VERA V.PETROVIĆ
SLOBODAN OBRADOVIĆ
MILAN MIJALKOVIĆ

OSNOVI INFORMATIKE I RAČUNARSTVA

Visoka škola elektrotehnike i računarstva strukovnih studija,
Beograd, 2009.

Autori:	*dr Vera V.Petrović* *dr Slobodan Obradović* *mr Milan Mijalković*
Recezenti:	*dr Zoran Banjac*
Izdavač:	*Visoka škola elektrotehnike i računarstva strukovnih studija u Beogradu*
Tehnička obrada:	*dr Vera V.Petrović* *Gabrijela Dimić* *Kristijan Kuk*
Korice:	*Gabrijela Dimić* *Kristijan Kuk*
Tiraž:	200
Štampa:	*Akademska izdanja, Beograd*

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

004 (075.8)

ПЕТРОВИЋ, ВЕРА В. , 1965 –
Osnovi informatike i računarstva / Vera V.Petrović, Slobodan Obradović, Milan Mijalković. – Beograd : Visoka škola elektrotehnike i računarstva strukovnih studija, 2009 (Beograd : Akademska izdanja).
- II, 196 str. : ilustr. ; 30 cm

Tiraž 200. – Bibliografija: str. 195-196.

ISBN 978-86-7982-038-9
1. Обрадовић, Слободан, 1955- [аутор] 2. Мијалковић, Милан, 1957- [аутор]
а) Рачунарство
COBBISS.SR-ID 157165068

PREDGOVOR

Udžbenik, pod nazivom ***OSNOVI INFORMATIKE I RAČUNARSTVA***, namenjen je za predmet Osnovi informatike i računarstva koji se predaje na studijskim programima Elektronsko poslovanje, Nove energetske tehnologije i Automatika i sistemi upravljanja vozilima prve godine Visoke škole elektrotehnike i računarstva strukovnih studija u Beogradu.

U uvodnoj glavi dat je kratak pregled razvoja računarske tehnike i računara i tehnologija za izradu elektronskih uređaja.

U prvoj glavi dat je uvod u računarske sisteme kroz kratak opis rada i sastava računarskih sistema koje čine: razni tehnički uređaji (hardver), programi (softver), podaci, procedure i ljudi. Hardver je tako konstruisan da pod kontrolom korisničkih i sistemskih programa obavlja: prihvatanje, smeštanje, čuvanje i ponovni pristup podacima, obradu podataka i saopštavanje dobijenih rezultata. Za svaku od ovih operacija u računarskom sistemu zaduženi su posebni uređaji, mada neki uređaji mogu obavljati i više pomenutih funkcija. Prihvatanje podataka obavljaju ulazne jedinice, smeštanje i čuvanje podataka obavljaju razne vrste memorija, obradu podataka obavlja centralna jedinica (računar), a prikazivanje rezultata obavlja se posredstvom izlaznih jedinica. Informacije, tj. podaci i instrukcije, u računaru su smeštene i obrađuju se u obliku binarnih brojeva.

U drugoj glavi obrađene su matematičke osnove računara kroz kratak opis raznih vrsta brojnih sistema, uz detaljan opis binarnog brojnog sistema i načina kodiranja numeričkih i nenumeričkih podataka.

U trećoj glavi, elektronske osnove računara - opisane su osnovne logičke operacije i logička kola koja čine osnovu funkcionisanja računara. U nastavku su opisana osnovna aritmetička kola i osnovna memorijska kola: flip-flop, registri i razne vrste memorija, kao i na koje se ova kola među sobom povezuju (magistrale). Mašinske instrukcije se izvode u dve faze: pribavljanje i izvršavanje. Izvršavanje pojedinačnih instrukcija i programa u računaru obavlja se automatski, bez intervencije čoveka. Automatsko izvršenje programa omogućeno je na taj način što procesor u svakom trenutku, dok izvršava jednu naredbu, zna koja je sledeća naredba koju treba izvršiti. To se postiže uz pomoć jednog registra u procesoru koji se zove programski brojač. Da bi se otpočelo izvršavanje programa treba samo saopštiti procesoru koja je prva naredba programa koju treba da izvrši.

U četvrtoj glavi, arhitektura računara, pored opisa faza izvođenja mašinskih instrukcija, dati su opis raznih načina realizacije upravljačkih jedinica i raznih načina obrade podataka, kao što su paralelna obrada, multiprogramiranje i

multiprocesiranje. Šta će se i u kom trenutku raditi u računaru, i sa kojim podacima, određuje niz instrukcija (koje odgovaraju nekim elementarnim obradama), a koje nazivamo programom.

I u petoj glavi, pod naslovom Vrste naredbi i načini adresiranja, opisane su razne vrste naredbi i razni načini adresiranja koji se koriste u različitim računarskim sistemima, sa posebnim osvrtom na mikroračunare, s obzirom na njihovu veliku rasprostranjenost i dostupnost.

U Beogradu, mart 2009. godine Autori

SADRŽAJ

UVOD – ISTORIJA RAČUNARA 1

1. UVOD U RAČUNARSKE SISTEME 27

1.1. Opšti model računarskog sistema 28
1.2. Funkcionalna blok-šema računara 30
1.3. Hijerarhijski model računarskog sistema 34
1.4. Zaključak 37
1.5. Pitanja 38
1.6. Ključne reči 39

2. MATEMATIČKE OSNOVE RAČUNARA 40

2.1. Brojni sistemi 40
2.2. Konverzija brojeva iz jednog brojnog sistema u drugi 43
2.3. Pojam komplementa 46
2.4. Binarni brojni sistem 49
2.5. Brojevi sa znakom, kodiranje negativnih brojeva 50
2.6. Predstavljanje negativnih brojeva drugim tehnikama 57
2.7. Brojevi sa nepokretnom tačkom 62
2.8. Brojevi sa pokretnom tačkom 70
2.9. Binarno kodirani dekadni brojevi 73
2.10. Kodiranje nenumeričkih podataka 77
2.11. Zaključak 81
2.12. Pitanja 83
2.13. Ključne reči 84

3. ELEKTRONSKE OSNOVE RAČUNARA 85

3.1. Digitalna logika 85
3.2. Bulova algebra 86
3.3. Elementarna logička kola 90
3.4. Minimizacija logičkih funkcija 94
3.5. Elementarna memorijska kola 101
3.6. Elementarna artimetička kola 104
3.7. Registri 107
3.8. Magistrale 109
3.9. Memorije 115
3.10. Zaključak 131
3.11.Pitanja 132
3.12.Ključne reči 133

4. ARHITEKTURA RAČUNARA 135

4.1. Pojednostavljena arhitektura računara 135

4.2. Instrukcije u mašinskom jeziku 146
4.3. Faza pribavljanja mašinskih instrukcija 149
4.4. Faza izvršavanja mašinskih instrukcija 154
4.5. Mikroprogramska organizacija upravljačkih jedinica 164
4.6. Hardverska organizacija upravljačkih jedinica 169
4.7. Arhitektura tekuće trake i paralelna obrada 169
4.8. Multiprocesorski sistemi 172
4.9. Zaključak 174
4.10. Pitanja 175
4.11. Ključne reči 176
5. VRSTE NAREDBI I NAČINI ADRESIRANJA 177
5.1. Registri 178
5.2. Artimetičko logička – jedinica 184
5.3. Tipovi mašinskih instrukcija 185
5.4. Binarno logičke instrukcije 190
5.5. Artimetičke binarne instrukcije 193
LITERATURA 195

UVOD - ISTORIJA RAČUNARA

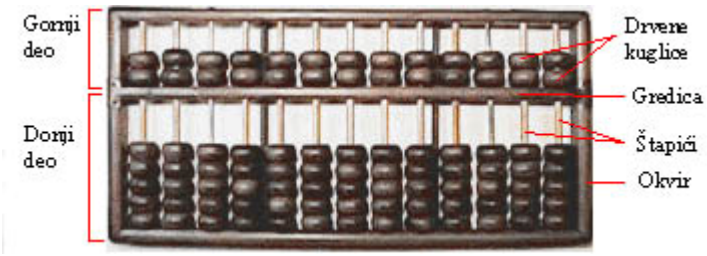
Uvod

Uticaj revolucije informacija na naše društvo i industriju je ogroman. U sve većoj želji da kontrolišemo sopstvenu sudbinu mi ne želimo samo da shvatimo trenutnu tehnologiju nego i da provirimo u prošlost kako bismo prepoznali trendove koji nam mogu omogućiti da predvidimo neke elemente budućnosti. Gledanje u nazad da bi se otkrile paralele i analogije sa modernom tehnologijom može obezbediti osnove za razvoj standarda po kojima možemo proceniti izvodljivost i potencijal tekuće ili predložene aktivnosti. Ali mi takođe imamo osećaj odgovornosti za očuvanje dostignuća naših prethodnika kroz uspostavljanje arhiva i muzeja uz očekivanje da će zadovoljstvo otkrivanja prevazići profitabilnost puke istorijske apstrakcije.

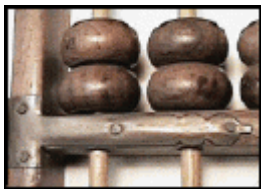
Rane godine

U ranim danima kalkulacija je bila potrebna onda kada je trebalo izvestiti o individualnim ili grupnim postupcima, pogotovo u vezi sa održanjem inventara (grupe ovaca) ili usklađivanjem finansija. Ranije su ljudi računali poređenjem jednog skupa objekata sa drugim (kamenje i ovce). Operacije dodavanja i oduzimanja bile su jednostavno operacije dodavanja ili oduzimanja grupa objekata na gomilu kamenja ili šljunka koja je služila za računanje. Prve table za računanje zvale su se **abaci** i nisu samo stvorile ovaj metod računanja nego su i uvele koncept pozicionog označavanja, koji i danas koristimo. Sledeći logični korak bilo je pravljenje prvog "personalnog kalkulatora" - abakus (engl. abacus) -- koji koristi isti koncept, da jedan skup objekta zamenjuje drugi skup, ali i koncept da jedan objekat zamenjuje kolekciju objekata -- poziciono označavanje. Ovaj odnos jedan prema jedan nastavio se kroz mnoga stoleća, čak i kada su prvi kalkulatori koristili poziciju rupe na krugu da bi označili broj -- kao kod telefona koji ima kružni brojčanik. Iako su ove mašine često imale simbol broja ugraviran pored rupa za biranje, korisnik nije morao da zna vezu između simbola i njihovih numeričkih vrednosti.

Proces računanja postao je proces manipulacije simbolima tek kada je proces računanja i aritmetike postao apstraktniji, pa su različitim grupama dodeljeni simbolički prikazi, a rezultati su se mogli zapisati na "medijumu za skladištenje", kao što su bili papirus ili glina.
Delovi računara (uključujući i softver) prikupljali su se kroz mnoga stoleća, a veliki broj ljudi pridodao je ponešto. Jedan od onih koji godinama nisu bili prepoznati jeste **Muhammad ibn Musa Al'Khowarizmi**, taškentski sveštenik koji je u dvanaestom veku razvio koncept pisanog procesa koji treba slediti da bi se postigao neki cilj i objavio knjigu koja je tom konceptu dala njegovo moderno ime - algoritam.
Abakus je mehaničko pomagalo koje se koristi za računanje. Na standardnom abakusu može se sabirati, oduzimati, deliti i množiti.



Konstrukcija i anatomija



Detalj konstrukcije - Dve drvene kuglice u gornjem delu, štapići i gredica.

Abakus je obično sastavljen od različitih vrsta tvrdog drveta i može biti različitih dimenzija. Njegov okvir ima niz vertikalnih *štapića* po kojima brojne *drvene kuglice* mogu slobodno da klize. Horizontalna *gredica* deli okvir na dva dela, *gornji* i *donji*.

Osnove

Računanje se obavlja postavljanjem abakusa položeno na sto ili u krilo i premeštanjem drvenih kuglica prstima jedne ruke. Svaka drvena kuglica na gornjem delu ima vrednost **5**; svaka kuglica u donjem delu ima vrednost **1**. Smatra

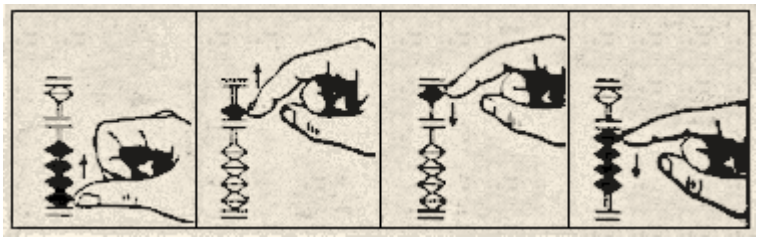
se da su kuglice uračunate kad su pomerene prema gredici koja razdvaja dva dela. Krajnja desna kolona je kolona jedinica; sledeća kolona na levo je kolona desetica, pa zatim kolona stotina itd. Nakon što je u donjem delu uračunato 5 kuglica rezultat se "prenosi" na gornji deo; nakon što su obe kuglice u gornjem delu uračunate, rezultat (10) prenosi se na sledeću kolonu sleva. Računanja sa pokretnim zarezom vrše se tako što se obeleži mesto između dve kolone kao decimalni zarez, pa svi redovi zdesna predstavljaju decimale, a svi redovi sleva cele brojeve.

Evolucija suan-pana

Na svakom štapiću klasični kineski abakus (suan-pan) ima dve kuglice u gornjem delu, a pet kuglica u donjem; takav abakus se naziva i abakus 2/5. Stil 2/5 nije se menjao do 1850. g., kada se pojavio abakus 1/5 (jedna kuglica u gornjem, a pet kuglica u donjem delu). Tridesetih se pojavio abakus 1/4 (soroban), stil koji danas ima prvenstvo, a proizvodi se u Japanu. Modeli 1/5 danas su uopšte retki, a modeli 2/5 retki su izvan Kine (osim u kineskim zajednicama u severnoj Americi i drugde). Uporedite kineski, japanski i astečki abakus (3 apleta).

Tehnika

Za postizanju veštine na abakusu važan je pravilan rad prstiju. Na kineskom abakusu se za manipulisanje kuglicama koriste palac, kažiprst i srednji prst.Kuglice u donjem delu pomeraju se *nagore* palcem, a *nadole* kažiprstom. U nekim kalkulacijama koristi se srednji prst za pomeranje kuglica u gornjem delu.



Tehnika prstiju

Japanski priručnik objavljen 1954. g. pokazuje pravilan rad prstiju za pomeranje kuglica.U japanskoj verziji koriste se samo kažiprst i palac. Kuglice se pomeraju *nagore* palcem, a *nadole* kažiprstom. Međutim, određene kompleksne operacije zahtevaju da kažiprst pomera kuglice nagore; primer je dodavanje 3 na 8 (dodavanje broja tri se naziva *Jian Chi Jia Shi* što doslovno znači oduzimanje 7 dodavanje 10). Ova Java verzija abakusa predstavlja ograničenu simulaciju pravog uređaja, jer je tehnika prstiju potpuno zamenjena mišem. Za postizanje efikasnosti i brzine računanja na pravom abakusu neophodna je stalna praksa.

Današnji abakus

Abakus i danas koriste vlasnici radnji u Aziji i "kineskim četvrtima" u severnoj Americi. Njegova upotreba se i dalje uči u azijskim školama, što na zapadu, na žalost, nije slučaj. Jedna od praktičnih upotreba abakusa jeste da deca nauče jednostavnu matematiku, a posebno množenje; abakus je odlična zamena za učenje tablice množenja na pamet, što deci predstavlja posebno težak zadatak. Abakus je odlična alatka za učenje drugih osnova numeričkih sistema, jer se lako prilagođava bilo kojoj osnovi. Slepa deca uče da koriste abakus tamo gde bi deca sa normalnim vidom koristila papir i olovku za računanje

1854

George Boole opisuje svoj sistem za simboličko i logičko rasuđivanje koji kasnije postaje osnova za kompjuterski dizajn.

1884

Osnovan je Američki Institut za elektrotehniku (AIEE); prva od organizacija koje će se vremenom objediniti u IEEE 1963. godine.

1890

Rastuća populacija u Americi i zahtevi kongresa da se u svakom popisu postavlja više pitanja doveli su do toga da obrada podataka postaje sve duži proces. Procenjeno je da se podaci popisa 1890 neće obraditi pre popisa 1900 ukoliko se nešto ne učini na poboljšanju metodologije obrade. Herman Hollerith je pobedio na takmičenju za isporuku opreme za obradu podataka koja bi pomogla u obradi podataka američkog popisa 1890 i dalje je asisistirao u obradi popisa u mnogim zemljama širom sveta. Kompanija koju je osnovao, Hollerith Tabulating Company, je vremenom postala jedna od tri kompanije koje su činile kompaniju Calculating-Tabulating-Recording (C-T-R) 1914, preimenovana u IBM 1924. Hollerith mašine su se prve pojavile na naslovnoj strani magazina.

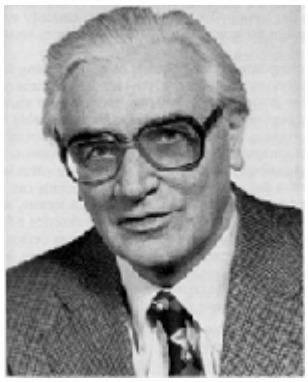
1912

Osnovan je Institut radio inženjera - druga od organizacija koje su se vremenom spojile u IEEE 1963. godine.

1925

Babbage i Hollerith metodi digitalnog računarstva su se retko koristili u naučnim proračunima iako su analogni uređaji kao što je logaritmar bili u širokoj upotrebi naročito u tehničkim proračunima. Vannevar Bush, MIT, je napravio veliki diferencijalni analizator sa dodatnim sposobnostima integracije i diferencijacije. Diferencijalni analizator kojeg je utemeljila Rockfeller fondacija je verovatno bio najveći uređaj za računanje na svetu 1930. godine. Digitalno računanje je ponovo došlo do izražaja 1930. godine kad su brojni naučnici uvideli da je tehnologija došla do stadijuma da su dostupne sve neophodne komponente računara. Svaki je u svom domenu trebalo da izumi (ili možda "ponovo izumi" nesvestan prethodnog rada Babbage-a) strukturu računara. Mada sada možemo da odredimo precizne datume kada je barem četiri ponira prepoznalo kapacitete tehnologije, kad prođe još sto godina našim potomcima će ovo izgledati kao jedan trenutak u vremenu kada su istovremeno nezavisni istraživači napravili računar.

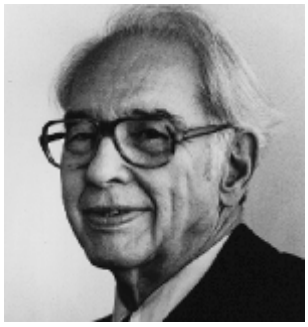
1935 - 1938



Konrad Zuso u Berlinu, u Nemačkoj razvio je svoj računar Z-1 u dnevnoj sobi svojih roditelja, relejni računar koji je koristio binarnu aritmetiku. 1938 je nastavio sa Z-2 uz pomoć Helmuta Schreyer-a. Tokom drugog svetskog rata se obratio nemačkoj vladi za pomoć u građenju mašina ali je odbijen jer bi duže trajalo da se njegov rad završi no što je vlada očekivala da će rat trajati. Pri kraju rata je pobegao u Hinterstein pa potom u Švajcarsku gde je rekonstruisao mašinu Z-4 na univerzitetu u Cirihu i osnovao računarsku kompaniju koja se vremenom pripojila koorporaciji Siemens.

Nedavno je nemački muzej u Minhenu rekonstruisao mašinu Z-1 kao centralni deo računarske izložbe. Zuse-ova mašina je sve do posle rata bila nepoznata izvan Nemačke i mada imaju hronološki prioritet ipak imaju mali uticaj na sveukupni razvoj industrije.

1936 - 1939



John Vincent Atanasoff je sa John Berry-em razvio mašinu koju sada nazivamo ABC -- Atanasoff-Berry Computer -- na univerzitetu u Iowi, Amerika kao mašinu posebne namene za rešavanje skupova linearnih jednačina u fizici. Verovatno najraniji primer elektronskog kalkulatora, ABC je razvio osnovne koncepte koji će se pojaviti kasnije u "modernim računarima" -- elektronsku aritmetičku jedinicu i regenerativnu, cikličnu memoriju.

1937

Mada nije koristio praktičnu tehnologiju epohe, **Alan Turing** je razvio ideju "univerzalne mašine" koja može da izvrši svaki algoritam koji se može opisati i koja predstavlja osnovu za koncept "računarstva". Verovatno je važnije to što su se njegove ideje razlikovale od ideja ljudi koji su rešavali aritmetičke probleme i što je uveo koncept "simboličke obrade".

Stibitz

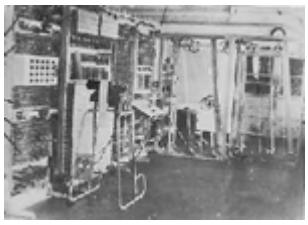
U Americi je još dvoje drugih ljudi razmatralo problem računanja: Howard Aiken na univerzitetu Harvard čiji je rad urodio plodom 1944. godine i **George Stibitz** u Bell Telephone Laboratories koji je proučavao korišćenje telefonskih releja u aritmetici. On je prvi konstruisao aritmetičku jedinicu na relejni pogon 1937. godine (koju je kasnije nazvao Model-K pošto je napravljena na Kitchen stolu) i nakon tako skromnog početka napravio brojne relejne mašine koje su se koristile tokom drugog svetskog rata.

1939

Jedan od glavnih problema u računarstvu u Bell Telephone Laboratories je bio domen kompleksnih brojeva. Stibitz-ov prvi potpun elektromagnetski relejni kalkulator rešio je problem i dato mu je ime Complex Number Calculator (kasnije Bell Labs Model 1). Godinu dana kasnije ova mašina je prva bila upotrebljena daljinski preko telefonskih linija pripremajući teren za povezivanje računara i sisteme komunikacije, deljenje vremena, i kasnije umrežavanje. U hodniku ispred sala za konferencije na godišnjoj konferenciji američkog matematičkog društva na koledžu Dartmouth instaliran je teleprinter i povezan sa Complex Number Calculator-om u New York-u. Među ljudima koji su iskoristili mogućnost i probali sistem bili su Norbert Wiener i John Mauchly

1940-1944

S one strane Atlantika glavna potreba za podršku ratnim naporima bilo je dešifrovanje uhvaćenih poruka nemačkih snaga. Za šifrovanje se u prvim godinama rata koristila ENIGMA, dizajnirana u SAD. Tim iz Bletchey Parka, koji se nalazi na pola puta između univerziteta Oxford i Cambridge, u kojem je bio i Alan Turing, izgradio je seriju mašina koje su dostigle vrhunac 1943. godine sa **Colossusom**.



Telephone Research Establisment pod komandom **Tommy Flowersa** (na slici sa desne strane, zajedno sa **Sir Harry Hinsleyem**, takođe rukovodiocem u Bletchey Parku, koji je nedavno objavio memoare) isporučio je decembra 1943. Colossus Mark I, a on je postao operativan 1944. Dešifrovanje poruka je pomoglo planiranje za dan D, kasnije te godine. Sledeće mašine isporučene su na vreme za iskrcavanje u Normandiji i igrale su značajnu ulogu u pobedi nad nacističkom Nemačkom. Postojanje Colossusa krilo se do 1970. godine, a algoritmi dešifrovanja su i dalje tajna. Turing i ostali su imali samo mali uticaj na razvoj britanskog računarstva nakon rata.

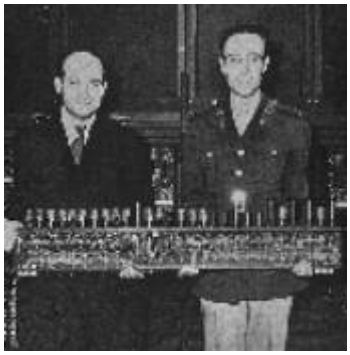
Flowers (sa desne strane)

Kopija Colossusa stoji sada u muzeju u Bletchey Parku u Engleskoj. U Americi je pokrenut sličan program u United States Naval Computing Machine Laboratory (USNCML) u Daytonu u državi Ohio, koji je koristio tehnologiju prenetu iz Bletchey Parka preuzet, a kasnije je nastavljeno u Wisconsin Avenue štabu National Security Agency (NSA). Pored pomoći u razbijanju nemačkih šifri, USNCML je radila i na japanskim šiframa. Nakon rata članovi ove grupe inženjera osnovali su društvo Electronic Research Associates (ERA) u Minneapolisu.

1943

Rad na ENIAC-u, započet 1943, vodio je John Brainer, dekan Moore School of Electronical Engineering na Univerzitetu Pennsylvania, zajedno sa **Johnom Mauchlyjem** i **J. Presperom Eckertom**, koji su bili zaduženi za implementaciju. Veza američke armije, u ime laboratorije Aberdeen Proving Ground (Ballistic Research Laboratory), bio je **Herman Goldstine**. Slike prikazuju Eckerta (levo) i Goldstinea (desno) kako drže aritmetički deo iz ENIAC-a.

Mauchly & Eckert



Eckert & Goldstine

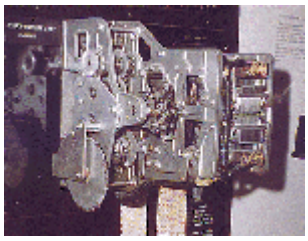
Drugi svetski rat

Potreba za računarstvom tokom Drugog svetskog rata postala je intenzivnija zbog iznenadnog naprednog razvoja brojnih artiljerijskih sredstava koja su trebala da se suprotstave sve boljim borbenim sredstvima kakav je avion. Stibitz je proširio svoje prenosive mašine uređajima za praćenje i za navođenje koji bi se priključili na protivavionske topove, ali je glavni nedostatak bio dostupnost "tabela za gađanje" za poljsku i mornaričku artiljeriju. Stoga su rani američki uređaji za računanje, kao što je Babbageova diferencijalna mašina, napravljeni da prave tabele, a ne da u realnom vremenu obavljaju računanje za rešavanje naučnih (ili vojnih) problema.

1944

Prvi veliki automatski elektromehanički kalkulator opšte namene je bio Harvard Mark I (tj. IBM Automatic Sequence Control Calculator [ASCC]). Njega je izumeo Howard Aiken krajem tridesetih godina, a implementirali su ga Hamilton, Lake i Durfee iz IBM-a. Mašina koju je sponzorisala američka mornarica trebalo je da izračunava elemente matematičkih tabela i tabela za navigaciju; istu svrhu je imala Babbageova diferencijalna mašina. Aiken je posvetio svoje rane izveštaje

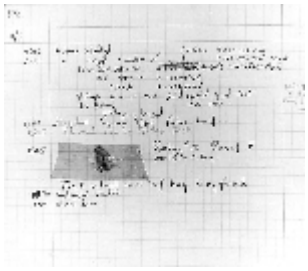
Babbageu kada je saznao za deo diferencijalne mašine na Harvardu 1937. ASCC nije mašina sa skladištenim programom, nego ju je pokretala papirna traka sa instrukcijama.



Grace Murray Hopper je radila za Aikena na Harvardu juna 1944. i postala je treći programer na računaru Mark I. Njena dva prethodnika, koje su tada nazivali "koderima", bili su zastavnici Robert Campbell i Richard Bloch.

1945

Grace Murray Hopper je radila u privremenoj zgradi iz Prvog svetskog rata na Univerzitetu Harvard. Na računaru Mark II ona je pronašla **prvu računarsku bubu** koja je nastradala od struje. Prilepila ju je na dnevnik računara i kad bi kasnije mašina stala (što se često dešavalo) rekli bi Howardu Aikenu da "uklanjaju bube" (engl. **debugging**) iz računara. Prva buba i dalje postoji u National Museum of American History of the Smithsonian Institutions. Reč buba i koncept uklanjanja ranije je verovatno koristio Edison, ali se pretpostavlja kako je ovo prva potvrda da se taj pojam primenjuje na računare.

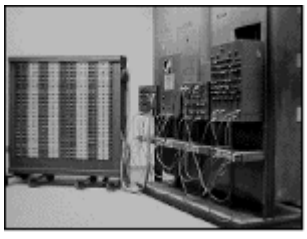


30. jun 1945: John von Neumann je napisao "Prvu skicu izveštaja o EDVAC-u" koja je utrla put za arhitektonski dizajn nekoliko generacija računara; izveštaj nikad nije prevazišao stadijum skice, ali njegovi koautori (očigledno ne i saradnici u pisanju) nisu nikad zvanično imenovani. Arhitektonski stil je postao poznat kao "von Neumannova arhitektura", a ovaj izvor za pojam "uskladištenog programa" postao je diskutabilan. Eckert i Mauchly su tvrdili da su oni razmišljali o tome pre no što se Neumann pridružio već započetom radu na univerzitetu Pennsylvania. Konrad Zuse je tvrdio kasnije da je i on razmišljao o tome još tridesetih godina

1946

ENIAC je otkriven u Filadelfiji. ENIAC je predstavljao tek korak prema pravim računarima, za razliku od Babbagea, Eckert i Mauchly su završili konstrukciju, iako su znali da mašina nije baš reprezentativna tehnologija. ENIAC je programiran kroz ponovo povezivanje međusobnih veza među različitim komponentama i imao je sposobnost paralelnog računanja. ENIAC je kasnije modifikovan u programsku mašinu za skladištenje, ali ne pre no što se tvrdilo da su druge mašine bile prvi računar.

1946 je bila godina u kojoj se desio prvi računarski sastanak, Univerzitet Pensilvanije je organizovao prvu seriju "letnjih sastanaka" gde su svetski naučnici saznali ponešto o ENIAC-u i planovima za EDVAC. Među učesnicima je bio Maurice Wilkes sa univerziteta u Kembridžu koji se vratio u Englesku da bi napravio EDSAC.

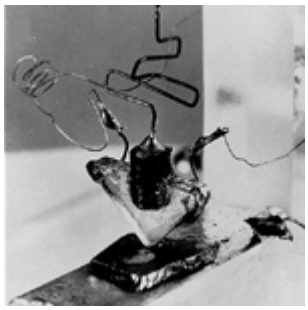


Kasnije te godine, Eckert i Mauchly zbog spora sa Univerzitetom u Pensilvaniji u vezi patenta napustili su univerzitet da bi uspostavili prvu računarsku kompaniju -- Electronic Control Corp. sa planom da izgrade Universal Automatic Computer (UNIVAC). Nakon mnogo kriza oni su izgradili BINAC za Northrup Aviation i preuzeti su od kompanije Remington-Rand pre no što je UNIVAC završen. U isto vreme je Electronic Research Associates (ERA) ujedinjen u Minneapolisu i preuzeo njihovo znanje o računarskim uređajima da bi napravili liniju računara; kasnije je ERA takođe spojena sa Remington-Rand.

Te iste godine je formairan AIEE komitet o računarskim uređajima velike skale čiji je predsedavajući bio Charles Concordia (maj/jun 1946-49); ovaj komitet je bio preteča IEEE Computer Society iz 1963 godine.

1947

William Shockley, John Bardeen i Walter Brattain su izumeli uređaj " prenosni otpornik ", kasnije poznat kao tranzistor koji će preinačiti računar i dati mu pouzdanost koja se nije mogla postići vakumskim cevima.



1948

Rad na programskom računaru za skladištenje se zbivao bar na četiri lokacije - na Univerzitetu u Pensilvaniji na konstrukciji EDSAC-a, na Princeton University na Institute for Advanced Study Machine (IAS) pod rukovodstvom John von Neumanna, na Cambridge University pod rukovodstvom Maurice Wilkesa i Univerzitetu u Manchesteru. Douglas Hartree je posetio razne lokacije u Americi i vratio se u Englesku da bi ubedio svoje kolege, Freddy Williamsa i Tom Kilburna da naprave računar. Max Newman, jedan od rukovodilaca istraživanja u Bletchey Parku je napravio Royal Society Computing Laboratory u Manchesteru i tražio je sredstva da bi napravio računar. 21. juna 1948 godine njihova prototip mašina "Baby" je radila po prvi put; svet se zaista pomakao iz domena kalkulatora u domen računara. Williams, Kilburn i Newman su nastavili da grade mašinu pune skale koju su nazvali **Manchester Mark I**. Ferranti Corporation je preuzeo dizajn i

započeo liniju računara koji su bili jedna od glavnih komponenti britanske računarske industrije.

T.J. Watson Sr. se naljutio na Howard Aikena zbog nedostatka posvećenosti Atomatic Sequence Control Calculatoru [ASCC] (Harvard Mark I) i iznerviran uspehom ENIAC-a naručio izgradnju **Selective Sequence Control Computer-a** (SSEC) za IBM. Iako to nije bio programerski računar za skladištenje, SSEC je bio prvi korak IBM-a od potpune posvećenosti tabulatorima sa bušenim karticama do sveta računara. Javne slike SSEC-a su modifikovane da ne bi sadržale kolone u mašinskoj sobi u IBM kancelarijama na Madison Avenue nakon što je Watson izrazio žaljenje što uopšte postoje!

1949

Samo godinu dana nakon što je Manchester Baby postala prva operativna programska mašina za skladištenje na svetu, prvi elektronski digitalni programski računar velike skale za skladištenje, kompletno funkcionalan, razvio je **Maurice Wilkes** sa osobljem matematičke laboratorije na Cambridge univerzitetu. Nazvan je EDSAC (Electronic Delay Storage Automatic Computer); primarni sistem za skladištenje je bio skup živinih korita (cevi ispunjene živom) kroz koje je generisano i regenerisano akustično pulsiranje predstavljalo bitove podataka. Wilkes je 1946 pohađao letnju školu na Univerzitetu u Pensilvanija i vratio se kući sa osnovnim planovima za mašinu u mislima.

