenje, na primjer bit 1 govori da li postoji koprocesor.

INT 12h - Memory size Kao rezultat poziva ovog servisa dobijamo u akumulatoru dužinu osnovne memorije. Treba naglasiti da je maksimalna vrijednost ovdje 1 MB, a da se sve iznad toga tumači kao proširena memorija.

Video izlaz

INT 10h - Video services. Ovaj servis nudi niz funkcija koje omogućuju ispis na ekran. Funkcije se biraju zavisno od sadržaja registra AH.

Na primjer, **AH = 00h - Set video mode** postavlja video mod, odnosno načina rada sa monitorom. Tako, ako je AL=03h podešavamo video mod na tekst, 80x25, sa 16 boja.

Sljedeći primjeri funkcija su: **AH = 01h - Set cursor type**, koja mijenja oblik tekstualnog kursora ili ga potpuno isključuje; **AH = 02h - Set cursor position** koja postavlja kursor na neku konkretnu poziciju; **AH = 03h - Read cursor position** kojom očitavamo podešavanja od prethodne dvije procedure. Funkcija **AH = 05h - Select active display page** omogućuje paralelno korišćenja više grafičkih strana, a funkcije **AH = 06h - Scroll up window AH = 07h - Scroll down window** omogućuju skrolovanje (pomjeranje sadržaja) ekrana nagore i nadolje.

Funkciju **AH = 06h - Scroll up window** najčešće koristimo za brisanje ekrana. Ona očekuje da na ulazu dobije u AL broj linija koliko treba skrolovati dati prozor, ako je nula, znači da se traži brisanje. A prozor se definiše pomoću dvije tačke, gdje su CH,CL koordinate gornjeg lijevog ugla, a DH,DL donjeg desnog ugla. U BH se stavlja atribut koji treba da se postavlja u prazne linije.

```
clrscr PROC
mov ax,0600h
mov bh,07h
xor cx,cx
mov dh,24
mov dl,79
int 10h
mov ah,2
xor bh,bh
xor dx,dx
int 10h
ret
clrscr ENDP
```

Navedena procedura briše sadržaj tekstualnog ekrana u standardnoj rezoluciji 80x25.

Dalje, funkcija **AH = 08h - Read character and attribute** očitava znak na trenutnoj poziciji kursora, kao i njegov atribut. Atribut može da se sastoji od dva dijela, viši dio bajta je boja pozadine, a niži dio boja teksta. Funkcija **AH = 09h - Write character and attribute** ispis znaka na ekranu i to sa određenim atributom, dok funkcija **AH = 0Ah - Write character only** ispisuje znak ali, za razliku od prethodne funkcije, nije potrebno navoditi atribut znaka koji se ispisuje. Funkcija **AH = 13h - Write string** ispisuje niz znakova koji se nalaze od adrese definisane registarskim parom ES:BP, i to na koordinate DH,DL.

Kontrola tastature

INT 16h - Keyboard services. Ovaj interapt daje funkcije koje omogućuju komunikaciju sa korisnikom preko tastature. Takve su **AH = 00h - Wait for keystroke and read** i **AH = 10h - Wait for keystroke and read ext**, koje zaustavljaju izvršavanje programa dok se ne pritisne neki taster, a zatim će kao izlaz dati u registru AL ASCII kod pritisnutog tastera, a u AH registru će se nalaziti tzv. scan kod. Druga funkcija se odnosi na tastature sa proširenim setom tastera, takva je većina današnjih tastatura. Funkcije **AH = 01h - Get keystroke status** i **AH = 11h - Get keystroke status ext** omogućuju da saznamo da li je i koji taster pritisnut, ali razlika u odnosu na prethodne funkcije je u tome što ove funkcije ne smatraju znak očitanim, i samim tim ga ne uklanjaju iz bafera.

Dalje, funkcije **AH = 02h - Get keyboard status** i **AH = 12h - Get keyboard status ext** omogućuju da provjerimo status kontrolnih tastera na tastaturi, na primjer da li Ctrl taster pritisnut.

11.11.4 DOS prekid 21h

Svaki operativni sistem koristi servise za neki vid komunikacije aplikacionih programa sa njim. U ovom naslovu govori se o DOS–ovom prekidu čiji je broj $(21)_{16}$, takozvanom **INT 21 - DOS Function Dispatcher**-u. Ovaj prekid poziva se pomoću naredbe INT 21. On ima veliki broj raznovrsnih funkcija, a mi ćemo nabrojati samo neke od njih. Kao i za ostale prekide, njegova funkcija se definiše tako što se u registar AH plasira odgovarajući broj, prije izvršavanja naredbe INT 21.

Četiri bajta u memoriji (u RAM–u) čije su adrese redom od 0000:0080 do 0000:0083 čine jedan član tabele vektora prekida. Taj član odnosi se naravno na INT 21. Postavlja se pitanje, koji je sadržaj upisan u ta četiri bajta? Odgovor na ovo pitanje zavisi od verzije operativnog sistema DOS odnosno Windows. Na primjer, moguće da pročitamo sadržaj segment:offset = 00C9:0FA8. Dakle, tekst rutine broj 21H (kod rutine) počinje upravo na toj adresi. Kod rutine zauzima u memoriji iduće bajte, sve do kraja rutine (može se i ova adresa saznati). Rutina zauzima u memoriji ukupno nekoliko kilo–bajta.

Uništenje procesa

AH = 4Ch - Terminate process

Ovo je funkcija koju smo već koristili u primjerima. DOS, kao operativni sistem, zahtijeva da svaki program koji radi pod njim bude završen pozivom ove funkcije, osim u slučajevima kada se radi o pisanju tzv. rezidentnih programa. Ovaj poziv je neophodan da bi se kontrola izvršavanja vratila komandnom interpreteru, tj. samom operativnom sistemu. Kao argument poziva ove funkcije u AL registru se navodi tzv. povratni kod, koji standardno ima vrijednost nula, ako je program legalno završen, odnosno sadrži kod greške, ukoliko je do nje došlo prilikom izvršenja programa.

Rad sa prekidima

AH = 25h - Set interrupt vector

Ova funkcija omogućuje redefinisanje prekidnih rutina. Kao parametre poziva u registru AL navodimo broj prekida koji želimo da preuzmemo, a u registarski par DS:DX stavljamo adresu prekidne rutine.

AH = 35h - Get vector. Ova funkcija daje adresu prekidne rutine. Njen jedini parametar poziva je registar AL, u kome se nalazi broj prekida čiju adresu želimo, dok se sama adresa nalazi u registarskom paru ES:BX.

Rad sa konzolom

BIOS je imao veoma "grubu" komunikaciju sa korisnikom, pa su se tvorci DOS-a pobrinuli za to da se funkcije za komunikaciju pojednostave za korišćenje, i da im se daju nove mogućnosti. Kod DOS-a, kao i kod mnogih drugih operativnih sistema, ekran i tastatura se posmatraju kao datoteke, ovdje one nose naziv STDIN za tastaturu, i STDOUT za ekran. Naravno iz komandne linije se saobraćaj može preusmjeriti, pa se za ulaz može koristiti prava datoteka, a i za izlaz takođe.

Osim ova dva postoje još i uređaji STDAUX, koji predstavlja serijsku liniju, i STDPRN, koji pokazuje na štampač. Naravno tu je i STDERR, koji je vezan za ekran, i služi za prikazivanje grešaka prilikom rada.

AH = 01h - Console input with echo učitava karakter sa standardnog ulaza (tastature) uz ispis na ekran (odjek, eho). Učitani karakter ide u akumulator (AL). Razmotrimo dio programa na jeziku asemblera.

```
MOV AH, 1   (AH ← 1, zadaje se funkcija prekida)
INT 21      (poziva se rutina)
```

Prilikom izvršavanja ovih naredbi ako otkucamo recimo X, onda će u AL biti upisana odgovarajuća ASCII vrijednost, konkretno to je vrijednost $(58)_{16}$ odnosno 88 dekadno, i još će slovo X biti dato na ekranu.

AH = 02h - Console output upisuje jedan znak u STDOUT, tj. prikazuje ga na ekranu. Znak (karakter) koji želimo da bude odštampan (da bude prikazan na ekranu) treba prethodno plasirati u registar DL. Navedimo mali primjer. Razmotrimo dio programa na jeziku asemblera koji se sastoji od tri naredbe kako slijedi.

```
MOV DL, 41  (DL ← 41; plasira se znak; u ASCII kodu je 'A' =(41)16 )
MOV AH, 2   (AH ← 2; plasira se broj funkcije )
INT 21      (poziva se rutina prekida; )
```

U rezultatu izvršavanja ove tri naredbe, imaćemo veliko slovo A prikazano na ekranu.

Funkcije **AH = 03h - Wait for auxiliary device input** i **AH = 04h- Auxiliary output** omogućuju komunikaciju sa serijskim portom, slično kao sa konzolom.

Funkcija **AH = 05h - Printer output** služi za rad sa štampačem. Znak koji želimo ispisati smještamo u DL registar.

Funkcije direktnog pristupa konzoli, što podrazumijeva nemogućnost prekida, objedinjene su u funkciji **AH = 06h - Direct console I/O**. Ukoliko prilikom poziva u DL registar smjestimo vrijednost FFh, vršiće se očitavanje ulaza, a ukoliko je bit nula status registra na nuli - u registru AL će nam se nalaziti očitani znak. U slučaju da se prilikom poziva u DL registru nalazi neka druga vrijednost, taj znak će biti proslijeđen na standardni izlaz.

Funkcija **AH = 07h - Direct input without echo** obavlja direktni pristup konzoli, ali bez odjeka. Slično, funkcija **AH = 08h - Console input without echo** obavlja ulaz jednog znaka bez odjeka. Sve je isto kao kod funkcije AH = 1 samo što sada nema odjeka tj. sada nema prikazivanja na ekranu. Kao primjer možemo napisati dvije naredbe MOV AH, 8 i INT 21.

Funkcija **AH = 09h - Print string** štampa string, tj. nisku znakova. Pomoću ove funkcije na ekranu će biti prikazan jedan niz slova. Članovi niza zauzimaju u memoriji niz uzastopnih bajta. Položaj prvog bajta definišu

tekuće vrijednosti dva registra DS i DX, adresa prvog bajta je segment:offset = DS:DX. Štampa se taj bajt i redom idući bajti sve dok se na tom putu ne naiđe na znak $ kome odgovara ASCII broj $(24)_{16}$. Taj znak $ neće biti prikazan. Kao primjer, možemo razmotriti šta će se desiti u rezultatu izvršavanja sljedećih naredbi:

```
MOV DS, 1C    (DS ← (001C)16)
MOV DX, EEE   (DX ← (0EEE)16)
MOV AH, 9     (AH ← 9; zadaje se funkcija prekida)
INT 21        (poziva se rutina za obradu prekida)
```

Prije puštanja programa u rad, treba pogledati šta piše na adresi 001C:0EEE i na narednim adresama i pronaći gdje tu piše $ prvi put. U svakom slučaju, na ekranu će biti odštampano nekoliko karaktera.

Osim štampanja niz znakova, potrebna nam je i funkcija koja bi omogućila unošenje više od jednog znaka. To je funkcija **AH = 0Ah - Buffered keyboard input**. Ona zahtijeva da se navede adresa bafera u memoriji gdje će unešeni podaci biti smješteni. Prvi bajt u baferu prije poziva mora sadržati maksimalan broj znakova koji se u bafer mogu smjestiti, dok će drugi bajt po povratku sadržati broj unijetih znakova do pritiskanja tastera ENTER. Preostali bajti bafera će sadržavati unijete znakove.

Ukoliko želimo da provjerimo da li je došlo do pritiskanja nekog tastera na tastaturi, to možemo učiniti pozivom funkcije **AH = 0Bh - Check standard input status**. Ona će nam kao odgovor dati u registru AL nula ukoliko nijedan taster nije pritisnut, a 0FFh ukoliko jeste.

Ukoliko prije unosa želimo isprazniti bafer tastature od nepročitanih podataka, to ćemo učiniti pozivom funkcije **AH = 0Ch - Clear keyboard buffer**, tako što ćemo joj kao parametar proslijediti, recimo, 0xFFH u AL registru. Ukoliko u njega smjestimo broj neke od funkcija unosa, ona će biti pozvana po pražnjenju bafera.

Rad sa fajlovima

Nisu sve funkcije prekida INT 21 tako proste kao ove dosad nabrojane. Vidjećemo da postoje i puno moćnije funkcije. Takve su, recimo, funkcije koje rade sa fajlovima koji su upisani na hard–disku. Opisaćemo samo nekoliko od mnogo funkcija u okviru INT 21 za rad sa fajlovima.

Tako funkcija **AH = 0F** služi za otvaranje fajla. U DS i DX treba pripremiti adresu DS:DX tzv. bloka za upravljanje fajlom FCB, File Control Block. I treba još naravno pripremiti sami FCB. FCB sadrži osnovne podatke o fajlu (naziv fajla i slično). Ova funkcija vraća tzv. status fajla u AL.

Funkcija **AH = 10** služi za zatvaranje fajla, funkcija **AH = 13** za brisanje; funkcija **AH = 16** za kreiranje fajla. Veličine DS, DX i AL imaju isti smisao kao maločas.

Funkcija **AH = 14 Sequential Read** čita jedan blok iz fajla u DTA, Disk Transfer Area. Slično, funkcija **AH = 15 Sequential Write** piše blok iz DTA.

Naravno, FCB ima tačno određenu strukturu: koliko bajta zauzima, koje podatke sadrži i gdje se (u bloku) koji podatak nalazi. Rutine se tokom svog rada obraćaju samom hard–disku. One čitaju sa hard–diska podatke o samom hard–disku: koliko traka ima, koliko sektora ima, kolika je veličina jednog sektora, i drugo. Sami hard–disk sadrži i svoju tzv. tabelu alokacije fajlova, FAT, engl. File Allocation Table.

Prekid broj 21 ima desetine i desetine svojih funkcija. One se odnose na razne stvari kao što su upravljanje informacijama, upravljanje direktorijumima, upravljanje procesima, upravljanje memorijom, itd.

11.11.5 Aplikacioni servisi

Preostale servise preuzimaju pojedini programi, oni nadopunjuju već postojeće servise ili formiraju nove. Većina tih servisa je nestandardna, a nije rijedak slučaj kada se poklapaju. Jedan od češće korišćenih dodatnih servisa predstavlja servis za opsluživanje miša.

INT 33 - Mouse service

Važnije funkcije ovog servisa su: **AX = 00h - Mouse reset** koja inicijalizuje drajvera miša; **AX = 01h - Show mouse cursor**, koja prikazuje pokazivač miša na ekranu; **AH = 02h - Hide mouse cursor** koja ga sklanja; **AX = 03h - Get position and button status** očitava trenutnu poziciju miša i status tastera; **AX = 04h - Set mouse cursor position** postavlja poziciju miša; **AX = 05h - Get mouse button press** očitava poziciju na kojoj je neki od tastera pritisnut; **AX = 06h - Get mouse button release** očitava poziciju na kojoj je neki od tastera otpušten; **AX = 07h - Define horizontal cursor range** ograničava horizontalno kretanje pokazivača miša; **AX = 08h - Define vertical cursor range** ograničava vertikalno kretanje pokazivača miša.

Sljedeći primjer demonstrira upotrebu nekih od dosad navedenih funkcija. Iz primjera se izlazi pritiskom na oba tastera miša istovremeno.

```
        .model small
        .stack
        .data
        .code
start :
        xor ax,ax
        int 33h
        or ax,ax
        jz kraj
        mov ax,1
        int 33h
        mov ax,7
        mov cx,10*8
        mov dx,70*8
        int 33h
        mov ax,8
        mov cx,5*8
        mov dx,20*8
        int 33h
ponovo:
        mov ax,3
        int 33h
        cmp bx,3
        jne ponovo
kraj:
        mov ax,2
        int 33h
        mov ax,4c00h
        int 21h
        end start
```

11.12 Pisanje obrađivača prekida

Kada razvijamo neku novu periferiju, ili želimo da bolje iskoristimo već postojeće uređaje, moramo se odlučiti da napišemo i softversku podršku na najnižem nivou. Obrađivači prekida reaguju na događaje koje prouzrokuje hardver. Tu spadaju prekidi iz grupe prekida na nivou računara, koje smo već objasnili na početku.

Postoji nekoliko stvari na koje treba obratiti pažnju prilikom pisanja obrađivača. Prije svega treba imati na umu da se prekid može desiti u bilo kom trenutku, tako da je na početku potrebno sačuvati na steku sadržaj svih registara, koji će biti izmijenjeni u tijelu obrađivača. Naravno, te vrijednosti treba na kraju ponovo vratiti ali u obrnutom redosljedu, zbog načina rada sa stekom.

Kada se prekid desi, procesor će sačuvati sadržaj status registra, kao i adresu povratka iz prekida na steku. Kada počne izvršavanje, automatski će se zabraniti dešavanje prekida, a ukoliko želimo da ih dozvolimo potrebno je da to eksplicitno učinimo, mada je to rijetko kad potrebno i može da dovede do pojave problema. To što je procesor sačuvao sadržaj status registra nam omogućava da možemo slobodnije da se koristimo njegovim sadržajem.

Tijelo obrađivača je potrebno da bude kratko i da sadrži samo operacije čitanja, odnosno pisanja, na port i smještanje podataka u bafere u memoriji, ili pak njihovo isčitavanje. Ovo je naročito bitno ako je procesor spor, a prekidi se često dešavaju.

Na primjer, možemo pogledati kako glasi kod rutine koja se poziva naredbom INT 5 a služi za štampanje ekrana.

Kod rutine počinje na adresi F000:FF54 (to smo saznali čitajući šta piše na adresama od 0000:0010 do 0000:0013). On se završava na adresi F000:FFD3. Tako da rutina u memoriji zauzima ukupno 130 bajta dekadno. Kada se izbroji, rutina se sastoji od ukupno 66 naredbi.

Zavisno od verzije DOS–a, tekst rutine nije uvijek isti. Kako su se iz godine u godinu hardver i softver razvijali tako se i rutina pomalo mijenjala i dopunjavala, od jedne verzije DOS–a do druge. Može se mijenjati i njena početna adresa (odgovara promjeni petog člana tabele vektora prekida). Mehanizam tabele daje korisniku kompatibilnost stare i nove verzije OS–a. Aplikativni program koji poziva INT 5 i ne primijeti promjene.

Iz koda može da se vidi da rutina na početku stavlja na stek i time čuva neke registre

(PUSH DX, PUSH CX, PUSH BX, PUSH AX, PUSH DS)

a na kraju svog rada ona ih vrati, odnosno uzme iz steka

(POP DS, POP AX, POP BX, POP CX, POP DX, vidimo da je sada redosljed obrnut).

To su registri koje rutina tokom svog rada koristi, odnosno mijenja njihov sadržaj. Rutina je obavezna da ostavi po starom procesorove registre kada obavi IRET (kada povratak u program, povratak iz rutine).

Pogledajmo u glavnim crtama šta se dešava tokom izvršavanja rutine. Rutina prvo pita kakav je oblik ekrana. Dobija obavještenje da ekran ima oblik 25 vrsta puta 80 kolona. Zatim redom u dvostrukoj petlji (za vrste i kolone) uzima jedan po jedan karakter iz video bafera i odmah šalje štampaču taj karakter.

Iz koda rutine se vidi da ona sa svoje strane poziva neke druge rutine. Konkretno, na više mjesta pojavljuju se naredbe INT 10 i INT 17.

11.13 Pisanje servisa

Za razliku od pisanja obređivača prekida, kod pisanja servisa ne mora se pretjerano voditi računa o brzini izvršavanja tijela prekida, jer se prekidi pozivaju na zahtev korisnika. Ovdje treba prije svega obratiti pažnju da se ne preuzme neki od već postojećih servisa, a ako se to učini namjerno, kada se odradi naš dio koda, potrebno je pozvati i prošli servis.

Servis obično podrazumijeva postojanje više funkcija, tako da osnova tijela servisa čini samo testiranje operanada i pozivanje procedura koje odrađuju ostatak posla.

Ovdje nije neophodno voditi računa ni o čuvanju sadržaja registara, jer se pozivanje vrši programski, samo je neophodno u dokumentaciji navesti koji su to registri koje ovaj poziv mijenja.

11.14 Sistem prekida procesora Intel 8086

U ovom naslovu biće ponovljene i sređene činjenice o sistemu prekida procesora Intel 8086: tabela vektora prekida, izvori zahtjeva za prekidom, obrada prekida i redosljed prioriteta među zahtjevima.

Prva 1024 bajta u memoriji rezervisani su za tzv. tabelu vektora prekida, adrese od 0000:0000 do 0000:03FF. Pojedini član tabele zauzima četiri bajta, tako da ukupno ima mogućnosti za 256 vrsta prekida. Za pojedini član tabele, prva dva bajta definišu vrijednost za IP a druga dva bajta definišu vrijednost za CS. Tako da jedan član tabele definiše jednu adresu CS:IP u memoriji. To je adresa u memoriji na kojoj počinje kod rutine za obradu prekida. Kada se pristupi obradi prekida čiji je broj n (gdje je $0 \leq n \leq 255$) onda će se izvršiti skok na odgovarajuću adresu.

Postoje tri moguća izvora–uzroka da dođe do prekida. Saglasno tome, sve vrste prekida ukupno dijele se u tri kategorije: a) *hardverski prekidi,* b) *softverski prekidi* i c) *izuzeci.*

a) Ako je prekid izazvan spoljašnjim događajem, gledano iz ugla procesora, onda je to hardverski prekid. Drugim riječima, takav je prekid izazvan signalom poslatim procesoru od strane nekog drugog dijela računarskog sistema. Zahtjev za hardverski prekid predstavlja asinhroni događaj gledano iz ugla procesora, tj. iz ugla programa odnosno niza naredbi čije izvršavanje je u toku. Dakle, zahtjev će se pojaviti tokom izvršavanja neke naredbe, ali se ne može unaprijed determinisati koja je to naredba.

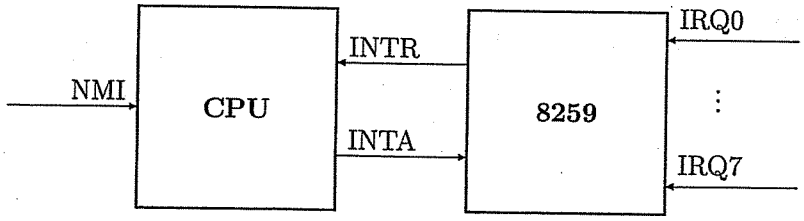
Zahtjevi pristižu preko pina NMI non–maskable interrupt ili preko pina INTR interrupt request. U slučaju NMI, zahtjev ne može da bude maskiran. To znači da će taj zahtjev svakako biti odobren. Zahtjevi koji dolaze preko pina INTR mogu da budu maskirani: zahtjev će biti usvojen ili će biti odbačen. Za pojedinu instancu zahtjeva oblika INTR, o tome odlučuje tekuća vrijednost interrupt flaga. IF = 1 usvaja se, IF = 0 ne usvaja se.

Ulogu posrednika između procesora i drugih djelova računarskog sistema, za potrebe maskirajućih hardverskih prekida, ima jedna posebna hardverska komponenta (integrisano kolo). To je tzv. kontroler prekida. Obično je to komponenta Intel 8259 (ili 8259A). Ima 28 pinova. Naime, može se desiti da nekoliko perifernih uređaja (hard–disk, tastatura, i slično) u istom trenutku istaknu svoje zahtjeve. Svi ti zahtjevi idu na kontroler prekida. On ima

upisano pravilo o prioritetu zahtjeva (o hijerarhiji), na osnovu čega će izabrati jedan zahtjev i uputiti ga procesoru.

Pored ostalih pinova, kontroler ima osam pinova označenih kao IRQ0, IRQ1, ..., IRQ7, interrupt request. Jednoj periferiji odgovara jedan od tih pinova. U slučaju više istovremenih zahtjeva IRQk, propustiće se onaj od njih koji ima najmanje k. Vidimo da se pravilo o prioritetu definiše rasporedom na nožicama. Treba vezati periferije na odgovarajuće nožice onako kako to odgovara datom računarskom sistemu. Jasno je da bi se promjenom rasporeda promijenili prioriteti, tj. imali bismo drukčiju konfiguraciju sistema.

Pored drugih pinova koji povezuju kontroler sa procesorom, postoje i dva pina koji su označeni (sa strane procesora) kao INTR i INTA. INTR je ulazna nožica. INTA je izlazna nožica, interrupt acknowledgement. Izložimo ukratko mehanizam za saopštavanje broja prekida. Procesor je dobio signal po liniji INTR. Ako je IF = 1 onda procesor šalje signal po liniji INTA. Kao odgovor, kontroler šalje procesoru osam binarnih vrijednosti (zavisno od vrste perifernog uređaja), preko osam linija adresne magistrale. Znamo da adresna magistrala ima ukupno dvadeset linija. Tih osam binarnih vrijednosti obrazuju jedan broj n u granicama $0 \leq n \leq 255$. Naravno, zatim će procesor preći na odgovarajuću rutinu za obradu prekida, ako nema prečih zahtjeva. Pravilo o korespondenciji (vrsta uređaja $\leftrightarrow$ broj n) nalazi se upisano u kontroleru. Pravilo se upisuje prije njegovog povezivanja sa drugim komponentama sistema. Isto tako, ono može da bude izmijenjeno po potrebi. Zato se za kontroler kaže da je on programabilan (slika 11.3).



Slika 11.3: Hardver za prekide

Osim glavnog kontrolera, u sistemu je obično prisutan i drugi, pomoćni kontroler. Pomoćni kontroler se vezuje na nožicu IRQ2 glavnog kontrolera,

po pravilu. U sistemu sa ovakvom konfiguracijom, očito je da 15 perifernih uređaja može da ispostavlja maskirajuće zahtjeve.

b) Ako je prekid nastupio kao rezultat izvršavanja naredbe INT onda je to softverski prekid. Ovoj kategoriji pripada i prekid prekoračenja INTO. Znamo da njegov uslov glasi OF = 1, overflow flag je setovan. $n = 4$.

c) Može se desiti da neke unutrašnje okolnosti u procesoru prouzrokuju prekid. To su tzv. izuzeci. Ovdje pripadaju $n = 0$ divide error (ili divide by zero) i $n = 1$ single step. Zahtjev $n = 0$ nastupiće u rezultatu izvršavanja naredbe za dijeljenje, ako je imenilac jednak nuli, odnosno ako je količnik prevelik. Zahtjev $n = 1$ nastupiće poslije izvršavanja bilo koje naredbe ako je u datom trenutku TF = 1, trap flag je setovan. Služi za izvršavanje programa korak po korak (naredba po naredba). Još se kaže i trace (trag). Ova mogućnost je uvedena kao podrška za dibagiranje.

Kao što smo ranije vidjeli, kada je izvršavanje tekuće naredbe pri kraju, procesor pregleda zahtjeve koji su ispostavljeni. On donosi odluku o tome da li će neki biti usvojen. Ako će biti, onda on i odabira broj n, odnosno odabira jedan zahtjev u slučaju da ih je bilo više. Uzmimo da je neki zahtjev usvojen i da je broj n odabran. Tada dalja dešavanja ne zavise od kategorije prekida ili od čega drugog. Vrše se radnje: pushf, if $\leftarrow 0$, tf $\leftarrow 0$ i call far $[4{\cdot}n]$, skok na rutinu. Kasnije će se izvršiti povratak, pomoću naredbe IRET. Odgovarajuće radnje su: pop ip, pop cs i popf.

Rang raznih klasa prekida hardverski je ugrađen u sam procesor, i ne može da bude promijenjen. U slučaju više istovremenih zahtjeva, biće odobren najjači. Konkretno, prioritet od jačih prema slabijima glasi: i) divide error, INT, INTO, ii) NMI, iii) INTR, iv) single step.

Na kraju, znamo da podrški za dibagiranje pripadaju i tzv. breakpoints, o čemu je bilo riječi ranije. Zanimljivo je pogledati kako se ostvaruju "prekidne tačke". Neka je P program sa greškama. Očekuje se da debager umjesto svake breakpoint naredbe (to je naredba iz programa P, očito) stavi naredbu INT 3. Naredba INT 3 zauzima samo jedan bajt (ona glasi CC Hex). Tako da bilo koja naredba može da bude zamijenjena, a da se ne traži više prostora. Postavlja se pitanje, kako će sad P da radi, kad su neki njegovi djelovi promijenjeni, odnosno kad su se izgubile neke naredbe? Očekuje se da dibager preuzme na sebe izgubljene naredbe: on pamti koje su to naredbe i vodi računa da one budu izvedene, tj. da se sve izvodi kako stvarni P predviđa. Dakle, dibager interpretira.

11.15 O mikroprocesoru Intel 80286

Procesor Intel 80286 pojavio se 1982. godine. Sadrži 134.000 tranzistora i ima 68 pinova. Procesor obezbjeđuje podršku za tzv. *multitasking* (rad na više poslova), tj. mogućnost da se nekoliko programa izvršava prepleteno, brzo se smjenjujući na procesoru. U okviru toga predviđen je i sistem zaštite, da se ne bi desilo da jedan program izmijeni podatke ili naredbe drugog programa. U tom smislu, procesor ima dva moda (režima, načina) rada – *realni način* (real mode) i *zaštićeni način* (protected mode) (ponekad se kaže i režim supervizora ili režim operativnog sistema). U realnom načinu on se ponaša potpuno isto kao procesor 8086. U zaštićenom načinu stvari se potpuno mijenjaju. U tom načinu rada procesora nikad se ne dopušta da korisnik nešto upisuje u segment koda, tj. u segment koji je namijenjen da sadrži naredbe, što u realnom načinu može da bude urađeno.

Hardver samo pruža preduslove za multiprogramski režim rada, odnosno izlazi u susret očekivanim potrebama softvera. Zadatak je softvera, odnosno operativnog sistema, da te preduslove iskoristi ili ne iskoristi, i da odluči kako će ih iskoristiti.

Proširen je skup mašinskih naredbi u odnosu na 8086 a proširen je i skup registara. Takođe je i mogući memorijski prostor povećan: postoji mogućnost da se adresuje 16 MB RAM–a; adresi jednog bajta u memoriji odgovaraju 24 bita. Mikroprocesor Intel 80286 je 16–bitni procesor, kao i njegov prethodnik Intel 8086, dok je već njegov nasljednik Intel 80386 32–bitni procesor.

Već je rečeno da nekoliko programa (*procesa*) može da se izvršava prepleteno, uz brzo smjenjivanje na procesoru. Pojedini proces ima ograničenje od 1 MB (jedan mega–bajt) za veličinu svog zauzeća memorije.

Kada se računar uključi ili kada se procesoru pošalje signal RESET, onda se procesor nalazi u realnom načinu rada. Ako je u nekom trenutku procesor prešao u zaštićeni način rada onda se više ne može vratiti u realni način rada na jednostavan način, već jedino resetovanjem. Kod procesora nasljednika ovo pitanje riješeno je na povoljniji način za korisnika.

Predviđeno je da se u memoriji drže tokom rada računara tri važne vrste tabela, vezano za zaštićeni način rada. Hardver ima sredstva da sa tim tabelama manipuliše. To su GDT – General Description Table, LDT odnosno LDTs – Local Description Tables (množina) i IDT – Interrupt Description Table. GDT služi kada se sa izvođenja jednog procesa pređe na izvođenje

nekog drugog procesa. Tabela LDT ima onoliko koliko u datom trenutku ima procesa čije je izvođenje u toku– svaki proces ima svoju lokalnu tabelu deskriptora. LDT govori gdje se zaista u memoriji nalaze naredbe i podaci tog procesa, i sl.

Da se ostvari zaštita, svakom procesu pridružen je njegov tzv. *nivo privilegije*. Postoje četiri moguća nivoa privilegije, označavaju se brojevima od 0 do 3, gdje je broj 0 najviši nivo. Proces sa manjim brojem ima veća prava. Koriste se sljedeći nazivi:

nivo 0 – jezgro operativnog sistema,
nivo 1 – servisi operativnog sistema,
nivo 2 – proširenja operativnog sistema i
nivo 3 – aplikativni programi.

11.16 Tridesetdvobitni mikroprocesori

U ovom naslovu govori se o 32–bitnim Intelovim procesorima 80386, 80486 i Pentium. Znamo da 386 predstavlja proširenje procesora 286, slično 486 predstavlja proširenje od 386 i isto tako Pentium je proširenje od 486.

Intel 386 se pojavio 1985. godine. Razmotrimo koje registre on ima, kako se vrši adresiranje memorije i koje režime rada podržava.

Ranija četiri registra AX, BX, CX i DX (akumulatori), koji su bili 16–bitni kod 286, zamijenjeni su 32–bitnim registrima EAX, EBX, ECX i EDX. Ovdje je E skraćenica od extended, što znači proširen. U programu na jeziku asemblera mogu da se pojavljuju naredbe kao MOV EAX, 003F ili kao ADD EAX, EBX. Može se pisati i samo AX ili BX ili CX ili DX, odnosno može se pristupiti posebno donjoj polovini bilo kog od četiri akumulatora. Slično važi i za ostale registre čija skraćenica počinje sa E. Dozvoljeno je pisati AH, AL,..., DL, kao u slučaju 8086 asemblera.

Znamo da četiri akumulatora i četiri ofsetna registra čine zajedno osam registara opšte namjene. Sada su četiri ofsetna registra ESP, EBP, ESI i EDI, i svaki od njih je 32–bitni. Program može da pristupi donjoj polovini SP, BP, SI odnosno DI.

386 ima 32–bitni registar flagova EFLAG. Njegova donja polovina (16 bita) FLAG poklapa se po svojoj strukturi sa registrom flagova prethodnika

(koji je označavan kao PSW). Dakle, EFLAG čine biti 31–0, a FLAG njegov dio 15–0. Nisu sva 32 flipflopa u registru EFLAG iskorišćena.

Slično, postoji EIP – extended instruction pointer – 32 bita. Program može da pristupi donjoj polovini čija je oznaka IP.

Četiri segmentna registra CS, DS, SS i ES ostali su 16–bitni. Pojavila su se još dva nova segmentna registra FS i GS, koji se takođe 16–bitni.

Adresa naredbe obrazuje se kao CS:EIP ili kao CS:IP. Adresa podatka izražava se recimo kao address = segment:offset = DS:EDX ili kao DS:DX. I dalje važi formula address = 16 × segment + offset. Budući da su sada ofsetni registri ESP, EBP, ESI i EDI, kao i registar EIP 32–bitni, to je već drugi sabirak u formuli (sabirak "offset") dovoljan da se adresuje svih 2^{32} bajta RAM–a. To je 4 GB, četiri giga–bajta. Dakle, 386 je predviđen da radi sa RAM–om čiji je kapacitet jednak 4 GB (ili manje).

Kod ovog procesora postoje realni i zaštićeni način rada, isto kao kod 286. U realnom načinu, 386 radi kao jedan puno brzi 8086, proširen na 32 bita ako se želi. Znamo da u zaštićenom načinu može da se izvršava nekoliko procesa istovremeno. Neki od tih procesa mogu da rade u tzv. *virtualnom 86 modu*, engl. virtual 8086 mode (virtual 86 mode ili V86 mode). Dakle, omogućeno je da se koristi softver koji je nekad ranije bio sastavljen za 8086, a da pritom računar ne napušta zaštićeni način rada. Dakle, virtualni 86 način predstavlja jedan slučaj zaštićenog načina.

Intel 486 pojavio se 1989. godine. Sa stanovišta softvera, sve je praktično ostalo isto kao kod 386. Sa stanovišta hardvera, procesor 486 ima jedinicu za rad sa brojevima u pokretnom zarezu, tako da je u računaru postao suvišan poseban čip, tzv. koprocesor, koji se ranije koristio za izvođenje operacija sa brojevima prikazanim u pokretnom zarezu.

Oktobra 1992. godine Intel je objavio da će se peta generacija njegove linije kompatibilnih procesora (čiji je interni naziv bio P5) zvati Pentium umjesto 586 ili 80586, kao što su svi očekivali. To je bila Intelova strategija da se omogući registrovanje marke i da se time zaštiti naziv spram konkurenata (najviše AMD).

Mikroprocesor se pojavio 1993. godine. Učestanost takta iznosi 60 MHz a može da izvrši 100 miliona naredbi u sekundi. Ima ugrađenu keš memoriju od 16 KB: 8 KB za naredbe i 8 KB za podatke. Ima poboljšanu jedinicu za

rad sa brojevima u pokretnom zarezu. Najvažnije, omogućava da se dvije naredbe izvršavaju istovremeno.

U martu 1994. godine Intel je plasirao na tržište drugu generaciju procesora Pentium.

Jedan matematičar je objavio preko Interneta u oktobru 1994. godine da Pentium ima bug kod dijeljenja brojeva prikazanih u pokretnom zarezu (naredba FDIV) za neke kombinacije brojeva. Naime, za određene kombinacije brojeva, računar saopštava količnik koji je pogrešan, i to relativna greška iznosi $5 \cdot 10^{-4}$. Ova greška je kasnije otklonjena.

U januaru 1997. godine pojavila se treća generacija Pentiuma. Dodato je 57 novih vrsta naredbi. Udvostručen je kapacitet keš memorije. Ima ugrađenu tehnologiju MMX (multimedia extensions), kako je Intel to nazvao.