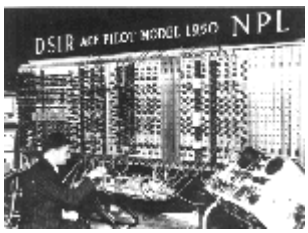
U Americi je NAtional Bureau of Standards počeo rad na dve mašine. Bureau je bio odgovoran za realizovnje ugovora o isporuci UNIVAC-a Census Bureau ali je spoznao da nema dovoljne resurse za njegov rad. Pošto nisu imali veliki budžet Bureau je odlučio da se takmiči sa National Physical Laboratory (ekvivalent u Engleskoj) pa je stvorio sopstvene mašine. One su postavljene u istočnim i zapadnim centrima. Sam Alexander je preuzeo kontrolu razvoja SEAC-a (Standards Eastern Automatic Computer) dok je Harry Huskey (koji je napravio Pilot ACE u National Physical Laboratory [NPL], britanski ekvivalent NBS-a) vodio razvoj SWAC-a (**Standards Western Automatic Computer**).

1950'te

Pedesetih godina prošlog veka Professional Group on Electronic Computers of the Institute of Radio Engineers (Profesionalna grupa za elektronske računare Instituta radio inženjera) je postala organizacija sa mnogim elementima današnje organizacije Computer Society, značajno prihvatajući tehničke i obrazovne komitete. Konferencije su bile najznačajnije rane aktivnosti ove grupacije, ali su publikacije rapidno rasle sa nekih 1 800 uvodnih stranica tokom dekade. Pri kraju pedesetih PGEC je bila najveća profesionalna grupa u IRE. Imala je 19 podružnica po Americi i 8 874 članova, uključujući 8 179 stalnih članova, 679 studenata i 66 pridruženih članova.

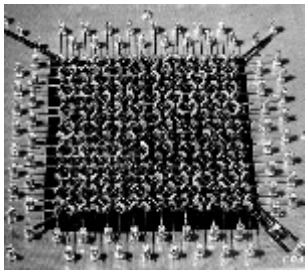
1950

Nakon drugog svetskog rata i svog rada u Betchley Parku, Alan Turing se pridružio upravi National Physical Laboratory u Teddingtonu, u Engleskoj, sa svojim planovima za pravljenje računara. Njegov projekat za **Automatic Computing Engine** (ACE) je završen 1947. godine, ali je direktor laboratorije dao zadatak konstrukcije fizičkom (Physics), a ne matematičkom (Mathematics) odeljenju gde se Turing nalazio. Usled toga je Turing napustio NPL da bi preuzeo mesto svog šefa u toku rata, Maxa Newmana na University of Manchester. Rad na prototipu mašine bazirane na Turingovim planovima je nazvan Pilot Ace, projekat je počeo Harry Huskey 1948. godine i završio ga 1951. godine. Potpuna verzija je završena nekoliko godina kasnije u Department of Scentific and Industrial Research.



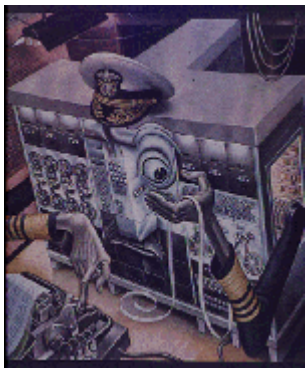
1951

Jay Forrester, Bob Everett i drugi u MIT su započeli rad na simulatoru za Air Force kasne 1946. godine, ali su se predomislili o upotrebi analognih tehnika i odlučili da koriste digitalnu obradu za prvi računar sa obradom u realnom vremenu - **Whirlwind**. Ovaj rad je takođe dobro poznat za razvoju operativne (engl. core) magnetne memorije. Osnovni koncept za operativnu memoriju je patentirao An Wang sa Harvard University 1949. godine, ali je njegova tehnika uključivala korišćenje jezgra na jednoj žici za formiranje linija kašnjenja. Projektom Whirlwind počelo je korišćenje tehnike feritnih jezgara povezanih u matricu i tako je stvorena memorija sa direktnim pristupom.



Nakon pet godina rada i nekoliko verzija prve računarske kompanije koju su ustanovili Eckert i Mauchly, **UNIVAC** je isporučen Census Bureau baš na vreme da bi se započeo rad na desetogodišnjem popisu. Budžet je premašen, ali Remington-Rand Corporation se nadala da će moći da proizvede dovoljan broj kopija da bi nadokadnila svoje gubitke na fiksiranom ugovoru sa Vladom 1946. godine. Proizvedeno je 46 kopija.

Maurice Wilkes je ubrzo nakon završetka posla na EDSAC-u na Cambridge University uvideo da će "dobar deo ostatka (svog) života provesti u pronalaženju grešaka u...programima". Zajedno sa Stanley Gillom i David Wheelerom razvio je koncept podprograma u programima za pravljenje ponovo upotrebljivih modula; zajedno su napisali prvi udžbenik o "Pripremi programa za elektronski digitalni računar", ("The Preparation of Programs for an Electronic Digital Computer", Addison-Wesley Publ. Co., New York, 1951). Formalizovani koncept razvoja softvera (koji nije dobio ime celu deceniju) je započeo. Treća mašina Howard Aikena, Mark III je predata Naval Surface Weapons Center, Dahlgren, Virginia marta 1951. Mark III je bio značajan jer je bio prva mašina pune skale koja je uključivala doboš (engl. drum) memoriju iako je Aiken insistirao da se podaci i instrukcije čuvaju na posebnim (i dimenziono različitim) dobošima.

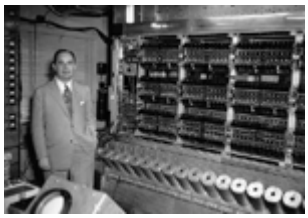


Na naslovnoj strani magazina Time je bila slika Mark III koju je naslikao Artzybasheff; to je prvo pojavljivanje računara. Slika se sada nalazi na univerzitetu Harvard.

1952

Grace Hopper, tada zaposlena u Remington-Rand i koja je radila na UNIVAC-u, uzela je koncept softvera koji se može ponovo koristiti u članku iz 1952 nazvanom "Education of a Computer" (Proc. ACM Conference, ponovo štampan u Annals of the History of Computing Vol. 9, No.3-4, pp. 271-281) u kojem je opisala tehnike kojima se koristi računar za biranje (ili prevođenje) prethodno napisanih segmenta kodova koji se objedinjuju u programe u skladu sa kodovima napisanim na jeziku visokog nivoa - opisujući tako koncept prevođenja i koncept opšteg prevođenja jezika. Sledećih četrdeset godina Hopper je bio predvodnik u razvoju lakih načina rešavanja problema i nije se obazirao na sumnjivce koji su rekli da to "ne može da se uradi". Rođena je ideja "automatskog programiranja".

Krajem 1952. godine, UNIVAC je postao uobičajeno ime za računar, kao što su Hoover i Xerox postali sinonimi za usisivače i fotokopir mašine, čemu je delimično doprinelo korišćenje UNIVAC-a u televizijskom programu u noći predsedničkih izbora. Korišćenjem obične tastature (konzole) u studiju, unošeni su rezultati glasanja koji su obrađivani na mašini u Remington-Rand fabrici u Filadelfiji. Sa samo 5% izbrojanih glasova UNIVAC je predvideo pobedu Eisenhowera, a iako je Charles Colinwood stalno tražio da "UNIVAC kaže šta misli", CBS je tek posle ponoći na istočnoj obali Amerike priznao da nije verovao predviđanjima i povukao rezultate programa koji su radili na UNIVAC-u. Izborne noći na televiziji više nikad neće biti iste, a UNIVAC je ustoličen kao glavni računar.



1952. godine **John von Neumann** je takođe završio svoju verziju naslednika ENIAC-a na Insitute for Advanced Study na Princeton University

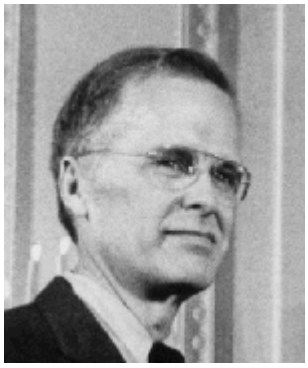
1953

Sredinom prve "policijske akcije" Ujedinjenih Nacija u Koreji, IBM je iskoristio priliku da doprinese ratnom naporu tako što je proizveo "Odbrambeni Kalkulator" koji je zapravo bio njihov prvi ulazak u računarski biznis. **IBM "Type 701 EDPM"** je bio rezultat ubeđenja T.J. Watsona Jr. da IBM treba da zakorači u ovo polje i njegovog ubeđivanja svog oca da računari neće odmah uništiti biznis obrade kartica. Serija mašina 700, uključujući 704, 709 i kasnije 7090 i 7094, dominirala je tržištem velikih računara tokom sledeće decenije i doprinela da IBM tada iz pozadine dođe na prvo mesto. Dok su mnogi univerziteti u Americi i ostalim zemljama proizvodili sopstvene računare, Cambridge University EDSAC se prvi komercijalizovao. Dalekovidom odlukom, kompanija od koje se ponajmanje očekivalo da ima snažan interes za računare, J. Lyons & Company Ltd, inače snabdevača poslastičarnica i operatora "čajdžinica" po Britaniji, uzela je EDSAC dizajn i konvertovala ga za sopstvene biznis aplikacije. Pod nazivom LEO (Lyons Electronic Office), zaokupio je pažnju kompanija sa sličnim potrebama poslovne obrade. Uspešnom realizacijom razvoja projekta za sopstvene potrebe, kompanija prerasta u novu računarsku kompaniju. LEO Computers Ltd je otkupila English Electric Company i zajedno su postali deo International Computers Ltd (ICL), glavnog proizvođača britanskih računara 70'tih.

1954

Otkako je 30'tih IBM proizveo seriju kalkulatora serije 600, koja je doprinela raznovrsnosti opreme za obradu kartica, bila je njegov glavni proizvod. Rani IBM računari (701 i 702) nisu bili kompatibilni sa opremom za bušene kartice ali je **IBM Type 650 EDPM**, prirodan produžetak serije 600, koristio iste periferijske uređaje za obradu kartica pa je on bio kompatibilan za mnoge postojeće IBM kupce.

Decimalna mašina **doboš memorije**, 650-ca je prva koja je masovno proizvođena iako IBM nije očekivao da pusti seriju 1000 odmah nakon objavljivanja. Za mnoge univerzitete je to bio prvi računar, njegova atraktivnost je znatno poboljšana nuđenjem institucijama popusta od 60% za obrazovne računarske kurseve..



Sledeći primer koji je postavio Grace Hopper i uspešnu implementaciju interpretatora digitalnog koda za IBM 701 nazvanog Speedcoding, **John Backus** je predložio razvoj programskog jezika koji će omogućiti korisnicima da izraze probleme pomoću uobičajenih matematičkih formula -- koji je kasnije nazvan FORTRAN. Sastavljajući tim od istraživača iz IBM-a i korisnika, Backus je stalno verovao da će im trebati 6 meseci da završe posao; kadgod ga je neko pitao kad će sistem biti spreman on bi odgovarao "za šest meseci"!

Dok je John von Neumann radio na IAS mašini, paralelno su tekli projekti pravljenja kopija u drugim institucijama. Da bi osigurala konformnost, Princeton grupa je uzela fotografije detalja konstrukcije IAS mašine i poslala ih sa beleškama drugim proizvođačima. U Los Alamos National Laboratory je Nick Metroplois pravio MANIAC, University of Illinois je pravio ILLIAC, a u Rand Corporationu Willis Ware je pravio **JOHNNIAC**. U martu 1954 JOHNNIAC je pušten i njime je rukovodio Keith Uncapher kasnije prvi čovek novo formirane asocijacije IEEE Computer Group, kasnije nazvanom Computer Society. 1994 **Willis Ware**u je uručena povelja IEEE Computer Society Pioneer Award za njegov rad na JOHNNIAC-u. Novoformirana National Science Foundation (NSF) je dozvolila da John van Neumann nastavi da radi na računarstvu, i to je bila Fondacijina prva podrška iz niza univerzitetske podrške razvoju računarstva.

1955

Manje od deset godina nakon otkrivanja ENIAC-a, ideja o računarstvu velike skale koju je sadržao ENIAC promenila se u koncept "superračunarstva". IBM je počeo rad na doprinosu nacionalnom naporu proizvođenjem mašine koja je obećavala 100 puta brži rad od najbrže mašine na svetu. Ova mašina je trebala da proširi trenutni nivo sofisticiranosti razvojne tehnologije pa je nazvana STRETCH. Kad je STRETCH napokon isporučen 1960 naznačena cena je morala da bude snižena jer nije dostignuta ciljna brzina. Iste godine IBM je predstavio računar **704** čiji je principijelni arhitekt bio Gene Amdahl koji je 90'tih osnovao sopstvenu kompaniju za proizvodnju superračunara. 704 se odlikovala time da je bila prva komercijalna

mašina sa hardverom za rad u tzv. pokretnom zarezu i mogla je da radi približnom brzinom od 5 kFLOPS-a. Brojni kupci su se udružili i oformili prve grupe korisnika da bi razmenjivali iskustva i programe i u isto vreme se predstavili kao jedinstveni naspram proizvođača. Korisnici (velikih) IBM mašina oformili su grupu SHARE (nije skraćenica ali je često naznačena kao "Society to Help Allieve Redundant Effort") a korisnici UNIVAC-a grupu USE. Računarstvo nisu više oblikovale samo računarske kompanije.

1956

Sperry-Rand, naslednik Remington-Randa, koji je i dalje održavao UNIVAC Division, napravio je superračunar za potrebe Lawrence Livermore National Laboratory (LLNL), koji je nazvan LARC (Livermore Automatic Research Computer). U Engleskoj je takođe započet rad na projektu superračunara. Projekat Atlas su zajednički započeli University of Manchestera i Ferranti Ltd-a, sa Tom Kilburnom na čelu.

Ne zaboravljajući da je svrha računara da reše problem, **John McCarthy** i Marvin Minsky su organizovali konferenciju na Dartmouth College-u, uz pomoć Rockefeller Foundation, o konceptu veštačke inteligencije. Zaključak ove konferencije je bio da će doći do razvoja veštačke inteligencije, što se nije ostvarilo u nekoliko narednih godina

1957

Rani računari su imali malu unutrašnju i sporu spoljašnju memoriju jer su se oslanjali na magnetnu traku. Vremenom je unutrašnja memorija poboljšana na magnetnu doboš pa na memoriju sa magnetnim jezgrima. Sledeći logičan korak bila je disk memorija sa pokretnim glavama za čitanje/pisanje da bi obezbedila

sposobnost poludirektnog pristupa i kapacitet skladištenja koji odgovara magnetnoj traci. IBM 305 RAMAC je bio prvi sistem sa memorijskim diskom.

Nakon tri godine rada Backus i njegove kolege su isporučili FORTRAN programski prevodilac za IBM 704, a odmah zatim su naišli na prvu poruku o greški – nedostaje zarez u izračunatoj GO TO naredbi. Herbert Bright iz Westinghouse u Pittsburghu primio je neoznačen paket od 2000 kartica i utvrdio da je to dugo očekivani prevodilac, a zatim napravio prvi korisnički program – zajedno sa greškom. Svet programskih jezika je napredovao od oblasti u kojoj su samo obučeni programeri mogli da završe projekat, do oblasti u kojoj su oni koji imaju probleme mogli sami da izraze svoja rešenja.

1958

Pronalazak tranzistora u drugoj polovini 40-tih otvorio je eru moderne elektronike korišćenja »elektrona u čvrstim telima« i napuštanje, mada na vrhuncu moći, staromodne elektronske vakuumske cevi koja koristi "elektrone u vakumu". 1958, Jack St. Clair Kilby je začeo i dokazao svoju ideju integrisanja tranzistora sa otpornicima i kondenzatorima na jednom poluprovodničkom čipu, koji je monolitsko integrisano kolo (IC). Njegova ideja monolitskog IC, zajedno sa planarnom tehnologijom Dr. Jean Hoernija i Robert Noyceovom idejom "o izolaciji spoja" za planarna vezivanja, učvršćuje veliki progres današnjeg poluprovodničkog IC i mikroelektronike koja je bazirana na njemu. Tehnologija je omogućila inovacije mnogobrojnih aplikacija u računarima i komunikacijama, koje su dramatično promenile naš stil života.

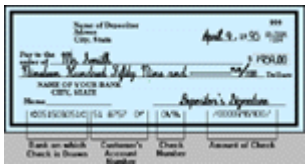
Originalni razvoj koji je počeo sa projektom Whirlwind postao je realnost 1958. godine sa instalacijom sistema SAGE za vazdušnu odbranu na McGuire AFB u NJ. Prvi efikasni sistem kontrole vazdušnog saobraćaja postao je operativan za severno istočnu Ameriku.

Tada tek osnovana korporacija Control Data Corporation pod vođstvom William Norrisa je dala doprinos tržištu superračunara sa potpuno tranzistorizovanim računarom --CDC 1604 --Seymour Cray je bio glavni arhitekt. U međuvremenu, nastavljajući svoj rad u razvoju veštačke inteligencije, John McCarthy je razvio koncepte programskog jezika LISP za obradu nizova simbola, tj. nenumerički procesni jezik. Naredne generacije studenata su promenile značenje jezika LISP, što je skraćenica za LISt Processing, u "Lots of Idiotic, Silly Parentheses".

1959

Dok je u mnogim kompanijama postojao pomak ka superračunarima, IBM je objavio mogućnost mašina veličine dva stola za male korisnike -- IBM 1401 za poslovne korisnike i IBM 1620 za naučnike. Mašina 1401 je postala najpopularnija mašina za obradu poslovnih podataka, a mašina 1620 je za mnoge studente bila prvo računarsko iskustvo na malim univerzitetima i srednjim školama. Obe mašine su uvele znakovno orijentisanu glavnu memoriju od 20-40k bajta u kojoj granice "reči" može programer da definiše da bi obezbedio "neograničenu preciznost". Obe mašine su imale aritmetičku jedinicu koja je koristila tabelu pretraživanja umesto binarnih sabirača. Prvobitno je IBM nameravao da mašinu 1620 nazove kao CADET, ali kad je ovo prevedeno u "Can't Add, Doesn't Even Try" (Ne može da sabere, čak ni ne pokušava) odustali su od tog imena.

Nakon nekoliko godina rada General Electric Corporation je isporučila 32 ERMA (Electronic Recording Machine --Accounting), računarski sistem za Bank of America u Kaliforniji da bi spasila bankarsku industriju od poplave rastućeg broja čekova koje je koristila rastuća klijentela. Zasnovan na SRI dizajnu, ERMA sistem je koristio Magnetic Ink Character Recognition (MICR) kao sredsvo za dobijanje podataka sa čekova i uveo sistem za obradu čekova kojeg nisu plašila dokumenta koja nisu bila primitivna. Bankarska industrija je automatizovana i otvarala je puteve za nove načine poslovanja uključujući ATM (bankomati) i elektronsko personalno bankarstvo. Sa druge strane, to je bio bitan događaj u istoriji proizvodnje računara u GE koji, izuzev razvijanja profitabilne linije mašina za NCR (NCR 304), nikada nije dostigao status koji se mogao očekivati od takvog finansijskog džina.



1960-te

PGEC servisi u ranim šesdesetim su bili skoro isti kao u kasnim pedesetim iako se povećavao broj konferencija i stranica u stručnim časopisima. Međutim, 1961. godine, vođstvo PGEC je počelo da razmatra stvaranje tehničkih komiteta. Ovi komiteti su trebali da obezbede više foruma za posebna interesovanja i u isto vreme da smanje da zainteresovani formiraju odvojene IRE grupe i da razdvoje oblast. U maju 1962. godine, prvi od ovih komiteta, komitet za logiku i teoriju prekidača, odobrio je zajednički rad sa komitetom AIEE koji je već funkcionisao. Istovremeno, nastavljeni su planovi na udruživanju IRE i AIEE. IRE-AIEE su se udružili u Institute of Electrical and Electronics Engineers (IEEE) na nivou rukovodstva 1963. godine. PGEC je tada postao Professional Technical Group on Electronic Computers, a odmah potom Computer Group. Početkom 1963. godine, grupa je započela rad sa Administrative Committee koji je uključivao ljude iz PGEC i AIEE CDC. Konačno udruživanje je završeno u aprilu 1964. godine.

Jula 1966. godine preduzet je važan korak sa prvim dvomesečnim izdanjem *Computer Group News,* koji je sadržao vesti o grupi i industriji, primenjene članke i uputstva za rad, vodič za računarsku literaturu, i mnogo računarskih članaka. Arhivski materijali su bili dostupni za profesionalce iz struke po nominalnoj ceni. *Computer Group News* je otvorio vrata za mnoge magazine u udruženju, kao i u IEEE. Takođe, bio je važan i na drugi način. Sa izdanjem prvog magazina, Computer Group je zaposlila i rukovodila radnicima koji su bili zaposleni puno radno vreme u području Los Angelesa za podršku izdavaštvu i druge

administrativne aktivnosti. Computer Group je bila prva IEEE grupa koja je zaposlila svoje ljude, i to je bio glavni faktor razvoja društva. 1968. godine, *IEEE Transactions on Computers* je postao mesečno izdanje. Broj periodično objavljenih stranica je narastao na skoro 9,700 u stručnim publikacijama i oko 640 u Computer Group News. Broj članova je narastao na 16 862, uključujući 4 200 studenata i 158 pridruženih članova. Dekada je završena sa 41 sekcijom.

1. UVOD U RAČUNARSKE SISTEME

Kroz čitavu svoju istoriju, ljudi su bili prinuđeni da vrše različita izračunavanja i obradu informacija dobijenih iz sveta koji ih okružuje. Obim i složenost ovih izračunavanja neprekidno su se povećavali, a manuelno izračunavanje, u kojem je čovek osnovno sredstvo, ima dva velika ograničenja u obavljanju ovih poslova:

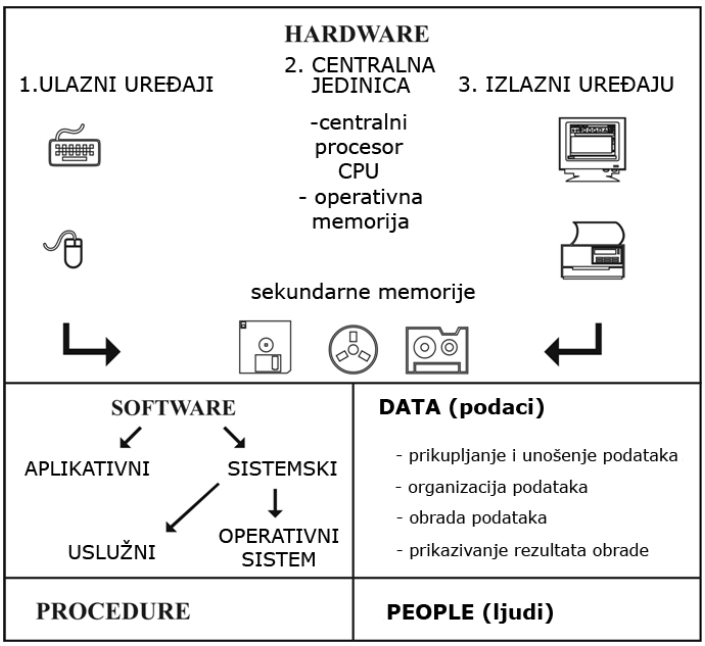
1. Čovekova *brzina je vrlo ograničena. Za obavljanje elementarnih operacija sabiranja ili množenja, čoveku je potrebno od nekoliko sekundi do nekoliko minuta.*
2. Čovek *pokazuje sklonost ka pravljenju grešaka, tako da su rezultati vrlo složenih izračunavanja i obrade, koje obavlja čovek, često nepouzdani. Za razliku od čoveka, mašine su potpuno imune na niz ljudskih grešaka u procesu obrade informacija, koje su posledica: rastrojenosti, zabrinutosti, umora čoveka itd.*

U procesu izračunavanja ljudi koriste razna pomoćna sredstva, sa ciljem da sebi olakšaju posao i da višestruko ubrzaju postupke obrade, da dobiju tačne rezultate i da donesu pravilne odluke. Ta sredstva su se neprekidno razvijala i usavršavala da bi, sredinom dvadesetog veka, dovela do pojave automatskih elektronskih cifarskih računskih mašina, ili skraćeno, računara. *Računar je uređaj koji samostalno obavlja obradu podataka izvršavajući digitalne logičke operacije na osnovu unesenog programa.*

Računarski sistem predstavlja skup svih sredstava koja koristimo u procesu rešavanja jednog ili grupe zadataka.

1.1. OPŠTI MODEL RAČUNARSKOG SISTEMA

Jedan od najopštijih modela računarskog sistema je onaj koji je dao **David Kroenke**, u kojem se računarski sistem sastoji od pet komponenti: tehničkog dela (**hardware,** hardver), programskog dela (**software,** softver), podataka (**data**), procedura (**procedures**) i ljudi (**people**), slika 1.1.



Slika 1.1. Opšti model računarskog sistema

Hardver računarskog sistema čini tehnički deo, a sastoji se od skupa različitih uređaja, od kojih svaki obavlja više jednostavnih operacija i obrada, koje se mogu grupisati u neke osnovne funkcije. Na osnovu osnovnih funkcija koje obavljaju, tehnički deo računarskog sistema čine: ulazni uređaji, uređaji za obradu, tj. računar u užem smislu, ili centralna jedinica i izlazni uređaji.

Ulazni uređaji omogućavaju pristup računarskom sistemu. Posredstvom ovih uređaja korisnici zadaju komande, traže informacije, unose nove podatke ili programe u računar. Računar prihvata ulaze vrlo različitog tipa, na primer pritisak na dugme ili taster, dodir prsta na ekran, optičko skaniranje, izgovaranje reči, itd.

Uređaj za obradu, centralna jedinica ili računar, sadrži elektronske komponente koje na osnovu ugrađenog ili unetog programa (niza instrukcija) izvršavaju različite aritmetičke ili logičke operacije, modifikuju tekst, donose različite odluke, prihvataju ulazne podatke i generišu izlazne podatke.

Kada je obrada podataka izvršena, rezultat mora biti dostupan korisniku. Ovu funkciju obavljaju izlazni uređaji, a u njih spadaju video displeji (monitori, ekrani), štampači (printers), ploteri, zvučnici itd.

Posebnu grupu uređaja, koji istovremeno obavljaju funkcije ulaza i izlaza, kao i funkciju dugotrajnog pamćenja i čuvanja podataka i programa, čine spoljne memorije u koje spadaju razne vrste diskova, traka itd.
Da bi tehnički deo računarskog sistema (hardver) ostvario svoje funkcije obrade, neophodan mu je niz instrukcija, tj. elementarnih operacija. Instrukcije se grupišu tako da zajedno omogućavaju rešavanje određenog zadatka, ili obavljanje neke složene operacije i obrade, i tada čine programe. Skup svih programa predstavlja softverski deo računarskog sistema. Programi mogu da sadrže niz detaljnih instrukcija u obliku nizova cifara **0** i **1** (u tzv. mašinskom jeziku), koje računar može da koristi neposredno. Programi mogu biti napisani i na nekom od programskih jezika kao što su **Assembler, COBOL**, **Pascal, BASIC** itd., koji su ljudima lakši za razumevanje, ali ih računarski sistem, pre nego što ih računar izvrši, mora prevesti na mašinski jezik.

Programi koji ulaze u sastav računarskog sistema svrstavaju se u dve grupe:

1. *Sistemski programi koji upravljaju radom i omogućavaju korišćenje tehničkog dela računarskog sistema s jedne strane, i korisnicima olakšavaju izradu sopstvenih programa, komuniciranje sa pojedinim uređajima i omogućavaju njihovo lako i efikasno korišćenje. Sistemski programi su posrednici između korisnika i njegovog programa s jedne strane, i hardvera računarskog sistema s druge strane.*
2. *Korisnički programi rešavaju neki konkretan zadatak ili obradu. Oni korisniku daju neke konkretne rezultate, na osnovu kojih on preduzima određene aktivnosti. U toku svog izvođenja, korisnički programi nužno koriste različite funkcije iz grupe sistemskih programa. Ovi programi se nazivaju još i aplikativni programi ili problemski programi.*

Podaci su treći deo modela. Podaci mogu imati oblik imena, brojeva i adresa. Oni mogu predstavljati meru najrazličitijih fizičkih veličina: od nivoa vode u posudi, do spektra svetlosti zvezda. Podaci su osnovni objekat obrade u računarskom sistemu, odnosno predmet rada i ujedno glavni razlog korišćenja računara. Bez računara naše mogućnosti da crpemo znanja iz podataka bile bi značajno smanjene.
Podaci su diskretni, zapisani fakti o pojavama i događajima iz sveta koji nas okružuje, iz kojih dobijamo informacije o svetu. Drugim rečima, podaci su činjenice, oznake ili zapažanja nastala u toku nekog procesa, a koja su zapisana, odnosno kodirana pomoću nekih fizičkih simbola, ili simbola neke azbuke, i imaju svojstvo da mogu da se beleže, čuvaju, prenose i obrađuju. Podaci su sredstva za izražavanje i dobijanje informacija, i oni predstavljaju izolovane i neinterpretirane činjenice. Podatke prikupljamo i zapisujemo, da bi ih mogli čuvati, i po potrebi koristiti. Podatak postaje informacija u momentu njegovog korišćenja.

Obrada podataka može se podeliti u četiri faze:

1. *prikupljanje i unos podataka,*
2. *organizacija podataka i njihovo čuvanje, tj. skladištenje,*
3. *obrada podataka pomoću računara, tj. obrada u užem smislu (računanje, sortiranje, grupisanje itd.),*
4. *izdavanje podataka i rezultata obrade (štampanje, crtanje, itd.).*

U nekim primenama računara ne moraju postojati sve četiri pomenute faze obrade. Tako, recimo, u mnogim naučno-tehničkim obradama nema uopšte ulaznih podataka, već se podaci izračunavaju u računaru kao, recimo, pri izračunavanju logaritamskih tablica, slučajnih brojeva itd.Procedure predstavljaju skup postupaka i pravila, koja se moraju poštovati i primenjivati u cilju pravilne upotrebe računarskog sistema, i pravilne obrade podataka, da bi korisnici, tj. ljudi, pomoću računara, rešili neki zadatak. Procedure predstavljaju instrukcije upućene ljudima, koje korake treba izvršiti, kako i koje podatke obraditi, i koje izlazne veličine treba dobiti.
Petu komponentu računarskog sistema čine ljudi. Ljudi se pridržavaju određenih procedura prilikom korišćenja računara. Oni obavljaju aktivnosti koje stvaraju ili prikupljaju podatke. Ljudi razvijaju programe kojima rešavaju nove postupke obrade i menjaju postojeće programe, da bi se zadovoljili novonastali zahtevi. Ljudi za druge ljude kreiraju procedure, kojih se oni u procesu korišćenja računara pridržavaju.

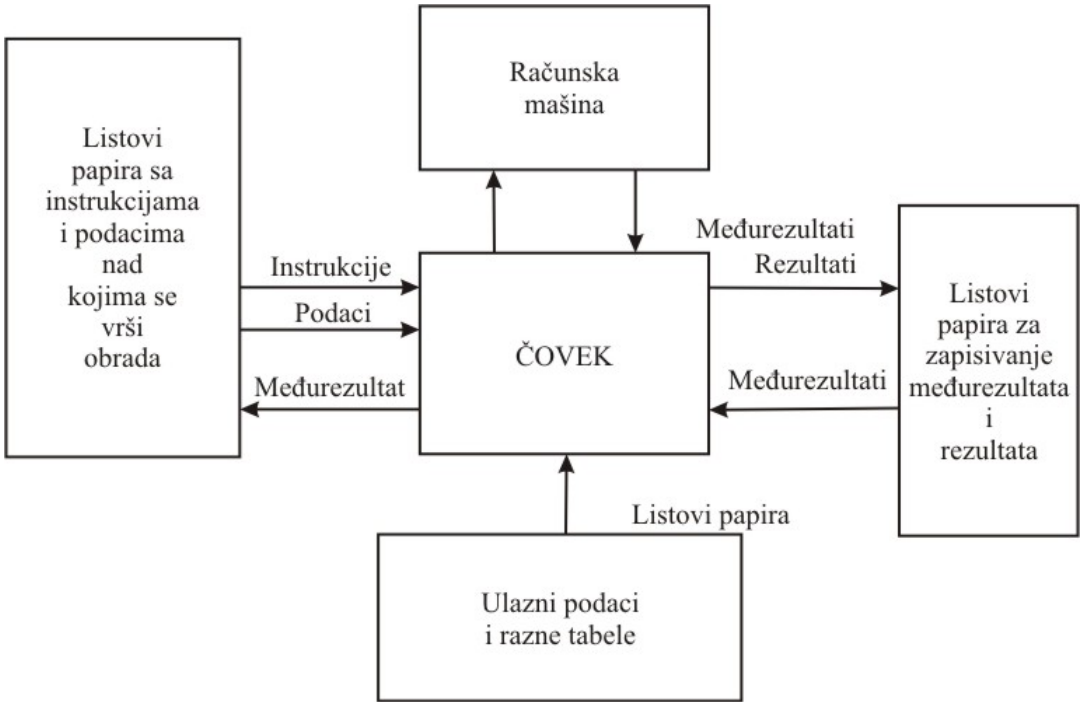
1.2. FUNKCIONALNA BLOK-ŠEMA RAČUNARA

Pre opisa rada računara razmotrimo najpre kako čovek rešava neki zadatak uz pomoć najjednostavnije stone računske mašine koja obavlja samo četiri osnovne računske operacije.