U konstrukciji procesora odavno je poznata tzv. ideja tekuće trake ili engl. pipeline. To znači da se simultano radi na nekoliko naredbi koje se nalaze u različitim fazama izvršavanja (po mašinskim ciklusima), da bi se dobilo na brzini rada računara. Pentium ima usavršene tekuće linije; teži se da opšta performansa računara bude: jedna izvršena naredba za jedan takt; ima dvije aritmetičko–logičke jedinice za koje se koriste oznake U i V redom. Lako se zapaža da uzastopne naredbe nisu uvijek nezavisne. Recimo, prva glasi INC AX a druga glasi INC AX takođe. Ili kada se i jedna i druga naredba obraćaju istoj lokaciji u memoriji. Slična je situacija prilikom izvršavanja naredbe za uslovni skok. U takvim situacijama, procesor uočava zavisnost i odustaje od istovremenog izvršavanja.

Slična je i tzv. ideja dohvatanja unaprijed. Kada se naredba n dohvati (sa lokacije ℓ) onda počinje njeno izvršavanje. Istovremeno sa tim izvršavanjem donosi se u procesor sadržaj lokacija $\ell + 1$, $\ell + 2$, ... (nekoliko bajta). To se smješta u tzv. instruction queue. Za ovo je zadužena posebna jedinica, prefetch unit (PU). Dobiće se na vremenu ako naredba n nije JMP, JA, CALL ili LOOP ili slično. Kod Pentiuma postoji i jedinica za predviđanje odredišta (BPU) zajedno sa svojim baferom. Ta jedinica odlučuje o tome šta se unaprijed donosi, sa ciljem da se od donošenja unaprijed ima koristi u slučaju da je n naredba za skok. Intel tvrdi da pomenuta jedinica sama za sebe povećava efikasnost izvršavanja programa za nekih 25%.

U tabeli su prikazani brojni podaci o nekim mikroprocesorima iz Intelove linije.

Procesor	God.	Tranzistori	Reg. (bit)	Data bus (bit)	Adr.bus (bit)	Clock (MHz)	Pin
8086	1978	29.000	16	16	20	5	40
80286	1982	134.000	16	16	24	20	68
Pentium	1993	3.100.000	32	64	32	60	273
Pentium II	1997	7.500.000	32	64	36	300	370
Pentium III	1999	28.000.000	32	64	36	1130	370
Pentium IV	2001	42.000.000	32	64	36	2000	423

11.17 Zadaci za vježbu

1. Neka su ulazni podaci programa cijeli brojevi a, b i c. Sastaviti program na jeziku asemblera (za procesor Intel 80x86) u kome se računaju i štampaju cijeli brojevi

$$y_1 = -16a - 20b - 24c, \quad y_2 = a^2 + ab + ac \text{ i } y_3 = \frac{y_1^2 + 1}{y_2^2 + 1}.$$

Program propustiti pomoću debug.exe. Predvidjeti konkretne vrijednosti za ulazne podatke a, b i c. Zapisati rezultate y_1, y_2 i y_3 koji su na kraju saopšteni.

2. Sastaviti program na jeziku asemblera za uređivanje tri broja po veličini.

3. Sastaviti program na jeziku asemblera čiji su ulazni podaci n i k $(1 \leq k \leq n)$ u kome se računaju i štampaju vrijednosti

$$P_n = n!, \quad C_{nk} = \binom{n}{k} \text{ i } V_{nk} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!}.$$

4. Program koji utvrđuje da li je dati prirodan broj prost ili složen.

5. Ulazni podatak je $n \geq 1$, a rezultat treba da bude

$$x_n = 1 - \frac{1}{2} + \frac{1}{3} - \ldots + \frac{(-1)^{n+1}}{n}; \quad \lim_{n \to \infty} x_n = \ln 2.$$

6. Znamo da je niz Fibonačijevih brojeva $\{f_n\}_{n=1}^{\infty}$ definisan sa:

$$f_1 = 1, \quad f_2 = 1 \text{ i } f_{n+2} = f_n + f_{n+1} \quad (n \geq 1).$$

Ulazni podatak je n, a rezultat treba da bude f_n.

7. Ulazni podatak je $n \geq 1$, a rezultat treba da bude n–ti Fermaov broj $F_n = 2^{2^n} + 1$.

8. Ulazni podaci su dva cijela broja a i b. To su koordinate tačke $M = M(a, b)$ u ravni. Razmotrimo kvadrat čija su tjemena tačke $(x, y) = (0, 0)$, $(x, y) = (10, 0)$, $(x, y) = (10, 10)$ i $(x, y) = (0, 10)$. Koliko je najkraće rastojanje tačke M od ruba datog kvadrata?

9. Koliko puta se znak "blanko" pojavljuje u niski? Za niz uzastopnih adresa u memoriji (bajt za bajtom) kaže se da predstavlja nisku ako se znak $ ne pojavljuje u nizu. Znak $ služi da označi kraj niza (zapisan je na kraju, u prvom bajtu koji ne pripada nizu). Referenca niske je adresa njenog prvog bajta.

10. Dat je prirodan broj n. Napisati program za generisanje n slučajnih brojeva.

11. Sastaviti program za računanje najvećeg zajedničkog djelioca (NZD) dva data prirodna broja.

12. Za uređenu trojku prirodnih brojeva (x, y, z) kaže se da je Pitagorina trojka ako važi jednakost $x^2+y^2 = z^2$. Sastaviti program koji će da odštampa sve Pitagorine trojke takve da je $x \leq 100$ i $y \leq 100$.

13. Računanje broja π. Znamo da je

$$\operatorname{arc\,tg} x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots \quad \text{za} \quad -1 \leq x \leq 1 \quad \text{i da je} \quad \operatorname{arc\,tg} \frac{1}{\sqrt{3}} = \frac{\pi}{6}.$$

Sastaviti program u kome se računa parcijalna suma reda kada je $x = \frac{1}{\sqrt{3}}$, sve do sabirka koji se u memoriji prikazuje kao nula (sve do sabirka koji je ispod donje granice za pozitivne realne brojeve).

14. Računanje kvadratnog korijena $\sqrt{a}$ po Heronovom obrascu. Neka je $a > 0$,

$$x_0 = 1 \ \text{ i } \ x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \quad \text{za} \quad n \geq 0.$$

Znamo da je $\lim_{n\to\infty} x_n = \sqrt{a}$. Broj a se učitava. Sastaviti program u kome se redom računaju članovi niza $x_1, x_2.x_3, \ldots$, sve dok ne postane $x_n = x_{n+1}$. Bolje rečeno, sve dok ne dođe do poklapanja prikaza u memoriji dva uzastopna člana niza.

15. Sastaviti program za rješavanje kvadratne jednačine $ax^2 + bx + c = 0$ $(a \neq 0)$.

16. Rad sa nizom. Ulazni podaci programa su $n \geq 1$ i niz brojeva $x_1, x_2, \ldots, x_n$. Sastaviti program za računanje srednje vrijednosti niza $\overline{x}$ i njegove standardne devijacije σ;

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2}.$$

17. Rad sa nizom. Ulazni podaci programa su $n \geq 1$ i niz brojeva $x_1, x_2, \ldots, x_n$. Sastaviti program za uređivanje niza.

18. Sastaviti potprogram za tzv. formatirani izlaz cijelog broja (dva bajta). Na primjer, ako u memoriji stoji 1111 1111 1110 1111 (potpuni komplement), onda treba da bude odštampano -17. Sastaviti i glavni program, da bi se potprogram testirao.

19. Sastaviti potprogram za učitavanje cijelog broja. Na primjer, ako se učitaju tri karaktera $-$, 1 i 7 (za njima dolazi "enter"), onda treba plasirati odgovarajući niz nula i jedinica u memoriju; slično kao u prethodnom zadatku. Sastaviti i glavni program, da bi se potprogram testirao.

20. Sastaviti potprogram za ispitivanje da li je broj n prost. Veličina n upućuje se potprogramu preko akumulatora AX, a potprogram isporučuje svoj odgovor takođe preko AX. Sastaviti i glavni program u kome se računa koliko ima prostih brojeva u intervalu $n_1 < n < n_2$, gdje su n_1 i n_2 ulazni podaci.

21. Sastaviti potprogram za računanje NZD dva broja. Sastaviti i glavni program u kome se utvrđuje da li su brojevi n_1 i n_2 uzajamno prosti, tj. da li je $\text{NZD}(n_1, n_2) = 1$. Ulazni podaci glavnog programa su n_1 i n_2.

22. Ackermannova funkcija $A(m, n)$ definisana je sa:

$$A(0, n) = n + 1,$$
$$A(m, 0) = A(m - 1, 1),$$
$$A(m, n) = A(m - 1, A(m, n - 1)), \quad m, n > 0.$$

Sastaviti program koji će tabelirati Ackermannovu funkciju za vrijednosti $0 \leq m \leq 3$ i $0 \leq n \leq 4$.

Da li bi računar imao bilo kakvih problema ako bi morao da izračuna $A(10, 10)$? Pokušajte da "pješice" izračunate $A(3, 3)$.

Glava 12

Operativni sistem

Operativni sistem je izuzetno važna komponenta računarskog sistema. Operativni sistem je softver, odnosno program. To je prvi program čiji rad vidimo kada uključimo računar, i posljednji kada isključujemo računar. Operativni sistem nam omogućuje da koristimo računar, odnosno da izvršavamo razne programe. On kontroliše čitav računar. U ovom poglavlju prvo ćemo vidjeti šta je to operativni sistem u glavnim crtama, zatim detaljnije objasniti funkcionisanje operativnog sistema.

Kao prvo, moramo razumjeti da svi računari nemaju operativni sistem. Na primjer, računar koji kontroliše neki kuhinjski uređaj ne mora imati operativni sistem. To je zato što taj računar izvršava neke jednostavne zadatke koji se mogu realizovati pomoću hardvera koji se ne mijenja. U tom slučaju nema potrebe da se u takve računare ugrađuje operativni sistem, što bi samo znatno usložnilo uređaj i povećalo mu cijenu. Prvi računari su radili na isti način - to su bili uređaji predviđeni samo za jedan posao, koji nisu imali nikakvog softvera, pa ni operativnog sistema.

Naravno, vremenom su računari postali sve složeniji i za njihovo funkcionisanje je postalo neophodno da imaju operativni sistem. Svi stoni računari imaju operativni sistem. Najčešće se sreću operativni sistemi iz *Windows* porodice firme *Microsoft*- to su *Windows 95, 98, ME, NT, 2000, CE, XP;* operativni sistemi zasnovani na sistemu *Unix - Linux, BSD Unix* i mnogi drugi; operativni sistemi za Macintosh računare. Postoje i stotine drugih operativnih sistema za posebne namjene, za velike računare, robotiku, proizvodnju, upravljanje raznim sistemima,...

12.1 Funkcije operativnog sistema

Operativni sistem obavlja dvije funkcije koje u osnovi nisu povezane. Zato se često posmatra na razne načine, zavisno od toga ko govori o njemu. Te dvije funkcije su:

- Upravljač resursima. Operativni sistem upravlja svim resursima računara, kakvi su, na primjer, procesor, memorija, diskovi, štampači, mrežni adapteri, itd.
- Virtuelna mašina. Ovo znači da operativni sistem omogućuje programima da rade sa hardverom bez znanja svih detalja hardvera.

Prva funkcija je važna jer se u računaru razni programi takmiče za razne resurse kao što je procesor (CPU), operativna i spoljašnja memorija, ulazne i izlazne jedinice. Zamislimo, na primjer, što bi se desilo u računaru u kojem bi praktično istovremeno tri programa koja se izvršavaju poslala po jedan dokument na štampač. Bez operativnog sistema štampač bi odštampao jedan red prvog teksta, pa drugog, pa trećeg,... Rezultat bi bio haotičan. Operativni sistem se stara da svi programi u određeno vrijeme dobiju potrebne resurse i da pri tome nikoga ne zapostavi. Resursi su često jako ograničeni i operativni sistem mora da se postara da se oni iskoriste na način koji je najbolji za sve.

Druga funkcija rezultat je ogromne složenosti računarskog sistema. I najosnovnije operacije u sistemu su izuzetno složene. Na primjer, operacija upisa nekih podataka na disk zahtijeva znanje raznih specifičnosti diska, recimo - adresa bloka, broj sektora po stazi, način snimanja, razmak među stazama; zatim treba izvesti veliki broj komplikovanih koraka kao što su praćenje da li je motor diska uključen, odrediti kada tačno upisati podatke (jer se disk okreće vrlo brzo), i sl. Štaviše, ovdje smo u cilju ilustracije stvarne složenosti naveli tek neke elemente.

Pisanje programa koji uključuje ovakve stvari je izuzetno teško (upravo tako su se pisali programi na prvim računarima). Ne može se očekivati od svakog programera da zna sve ove detalje. Zato u računarskom sistemu mora postojati program (naravno, to je operativni sistem) koji štiti programere od ove složenosti i obavlja ove poslove za njih. Programer (tj. njegov program) treba samo da se obrati operativnom sistemu sa zahtjevom 'upiši to i to na disk' i sve ostalo je zadatak operativnog sistema. Slično, operativni sistem mora da obezbijedi da isti program radi na raznim računarima istog tipa.

Windows operativni sistemi su odličan primjer fleksibilnosti operativnog sistema: potpuno isti program će raditi na hardveru proizvedenom od strane hiljada različitih proizvođača. *Windows* bez problema prihvata hiljade različitih štampača, monitora, diskova, memorija, ...

12.2 Tipovi operativnog sistema

Iako postoji veliki broj operativnih sistema, oni se mogu podijeliti u četiri tipa, zavisno od vrste računara koje kontrolišu kao i vrste primjena koje podržavaju. Tipovi su:

- Operativni sistemi realnog vremena (**Real-Time Operating System**). Ovi sistemi kontrolišu rad mašina, instrumenata, proizvodnje. Realno vrijeme znači da se od ovih sistema zahtijeva obavljanje operacija u strogo kontrolisanom vremenu jer svako kašnjenje, recimo u pomjeranju nekog dijela kompleksne mašine, može izazvati katastrofalne posljedice.
- Jedan korisnik-jedan zadatak (**Single-User, Single Task**). Kao što i ime kaže, ovi operativni sistemi su dizajnirani tako da jedan korisnik može u jednom trenutku izvršavati samo jedan program. U ovu grupu spadaju operativni sistemi kod modernih džepnih računara, recimo PALM OS.
- Jedan korisnik-više zadataka (**Single-User, Multi-Tasking**). Ovaj tip operativnog sistema se najčešće koristi za stone i laptop računare. Ovom tipu pripadaju i Windows 98 i MacOS - oni omogućavaju da jedan korisnik izvršava više programa u isto vrijeme. Na primjer, dok unosite neki tekst u Word-u, vi možete sa Interneta snimati neki fajl, slušati muziku i štampati nešto.
- Višekorisnički (**Multi-User**). Ovi operativni sistemi omogućuju da mnogo korisnika u isto vrijeme koriste računarske resurse. Operativni sistem mora balansirati između zahtjeva raznih korisnika, tako da programi koje oni koriste imaju dovoljno broj resursa, i to na taj način da rad jednog korisnika ne zasmeta u radu drugima. *Unix*, *VMS* i operativni sistemi za velike računare predstavnici su ove grupe. Ovdje treba istaći značajnu razliku između višekorisničkog operativnog sistema i

jednokorisničkog koji podržava mrežni rad. I *Windows 2000* i *Novell Netware* podržavaju stotine i hiljade korisnika u mreži, ali nijedan od njih nije pravi višekorisnički operativni sistem. U suštini, u ovim slučajevima sav mrežni rad i prijavljivanje sa daljine, koje omogućava mrežni rad, izvršava jedan korisnik, koji se naziva administratorom.

12.3 Istorija operativnih sistema

Kako je operativni sistem jako povezan sa hardverom i arhitekturom računara, razvoj operativnih sistema pratio je razvoj računara po generacijama. Zato ćemo kroz kratak pregled istorije operativnih sistema preći i generacije računara.

12.3.1 Prva generacija (1945-55)

Prvi računari su bili zasnovani na vakumskim cijevima. Bili su ogromni, zauzimali su cijele sobe. Nije bilo programskih jezika, cjelokupno programiranje je bilo na mašinskom jeziku i to u hardveru, pomoću desetina hiljada vakumskih cijevi. Program je bio ploča sa mnogo vakumskih cijevi koja je specijalno napravljena za svaki konkretni zadatak. Naredni zadatak računar je izvršavao tako što se ploča zamijeni drugom.

U ranim 50-im, u upotrebu se uvode bušene kartice, koje su donekle olakšale rad. Postalo je moguće pisati programe na karticama i unositi ih u računar, ali osnovni princip rada ostao je isti, tako da se u ovoj generaciji praktično ne može govoriti o operativnom sistemu.

12.3.2 Druga generacija (1955-65)

Izum tranzistora sredinom 50-ih radikalno je promijenio stanje. Računari su postali znatno pouzdaniji, jeftiniji i, što je najvažnije, efikasniji. Prvi put je uspostavljena razlika između dizajnera, proizvođača, operatera, programera i radnika na održavanju računara. Programi su se pisali na FORTRAN-u ili asembleru i unosili u računar preko bušenih kartica ili magnetnih traka.

Što se tiče načina rada ovih računara, njih je karakterisalo to što su oni bili takozvani *batch* sistemi, što označava da se računaru predaje nekoliko programa koji se zatim izvršavaju jedan za drugim. Operativni sistemi su

bili jednostavni. Tipični operativni sistemi iz ove generacije su FMS (Fortran Monitor System) i IBSYS firme IBM.

12.3.3 Treća generacija (1965-80)

Ovu generaciju karakteriše pojava integrisanih kola. Računari postaju sve dostupniji, manji i bolji.

Za ovu generaciju računara i operativnih sistema važna je pojava *multiprogramiranja,* koje označava da računar može da izvršava više programa praktično istovremeno, kao i njegova varijanta *timesharing,* odnosno *dijeljenje vremena.* Pod dijeljenjem vremena podrazumijeva se da više korisnika radi sa istim računarom, preko svog terminala. Računar realno izvršava samo jedan program u jednom trenutku, ali vrlo često prelazi sa jednog zadatka na drugi, stvarajući kod svakog korisnika utisak da je on jedini korisnik računara.

Karakteristični operativni sistem ove generacije je *MULTICS,* iz koga je nešto kasnije nastao operativni sistem *Unix.*

12.3.4 Četvrta generacija (1980-)

Prvi mikro–računari pojavili su se sedamdesetih godina i bili su zasnovani na 8–bitnim mikroprocesorima, dok su se prvi personalni računari pojavili 1977. godine (firma Apple) odnosno 1981. godine (firma IBM).

Osnova pojave mikro i personalnih računara je pojava integrisanih kola velikog stepena integracije, tj. čipova koji su sadržali veliki broj tranzistora. Glavni operativni sistemi na početku ove generacije su bili *CP/M, MS-DOS* firme *Microsoft* i *Unix.*

Najvažniji operativni sistemi ove generacije su *MS-DOS* i sistemi iz *Windows* porodice firme *Microsoft*– to su *Windows 95, 98, ME, 2000, NT, CE , XP;* zatim operativni sistemi zasnovani na sistemu *Unix - Linux, BSD Unix* i mnogi drugi; operativni sistemi za Macintosh računare.

12.4 Važniji operativni sistemi

12.4.1 CP/M

Operativni sistem CP/M koristio se na mikro–računarima koji su bili zasnovani na Intelovim osmobitnim mikroprocesorima, kakav je recimo Intel 8080. CP/M je skraćenica od Control Program for Microcomputers. To je mali ali dobro projektovan operativni sistem, koji se pojavio 1974. godine. CP/M je jednokorisnički i jednoprogramski: korisnik može da izvršava jedan program u datom vremenu, dok se program ne okonča. Početkom osamdesetih godina njegova upotreba se postepeno smanjuje i onda potpuno prestaje. Danas je najviše poznat kao "operativni sistem prije DOS–a".

12.4.2 Operativni sistemi DOS i Windows

Prvi personalni računari pojavili su se na prelasku iz sedamdesetih u osamdesete godine. Prva verzija operativnog sistema DOS pojavila se 1981. godine, firma Microsoft. Taj operativni sistem koriste IBM–ovi personalni računari zasnovani na Intelovim 16–bitnim mikroprocesorima. Prva verzija bila je jednokorisnička i jednoprogramska. Druga verzija pojavila se 1983. godine. Podržava rad sa hard diskom i hijerarhijsku strukturu direktorijuma. Pored toga, ovaj sistem podržava i multiprogramski režim rada, mada na primitivan način. Treća verzija DOS–a pojavila se 1984. godine. Kasnije se pojavilo još niz novih verzija. Svaka nova verzija donijela je neke novine. DOS je osposobljen da podržava rad mreže personalnih računara, kao i da podržava rad nekih novih uređaja.

U razvoju personalnih računara postojale su dvije glavne linije, koje su konkurentske. Jednu liniju činile su firme Intel i Microsoft, a drugu Motorola i Apple. Njihov personalni računar naziva se Macintosh ili kraće Mac. Odgovarajući operativni sistem je Mac OS. Apple je prvi uveo 1984. godine GUI za personalne računare. GUI je skraćenica za Graphical User Interface (grafički korisnički interfejs). To znači da korisnik sa računarom radi u grafičkom režimu, korišćenjem prozora i pokazivačkih uređaja, kao što je miš.

Od 1987. godine Microsoft je već radio na proizvodnji GUI za DOS. Tako su se pojavile prve verzije Windowsa, 1, 2 i 3. One su bile 16–bitne. Predstavljale su samo dopunu ili nadgradnju DOS–a. Preuzimale su od DOS–a neke poslove i preklapale ga. Sve te verzije bile su jednokorisničke i više-

programske: jedan korisnik mogao je da ima više od jednog posla (jednog procesa) aktivnog u bilo kom datom trenutku. Kasnije verzije Windowsa podržavaju rad mreže personalnih računara. Takve su, na primjer, Windows 95 i Windows 98. One su iskoristile to što su tekući Intelovi 32–bitni čipovi puno moćniji. To su potpuni operativni sistemi, ne oslanjaju se na DOS.

Čitava linija DOS i Windows operativnih sistema je softverski kompatibilna naniže. To znači da računar sa novijom verzijom operativnog sistema može da izvršava program napravljen za stariju verziju. Težnja da kompatibilnost bude očuvana stvara teškoće prilikom projektovanja novih verzija.

Microsoft svoje bogatstvo i mjesto na tržištu duguje tome što su osnivači ove firme pravilno procijenili kakve perspektive ima tržište personalnih računara. Ostali proizvođači su ovo tržište zanemarili, smatrajući da je jedino ozbiljno tržište velikih računara. Paralelno sa razvojem *DOS*-a i *Windows*-a *Microsoft* je razvijao operativni sistem *Windows NT* (Windows New Technology). Iako nalik na *Windows*-e nastale iz *DOS*-a, ovaj sistem je kompletno iznova napisan i predviđen je za veće računare. U početku nije bilo kompatibilnosti Windows NT-a sa verzijama Windows 95 i 98. U skorije vrijeme, došlo je do konvergencije ovih tehnologija, pojavom Windows 2000, zatim Windows XP.

Windows nastavlja da se razvija - Microsoft i dalje proizvodi nove verzije operativnog sistema Windows.

Firme IBM pokušala je da osvoji tržište operativniim sistemom pod nazivom OS/2, ali Windows je prevladao.

12.4.3 Unix i Linux

Krajem šezdesetih godina pojavili su se mini–računari, koji imaju daleko manje hardvera od velikih računara, a ipak mogu dobro da posluže za neke obrade. Sa njima su se pojavili i odgovarajući operativni sistemi, među kojima je i slavni Unix. Napravljen je početkom sedamdesetih godina. Njegov tvorac je Denis Ritchy. Računari PDP–11 firme DEC koristili su operativni sistem Unix. Podržavao je multiprogramski režim rada, a isto i rad u razdijeljenom vremenu.

Denis Ritchy je u isto vrijeme stvorio i čuveni programski jezik – jezik C. Operativni sistem Unix sastoji se (računajući samo jezgro, ne računajući tranzitni dio) od oko 10.000 linija na jeziku C i još približno 1.000 naredbi

na jeziku asemblera. Dakle, svega 5–10% je napisano na jeziku asemblera – Unix je u najvećoj mjeri napisan na višem programskom jeziku. To je dobro svojstvo operativnog sistema, zato što puno olakšava prenosivost operativnog sistema na drugu vrstu računara. Naime, pri prenošenju, samo 5–10% treba da bude ponovo napisano. Ono što je napisano na jeziku C dovoljno je da jednostavno bude prevedeno pomoću odgovarajućeg prevodioca za tu drugu vrstu računara.

U vrijeme nastanka personalnih računara i DOS-a, operativni sistem *Unix* bio je dominantan na radnim stanicama i drugim većim računarima. Kao što smo već istakli, to je izuzetno kvalitetan i pouzdan operativni sistem i zato nije iznenađenje da je većina većih računara u svijetu radila (i danas radi) na tom operativnom sistemu ili nekoj od njegovih mnogobrojnih varijanti, kakva je, recimo, BSD Unix.

Razna dobra rješenja iz operativnog sistema Unix dosta su uticala na sve operativne sisteme koji su se poslije njega pojavili.

Postoje razne varijante Unixa, namijenjene za razne klase–vrste računara. U posljednje vrijeme, sa sve većim napretkom *Microsoft*-ovih operativnih sistema, primat Unix-a više nije tako siguran.

U zadnjih nekoliko godina došlo je i do jednog novog fenomena, pojave *open-source* (*otvoreni kod*) programa. Open-source označava programe čiji se izvorni tekst može slobodno čitati i prepravljati. On je nastao kao reakcija programera-entuzijasta na monopol velikih firmi kao što je *Microsoft* (mada ima mišljenja da iza svega toga stoje *Microsoft*-ovi konkurenti). Hiljade programera pišu programe i besplatno omogućuju zainteresovanim ljudima da čitaju i prepravljaju izvorni kod tih programa, čime se programi sve više dopunjuju i postaju kvalitetniji.

Tako je nastao i danas veoma popularni operativni sistem *Linux*. Prvu njegovu verziju napisao je finski programer Linus Torvalds. Ona u osnovi ima dosta sličnosti sa Unix-om. Vremenom se Torvaldsu sve više ljudi pridruživalo i danas je Linux postao prilično dobar operativni sistem, koji stiče sve veću popularnost. Toliku da već ugrožava i giganta kakav je *Microsoft*. Linux sve više liči na *Windows* a, za razliku od *Windows*-a, besplatan je. Za svaki važniji program za *Windows*, na *Linux*-u postoji sličan koji je potpuno besplatan.

12.5 Startovanje operativnog sistema

Kada se uključi računar, prvi program koji se izvršava je smješten u ROM (samo za čitanje) memoriji. Taj program prvo provjerava da li je sistem ispravan - provjerava rad procesora, memorije, osnovnih ulazno/izlaznih operacija i rezultat upisuje na posebno predviđeno mjesto u memoriji. Kada se provjere uspješno završe, jedan program u ROM-u (koji se ponekad naziva *firmware*) aktivira diskove (flopi, hard ili CD) računara i učitava prvi dio operativnog sistema koji se zove **bootstrap loader**. To je mali program koji ima samo jednu funkciju: da učita operativni sistem u memoriju i omogući da on počne da se izvršava. On učitava male programe, koji se zovu *drajveri*, koji kontrolišu razne hardverske djelove računara. Zatim, on dijeli memoriju na djelove koji će sadržati operativni sistem, informacije o korisnicima, programe, podatke o signalima, komunikaciji,... Na kraju, predaje se kontrola operativnom sistemu.

Kao primjer, razmotrimo računar baziran na Intel 8086 procesoru koji ima operativni sistem MS DOS 3.3.

Računar ima 1MB unutrašnje memorije, od čega je 8KB izrađeno kao ROM čip (kaže se i BIOS ROM čip). ROM čipu odgovaraju najveće adrese u unutrašnjoj memoriji, tako da mu očito pripada i adresa $FFFF:0000 = (FFFF0)_{16}$. Sadržaj ove adrese je $c(FFFF:0000) = jmp\ F000:E05B$. Kaže se da je to "jump address". U ROM-u su upisani bootstrap program ("inicijalna kapisla") i primitivna ulazno-izlazna podrška.

Znamo da procesor ima pin RESET i da se računar resetuje u rezultatu dovođenja signala po toj ulaznoj liniji. Preciznije, biće urađeno $CS \leftarrow FFFF$ i $IP \leftarrow 0000$, pored drugih radnji. Dakle, prilikom pokretanja ili resetovanja računara izvršava se program od adrese $segment : offset = FFFF:0000$.

U okviru izvršavanja tog programa, nakon evidentiranja osnovnih ulazno–izlaznih uređaja, biće prepisana tri fajla (sa "boot" diska) u unutrašnju memoriju, u rezidentni dio DOS-a. To su io.sys (veličine 22KB) koji sadrži BIOS-ove rutine prekida (od $n = 0$ do $n = 31$), msdos.sys (30KB) koji sadrži DOS-ove rutine prekida (od $n = 32$ do $n = 255$) i command.com (25KB) – komandni procesor operativnog sistema. Korisnik će dobiti poruku kada sve dođe na svoje mjesto, tj. kada budu uređene sve pripremne radnje.

Tranzitini dio operativnog sistema zasad ostaje na disku; to su fajlovi

print.exe, diskcomp.com, i drugi. Ovaj operativni sistem isporučivao se na jednoj disketi, imao je oko 1MB.

12.6 Osnovne komponente operativnog sistema

Zbog velikog broja zadataka koje mora izvršiti, operativni sistem je izuzetno složen softver. Da bi pisanje i održavanje operativnog sistema bilo lakše, u njemu se izdvaja nekoliko komponenti koje imaju različite funkcije. Pri tome se uvijek pokušava da se komponente što je moguće više razdvoje, tako da ih je lakše programirati. U idealnom slučaju, komponente moraju biti tako razdvojene da se neka od njih može zamijeniti, a da se ne vrše izmjene u ostalim komponentama.

Osnovne komponente operativnog sistema su:

- upravljanje procesima;
- upravljanje memorijom;
- upravljanje ulazom i izlazom;
- upravljanje fajl-sistemom;
- programski i korisnički interfejs.

Mnogi proizvođači uključuju u operativne sisteme veliki broj dodatnih pomoćnih programa i funkcija, ali praktično svi obavezno uključuju ovih šest funkcija. Razmotrimo ih detaljnije.

<table>
<tr><td colspan="2">PROGRAMI</td></tr>
<tr><td rowspan="4">OPERATIVNI SISTEM</td><td>UPRAVLJANJE FAJL SISTEMOM</td></tr>
<tr><td>UPRAVLJANJE ULAZOM/IZLAZOM</td></tr>
<tr><td>UPRAVLJANJE MEMORIJOM</td></tr>
<tr><td>UPRAVLJANJE PROCESIMA</td></tr>
<tr><td colspan="2">HARDVER</td></tr>
</table>

Slika 12.1: Komponente operativnog sistema

12.6.1 Upravljanje procesima

Jedan od najvažnijih pojmova kod operativnog sistema je pojam *procesa*. Proces je, prosto rečeno, program u izvršavanju. Proces može izazvati izvršavanje drugih procesa (na primjer, za komunikaciju među računarima), a u računaru postoji i mnogo procesa čijeg postojanja korisnik nije ni svjestan. Dakle, proces je softver koji izvodi neku akciju i može biti kontrolisan- od strane korisnika, drugih aplikacija ili operativnog sistema.

Istakli smo da moderni operativni sistemi moraju imati mogućnost da se u isto vrijeme na računaru odvija više poslova, odnosno procesa. Na primjer, dok unosimo tekst u neki program, možemo sa Interneta snimati neki fajl, slušati muziku i štampati neke dokumente. Iako se korisniku čini da se svi ovi procesi izvršavaju u isto vrijeme, to nije tako.

Na jednom procesoru u jednom trenutku može da se izvršava *samo* jedan proces. Da bi se stvorio utisak da više procesa radi istovremeno procesor mora da hiljade puta u sekundi pređe sa jednog na drugi proces. Ovo se naziva *smjenjivanjem procesa na procesoru*.

Pod upravljanjem procesima podrazumijeva se da operativni sistem mora da obezbijedi da svaki proces dobije dovoljno vremena na procesoru i da procesor bude što je moguće bolje iskorišćen.

Izvršavanje procesa se prekida u slučaju čekanja na unos nekih podataka od strane korisnika ili zbog prekida. Vidjeli smo da je prekid specijalni signal koji hardver ili softver šalje procesoru. Interaptom neki dio računarskog sistema zahtijeva pažnju procesora. To može biti u cilju informisanja procesora da je vrijeme za smjenu procesa; obavještenje da je neki posao završen (recimo, da je završeno čitanje sa diska); obavještenje o greški, ... Neki prekidi se mogu ignorisati zbog obavljanja važnijeg posla, a neke je potrebno odmah obraditi.

Svu informaciju potrebnu za smjenu procesa operativni sistem čuva u dijelu RAM-a koji se naziva *tabela procesa*. Tabela procesa sadrži identifikator (oznaku) procesa, mjesta dokle je stiglo izvršavanje, sadržaje registara koje proces koristi, podatke o tome koji dio memorije proces koristi, listu fajlova koje koristi proces, važnost procesa, njegov trenutni status (izvršava se, blokiran, spreman), itd. Pri smjenjivanju procesa operativni sistem mora zapamtiti sve te podatke u tabeli procesa. Nakon toga on mora u odgovarajuća mjesta upisati iste takve podatke nekog drugog procesa, čime će taj drugi proces početi da se izvršava.

Takođe, operativni sistem mora donijeti odluku koji proces će se sljedeći izvršavati. Tu odluku razni sistemi donose na razne načine. Neki sistemi daju svim procesima isto vrijeme; neki uvode prioritete, tj. važnost procesa; neki daju prednost kraćim procesima,... Nema optimalnog načina, svaki ima prednosti i nedostatke. Naravno, sve postaje znatno komplikovanije ako sistem ima više procesora.

12.6.2 Upravljanje memorijom

U računarskom sistemu memorija je jedan od najvažnijih resursa. Dok programi, odnosno zahtjevi za memorijom, bivaju sve veći - memorija je ograničena. Na primjer, dok većina personalnih računara danas ima 128-512 MB RAM, programi su često znatno veći. Dakle, često se dešava da se ni jedan program ne može kompletno smjestiti u operativnu memoriju. Opet, programi i u toku svog rada mogu trebati znatne količine dodatne memorije. Problem postaje još komplikovaniji kada se uzme u obzir da danas većina računara ima multitasking, tj. u operativnoj memoriji se u jednom trenutku nalazi više procesa.

Da bi se prevazišla ograničenost memorije, kao što smo vijeli, u računarski sistem se uvodi nekoliko nivoa memorije -keš memorija (mala, skupa, izuzetno brza), operativna memorija (srednje velika, srednje skupa, srednje brza) i spoljašnja memorija (velika, jeftina, spora). Zadatak je operativnog sistema da koordiniše rad ovih memorija, odnosno da upravlja memorijom. Upravljanje memorijom se, u osnovi, sastoji od dva zadatka:

- svaki proces mora imati dovoljno memorije da bi se mogao izvršavati i on ne smije dirati memorijski prostor drugog procesa;
- razne vrste memorije moraju se pravilno koristiti.

Dio operativnog sistema koji je zadužen za ove zadatke zove se *upravljač memorijom.*

Njegov posao je da vodi računa o tome koji se djelovi memorije koriste i koji su slobodni. On dodjeljuje memoriju procesima kada im je potrebna, i oslobađa je kada više nije potrebna. Takođe, djelovi memorije se, prema potrebi, prebacuju iz operativne memorije na spoljašnju i obratno. Ovaj proces se naziva *swapping* (eng. zamjena). Kod većine operativnih sistema prebacivanje ide u *blokovima* ili *stranicama,* čija je veličina unaprijed određena

(najčešće nekoliko kilobajta). Kada je memorija puna, operativni sistem odlučuje koju stranicu treba izbaciti. Razni operativni sistemi koriste razne pristupe– neki izbacuju stranice proizvoljno, neki prema tome koliko je korišćena, neki prema vremenu kada je stranica stigla u memoriju,... Moderni operativni sistemi omogućavaju i *virtuelnu memoriju.* To je mehanizam koji omogućuje da programi imaju utisak da rade sa memorijom znatno većeg kapaciteta od operativne.

Kod većine računara moguće je povećati kapacitet memorije. Na primjer, možete povećati operativnu memoriju na vašem PC računaru sa 256MB na 512MB. Operativni sistem to zna i prilagodiće svoj rad tome.

12.6.3 Upravljanje ulazom i izlazom

Komunikacija operativnog sistema i hardvera ide kroz specijalni program koji se naziva *drajver.* Drajver je, dakle, posrednik između električnih signala u hardveru, s jedne strane, i programa, s druge. Na primjer, drajver uzima podatke koje ostali dio sistema smatra za *fajl* i prevodi ih u bitove koje upisuje na određene lokacije na disku, ili u seriju operacija lasera u štampaču.

Naravno, u računarskom sistemu postoje velike razlike u hardveru pa postoje i velike razlike u drajverima. Isto je to da kad god je potreban neki uređaj - koristi se drajver. U suštini, drajver je proces kao i bilo koji drugi. Operativni sistem će često dodijeliti drajveru najveće prioritete da bi se uređaj, tj. resurs, što prije oslobodio za druge procese.

Razlog zašto je drajver izdvojen od ostalog dijela operativnog sistema je u tome što se mogu dodavati nove funkcije u drajver bez izmjena ostatka operativnog sistema. Takođe, najbolje će proizvođač samog uređaja, recimo diska, napisati drajver za njega i u potpunosti iskoristiti sve mogućnosti uređaja, nego što će to uraditi proizvođač operativnog sistema. Na primjer, svaki monitor i grafička kartica imaju svoje specifičnosti, drugačiji način rada. Te mogućnosti mogu biti iskorišćene jedino ako se u računar instalira odgovarajući drajver, koji se isporučuje zajedno sa uređajem.