RUČNA OBRADA PODATAKA

U ručnoj obradi podataka (slika 1.2.), postupak računanja, tj. rešavanje bilo kog problema, sastoji se od niza pojedinačnih koraka od kojih svaki sadrži po jednu od elementarnih operacija (+, -, * , /). Svaki pojedinačni korak propisuje na koji način i nad kojim podacima treba izvršiti izračunavanje, da bi se na kraju dobio tačan rezultat, odnosno uradio postavljen zadatak. Čovek svaki zadatak obrade, ma koliko bio prost ili složen, najčešće formalizuje u obliku kolona tabele, koju zapisuje na jednom ili više listova papira. U svaku vrstu tabele čovek zapisuje po jednu elementarnu operaciju i podatke nad kojima se ona trenutno izvršava. Svaka vrsta ove tabele, tj. elementarna operacija, predstavlja elementarnu instrukciju, ili kraće rečeno instrukciju.Određeni niz instrukcija koji rešava postavljeni zadatak, a nakon čijeg izvršavanja se dobija traženo rešenje, zove se program obrade. U

programu, elementarne obrade, tj. instrukcije, najčešće su poređane onim redosledom kojim se izvršavaju. To znači da je redosled izvršavanja elementarnih operacija strogo definisan redosledom instrukcija u programu obrade, i svakoj od ovih instrukcija pridružuje se redni broj. Po pravilu se npr. posle pete instrukcije izvršava šesta, a posle šeste izvršava se sedma instrukcija, i tako redom. Od ovog prirodnog redosleda može se odstupiti kada se steknu određeni uslovi, tj. zavisno od konkretnih vrednosti početnih podataka ili dobijenih rezultata. Na primer, zavisno od toga kakve su vrednosti koeficijenta **a** i diskriminante **D** (**$D=b^2- 4ac$**), postupci za izračunavanje realnih korena $\mathbf{x_1}$ i $\mathbf{x_2}$ kvadratne jednačine (**$ax^2 + bx + c = 0$**) su različiti, i izračunavaju se po različitim formulama. Naime, ako je **a = 0** jednačina nije kvadratna, ako je **D = 0** onda su koreni jednaki $\mathbf{x_1 = x_2 = - b / (2 a)}$, a ako je **D < 0,** jednačina nema realnih korena, već su koreni konjugovano kompleksni, pa se realni i imaginarni deo moraju prikazati kao posebni delovi jednog složenog podatka.



Slika 1.2. Ručna obrada podataka pomoću računske mašine

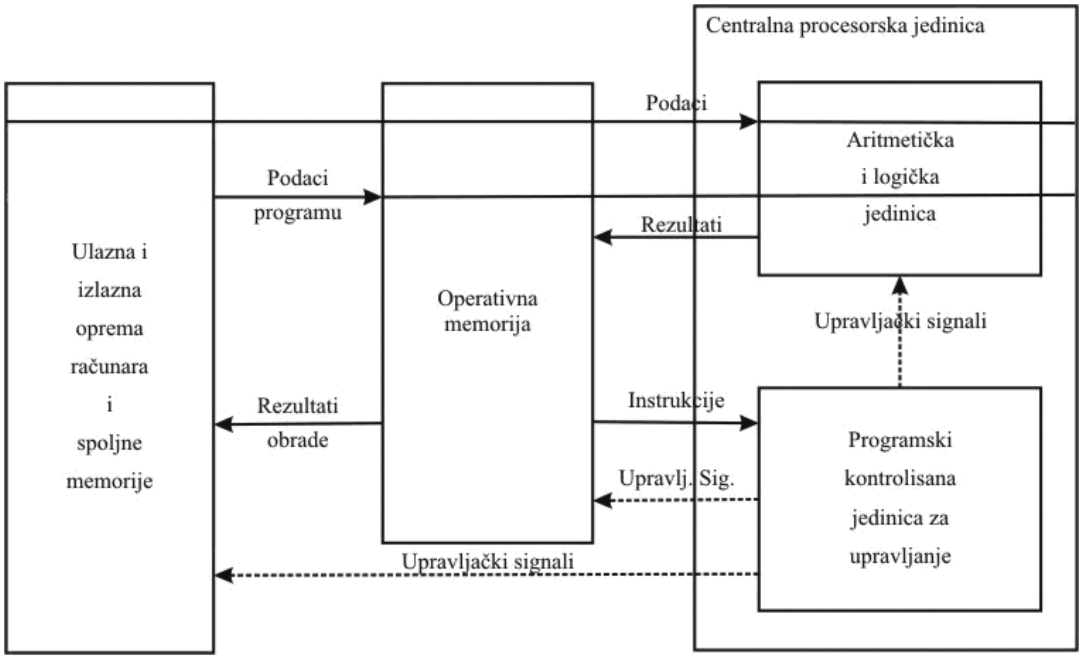
Podaci nad kojima se vrše elementarne operacije obično se prikupljaju pre samog procesa obrade (u kraćem ili dužem periodu vremena), i nalaze se zapisani (memorisani) na posebnim listovima papira. Vrlo često se, u cilju dobijanja različitih informacija nad jednim istim podacima vrše različite obrade. U pojedinim koracima obrade učestvuju samo neki podaci, pa se samo ti podaci privremeno prepisuju na listove papira, ili pridružuju instrukcijama u tabelama koje sadrže rezultate obrade.

Čovek koji vrši izražunavanje, često u cilju rešavanja svog problema, tj. zadatka, koristi opštepoznate veličine, kao i razne matematičke, statističke i druge tabele (logaritme, trigonometrijske funkcije itd.).

U toku obrade formiraju se međurezultati (koje čovek privremeno pamti, ili zapisuje), a na kraju se dobijaju konačni rezultati obrade koje čovek zapisuje na papiru. Međurezultati i konačni rezultati jednog postupka obrade, mogu biti početni podaci u nekom drugom postupku obrade.

AUTOMATIZOVANA OBRADA PODATAKA POMOĆU RAČUNARA

Struktura elektronskog sistema za automatizovanu obradu podataka pomoću računara (slika 1.3.) je vrlo slična strukturi pri ručnoj obradi podataka (koja je prikazana na slici 1.2.), ali je sastav drugačiji.



Slika 1.3. Glavne komponente elektronskog uređaja za obradu podataka

Umesto računske mašine, u automatizovanoj obradi pojavljuje se element za računanje, tj. aritmetičko-logička jedinica računara (**arithmetic and logic unit, ALU**). Funkciju papira, koji je čoveku služio kako za formulisanje zadataka obrade, tako i za upisivanje tabela sa početnim podacima, međurezultatima i finalnim rezultatima obrade, preuzimaju sada razne vrste memorija. Čoveka sada zamenjuje programski kontrolisana upravljačka jedinica (**control unit, CU**), odnosno komandna ili kontrolno-upravljačka jedinica. Kako je čovek na ovaj način,

u postupku obrade podataka, znatno rasterećen od izvršavanja niza rutinskih poslova obrade, onda on može da se posveti isključivo formulisanju zadataka i postupaka obrade, tj. izradi programa.

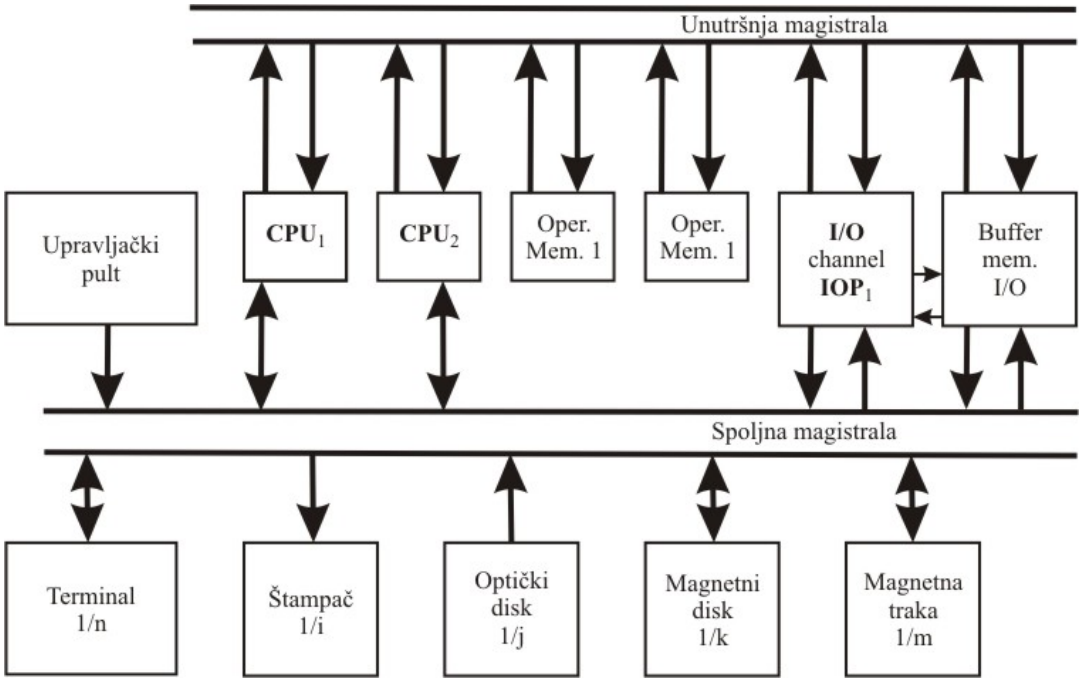
Unutrašnja, odnosno operativna memorija je osnovna ili glavna memorija računara (**main memory, operating memory, primary memory**). Ona sadrži podatke nad kojima se vrši obrada, međurezultate, kao i sam način na koji se vrši obrada, tj. zadatak koji treba obaviti. Način kako se vrši obrada sadržan je u programu koji je pisao čovek, i to u obliku niza pojedinačnih instrukcija, odnosno elementarnih operacija.

Upravljačka jedinica uzima iz unutrašnje memorije jednu instrukciju za drugom, interpretira ih i izvršava neposredno, ili uz pomoć aritmetičko-logičke jedinice. Aritmetičko-logička jedinica najčešće izvršava samo osnovne računske operacije i radnje, i ne može da izvršava složenije računske operacije. Kontrolna jedinica, zajedno sa računskom jedinicom, čini jedinicu za obradu, tj. centralnu jedinicu za obradu (**central processing unit, CPU**), ili kraće centralni procesor računara.Centralni procesor zajedno sa unutrašnjom memorijom čini centralnu jedinicu računarskog sistema, odnosno računar u užem smislu. Danas su računari realizovani u obliku elektronskih, digitalnih, logičkih sklopova u poluprovodničkoj tehnologiji sa velikom brzinom rada. Spoljašnje, masovne memorije (**external memory, secondary storage, mass storage**) služe za ulaz, izlaz i trajno čuvawe velike količine podataka, međurezultata i rezultata obrade. Ove memorije su realizovane na različitim medijima (nosioci informacija), koji imaju svojstvo trajnog pamćenja, kao što su: magnetne trake i diskovi, optički diskovi, itd.

Ovi nosioci informacija omogućavaju ujedno da se podaci i rezultati obrade na jednom računarskom sistemu, prenesu na drugi uređaj za automatsku obradu podataka radi dalje obrade, bez uspostavljanja fizičke veze izmedju ovih uredjaja.
Računari su u svom razvoju od nastanka do današwih dana prošli kroz nekoliko faza razvoja koje nazivamo generacijama. Prvi računari imali su arhitekturu koja je slična onoj sa slike 1.3., i koja se pojednostavljeno može prikazati kao na slici 1.4. Može se reši da je arhitektura računara prve generacije gotovo identična strukturi ručne obrade podataka, jer se prenos podataka obavljao ne samo uz kontrolu, već i uz učešće centralnog procesora. Na slikama 1.3. i 1.4. prikazana je arhitektura u kojoj su ulazno-izlazni uređaji direktno povezani sa centralnim procesorom ili sa glavnom memorijom računara. U najvećem broju savremenih računara ovo komuniciranje obavlja se preko odgovarajućih uređaja, kanala (**input-output processors, IOP, channels**), i uz posredovanje prihvatnih memorija za spregu, međumemorija (**buffer memory**), kao što je prikazano na slici 1.5.

Uočavamo da u okviru jednog računarskog sistema možemo imati više centralnih procesora koji paralelno rade, a omogućavaju znatno ubrzanje rada računara i paralelno izvršavanje jednog ili više programa. Za kontrolu prenosa podataka između ulaznih jedinica, izlaznih jedinica i jedinica masovnih memorija, koje se

jednim imenom nazivaju periferijske jedinice, s jedne strane, i centralnih procesora i unutrašnje memorije s druge strane, najčešće se koriste ulazno-izlazni kanali i procesori. Ovi uređaji često sadrže autonomne memorije za spregu (**buffer memory**), čime se ostvaruje dodatna autonomnost rada uređaja. Neposrednu kontrolu ovih ure|aja vrše centralni procesori, a uređaji samostalno kontrolišu prenos podataka, i ujedno vrše prilagođenje (po brzini) između vrlo brzih procesora i unutrašnje memorije, i vrlo sporih mehaničkih sklopova i spoljnih memorija.



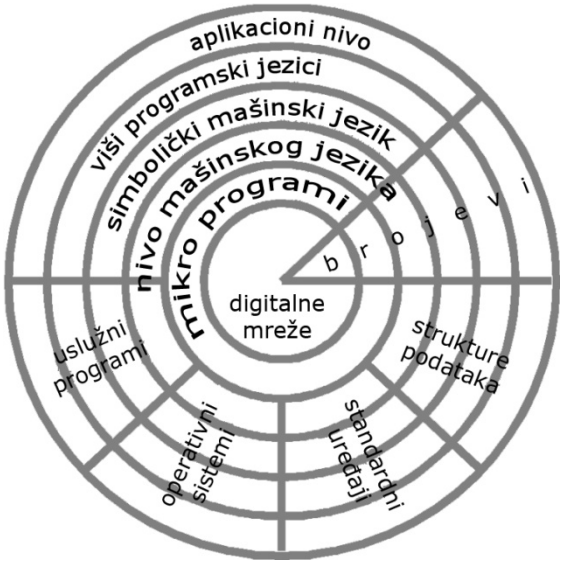
Slika 1.5. Struktura računarskog sistema sa ulazno-izlaznim kanalima

Jedan kanal u jednom intervalu vremena može opsluživati više periferijskih uređaja, takozvani multipleksorski kanal, ili samo jednu relativno brzu spoljnu jedinicu (magnetni disk, optički disk, ...), takozvani selektorski kanal.

1.3. HIJERARHIJSKI MODEL RAČUNARSKOG SISTEMA

Mnogi autori pod pojmom računarskog sistema podrazumevaju uglavnom prve dve komponente opšteg modela, odnosno hardver (tehnički deo), i softver (programski deo u čiji sastav ulaze i podaci), što čini računarski sistem u užem smislu.

Ovakav računarski sistem može se prikazati hijerarhijskim modelom u obliku koncentričnih krugova (slika 1.5.).



Slika 1.6. Hijerarhijski model računarskog sistema

U centru ovog modela su digitalna logička kola (**digital logic**) iz sastava računarskog hardvera. Ovaj prsten određuje osnovne aritmetičke i logičke operacije računara. Ovom prstenu (jezgru) pripadaju i osnovna memorijska kola. Svaka komponenta ovih logičkih mreža izvršava po jednu vrlo jednostavnu operaciju. Jezgro može sadržati desetine, ili čak stotine operacija, koje upotpunjavaju nešto što ljudi u računskoj tehnici smatraju skupom elementarnih obrada.

U mnogim modernim računarima, rad ovih elementarnih logičkih mreža koordinira se pomoću niza programa koji se nazivaju mikroprogrami (**microprogram, microcode**).

Mikroprogrami sadrže niz instrukcija poznatih kao mikrooperacije, koje koordiniraju rad logičkih kola, i stvaraju mogućnost da ljudima poznate, jednostavne instrukcije, poput sabiranja ili oduzimanja, budu realizovane kao nizovi mnoštva još prostijih operacija.

Treći nivo čini mašinski jezik (**conventional machine level**) i on predstavlja najniži nivo zajednički za sve programe, jer centralni procesor razume samo ove instrukcije, i sposoban je da ih interpretira i izvrši. Svaka instrukcija koja nam je na raspolaganju na ovom nivou u stvarnosti je podržana - realizovana ili kao jedan mikroprogramom ili kao jedna logička mreža.

Sledeća dva nivoa su simbolički mašinski jezik (asembler, **assembly language**) i viši programski jezici (**high-level language**) koji čine programsku spregu čoveka sa računarom. Asemblerski jezik je oslonjen na mikroprogramski nivo i digitalne logičke mreže. Pisanje programa na ovom jeziku je znatno lakše nego na mašinskom, ali takođe zahteva veliko poznavanje arhitekture i organizacije centralnog procesora i računara. Viši programski jezici (**BASIC, Pascal,**

FORTRAN) su ljudima znatno razumljiviji, traže vrlo malo poznavanje računara, a takođe ne zavise od tipa računara.

Najopštiji nivo je korisnički (**applications**), koji čine ljudi koji čak ne moraju biti ni programeri već su samo krajnji korisnici. Oni računar posmatraju kroz određenu grupu programa, posebno projektovanih za rešavanje specifičnih problema, tako da računarski resursi optimalno služe korisniku.

Brojevi (**numbers**) povezuju sve nivoe hijerarhijskog modela, odnosno sastavni su deo svih nivoa. Digitalna logička kola operišu samo sa dve cifre: **0** i **1**, i na bazi wih proizvode nove podatke koji su, takođe, prikazani kao nule i jedinice. Na nivou mikroprograma i mašinskog jezika, nizovi nula i jedinica predstavljaju instrukcije, podatke i memorijske lokacije koje obrađuju digitalna logička kola. Programeri u asembleru i jezicima višeg nivoa, koriste brojeve i simboličke kodove, da označe šta da se obradi, kako da se to uradi i nad kojim objektom da se uradi.

Brojevi i kodovi se često organizuju u grupe koje predstavljaju vrlo različite pojave iz sveta koji nas okružuje. Organizovane grupe podataka poznate su kao strukture podataka, i značajne su kako programerima, tako i operativnom sistemu. Čak i mali personalni računari za većinu ljudi suviše su složeni da bi ih oni koristili direktno. Čitav niz sistemskih programa, poznatih kao operativni sistem (**operating system**), služi kao zaštitna školjka ili sprega koja razdvaja korisnika i aplikacione programe od složenosti hardvera. Operativni sistem, sam za sebe, radi vrlo malo vremena i bez konkretnih rezultata obrade. Njegov zadatak je opsluživanje, odnosno on čini okruženje u kome se radi.

Ma koliko neki računarski sistem bio veliki, on ipak ima ograničene resurse (uređaji iz sastava računarskog sistema), unutar kojih se koris-nički programi moraju izvršavati. Operativni sistem u procesu izvođenja programa vrši podelu tih resursa. Jedan od glavnih resursa koje on kontroliše je operativna memorija. Svaki korisnik i aplikacija zahteva da memorija računara sadrži njegove programe i podatke. Ko će, kada i koliko operativne memorije dobiti, određuje operativni sistem. Logička mreža koja izvršava aritmetičke i logičke operacije, nalazi se u centralnoj jedinici za obradu (**central processing unit, CPU**), a mnogi računari imaju samo jednu CPU koja mora da se deli na više korisnika ili programa. Operativni sistem određuje ko i koliko dugo ima pristup centralnom procesoru.

Uređaji iz sastava računarskog sistema rade pod kontrolom operativnog sistema, koji u sebi sadrži programe koji korisniku i njegovom programu omogućavaju korišćenje disketa, diskova ili drugih memorijskih medija, zatim štampača (printera), terminala, komunikacionih linija itd.

1.4. ZAKLJUČAK

Od davnina, ljudi u procesu izračunavanja koriste razna pomoćna sredstva sa ciljem da sebi olakšaju i višestruko ubrzaju postupke obrade, dobijanja tačnih rezultata i donošenja pravilnih odluka. Danas su računarski sistemi postali neizostavni sastavni deo obavljanja mnogih poslova, naročito u sferi nauke, tehnike i obrade podataka. No, isto tako veliku primenu našli su i u svakodnevnom životu, prvenstveno zahvaljujući razvoju i primeni mikroračunara (naročito sa pojavom mikroprocesora).

U sastav računarskog sistema ulaze razni tehnički uređaji, programi koji rešavaju neki problem ili pomažu u korišćenju uređaja ili rešavanju konkretnih problema, podaci koji se obrađuju, procedure koje obezbe|uju efikasno korišćenje računarskog sistema i ljudi, programeri ili korisnici koji uz pomoć računara rešavaju neke svoje probleme.

Svaki intelektualni postupak, koji možemo izraziti pomoću konačnog broja elementarnih operacija, može se predati računarskom sistemu da ga izvede. Time se postiže prenošenje jednog dela rutinskog intelektualnog rada na mašinu. Međutim, za realizaciju ovog vrlo važnog vida oslobađanja čoveka od zamornog rutinskog rada, potrebno je mnogo složenih postupaka, a njihova izrada, proveravanje, izvođenje i korišćenje traži mnogo truda i posla.

Osnovu za izradu računara predstavljaju digitalna logička kola koja operišu sa samo dve cifre: 1 i 0. Sve informacije i podaci unutar računarskog sistema prikazani su u obliku nizova binarnih brojeva. Ovi brojevi se organizuju u veće celine radi lakšeg pamćenja i obrade.

Jedan od ključnih i, slobodno se može reći, najsloženijih delova računarskog sistema, koji omogućava prenošenje velikog dela intelektualnog rada na mašinu, jeste sistemski softver. Glavni deo sistemskog softvera čini operativni sistem. Pod kontrolom operativnog sistema radi ne samo korisnikov program već i svi uređaji iz sastava računarskog sistema.

1.5. PITANJA

1. Opisati svaki od pet sastavnih delova opšteg modela računarskog sistema.
2. Koja se sredstva koriste u postupku ručne obrade podataka?
3. Kako se automatizuje postupak obrade podataka pomoću računara?
4. Objasniti funkcionalnu blok šemu računara i funkcije svakog od blokova.
5. Kakve veze postoje između raznih nivoa računarskog hardvera i softvera, sadržanih u hijerarhijskom modelu računarskog sistema?
6. Da li su resursi računarskog sistema ograničeni, i šta radi operativni sistem?

1.6. KLJUČNE REČI

- aritmetičko-logička jedinica (**Arithmetic and Logic Unit, ALU**)
- asembler (**assembler**)
- centralna jedinica za obradu (**Central Processing Unit, CPU**)
- digitalna logička kola (**digital logical circuit**)
- instrukcije (**instructions**)
- izlaz (**output**)
- kontrolno-upravljačka jedinica (**Control Unit, CU**)
- mašinski jezik (**machine language**)
- memorija (**memory, store**)
- mikroprogram (**microprogram**)
- obrada (**processing**)
- operativni sistem (**operating system**)
- podatak (**data**)
- program (**program**)
- programi (**software**)
- programski jezik (**programming language**)
- računar (**computer**)
- računarski sistem (**computer system**)
- strukture podataka (**data structures**)
- tehnički deo sistema (**hardware**)
- ulaz (**input**)
- viši programski jezik (**high level language**)

2. MATEMATIČKE OSNOVE RAČUNARA

Osnovni objekti koji se pamte i obrađuju u računarskom sistemu su brojevi. Brojevi su osnova funkcionisanja i korišćenja računara na svim nivoima, počev od nivoa digitalnih logičkih kola, do obrade na korisničkom nivou. Brojevi se koriste ne samo u aritmetičkim operacijama, već se i same računarske instrukcije prikazuju kao nizovi brojeva. Specijalni znaci-simboli takođe se kodiraju kao brojevi. Na početku izučavanja brojeva koji se koriste u računaru, treba najpre razmotriti strukturu brojnih sistema.

2.1. BROJNI SISTEMI

Brojni sistem predstavlja način prikazivanja bilo kog broja pomoću niza simbola koji se nazivaju cifre brojnog sistema, no, istovremeno, on predstavlja i skup pravila po kojima se realizuju osnovne operacije nad brojevima. Skup pojedinih cifara koje se mogu prikazati u računaru jedna je od komponenti njegovog brojnog sistema.

Moderni računari koriste binarni brojni sistem, koji ima samo dve cifre: **0** i **1**. Binarni sistem je izabran, jer računar mora biti sposoban da prika`e bilo koju cifru na jedinstven način, a postoji veliki broj elektronskih sklopova koji mogu da se nalaze u dva jedinstvena stabilna stanja. Ova stanja mogu biti: otvoren-zatvoren, visok-nizak, levo-desno, ili uključen-isključen. Mnogo je teže realizovati elektronske sklopove koji će imati tri, četiri ili više različitih stabilnih stanja. Sem toga binarni sistem je pogodan za predstavljanje jedne oblasti matematičke logike, poznate kao Bulova (*Boolean*) algebra, koja operiše sa binarnim brojevima i obezbeđuje teorijsku osnovu za razvoj osnovnih računarskih komponenti (digitalnih logičkih kola i mreža).

Postoje dve osnovne vrste brojnih sistema:

- pozicioni
- nepozicioni.

Ako jedna cifra ima uvek istu vrednost, bez obzira na kom se mestu u zapisu broja nalazi, onda je to **nepozicioni sistem**. Tipičan primer je rimski brojni sistem sa ciframa: **I, V, X, L, C, D, M**. Posmatrajmo rimske brojeve III i IX. Svaka cifra Rimskog sistema uvek označava jednu istu vrednost, u ovom primeru cifra I uvek ima vrednost jedan (1). U prvom primeru tri I imaju zajedno vrednost tri, a u drugom primeru I opet ima vrednost jedan samo se oduzima od X (deset), jer se manja cifra nalazi ispred veće.

Ako jedna ista cifra ima različite vrednosti, određene položajem cifre u nizu (cifara) koji predstavlja zapis broja, onda je to **pozicioni**, ili težinski brojni sistem. Posmatrajmo arapske brojeve 111, 27, i 207. Kod težinskog brojnog sistema, u prvom broju, svaka od tri jedinice ima različitu vrednost: prva s leva vredi 100 (sto), srednja 10 (deset), a desna 1 (jedan), tj. svaka pozicija ima svoju težinu. Po{to je pozicija vrlo bitna, u ovakvim sistemima nužno mora da postoji specijalna cifra NULA (0), pomoću koje se označava pozicija koja ne sadrži ni jednu cifru. Bez nule (0) između brojeva dvadeset sedam (27) i dvesta sedam (207) ne bi bilo razlike u zapisu. Svaka pozicija ima težinu, tj. vrednost koja zavisi od baze (osnove) brojnog sistema. Mi u svakodnevnom životu koristimo sistem sa osnovom 10 - dekadni brojni sistem. Pozicioni brojni sistem ima više jedinstvenih cifara, uključujući i nulu, ali ni u jednom brojnom sistemu ne može postojati pojedinačna cifra jednaka osnovi sistema.

Baza sistema jednaka je broju različitih cifara S, a same cifre brojnog sistema su: 0,1, . . . , (S-1), a u dekadnom sistemu cifre su (0,1,2,3,4,5,6,7,8, i 9).

Osnova brojnog sistema uvek se zapisuje kao složeni broj. U dekadnom sistemu to je dvocifren broj 10, tj. sastoji se od jedinice iza koje sledi cifra nula.

U opštem slučaju, proizvoljan broj x se može u brojnom pozicionom sistemu sa osnovom S predstaviti kao suma:

$$x = a_r S^r + a_{r-1} S^{r-1} + ... + a_1 S^1 + a_0 S^0 + a_{-1} S^{-1} + ... + a_{-p} S^{-p} + ...$$

gde su: $a_r , a_{r-1} , ... , a_2 , a_1 , a_0 , a_{-1} , ... , a_{-p} , ...$ cifre broja, i pripadaju skupu {0, 1, ... , S-1}, tj. pripadaju grupi od S različitih cifara.

Broj **x** se može prikazati u sažetom obliku kao niz:

$$x = a_r a_{r-1} ... a_2 a_1 a_0 . a_{-1} a_{-2} ... a_{-p}$$

gde tačka (.) (ili zapeta (,)), odvaja ceo deo od razlomljenog dela, i pri čemu svaka cifra a_i u zapisu ima neku težinu S^i, zavisno od njene pozicije **i,** računajući levo i desno od decimalne tačke.

Za unos numeričkih informacija u računar i za njihovo štampanje koriste se:

- dekadni,
- heksadekadni (heksadecimalni),
- oktalni

- binarni brojni sistemi.

U literaturi se za označavanje ovih brojnih sistema koriste skraćenice: **DEC, HEX, OCT** i **BIN**.

DEKADNI brojni sistem ima deset cifara koje uzimaju celobrojne vrednosti 0,1,2,3,4,5,6,7,8,9, a svaki broj x se može predstaviti kao:

$x = d_r\, 10^r + d_{r-1}\, 10^{r-1} + ... + d_1\, 10^1 + d_0\, 10^0 + d_{-1}\, 10^{-1} + ... + d_{-p}\, 10^{-p} + ...,$

Težinske vrednosti određene su pozicijom cifre i iznose 10^r,… 10^2, 10^1, 10^0, 10^{-1}, 10^{-2},…, 10^{-p}, a d_r,…, d_0, d_{-1},…, d_{-p} su decimalne cifre. Primeri decimalnih brojeva su:

$1992 = 1*10^3 + 9*10^2 + 9*10^1 + 2*10^0$,

$738{,}387 = 7*10^2 + 3*10^1 + 8*10^0 + 3*10^{-1} + 8*10^{-2} + 7*10^{-3}$.

HEKSADECIMALNI brojni sistem ima 16 cifara koje uzimaju celobrojne vrednosti od 0 do 15, pri čemu se za prikaz cifara od 10 do 15 koriste slova A,B,C,D,E,F. U ovom brojnom sistemu svaki broj se može napisati u obliku:

$$x = h_r\, 16^r + ... + h_1\, 16^1 + h_0\, 16^0 + h_{-1}\, 16^{-1} + ... + h_{-p}16^{-p} + ...\ ,$$

{to znači da su težinske vrednosti određene pozicijom cifre i iznose 16^r, ... , 16^2, 16^1, 16^0, 16^{-1}, 16^{-2}, ... , 16^{-p} , a h_r, ... , h_0, h_{-1}, ... , h_{-p} su heksadecimalne cifre. Primeri heksadecimalnih brojeva su:

$$12F_{16} = 1*16^2 + 2*16^1 + 15*16^0 = 303_{10}\ ,$$

$$1F9A.A_{16} = 1*16^3 + 15*16^2 + 9*16^1 + 10*16^0 + 10*16^{-1} = 8090.625_{10}.$$

OKTALNI brojni sistem ima 8 cifara 0,1,2,3,4,5,6,7, tj. osnova mu je 8, a broj se predstavlja kao:

$$x = O_r\, 8^r + ... + O_1\, 8^1 + O_0\, 8^0 + O_{-1}\, 8^{-1} + ... + O_{-p}8^{-p} + ...,$$

gde svaka cifra O_i uzima vrednost iz skupa {0,1,2,....,6,7}, a težinska vrednost joj zavisi od pozicije **i,** i iznosi 8^i, kao u sledećem primeru:

$$2057_8 = 2*8^3 + 0*8^2 + 5*8^1 + 7*8^0 = 1071_{10}.$$

BINARNI brojni sistem takođe spada u pozicione brojne sisteme, i pri tome svaki broj **x** može se prikazati samo uz pomoć dve cifre: 0 i 1, jer mu je osnova 2,

$$x = b_r\, 2^r + ... + b_1\, 2^1 + b_0\, 2^0 + b_{-1}\, 2^{-1} + ... + b_{-p}2^{-p} + ...$$

gde je: b_i ili 0 ili 1 za svako i.

Primeri binarnih brojnih zapisa su:

$$11001.1101_2 = 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 1*2^{-2} + 0*2^{-3} + 1*2^{-4} =$$
$$= 25.8125_{10}.$$

$1101011.01_2 = 107{,}25_{10}$.

2.2. KONVERZIJA BROJEVA IZ JEDNOG BROJNOG SISTEMA U DRUGI

Konverzija brojeva iz **BIN, OCT, HEX**, u dekadni brojni sistem vrši se jednostavnom operacijom sumiranja elementarnih proizvoda cifara i njihovih težinskih vrednosti, pri čemu se čitava aritmetika izvodi u dekadnom sistemu:

$$1F9A_{16} = 1*16^3 + 15*16^2 + 9*16^1 + 10*16^0 = 8090_{10}.$$

$$1267_{(8)} = 1*8^3 + 2*8^2 + 6*8^1 + 7*8^0 = 695_{10}.$$

Celi brojevi

Dekadni broj **x** konvertuje se u broj sa osnovom **S** metodom **sukcesivnih deljenja**, preko sledećeg niza koraka:

1. Broj **x** se podeli osnovom **S**. Ostatak pri deljenju pamti se kao cifra najmanje težine ($\mathbf{S}^0$) traženog broja sa osnovom S.
2. Dobijeni količnik se ponovo podeli sa **S**. Ostatak predstavlja cifru sa težinom $\mathbf{S}^1$, traženog broja sa osnovom **S**.
3. Postupak se ponavlja sve dok rezultat deljenja ne postane jednak nuli.

Postupak konverzije celog dekadnog broja u binarni broj prikazaćemo na primeru broja 120_{10}:

120 : 2 = 60 60 : 2 = 30 30 : 2 = 15 15 : 2 = 7 7 : 2 = 3 3 : 2 = 1 1 : 2 = 0

0	0	0	1	1	1	1	*ostatak*

cifra najmanje težine ← cifra najveće težine

Niz binarnih cifara treba pročitati od kraja ka početku, od poslednjeg ostatka ka prvom, a zapisuje se u jednom od sledećih oblika:

$$120_{(10)} = 1111000_{(2)},$$
$$(120)_{10} = (1111000)_2,$$
$$120_{10} = 1111000_2.$$

Dekadni broj manji od jedinice

Dekadni broj **x** manji od jedan **($x=0.a_{-1}a_{-2}$ ...)** pretvara se u odgovarajući broj sa osnovom **S** metodom **sukcesivnih množenja**, na sledeći način:

1. Pomnože se broj **x** i osnova **S**. Celobrojna vrednost ovog proizvoda predstavlja cifru sa težinom $\mathbf{S}^{-1}$ traženog broja u sistemu sa osnovom **S**. Decimalni deo proizvoda služi za dobijanje drugih cifara.
2. Druga cifra sa težinom $\mathbf{S}^{-2}$ dobija se množenjem decimalnog dela iz prethodnog množenja sa osnovom S, i predstavlja celobrojnu vrednost ovog proizvoda.
3. Postupak se ponavlja sve dok se ne dobije tražena tačnost.