Za slučaj da nemamo odgovarajući drajver, ustanovljen je standard po kome svi proizvođači moraju na isti način podržavati neke osnovne funkcije. Zato proizvođač operativnog sistema, recimo Microsoft, može napisati standardni drajver, koji će raditi sa svim grafičkim karticama i monitorima. Naravno, mogućnosti tog drajvera su ograničene- imaćemo mali broj boja i rezoluciju (broj tačaka na ekranu).

12.6.4 Upravljanje fajl sistemom

Istakli smo ranije da se podaci u spoljašnjoj memoriji računara organizuju u jedinicama koje se nazivaju *fajlovima*. Fajlovi su veoma važni za svaki operativni sistem. Fajlovima upravlja operativni sistem. Operativni sistem mora da obezbijedi udoban, brz i pouzdan rad sa fajlovima. Za korisnika računara je važno kako on vidi fajl, kakva imena su dozvoljena za fajlove, koje sve operacije mogu da se rade sa fajlovima, koje operacije su dozvoljene tom korisniku, itd. Sa druge strane, za sam sistem važno je kakav je način smještanja fajlova, koji djelovi diska pripadaju kom fajlu, kako se zna koji dio diska je slobodan, a koji ne, ...

Kod davanja imena fajlovima postoje razni pristupi. Na primjer, operativni sistem UNIX razlikuje velika i mala slova. Tako su fajlovi sa imenima 'Marko', 'marko',' MaRkO' različiti, dok za operativni sistem MS-DOS ova imena označavaju isti fajl. Neki sistemi ograničavaju i dužinu imena fajla. Osim toga, neki operativni sistemi, recimo MS-DOS, zahtijevaju da fajlovi imaju *ekstenziju,* tj. dio imena koji se nalazi iza tačke. Na primjer, ekstenzija fajla 'marko.txt' je 'txt'. Kod drugih sistema ekstenzija nije obavezna. UNIX fajl može imati i više ekstenzija, na primjer, 'prog.c.Z', a ne mora imati nijednu, na primjer 'pismo'. Ekstenzija obično daje neku dodatnu informaciju o fajlu. U Windows sistemima ekstenzija sistemu kazuje koji je program povezan sa fajlom. Tako je, na primjer, za fajlove nastale iz Word-a ekstenzija 'doc', za fajlove iz Excela - 'xls'. Zato kada dva puta kliknete na fajl 'primjer.doc', operativni sistem startuje Word i u njemu otvara vaš fajl.

Fajlovi se grupišu u *direktorijume* ili, kako to novije verzije Windows-a nazivaju, *foldere.* Za korisnika, direktorijumi su organizovani u drvo na čijim se čvorovima fajlovi. Korijenski direktorijum se u Windows-u označava sa '\'. Puno ime fajla je put od korijena do tog fajla. Na primjer, ime 'c:\Program Files\Microsoft Office\Office\Winword.exe' označava da se fajl 'Winword.exe' nalazi u direktorijumu Office, koji je poddirektorijum direktorijuma 'Microsoft Office', koji je poddirektorijum direktorijuma 'Program Files', a koji je konačno poddirektorijum korijenskog direktorijuma '\' na 'c' disku računara.

Za operativni sistem, direktorijum je takođe fajl u kome je sadržana informacija o tome koji fajlovi pripadaju tom direktorijumi i gdje su oni stvarno na disku. Direktorijum je fajl specijalne vrste, kome korisnik ne može direktno pristupiti. Postoje i druge vrste specijalnih fajlova koje operativni sistem koristi za pristup ulazno-izlaznim uređajima.

Operativni sistem mora voditi računa o tome ko sve može da pristupi fajlu, tj. mora da vodi računa o problemima bezbjednosti. U modernim računarskim sistemima, u kojima je više korisnika, sa velikim mogućnostima mrežnog rada, pitanje bezbjednosti je od izuzetno velikog značaja.

12.6.5 Programski interfejs

Upravljanje svim resursima računara je glavni zadatak operativnog sistema. Za neke sisteme ovo je ujedno i sva funkcionalnost koji treba da imaju. Drugi sistemi, opet, moraju da obezbijede relativno jednostavan način da programi i ljudi koriste računar.

Baš kao što drajveri treba da obezbijede da aplikacije koriste hardver računara bez znanja svih detalja kako taj hardver funkcioniše, *programski interfejs* (Application Program Interface, API) dozvoljava programima da koriste funkcije računara i operativnog sistema bez potrebe da direktno znaju sve potrebne operacije za izvršavanje tih funkcija.

Na primjer, naš program treba samo da kaže operativnom sistemu da napravi neki fajl, a sve ostalo je na operativnom sistemu. Operativni sistem mora da se obrati disku da bi dobio lokaciju prvog raspoloživog mjesta u memoriji. Kada dobije tu informaciju, operativni sistem mora u svoj fajl sistem dodati informaciju o tom fajlu koja, opet, uključuje početnu i krajnju lokaciju fajla, ime i tip fajla, koji korisnici imaju pravo da pristupe fajlu ili promijene njegov sadržaj, datum i vrijeme kreiranja fajla. Zatim operativni sistem mora zapisati na početak fajla razne podatke o tom fajlu, koji će omogućiti korišćenje fajla. U čitavom ovom poslu, cijela komunikacija sa disk uređajem izuzetno je zavisna od proizvođača i modela diska. Sva kompleksnost ovog posla je unutar operativnog sistema i programer ne mora da zna ništa od toga. Njegovo je samo da kaže šta mu treba i da čeka da operativni sistem završi posao.

12.6.6 Korisnički interfejs

Isto kao što programski interfejs omogućuje da aplikacije koriste resurse operativnog sistema, *korisnički interfejs* omogućuje ljudima da koriste računar.

Odmah treba podvući: korisnički interfejs je program, ili skup programa, koji ostvaruje komunikaciju korisnika sa računarom. Ono što vidimo kada uključimo računar je upravo korisnički interfejs.

On sakriva od korisnika svu složenost računara tako da korisnik uopšte i ne mora da zna ništa o tome šta se sve dešava unutar računara. Korisnik ne mora da zna ništa o upravljanju procesima, memorijom, ulazom i izlazom,... Jedino treba da ima neka znanja o fajlovima i direktorijumima i uz minimalnu obuku, može da koristi računar.

Korisnički interfejsi su u zadnjih deset godina uglavnom grafički (eng. *Graphical User Interface* ili GUI). Od njih najviše popularnosti imaju dva modela - *Macintosh* i *Windows*. Naravno, postoji i veliki broj drugih korisničkih interfejsa, i grafičkih i onih koji to nisu. *Unix*, na primjer, ima korisničke interfejse koji se zovu *školjke* (**shells**). Takvi su, recimo, Korn Shell i C Shell. Oni su tekstualni interfejsi koji imaju razne mogućnosti, ali njihova osnovna namjena je da olakšaju korisniku korišćenje računara. Postoje i grafički interfejsi kao što su X-Windows i Gnome pomoću kojih Unix i Linux više liče na Windows i Macintosh operativne sisteme, bar sa tačke gledišta korisnika.

12.7 Pitanja i zadaci

1. Šta je to operativni sistem?
2. Koji su glavne funkcije i zadaci operativnog sistema?
3. Kakvi tipovi operativnih sistema postoje?
4. Koje su glavne komponente savremenih operativnih sistema?
5. Koje su glavne karakteristike operativnih sistema po generacijama?
6. Zašto se primjenjuje modularna struktura pri izradi operativnih sistema?
7. Opišite karakteristike operativnog sistema na kome radite.
8. Objasnite pojam procesa i njihovog smjenjivanja na procesoru.
9. Šta je to tabela procesa?
10. Navedite zadatke upravljača memorijom.
11. Objasnite pojam swapping-a.

12. Šta je to drajver? Nabrojite instalirane drajvere na vašem računaru.
13. Objasnite razliku između fajla i direktorijuma.
14. Šta je to API operativnog sistema?
15. Šta je to shell?

Glava 13

Uvod u strukture podataka

Strukture podataka koriste se prilikom sastavljanja programa. Pod strukturom podataka podrazumijeva se složeni podatak ili složena promjenljiva. Obično se misli na vrstu podataka koja tipično nije ugrađena u programski jezik koji se koristi, i koja se mora izgraditi sa onim što je na raspolaganju. Različiti programski jezici podržavaju razne strukture podataka. Tako, na primjer, strukture niz i skup ugrađene su u Paskal, dok skup ne postoji u jeziku C.

Lista, stek i *red* su neke od najvažnijih struktura podataka. O njima ćemo govoriti u ovom poglavlju.

13.1 Lista

Definišimo prvo pojam liste, a zatim ćemo razmotriti jedan mogući način da se lista implementira u Paskalu.

Lista se definiše kao konačan niz oblika $x_1, x_2, \ldots, x_n$, gdje je $n \geq 0$. Ako je $n = 0$ onda imamo tzv. praznu listu. Za x_k se kaže da je *član liste*. Član liste je podatak određenog tipa. Recimo, član liste može da bude broj, može da bude podatak tipa `integer`. Obično su svi članovi liste podaci jednog te istog tipa. Budući da se radi o (konačnom) nizu to među članovima liste može da ima ponavljanja. Po pravilu, kod liste nije unaprijed poznat maksimalan broj elemenata.

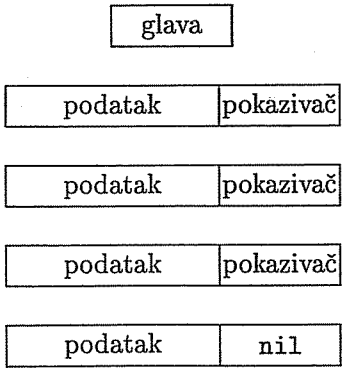
Umjesto lista kaže se i *povezana lista* ili *spisak*. To je zato što se podaci u listi obično ne nalaze na susjednim memorijskim lokacijama (kao što je

to kod strukture podataka niz). Zato kod liste mora postojati način da se povežu susjedni članovi. Obično je povezan prvi element sa drugim, drugi sa trećim, itd. Kada u listi postoje i obratne veze (drugi sa prvim, treći sa drugim, itd.) tada se radi o *dvostruko povezanoj listi.*

Svaka struktura podataka realizuje se pomoću neke postojeće. Za realizaciju liste u, recimo Pascalu, postoji više načina. Mi ćemo razmotriti jedan način, pomoću pokazivača (poentera). Pokazivači su posebne promjenljive koje sadrže adresu neke druge promjenljive. Sa njima smo se sreli i u prethodnim poglavljima - pojam pokazivača povezan je sa pojmom indirektnog adresiranja.

Tokom rada programa, za račun svakog novog člana liste, u memoriji se stvori jedna cjelina koja se sastoji iz dva dijela. Prvi dio sadrži sami član liste, a drugi dio je namijenjen za veznik, odnosno on sadrži pokazivač koji pokazuje na cjelinu koja odgovara sljedećem članu liste. Stvaranje ovakvih cjelina u memoriji, tj. izdvajanje tih cjelina iz slobodne memorije, naziva se *alociranjem memorije.* Alociranje memorije u ovom slučaju se ne obavlja unaprijed, već prema potrebi, kad se pojavi novi element. Kažemo da se ovdje radi o *dinamičkom* alociranju memorije.

Kada radimo sa nizovima, što je jedan mogući način za predstavljanje liste, unaprijed je poznata njegova maksimalna veličina, pa se alociranje obavlja prije startovanja programa, pri loadovanju. To je slučaj *statičkog* alociranja memorije, koje je definisano još prilikom pisanja programa.

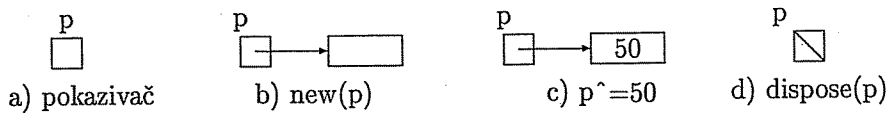


Slika 13.1: Ilustracija liste

Da ponovimo, kod predstavljanja liste pomoću pokazivača jednom članu odgovara jedna cjelina čiji prvi dio sadrži sami član liste, a drugi dio sadrži pokazivač. Pokazivač pokazuje na cjelinu koja odgovara sljedećem članu liste (slika 13.1). Ako pokazivač ima vrijednost `nil` onda znači da smo došli do kraja liste.

Posebnu ulogu ima promjenljiva `glava`, to je tzv. glava liste. Ona pokazuje na prvu cjelinu, tj. na cjelinu gdje je prvi član liste upisan. Ako je `glava = nil` onda to znači da je lista u datom trenutku prazna. Iz ugla programera gledano, lista to je `glava`, odnosno listi se pristupa posredstvom (pokazivačke) promjenljive `glava`. Ako lista treba da se pregleda onda se polazi od glave liste i onda se redom prolazi lanac cjelina, sve dok se ne dođe do kraja liste.

Da bi se razumjelo naprijed izloženo, treba znati osnovne stvari o pokazivačima u Paskalu: to su pojam pokazivača, prostor u memoriji na koji on pokazuje i dvije procedure `new` i `dispose`. Izložimo ove pojmove ukratko (slika 13.2).



Slika 13.2: Pokazivači u Pascalu

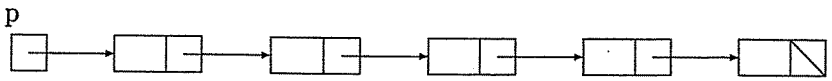
Deklaracija `var p: ^integer` govori da je p adresna promjenljiva, tj. da je p pokazivač na cio broj. Treba dodijeliti vrijednost promjenljivoj p. Naredba `new(p)` ima sljedeći efekat. U memoriji se alocira prostor veličine dva bajta (za cio broj), a adresa tog prostora dodjeljuje se promjenljivoj p da bude njena vrijednost. Kao rezulat ove naredbe, imamo da pokazivač p pokazuje na izdvojeni prostor od 2 bajta u memoriji. Ako želimo da popunimo ta dva bajta, tj. da tamo upišemo neku vrijednost, to radimo pomoću, recimo, naredbe `p^:=50`. Konačno, naredba `dispose(p)` ima sljedeći efekat: razduži se prostor na koji p pokazuje, a vrijednost promjenljive p vraća se na `nil` = adresa koje nema (ili njena vrijednost ostaje nedefinisana). Za naše potrebe, za predstavljanje liste, nisu nam potrebni pokazivači na cijele brojeve, nego pokazivači na zapise (na `record`).

Dakle, lista cijelih brojeva je u Pascalu pokazivač na cjelinu (koja je record), a koja sadrži jedan cijeli broj (podatak, član) i jedan pokazivač koji ukazuje na (sljedeću) cjelinu i koji je u u suštini opet lista. Stoga možemo napraviti sljedeću definiciju tipa:

```
type lista=^cjelina;
cjelina=record
          podatak: integer;
          nastavak: lista
        end;
```

Konkretnu listu, recimo p,možemo sada definisati sa:

```
var p: lista;
```



Slika 13.3: Lista u Pascalu

Postoje dvije glavne operacije sa listom. To su operacija umetanja novog člana i operacija brisanja člana iz liste. Za označavanje tih operacija često se koriste njihovi nazivi na engleskom: INSERT i DELETE, redom.

Razmotrimo prvo operaciju umetanja objekta y u listu. Dakle, objekat y treba da postane član liste, tako da se broj članova liste povećava za jedan. Ako je prije umetanja u listi bilo n članova, sada ih ima $n+1$. Postoje razna mjesta na kojima može da se izvrši umetanje. Recimo, neka y treba da bude umetnut između susjednih članova na pozicijama $k = 2$ i $k = 3$. Novo stanje u listi biće $x_1, x_2, y, x_3, \ldots, x_n$ kada se ostvari umetanje. Opet, ako treba dodati y na samom kraju liste, novo stanje u listi biće $x_1, x_2, \ldots, x_n, y$. I slično.

Za ilustraciju operacije brisanja, uzmimo da je tekuće stanje u listi koju održavamo $x_1, x_2, \ldots, x_n$. Ako se iz liste ukloni član čiji je indeks $k = 3$

onda će novi sastav liste da glasi očito $x_1, x_2, x_4, \ldots, x_n$. Ako bi se uklonio (kažemo i izbacio, isključio, izbrisao) posljednji član, onda bi novo stanje u listi postalo $x_1, x_2, \ldots, x_{n-1}$. Ako bi se uklonio prvi član onda bi novo stanje u listi postalo $x_2, x_3, \ldots, x_n$.

U nastavku će biti dat primjer – program na Paskalu, u kome se iz koraka u korak ažurira stanje u listi L. To znači da se tokom rada programa na listu primjenjuje proizvoljni niz naredbi DELETE i INSERT. Naravno da svaka naredba treba da bude u potpunosti definisana: koji objekat se umeće, na koje mjesto se umeće, koji član se briše, i slično. Pretpostavlja se da je na početku rada programa lista – prazna lista. Uveden je mehanizam da se saopšti da je niz naredbi završen: kada se na ulazu otkuca da je ch = 'p'.

Očigledno je da u slučaju INSERT(y) (ako y–a nema u listi) treba uraditi naredbu NEW u programu, čime se generiše jedan memorijski objekat (cjelina), i podesiti sve pokazivače tako da se ova cjelina uključi u listu. U slučaju DELETE(y), ako je y član liste treba uraditi DISPOSE u programu. Naredbu DISPOSE treba primijeniti na odgovarajuću cjelinu, čime se prostor dealocira. Naravno, prije toga potrebno je preusmjeriti sve pokazivače tako da "zaobiđu" ovu cjelinu.

Donji program na Paskalu služi samo da se izloži kako se izvode glavne operacije sa listom. Realni program bi vjerovatno između dvije uzastopne operacije sa listom vršio druge radnje (koje se tiču liste L ili se ne tiču liste L). Takođe, moguće je u programu ažurirati nekoliko listi, itd.