Postupak nije u opštem slučaju konačan, a prekida se kada se dobije određen broj cifara kojim se može zadovoljiti tražena tačnost predstavljanja brojeva, ili kada je dobijen ceo broj (nema deo < 1). Ovo dalje znači, da tačni dekadni decimalni brojevi nemaju tačan binarni ekvivalent, pa se moraju tražiti i drugi načini predstavljanja dekadnih brojeva. Postupak sukcesivnih množenja prikazaćemo na primeru konverzije razlomljenog dekadnog broja 0.375, u binarni brojni sistem:

	celobrojni deo	
0.375 x 2 = 0,75	0	↓
0.75 x 2 = 1,5	1	
0.5 x 2 = 1,0	1	

gde je kao rezultat konverzije dobijen broj:

$(0{,}375)_{10} = (0{,}011)_2$.

Konverzija mešovitih dekadnih brojeva

Konverzija mešovitih dekadnih brojeva (ceo deo + razlomljen), vrši se tako što se ceo deo konvertuje metodom sukcesivnih deljenja, dok se razlomljen deo konvertuje metodom sukcesivnih množenja. Pri tome se ceo deo uvek pretvara u konačan niz 0 i 1, dok razlomljeni deo ne mora imati konačan prikaz, tj. njegova konverzija se ne može izvesti do kraja već sa određenom tačnošću, odnosno aproksimativno.

KONVERZIJA BIN, OCT i HEX BROJEVA U DRUGE BROJNE SISTEME

Binarni broj se konvertuje u dekadni (BIN → DEC) već opisanim postupkom sumiranja elementarnih prozvoda binarnih cifara i njihovih težinskih koeficijenata 2^i, kao u sledećem primeru:

$$1101.11_2 = 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 1*2^{-2} = 13.75_{10}.$$

Konverzija binarnog broja u oktalni i heksadecimalni

Konverzija binarnog broja u oktalni i heksadecimalni vrši se grupisanjem tri, odnosno četiri binarne cifre, počev od decimalne tačke ulevo i udesno. Kada se binarni broj konvertuje u oktalni (BIN → OCT), grupišu se po tri binarne cifre, i na osnovu tabele 1 određuje se odgovarajuća oktalna cifra. Ako pri grupisanju u krajnje levoj i krajnje desnoj grupi nema dovoljnog broja binarnih cifara, dopisuje se potreban broj nula (jedna ili dve).

Pri konvertovanju iz binarnog u heksadecimalni sistem (BIN → HEX), postupak je ekvivalentan, samo se prave grupe od po četiri binarne cifre, počev od decimalne tačke. Postupak konverzije ćemo prikazati na primeru binarnog broja $10111011011101.11111100010_2$:

$10111011011101.11111100010_2$= 010 111 011 011 101.111 111 000 10 0 =
$= 27335.7704_8$.

$10111011011101.11111100010_2$= 0010 1110 1101 1101. 1111 1100 010 0 =
$= 2EDD.FC4_{16}$.

Konverzija oktalnog i heksadecimalnog broja u binarni

Obzirom na činjenicu da je $8=2^3$, a $16=2^4$, to se svaka oktalna cifra može kodirati pomoću 3 binarne cifre, a heksadecimalna pomoću 4 binarne cifre, sa težinskim faktorima $\mathbf{P_1=2^0=1,\ P_2=2^1=2,\ P_3=2^2=4}$, a za heksadecimalne $\mathbf{P_1=2^0=1,\ P_2=2^1=2,\ P_3=2^2=4,\ P_4=2^3=8}$ **.** To znači da se oktalni broj pretvara u binarni(OCT → BIN), zamenom svake oktalne cifre njenim trocifrenim binarnim zapisom.

Heksadecimalni broj pretvara se u odgovarajući binarni (HEX → BIN), zamenom heksadacimalnih cifara njihovim četvorocifrenim binarnim ekvivalentom.

U tabeli 1 dati su binarni kodovi, oktalnih, heksadecimalnih i dekadnih cifara. Postupak konverzije ćemo prikazati na primeru oktalnog broja 157, i heksadecimalnog broja ABCD:

$(157)_8 = (001)_2\,(101)_2\,(111)_2 = 1101111_2$,
$ABCD_{(16)} = 1010_{(2)}\ 1011_{(2)}\ 1100_{(2)}\ 1101_{(2)} = 1010101111001101_2$.

Broj	Oktalna cifra	Binarni broj	Heksa cifra	Binarni broj	Dekadna cifra	Binarni broj
0	0	000	0	0000	0	0000
1	1	001	1	0001	1	0001
2	2	010	2	0010	2	0010
3	3	011	3	0011	3	0011
4	4	100	4	0100	4	0100

Broj	Oktalna cifra	Binarni broj	Heksa cifra	Binarni broj	Dekadna cifra	Binarni broj
5	5	101	5	0101	5	0101
6	6	110	6	0110	6	0110
7	7	111	7	0111	7	0111
8	10	1000	8	1000	8	1000
9			9	1001	9	1001
10			A	1010	10	
11			B	1011		
12			C	1100		
13			D	1101		
14			E	1110		
15			F	1111		
16			10			

Tabela 1. Tabela binarnih kodova heksadecimalnih, dekadnih i oktalnih cifara

Konverzija oktalnih brojeva u heksadecimalni i obratno

Konverzija oktalnih brojeva u heksadecimalne i obratno vrši se preko binarnog sistema (OCT → BIN → HEX; HEX → BIN → OCT). Najpre se svaka oktalna cifra zameni sa tri binarne, a zatim se počev od decimalne tačke ulevo i udesno grupišu po četiri binarne cifre. Svakoj grupi odgovara po jedna heksadecimalna cifra. Pri konvertovanju heksadecimalnih brojeva u oktalne, najpre se svaka heksacifra zameni sa četiri binarne (kao u tabeli), a zatim se počev od decimalne tačke grupišu po tri binarne cifre. Svakoj grupi od tri binarne cifre odgovara po jedna oktalna.

Postupak ćemo prikazati na primeru konverzije oktalnog broja 127.15_8 u heksadecimalni:

$127.15_8 = 001\ 010\ 111.001\ 101_2 = 0101\ 0111.0011\ 0100_2 = 57.34_{16}$.

2.3. POJAM KOMPLEMENTA

Komplement je pojam koji se često koristi kada se govori o brojnim sistemima. Praktični smisao ima kod prikazivanja negativnih brojeva i kod operacije oduzimanja, tačnije za realizaciju oduzimanja pomoću sabiranja. Komplementi pozitivnih brojeva su isti kao i sam taj broj. Najopštija, uprošćena definicija komplementa bi bila da je komplement dopuna datog broja do neke unapred definisane vrednosti. Iako se u svakom brojnom sistemu može definisati onoliko različitih komplemenata koliko cifara ima taj brojni sistem, od praktičnog značaja su samo komplement do broja koji predstavlja brojnu osnovu sistema i komplement do najveće cifre u sistemu. Na primer, za dekadni brojni sistem to su **komplement desetke** (jer je deset osnova brojnog sistema) i **komplement devetke** (jer je devet najveća cifra u dekadnom brojnom sistemu).

Umesto složenih definicija, pojam ova dva komplementa definisaćemo kroz primer. Posmatrajmo jedan četvorocifreni dekadni broj X (recimo 3704). **Komplement devetke** ovoga broja je ponovo četvorocifreni broj koji treba dodati ovom broju da bi se dobio četvorocifreni broj sastavljen samo od devetki (9999).
Dakle, komplement devetke broja 3704 je broj 6295 jer je:

3704	(polazni broj X)
+ 6295	(komplement devetke broja X)
9999	(zbir 9999)

Slično tome, komplement devetke broja 37 je broj 62, a komplement devetke broja 912 je broj 087.

Komplement desetke (brojne osnove sistema) broja X je definisan kao broj koji dodat broju X daje rezultat koji ima sve nule i prenos (jedinicu na mestu pete cifre za četvorocifreni broj u primeru).
Tako je komplement desetke broja X = 3704, broj 6296 jer je:

3704	(polazni broj **X**)
+ 6296	(*komplement desetke broja* **X**)
1 0000	(zbir 10000)

Slično tome, komplement desetke broja 37 je broj 63, a komplement desetke broja 912 je broj 088. Treba uočiti da, kada se četvorocifreni broj X sabere sa svojim komplementom desetke, kao četvorocifreni rezultat se dobija nula (ako se peta cifra smatra viškom koji biva odbačen u sistemima koji rade sa najviše četiri cifre). Ova osobina koja proizilazi direktno iz definicije komplementa desetke iskorišćena je za prikazivanje negativnih brojeva.
Slično komplementima desetke i devetke u dekadnom brojnom sistemu, u oktalnom sistemu su od značaja komplementi sedmice i osmice. Definisani su analogno definicijama odgovarajućih komplemenata u dekadnom brojnom sistemu kao dopuna do 7777_8 (za četvorocifreni broj), odnosno do 10000_8, respektivno. U heksadecimalnom sistemu, komplement petnaestice je dopuna do $FFFF_{16}$, a komplement šesnaestice je ponovo dopuna do 10000_{16}.
Iz primera je lako uočiti da je komplement desetke uvek za 1 veći od komplementa devetke istog broja.

Ovo je osobina koja važi za sve brojne sisteme: **Komplement brojne osnove je za jedan veći od komplementa najveće cifre**, **i ova osobina se često koristi za izračunavanje komplementa brojne osnove tako što se komplement najveće cifre (koji se dobija lako u svim brojnim sistemima) jednostavno uveća za jedan.**

Komplement brojne osnove se dobija tako što se sve nule sa desne strane broja prepišu, prva nenulta cifra sdesna komplementira se do osnove sistema, a ostale cifre komplementiraju se do najveće cifre.

KOMPLEMENTI BINARNOG BROJA

U binarnom brojnom sistemu mogu se definisati samo dva komplementa i oba su od praktičnog značaja. To su komplement jedinice (najveće cifre u sistemu) i komplement dvojke (brojne osnove sistema). Za četvorocifreni binarni broj **X** (na primer 1011_2), *komplement jedinice* je dopuna do broja 1111_2, {to za broj **X** iz primera iznosi 0100_2 jer je:

1011_2	(polazni broj **X**)
$+\ 0100_2$	(*komplement jedinice* broja **X**)
1111_2	(zbir 1111_2)

Treba primetiti da se komplement jedinice binarnog broja može dobiti tako što se svaka binarna cifra polaznog broja promeni (invertuje), pa od jedinice postaje nula, a od nule, jedinica.

Komplement dvojke je za četvorocifreni binarni broj X iz primera (1011_2), dopuna do broja 10000_2, što za broj Y iznosi 0101_2 jer je:

1011_2	(polazni broj Y)
$+\ 0101_2$	(komplement dvojke broja Y)
10000_2	(zbir 10000_2)

Komplementi u binarnom brojnom sistemu igraju veoma važnu ulogu. Verovatno je to jedan od razloga što postoji više termina za pojam komplementa jedinice i komplementa dvojke. Komplement jedinice se naziva još i inverzni kod, nepotpuni komplement ili prvi komplement. Komplement dvojke se naziva još i dopunski kod, potpuni komplement ili drugi komplement.

Iako ne odgovaraju u potpunosti definicijama, termini (prvi komplement i drugi komplement), najviše se koriste u računarskoj terminologiji. Zbog toga će se i u ovom udžbeniku nadalje uglavnom koristiti upravo ovi termini, a u tabeli 2. dato je nekoliko primera komplemenata.

X	1011_2	10100011_2	00000000_2	11111111_2	01010_2	11110_2	00000001_2
Prvi komplem	0100_2	01011100_2	11111111_2	00000000_2	10101_2	00001_2	11111110_2
Drugi komplement	0101_2	01011101_2	00000000_2	00000001_2	10110_2	00010_2	11111111_2

Tabela 2. Nekoliko primera binarnih brojeva i njihovih komplemenata

Kao i u drugim brojnim sistemima, i u binarnom važi da je komplement osnove brojnog sistema, u ovom slučaju drugi komplement, za jedan veći od komplementa najveće cifre, u ovom slučaju, prvog komplementa, što je najjednostavniji put za izračunavanje. Najpre se izračuna prvi komplement broja tako što se svaka binarna cifra polaznog broja promeni (od nule na jedinicu i obrnuto), pa se zatim dobijenom binarnom broju doda jedinica. Sabiranje sa jedinicom obavlja se po principima binarnog sabiranja opisanom u poglavlju 2.4.1. Drugi komplement negativnog binarnog broja dobija se tako što se sve nule i prva jedinica s desne strane prepišu a ostale cifre se invertuju.

Ponovimo neke važne osobine komplemenata binarnih brojeva:

- Prvi komplement se dobija invertovanjem svakog bita polaznog binarnog broja.
- Drugi komplement se dobija dodavanjem jedinice na prvi komplement.
- Prvi i drugi komplement imaju onoliko binarnih cifara koliko i polazni broj.
- Prvi i drugi komplement pozitivnih brojeva su isti kao sam taj broj.
- Drugi komplement negativne nule je ponovo nula.

2.4. BINARNI BROJNI SISTEM

Jedna binarna cifra 0 ili 1 predstavlja minimalnu količinu informacija, odnosno najmanji podatak koji se može obraditi u računaru, i naziva se bit (**bit**). Kada se posmatra binarni zapis nekog broja, prvi bit sleva je bit najveće težine **MSB** (**Most Significant Bit**), a prvi bit zdesna, bit najmanje težine **LSB** (**List Significant Bit**). U većini modernih računara koristi se grupa od osam bitova koja se naziva bajt-**byte** (1 bajt = 8 bita). Bajt predstavlja dve heksadecimalne cifre. Jedan bajt se sastoji od dve tetrade (1 **nibble** = 4 **bit**-a = 1 polubajt), od kojih svaka predstavlja jednu heksadecimalnu cifru.

Kod mikroračunara osnovni podatak koji se može smestiti u unutrašnju memoriju predstavlja jedan bajt, odnosno grupa od 8 bita. Veći računari, najčešće, memorišu podatke u grupama od 2, 4, ili više bajta, i nazivaju se memorijske reći (registar). Postoje, takođe, računari koji uopšte ne koriste bajtove. Osnovna jedinica memorisanih podataka i kod ovih računara zove se reč (**word**), pa neki računari imaju reč (registar) od 36 bita, a neki reč od 60 bita. Jedna memorijska reč je najveća količina informacija koja se u jednom ciklusu (pristupu) može pribaviti iz memorije, ili u nju smestiti.
Snaga jednog računara umnogome zavisi od dužine memorijske reči. Sa povećanjem memorijske reči povećava se brzina prenosa podataka između pojedinih delova računara.

BINARNO SABIRANJE

Binarno sabiranje je vrlo jednostavno jer postoje samo četiri pravila. Prva tri su vrlo prosta: 0+0=0, 0+1=1 i 1+0=1. Problem se javlja kod slučaja 1+1=10. Kako u jedan bit može biti zapisana samo jedna cifra, rezultat sabiranja je 1+1=0 i javlja se prenos jedne 1 u sledeći bit (bit veće težine). Kod sabiranja višecifrenih binarnih brojeva, pri sabiranju cifara treba uzeti u obzir i prenos (**carry**) iz prethodnog bita, sem kod bita najmanje težine (**LSB**). Razmotrimo sledeći problem, sabiranje dva binarna broja: $0101_{(2)} + 0111_{(2)}$. Zbir cifara najmanje težine LSB (krajnje desne cifre u zapisu broja) je $10_{(2)}$, odnosno nula sa prenosom jedan. Zbir sledećih cifara (0 i 1) je 1, ali treba dodati prenos iz LSB, pa je zbir opet $10_{(2)}$, tj. nula sa prenosom u sledeću kolonu. Zbir u trećoj koloni je: 1+1+prenos daje $11_{(2)}$,tj. zbir je 1 i prenos **C** (**carry**) je 1. Na kraju, poslednja cifra je rezultat zbira 0+0+prenos, što daje jedan, pa je konačni rezultat sabiranja $1100_{(2)}$.

2.5. BROJEVI SA ZNAKOM, KODIRANJE NEGATIVNIH BROJEVA[1]

Računar sve podatke predstavlja pomoću binarnog zapisa određene dužine. U savremenim računarima dužina binarnog zapisa je najčešće umnožak broja 8. Dakle, dužina zapisa može biti 8, 16, 32, 64... bita (mada postoje i računari sa dužinama zapisa koje nisu umnožak broja 8). Ako se koristi binarni zapis dužine osam bita, za takve podatke se kaže da su osmobitni. Kako se skup osam bita naziva bajtom, to se za osmobitne podatke kaže još i da su jednobajtni (zauzimaju jedan bajt). Shodno tome postoje i dvobajtni (šesnaestobitni), četvorobajtni (tridesetdvobitni) i tako dalje.

Jednobajtni podaci su podaci koji su zapisani binarnim zapisom dužine jedan bajt (osam bita).

[1] Delovi poglavlja 2.5 preuzeti su iz knjige "Programiranje MSC96 serije mikrokontrolera"

Dužina binarnog zapisa direktno određuje koliko se različitih podataka može zapisati. Lako je pokazati da se sa zapisom dužine N bita može zapisati 2^N različitih podataka. Ako bi se koristio samo jedan bit, pomoću njega se mogu zapisati dva podatka: Jedan koji odgovara jedinici i jedan koji odgovara nuli. Sa dva bita mogu se kodovati 4 različita podatka. Prvi odgovara kombinaciji binarnih brojeva 00, drugi kombinaciji 01 (binarno), treći kombinaciji 10 (binarno) i poslednji, četvrti, podatak kombinaciji 11 (binarno). Sa 8 bita može se zapisati ukupno $2^8 = 256$ različitih podataka. Svih raspoloživih 256 različitih podataka može se iskoristiti da se prikaže 256 pozitivnih brojeva (od 0 do 255) i tada se za takve podatke kaže da su **neoznačeni** (*unsigned*). S druge strane, ako je potrebno koristiti i pozitivne i negativne brojeve, deo raspoloživih 256 mogućnosti može se iskoristiti da se prikažu negativni brojevi. Normalno, ukupan broj različitih podataka je ponovo 256. Podaci koji sadrže i pozitivne i negativne brojeve nazivaju se **označeni** (*signed*).

Treba napomenuti da termin označeni podaci ne znači da su ti podaci negativni brojevi, već da mogu biti i pozitivni i negativni. Pozitivni brojevi se mogu predstavljati i kao označeni i kao neoznačeni, negativni, samo kao označeni. Ova dva pojma označeni podaci (*signed*) i neoznačeni podaci (*unsigned*) su vrlo karakteristična za računarsku terminologiju i provlače se kroz skoro sve oblasti vezane za obradu numeričkih podataka, pa je stoga od velike važnosti jasno ih definisati i uočiti razliku među njima.

U zavisnosti od dužine binarnog zapisa i od toga da li je potrebno predstavljati samo pozitivne (ili i pozitivne i negativne) brojeve, moguće je definisati nekoliko tipova podataka:

- Označeni bajt i neoznačeni bajt (jednobajtni, osmobitni zapis).
- Označena reč i neoznačena reč (dvobajtni, šesnaestobitni zapis).
- Označeni i neoznačeni **dvorečni** (*long*) (četvorobajtni, tridesetdvobitni zapis).

Konkretni nazivi tipova podataka variraju od jezika do jezika i od primene do primene, ali gotovo svi viši programski jezici koriste ovih šest tipova podataka definisanih na navedeni način. Kod osmobitnih (ili jednobajtnih) podataka, jednobajtni podatak može biti označen bajt ili neoznačen bajt. Pritom se pomoću tipa neoznačeni bajt može predstaviti 256 pozitivnih brojeva (0 do 255) a pomoću tipa označen bajt može predstaviti ponovo 256 različitih brojeva ali ovaj put i pozitivnih i negativnih (i to brojevi u opsegu od -128 do +127).
Nameće se jednostavno pitanje: Kako procesor razlikuje označene od neoznačenih podataka? Odgovor je još jednostavniji: NIKAKO !!! Naime, sve operacije nad jednobajtnim podacima se izvode na identičan način i procesor ni po čemu ne

razlikuje podatke tipa bajt od podataka tipa označen bajt. Može se postaviti i pitanje: Da li je moguće na neki način saopštiti procesoru da li je podatak označen ili neoznačen? Odgovor na ovo pitanje je negativan.

U asembleru, kao i u drugim mašinski-orijentisanim jezicima, ne postoji način da se određeni podatak deklariše kao označen ili neoznačen. Mašinski orijentisani jezici razlikuju podatke jedino po veličini i u njima je moguće jedino razlikovati jednobajtne (ili osmobitne), dvobajtne (ili šesnaestobitne) i četvorobajtne (ili tridesetdvobitne). Ako je već tako (procesor ne razlikuje označen bajt i bajt, niti je moguće deklarisati ih različito), čemu onda dva tipa podataka? Ko pravi razliku?
Razliku pravi programer i on je taj koji je dužan da vodi računa o tome koji jednobajtni podatak treba da bude tipa bajt, a koji tipa označen bajt. Posle svake aritmetičke operacije, procesor definiše (postavlja ili briše) indikatore-zastavice (flegove) u registru stanja[2], a programer sam mora da tumači rezultat na osnovu postavljenih zastavica. Postoje zastavice koje su značajne pri tumačenju rezultata operacija nad označenim podacima, a postoje i zastavice značajne za rad sa neoznačenim. Nezavisno od toga da li su ulazni podaci označeni ili ne, obe grupe indikatora se postavljaju / brišu, programer je taj koji treba da odluči koje će indikatore koristiti za tumačenje rezultata, a koje neće uzimati u obzir!
Razumevanje odgovora na prethodna pitannja je od izuzetne važnosti za pravilno programiranje aritmetičkih operacija. Prethodna razmatranja odnose se isključivo na princip rada samog procesora i obradu podataka korišćenjem mašinski-orijentisanih jezika.

Viši programski jezici, tačnije njihovi prevodioci preuzimaju na sebe pomenuto tumačenje rezultata pa je u njima moguće deklarisati različito označene podatke od neoznačenih. Treba napomenuti još i da neki procesori imaju različite instrukcije za označene i neoznačene podatke (na primer, instrukcije množenja i deljenja postoje i za označene i za neoznačene podatke), ali je ponovo programer taj koji mora da odluči koju od instrukcija će primeniti (zavisno od toga kako tumači ulazne podatke).

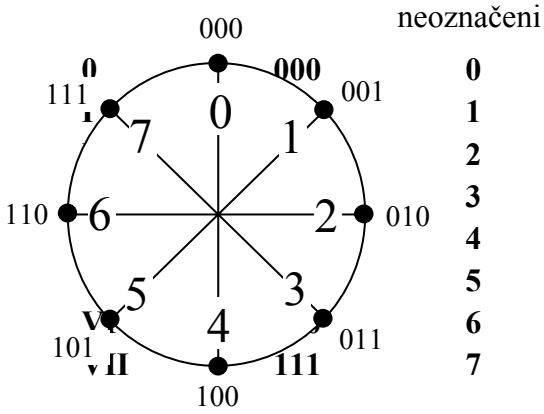
KODIRANJE NEGATIVNIH BROJEVA - DRUGI KOMPLEMENT

Pretpostavimo da podatke zapisujemo sa tri bita. Svi zaključci će važiti i za osmobitne, šesnaestobitne i tridesetdvobitne podatke. Trobitni podaci su uzeti kao primer samo zbog jednostavnosti. Pomoću N bita moguće je predstaviti 2^N različitih podataka. Prema tome, pomoću tri bita moguće je prikazati ukupno $2^3=8$ podataka. To može biti osam brojeva od 0 do 7, ali isto tako može biti i osam različitih boja, ili osam brojeva od 13 do 20. Ako želimo da radimo i sa pozitivnim

[2] Pogledati poglavlje 5.1.6

i sa negativnim brojevima, a da pri tom zadržimo i prikazivanje nule, u obzir dolaze kombinacije od -4 do 3 ili od -3 do 4.

Pogledajmo kombinacije bita, označene rimskim brojevima:



Kada se radi isključivo sa pozitivnim brojevima (neoznačeni podaci), tu uglavnom, nema nedoumica. Kombinacija označena rednim brojem I odgovara binarnom kodu broja 1, kombinacija broj VI, odgovara binarnom kodu broja 6 i tako sa svakom od osam kombinacija. Time je treća kolona definisana. Korisno je zamisliti sve brojeve raspoređene u krug kao na dijagramu sa desne strane. Naime, binarnim uvećavanjem za 1, tačka na krugu se pomera za jedno mesto udesno (u smeru kretanja kazaljke na satu). To je tačno i kada se 7 uveća za 1, u trobitnom sistemu se dobija nula kao rezultat. Sabiranje sa brojem K je u stvari pomeranje udesno po krugu za K mesta, oduzimanje, pomeranje ulevo. Tako je u trobitnom sistemu 6+3=1. Ova osobina je posledica činjenice da se binarno sabiranje obavlja po modulu 2^N gde je N broj bita (dužina) binarnog zapisa. Za sistem sa tri bita, aritmetičke operacije se obavljaju po modulu 8 što znači da će svaki rezultat R veći od 7 biti zamenjen ostatkom deljenja tog broja sa 8 (R po modulu 8). Tako se umesto 6+3=9, kao rezultat dobija ostatak deljenja 9 sa 8 (9 po modulu 8) što je 1. Ova zamena je nešto što je prirodni, sastavni deo binarne aritmetike.

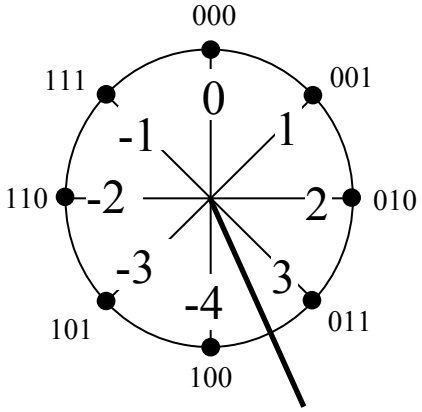
Postavlja se pitanje kako predstaviti negativne brojeve. Negativni broj je matematička fikcija. Na primer, -1 se može posmatrati kao broj koji sabran sa brojem 1 daje nulu. Pokušajmo među ovim kombinacijama da pronađemo broj koji u zbiru sa 1 daje nulu. To je kombinacija VII:

0 0 1	**(I)**
+ 1 1 1	**(VII)**
1 0 0 0	

Jedinca na mestu najznačajnijeg bita (MSB) je četvrti bit i u sistemu sa trobitnim podacima će jednostavno biti odsečena. Prema tome, binarnim sabiranjem kombinacija I i VII, dobija se nula, pa je zgodno kombinaciju VII proglasiti brojem

-1. Na taj način su dobijeni i ostali negativni brojevi u četvrtoj koloni sledeće tabele:

		Neoznačeni	Označeni
0	**000**	**0**	**0**
I	**001**	**1**	**1**
II	**010**	**2**	**2**
III	**011**	**3**	**3**
IV	**100**	**4**	**-4**
V	**101**	**5**	**-3**
VI	**110**	**6**	**-2**
VII	**111**	**7**	**-1**



Ovo predstavlja negativne brojeve u drugom komplementu. Pored ove osobine, drugi komplement ima još puno korisnih osobina koje se mogu iskoristiti kod aritmetičkih operacija tako da je postao gotovo isključiva tehnika za predstavljanje negativnih brojeva. Na kružnom dijagramu se vide mesta koja zauzimaju negativni brojevi. Prelaz preko crne podebljane linije predstavlja prekoračenje kod ovakvog tumačenja binarnog sabiranja. Ako se u toku aritmetičkih operacija pređe preko ove linije, zastavica koja označava prekoračenje se postavlja.

Drugi komplement se može dobiti dodavanjem jedinice na prvi komplement, a prvi komplement se dobija invertovanjem svih jedinica u nule i obrnuto. Tako broj -2 možemo dobiti polazeći od 2 na sledeći način:

(broj 2) 0 1 0 ➔ 1 0 1 (prvi komplement broja 2)

(prvi komplement broja 2) 1 0 1
+ 1
1 1 0 (drugi komplement broja 2=-2)

Opšti oblik zapisa označenih brojeva sa N bita bi bio sledeći:

$$- b_{N-1} \cdot 2^{N-1} + b_{N-2} \cdot 2^{N-2} + \ldots + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

Radi poređenja, opšti oblik zapisa *neoznačenih* brojeva se razlikuje samo u prvom članu:

$$b_{N-1} \cdot 2^{N-1} + b_{N-2} \cdot 2^{N-2} + \ldots + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

Za *označene* osmobitne podatke, taj opšti oblik postaje:

$$- b_7 \cdot 2^7 + b_6 \cdot 2^6 + \ldots + b_1 \cdot 2^1 + b_0 \cdot 2^0, \quad \text{ili}$$

$$- b_7 \cdot 128 + b_6 \cdot 64 + \ldots + b_1 \cdot 2 + b_0 ,$$

gde su b_7 do b_0, binarne cifre (0 ili 1) osmobitnog podatka $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$. Na primer, za broj 9 = $0000\ 1001_2$, cifre b_3 i b_0 su jedinice, ostale su nule.

Često se o najznačajnijem bitu (MSB) govori kao "bitu znaka" što je u principu tašno, ali sa takvim tumašenjem treba biti jako oprezan jer ostatak - sedam preostalih bita ne predstavljaju apsolutnu vrednost.

Iz opšteg oblika označenih brojeva mogu da se sagledaju neke važne osobine ovog načina zapisa:

- **Svi negativni brojevi imaju jedinicu kao najznačajniji bit.** Međutim, nije dovoljno samo tu jedinicu pretvoriti u nulu da bi se dobio isti pozitivan broj. Ako pođemo od broja -3, u trobitnom sistemu 101, (broj V), menjanjem prve jedinice u nulu dobijamo 001 što jeste pozitivan broj, ali 1, a ne 3! Ova činjenica se često ispušta iz vida.
- **Uopšte nije svejedno kolika je dužina zapisa označenog broja.** Naime, samo prvi član opšteg zapisa je negativan pa nije svejedno da li se najznačajniji bit množi sa 2^7, ili na primer, sa 2^2. Tako se broj -3 u trobitnom sistemu zapisuje kao 101, ali u osmobitnom sistemu broj 00000101_2 nije -3, već +5. Broj -3 kao osmobitni podatak je $1111\ 1\underline{101}_2$. Znači, kada se radi sa označenim brojevima mora se voditi računa o tome kolika je dužina podatka. To se u praksi svodi na to da označeni broj zapisan kao jednobajtni podatak, nije isti kao zapis istog broja u formatu reč ili dvostruka reč (long). Da bi se osmobitni negativni broj proširio do šesnaestobitnog, viši bajt treba napuniti jedinicama (11111111_2), a ukoliko je označen broj pozitivan, viši bajt treba da bude nula.
- **Opseg označenih osmobitnih brojeva je od –128 (10000000_2) do +127 (01111111_2) dok je opseg neoznačenih osmobitnih brojeva od 0 (00000000_2) do +255 (11111111_2).** Očigledno je da je opseg apsolutnih vrednosti označenih brojeva manji nego kod neoznačenih. Na primer, broj +129 se ne može predstaviti kao označen osmobitni broj.
- **Primenom operacije drugog komplementa nad binarnim zapisom nekog broja, tom broju se menja znak.** Podrazumeva se da se radi sa označenim podacima. Recimo, 10000001_2 je binarni zapis broja -127. Primenom operacije računanja drugog komplementa (prvi komplement +1) nad ovim binarnim zapisom dobija se 01111111_2 što je binarni zapis broja +127. Isti slučaj bi bio i ako bi se krenulo od nekog označenog pozitivnog broja, primenom drugog komplementa dobio bi se taj isti broj samo negativan. Ponovo, ovo važi samo za označene podatke.

Primeri označenih i neoznačenih brojeva

Ako se u memoriji osmobitnog računara nalazi podatak 10000001_2 koji decimalni broj je predstavljen ovim podatkom?

Da bi se dao odgovor na ovo pitanje, mora se raspolagati informacijom da li programer želi da ovo bude označen ili neoznačen broj. Ako je taj podatak za programera neoznačen, tada je to decimalni broj 129 ($1\cdot2^7+0\cdot2^6+0\cdot2^5+0\cdot2^4+0\cdot2^3+0\cdot2^2+0\cdot2^1+1\cdot2^0=128+1$), ako je broj označen onda je to -127

$(-1 \bullet 2^7 + 0 \bullet 2^6 + 0 \bullet 2^5 + 0 \bullet 2^4 + 0 \bullet 2^3 + 0 \bullet 2^2 + 0 \bullet 2^1 + 1 \bullet 2^0 = -128 + 1)$. Dakle, broj 129 i broj -127 zapisani su na potpuno isti način i operacije koje procesor izvodi nad njima, izvodi na potpuno isti način, ne znajući da li se radi o podatku 129 ili -127. Zahvaljujući osobinama drugog komplementa, moguće je da rezultat bude tačan u oba slučaja.

2. Kojim binarnim zapisom treba zapisati broj –9?

Kako se ovaj podatak može predstaviti samo kao označen broj, da bi se dao odgovor na ovo pitanje, mora se raspolagati podatkom o željenoj dužini binarnog zapisa. Najmanja dužina binarnog zapisa je 5 bita. U tom slučaju taj binarni zapis je 10111_2 zato što je $-9 = -1 \bullet 2^4 + 0 \bullet 2^3 + 1 \bullet 2^2 + 1 \bullet 2^1 + 1 \bullet 2^0$. Kako binarni zapisi manje dužine od osam bita nisu od interesa, treba pronaći osmobitni zapis. Pogrešno bi bilo kada bi rekli da je osmobitni zapis broja ovog broja 00010111_2. Zašto? Zato što bi u tom slučaju najznačajniji bit bio 0 pa bi broj bio pozitivan jer jedini negativni član opšteg izraza zapisa označenog broja je upravo član uz 2^7 (najznačajniji bit).

–9 u osmobitnom binarnom zapisu je 11110111_2.

–9 u šesnaestobitnom binarnom zapisu je 11111111 11110111_2.

Do binarnog zapisa negativnog broja može se doći na različite načine. Možda je najjednostavniji poći od apsolutne vrednosti broja (9). Apsolutna vrednost je pozitivan broj koji je lako konvertovati u binarni. Nju treba predstaviti kao binarni broj sa onoliko bita kolika se dužina binarnog zapisa traži (00001001_2). Na kraju treba izračunati drugi komplement dobijenog binarnog broja tako što se na prvi komplement (11110110_2) doda jedinica. Tako se dobija da je osmobitni zapis traženog negativnog broja (-9) binarni broj 11110111_2. *Naravno, kod binarnog zapisa negativnih brojeva nužno je voditi računa o dužini zapisa!*

Dakle: *Osmobitni binarni zapis broja 9 je broj 00001001_2. Prvi komplement ovog binarnog broja je 11110110_2 pa je drugi komplement (koji je jednak prvom komplementu uvećanom za 1) 11110111_2. Ovo je sada osmobitni binarni zapis broja -9.*

Drugi način za dobijanje binarnog zapisa negativnog broja je da se po kružnom dijagramu kakav je dat uz tabelu označenih brojeva od nule pomerimo za 9 mesta ulevo (-9 = 0-9). Za osmobitni zapis kružni dijagram ima ukupno 256 tačaka pa se to pomeranje nalevo svodi na računanje razlike 256-9 što je 247. To bi značilo da označeni broj -9 ima isti kod kao neoznačeni 247. Binarni kod za 247 se izračunava nekom od standardnih metoda. Dakle, za osmobitni zapis važi:

KOD ZA OZNAČENI -9 = KOD ZA NEOZNAČENI 247 = 11110111_2.

Ako bi se tražio šesnaestobitni zapis, odgovarajući neoznačeni broj bio bi 65536-9.

3. Koji označen broj je predstavljen zapisom 10101001_2?

Traženi broj se može izračunati direktno iz opšteg oblika zapisa označenog binarnog broja kao -128+32+8+1 = -87.

Drugi način je da se, vodeći računa da je broj negativan, izračuna drugi komplement ovog broja (komplement komplementa broja je sam taj broj) a to je 01010111_2, što je binarni kod za 87 (64+16+4+2+1). Prema tome, polazni broj je -87.

2.6. PREDSTAVLJANJE NEGATIVNIH BROJEVA DRUGIM TEHNIKAMA

Negativne brojeve moguće je predstaviti i na druge načine. U počecima razvoja korišćen je takozvani direktan kod gde je bit najveće težine (**MSB**) nosio informaciju o znaku, a ostali biti informaciju o apsolutnoj vrednosti broja. Iako je ovaj pristup blizak predstavi negativnih brojeva u dekadnom sistemu na koji su ljudi navikli, on ima neke nedostatke zbog kojih se više ne koristi u računarskoj tehnici. Osnovni nedostatak je što binarno sabiranje kakvo je implementirano u aritmetičko-logčkoj jedinici ne daje tačne rezultate ukoliko u sabiranju učestvuju negativni brojevi. Logička mreža koja se mora dodati da bi se taj problem rešio u stvari pretvara negativni broj iz direktnog koda u broj u drugom komplementu.

Prikažimo šta se događa kada hoćemo da izračunamo razliku brojeva 9 i 4, tj. 9 - 4. Po zakonima aritmetike ovaj problem se može prikazati kao 9 +(-4) tj, devet plus minus četiri. U sledećem primeru prikazano je šta se događa u računaru pri sabiranju apsolutnih vrednosti sa znakom, brojeva 9 i -4. Uočimo da je rezultat netačan (-13). Problem je, dakle: kako naći metod za prikazivanje negativnih brojeva da bi rezultati aritmetičkih operacija bili korektni?

```
 +9 =  00001001
(-4) =  10000100
      10001101  = -13
```

Druga tehnika koja je neko vreme korišćena je zapis negativnih brojeva pomoću prvog komplementa. Princip se sastoji u tome da se negativan broj predstavlja kao prvi komplemet binarnog zapisa apsolutne vrednosti tog broja. Na primer, broj -3 bi bio predstavljen tako što bi se krenulo od binarnog koda za broj 3 (00000011_2) pa bi se izračunao prvi komplement tog zapisa. Tako bi osmobitni binarni zapis za -3 korišćenjem tehnike prvog komplementa, bio 11111100_2.

Upotreba prvog komplementa je pogodna u aritmetičkim operacijama. Razmotrimo sada problem izračunavanja izraza 9-4, pošto smo ga ponovo napisali u obliku (+9)+(-4).

```
(+9) =  00001001
(-4) =  11111011
      1 00000100
```

Broj -4 u prvom komplementu dobija se inverzijom binarnih cifara broja +4. Kako je $+4_{(10)} = 00000100_{(2)}$, to nakon inverzije dobijamo $11111011_{(2)}$, a to je (-4_{10}). Analizirajmo dobijeni rezultat. Uočimo da je 9-4=5, a dobijeni rezultat nije pet. U stvari, rezultat ima čak jedan bit više od samih osnovnih brojeva. Taj dodatni bit nije bit znaka, već bit prenosa **C** (**carry**), i mora se tretirati na poseban način, tj. dodaje se na dobijeni rezultat:

```
1 00000100
      _____1
  00000101
```

Kada se u aritmetici prvog komplementa generiše bit prenosa, da bi se dobio tačan rezultat, on mora biti vraćen sa **MSB** kraja, i dodat na bit najmanje težine (**LSB**). To je ciklično vraćanje prenosa, rotacija bita prenosa (**end-around carry**), kao što je prikazano u prethodnom primeru.

U narednom primeru je pokazano šta se u aritmetici prvog komplementa događa pri oduzimanju većeg od manjeg broja, na primer, 15-20:

$+15_{(10)} = 00001111_{(2)}$ $+20_{(10)} = 00010100_{(2)}$ $-20_{(10)} = 11101011_{(2)}$

15 - 20 = (+15) + (-20)

```
+15 =     00001111
-20 =     11101011
     C=0 11111010   Pošto je C=0 ne vrši se korekcija
```

Uočimo najpre da je bit prenosa nula (pa neće biti korekcije rezultata), a da je bit znaka jedinica, odnosno rezultat je negativan, kao što i treba da bude. Da bismo videli koji je to broj, treba najpre da izvršimo konverziju negativnog broja (u prvom komplementu) u pozitivan. Konverzija negativnog broja u pozitivan, takođe se vrši inverzijom bitova.

Inverzijom cifara broja $11111010_{(2)}$ dobijamo $00000101_{(2)}$, što je jednako pet i, ne zaboravimo, rezultat je negativan, pa je dakle konačno rešenje zadatka u stvari **-5**, kao što i treba da bude. Pojavu vraćanja jedinice prenosa sa izlaza na ulaz (sa **MSB** na **LSB**) treba posmatrati kao cenu koja se mora platiti, da bi se uprostio hardver, tj, izbegla digitalna logička mreža za oduzimanje.

Dakle, kada su negativni brojevi zapisani primenom prvog komplementa, moguće je koristiti binarno sabiranje uz jednostavne korekcije. Korekcije se sastoje upravo u dodavanju jedinice (za koju se razlikuju prvi i drugi komplement) rezultatu u nekim slučajevima. Potreba te korekcije je ujedno i najveći nedostatak ove tehnike.

Drugi nedostatak je postojanje dve nule *pozitivne* i *negativne*. Naime, menjanjem znaka nuli trebalo bi da se ponovo dobije nula, što u sistemu sa prvim komplementom nije slučaj. Kako se menjanje znaka (negiranje) u ovoj tehnici svodi na računanje prvog komplementa, polazeći od zapisa nule (00000000_2)

negiranjem bi se dobio zapis 11111111_2 što bi trebalo da ponovo bude nula. Ova činjenica može donekle da oteža programiranje jer testiranje da li je broj negativan ako je rezultat bio negativna nula daje pogrešnu informaciju.

Zbog opisanih nedostataka praktično jedina tehnika zapisa negativnih celih brojeva koja se koristi je tehnika primene drugog komplementa. Korišćenjem ove tehnike binarno sabiranje daje tačne rezulte i za pozitivne i za negativne označene brojeve bez ikakvih korekcija. Pored toga, potpuno isto binarno sabiranje daje tačne rezultate i za neoznačene brojeve.

Tehnika korišćenja drugog komplementa za zapis negativnih brojeva nema ni problem dvostruke nule. Drugi komplement osmobitnog zapisa nule (00000000_2) je ($1\ 00000000_2$) što je ponovo nula u osmobitnom sistemu. Ovo odsecanje bita prenosa (devetog bita u osmobitnom sistemu), koje je inače "prirodno" i diktirano hardverom, upravo omogućava da sabiranje bude jedinstveno i za označene (bilo pozitivne ili negativne) i za neoznačene brojeve.

Treba napomenuti da je informacija o odsečenom devetom bitu dostupna programeru kroz zastavicu (fleg) za prenos **C** i programer je može koristiti za tumačenje dobijenog rezultata kada je to potrebno.

ANALIZA REZULTATA SABIRANJA – ZASTAVICE

Posle svake aritmetičke operacije, procesor vrši analizu dobijenog rezultata i postavlja nekoliko zastavica[1] (flegova) koje u stvari pokazuju stanje aritmetičko-logičke jedinice posle obavljene aritmetičke operacije. Tih zastavica ima više i razlikuju se od procesora do procesora, ali postoji nekoliko koje su zastupljene u skoro svim procesorima i čije su definicije gotovo jedinstvene. Zastavice su dostupne programeru i njihovo tumačenje čini nerazdvojni deo svake aritmetičke operacije. Programer (u programu) na osnovu stanja zastavica može tumačiti rezultat na različite načine. Postoje naredbe koje granaju program zavisno od stanja svake zastavice pojedinačno ili kombinacije stanja zastavica, tako da zavisno od toga kakvo je stanje **ALU,** procesor može nastaviti da izvršava različite naredbe, pa čak i programe (podprograme).

Sve definicije zastavica koje slede date su pod pretpostavkom osmobitne aritmetičko-logičke jedinice. Ako ALU može da obavlja i dvobajtne ili četvorobajtne operacije, za svaku dužinu podatka postoji analogna definicija zastavice i stanje zastavice će se određivati različito za različite dužine. Zastavice koje su trenutno od interesa su sledeće:

[1] Zastavice su obrađene detaljnije u poglavlju 5.1.6, a u ovom poglavlju biće pomenute samo one koje su u neposrednoj vezi sa načinom predstavljanja označenih i neoznačenih podataka kao i sa operacijom binarnog sabiranja.

C (od ***C**arry*) koja označava prenos iz bita najveće težine. Ova zastavica *posle svake aritmetičke operacije* dobija vrednost ili nula (uobičajen termin - *briše se* (**Clear**)) ili jedan (uobičajen termin - *postavlja se* (**Set**)).

- **C** će biti nula ako u toku aritmetičke operacije nije došlo do prenosa u deveti bit (nepostojeći deveti bit bi bio nula).
- **C** će biti jedinica ako je u toku aritmetičke operacije došlo do prenosa u deveti bit (nepostojeći deveti bit bi bio jedinica).

N (od ***N**egative*) koja označava negativan rezultat pod pretpostavkom da su ulazni podaci označeni brojevi. I ova zastavica posle svake aritmetičke operacije dobija vrednost ili nula ili jedan.

- ***N** će biti nula ako je aritmetička operacija dala rezultat koji je negativan broj ukoliko se ulazni podaci tretiraju kao označeni brojevi.*
- ***N** će biti jedinica ako je aritmetička operacija dala rezultat koji je pozitivan broj ukoliko se ulazni podaci tretiraju kao označeni brojevi.*

V (od *o**V**erflow*) koja označava da je rezultat van opsega označenog bajta (-128 do +127) pod pretpostavkom da su ulazni podaci označeni brojevi. I ova zastavica posle svake aritmetičke operacije dobija vrednost ili nula ili jedan.

- ***V** će biti nula ako je aritmetička operacija dala rezultat koji je u opsegu -128 do +127 ukoliko se ulazni podaci tretiraju kao označeni brojevi.*
- ***V** će biti jedinica ako je aritmetička operacija dala rezultat koji je van opsega -128 do +127 ukoliko se ulazni podaci tretiraju kao označeni brojevi.*

Posmatrajmo dva slučaja sabiranja u osmobitnoj ALU:

- U prvom slučaju, saberimo brojeve 1 i 253. Rezultat koji očekujemo je svakako 254. Ovo sabiranje označićemo rimskim brojem **I**.
- U drugom slučaju, saberimo brojeve 1 i -3. Rezultat koji očekujemo je svakako -2. Ovo sabiranje označićemo rimskim brojem **II**.

Posmatrajmo sada binarne kodove ova dva sabiranja:

- **I** slučaj

0000 0001	1
+1111 1101	+253
1111 1110	254

- **II** slučaj

0000 0001	1
+1111 1101	-3
1111 1110	-2

Ako pogledamo binarne oblike podataka (koje procesor jedino razume), primetićemo da su oba sabiranja identična, *kako po ulaznim podacima, tako i po rezultatu*. Rezultat je u oba slučaja 1111 1110_2.

Da li je to 254 ili -2? Odgovor je u oba slučaja DA.

Upravo tako, rezultat je i 254 i -2, kako ko voli. Tačnije, ako programer želi da ulazni podaci budu označeni (1 i -3), rezultat treba da tumači kao označen broj (a tada je 1111 1110_2 zaista -2), a ako želi da ulazni podaci budu neoznačeni (1 i 253), rezultat treba da tumači kao neoznačen broj (a tada je 1111 1110_2 zaista 254). O programerovim željama, procesor niti ima, niti može da ima pojma! Znači, jedini koji može da napravi razliku između slučaja I i II je programer. Naravno u višim programskim jezicima o tome brine prevodilac. Postavqa se pitanje koji će se flegovi setovati u primeru I, a koji u primeru II. Posmatra}emo samo zastavicu prekoračenja (V) i zastavicu koji pokazuje da je rezultat negativan (N zastavica), ostale nisu problematične.

- I slučaj: Izgleda da ima prekoračenja (rezultat 254 je van opsega -128 do 127) i da bi V zastavica trebalo da bude setovana. Rezultat je pozitivan (+254), pa izgleda da N zastavica ne bi trebalo da bude postavqea. **N E T A Č N O !**
- II slučaj: Izgleda da nema prekoračenja (rezultat -2 je u opsegu -128 do 127) i da bi V zastavica trebalo da bude obrisana. Rezultat je negativan (-2), pa izgleda da bi N zastavica trebalo da bude postavljena. **T A Č N O !**

Međutim, teško je očekivati od procesora, ma kako verovali u njegove sposobnosti, da od istih ulaznih podataka, koji daju isti rezultat, različito postavlja zastavice u zavisnosti od želje programera. Zbog toga će zastavice V i N biti uvek određene uz pretpostavku da programer podatke tumači kao označene, znači, kao u slučaju **II**. Ako programer radi sa neoznačenim brojevima, prekoračenje može da proveri testiranjem C zastavice, a ne testiranjem V zastavice (V zastavica tada nema nikakav smisao). Ako radi sa označenim podacima, V zastavica je ta koji nosi informaciju o prekoračenju, a ne C zastavica. Informacija o tome da je broj negativan, kada programer radi samo sa pozitivnim brojevima nije potrebna, jer ni jedan podatak programer neće tumačiti kao negativan broj. Negativan rezultat posle oduzimanja dva neoznalena broja, biče naznalen prekoralenjem (opet pomoću C zastavice).

Ovo razmatranje važi isključivo za osmobitni zapis.

Sve ulazne podatke (kao i izlazni) bi trebalo tumačiti na isti način, ili kao označene ili kao neoznačene. Kombinacije označenih i neoznačenih ulaznih podataka daju rezultate koje je teško interpretirati i takvo programiranje nosi veliku opasnost od grešaka koje mogu biti veoma neprijatne i problematične za otkrivanje.

Sabiranje radi potpuno isto i za označene i neoznačene podatke i daje tačan označeni rezultat kada su ulazni podaci označeni, a tačan neoznačeni rezultat kada su ulazni podaci neoznačeni. Ovo važi dok su i ulazni i izlazni podaci u dozvoljenom opsegu.

2.7. BROJEVI SA NEPOKRETNOM TAČKOM

Snagu jednog računara u mnogome određuju i brojevi nad kojima se može vršiti obrada. Na izbor vrste brojeva koje ćemo u programu koristiti utiču sledeći činioci:

1. *vrste brojeva koji se mogu prikazati u računaru, tj. celi, realni i kompleksni brojevi,*
2. *opseg vrednosti koje ti brojevi mogu imati,*
3. *tačnost prikazivanja brojeva,*
4. *cena hardvera koji je potreban za pamćenje i obradu brojeva.*

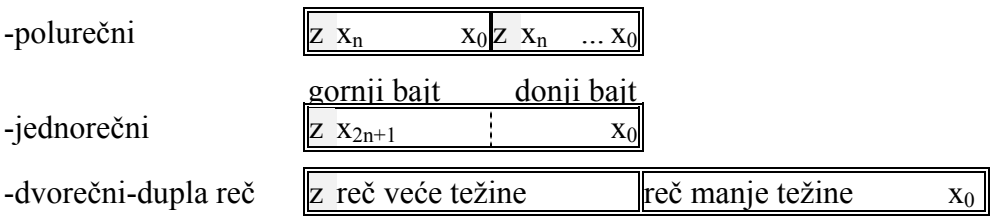
Mi smo do sada razmatrali samo kako se u računaru prikazuju celi binarni brojevi, ali i decimalni (realni) brojevi se vrlo često koriste pre svega za prikazivanje vrlo malih brojeva (čija je apsolutna vrednost oko nule) i vrlo velikih brojeva.

Neka je binarni broj **x** dat u obliku: $x = x_p\, x_{p-1}...x_1\, x_0\, .\, x_{-1}\, x_{-2}\, ...x_{-k}$, gde su x_i, i = -k....-2,-1,0,1,....., (p-1) binarne cifre broja x, a x_p znak broja. Svaka binarna cifra ima određeno mesto u mašinskoj reči, a to mesto se zove razred ili ćelija. Prema tome, binarni brojevi u računaru imaju konačnu dužinu.

Broj x je predstavljen u nepokretnom zarezu tako što se položaj zareza (ili decimalne tačke) utvrđuje na određenom mestu u odnosu na razrede broja i ostaje neizmenjen za sve brojeve prikazane mašinskom reči. Ako se zapeta utvrdi iza krajnje levog razreda koji je u mašinskoj reči označen kao pozicija bita najveće težine (iza **MSB**), onda su svi memorisani brojevi po modulu manji od jedinice.

Ako se zapeta fiksira iza pozicije razreda koji sadrži bit najmanje težine onda su svi memorisani brojevi iz skupa celih brojeva, tj. celi brojevi su poseban slučaj brojeva u nepokretnom zarezu. Sve što je rečeno o celim brojevima odnosi se i na brojeve sa nepokretnom tačkom.

Zavisno od toga koliko mašinskih reči se koristi za predstavljanje numeričkih podataka, brojevi nepokretnom tačkom se mogu predstaviti kao:



Dužina broja neposredno određuje tačnost njegovog prikazivanja (broj značajnih cifara), ali i potreban memorijski prostor za njegovo zapisivanje. Bez obzira na dužinu zapisa u binarnu poziciju (*ćeliju-razred*) najveće težine (**MSB**) uvek se mora memorisati i znak.

Polurečni format omogućava smećtanje dva binarna broja (sa znakom) u jedan registar u memoriji. Služi najčešće za zapis celih brojeva.

Jednorečni format omogućava smeštanje jednog numeričkog podatka u jedan registar u memoriji.

Dvorečni format omogućava zapis jednog numeričkog podatka u dva registra pri čemu je u prvoj reči binarna pozicija najveće važnosti rezervisana za znak čitavog broja, a druga reč predstavlja neoznačen numerički podatak.

Zapisivanje razlomljenih, necelih, brojeva pomoću fiksne zapete je tehnika koja se puno primenjuje naročito u računarima gde je brzina obavljanja aritmetičkih operacija od presudne važnosti. Osim u računarima čiji procesori rade direktno sa brojevima u pokretnom zarezu, ili imaju poseban hardverski sklop (koprocesor) za podršku pokretnog zareza, sve aritmetičke funkcije se obavljaju koristeći celobrojnu aritmetiku.

Osnovni princip se sastoji u tome da se razlomljeni broj zameni celim brojem tako što će se pomnožiti konstantom oblika 2^N. Kada se to uradi, celokupna aritmetika se obavlja koristeći binarne aritmetičke operacije aritmetičko-logičke jedinice kao da se radi o celim brojevima. Dakle, sve aritmetičke operacije obavlja hardver i nisu potrebni posebni programi kao što je to slučaj u pokretnom zarezu.

Neoznačeni brojevi predstavljeni pomoću fiksne zapete

Posmatrajmo najpre samo neoznačene (pozitivne) brojeve. Kako bi u binarni oblik pretvorili broj 5,5? To je broj $101,1_2$ ali kako ovaj binarni broj zapisati? Kako bi njegov zapis izgledao u memoriji, na primer, osmobitnog računara? Decimalni zarez je, na žalost nemoguće zapisati u memoriji. Preostaje jedino mogućnost da se zapišu samo binarne cifre a da se informacija o položaju decimalnog zareza sačuva na neki drugi način. Kada se radi o zapisu razlomljenih decimalnih brojeva pomoću fiksnog zareza, informaciju o položaju decimalnog zareza čuva programer i on je taj koji vodi računa o položaju zareza u rezultatu.

Da ponovimo, u zapisu pomoću fiksnog zareza informaciju o položaju zareza čuva programer, a zarez se ne upisuje u memoriju (tačnije, ne upisuje se njegov položaj).

Dužina binarnog zapisa (broj bita) i položaj decimalnog zareza definišu format zapisa. U primeru broja 5,5 format zapisa ovog broja određen je podacima da je zapis osmobitni i da je zarez između bita 0 i bita 1.

Format je informacija koja nužno ide uz svaki podatak pojedinačno kada se koristi zapis pomoću fiksnog zareza. Informaciju o formatu čuva programer. Dakle, u

osmobitnom zapisu broj 5,5 bi izgledao 00001011_2, a programer bi morao da pamti da se decimalni zarez nalazi između bita 0 i bita 1.

Primetimo na ovom mestu da je pomenuti binarni broj (00001011_2) u stvari binarni kod broja 11 i da jedino programer može da zna da li je zapisani broj 5,5 ili 11. Broj 11 je dvostruka vrednost broja 5,5 i programer sve vreme mora voditi računa o tome da će u aritmetičkim operacijama učestvovati broj 11, a ne 5,5. Posle obavljenih aritmetičkih operacija, programer mora da koriguje rezultat.

Na primer ako treba broj 5,5 pomnožiti sa 3, računar će množiti 00001011_2 (što je u stvari 11) sa 3 i dobiti 33. Programer mora znati da je prvi ulazni podatak u formatu fiksnog zareza sa zarezom između bita nula i bita 1 (što praktično znači da je uvećan 2 puta) i da je shodno tome i rezultat u istom formatu, sa decimalnim zarezom između bita 0 i bita 1 pa je i on dva puta veći.

Da bi se programeru olakšalo pamćenje formata u kome su pojedini podaci zapisani, uvedene su standardizovane oznake za oznaku položaja decimalnog zareza (formata). Takođe su definisana formalna pravila koja određuju u kom formatu je rezultat, ako se znaju formati ulaznih podataka.

Za oznaku formata koristi se oblik: $Q_{N-R,\,R}$

Pri tom je N-R broj binarnih cifara ispred decimalnog zareza, a R broj binarnih cifara iza decimalnog zaraza. Broj R se naziva "**radix**" i taj termin je odomaćen i u literaturi na srpskom jeziku. N određuje dužinu binarnog zapisa (ukupan broj bita).

Kada je $R = 0$, zapis je celobrojni, a kada je $R = N$, za zapis se kaže da je normalizovan. Pomoću takvog zapisa se mogu prikazati pozitivni brojevi manji od jedinice ($0 \leq \text{broj} < 1$).

Pravila koja određuju formate rezultata za sabiranje i oduzimanje svode se na to da se mogu sabirati ili oduzimati jedino brojevi u istom formatu i da je rezultat koji se dobije ponovo u tom formatu.

Format rezultata množenja i deljenja je određen sledećim definicijama:

- Broj u formatu $Q_{A,B}$ • broj u formatu $Q_{C,D}$ dobija se broj u formatu $Q_{A+C,B+D}$
- Broj u formatu $Q_{A,B}$ / broj u formatu $Q_{C,D}$ dobija se broj u formatu $Q_{A-C,B-D}$

Problem eventualnih negativnih vrednosti radiksa (A-C) ili broja binarnih cifara pre zareza (B-D) (nastalih kod deljenja) rešava se jednostavnom korekcijom. Na primer, ako se posle deljenja dobio format rezultata $Q_{-1,9}$ on se može posmatrati kao da je broj u formatu $Q_{0,8}$ samo ga treba pomnožiti sa 2^{-1}. Detalji vezani za formate prilikom deljenja prevazilaze nivo ovog kursa.

Na pitanje koji je osmobitni binarni zapis broja 5,5 iz primera, precizan i potpun odgovor bi bio:

$5{,}5 = 00001011_2$ u formatu $\boldsymbol{Q_{7,1}}$

Iz primera se vidi da sam osmobitni binarni zapis (bez naznake formata) nije dovoljan jer isti binarni zapis može da odgovara velikom broju brojeva. Već smo videli da to može biti broj 11 ako je ceo broj (u formatu $Q_{8,0}$), a može odgovarati i broju 1.375 ako je zapisan u formatu $Q_{6,2}$. Sledeći primer prikazuje nekoliko različitih vrednosti koje mogu odgovarati istom zapisu, kao i više različitih zapisa iste vrednosti:

00001011_2 može biti:

11	u formatu $Q_{8,0}$
5,5 ($11/2^1$)	u formatu $Q_{7,1}$
2,75 ($11/2^2$)	u formatu $Q_{6,2}$
1,375 ($11/2^3$)	u formatu $Q_{5,3}$
0,6875 ($11/2^4$)	u formatu $Q_{4,4}$
0,34375 ($11/2^5$)	u formatu $Q_{3,5}$

S druge strane, isti broj 5,5 može se zapisati kao:

00001011_2	u formatu $Q_{7,1}$
00010110_2	u formatu $Q_{6,2}$
00101100_2	u formatu $Q_{5,3}$
01011000_2	u formatu $Q_{4,4}$
10110000_2	u formatu $Q_{3,5}$

Iz primera se može uočiti da se isti broj mo`e napisati u vi{e formata a da se pretvarawe iz jednog formata u drugi koji ima radiks (broj binarnih cifara iza zapete, odnosno decimalne tačke) za jedan veći, vrši tako što se binarni zapis pomeri za jedno mesto ulevo (što odgovara množenju sa 2). Dalje povećavanje radiksa posle formata $Q_{3,5}$ bi dovelo do gubitka informacije (jedinica na mestu najznačajnijeg bita bi otpala). Treba napomenuti da govorimo isključivo o neoznačenim brojevima, Format $Q_{3,5}$ se ne bi smeo koristiti da je reč o označenim brojevima (jer bi u tom slučaju 10110000_2 bio negativan broj).

Formatom je određen broj cifara ispred zareza i broj cifara iza zareza.

- Broj cifara ispred zareza definiše opseg brojeva koji se mogu tim formatom predstaviti.
- Broj cifara iza zareza definiše tačnost sa kojom se brojevi prikazuju.

Zbir ova dva broja (broj binarnih cifara ispred i iza) je jednak dužini zapisa koja je najčešće fiksna i određena hardverom, tako da je programer pri određivanju formata u kome će zapisati neki promenljivi podatak prinuđen da trguje između što

veće tačnosti i što većeg opsega. Prilikom aritmetičkih operacija programer može da menja formate istog podatka u želji da poveća tačnost ili opseg. Operacija menjanja formata koja se svodi na množenje ili deljenje sa 2^N, tačnije na pomeranje (šiftovanje) podatka za N mesta ulevo ili udesno, je nešto što se stalno primenjuje gotovo pri svakoj složenijoj aritmetičkoj operaciji.

Kada su u pitanju konstantni podaci, uvek postoji jedan optimalni format. To je onaj koji ima dovoljan broj binarnih cifara ispred zareza da se zapiše celobrojni deo konstante. Tako iza zareza ostaju svi bitovi do zadate dužine zapisa čime se omogućava maksimalna tačnost. Na primer: $3,6 = 11,1001100110011001..._2$

Očigledno je da je za zapis tačne vrednosti ove konstante potreban beskonačan broj binarnih cifara. Kako je broj binarnih cifara zapisa ograničen (recimo na 8), treba pokušati da što veći broj binarnih cifara posvetimo delu iza zareza kako bi ostvarili što veću tačnost. Međutim, da bi zapisali celobrojni deo neophodna su dva bita pa je broj bita ispred zareza minimalno 2. Dakle, optimalni zapis ove konstante bio bi u formatu $Q_{2,6}$. Normalno, u obzir dolaze i formati $Q_{3,5}$, $Q_{4,4}$, $Q_{5,3}$… ali je u njima tačnost predstavljanja sve manja i manja.

$3,6 = 11100110_2$ u formatu $Q_{2,6}$.

Da bi bili do kraja precizni, ovo je najpribližniji zapis broju 3,6. Broj koji smo zapisali nije 3,6 već je:

3,59375 (11,100110= 2+1+0.5+0+0+0.0625+0.03125+0).

Ako bi celobrojni deo bio veći, na primer tražimo zapis broja 129,6 tada bi zapis morao da ima čak sedam bita ispred zareza, dakle bio bi nužan format $Q_{7,1}$. Broj $129,6 = 10000011_2$ u formatu $Q_{7,1}$. Umesto sa 129,6 tada bi računar raspolagao podatkom $100001,1_2$ što je 129,5, ali to je najboqe što možemo dobiti u osmobitnom zapisu.

Sa konstantama, određivanje optimalnog formata je prilično jasno. Pitanje je koji format usvojiti za neku promenljivu koja se, na primer, menja u granicama od 3 do 129. Očigledno je da bi bilo nužno ostaviti dovoljno bita ispred zareza za zapis maksimalne vrednosti $Q_{7,1}$, ali tada je tačnost zapisa mala kada promenljiva dobije vrednosti bliske donjoj granici. To se u praksi rešava stalnim menjanjem formata što nas dovodi do potrebe za uvođenjem formata pokretnog zareza gde svaki podatak u memoriji sadrži i zapis o položaju zareza i sve aritmetičke operacije se obavljaju uzimajući u obzir obe informacije.

Opšti oblik osmobitnog zapisa neoznačenog broja u formatu $Q_{8-R,\,R}$ je

$$\mathbf{2^{-R}\bullet(\ b_7\bullet 128 + b_6\bullet\ 64 + \ldots + b_1\bullet 2 + b_0)}$$

dakle, razlikuje se od opšteg oblika osmobitnog celog broja jedino po tome što je pomnožen konstantom $\mathbf{2^{-R}}$, (ili podeljen konstantom $\mathbf{2^R}$). Na primeru broja 3,6 u

formatu $Q_{2,6}$ može se pokazati značenje ovog opšteg oblika. U ovom slučaju radiks (R) je 6 pa je:

3,6 = 1 1 1 0 0 1 1 0 $_2$ u formatu $Q_{2,6}$.

3,6 = 2^{-6}•(1•128 + 1•64 + 1•32 + 0•16 + 0•8 + 1•4 + 1•2 + 0•1)

To bi značilo da "doprinos" najznačajnijeg bita više nije 128 već 2 ($128/2^6$) i tako sa svakim bitom. Celokupno razmatranje odnosilo se isključivo na neoznačene brojeve.

Označeni brojevi predstavljeni pomoću fiksnog zareza

Predstavljanje označenih brojeva u fiksnom zarezu se u principu ne razlikuje od predstavljanja neoznačenih. Brojevi predstavljeni pomoću fiksnog zareza su u stvari celi brojevi, pa sve što važi za označene cele brojeve važi i za označene brojeve predstavljene pomoću fiksnog zareza. Da bi se naznačilo da je u pitanju označeni broj, neki autori obeležavaju formate označenih brojeva simbolom Q^s. Tako oznaka formata $Q^s_{3,5}$ znači da se radi o označenom broju zapisanom u formatu $Q_{3,5}$.

Kada je u pitanju određivanje optimalnog formata konstante, u slučaju označenih brojeva treba voditi računa o tome da minimalna dužina zapisa označenog broja zahteva jedan bit više u odnosu na minimalnu dužinu zapisa neoznačenog broja iste apsolutne vrednosti. Tako je u formatu $Q^s_{3,5}$, maksimalna celobrojna vrednost označenih brojeva je 3, a ne 7 kao što je to slučaj kod neoznačenih.

Na primeru ovog formata pokazaćemo opseg označenih i neoznačenih brojeva

$Q^s_{3,5}$ od - 4 (10000000_2) do 3,96875 (01111111_2) OZNAČENI

$Q_{3,5}$ od 0 (00000000_2) do 7,96875 (11111111_2) NEOZNAČENI

Tabela 3. pokazuje najmanji i najveći (po apsolutnoj vrednosti) pozitivni i negativni broj različit od nule. Tabela se odnosi na format $Q^s_{3,5}$ označenih brojeva:

NEGATIVNI		POZITIVNI	
najmanji po aps. vrednosti	**najveći po aps. vrednosti**	**najmanji po aps. vrednosti**	**najveći po aps. vrednosti**
- 0,03125 (11111111_2)	- 4 (10000000_2)	0,03125 (00000001_2)	3,96875 (01111111_2)

Tabela 3. Najmanji i najveći pozitivni i negativni u formatu $Q^s_{3,5}$

Iz ovih opsega proizilazi da optimalni zapis konstante 3,6 iz prethodnog primera neće moći da bude u formatu $Q_{2,6}$ ako hoćemo da bude zapisana kao označen broj, jer bi zapis u ovom formatu (11100110_2) u slučaju označenih brojeva predstavljao negativan broj.