U primjeru je uzeto da lista služi za održavanje jednog promjenljivog skupa. Zato je predviđeno da među članovima liste ne treba da bude ponavljanja. Ulazni podaci programa sastoje se iz nekoliko redova, gdje pojedini red ima oblik d x ili i x, gdje je x objekat; d znači DELETE a i – INSERT. Predviđeno je da se na kraju rada programa štampa finalni sadržaj liste.

```
program lista(input,output);
label 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
const INSERT='i'; DELETE='d'; PRINT='p';
type lista=^cjelina;
cjelina=record podatak: integer; nastavak: lista end;
var ch: char;
broj: integer;
glava, p, tekuca, brisanje, prethodna: lista;
begin
```

```
    glava:=nil;
 1: {DA VIDIMO STANJE U LISTI}
    write('p');
    tekuca:=glava;
 9: if tekuca=nil then goto 10;
    write(tekuca^.podatak,'x'); tekuca:=tekuca^.nastavak; goto 9;
10: writeln('k');
    {DA URADIMO JEDNU OPERACIJU SA LISTOM}
    read(ch);
    if ch=INSERT then goto 2;
    if ch=DELETE then goto 4;
    if ch=PRINT then goto 6;
 2: readln(broj);
    if glava=nil then
    BEGIN new(p); p^.podatak:=broj; p^.nastavak:=nil; glava:=p;
    goto 1 END;
    tekuca:=glava;
 3: if tekuca^.podatak=broj then goto 1;
    if tekuca^.nastavak=nil then
    BEGIN new(p); p^.podatak:=broj; p^.nastavak:=nil;
    tekuca^.nastavak:=p; goto 1 END;
    tekuca:=tekuca^.nastavak; goto 3;
 4: readln(broj);
    if glava=nil then goto 1;
    if glava^.podatak=broj then
    BEGIN brisanje:=glava; glava:=glava^.nastavak;
    dispose(brisanje); goto 1 END;
    prethodna:=glava; tekuca:=prethodna^.nastavak;
 5: if tekuca=nil then goto 1;
    if tekuca^.podatak=broj then
    BEGIN prethodna^.nastavak:=tekuca^.nastavak;
    dispose(tekuca); goto 1 END;
    prethodna:=tekuca; tekuca:=tekuca^.nastavak; goto 5;
 6: write('pocetak liste ');
    tekuca:=glava;
 7: if tekuca=nil then goto 8;
    write(tekuca^.podatak,' '); tekuca:=tekuca^.nastavak; goto 7;
 8: writeln('kraj liste')
```



```
    end.
```

```
PRVO RACUNAR ODSTAMPA pk
ONDA MI UNESEMO i 10
ONDA RACUNAR ODSTAMPA p10xk
ONDA MI UNESEMO i 20
ONDA RACUNAR ODSTAMPA p10x20xk
ONDA MI UNESEMO i 30
ONDA RACUNAR ODSTAMPA p10x20x30xk
ONDA MI UNESEMO i 30
ONDA RACUNAR ODSTAMPA p10x20x30xk
ONDA MI UNESEMO d 20
ONDA RACUNAR ODSTAMPA p10x30xk
ONDA MI UNESEMO d 10
ONDA RACUNAR ODSTAMPA p30xk
ONDA MI UNESEMO d 22
ONDA RACUNAR ODSTAMPA p30xk
ONDA MI UNESEMO i 40
ONDA RACUNAR ODSTAMPA p30x40xk
ONDA MI UNESEMO i 50
ONDA RACUNAR ODSTAMPA p30x40x50xk
ONDA MI UNESEMO p
ONDA RACUNAR ODSTAMPA pocetak liste 30 40 50 kraj liste
```

U programu, prilikom izvršavanja naredbe DELETE(y) razlikujemo dva slučaja. Prvi slučaj je da objekat y nije prisutan u listi. Tada u rezultatu izvršavanja zadate naredbe DELETE stanje u listi očito ostaje po starom, tj. lista se ne mijenja. Drugi slučaj je ako je u listi prisutan objekat y. Tada ga ukloniti, čime će se lista promijeniti. Slično imamo dva slučaja i kod naredbe INSERT(y). Ako y–a nema u listi onda ga treba umetnuti. A ako je y već prisutan onda listu treba ostaviti po starom, budući da lista služi za evidenciju o **skupu**, tako da ne treba da bude ponavljanja (dupliranja).

Vidimo da se u programu redom izvršava jedna po jedna naredba DELETE ili INSERT. Operacije se zadaju sa ulaza (kao ulazni podaci programa). Prilikom izvršavanja pojedine naredbe DELETE razlikujemo nekoliko slučajeva. Lista je prazna ili y je prvi član liste ili y–a nema u listi ili y

je član ali nije prvi član (tada iz lanca izostavi jednu cjelinu). Slično za INSERT. Lista je prazna ili y je prisutan ili y nije prisutan (tada ga dodajemo da bude zadnji član).

Vidimo da je `glava` statička promjenljiva, a cjeline su dinamičke promjenljive. Da bi se u cjelini nešto promijenilo treba usmjeriti recimo promjenljivu `tekuca` na željenu cjelinu i onda promjenljivoj `tekuca^` dodijeliti odgovarajuću vrijednost.

Naknadno je program dopunjen tako da prije svake operacije štampa tekući sadržaj liste, to je dio programa od `write('p')` do `writeln('k')`.

13.2 Stek i red

Dva važna specijalna slučaja liste su stek i red. Na engleskom se kaže stack odnosno queue. Razlika u odnosu na listu sastoji se u tome što je ograničeno manipulisanje sa strukturom podataka. Drukčije rečeno, manji je skup dozvoljenih operacija, odnosno manje su mogućnosti pojedinih operacija u odnosu na običnu listu. Prvo ćemo razmotriti stek, a zatim red. Dakle, i u jednom i u drugom slučaju radi se o konačnom nizu $x_1, x_2, \ldots, x_n$, gdje je $n \geq 0$.

Za stek je karakteristično da se sve operacije vrše na njegovom kraju, što se uzima i kao definicija steka. Posljednji ili krajnji član steka $x_1, x_2, \ldots, x_n$ jeste x_n. Još se kaže da je x_n *gornji član* steka ili da se x_n nalazi na *vrhu steka*. Slično tome, kaže se da je x_1 *donji član* steka ili da se x_1 nalazi na *dnu steka*.

Prva operacija sa stekom jeste operacija POP – ukloniti iz steka član koji je na vrhu steka. Kada se ova operacija izvrši onda će novo stanje u steku biti očito $x_1, x_2, \ldots, x_{n-1}$. Moguće je da je stek prazan (da je $n = 0$) u trenutku kada je izdata naredba POP. Obično se kaže da je djelovanje naredbe POP u tim okolnostima nedefinisano, pa je zato bolje da se ispita da li je stek prazan prije nego što naredba POP bude izdata.

Druga operacija sa stekom jeste EMPTY (prazan), što predstavlja skraćenicu od "da li je stek prazan". U rezultatu izvršavanja naredbe EMPTY dobićemo odgovor jeste prazan (ako je $n = 0$) ili nije prazan (ako je $n > 0$).

Sljedeća operacija sa stekom jeste WRITE – pročitaj čemu je jednak gornji član steka. U rezultatu izvršavanja naredbe WRITE biće nam saopštena vrijednost x_n. Naredba WRITE samo odštampa x_n, ona ne uklanja x_n iz steka; sastav steka ostaje po starom.

Operacija PUSH ima jedan ulazni parametar odnosno ima jedan argument, označimo ga sa y. Tako da se naredba zapisuje kao PUSH(y) ili kao PUSH y. Značenje je: stavi y na vrh steka odnosno dodaj y da postane novi član steka. Kada se naredba izvrši onda stek ima jedan član više. Dakle, novo stanje u steku biće očito $x_1, x_2, \ldots, x_n, y$.

Izložili smo matematički pojam steka, a sada ćemo se pozabaviti programiranjem, odnosno implementacijom. Prvo pitanje koje se postavlja je koji su mogući načini za memorijsko predstavljanje steka, a zatim treba vidjeti kako se izvode operacije, u zavisnosti od izabranog načina.

Postoje dva glavna načina za predstavljanje steka, pomoću liste i niza. Na primjer, može se postupiti kao u slučaju liste. Dakle, pored glave steka, ima onoliko cjelina u memoriji (oblika član steka + pokazivač) koliko stek ima članova u datom trenutku. Za pojedinu naredbu PUSH y treba uraditi jednu radnju NEW, ako se programira u Paskalu, dok za pojedinu naredbu POP pozvati jednom proceduru DISPOSE. Međutim, implementacija steka pomoću niza puno je jednostavnija i češće se koristi. Zato u nastavku govorimo o predstavljanju steka pomoću niza.

Za stek se izdvoji prostor u memoriji veličine $k \cdot m$ bajta, gdje je k – broj bajta koliko traži pojedini član, a m je maksimalni predviđeni broj članova u steku. U primjeru koji je dat niže izabrano je da bude $m = 10$. Dakle, stek se predstavlja pomoću jednog niza, gdje jedan član niza služi da čuva jedan član steka. U primjeru imamo niz a, a članovi steka su cijeli brojevi (podaci tipa `integer`).

Potrebna je još samo jedna promjenljiva u oznaci obično top (vrh), pa da memorijsko predstavljanje naše strukture podataka (steka) bude zaokruženo. Moguće vrijednosti promjenljive top su top = 0, top = 1, ..., top = m. Smisao promjenljive top je sljedeći. Njena vrijednost jednaka je poziciji u nizu (u nizu a u našem primjeru) na kojoj se u datom trenutku nalazi gornji član steka. Na primjer, ako je top = 3 onda to očito znači da stek ima $n = 3$ člana i da je sastav steka a_1, a_2, a_3 u datom trenutku. Ako je top = 0 onda je stek prazan. Ako je top = m onda je stek pun. Broj članova steka uvijek je jednak tekućoj vrijednosti promjenljive top.

Može se reći da top određuje gdje je logički kraj steka, a da je tokom rada programa veličina prostora izdvojenog (alociranog) za stek stalno jedna te ista ($k \cdot m$ bajta), odnosno da ta veličina ne zavisi od trenutnog broja članova steka. Ovim metodom alociranje je predviđeno još tokom prevođenja programa, odnosno izvršeno je na početku rada programa ili bolje rečeno prije početka rada programa. Kao što smo ranije istakli, kažemo da se tada radi o statičkom alociranju, dok je dinamičko alociranje ono koje se izvodi tokom rada programa, korišćenjem pokazivača.

Do sada smo naveli četiri operacije sa stekom i to POP, EMPTY, WRITE i PUSH a sada će spisak mogućih operacija da bude kompletiran navođenjem još dvije preostale operacije: FULL i inicijalizacija.

Prije dodavanja novog člana (naredba PUSH) bolje je da se pogleda da li ima mjesta da se doda. Pogledati da li je u datom trenutku stek pun, tj. da li je top $= m$. Operacija FULL saopštava "stek je pun" ako je trenutno top $= m$, odnosno "stek nije pun" ako je u datom trenutku top $< m$.

Posao oko inicijalizacije steka sastoji se u tome da se na početku promjenljivoj top dodijeli vrijednost nula – na početku rada programa stek je prazan. Pored radnje top $\leftarrow 0$, u posao oko inicijalizacije ponekad se ubraja i radnja alociranja memorijskog prostora za članove steka, o čemu je bilo riječi maločas.

Sastavimo sada program na Paskalu za ilustraciju rada sa stekom. Neka se tokom rada programa na jedan stek primjenjuju razne operacije EMPTY, POP, FULL, PUSH i WRITE, u proizvoljnom redosljedu.

```
   program stack(input,output);
   label 1, 2, 3, 4, 5, 6, 7, 8;
   var a: array[1..10]of integer;
   top: integer;
   {a + top = struktura podataka}
   podatak: integer;
   ch: char;
   begin
   {prvo inicijalizovati stek}
   top:=0;
8: {moze write top, a[1], ..., a[top]}
       {if top>0 then
```



```
       for podatak:=1 to top do
       write(a[podatak],' ');
       writeln(top);}
   {redom jedna po jedna operacija}
   readln(ch);
   if ch='u' then goto 1;
   if ch='o' then goto 2;
   if ch='w' then goto 3;
   if ch='e' then goto 4;
   if ch='f' then goto 5;
   if ch='x' then goto 6;
1: readln(podatak);
   top:=top+1;
   a[top]:=podatak;
   writeln('uradio sam push');
   goto 7;
2: top:=top-1;
   writeln('uradio sam pop');
   goto 7;
3: writeln('na vrhu je ',a[top]);
   goto 7;
4: if top=0 then writeln('jeste prazan')
   else writeln('nije prazan');
   goto 7;
5: if top=10 then writeln('jeste pun')
   else writeln('nije pun');
   goto 7;
7: goto 8;
6: writeln('kraj programa')
   end.
```

Program prihvata sljedeće komande:
e – empty – prazan;
f – full –pun;
u – push – stavi (dodaj);
o – pop – skini (oduzmi);
w – write – odštampaj gornji član;
x – kraj rada.

Jedan primjer rada ovog programa dajemo u tabeli 13.1. U tabeli su prikazani izdvojeno ulazni podaci, koje unosimo tokom rada programa, kao i rezultati koje računar saopštava.

STEK GLASI	MI ŽELIMO	MI OTKUCAMO	RAČUNAR ODŠTAMPA
	prazan?	e	jeste prazan
	pun?	f	nije pun
	push 40	u ENTER 40	uradio sam push
40	prazan?	e	nije prazan
40	pun?	f	nije pun
40	push 50	u ENTER 50	uradio sam push
40, 50	push 60	u ENTER 60	uradio sam push
40, 50, 60	pop	o	uradio sam pop
40, 50	pop	o	uradio sam pop
40	push 20	u ENTER 20	uradio sam push
40, 20	write	w	na vrhu je 20
40, 20	write	w	na vrhu je 20
40, 20	prazan?	e	nije prazan
40, 20	pun?	f	nije pun
40, 20	završeno	x	kraj programa

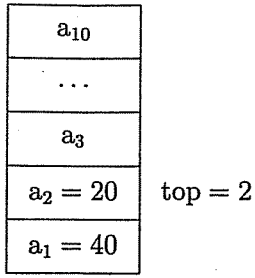
Tabela 13.1: Primjer rada programa stack.pas

Na slici 13.4 prikazano je finalno stanje steka za ulazne podatke koji su bili uneseni. Prikazane su vrijednosti $a_1, a_2, \ldots, a_{10}$ i top.

Kao što smo vidjeli, kod savremenih računara rad sa stekom često je hardverski podržan. Recimo, u slučaju Intelovog procesora 8086 kažemo da je SS:SP adresa gornjeg člana steka. Takođe znamo da sistemski stek služi za spašavanje povratnih adresa u slučaju prekida. Što se tiče aplikativnog programera, bolje je da za svoje potrebe organizuje poseban stek.

Red

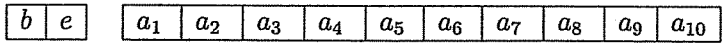
Razmotrimo pojam reda. Znamo da opšti oblik liste glasi $x_1, x_2, \ldots, x_n$, gdje je $n \geq 0$. Kaže se da se x_1 nalazi na početku liste i da se x_n nalazi na kraju liste. Već je rečeno da red predstavlja jedan poseban slučaj liste. Za red



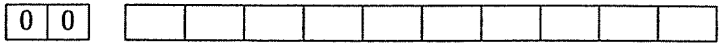
Slika 13.4: Konačno stanje steka

je karakteristično da se operacija umetanja člana izvodi na njegovom kraju, a operacija brisanja člana na njegovom početku; ili obrnuto. Na primjer, ako se izda naredba `insert` y, onda će poslije njenog izvršavanja novo stanje u redu da bude $x_1, x_2, \ldots, x_n, y$. Ako se izda naredba `delete`, u rezultatu izvršavanja te naredbe imaćemo $x_2, \ldots, x_n$ kao novo stanje u redu. Drugi naziv za red je *bafer*.

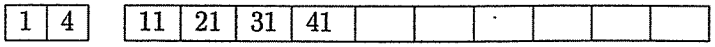
Za memorijsko predstavljanje reda obično se koristi jedan niz, koji sadrži članove reda, zajedno sa dvije promjenljive b (begin, početak) i e (end, kraj). b govori koja je pozicija u nizu prvog, a e posljednjeg člana reda. b i e su promjenljive tipa indeks. U primjeru koji slijedi uzeto je da su moguće vrijednosti za indeks od 1 do 10. Takođe, uzeto je da b = e = 0 znači da je red prazan. Prirodno je da predstavljanje bude dograđeno dodavanjem kružnog svojstva, da bi se izbjeglo premještanje članova reda kada se dođe do kraja u nizu. Ako je $a_{10} = x$ i $a_1 = y$ onda red glasi $\ldots, x, y, \ldots$ u većini slučajeva. Primjer koji slijedi služi i da ilustruje kružno svojstvo. Oznake imaju smisao



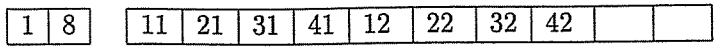
Kada je red prazan, stanje memorije je sljedeće:



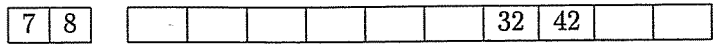
Izvršimo naredbe `insert 11`, `insert 21`, `insert 31` i `insert 41`. Red sada glasi 11, 21, 31, 41. U memoriji imamo:



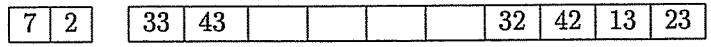
Izvršimo sada naredbe `insert 12`, `insert 22`, `insert 32` i `insert 42`. Red glasi 11, 21, 31, 41, 12, 22, 32, 42. Stanje memorije je



Nakon što izvršimo naredbe `delete`, `delete`, `delete`, `delete`, `delete` i `delete`, red glasi 32, 42, a memorija ima izgled:



Konačno, izvršimo naredbe `insert` 13, `insert` 23, `insert` 33 i `insert` 43, nakon čega red glasi 32, 42, 13, 23, 33, 43. Finalno stanje memorije je



Primjer za upotrebu reda u sistemskom softveru jeste tzv. kružni bafer tastature, čija veličina iznosi recimo 20 bajta. 20 karaktera može da se otkuca unaprijed.

13.3 Zadaci za vježbu

1. Pomoću nekog komercijalnog softverskog paketa editovati fajl `lista.pas` i onda generisati izvršni fajl `lista.exe`; to znači da se izvorni fajl prevede i da se prevedeni oblik zapiše na disk. Nakon toga, vježbe radi, iz operativnog sistema pozivati izvršni fajl.

2. Slučaj kada se ulazni podaci čitaju iz posebnog fajla umjesto sa tastature, a rezultat se šalje u drugi fajl umjesto na ekran. Prvo editovati fajl `listaula.dat` i zapisati ga na disk; sadrži ulazne podatke. Zatim otkucati

```
lista < listaula.dat > listaizl.dat
```

u `ms dos promptu` (pretpostavljamo da se radi u Windows operativnom sistemu). Kada računar završi rad, osvjedočiti se da je na disku prisutan i fajl `listaizl.dat` i da sadrži rezultate rada programa. Kaže se da je izvršeno preusmjeravanje ulaza (pomoću `< listaula.dat`), a takođe i izlaza (pomoću `> listaizl.dat`).

3. Za listu $L = (x_1, x_2, \ldots, x_n)$ kaže se da je uređena ako važi relacija $x_1 \leq x_2 \leq \ldots \leq x_n$. Neka ulazni podatak programa bude niz cijelih brojeva. Sastaviti program na Paskalu koji, čim se jedan broj učita, stavlja taj broj u listu, i to tako da lista bude uređena; redosljed u listi ne poklapa se obavezno sa redosljedom u nizu.