$3,6 = 11100110_2$ u formatu $Q_{2,6}$ kao NEOZNAČENI BROJ

$3,6 = 01110011_2$ u formatu $Q^s_{3,5}$ kao OZNAČENI BROJ

Da napomenemo još jednom, pozitivni brojevi se mogu predstavljati i kao označeni (ako treba da učestvuju u operacijama sa drugim brojevima koji mogu biti negativni) i kao neoznačeni (ako su svi učesnici u svim aritmetičkim operacijama isključivo pozitivni brojevi).

Opšti oblik osmobitnog zapisa označenog broja u formatu $Q^s_{8-R,\,R}$ je

$$2^{-R} \bullet (-b_7 \bullet 128 + b_6 \bullet 64 + \ldots + b_1 \bullet 2 + b_0)$$

dakle, razlikuje se od opšteg oblika označenog osmobitnog celog broja jedino po tome što je pomnožen konstantom 2^{-R}, (ili podeljen konstantom 2^R). Na primeru broja 3,6 u formatu $Q^s_{3,5}$ može se pokazati značenje ovog opšteg oblika. U ovom slučaju radiks (R) je 5 pa je:

3,6 = 0 1 1 1 0 0 1 1_2 u formatu $Q^s_{3,5}$

$$3,6 = 2^{-5} \bullet (-0 \bullet 128 + 1 \bullet 64 + 1 \bullet 32 + 1 \bullet 16 + 0 \bullet 8 + 0 \bullet 4 + 1 \bullet 2 + 1 \bullet 1)$$

Najčešće korišćeni formati i primeri

U praksi se koriste različiti formati za zapis brojeva u pokretnom zarezu. Osmobitni zapis se za većinu primena pokazuje kao nedovoljan pa se mnogo češće koristi dvobajtni ili četvorobajtni zapis. Kako je stalno konvertovanje formata mukotrpan posao i kako uvek postoji rizik da je usvojeni format nedovoljan da prihvati rezultat, to se često pribegava metodu da se svi podaci normalizuju. Tačnije, brojevi se predstave kao veličine između nule i jedinice po apsolutnoj vrednosti tako da se što više aritmetičkih operacija obavi koristeći taj format. To čini da se format $Q_{0,16}$ za neoznačene brojeve, i format $Q^s_{1,15}$ za označene, koriste nešto više od drugih formata.

U formatu $Q_{0,16}$ za neoznačene brojeve, najmanji broj različit od nule je 1/65536 što je oko $1,53 \bullet 10^{-5}$ dok je najveći broj za toliko manji od jedinice (oko 0,9999847). U formatu $Q^s_{1,15}$ za označene brojeve, najmanji pozitivni broj različit od nule je 1/32768 što je oko $3,05 \bullet 10^{-5}$ dok je najveći pozitivni broj oko 0,9999695.

Najmanji negativni broj po apsolutnoj vrednosti je -1/32768 (oko $-3,05 \bullet 10^{-5}$) dok je po apsolutnoj vrednosti, najveći negativni broj tačno -1.

1. Kako u osmobitnom zapisu zapisati broj π ($3,1416_{10}$)?

Pretvaranjem u binarni zapis $3,1416_{10} = 11,00100100001111\ldots_2$ Pre no što odredimo koji bi format zapisa bio optimalan, moramo znati da li želimo da ovaj broj zapišemo kao označen ili kao neoznačen broj. Optimalni zapis je sledeći:

$3,1416 = 11001001_2$ u formatu $Q_{2,6}$ kao NEOZNAČENI BROJ

$3{,}1416 = 01100100_2$ u formatu $Q^s_{3,5}$ kao OZNAČENI BROJ

Treba primetiti da je broj sa kojim će računar raditi ako se koristi format $Q^s_{3,5}$ u stvari 3,125, ali to je najpribližnije što može da se dobije pomoću osmobitnog zapisa. Kod većih konstanti, greške su u principu još veće jer ostaje manji broj binarnih cifara iza zareza.

2. Kako u osmobitnom zapisu zapisati broj - π (-3,1416)?

Ako pođemo od apsolutne vrednosti (X) negativnog broja (-X), i tu apsolutnu vrednost zapišemo u formatu označenog broja, drugi komplement tog broja će biti binarni zapis broja -X Pretvaranjem u binarni zapis apsolutne vrednosti $3{,}1416 = 11{,}001001000011111\ldots_2$. Negativni broj se može predstaviti samo kao označen pa ćemo zato i ovu apsolutnu vrednost zapisati u formatu označenih brojeva:

$3{,}1416 = 01100100_2$ u formatu $Q^s_{3,5}$ kao OZNAČENI BROJ

Prvi komplement ovog broja je 10011011_2 a drugi komplement se dobija dodavanjem jedinice 10011100_2. Prema tome,

$-3{,}1416 = 10011100_2$ formatu $Q^s_{3,5}$

Drugi način za dobijanje binarnog zapisa negativnog broja je da se pronađe ekvivalentni neoznačeni broj koji ima isti kod. Taj broj je za osmobitni_zapis 256-3,1416=252.8584. Kod za ovaj neoznačeni broj se može naći na već opisani način:

KOD ZA OZNAČENI -3,1416=KOD ZA NEOZNAČENI $252.8584=10011100_2$ formatu $\boldsymbol{Q^s_{3,5}}$

Ovaj princip je objašnjen u primerima pretvaranja negativnih celih brojeva, a važi i za razlomljene zapisane pomoću fiksnog zareza jer su oni u stvari zapisani kao celi brojevi. Ako bi se tražio šesnaestobitni zapis, odgovarajući neoznačeni broj bio bi 65536-3,1416.

3. *Koji heksadecimalni broj se nalazi u memoriji šesnaestobitnog računara kao zapis konstanti 0.75_{10} i -0.75_{10} ako se zna da su zapisane kao označeni brojevi u formatu $Q^s_{1,15}$?*

Pretvaranjem u binarni zapis $0{,}75_{10} = 0{,}11_2$. Prema tome,

$0{,}75_{10} = 0110\ 0000\ 0000\ 0000_2$ u formatu $Q^s_{1,15}$.

To znači da heksadecimalni broj 6000_{16} predstavlja zapis broja 0,75 u formatu $Q^s_{1,15}$. Zapis broja -0,75 se dobija računanjem drugog komplementa zapisa broja 0,75 u istom formatu. Prvi komplement je $1001\ 1111\ 1111\ 1111_2$ pa je drugi komplement $1010\ 0000\ 0000\ 0000_2$. Prema tome broj je:

$-0{,}75_{10} = 1010\ 0000\ 0000\ 0000_2$ u formatu $Q^s_{1,15}$.

To znači da heksadecimalni broj $\mathbf{A000_{16}}$ predstavlja zapis broja -0,75 u formatu $Q^{s}_{1,15}$.

2.8. BROJEVI SA POKRETNOM TAČKOM

Kada memorišemo realne brojeve u računaru najčešće se koristi predstava pomoću pokretne tačke. Broj R se može izrazili u obliku pokretne tačke kao

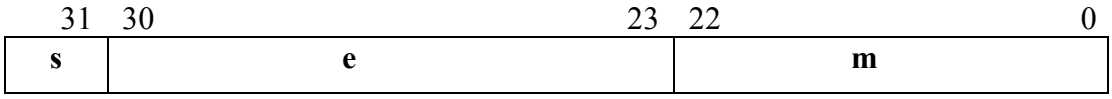
$$R = (-1)^{s} \cdot M \cdot 2^{E},$$

gde je s bit za znak broja, M mantisa, a E eksponent (pozitivan ili negativan ceo broj). Ovaj broj može biti memorisan u lokaciji sa tri polja: znak s, mantisa M i eksponent E. Za svako R postoji mnogo ovakvih predstava, pa se skoro isključivo koristi tzv. normalizovani oblik broja u kome mantisa M ima standardni oblik O.lbb ... b, tj. $0.5 \leq M < 1$; b je bilo koja binarna cifra (0 ili 1). U novom standardu, normalizovana mantisa ima drugi standardni oblik l.bb -b,tj.$1 \leq M < 2$.
Kod broja sa pokretnom tačkom, pozicija binarne tačke mantise može se dinamički pomerati ili "klizati" kako se program izvodi, pri čemu vrednostieksponenta specificiraju tačnu poziciju binarne tačke. Stoga se ovi brojevi često nazivaju FP brojevi (floating point) ili realni brojevi, a u višim programskim jezicima lo su elementarni podaci tipa real ili float.
Principi koji se koriste u predstavljanju binarnih brojeva sa pokretnom tackom najbolje se objašnjavaju primerom. Slika 2.1 pokazuje tipičan 32 - bitni format. Bit označen sa s kodira znak broja (0 = pozitivan, 1 - negativan). Eksponent E se memoriše u kodu sa viškom. U ovome, vrednost pomeraja 2^{k-1} (k je broj bitova eksponenta) se dodaje označenom eksponentu pre nego što se on memoriše. U ovom slučaju, 8-bitni eksponent E može biti bilo koji ceo broj od -2^{7} do $2^{7} - 1$, pa je pomereni eksponent $e = 2^{7} + E$.

Prema tome, granice za e su

$$0 \leq e < 2^{8} - 1.$$

Ovo znači da se sve vrednosti eksponenta memorišu kao pozitivni brojevi, što pomaže da se uprosti rukovanje brojem od strane procesora.Mantisa M ima normalizovani oblik O.lbb ... b. Cifra iza binarne tačke svih mantisa je 1. Ovo znači da nije potrebno da memorišemo ovu cifru (podrazumevajući njenu vrednost) i samo memorišemo bb ... b, tj. modifi-kovanu mantisu m. Kada se m čita, u procesoru se automatski dodaje jedinica koja nedostaje.



Slika 2.1. 32-bitni format za predstavljanje brojeva sa pokretnom lačkom

Podrazumevana 1 u mantisi brojeva sa pokretnom tackom omogućava dodatni bit preciznosti u predstavi ali sprečava vrednost 0 da se predstavi tačno, pošto m = 00 ... 0 predstavlja mantisu 0.1. Pošto je predstavljanje 0 tačno veoma značajno za numerička izračunavanja, usvojeno je da broj sa pokretnom tačkom sa s = 0, e = 00 ... 0 predstavlja 0 tačno.
Kombinacija s = 1, e - 00 ... 0 je takođe rezervisana. Ona nastaje u slučaju prekoračenja opsega brojeva i interpretira se kao kod greške, a ne kao broj.

Kako je m_{min} = 00 ... 0 i m_{max} = 11 ... 1, sledi da je:

$$M_{min} = 0.100 \ldots 0 = 2^{-1}$$

$$M_{max} = 0.111 \ldots 1 = 1 - 2^{-24} \approx 1.$$

Imajući u vidu da je kombinacija e = 00 ... 0 zauzeta, dobija se

e_{min}=0...01=1 i e_{max}= 11...1=2^8-1

Dalje se dobija

$$E_{min} = e_{min} - 2^7 = 1 - 2^7 = -127$$

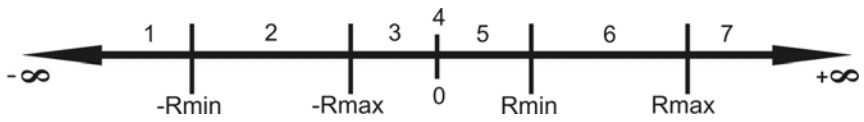
$$E_{max} = e_{max} - 2^7 = 2^8 - 1 - 2^7 = 2^7 - 1 = -127.$$

Prema tome, R_{min} i R_{max} određuju se na sledeći način:

$$R_{min} = M_{min} * 2^{E_{min}} = 2^{-1} * 2^{-127} = 2^{-128}$$

$$R_{max} = M_{max} * 2^{E_{max}} = 1 * 2^{127} = 2^{127}$$

Opsezi brojeva sa pokretnom tačkom koji se mogu predstaviti prikazan na slici 2.2 (oblasti 2 i 6).



Slika 2.2. Predstavljanje brojeva sa pokretnom tačkom kod 32-bitnog formata

Oblasti 1 i 7 predstavljaju prekoračenje (overflow), a oblasti 3 i 5 potkoračenje (underflow). Brojevi koji pripadaju tim oblastima se ne mogu pr staviti. Prekoračenje se javlja kada su rezultati aritmetičke operacije veličini veći nego što se mogu izraziti eksponentom 127. Prekoračenje posledica konačne prirodne predstave za brojeve i neotklonjivo je. Pot račenje se javlja kada je veličina

razlomka previše mala. Potkoračenje manje ozbiljan problem zbog toga što se rezultat može generisati zado ljavajućom aproksimacijom pomoću 0 (tzv. mašinska nula).

Primeri prekoračenja i polkoračenja prikazani su ispod:

$2^{80} * 2^{100} = 2^{180}$

$2^{-80} * 2^{-100} = 2^{-180}$

Samo neki brojevi sa pokretnom tačkom iz oblasti 2 i 6 predstavljaju tačno, a ostali postupkom zaokruživanja kao približni brojevi sa pokretnom tačkom. Dakle, skup brojeva sa pokretnom tačkom je diskretan i konačan, za razliku od skupa realnih brojeva koji je kontinualan i beskonačan.
Za format sa slike 2.1, broj značajnih decimalnih cifara je $d = 0.3 \cdot (p+l)$, gde je 1 - broj koji odgovara jedinici koja se podrazumeva, a p = 23. Sledi da je d = 0.3 - (23 + 1) = 7.2.

Ovaj način zapisa realnih brojeva poznat je kao DEC format i zastupljen je na miniračunarima tipa DEC VAX.

Predstavljanje u drugim brojnim sistemima

Realni broj se može predstaviti u obliku sa pokretnom tačkom sa mantisom i eksponentom koji su zadati u brojnom sistemu sa osnovom b:

$R = M \cdot b^E, \qquad l/b \le |M| < 1 \quad \text{ili} \quad 1 \le |M| < b.$

Nije obavezno da M i E budu u brojnom sistemu sa istom osnovom; sve i zavisi od interpretacije broja. Korisnici računara najčešće upotrebljavaju broj sa pokretnom tačkom u obliku

$R = M \cdot 10^d, \qquad 0.1 \le |M| < 1 \quad \text{ili} \quad 1 \le |M| < 10,$

gde su M i d decimalna mantisa i decimalni eksponent. Broj se zapisuje kao M E d, pri čemu je simbol E razdvajač mantise i eksponenta.

Korišćenje stepena broja 16 je efikasnije za računare od korišćenja stepena broja 10, jer množenje nekog broja sa brojem 16^E u binarnoj notaciji podrazumeva prosto pomeranje razlomljene tačke u levu ili desnu stranu za broj mesta koji je jednak 4·E. Zbog toga se za predstavljanje realnih brojeva kod IBM standarda koristi brojna osnova 16 a ne 10.

Opseg realnih brojeva koji se u nekoj lokaciji mogu predstaviti zavisi od osnove brojnog sistema. Sa povećanjem brojne osnove, taj opseg se povećava, ali se povećava i njihovo rastojanje na brojnoj osi, što dovodi do smanjenja preciznosti.

Zaokruživanje

Kada izračunavanje stvori vrednost koja ne može biti predstavljena ta-čno pomoću formata pokretne tačke, hardver mora zaokružiti rezultat na vrednost koja može biti predstavljena tačno.

Na primer decimalni razlomak 0.2 je neočekivano ekvivalent beskonačnom heksadecimalnom razlomku $0.333333..._{16,}$ pa sledi da je približna vrednost broja 0.2 ekvivalentna $0.333333_{16} \cdot 16^{0}$, u slučaju 24-bitne heksadecimalne mantise. Slično, broj $0.4356234 \cdot 10^{-7}$ u 4-cifarskom decimalnom sistemu se zbog usvojene predstave sa četiri cifre zaokružuje na $0.4356 \cdot 10^{-7}$.

U IEEE 754 standardu, podrazumevani način zaokruživanja je zaokruživanje-ka-najbližem. U zaokruživanju-ka-najbližem, vrednosti se zaokružuju na najbliži broj koji se može predstaviti, i rezultati koji se nalaze tačno na polovini između dva broja koji se mogu predstaviti, zaokružuju se tako da najmanje značajna cifra njihovog rezultata je parna. Standard specifra nekoliko drugih načina zaokruživanja koji mogu biti izabrani od strane programa, uključujući zaokruživanje-ka-nuli, zaokruživanje-ka-plus beskonačnosti i zaokruživanje-ka-minus beskonačnosti.

2.9. BINARNO KODIRANI DEKADNI BROJEVI

Pri pretvaranju dekadnog u binarni broj može se koristiti metod sukcesivnih deljenja (za ceo deo), i sukcesivnih množenja (za razlomljeni deo). Tako dobijeni binarni broj se naziva prirodni binarni kod dekadnog broja. Pri konvertovanju razlomljenih dekadnih brojeva u binarne, zbog ograničenog broja bitova uvek se javljaju određene greške, tj. nepreciznosti. Dodatna greška nastaje zbog toga što tačan racionalan dekadni broj najčešće nema tačan binarni ekvivalent, tj. postupak konverzije nije konačan.

Ove nepreciznosti u prikazivanju brojeva u binarnom brojnom sistemu mogu biti vrlo problematične u automatskoj obradi podataka. Mnoge tačne metode koriste se za prikazivanje racionalnih dekadnih brojeva, da bi ovaj problem bio prevaziđen. Zajedničko za sve metode je da se svaka dekadna cifra zamenjuje ekvivalentom od četiri binarne cifre, tj. sa četiri bita. Jedan od mogućih načina kodiranja dat je tabeli 4, a poznat je kao binarno kodirani dekadni brojevi, ili **BCD** (**Binary Coded Decimal**).

Program mora da “ zna “ ili da “ zapamti “ da su poslednje tri cifre desno od decimalne tačke. Korišćenje kodova kao što je **BCD** omogućava tačno prikazivanje racionalnih dekadnih brojeva, ali se zbog njihove upotrebe u računarima javlja čitav niz drugih problema.

Posmatrajmo binarni broj $01000111_{(2)}$. Kako znati da li on predstavlja broj $71_{(10)}$, $47_{(10)}$ ili $14_{(10)}$? Prvi odgovor podrazumeva da su ovih osam bita normalan binarni broj, drugi da se radi o binarno kodiranom dekadnom broju u kodu "8421", a treći da se radi o **BCD** broju u kodu "više 3".

Binarni ekvivalent dekadna cifra	"8421" **BCD** kod	kod " više 3 "
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 4. Prikaz dekadnih cifara binarnim ekvivalentom u **BCD** kodu

Po ovom postupku broj $29.375_{(10)}$ prikazuje se binarnim ekvivalentom:

0010	1001	0011	0111	0101	kod 8421
0101	1100	0110	1010	1000	kod više 3
2	9	3	7	5	dekadna cifra

Kom dekadnom broju odgovara binarni niz $10010110_{(2)}$? To može biti $150_{(10)}$ (prirodni binarni broj bez znaka), $96_{(10)}$ (8421 BCD), $63_{(10)}$ (vi{e 3 BCD kod), $-105_{(10)}$ (prvi komplement), ili $-106_{(10)}$ (drugi komplement). Mnogi računari ne poseduju sposobnost da razlikuju ove mogućnosti. Programer mora da vodi računa o tome da korišćeni binarni brojevi budu korektno interpretirani i obrađeni.

BCD brojevi imaju još nekoliko značajnih problema. Jedan je pitanje iskorišćenosti prostora brojeva. **BCD** cifre koriste samo 10 od 16 mogućih kombinacija sa četiri bita, pa je efikasnost korišćenja memorije manja. Drugi problem se tiče aritmetike. Sabiranje i oduzimanje **BCD** brojeva ne može se izvesti u prostoj binarnoj aritmetici, kao što je prikazano u primeru gde se vrši sabiranje brojeva 2876_{10} i 6943_{10} koji su predstavnjeni u **BCD** kodu 8421.

```
prenos između tetrada  0    1    0    0
        2875 =  0010 1000 0111 0101
        6943 = +0110 1001 0100 0011
               1001 0001 1011 1000
```

U rezultatu se javljaju sledeće tetrade $1000_{(2)}$, $1011_{(2)}$, $1000_{(2)}$, i $1001_{(2)}$ od kojih druga uopšte nije dekadna cifra. Kada **BCD** aritmetika generiše kod veći od $9_{(10)}$, rezultat mora biti korigovan tako da stvarno predstavlja dekadnu cifru. Tetrada $1011_{(2)}$ je broj $11_{(10)}$, i mora biti konvertovana u broj $0001_{(2)}$ ($1_{(10)}$), sa prenosom u sledeći viši razred. Isto tako između treće i četvrte tetrade se pojavio prenos pa i ovaj rezultat mora biti korigovan.

Ovo podešavanje može izvesti hardver ili softver. Neki računari imaju posebne **BCD** aritmetičke instrukcije, a neki posebne instrukcije za podešavanje rezultata. Podešavanje rezultata se sastoji u tome da se na nepostojeću kombinaciju i na tetradu u kojoj se pojavio prenos u sledeću tetradu doda broj 0110_2, odnosno 6_{10}. Ne obazirući se na sredstva, **BCD** aritmetika zahteva dodatne korake i obično je sporija od binarne aritmetike. Cena kojom su plaćeni tačni dekadni razlomljeni brojevi jeste manja efikasnost korišćenja memorije i sporija obrada. Praktično sabiranje BCD brojeva realizuje se u više koraka:

1. U prvom koraku se vrši sabiranje BCD brojeva po pravilima za binarno sabiranje (bit po bit ne obazirući se na tetrade);
2. U drugom koraku se vrši korekcija nad tetradama po sledećim pravilima:
   a) ako nema prenosa u sledeću tetradu i ako je dobijeni broj u skupu dozvoljenih (manji od 1010_2) ne treba vršiti korekciju;
   b) ako nema prenosa u sledeću tetradu, a dobijena tetrada nije u skupu dozvoljenih ($\geq 1010_2$), onda ovoj tetradi treba dodati broj 6_{10} tj., 0110_2.
   v) ako postoji prenos u sledeću tetradu onda ovoj tetradi treba dodati broj 6_{10} tj., 0110_2.
3. Ako se u drugom koraku pojavi jedna ili više tetrada u kojima se javljaju nedozvoljene kombinacije, korekciju treba ponavljati sve dok sve cifre zadovolje uslov da su manje od 1010_2.

Dakle, rezultat dobijen u prvom koraku, korigujemo prema navedenim pravilima i dobijamo konačan (i tačan) rezultat:

$$\begin{array}{l} {}^{0}1001^{0}\ 0001^{1}1011^{0}1000 \\ \underline{0000\quad 0110\quad 0110\ 0000} \\ \quad 1001\quad 1000\quad 0001\ 1000 \\ \quad\ 9\quad 8\quad 1\quad 8 \end{array}$$

Jedan od često korišćenih načina za zapisivanje BCD brojeva je, takozvani, kod više 3. U ovom kodu se na 8421 kod dekadne cifre doda 0011_2, odnosno 3_{10}. Kod ovog koda rezultat se uvek dobija nakon dva binarna sabiranja, pri čemu se u prvom sabiranju (po pravilima za binarne brojeva) odredi međurezultat, a drugom

koraku se vrši korekcija tako što se prenos između tetrada zanemaruje. Pravila za korekciju su sledeća:

1. ako kod prvog sabiranja nema prenosa u sledeću tetradu, onda od te tetrade treba oduzeti tri, odnosno treba joj dodati 13_{10} tj., 1101_2 (ovo je drugi komplement cifre 3 u heksadecimalnom brojnom sistemu);
2. ako se kod prvog sabiranja pojavi jedinica prenosa iz neke tetrade, onda toj tetradi treba dodati 3_{10}, tj., 0011_2.

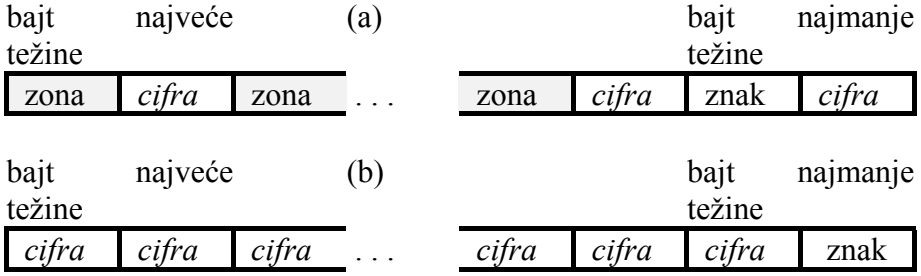
Pogledajmo upotrebu koda više 3 na primeru sabiranja dekadnih brojeva 2875_{10} i 6943_{10}.

prenos između tetrada		0 1 1 0
	2875 =	0101 1011 1010 1000
	6943 = +	1001 1100 0111 0110
međurezultat		1111 1000 0001 1110
međurezultat		1111 1000 0001 1110
korekcija		1101 0011 0011 1101
konačan rezultat u kodu više 3		1100 1011 0100 1011
dekadna vrednost		9 8 1 8

BCD brojevi se smeštaju u memoriju po principu jedna cifra-jedan bajt, i to je takozvani raspakovani oblik. Pri tome se u donji polubajt zapisuje **BCD** kod cifre, a u gornji polubajt specijalni kod koji se zove **zona**, pa se ovaj način pamćenja **BCD** brojeva naziva i zonsko pakovanje.

U poslednji bajt (bajt najmanje težine) pamti se cifra najmanje težine, a umesto zone u tom bajtu pamti se i znak broja, slika 2.2. (a).

Kao što se vidi gotovo polovina memorijskog prostora ostaje neiskorišćena pri zonskom pamćenju brojeva.



Slika 2.2a. Memorisanje i obrada **BCD** brojeva

Kada se **BCD** brojevi obrađuju u aritmetičko-logičkoj jedinici, onda se pakuju po dve dekadne cifre u jedan bajt, i to je pakovani oblik. U pakovanom obliku svaki bajt sadrži po dve dekadne cifre, a bajt najmanje težine sadrži cifru i znak broja kao na slici 2.2. (b). Po završetku obrade **BCD** brojevi se ponovo raspakuju i smeštaju u memoriju.

2.10. KODIRANJE NENUMERIČKIH PODATAKA

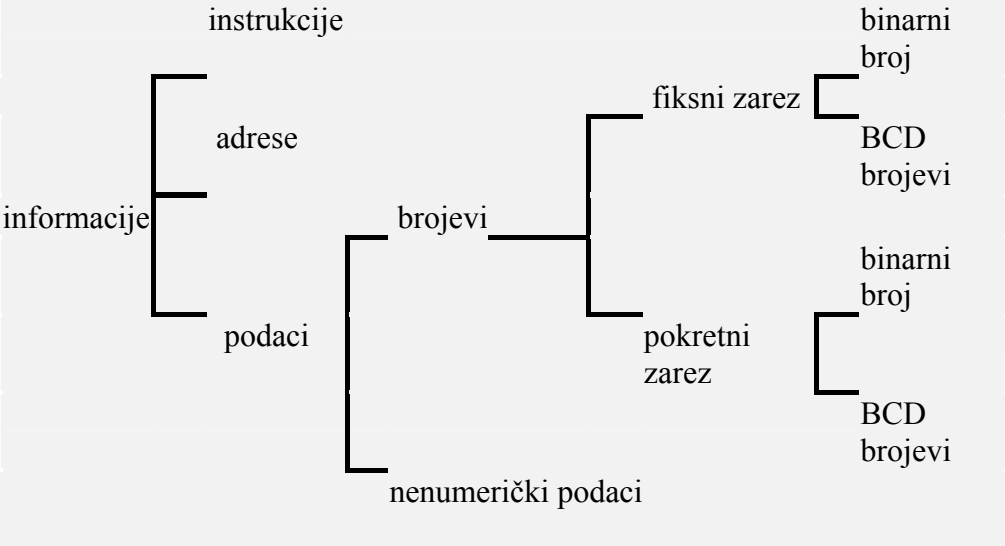
Računari moraju posedovati mogućnost da skladište i obrađuju kako numeričke (brojne), tako i nenumeri~ke tj. tekstualne podatke (alfa numerički podaci). Nenumerički podaci mogu biti: znakovi odnosno karakteri (**character data**), među kojima su: *slova*, *znakovi interpunkcije*, *cifre* 0-9, *matematički znakovi*, *specijalni znakovi* i *kontrolne informacije*, ili nizovi znakova-*niske* tj. *stringovi* (**string**). Podaci ovog tipa su memorisani na poseban način u obliku nizova bitova. Za predstavljanje znakova u početku su se mnogo koristili šestobitni kodovi što se kasnije pokazalo nedovoljnim, jer se sa 6 bitova mogu prikazati samo 64 znaka.

Dva načina kodiranja nenumeričkih podataka koji se najčešće koriste u savremenim računarima su: **ASCII (American Standard Code for Information Interchange)** i **EBCDIC** (**Extended Binary Coded Decimal Interchange Code**). EBCDIC kod je osmobitni kod koji je razvio **IBM**, i koriste ga samo IBM-ovi veliki računari, i neki računari koji su IBM kompatibilni. ASCII je sedmobitni kod, široko rasprostranjen u komunikacijama između računara. Niz bitova $1111000_{(2)}$ predstavlja slovo **x**, a niz $1111001_{(2)}$ označava **y**. Skoro svi mikroračunari koriste ovaj kod za prikaz slova i simbola. Kako i mikrora~unari memorišu podatke u osmobitnim bajtovima, onda oni dodaju jedan ekstra bit na standardni sedmobitni ASCII kod, da bi se dobio ceo bajt. Oba koda ASCII i EBCDIC rezervišu neke nizove bitova za kontrolne informacije. Ovi specijalni kodovi se koriste za označavanje početka (**start**) i kraja (**end**) neke poruke poslate preko komunikacionih linija ili drugih sredstava.

Oba ova koda imaju posebne nizove bitova koji označavaju mala i velika slova, interpunkciju i druge specijalne karaktere. **IBM-PC** kompatibilni mikroračunari koriste 8-bitnu varijaciju **ASCII**-koda, gde svaki novi niz bitova (ekstra kod), označava neki grafički karakter. Postojanje ovih kodnih šema dodatno komplikuje interpretaciju podataka koji se nalaze u memoriji računara. Pomenuti niz bitova $10010110_{(2)}$, pored već pomenutih značewa ($150_{(10)}$, $96_{(10)}$, $63_{(10)}$, $-105_{(10)}$, $-106_{(10)}$), može takođe predstavljati i malo slovo **o** u računaru koji koristi **EBCDIC** kod.

U **ASCII** i **EBCDIC** tabeli se pored slova, brojeva, specijalnih znakova +, -, ? itd. nalaze i kontrolni znaci. Ovi kontrolni znaci ne mogu se videti na ekranu monitora, niti se mogu odštampati. Oni služe za upravljanje radom ulazno-izlaznih uređaja, zatim pri prenosu informacija izme|u dva računara itd. Tabele **ASCII** i **EBCDIC** kodova date su u Prilogu A. Na kraju možemo možemo reći da se u računarima

koriste različiti tipovi binarno kodiranih informacija, čija se osnovna podela može prikazati kao na slici 2.3.



Slika 2.3. Osnovni tipovi informacija u računarskim sistemima

ASCII (AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE)

Najčešće korišćeni znakovni kod je ASCII (American Standard Coda for Information Interchange). Svaki ASCII znak ima 7 bitova, dopuštajući 128 znakova ukupno. ASCII znaci se skoro uvek memorišu ili prenosd korišćenjem 8 bitova po znaku. Osmi bit je na poziciji najznačajnijeg bita i može se ignorisati (postaviti na 0) ili upotrebiti za kontrolu parnosti.

Tabela 2.7 pokazuje ASCII kod i odgovarajući kod srpske latinice. Kod za srpsku ćirilicu je isti, sem što se na mesto slova q, w i x nalaze slova љ, њ i џ, redom. Naši kodovi su definisani po Jugoslovenskom standardu Prve 32 kombinacije, kao i poslednja predstavljaju upravljačke znakove. Neki od njih imaju jedinstvenu interpretaciju kod svih uređaja, a kod nekih interpretacija se razlikuje od uređaja do uređaja.

ASCII znaci koji se štampaju obuhvataju velika i mala slova, cifre, zra ke interpunkcije i nekoliko matematičkih simbola. Kodovi od 30_{16} do 39_{16} su za cifre 0, 1, ... , 9; kodovi od 41_{16} do $5A_{16}$ predstavljaju velika slova; kodovi od 61_{16} do $7A_{16}$ odgovaraju malim slovima. Kod za mala slova razlikuje se od koda za velika slova za 20_{16}, što čini prevođenje velikih slova u mala i obrnuto jednostavnim.

Velika i mala slova su kodirana na način kojj je pogodan i za sortiranje tekstualne informacije. Kodovi za A do Z su u rastućoj numeričkoj sekvenci, što znači da se zadatak sortiranja slova (u reči) ostvaruje pomoću aritmetičkog poređenja kodova koji predstavljaju slova.
Za ASCII bit šablon 011XXXX, cifre 0 do 9 se predstavljaju pomoću njihovih binarnih ekvivalenata, 0000 do 1001, u krajnje desna 4 bita. Isti kod ima i NBCD kod. To olakšava konverziju između 7-bitnog ASCII pakovane BCD predstave.

10	16	Znak	10	16	Znak	10	16	Znak		10	16	Znak	
0	00	NUL	32	20		64	40	@	Ž	96	60	`	ž
1	01	SOH	33	21	!	65	41	A		97	61	a	
2	02	STX	34	22	“	66	42	B		98	62	b	
3	03	ETX	35	23	#	67	43	C		99	63	c	
4	04	EOT	36	24	$	68	44	D		100	64	d	
5	05	ENQ	37	25	%	69	45	E		101	65	e	
6	06	ACK	38	26	&	70	46	F		102	66	f	
7	07	BEL	39	27	‘	71	47	G		103	67	g	
8	08	BS	40	28	(	72	48	H		104	68	h	
9	09	HT	41	29	)	73	49	I		105	69	i	
10	0A	LF	42	2A	*	74	4A	J		106	6A	j	
11	0B	VT	43	2B	+	75	4B	K		107	6B	k	
12	OC	FF	44	2C	’	76	4C	L		108	6C	1	
13	OD	CR	45	2D	-	77	4D	M		109	6D	m	
14	0E	SO	46	2E	.	78	4E	N		110	6E	n	
15	0F	SI	47	2F	/	79	4F	O		111	6F	o	
16	10	DLE	48	30	0	80	50	P		112	70	P	
17	11	DC1	49	31	1	81	51	Q		113	71	q	
18	12	DC2	50	32	2	82	52	R		114	72	r	
19	13	DC3	51	33	3	83	53	S		115	73	s	
20	14	DC4	52	34	4	84	54	T		116	74	t	
21	15	NAK	53	35	5	85	55	U		117	75	u	
22	16	SYN	54	36	6	86	56	V		118	76	V	
23	17	ETB	55	37	7	87	57	w		119	77	w	
24	18	CAN	56	38	8	88	58	X		120	78	X	
25	19	EM	57	39	9	89	59	Y		121	79	y	
26	1A	SUB	58	3A	:	90	5A	Z		122	7A	z	
27	1B	ESC	59	3B	;	91	5B	[	Š	123	7B	{	š
28	1C	FS	60	3C	<	92	5C	\	Đ	124	7C		đ
29	1D	GS	61	3D	=	93	5D	]	Ć	125	7D	}	ć
30	1E	RS	62	3E	>	94	5E	^	Č	126	7E	~	č
31	1F	US	63	3F	?	95	5F	-		127	7F		DEL

Tabela 5. ASCII kod i odgovarajući kod srpske latinice

EBCDIC (Extended Binary Coded Decimal Interchange Code)

Slova	EBCDIC		Cifre	EBCDIC	
A	1100	0001	0	1111	0000
B	1100	0010	1	1111	0001
C	1100	0011	2	1111	0010
D	1100	0100	3	1111	0011
E	1100	0101	4	1111	0100
F	1100	0110	5	1111	0101
G	1100	0111	6	1111	0110
H	1100	1000	7	1111	0111
I	1100	1001	8	1111	1000
J	1101	0001	9	1111	1001
K	1101	0010	Znaci		
L	1101	0011	blanko	0100	0000
M	1101	0100		0100	1011
N	1101	0101	(	0100	1101
O	1101	0110	+	0100	1110
P	1101	0111	$	0101	1011
Q	1101	1000	*	0101	1100
R	1101	1001	)	0101	1101
S	1110	0010		0110	0000
T	1110	0011	/	0110	0001
U	1110	0100		0110	1011
V	1110	0101		0111	1101
W	1110	0110	n	0111	1111
X	1110	0111	—	0111	1110
Y	1110	1000			
Z	1110	1001			

Tabela 6. EBCDIC kod

GREJOV KOD

Grejov BCD kod je najpoznatiji predstavnik kodova sa jediničnim raz- makom. On se menja u samo jednom bitu od jedne kodne reći do sledeće susedne kodne reci (i od poslednje do prve). Takvi kodovi obično imaju principu, manju mogućnost nastanka grešaka od kodova koji imaju veće razmake između pojedinih reci. Ova činjenica je značajna za neke primene, kao što je analogno-digitalna konverzija ugaonog pomeranja. Grejov BCD kod nema težinski karakter, pa nije pogodan za izvođenje aritmetičkih operacija; zbog toga se pri ulazu u računar prevodi u običan binarni kod.
Grejov kod se može formirati i za šesnaest binarnih nizova dužine 4, kao i za binarne nizove sa bilo kojom konstantnom dužinom. Grejov binarni kod je prikazan u tabeli 7. Grejov binarni kod prevođenje analognih podataka u cifarske podatke kod analogno - digitalnih pretvarača. Prednost Grejovog binarnog koda u odnosu na binarne brojeve je u tome, što se pri prelazu od jednog broja ka sledećem po redu u Grejovom binarnom kodu menja samo jedan bit, kao i kod Grejovog BCD koda.

0	1	2	3	4	5	6	7
0000	0001	0011	0010	0110	0111	0101	0100
8	9	10	11	12	13	14	15
1100	1101	1111	1110	1010	1011	1001	1000

Tabela 7. Grejov binarni kod

Kaže se da je kod simetričan ako se dva broja koja su udaljena sa bilo koje strane od centralnih brojeva 7 i 8 razlikuju samo jednom bitu. Takav kod je Grejov binarni kod. Može da se čini da je kod težak za pamćenje, ali je upotrebljiv. Lak za pamćenje način prevođenja iz čistog binarnog broja u Grejov binarni kod je sledeći: Prvo, staviti nulu ispred najznačajnijeg bita. Zatim operaciju isključivo-ILI na susedne bitove, počinjući sa leva.

2.11. ZAKLJUČAK

Binarni brojevi su osnova za funkcionisanje računara. Digitalna kola kombinuju nule i jedinice, i generišu nove nule i jedinice. Mašinske instrukcije i mikroprogrami se takođe prikazuju kao nizovi 0 i 1. Svi programi napisani u asembleru ili bilo kom višem jeziku, da bi mogli da rade, moraju se prevesti u nizove nula i jedinica, odnosno u neke binarne brojeve.

Brojevi unutar računara mogu biti binarni reprezenti pozitivnih i negativnih veličina, celi ili razlomljeni brojevi, u fiksnom ili pokretnom zarezu. Oni se takođe

mogu koristiti kao reprezenti simboličkih informacija i specijalnih kodova. Korišćenje brojeva je od značaja na svim nivoima organizacije i rada računara. Svaki računar ima svoj skup brojeva koje koristi. Kako unutar računara nije moguće razlikovati o kom tipu brojeva ili kodova se radi, programer mora o tome voditi računa da ne bi došlo do pogrešne interpretacije i grešaka u obradi.

2.12. PITANJA

1. Koja je osnovna razlika između rimskog i arapskog brojnog sistema?
2. Kako osnova brojnog sistema utiče na vrednost cifre?
3. Objasniti značaj pozicije cifre u zapisu broja u arapskom brojnom sistemu.
4. Kako se dekadni brojevi konvertuju u binarne, oktalne i heksadecimalne?
5. Kako se izračunava dekadna vrednost broja 10101101 ako je to: a) binarni, b) oktalni ili c) heksadecimalni broj?
6. Koje sve metode postoje za predstavljanje negativnih brojeva u računaru?
7. Konvertovati binarni broj bez znaka -0110110 u prvi i drugi komplement.
8. Koje probleme izaziva korišćenje direktnog koda tj. apsolutne vrednosti sa znakom, i kako se to prevazilazi pomoću komplemenata?
9. Izračunati zbir dva binarna broja bez znaka 10100010 i 11100111.
10. 10.Izračunati razliku prethodna dva binarna broja upotrebom prvog i drugog komplementa.
11. Kako se određuje dekadna vrednost binarnih brojeva u prvom i drugom komplementu?
12. Opisati brojeve u nepokretnom zarezu.
13. Šta je prekoračenje, a šta podkoračenje?
14. Kako se prikazuju brojevi u pokretnom zarezu?
15. Kakve se sve nule mogu pojaviti u računarskom sistemu?
16. Konvertovati dekadne brojeve 200 i 839 u BCD brojeve i sabrati ih po pravilima za sabiranje prirodnih binarnih brojeva. Da li je dobijeni rezultat ispravan?
17. Opisati pogodnosti i mane primene BCD brojeva u poređenju sa prirodnim binarnim brojevima.
18. Da li se kod sabiranja BCD brojeva po pravilima za binarno sabiranje rezultat dobija odmah? Šta treba uraditi da bi se dobio tačan rezultat?
19. Zašto je kod više 3 pogodniji od koda 8421?
20. Kako se u računaru zapisuju podaci tipa karakter?
21. Iskoristiti tabelu ASCII kodova za prikazivanje reči COMPUTER kao niza binarnih brojeva.

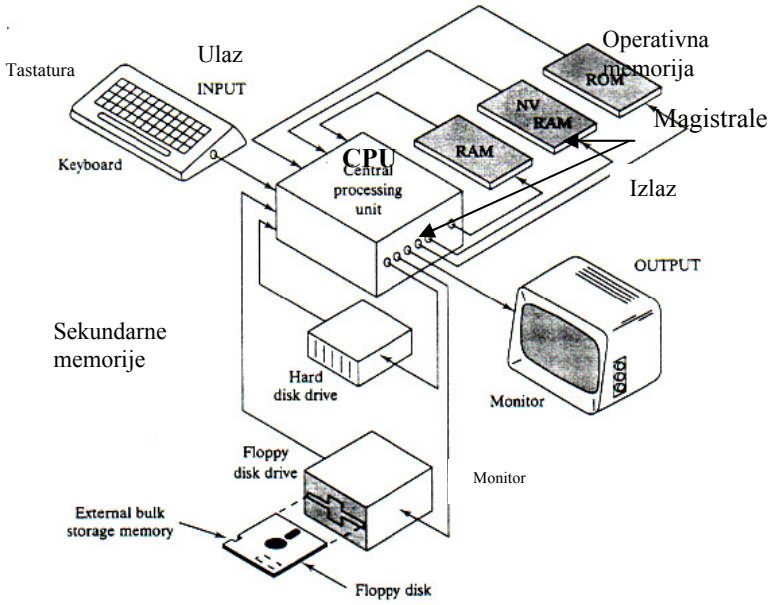
2.13. KLJUČNE REČI:

- arapski brojni sistem
- **ASCII** kod (**American Standard Code For Information Interchange**)
- bajt (**byte**)
- baza (osnova) brojnog sistema
- **BCD** binarno kodirani dekadni brojevi (**binary coded decimal**)
- binarni brojni sistem (**binary system**)
- bit (**bit**)
- decimalna tačka (**decimal point**)
- dekadni brojni sistem (**decimal**)
- direktni kod
- drugi komplement, dopunski kod, komplement dvojke (**two's complement**)
- **EBCDIC** kod (**Extended Binary Coded Decimal Interchange Code**)
- heksadecimalni broj, brojni sistem (**hexadecimal**)
- karakter, znak (**character**)
- **LSB** bit najmanje težine (**least significant bit**)
- **MSB** bit najveće težine (**most significant bit**)
- negativni brojevi (**negative numbers**)
- neoznačeni brojevi (**unsigned numbers**)
- nepozicioni brojni sistem
- nula, pozitivna nula, negativna nula
- oktalni broj, brojni sistem (**octal**)
- pozicioni brojni sistem
- prenos (**carry**)
- prvi komplement, inverzni kod, komplement jedinice (**one's complement**)
- reč (**word**)
- rimski brojni sistem
- string, niska (**string**)
- tetrada, polubajt **(nibble**)
- ciklično vraćanje prenosa, rotacija bita prenosa (**end-around carry**)

3. ELEKTRONSKE OSNOVE RAČUNARA

3.1. DIGITALNA LOGIKA

Hardver računara se najčešće deli na nekoliko glavnih jedinica kao što je prikazano na slici 3.1. Centralna procesorska jedinica (**CPU**) sadrži u sebi glavna aritmetička, logička i upravljačka kola računara. **CPU** se logično deli na: upravljačku jedinicu koja je odgovorna za rad svih ostalih sastavnih delova, aritmetičko-logičku jedinicu u kojoj se obavljaju sve elementarne aritmetičke, logičke i druge osnovne operacije, i izvestan broj specijalnih memorijskih sklopova koji se nazivaju registri.



Slika 3.1 Glavne komponente hardvera računara

CPU je sa drugim delovima računara spojen pomoću jedne ili više sabirnica, tj. magistrala (**bus**), preko kojih u toku rada, različite vrste informacija ulaze ili izlaze iz **CPU**-a. Na magistrale su takođe spojene: jedinice glavne (unutrašnje) memorije, u kojima se pamte podaci i programi koji se upravo koriste, disk jedinice za dugotrajno skladištenje podataka i programa, kao i veliki broj drugih ulaznih i izlaznih jedinica. U srcu bilo kog računara nalaze se digitalna kola koja izvršavaju: upravljačku, logičku, aritmetičku i funkciju pamćenja (skladištenja). Ova kola nalaze se u centru hijerarhijskog modela računarskog sistema. Ona omogućavaju da se podaci i kodovi porede po jednakosti ili raznim formama nejednakosti. Ona izvršavaju sabiranje, oduzimanje i sve logičke operacije. Digitalna kola takođe omogućuju čuvanje podataka za kasniju upotrebu. Osnova za rad digitalnih kola su logičke operacije nad bivalentnim iskazima tj. operacije nad iskazima koji mogu imati samo dve istinitosne vrednosti: tačan (**true**) i netačan (**false**). Za ove dve vrednosti ima mnogo sinonima i primera u svakodnevnom životu: *pozitivno-negativno*, *da-ne*, *nisko-visoko*, *istina-laž, uključeno-isključeno*. Ovakva stanja se vrlo lako mogu kodirati binarnim brojnim sistemom, tj. pomoću **1** i **0**. Teorijske osnove za bivalentnu logiku sadržane su u delu matematike poznate pod imenom Bulova algebra (**George S. Boole**).

3.2. BULOVA ALGEBRA

Bulova algebra je deduktivni matematički sistem koji počiva na aksiomama, na bazi kojih se dalje razvijaju teoreme. Zasniva se na binarnim zakonima mišljenja, gde jedan iskaz može biti ili istinit (tačan) ili neistinit (netačan), a nikada ne može biti delimično tačan ili delimično netačan.

Aksiome i teoreme Bulove algebre

Neka je dat skup **S** = {**x, y, z,** ...} koji sadrži najmanje dva različita elementa, i neka su na ovom skupu definisana dva binarna operanda sa oznakom + (logičko sabiranje, ILI**)** i · (logičko množenje, I), i jedan unarni operand **-** (negacija, NE)**.** Bulova algebra sadrži dva specijalna elementa **0** i **1**, takva da sve promenljive **x, y, z, ...** uzimaju vrednost iz skupa {**0, 1**}. Da bi ovaj skup **S,** i operacije +,- i · sačinjavali Bulovu algebru, neophodno je da budu zadovoljene aksiome **Hantingtona**:

A-1: Binarne operacije + i · su komutativne na skupu **S,** i međusobno su distributivne tako da za svako **x**, **y**, **z**, koji pripadaju skupu **S**, važi:

$$x + y = y + x \qquad x \cdot y = y \cdot x$$

$$x \cdot (y + z) = x \cdot y + x \cdot z \qquad x + (y \cdot z) = (x + y) \cdot (x + z).$$

A-2: Binarne operacije + i · na skupu **S** poseduju neutralne elemente **1** i **0,** tako da za svako **x** koje pripada skupu **S**, postoje elementi **1** i **0**, koji takođe pripadaju skupu **S**, tako da je:

$x + 0 = 0 + x = x$ $\qquad$ $x \cdot 1 = 1 \cdot x = x.$

A-3: Na skupu **S**, za svako **x** koje pripada skupu **S**, postoji jedinstven inverzni element $\overline{x}$, koji takođe pripada skupu **S**, takav da je:

$x + \overline{x} = 1$ $\qquad$ $x \cdot \overline{x} = 0.$

T-1: Teorema idempotentnosti: $\qquad$ $x + x = x$

$x \cdot x = x.$

T-2: Teorema o nultim elementima: $\qquad$ $x + 1 = 1$

$x \cdot 0 = 0.$

T-3: Teorema o involuciji: $\qquad$ $\overline{(\overline{x})} = x$

T-4: Teorema o apsorpciji: $\qquad$ $x + x \cdot y = x$

$x \cdot (x + y) = x.$

T-5: Teorema o asocijativnosti: $\qquad$ $x + (y + z) = (x + y) + z$

T-6: De-Morganovi zakoni: $\qquad$ $\overline{(x + y)} = \overline{x} \cdot \overline{y}$

$\overline{(x \cdot y)} = \overline{x} + \overline{y}.$

Napomena: Ovde je važno uočiti da u Bulovoj algebri važi princip dualnosti, tj. sve aksiome i teoreme su date u paru, pa sve što važi za logičko množenje važi i za logičko sabiranje, samo se **+** zameni sa **·** i **0** sa **1**. Teoreme T-1, T-2**,** a naročito T-4 ukazuju na jednu vrlo bitnu osobinu logičkih funkcija da se složene logičke funkcije mogu minimizirati, tj. transformisati u ekvivalentne logičke funkcije iste istinitosne vrednosti, ali znatno jednostavnijeg oblika sa manje sastavnih delova (manje promenljivih i manje operacija).

Napomena: De-Morganovi zakoni nam kazuju da se složeni logički iskazi negiraju tako što se negira svaki iskaz ponaosob, ali se negira i operacija. Na primer, negacija iskaza: *"Ako sam slobodan i ako je lepo vreme, ići ću u šetnju"* je iskaz: *"Ako nisam slobodan ili ako nije lepo vreme, neću ići u šetnju"*.

OSNOVNE LOGIČKE OPERACIJE NAD BINARNIM CIFRAMA

Digitalna kola su projektovana tako da implementiraju principe binarne aritmetike, Bulove algebre i bivalentne logike. Naime, ova kola se mogu naći u jednom od dva stabilna stanja, tako da se na njihovom izlazu javlja ili visok naponski signal (1) ili nizak (0). Logička kola koriste binarne cifre **0** i **1** za predstavljanje istinitosnih vrednosti netačan i tačan. Uobičajeno je da se vrednost tačan kodira kao binarna jedinica, a netačan kao binarna nula.
Postoje dve vrste logičkih operacija, zavisno od broja operanada koje u njima učestvuju, i to su:

- unarne, logičke operacije nad jednim operandom (negacija),
- binarne, logičke operacije nad dva operanda (sve druge operacije).

Operandi koji učestvuju u logičkim operacijama u računaru mogu biti:

- logički podaci, gde se vrednosti tačan i netačan zamenjuju specijalnim nizom binarnih cifara, tj. imaju specijalan kod,
- binarni brojevi (višecifreni) gde se željena logička operacija primenjuje na svaki bit odvojeno, a rezultat zavisi samo od sadržaja to parag botova (ili para botova iste težine - na istom mestu u zapisu broja),i ne utiče na rezultat operacije nad binarnim ciframa na bilo kom drugom mestu u zapisu broja.

Kombinovanjem elementarnih logičkih operacija, i njihovom primenom na logičke promenljive, dobija se veliki broj različitih logičkih izraza i funkcija. Ako imamo samo dve logičke promenljive (n=2), možemo napraviti 2^p ($p=2^n$), odnosno 16 funkcija.

Negacija (NOT)

Najprostija logička operacija koja se obavlja nad jednom operandom zove se negacija ili **NE** operacija (inverzija ili komplementiranje). Negacija uzima vrednost tačan (1), i konvertuje je u vrednost netačan (0) i obrnuto. Na slici 3.2 je pokazana tabela negacije. X je ulazna veličina (operand), a Z je izlazna veličina (rezultat).

X	Z
0	**1**
1	**0**

$z = \overline{x}$

Slika 3.2. Tabela istinitosnih vrednosti negacije

Za izračunavanje rezultata logičkih operacija obično se koriste tabele (kao na slici 3.2.), koje se zovu tabele istinitosti, kombinacione tabele ili tabele stanja. Ove tabele sadrže sve moguće vrednosti za ulazne veličine (X, Y, A, B,...), i odgovarajući rezultat, odnosno izlaznu vrednost (Z, X, Y, ...). Broj mogućih kombinacija ulaznih veličina jednak je 2^n gde je **n**-broj ulaznih veličina. Ako imamo jednu ulaznu veličinu (negacija), onda je broj kombinacija 2 (2^1), ako imamo dve ulazne veličine broj kombinacija je 4 (2^2), za tri je 8 (2^3) itd.

ILI operacija (OR)

Ova operacija se vrši nad dve ili više ulaznih vrednosti, a naziva se još i logičko sabiranje, disjunkcija. Da bi rezultat operacije imao vrednost **1** (tačan) mora bar jedna ulazna veličina imati vrednost **1** (tačan). Na slici 3.3 je prikazana tablica istinitosti za ILI operaciju nad dve ulazne vrednosti X i Y, kao i tablica istinitosti za **n** ulaznih vrednosti $X_1,..., X_n$. Uočavamo da kombinacije X=1, Y=0 i X=0, Y=1

nisu iste, ali je rezultat operacije isti, tj. Z=1. Rezultat Z=1 dobija se kada su jedna ili više ulaznih vrednosti jednovremeno jednake 1.

X	Y	**Z**
0	0	0
0	1	1
1	0	1
1	1	1

z = x+y

X_1	X_2	...	X_{n-1}	X_n	**Z**
0	0	...	0	0	0
0	0	...	0	1	1
0	0	...	1	0	1
0	0	...	1	1	1
...	...	...	...	...	...
1	1	...	1	0	1
1	1	...	1	1	1

$$\mathbf{Z = X_1 + X_2 + ... + X_n}$$

voz sa leve strane	voz sa desne strane	rampa spuštena
ne	*ne*	*ne*
ne	*da*	*da*
da	*ne*	*da*
da	da	da

Slika 3.3. Tabela istinitosti logičke operacije ILI

Operacija I (AND)

Rezultat ove operacije je istinit (1), samo ako su sve ulazne vrednosti takođe istinite. Drugim rečima, rezultat operacije I (**AND**) je jednak nuli, ako je bar jedna ulazna vrednost jednaka nuli. Operacija I se još naziva logičko množenje ili konjunkcija. Tabela istinitosti za dve vrednosti X i Y, i za niz **n** ulaznih vrednosti $X_1,...X_n$ data je na slici 3.4.

X	Y	**Z**
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

z = x · y

X_1	X_2	...	X_{n-1}	X_n	**Z**
0	0	...	0	0	**0**
0	0	...	0	1	**0**
0	0	...	1	0	**0**
0	0	...	1	1	**0**
.	.	.	.	.	**0**
1	1	...	0	1	**0**
1	1	..	1	0	**0**
1	1	...	1	1	**1**

$$\mathbf{Z = X_1 \cdot X_2 \cdots X_N}$$

imamo slobodno	*lep dan*	*idem u šetnju*
ne	ne	ne
ne	da	ne
da	ne	ne
da	da	da

Slika 3.4. Tabela istinitosti logičke operacije I

Logičko množenje daje rezultat tačan samo ako ni jedan ulazni signal nije jednak nuli, tj. da bi rezultat bio Z=1, moraju svi ulazni signali istovremeno biti jednaki jedinici: X = Y=**1** tj. $X_1 = X_2 = ... = X_n$ =**1**.

Ekskluzivno ILI (XOR)

Ova operacija se naziva još i isključivo ILI, a daje istinit rezultat (tačan, 1), ako je jedna i samo jedna od ulaznih veličina istinita. Tabela istinitosti operacije ekskluzivno ILI data je na slici 3.5. Ako pažljivije pogledamo rezultat ove operacije, uočićemo da on odgovara zbiru binarnih cifara (ne uzimajući u obzir prenos), pa se zato ova operacija naziva i sabiranje po modulu dva**.**

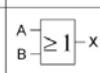
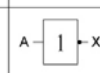
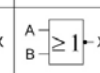
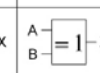
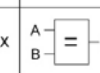
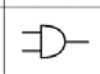
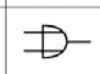
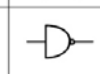
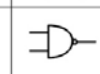
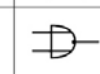
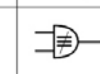
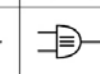
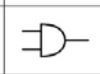
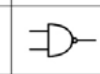
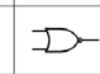
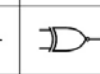
x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

$\mathbf{z = x \oplus y}$

Slika 3.5. Tablica istinitosti ekskluzivnog ILI (**XOR**)

3.3. ELEMENTARNA LOGIČKA KOLA

Osnovne logičke operacije su: NE**,** ILI**,** I i ekskluzivno ILI. Ove operacije, da bi generisale rezultat, slede pravila matematičke logike sa samo dve vrednosti: tačan i netačan (**1** i **0**). Elektronske komponente koje izvršavaju logičke operacije, izraze i funkcije nazivaju se logička kola. Standardni simboli ovih kola dati su na slici 3.6.

KOLO	I	ILI	NE	NI	NILI	EKSLUZ. ILI	KOMPARATOR
IEC							
DIN							
AMERIČKI.							
FUNKCIJA	$X = AB$	$X = A + B$	$X = A$	$\overline{X} = AB$	$\overline{X} = A + B$	$\overline{X} = A\overline{B} + \overline{A}B$	$X = \overline{A}\overline{B} + AB$

Slika 3.6. Osnovna logička kola

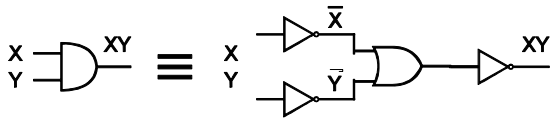
Kombinovanjem ovih kola mogu se realizovati proizvoljne logičke funkcije, kao i sve druge elementarne logičke operacije. Na slici 3.7 prikazana je tabela mogućih logičkih funkcija sa dve ulazne veličine, i sve one se mogu prikazati pomoću elementarnih logičkih operacija I, ILI i NE. Skup operacija pomoću kojih se može realizovati svaka druga logička operacija, i pomoću kojih se može predstaviti svaka

logička funkcija, naziva se baza logičkog sistema. U slučaju Bulove algebre skup operacija {I, ILI, NE} predstavlja bazu. No, operacija I se može realizovati pomoću operacije NE i ILI (kao što se to vidi na slici 3.8.), pa skup {NE, ILI} takođe čini bazu logičkog sistema. Takođe se i operacija ILI može realizovati pomoću operacija I i NE (slika 3.9.), pa i skup {NE, I} čini bazu.

X	0 1 0 1	analitički oblik	NAZIV FUNKCIJE
Y	0 0 1 1	funkcije	
F_0	0 0 0 0	0	konstanta nula
F_1	0 0 0 1	X Y	operacija I, konjunkcija
F_2	0 0 1 0	$\overline{X}Y$	zabrana po X
F_3	0 0 1 1	Y	promenljiva Y
F_4	0 1 0 0	$X\overline{Y}$	zabrana po Y
F_5	0 1 0 1	X	promenljiva X
F_6	0 1 1 0	$X\overline{Y} + \overline{X} \cdot Y$	isključivo ILI
F_7	0 1 1 1	X + Y	operacija ILI, disjunkcija
F_8	1 0 0 0	$\overline{X} \cdot \overline{Y}$	Pirsova funkcija, operacija NILI
F_9	1 0 0 1	$\overline{X} \cdot \overline{Y} + X \cdot Y$	ekvivalencija, komparator
F_{10}	1 0 1 0	$\overline{X}$	negacija X
F_{11}	1 0 1 1	$\overline{X} + Y$	implikacija od X prema Y
F_{12}	1 1 0 0	Y	negacija Y
F_{13}	1 1 0 1	$X + \overline{Y}$	implikacija od Y prema X
F_{14}	1 1 1 0	$\overline{X} + \overline{Y}$	Šeferova funkcija, operacija NI
F_{15}	1 1 1 1	1	konstanta jedan

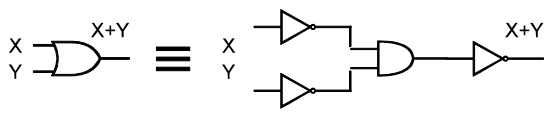
Slika 3.7. Tabela funkcija sa dve promenljive

X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X} + \overline{Y}$	$\overline{\overline{X} + \overline{Y}}$	XY
0	0	1	1	1	0	0
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	0	1	1



Slika 3.8. Realizacija operacije I pomoću operacija NE i ILI

X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X} \cdot \overline{Y}$	$\overline{\overline{X} \cdot \overline{Y}}$	X+Y
0	0	1	1	1	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	0	1	1

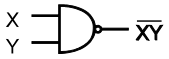


Slika 3.9. Realizacija operacije ILI pomoću operacija I i NE

Posmatrajući tabelu na slici 3.7. uočavamo dve funkcije NI i NILI, koje predstavljaju negaciju dveju osnovnih logičkih operacija. Može se pokazati da je svaka od ovih operacija (i NI i NILI), ujedno baza logičkog sistema, tj. sve druge operacije se mogu realizovati preko samo jedne od ove dve funkcije. Upotreba ovih

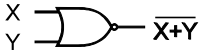
logičkih kola pruža niz pogodnosti u realizaciji digitalnih mreža u računarskim sklopovima i uređajima. Na slici 3.10. data je tabela istinitosti za NI kolo, kao i njihov simbol, a na slici 3.11. data je tabela istinitosti i simbol (koji se koristi u električnim šemama) za kolo NILI.

X	Y	X Y	NI
0	0	0	**1**
0	1	0	**1**
1	0	0	**1**
1	1	1	**0**



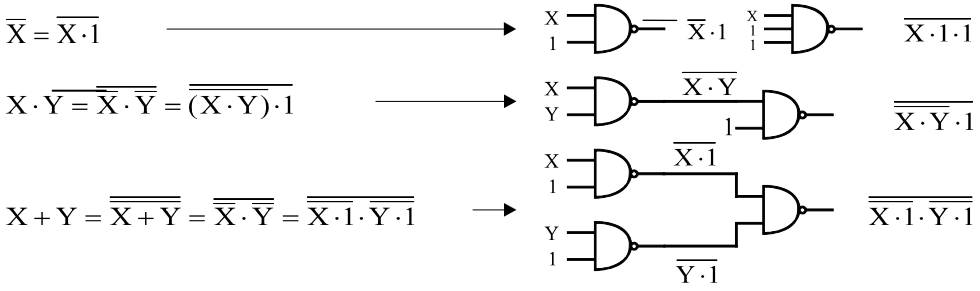
Slika 3.10. Tabela istinitosti i simbol NI kola

X	Y	X+Y	NILI
0	0	0	**1**
0	1	1	**0**
1	0	1	**0**
1	1	1	**0**



Slika 3.11. Tabela istinitosti i simbol NILI kola

Na slici 3.12. pokazano je kako se pomoću NI kola mogu realizovati osnovne logičke operacije NE, ILI i I. Na slici 3.13. prikazana je realizacija NI, ILI i I operacija pomoću NILI operacije.



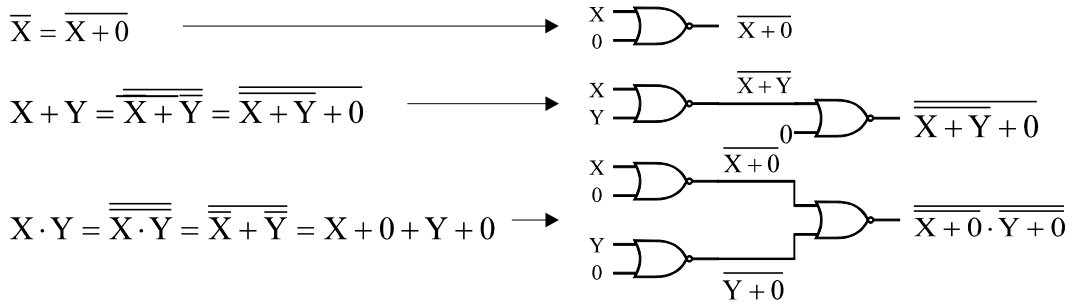
Slika 3.12. Realizacija operacija NE, ILI i I pomoću NI kola

Sa izuzetkom NE kola, sva pomenuta logička kola su dvoulazna. U računarskoj tehnici vrlo često se javlja potreba za primenom I i ILI operacija nad tri, četiri, osam, šesnaest i više ulaza istovremeno.

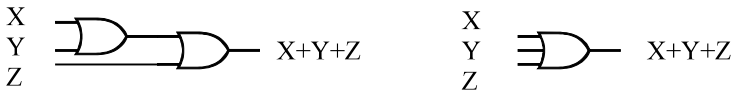
Ovaj problem se može rešiti dvojako:

1. upotrebom višeulaznih logičkih kola,
2. povezivanjem više dvoulaznih kola.

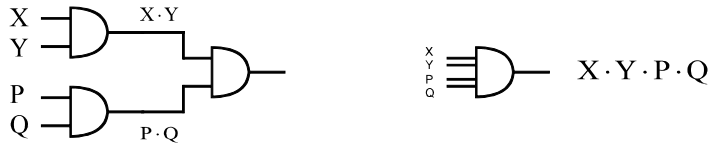
Na slici 3.14. prikazana je realizacija ILI kola sa tri ulaza (troulazno kolo), a na slici 3.15. realizacija I kola za četiri ulaza. Upotreba standardnih logičkih kola, a naročito NI i NILI kola, umnogome pojednostavljuje i pojeftinjuje izradu složenih logičkih mreža. Naime logička kola se nikada ne prave pojedinačno, već se u jednom integrisanom kolu (čipu, **chip**) obično nalazi nekoliko logičkih kola iste vrste. Na primer, u jednom čipu sa 14 izvoda obično se nalaze četiri dvoulazna logička kola I, ILI**,** NI, ili NILI.