4. Neka ulazni podatak programa bude niz cijelih brojeva. Sastaviti program na Paskalu koji od tih brojeva postepeno obrazuje uređenu listu, ali tako da u listi nema ponavljanja. Ako se broj x pojavljuje u nizu dva puta ili više puta, onda će x samo jednom biti upisan u listu (prilikom prvog pojavljivanja).

5. Učitava se n brojeva i obrazuje se njihova lista ($n \geq 0$). Ispituje se da li u listi ima više parnih nego neparnih brojeva i saopštava se odgovor. Sastaviti odgovarajući program na Paskalu.

6. Učitava se n brojeva i obrazuje se njihova lista ($n \geq 0$). Zatim se iz liste uklanjaju svi negativni brojevi, s tim da redosljed nenegativnih članova ostane neizmijenjen. Sastaviti odgovarajući program na Paskalu.

7. Učitava se n brojeva i obrazuje se njihova lista ($n \geq 0$). Treba obrazovati drugu listu u kojoj je redosljed članova preokrenut. Sastaviti odgovarajući program na Paskalu.

8. Sastaviti potprogram na Paskalu koji ima dva ulazna parametra $glava_1$ i $glava_2$ koji su početni pokazivači listi cijelih brojeva L_1 i L_2. Dakle, pretpostavlja se da su dvije liste već obrazovane. Pored toga, pretpostavlja se da su jedna i druga lista uređene. Treba obrazovati listu L_3 koja sadrži sve članove jedne i druge liste i koja je takođe uređena. Dakle, $n_3 = n_1 + n_2$, gdje je n_i broj članova liste L_i ($1 \leq i \leq 3$). Tako da potprogram ima i treći parametar (izlazni) u oznaci $glava_3$, čiji je smisao očito: glava, odnosno početni pokazivač, liste L_3.

9. Rad sa stekom, evidencija o pozivima potprograma. Sastaviti program na Paskalu koji će modelirati dešavanja u nekom programu P_0, u dijelu pozivanja potprograma. Neka P_0 sadrži tri potprograma P_1–P_3 i neka je moguće da P_i poziva P_j ($0 \leq i \leq 3$, $1 \leq j \leq 3$). Ako bi se P_0 izvršavao onda bi bilo puno pozivanja, čija se priroda i redosljed ne mogu unaprijed utvrditi. Za račun svakog pozivanja (CALL) treba staviti povratnu adresu na stek, a za račun svakog povratka (RETURN) treba naravno sa vrha steka uzeti povratnu adresu.

10. Sastaviti program na Paskalu `queue.pas` za ilustraciju rada sa redom. Neka se članovi reda drže u kružnom baferu.

Glava 14

Paradigme programiranja

Paradigma programiranja je generalan pristup programiranju ili rješavanju problema korišćenjem programskog jezika. Stoga se programski jezici koji imaju slične karakteristike grupišu u određenu paradigmu.

Programski jezik je formalni jezik za neki oblik rješavanja problema. Treba razlikovati programske jezike od nekih specijalnih jezika, kao što su jezici za opis hardvera, jezici za opis dokumenata (SGML i HTML) ili gramatike. Programski jezici obavezno sadrže neku predstavu programa koji se izvršava na računaru da bi se obavio neki postupak.

Paradigma programiranja je istovremeno i metod za rješavanje problema, kao i pristup konstrukciji programskog jezika. Treba napraviti distinkciju između programskog jezika i *programskog okruženja* ili *programskog alata.* Programsko okruženje je skup programa koji omogućuju unos koda, prevođenje, ispravljanje grešaka, kao i druge alate koji pomažu korisniku u razvoju programa. Takođe, programsko okruženje sadrži veliki broj gotovih procedura, koje se lako uključuju u program. Česta je pojava da se nekorektno smatra da je neki jezik bolji od drugog samo iz razloga što za jedan od njih postoji bolje programsko okruženje.

Potrebno je napraviti razliku između simboličkog i nesimboličkog (ili numeričkog) programiranja. U vrijeme nastanka računara mislilo se da je jedino potrebno da računar izvodi razna izračunavanja, pa je to bila glavna upotreba računara. Zato su jezici kao što je FORTRAN uglavnom korišćeni za izvođenje kompleksnih kalkulacija. Vremenom su znakovi i stringovi (nizovi znakova) postali tipovi podataka sa kojima se moglo manipulisati u konvencionalnim programskim jezicima. S druge strane, simbolički program-

ski jezici, kao što su LISP i PROLOG, specijalno su razvijeni da manipulišu sa apstraktnim simbolima, i da izvode obradu lista. Svaki program koji zahtijeva neki oblik 'razumijevanja', zaključivanja ili kompleksne interpretacije mora da izvodi simboličku obradu i zato se dobro izvodi korišćenjem jezika kao što su LISP ili PROLOG. To su razlozi zbog čega su LISP i PROLOG glavni jezici koji se koriste u vještačkoj inteligenciji.

Glavne paradigme programiranja su:

- proceduralno programiranje;
- objektno-orijentisano programiranje;
- funkcionalno programiranje;
- logičko programiranje;
- rule-based (na pravilima zasnovano) programiranje;
- upitni jezici baza podataka;
- vizuelno programiranje;
- script-based (na skriptovima, tj. scenarijima zasnovano);
- programiranje na osnovu demonstracije.

Ostale paradigme programiranja uključuju simulaciju (npr. SIMULA) i tabelarne proračune (spreadsheets).

14.1 Proceduralno programiranje

Proceduralno programiranje dugo je bilo najčešća forma programiranja. Program je serija instrukcija koje rade na promjenljivim. Drugi naziv je proceduralno programiranje je *imperativno* programiranje. Proceduralno programiranje je prilično povezano sa Fon Nojmanovom arhitekturom računara i rani proceduralni jezici su bili samo malo složeniji od asemblera.

Primjeri proceduralnih programskih jezika su FORTRAN, ALGOL, Pascal, C, MODULA2, Ada, BASIC. Bez obzira na njihove razlike, svi oni imaju zajedničku karakteristiku proceduralnog programiranja.

Kao metod dizajna, proceduralno programiranje pokušava da oponaša ljudski metod rješavanja problema putem izvođenja niza operacija. Dakle, čovjek prvo uradi korak 1, zatim korak 2, itd. Dodatno, postoje kontrolne strukture, kao što je IF-THEN-ELSE, u svim modernim proceduralnim jezicima, koje omogućuju promjenu redosljeda koraka. Promjenljive igraju centralnu ulogu u programima, i njihov opseg važenja je važan pojam u blokovski strukturiranim jezicima.

Prednosti proceduralnog programiranja uključuju njegovu relativnu jednostavnost, lakoću implementacije kompajlera i interpretera. Proceduralni jezici su važni u istoriji računarstva zbog manje upotrebe memorije. Dijkstra je vjerovao da uvođenje mikrokompjutera ometa razvoj računarskih nauka. To je dijelom zato što su mikrokompjuteri sa malom memorijom bili sposobni da izvršavaju samo programe na jezicima kao što je BASIC sa 8 KB interpreterima, a da jezike koji su postojali za velike (mainframe) i miniračunare (kakav je bio, recimo, LISP) nije bilo moguće implementirati na mikroračunarima. Naravno, ova situacija je kasnije drastično promijenjena pojavom mikroračunara sa velikim memorijma, ali, ipak, ova istorija je imala uticaj na tržište i ljudsku percepciju programskih jezika.

Nedostaci proceduralnih jezika uključuju teškoće u razumijevanju programa i, do određenog nivoa, teškoće paralelizacije. Takođe, proceduralni jezici sve više postaju jezici niskog nivoa u poređenju sa drugim savremenim paradigmama, koje su znatno bogatije u konceptima, i kao rezultat mogu biti znatno manje produktivni.

14.2 Objektno-orijentisano programiranje

Objektno-orijentisano programiranje uveo je Xerox jezikom Smalltalk. Namjera je bila da se riješi kriza softvera, tako što bi se napravio programski jezik za krajnjeg korisnika. Ipak, Smalltalk ima prilično kompleksnu sintaksu i nije lak za korišćenje, naročito ne za krajnje korisnike.

U isto vrijeme dok se razvijao Smalltalk, jedan drugi dio Xerox-a razvio je jedan objektni sistem unutar LISP-a, poznat pod nazivom LOOPS. Ovo je bila osnova za noviji CLOS (Common LISP Object System).

Najviše korišćeni objektno-orijentisani jezik je C++, koji je objektno proširenje jezika C, mada se to sve više mijenja velikom popularnošću jezika Java, koji ima puno sličnosti sa C++.

Glavni koncepti kod ove paradigme programiranja su:

- objekat
- klasa.

Objekat je fizička ili konceptualna stvar koju srećemo u našem univerzumu. Objekti su, na primjer, ljudi, dokumenti, predmeti, ali i apstraktne stvari kao što su brak, ljubav, zaposlenje, itd.

Objekti imaju određeno *stanje*. Stanje je skup okolnosti koje opisuju objekat. Recimo, stanje prekidača za struju može biti "uključen" ili "isključen". Za kompleksne objekte, kakav je recimo neki čovjek, potrebno je mnogo više da se opiše njegovo stanje.

Objekti se opisuju *svojstvima* ili *atributima*. Tako nekog konkretnog čovjeka možemo opisati kroz ime, prezime, datum rođenja, itd.

Slični se objekti grupišu u *klase*. Pojam klase ima razna značenja, zavisno od konteksta. Klasa je šablon, model, nacrt za kategoriju stukturno identičnih stvari. Tako možemo reći da je čovjek klasa, a svaki konkretni čovjek je objekat, odnosno *primjerak* ili *instanca* te klase.

Klasa se opisuje kroz opis atributa koji karakterišu objekte te klase, kao i kroz *metode* odnosno *funkcije članice* (klase), koje služe za manipulisanje podacima, tj. vrijednostima atributa.

Razmotrimo ovo kroz jedan primjer. Kod koji slijedi napisan je u jeziku C++:

```
class sfera       //deklaracija klase sfera
{
public:
  float r;       //poluprečnik
  float x,y,z;  //koordinate centra
  float povrsina();
  float zapremina();
  void povecaj_r(float delta);
}

float sfera::povrsina() //definicije njenih metoda
{
  return(4*r*r*PI);
}
float sfera::zapremina()
{
  return(4*r*r*r*PI/3);
}
void sfera::povecaj_r(float delta);
{
   r=r+delta; //uvećaj r za delta
}
```


Klasa sfera ima četiri atributa, r, x, y i z – svi imaju *domen* (tip) float, odnosno realni broj. Konkretne vrijednosti ovih atributa definišu jedan objekat – sferu u prostoru.

Ova klasa ima i tri metode– dvije funkcije koje daju zapreminu i površinu sfere, i jednu koja ne vraća ništa ali zato ima jedan float argument, za koji se uvećava poluprečnik sfere.

Nakon ovakve definicije klase možemo konstruisati njene objekte.

```
sfera a;
sfera b;
```

Ovaj kod definiše dva objekta klase sfera – a i b. Oni imaju iste atribute (r, x, y i z), ali svaki od njih ima sopstvene vrijednosti tih atributa. Recimo:

```
a.x=1.0;  a.y=1.0; a.z=1.0; a.r=4;
b.x=2.0;  b.y=3.0; b.z=4.0; b.r=5;
```

Metoda se poziva sa nekim primjerkom klase. Tako će poziv funkcije

```
a.zapremina()
```

dati zapreminu objekta a (268.08) dok će

```
b.zapremina()
```

dati zapreminu objekta b (523.59). Slično,

```
a.povecaj_r(1.5)
```

uvećaće poluprečnik sfere a za 1,5.

Važan koncept u objektno-orijentisanom programiranju je i *enkapsulacija* (engl. encapsulation) ili *učaurenje*. To je mogućnost da se neki atributi ili metode klase proglase privatnim, odnosno nevidljivim van metoda klase. Time se postiže da se objektima može manipulisati samo kroz precizno određene metode i atribute.

Tako u klasi "avion" atribut "visina" može biti nedostupan za direktne izmjene, pa ne postoji opasnost da ga neki dio programa – slučajno ili (zlo)namjerno – postavi na nemoguću vrijednost, recimo -100000. Sve izmjene će ići kroz metode "povećaj visinu" i "smanji visinu", koje strogo kontrolišu ovaj atribut. Znači, objekat ostaje učauren, tj. programer može zatvoriti unutrašnjost objekta za spoljni svijet, i dati precizan interfejs za manipulaciju objektom. Tako se smanjuje mogućnost greške i olakšava korišćenje objekta. Da se poslužimo metaforom, da bi vozili automobil dovoljno je da znamo da radimo sa volanom, mjenjačem, gasom i kočnicom, a ne moramo da znamo ništa o klipovima, karburatoru, anlaseru i drugim unutrašnjim djelovima i sistemima.

Izuzetno važan je i koncept *nasljeđivanja* klasa. Njime se omogućuje da se jedna klasa izvede iz druge, tj. da se pri konstrukciji klase iskoriste (*naslijede*) svi atributi i metode neke druge klase. Pri tome se mogu dodati novi atributi i metode, kao i predefinisati neki od postojećih.

```
class kolor_sfera(sfera)        // klasa kolor_sfera
{                               // je nasljednik klase sfera
public:
  int boja;       //boja sfere
}
```

Klasa kolor_sfera je nasljednik klase sfera. Objekat ove klase ima atribut *boja*, ali i atribute r, x, y i z, a na njega su primjenljive metode *povrsina*, *zapremina* i *povecaj_r*.


```
kolor_sfera c;
float m;
c.x=4.2;  c.y=-2.0; c.z=1.5; c.r=15;
c.boja=3;
c.povecaj_r(2.17);
m=c.zapremina();
```

Objektno-orijentisano programiranje karakteriše definisanje klasa objekata i njihovih svojstava. Nasljeđivanje svojstava jedan je od načina da se smanji obim programiranja, a uključivanje gotovih biblioteka klasa u programsko okruženje, koje se zatim lako koriste u programu - takođe može značajno da smanji potreban trud.

Objektno-orijentisano programiranje dokazalo se kao naročito uspješno pri programiranju korisničkog interfejsa (posebno u Windows okruženju), što je jedan od glavnih razloga za njegovu sve veću popularnost, zbog koje ova paradigma značajno potiskuje proceduralnu.

14.3 Funkcionalno programiranje

Funkcionalno programiranje je bazirano na pojmu programa kao funkcije, u sličnom smislu kao u matematici. Programi se dizajniraju tako da budu kompozicija funkcija.

Najraniji funkcionalni jezik je LISP koji je razvio John McArthy sa MIT-a, kasnih 1950-ih. On je baziran na lambda računu. Ipak, u ranoj fazi razvoja LISP-a uvedeni su elementi ne-funkcionalnog programiranja, i zato je moguće dizajnirati programe u LISP-u u proceduralnom stilu. Riječ LISP nastala je od LIST Processing (obrada lista) i to je glavni jezik za aplikacije obrade lista.

Među kvalitetima LISP-a posebno se izdvaja njegova sposobnost da kompajlira funkcije za vrijeme izvršavanja (u run–time režimu). LISP programi su i sami strukture podataka kojima je moguće manipulisati u LISP-u i zato nije teško napisati programe koji mijenjaju sami sebe dok se izvršavaju. Takođe, moguće je dizajnirati i kompajlirati programe top - down metodom bez potrebe da funkcije nižeg nivoa budu prethodno dizajnirane. Kako modifikacije dizajna jedne funkcije nemaju negativnih bočnih efekata na druge

funkcije, direktno je omogućeno modularno programiranje. LISP se značajno koristi za dizajn novih programskih jezika.

Zbog LISP-ove nadmoći u razvoju programa za vještačku inteligenciju (AI), on je uvijek zahtijevao sofisticirana programska okruženja. Symbolics Inc. je razvio cijelu seriju računara čiji su procesori bili specifično dizajnirani da efikasno izvšavaju LISP programe. To je išlo toliko daleko da su njihovi sistemi imali dugme na tastaturi označeno sa DWIM (Do What I Mean)– uradi ovo što mislim! Korisnik zato može očekivati da LISP sistem u određenoj mjeri razumije njegove namjere i ovo ilustruje nešto od te moguće sofistikacije. LISP sistemi su bili među prvima koji su sadržali editore interfejsa koji su omogućavali crtanje interfejsa i automatsku konstrukciju koda.

Interesantno je da LISP koristi tzv. prefiksnu notaciju. To znači da se kod izraza prvo piše operacija pa onda operandi. Recimo, ne piše se n+1, nego + n 1.

Na primjer, u LISP-u se ovako definiše funkcija koja uoređuje da li je suma dva broja (x i y), veća od trećeg (z):

```
(defun suma-veca (x y z)
  (> (+ x y) z))
```

Slično, rekurzivna funkcija koja računa faktorijel $n!$ može se definisati kako slijedi:

```
;;; Funkcija koja računa faktorijel broja.
(defun faktorijel (n)
   (if (= n 1)
       1
       (* (faktorijel (- n 1)) n)))
```

Konačno, navedimo i jedan primjer funkcije koja radi sa listama.

```
;;; funkcija koja računa dužinu liste
(defun duzina (lst)
  (if (null lst)
      0
      (+ (duzina (cdr lst)) 1))) ;; cdr uklanja glavu liste
```


Među funkcionalne jezike spadaju SCHEME, HOPE i ML.

Glavne prednosti funkcionalnog programiranja su u tome što se program lako razumije i lako je formalno zaključivati o njemu, kao i činjenica da se funkcije lako koriste na drugim mjestima. Moguće je razvijati i održavati jako velike programe sa hiljadama funkcija, jer funkcije nemaju bočnih efekata i zato ih je moguće u potpunosti testirati. Svakako, kao i kod drugih programskih jezika, moguće je i u LISP-u pisati nerazumljive programe.

U veće nedostatke funkcionalnog programiranja spada teškoća izvođenja ulaza-izlaza jer to inherentno nije funkcionalno. Postoje i drugi aspekti rješavanja problema koji se ne mogu lako izvesti u funkcionalnom maniru. Ipak, veliki programi mogu biti razvijeni sa oko 80% čisto funkcionalno dizaniranog koda.

14.4 Logičko programiranje

Logičko programiranje često se izjednačava sa programiranjem u PROLOG-u, iako postoje drugi logički programski jezici. Razlog tome je raširena upotrebe PROLOG-a.

Logičko programiranje usvaja drugačiji pristup rješavanju problema u odnosu na proceduralno i funkcionalno programiranje. U suštini, logičko programiranje zahtijeva logički deklarativan opis prirode problema. Kako se program izvršava nije ono o čemu treba programer da brine - to je zadatak PROLOG-ove "mašine" za izvođenje (zaključivanje). Rezultat je da se značajno podiže nivo apstrakcije u programiranju, jer program ne sadrži eksplicitnu kontrolnu informaciju.

PROLOG program sastoji se od činjenica i pravila. Izvršavanje programa sastoji se u postavljanju pitanja o činjenicama.

Na primjer, naš program može da sadrži sljedeće činjenice

```
voli(Marko,kruška).      /* 'Marko voli krušku' */
voli(Marko,jabuka).      /* 'Marko voli jabuku' */
voli(Janko,jabuka).      /* 'Janko voli jabuku' */
voli(Petar,jabuka).      /* 'Petar voli jabuku' */
```

i pravilo

```
voli_krušku_jabuku(X):-(X,kruška),(X,jabuka).
 /* Neko voli krušku i jabuku,
     ako on voli krušku i on voli jabuku */
```

Sada možemo postaviti pitanja PROLOG sistemu:

```
?- voli(Marko,kruška).    /* da li Marko voli krušku?  */
yes                       /* PROLOG sistem odgovara DA */
?- voli(Petar,jabuka).    /* Da li Petar voli jabuku?  */
yes                       /* PROLOG sistem odgovara DA */
?- voli(Vesna,jabuka).    /* Da li Vesna voli jabuku?  */
no                        /* PROLOG sistem odgovara NE,
                  jer nema relacije između Vesne i jabuke */
?- voli(Marko,_).         /* Voli li Marko bilo šta?   */
yes                       /* PROLOG sistem odgovara DA */
?- voli(_,banana).        /* Voli li ko banane?        */
no                        /* PROLOG sistem odgovara NE */
?- voli(Marko,X)          /* Šta sve voli Marko?       */
X=kruška ;                /* krušku, pritisnemo ; za dalje */
X=jabuka ;                /* jabuku, pritisnemo ; za dalje */
no                        /* ništa više                */
?- voli_krušku_jabuku(X) /* Ko voli i krušku i jabuku  */
X=Marko ;                 /* Marko, pritisnemo ; za dalje */
no                        /* niko više                 */
```

Prednosti logičkog programiranja uključuju razvoj konciznih rješenja problema, kao i mogućnost zaključivanja o programima.

Glavni nedostaci PROLOG-a su sporo izvršavanje, kao i teškoće u savladavanju kompleksnog PROLOG programiranja. Tome mnogi dodaju i teškoće u razumijevanju složenih programa i probleme u dibagovanju većih programa. Ipak, pažljivim dizajnom moguće je kreirati PROLOG programe koji se u potpunosti deklarativno razumiju. Dakle, ne bi trebalo da se zahtijeva od onoga ko čita program da pokušava da vidi kako bi se izvodilo izvršavanje programa, što je poznato kao proceduralna semantika. Nažalost, mnogi PROLOG programi se ne mogu razumjeti bez razmatranja proceduralne semantike.

14.5 Rule-based programiranje

Rule based (na pravilima zasnovano) programiranje poznato je i kao IF-THEN programiranje i kao dizajn ekspertnih sistema. Ono ima mnogo sličnosti sa logičkim programiranjem, i u suštini je njegova varijanta. Ipak, praktični rule-based sistemi su vrlo različiti u svom dizajnu i načinu korišćenja u odnosu na PROLOG.

Rule-based programiranje ima korijene u jednom od prvih računara – Colossus-u, razvijenom od strane Alena Tjuringa i njegovog tima u cilju dekodiranja poruka sa njemačkih podmornica u II svjetskom ratu. Rule based sistemi su često poznati i kao Postovi produkcioni sistemi, prema čovjeku koji je prvi razvio ovu ideju.

Kao kod logičkog programiranja, programer ne mora vodi računa o detaljima izvršavanja programa. Oni su ostavljeni sistemu za izvršavanje ekspertnog sistema. Zato ovo podiže nivo apstrakcije u programiranju.

Dio razvoja metoda rule-based programiranja proizašao je iz pokušaja da se modelira način rješavanja problema od strane čovjeka-eksperta. Ideja je bila da, ako se razmišljanje eksperta može shvatiti i konceptualizovati u ekspertnom sistemu, onda bi trebalo da je lako da se dizajniraju takvi sistemi i da se oni koriste da korektno izvode slične zadatke. Ekspertiza se prvenstveno posmatra preko pojma IF-THEN pravila.

Postoje dvije glavne škole mišljenja u dizajnu ekspertnih sistema. Prva pokušava da prouči čovjeka eksperta preko intervjua i evidentiranja aktivnosti rješavanja problema, što je poznato kao analiza protokola. Ovo rezultira skupom pravila koja reflektuju aktuelno znanje eksperta u rješavanju problema. Drugi pristup usvaja formalniju analizu zadatka i proizvodi odgovarajuće drvo ili graf da karakterizuje moguća stanja u rješavanju problema. Prvi pristup stoga bolje odgovara aktuelnom ljudskom iskustvu, ali nije neophodno pouzdan, dok bi drugi morao biti pouzdan, ali možda ne odgovara najbolje načinu na koji ljudi rješavaju probleme.

Rule-based sistemi često se koriste u razvoju sistema kod kojih na početku rada nije jasno kako se tačno problem rješava, ali se detaljnom analizom ponašanja eksperata u rješavanju problema i sa što je moguće više testiranja može otkriti skrivena logika u ekspertskom ponašanju. To je zato što su rule-based sistemi odlični za brzu izradu prototipa.

Nedavno su neki autori (Schank i drugi) iznijeli mišljenje da eksperti ne rješavaju probleme koristeći pravila već slučajeve! Schank ovo prilično ubjedljivo argumentuje u zaključivanju u medicini ili pravu. Ovo je dovelo od razvoja Case Based Reasoning systems (na slučajevima zasnovani sistemi zaključivanja).

Glavna prednost rule-based programiranja je u tomo što se često veoma brzo mogu napraviti prototipi. Neki rule-based jezicu su i laki za učenje.

Nedostaci rule-based sistema ogledaju se prvenstveno u sporosti izvršavanja i teškoći dibagovanja. Napredniji rule-based sistemi imaju sposobnost da automatski generišu proceduralni kod (recimo u C-u), koji ima istu funkciju kao i ekspertni sistem. Ovo često rješava problem brzine izvršavanja.

14.6 Upitni jezici baza podataka

Upitni jezici baza podataka nastali su kao sredstvo za komunikaciju sa sistemom za upravljanje bazama podataka (SUBP), tj. za postavljanje *upita*.

Oni su suštinski povezani sa modelom podataka koji podržava konkretni sistem. Stoga imamo relacione upitne jezike, koji se koriste kod relacionih SUBP, objektno-orijentisane upitne jezike, koji se koriste kod SUBP sa objektno-orijentisanim modelom, itd.

Najpopularniji upitni jezici, kakav je recimo SQL (Structured Query Language, strukturirani upitni jezik), imaju određene proceduralne i neproceduralne karakteristike. Vremenom, kako se ne mogu svi zahtjevi prema bazi ostvariti kroz upit, ovi su jezici postali sve složeniji. Tako danas većina njih predstavlja kombinaciju paskalolikog proceduralog jezika i upitnog jezika u užem smislu. Dakle, to su jezici u kojima je moguće pisati programske cjeline kombinujući obraćanja bazi podataka sa deklarisanjem promjenljivih, petljama, uslovima, itd.

Na primjer, razmotrimo jednu proceduru na jeziku T-SQL, koji je upitni jezik za SUBP SQL Server firme Microsoft. Napomenimo da se radi o relacionom SUBP, u kome se podaci predstavljaju u vidu *relacija* ili *tabela*.

```
create procedure info_o_autoru(
@prezime varchar(30),@ime varchar(18),@dugih_naslova int output)
//procedura ima dva ulazna i jedan izlazni parametar
```



```
as
declare @duzina as int                  //cjelobrojna promjenljiva
declare @pom_naslov as varchar(50) //tip varchar(50) znači
                                        // tekst do 50 znakova
//kursor je promjenljiva koja omogućuje da
//prolazimo kroz tabelu red po red
declare kursor_naslovi cursor for
   select naslov                    //upit koji nalazi
   from autor inner join knjiga     //naslove našeg autora
     on autor.a_id = knjiga.au_id
     where a_prez= @prezime
     and a_ime= @ime
set @dugih_naslova=0                //inicijalizacija
open kursor_naslovi                 //otvori kursor
fetch next from kursor_naslovi      //dohvati prvi red
  into @pom_naslov                  //u promjenljivu @pom_naslov
while @@fetch_status = 0            //da li smo stigli do kraja
begin
 set @duzina=length(@pom_naslov)    //nadji dužinu naslova
 if (@duzina>40)                    //ako je naslov dug
  @dugih_naslova=@dugih_naslova+1 //povećaj broj
 fetch next from kursor_naslovi     //dohvati sljedeći red
   into @pom_naslov
end
close kursor_naslovi              //zatvori kursor
deallocate kursor_naslovi         //oslobodi memoriju
```

Ova procedura vraća kroz izlaznu promjenljivu @dugih_naslova broj dugih naslova knjiga autora čije su ime i prezime dati kroz dva ulazna paramatra.

Proceduru sada možemo pozvati sa

```
declare @d_nas as integer
exec info_o_autoru('Marko', 'Marković', @d_nas output)
print 'Marko Marković ima ' +cast(@d_nas as varchar)
                + ' knjiga sa dugim naslovima'
```

a kao rezultat dobijamo poruku

```
Marko Marković ima 8 dugih naslova
```

Neki upitni jezici idu i dalje. PL/SQL je upitni jezik za Oracle SUBP. Njegovo ime je skraćenica od "Procedural Language extensions to SQL," tj. jezik koji je proceduralno proširenje upitnog jezika SQL.

Kao i kod T-SQL-a, u PL/SQL-u se mogu pisati programi koji izgledaju kao tradicionalne procedure, kao i oni koji uključuju SQL upite, manipulišu podacima kroz kursore. Osim toga, PL/SQL podržava pakete (package) koji omogućuju grupisanje procedura i promjenljivih i time dozvoljavaju objektno-orijentisani dizajn softvera. On ima dobre mehanizme za praćenje izvršavanja programa i obradu grešaka.

Važna prednost PL/SQL je u tome što je Oracle razvio poseban razvojni alat – Oracle Forms – za kreiranje klijentskih aplikacija koje rade sa SUBP, koji je jako integrisan sa PL/SQL-om. Tu se pojavljuje kombinacija upitnih jezika baza podataka i vizuelnog programiranja.

Glavni kvalitet upitnih jezika baza podataka su lakoća upotrebe i činjenica da sadrže i proceduralne i logičke elemente, tj. da imaju mogućnost proceduralnog rada sa podacima koji su predstavljeni u modelu podataka (recimo, relacionom) koji ne odgovara najbolje proceduralnom načinu obrade.

Nedostaci su brzina izvršavanja i u nekim slučajevima ograničene mogućnosti obrade. Naravno, ovi su jezici limitirani na aplikacije koje uključuju obraćanje bazama podataka.

14.7 Vizuelno programiranje

Postoje dvije različite vrste vizuelnog programiranja.

Prva i češća vrsta je u suštini vizuelno programsko okruženje kod koga se koristi neka druga paradigma programiranja. Takvi su na primjer, Visual BASIC i Visual C. Kod ove vrste programiranja okruženje daje dobru mogućnost dizajniranja vizuelnih elemenata programa (recimo, prozora, dugmadi, boksova za unos teksta. itd.), na koje se zatim vezuje programski kod tipično sa elementima objektno-orijentisanog i proceduralnog programiranja. Tako, recimo, kad korisnik pritisne taster miša dok je pokazivač iznad nekog dugmeta, izvodi se procedura koja se, na primjer, zove dugme_click.

Druga vrsta vizuelnog programiranja vidi konstrukciju programa kao vizuelni zadatak. Kod ovog tipa programiranja korisnik manipuliše elementima programa u grafičkom intefejsu.

14.8 Script based programiranje

Script based programiranje koristi direktnu reprezentaciju korisničkih akcija u vidu scenarija. Zato je ovdje programski jezik dizajniran znatno ekspresivnije nego u slučaju drugih programskih jezika. Štaviše, program se konstruiše tako da direktno odgovara operacijama koje bi korisnik izveo. Stoga programi se često konstruišu tako što se snima niz operacija koje izvodi korisnik. Takva snimanja su obično bukvalna i ne sadrže nikakve pokušaje generalizacije (za razliku od programiranja demonstracijom).

Primjeri uključuju HyperTalk, jezik scenarija za Hypercard, i njegove nasljednike u drugim sistemima.

14.9 Programiranje demonstracijom

Programiranje demonstracijom (programming by demonstration, PBD) je jedan od skorijih pristupa porgramiranju i u značajnoj mjeri se koristi u istraživačkim laboratorijama i u ograničenim primjenama, kao što je crtanje. Suštinski pojam je da korisnik demonstrira program sistemu radeći jedan ili više primjera. Nakon toga sistem generalizuje niz korisničkih akcija tako što pravi opšti program. Jedan od poznatijih takvih sistema je Cypher-ov EAGER sistem koji može automatski detektovati ponavljanja u korisnikovim akcijama i napraviti program koji ih izvodi u budućnosti.

14.10 Zaključak

Svi programski jezici mogu implementirati Tjuringovu mašinu. Kao rezultat Čerč-Tjuringove teze bilo koji problem koji može biti riješen korišćenjem jednog programskog jezika, može takođe biti riješen korišćenjem drugog. Ipak, postojaće ogromne razlike u veličinama tih programa, vremenu potrebnom za razvoj, njihovoj pouzdanosti i vremenu izvršavanja. Tekst programa koji izvodi diferencni račun može biti kratak u LISP-u ili PROLOG-u,

ali značajno složeniji i duži u drugim jezicima. Slično, nije naročito dobro razvijati programe za rad sa bazama podataka u, recimo, LISP-u, mada ako sistemu treba neka forma zaključivanja ponekad se oni implementiraju u PROLOG-u.

Istorija računarstva dijelom je istorija paradigmi programiranja. Proceduralno programiranje je bilo široko zastupljeno, ali, kako se traži sve veća produktivnost, pouzdanost programa, bolje mogućnosti komunikacije - danas smo svjedoci da sve više ustupa mjesto objektno-orijentisanoj paradigmi, integrisanoj sa vizuelnim programskim okruženjima. Takođe, svjedoci smo da specijalizovani sistemi sa naprednim okruženjima omogućuju efikasno i brzo programiranje u nekim oblastima.

Neizbježno, paradigme koje su lakše za upotrebu, pouzdanije i produktivnije biće više korišćene. Takođe, nema sumnje da će u budućnosti doći do razvoja novih paradigmi. Postoje i sada optimisti koji kažu da će Xerox-ov san o krajnjem korisniku koji sam programira biti realizovan.

U oblasti kao računarstvo, koja se brzo mijenja, razumijevanje paradigmi programiranja može značajno biti od koristi programeru u smislu pravilnog pristupanja problemu, ali i u lakšem adaptiranju na nove programske alate.

14.11 Pitanja i zadaci

1. Šta je to paradigma programiranja?

2. Navedite koje paradigme programiranja znate, i koje su njihove glavne karakteristike.

3. Koji su glavni koncepti u objektno-orijentisanoj paradigmi?

4. Koje alate za vizuelno programiranje znate?

5. Kojoj paradigmi (ili paradigmama) pripada vaš omiljeni programski sistem?

6. Napišite po jedan program u svakoj paradigmi programiranja koju poznajete.

Literatura

[1] William Stallings: Computer Organization and Architecture, Designing and Practice, Fifth Ed., Prentice–Hall, 2003.

[2] M. Mano: Computer system architecture, Prentice–Hall, Englewood Cliffs, NJ, 1982.

[3] J. F. Wakerly: Digital Design, Principles and Practice, 3rd ed., Prentice–Hall, NJ, 1982.

[4] D. Živković, M. Popović: Impulsna i digitalna elektronika, Elektrotehnički fakultet, Beograd i Nauka, Beograd, 1995.

[5] С. В. Яблонский: Введение в дискретную математику, Наука, Москва, 1986.

[6] Г. П. Гаврилов, А. А. Сапоженко: Сборник задач по дискретной математике, Наука, Москва, 1977.

[7] D. Tošić: Računarski sistemi i elektronska obrada podataka, za *IV* razred srednjeg obrazovanja i vaspitanja, Naučna knjiga, Beograd i Zavod za izdavanje udžbenika, Novi Sad, 1990.

[8] D. Tošić: Mikroprocesori sa elementima programiranja, za *IV* razred elektrotehničke škole, Zavod za udžbenike i nastavna sredstva, Beograd, 1995.

[9] Arthur W. Burks, Herman H. Goldstine, John von Neumann: Preliminary discussion of the logical design of an electronic computing instrument, the report to U. S. Army Ordnance Department, 1946. Može se naći i na Internetu, na adresi

`research.microsoft.com/users/gbell/`

Computer_Structures__Readings_and_Examples/00000112.htm

ili

www.eecs.harvard.edu/~jonathan/neumann/neumann.html

[10] PDP–5 Handbook, Programmed Data Processor–5 Handbook, published by DEC, Digital Equipment Corporation, Maynard, MA, 1964. Na Internetu:

www.bitsavers.org/pdf/dec/pdp5

[11] Dario Alejandro Alpern: Microprocesadores de la linea Intel, El micrroprocesador 80286, El microprocesador Pentium. Sa adrese

www.alpertron.com.ar/intel.htm

[12] Fran Golden: Debug/Assembly Tutorial, Chapter 1 – Chapter 11. Na Internetu:

www.datainstitute.com/debug1.htm

ili

www.armory.com/~rstevew/Public/Tutor/Debug

[13] Miodrag Milanović: Priručnik za asemblersko programiranje na procesorima iz familije Intel 80x86, Novi Sad, 2000.

[14] Peter Norvig: Paradigms of AI Programming: Case Studies in Common Lisp, Morgan Kaufmann, 1992.

[15] Eugene Charniak, Christopher K. Riesbeck, Drew V. McDermott and James R. Meehan: Artificial Intelligence Programming, 2nd edition. Lawrence Erlbaum Associates (Hillsdale, NJ), 1987.

[16] John C. Martin: Introduction to Languages and Theory of Computation, 2nd edition. McGraw-Hill, 1997.

[17] Guy L. Steele: Common Lisp: The Language, 2nd Edition, Digital Press, 1990.

[18] Peter Jackson: Introduction to Expert Systems, Addison-Wesley, 1999.

CIP - Каталогизација у публикацији
Централна народна библиотека Црне Горе, Цетиње

004 (075.8)
004.42 (075.8)

МАРТИНОВИЋ, Милан
Računari i principi programiranja / Milan Martinović, Predrag Stanišić. - 2. izd. - Podgorica : Univerzitet Crne Gore, 2009 (Podgorica : Pobjeda). - XIV, 405 str. : tabele ; 25 cm. - (Biblioteka Prirodno-matematičkih nauka

Tiraž 400. - Bibliografija: str. 405-406

ISBN 978-86-7664-071-3
1. Станишић, Предраг [аутор]
a) Рачунари - Уџбеници b) Програмирање - Уџбеници
COBISS. CG-ID 141843664

БИБЛИОТЕКА
ТЕХНИЧКИХ ФАКУЛТЕТА
ТИТОГРАД