Slika 3.13. Realizacija operacija **NE, I** i **ILI** pomoću **NILI** kola



Slika 3.14. Realizacija troulaznog **ILI** kola



Slika 3.15. Realizacija četvoroulaznog **I** kola

Određene logičke funkcije mogu naći i praktičnu primenu u računarskoj tehnici. Realizacija sklopova koji podržavaju rad ovakvih funkcija zahteva niz elektronskih elemenata. Što je veći broj operatora u izrazu za logičku funkciju to je i njeno izvođenje složenije. No, obzirom na to da se u izradi jednog računara koristi od nekoliko stotina do nekoliko hiljada logičkih funkcija, jasno je da je svaka ušteda u tom pogledu od značaja. Da bi se ostvarile uštede u potrošnji logičkih kola, neophodno je minimizirati sve logičke funkcije koje se javljaju u jednom računarskom sistemu. Minimizacija je neophodna da bi se uprostila električna mreža, i smanjio broj logičkih kola potrebnih za realizaciju. U ovom slučaju to znači da funkcije treba prikazati sa što manje operatora i promenljivih, a da pri tome funkcija ima isto značenje, odnosno isti skup vrednosti.

3.4. MINIMIZACIJA LOGIČKIH FUNKCIJA

Minimizacija je postupak transformacije složene logičke funkcije u funkciju koja ima istu istinitosnu vrednost, ali manji broj elemenata, manje operanada i operacija koje nad njima treba izvršiti.

Logičke funkcije mogu se prikazati na različite načine, a mi smo u dosadašnjem izlaganju već koristili tri osnovna:

- šematsko prikazivanje pomoću logičkih kola,
- tabelarno, pomoću tablica istinitosti,
- analitičko, pomoću osnovnih logičkih operacija.

Sve logičke funkcije analitički se mogu prikazati u dva oblika:

- disjunktivna forma, koja predstavlja logičku sumu logičkih proizvoda,
- konjunktivna forma, koja predstavlja logički proizvod logičkih suma.

Ako je funkcija prikazana tabelarno, može se odrediti njen analitički oblik kao disjunktivna ili konjunktivna forma, pri čemu se za određivanje disjunktivne forme grupišu elementi koji odgovaraju jedinici, tj. za koje je vrednost funkcije jednaka logičkoj jedinici, dok se za konjunktivnu formu grupišu elementi koji odgovaraju nulama funkcije, kao što je pokazano na slici 3.16. Mada postoje dva standardna načina ovog prevođenja, ovde ćemo razmotriti samo jedan od njih, a drugi se realizuje po analogiji. Taj algoritam definiše logički izraz funkcije u obliku disjunktivne normalne forme (**DNF**), a sastoji se od sledećih koraka:

a) prvo se za sve vrednosti funkcije F=1 pravi konjunkcija logičkih promenljivih i to samih promenqivih ukoliko je njihova vrednost u tom slučaju jednaka 1, odnosno negacija promenljivih ako je njihova vrednost jednaka 0. Na primer, za broj 3 je $X_1=0$, $X_2=1$, $X_3=1$, pa je njoj odgovarajuća konjunkcija: $\overline{X}_1 \cdot X_2 \cdot X_3$, odnosno $\overline{X}_1 X_2 X_3$.

b) kada su formirane sve konjunkcije za vrednosti F=1, onda se one povezuju operatorom disjunkcije (+) i na taj način se dobija disjunktivna normalna forma date funkcije F.

Očito je da disjunktivna normalna forma daje veoma složen oblik funkcije, pa ćemo posebnu pažnju posvetiti minimizaciji funkcija datih u ovom obliku. Postoji više načina na koje se jedna logička funkcija može minimizirati. Kao prvo, navedimo primenu aksioma i teorema Bulove algebre.

Tako npr. ako se u jednoj funkciji nalazi izraz **x+0**, on se na osnovu aksiome **A-2** (x+0=x) može zameniti samo sa promenljivom **x**, čime se složenost funkcije smanjuje za jedan operator. Isto tako, ako je u funkciji dat izraz oblika **x·(x+y)**, koristeći teoremu o apsorpciji, ovaj izraz se može zameniti sa **x**, pa se celokupna

funkcija pojednostavljuje za dva operatora (konjunkciju i disjunkciju) i ima jednu promenljivu manje.

dec.br.	X_1	X_2	X_3	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

konjunktivna forma grupiše nule (0, 1, 2, 5)

$$F = (X_1+X_2+X_3)(X_1+X_2+\overline{X}_3)(X_1+\overline{X}_2+X_3)(\overline{X}_1+X_2+\overline{X}_3)$$

(0) (1) (2) (5)

disjunktivna forma grupiše jedinice (3, 4, 6, 7)

$$F=\overline{X}_1X_2X_3+X_1\overline{X}_2\overline{X}_3+X_1X_2\overline{X}_3+X_1X_2X_3$$

(3) (4) (6) (7)

Slika 3.16. Određivanje normalne forme na bazi tabele

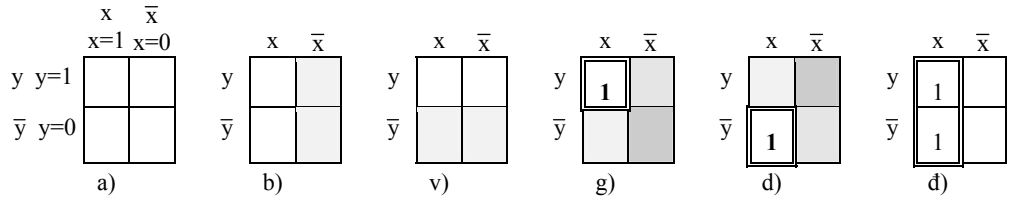
Primena navedenih zakona ne daje određeni algoritamski put kako da se funkcija minimizira. Ona omogućuje projektantima da se na osnovu svog iskustva i sagledavanja pojedinih elemenata funkcije snađu i smanje broj operatora u funkciji. Zato su u okviru Bulove algebre razvijene posebne metode minimizacije, a jedna od najpoznatijih zasniva se na primeni Karnoovih mapa (Karnaugh).

Metoda Karnoovih mapa

Po ovoj metodi minimizacija se izvodi grafičkim putem. Ona je jednostavna i praktična, a zasniva se na upisivanju funkcije u specijalnu tabelu, Karnoovu mapu odnosno dijagram. U sledećem koraku vrši se minimizacija, i konačno, funkcija se može predstaviti u svom minimalnom obliku. Karnoove mape su različite i zavise od broja promenljivih u funkciji. Obično se ovaj metod primenjuje za minimizaciju funkcija sa 2, 3 i 4 promenljive, a za funkcije koje imaju više promenljivih koristi se neka od drugih poznatih metoda, na primer tablična minimizacija. Pogledajmo sada kako izgleda Karnoova mapa za dve promenljive **x** i **y** slika 3.17 a). Ona se sastoji od 4 polja. Svako od tih poqa rezervisano je za jednu moguću konjunkciju promenljivih ili njihovih negacija. Ukoliko se određena konjunkcija pojavljuje u funkciji, onda se u dato polje zapisuje 1, a ako se ne pojavljuje, ne zapisuje se ništa. Na taj način se preslikava funkcija dve promenljive na odgovarajuću Karnoovu mapu.

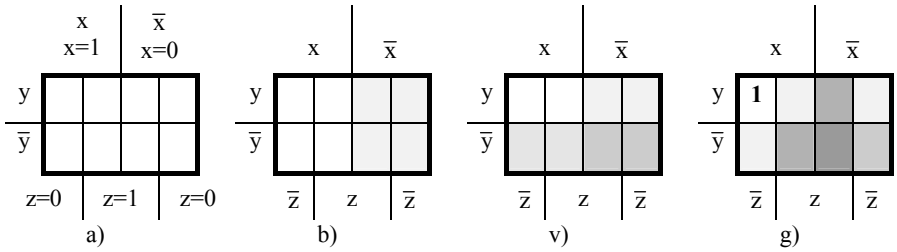
Na primer, ako je data funkcija $F(x,y)=(x\cdot y)+(x\cdot\overline{y})$, u njoj se pojavljuju dve konjunkcije: **x·y** i $x\cdot\overline{y}$. Mesto konjunkcije **x·y** u mapi određuje se na sledeći način: pošto konjunkcija **x·y** sadrži promenljivu **x** (bez negacije) onda se ona mora nalaziti u oblasti gde je **x =1** (a osenčimo oblast **x =0**, slika 3.17. b)), a kako sadrži i

promenljivu **y** to mora biti i u oblasti gde je **y=1** (neosenčena oblast na slici 3.17. v)). Dakle, konjunkcija **x·y** se nalazi na polju koje je u preseku ove dve oblasti, u koje upisujemo 1 (slika 3.17. g)).



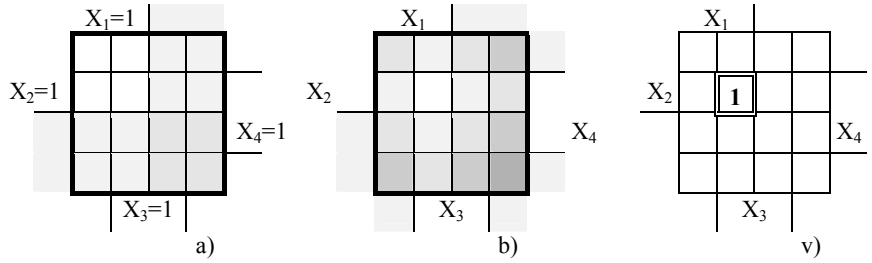
Slika 3.17. Karnoova mapa za funkciju sa dve promenljive

Na isti način se određuje polje koje sadrži konjunkciju $x \cdot \bar{y}$, slika 3.17. d), a Karnoova mapa čitave funkcije **F** je data na slici 3.17. đ). Problem je nešto složeniji kod funkcija za 3 promenljive. Neka su to promenljive **x**, **y** i **z**, Odgovarajuća Karnoova mapa ima izgled prikazan na slici 3.18 a). I u ovom slučaju svako polje mape rezervisano je za jednu tačno određenu konjunkciju promenljivih ili njihovih negacija. Kako se za zadatu konjunkciju pronalazi odgovarajuće polje? Pronalaženje odgovarajućeg polja ide postepeno, eliminacijom polja koja ne zadovoljavaju. Neka je data konjunkcija $x \cdot y \cdot \bar{z}$. Pošto se u zadatom izrazu promenljiva **x** javlja u afirmativnom obliku, u obzir dolaze 4 leva polja (desna polovina mape se osenči jer ta polja ne dolaze u obzir, slika 3.18. b)). Druga promenljiva je **y**, pa se 4 leva polja smanjuju samo na 2, i to u gornjoj vrsti, a donja vrsta se osenči (slika 3.18. v)). Konačno, treći element je $\bar{\mathbf{z}}$, pa treba osenčiti polja u sredini u kojima je **z=1** (slika 3.18. g)). Preostalo, neosenčeno polje određuje polje datog izraza, i u njega upisujemo **1**, slika 3.18. g). Analognim postupkom može se naći polje za bilo koju konjunkciju 3 promenljive.



Slika 3.18. Karnoova mapa za tri promenljive

Na kraju ćemo prikazati kako se popunjava Krnoova mapa (slika 3.19. a)) za funkcije sa 4 promenljive $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$ i $\mathbf{X}_4$. Istim postupkom eliminacije odredićemo položaj konjunkcije: $\mathbf{X}_1 \cdot \mathbf{X}_2 \cdot \mathbf{X}_3 \cdot \mathbf{X}_4$. Pošto su sve promenljive bez negacije, treba osenčiti polja u kojima promenljive uzimaju vrednost 0 (slika 3.19. a) i b)), a preostalo neosenčeno polje je polje u koje se upisuje 1 koja odgovara ovoj konjunkciji, slika 3.19. v).

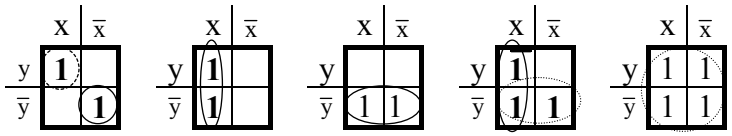


Slika 3.19. Karnoova mapa za četiri promenljive

Do sada smo videli kako se funkcija, koja je zadata disjunktivnom normalnom formom, preslikava na odgovarajuću Karnoovu mapu. Sledeći korak je upravo proces minimizacije. On se sastoji u grupisanju jedinica u okviru mape u veće celine. Pri tome se nastoji da ove celine budu što veće, ali one ne smeju da sadrže polja koja nemaju 1. Može se grupisati: 1 polje samostalno, 2 polja, 4 polja, 8 polja ili 16 polja. Pošto svaka Karnoova mapa ima neke svoje specifičnosti, razmotrimo mogućnost grupisanja kod svake vrste.

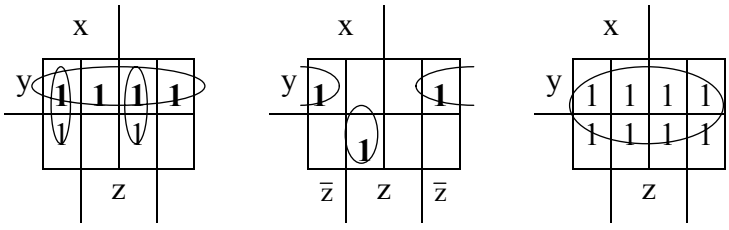
Kod mape sa 2 promenljive mogu se grupisati: samo 1 polje, dva susedna polja ili sva 4 polja. Kada se u jednoj mapi pravi više grupa, nastoji se da te grupe budu što veće, makar i po cenu zajedničkog polja u grupama.

Primeri pravilnog grupisanja su dati na slici 3.20.:



Slika 3.20. Grupisanje jedinica u Karnoovoj mapi sa dve promenljive

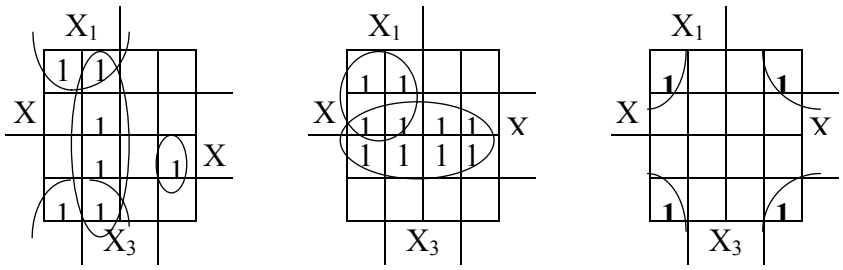
Kod mape sa 3 promenljive prethodnim principima grupisanja potrebno je dodati još dva: mogu se grupisati osam jedinica i elementi sa strane, kao što je pokazano na slici 3.21. Ako se pažljivije posmatra Karnoova mapa, može se uočiti da se područje nezavisno promenljive $\bar{z}$ proteže direktno sleva na desnu stranu tabele.



Slika 3.21. Grupisanje jedinica u Karnoovoj mapi sa tri promenljive

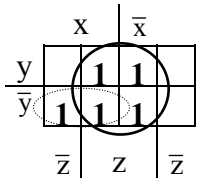
To se može zamisliti kao da je tabela nacrtana na valjku, čiji je omotač po visini razvijen u ravan.

Kod mape sa 4 promenljive, sva prethodna pravila važe, a mogu se grupisati i krajwe gornja sa krajnjim donjim poljima, kao i polja u uglovima. Primeri pravilnog grupisanja u mapama sa 4 promenljive su dati na slici 3.22. Osnovno pravilo pri formiranju grupa je da treba formirati minimalan broj grupa tako da budu obuhvaćene sve jedinice u Karnoovoj mapi.



Slika 3.22. Grupisanje jedinica u Karnoovoj mapi sa četiri promenljive

Nakon prve i druge faze minimizacije, koje se sastoje od preslikavanja logičke funkcije na Karnoovu mapu i grupisanja jedinica, pristupa se poslednjoj fazi: određivanju analitičkog oblika minimalne forme logičke funkcije na osnovu formiranih grupa. Ova faza sastoji se u sledećem: za svaku formiranu grupu definiše se konjunkcija promenljivih ili njihovih negacija, ali samo onih koje su za celu grupu nepromenjene. Kada su formirane konjunkcije za sve grupe, konačna funkcija dobija se kao disjunkcija ovih pojedinačnih članova. Razmotrimo to na sledećem primeru mape sa 3 promenljive gde su već formirane grupe kao na slici 3.23:



Slika 3.23. Određivanje analitičkog izraza za grupu jedinica

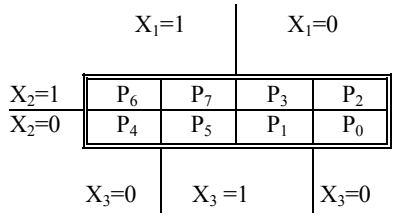
Prvo definišemo odgovarajuću konjunkciju za grupu od 4 srednje jedinice. U ovom slučaju grupa ima jedinice u poljima **x** (**x=1**) i $\bar{\mathbf{x}}$ (**x=0**) odnosno prelazi iz afirmacije u negaciju, pa ta varijabla neće figurisati u rezultujućem izrazu. U datoj grupi ima polja i sa **y=1** i sa **y=0**, pa ni ova varijabla ne utiče na rezultujuću konjunkciju. Grupa je definisana u svim poljima sa **z=1** (oblast **z**), tako da varijabla **z** ulazi u konjunkciju. Konačno, prvi faktor minimalne logičke funkcije je **z** (nije vezan konjunkcijom jer je samo jedna promenljiva). Druga grupa sastoji se od dva polja u donjem levom uglu mape. Vrednost varijable **x** je jedinstvena u ovoj grupi, pa ulazi u konjunkciju. Zatim, u grupi postoji polje sa varijablom **z**, ali i sa $\bar{\mathbf{z}}$, pa, prema tome, ova varijabla ne ulazi u konačni izraz za ovu grupu. Kako je za

ovu grupu i varijabla $\bar{y}$ jedinstvena, to i ona ulazi u izraz, pa je konačan oblik konjunkcije za ovu grupu: $\mathbf{h \cdot \bar{y}}$. Minimalni oblik ove funkcije dobija se povezivanjem konjunkcija za pojedine grupe operatorom disjunkcije, pa je u našem slučaju: $\mathbf{F = z + (x \cdot \bar{y})}$

Prilikom minimizacije logičkih funkcija moguće je poći i od tablice istinitosti, a ne samo od disjunktivne normalne forme, slika 3.24. Da bi se tablica istinitosti preslikala na Karnoovu mapu, posmatraju se svi redovi gde je funkcija jednaka **1** i odgovarajuća kombinacija 1 i 0 promenljivih definiše polje u mapi. Naime, kada promenljiva ima vrednost 1, posmatra se u Karnoovoj mapi oblast u kojoj je ta promenljiva jednaka jedinici, a ako ima vrednost 0, posmatra se njena negacija.

dek. br.	X_1	X_2	X_3	Z
0	0	0	0	P_0
1	0	0	1	P_1
2	0	1	0	P_2
3	0	1	1	P_3
4	1	0	0	P_4
5	1	0	1	P_5
6	1	1	0	P_6
7	1	1	1	P_7

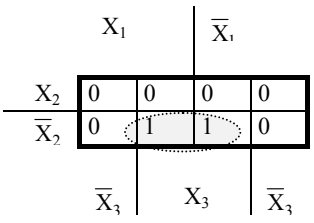
a)



b)

dekad.	X_1	X_2	X_3	Z
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

c)



d)

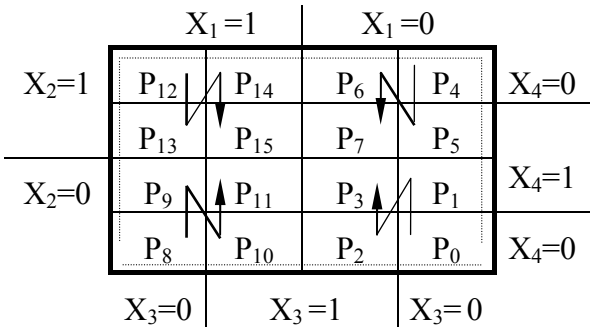
Slika 3.24. Popunjavanje Karnoove mape i minimizacija funkcije **z**

Međutim, ako se promenljive X_1, X_2, X_3 rasporede kao na slici 3.24. a), onda se Karnoova mapa može popuniti po šablonu jer svakoj vrsti iz tabele uvek odgovara jedno te isto mesto u Karnoovoj mapi, kao na slici 3.24. b). Kako je položaj konjunkcija fiksiran, to se samo umesto P_0, P_1, ... P_7 upišu jedinice ili nule (slika 3.24. g)), saglasno tabeli istinitosti, slika 3.24. v). Na stranicama dijagrama d) naznačene su oblasti nenegiranih promenljivih (afirmacija) i njihovih komplemenata (negacija). Grafički se minimizacija izvodi tako što se u tabeli zaokruže dve susedne jedinice, kao u tabeli na slici 3.24. g), za funkciju zadatu tabelom v). Posmatra se koja od promenljivih u okviru grupe jedinica ne prelazi iz

afirmacije u negaciju. To su u datom primeru $\overline{X}_2$ i X_3 . Njihov logički proizvod će predstavljati traženu minimalnu disjunktivnu formu funkcije:

$$\mathbf{F} = \overline{\mathbf{X}}_2 \cdot \mathbf{X}_3$$

Za funkciju sa četiri promenljive, pri fiksnom rasporedu promenljivih, redosledređanja u Karnoovoj mapi elemenata dat je na slici 3.25. Ovu tabelu treba zamisliti kao da je nacrtana na torusu, pa razvijena u ravan. Drugim rečima, njena leva ivica se direktno naslanja na desnu, a donja ivica na gornju. Ovo treba imati u vidu prilikom zaokruživanja susednih članova.



Slika 3.25. Karnoova mapa za četiri promenljive

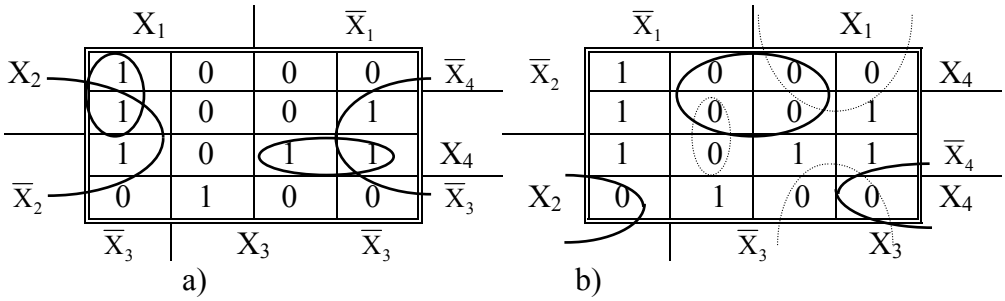
Za nalaženje minimalne konjunktivne forme (**MKF**) funkcije, treba najpre popuniti Karnoovu mapu i to na isti način kao kod disjunktivne forme. Zatim treba grupisati pojedinačna polja, parove, kvartete i oktete nula i to na isti način kao što se grupišu jedinice kod disjunktivne forme. Na kraju treba odrediti analitički izraz za minimalne disjunkcije, s tim što afirmacije i negacije promenljivih na stranicama Karnoove mape međusobno zamenjuju mesta. Drugim rečima, polja u oblasti gde je promenljiva jednaka nuli uzimaju se kao afirmacija, npr. oblast **x=0** je oblast **x**, a oblast **x** =**1** je oblast $\overline{\mathbf{x}}$. Tabela na slici 3.26. a) pokazuje Karnoovu mapu za dobijanje disjunktivne, a tabela b) pokazuje Karnoovu mapu za dobijanje konjunktivne forme funkcije. Kao što se vidi, one su identične, samo su oblasti negacije i afirmacije promenljivih ukrštene.Minimalna disjunktivna forma (**MDF**) je:

$$F = X_1X_2\overline{X}_3 + \overline{X}_3X_4 + \overline{X}_1\overline{X}_2X_4 + X_1\overline{X}_2X_3\overline{X}_4 .$$

MKF ima onoliko logičkih suma (disjunkcija) koliko u tabeli postoji zaokruženih polja, parova, kvarteta i okteta nula. U svakoj od tih suma učestvuju one promenljive koje u intervalu zaokruženih grupa ne prelaze iz afirmacije u negaciju. Kvartet nula u sredini daje sumu $\overline{X}_2 + \overline{X}_3$, a kvartet zaokružen isprekidanom linijom daje sumu X_1+X_4. Od para nula zaokruženih isprekidanom linijom dobija se član $\overline{X}_1 + \overline{X}_3 + \overline{X}_4$, a par zaokružen punom linijom daje disjunkciju $X_2+X_3 + X_4$.

Finalni oblik MKF dobija se kada se napravi logički proizvod (konjunkcija) ovih disjunkcija:

$$F = (\overline{X}_2 + \overline{X}_3) \cdot (\overline{X}_1 + \overline{X}_3 + \overline{X}_4) \cdot (X_1 + X_3) \cdot (X_2 + X_3 + X_4)$$



Slika 3.26. Primer Karnoovih mapa za funkciju sa četiri promenljive

U praktičnim kolima može se desiti da se određene kombinacije nezavisno promenljivih nikada ne pojavljuju. Tada se mesto u tabeli koje odgovara toj kombinaciji promenljivih može tretirati i kao logička nula i kao logička jedinica, zavisno od toga kako se dobija optimalnije grupisanje sa susednim jedinicama (kod MDF), odnosno nulama (kod MKF).

3.5. ELEMENTARNA MEMORIJSKA KOLA

Logička stanja **0** i **1** u digitalnim logičkim kolima predstavljena su niskim i visokim naponom. Tokom rada računara (u postupku obrade), naponski signali prolaze kroz elektronska kola i pri tome se brzo i neprekidno menjaju. Podaci dolaze na ulaze logičkih kola u vidu električnih impulsa preko jedne ili više ulaznih linije veze. Logička kola obrađuju ove signale i generišu izlazni naponski signal, koji se takođe vodi na neke ulazne linije drugih logičkih kola. Čim se promeni nivo napona na ulazu, menja se i nivo napona na izlazu (saglasno zakonu obrade u toj logičkoj mreži). Logičke mreže, kod kojih izlazna stanja zavise samo od trenutne vrednosti ulaznih veličina nazivaju se kombinacione mreže.

Drugu vrstu logičkih mreža tzv. **sekvencijalne mreže**, čine mreže kod kojih logičko stanje na izlazu zavisi ne samo od trenutne vrednosti signala na ulazu, već i od prethodnog stanja u kom se ta logička mreža nalazila. Da bi bila moguća realizacija ovakvih mreža, moraju postojati takvi elektronski sklopovi sposobni da zapamte prethodno stanje. Takva elektronska kola nazivaju se memorijski elementi. Minimalna količina podataka koja se može zapamtiti je jedna **0** ili **1**, tj. jedan bit. Logička mreža koja može da zapamti jedan bit (jednu binarnu cifru), zove se flip-flop.

FLIP-FLOP

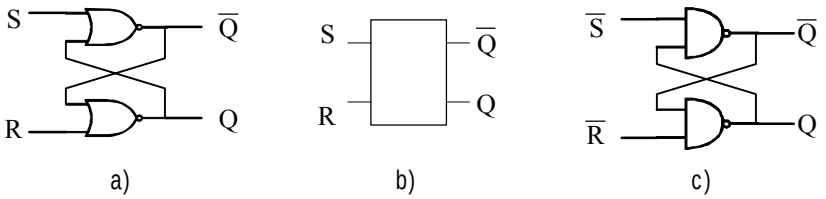
Flip-flop je jedno od najprostijih kola sa dva stabilna stanja koja se koriste za skladištenje, odnosno memorisanje podataka u binarnom obliku, a istovremeno i jedno od osnovnih kola digitalne tehnike.

Bit informacije se kodira prisustvom ili odsustvom impulsa, ili logičkog nivoa 1 ili 0, pa samim tim jedan flip-flop može da pamti u određenom vremenu samo jednu binarnu cifru, tj. jedan bit informacije. Podaci veći od jednog bita, pamte se u uređenom skupu flip-flopova koji se naziva registar. Skup više registara, organizovanih na određeni način, čini memoriju. Rad flip-flopa kao memorijskog elementa može biti prikazan tabelom istinitosnih vrednosti ili pomoću odgovarajućih logičkih funkcija prikazanih u analitičkom obliku.

R/S flip-flop

R/S flip-flop se sastoji od ukrštene veze dva NILI (ili dva NI) kola, tako da je izlaz prvog spojen na ulaz drugog, a izlaz drugog na ulaz prvog. Na taj način je ostvarena pozitivna povratna sprega potrebna za kumulativni proces pri promeni stabilnih stanja.

Logička struktura R/S flip-flopa, realizovanog ukrštanjem dva dvoulazna NILI kola, prikazana je na slici 3.27 a). Na slici 3.27 b) prikazan je grafički simbol flip-flopa.

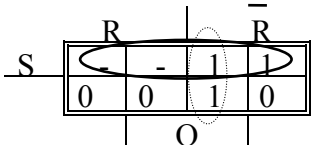


Slika 3.27. **R/S** flip-flop realizovan **NILI** i **NI** kolima

Konvencijom je usvojeno da se stanje flip-flopa izražava i interpretira logičkom vrednošću napona na jednom izlazu, koji se naziva glavni izlaz flip-flopa i najčešće označava sa **Q**. Drugi izlaz je uvek komplementaran $\overline{Q}$, i ukoliko se koristi, eliminiše upotrebu jednog NE kola. Za početno stanje flip-flopa usvojeno je stanje logičke 0 na glavnom izlazu, tj **Q = 0**. Ulaz **S** (**Set**) služi za postavljanje izlaza flip-flopa na stanje logičke jedinice (setovanje), tj. **Q = 1**, $\overline{Q}$ **= 0**. Drugi ulaz **R** (**Reset**) služi za dovođenje izlaza u stanje logičke nule (resetovanje), tj. **Q = 0**, $\overline{Q}$ **= 1**.

Stanje izlaza u koje će flip-flop preći u narednom trenutku (t+1), označava se kao Q(t+1), i ne zavisi samo od ulaznih signala S(t) i R(t), već i od stanja flip-flopa Q(t), u posmatranom trenutku t. Tabela stanja, na slici 3.28, u potpunosti određuje rad R/S flip-flopa.

R(t)	S(t)	Q(t)	**Q(t+1)**
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**0**
1	1	0	**-**
1	1	1	**-**



Slika 3.28. Tabela stanja R/S flip flopa i Karnoova mapa

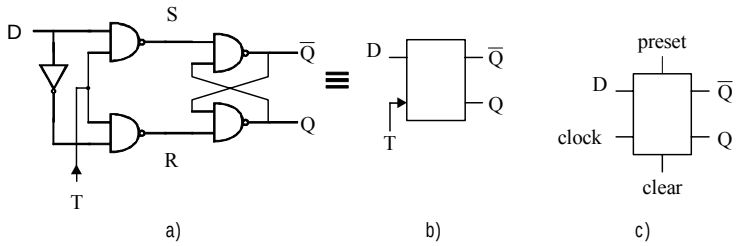
Minimizacijom na osnovu Karnoove mape sa slike 3.28, dobijamo minimalnu disjunktivnu formu funkcije R/S flip-flopa:

$$Q(t+1) = S(t) + \overline{R}(t)Q(t),$$

uz uslov da R(t)S(t) = 0, kojim se zabranjuje da oba ulaza istovremeno budu jednaka jedinici (ne može istovremeno biti S(t) = 1 i R(t) = 1), jer tada stanje izlaza nije definisano. Naime, pošto je S = 1 onda izlaz Q mora biti jedinica, ali zbog R = 1 izlaz bi morao da bude nula. Dakle, izlaz bi u istom trenutku vremena trebao da ima dve različite vrednosti, što je nemoguće. R/S flip-flop se može realizovati i pomoću dva **NI** kola, slika 3.27 c). Kod ovog flip-flopa postoji neodređenost izlaza u slučaju da na oba ulaza i S i R do|u nule (S = 0 i R = 0, odnosno kada je $\overline{R}(t) \cdot \overline{S}(t) = 1$). Zbog pojave ovih neodređenosti izlaza, prave se i druge vrste flip-flopova, kao na primer **R/S** flip-flopovi sa jednim dominantnim ulazom, zatim **D** flip-flop, **JK** flip-flop, **T** flip-flop, **JKT** flip-flop, **MS** flip-flop, itd.

D flip-flop

R/S flip-flop nije pogodno kolo za pomeranje podataka u računaru, iz prostog razloga što je potrebno ostvariti vezu na oba ulaza. Upisivanje jedinice se vrši dovođenjem signala S = 1, dok se upisivanje nule obavlja dovođenjem signala R = 1. Znatno bolje kolo za prihvatanje podataka i njihov prenos, predstavlja **D** flip-flop (**data flip-flop**). Ovaj flip-flop osigurava da ne dođe do istovremene pobude na oba ulaza. Logička struktura, tabela istinitosti i simbolička oznaka ovog flip-flopa, dati su na slici 3.29.



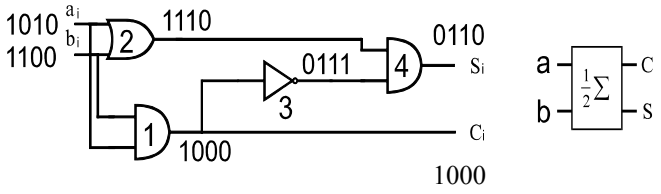
Slika 3.29. **D** flip-flop

Ulaz sa oznakom **T** naziva se sinhronizacioni ulaz, ili takt signal (clock). Treba napomenuti da sva digitalna logička kola u jednom računaru najčešće rade pod dejstvom jednog jedinstvenog vremenskog signala, koji obezbeđuje jednovremenost, sinhronizaciju u čitavom računaru. Učestanost takt signala se meri jedinicom koja se zove megaherc (**MHz**), a to je milion impulsa u sekundi. Ako je takt signal **50 MHz**, to znači da ima 50 miliona impulsa u sekundi, ili jedan impuls svakih **20** nano sekundi. Takt signali se generišu pomoću posebnih elektronskih sklopova koji se zovu oscilatori ili takt generatori. Takt signal se uz pomoć kola za kašnjenje može podeliti na manje vremenske intervale. Isto tako, uz pomoć specijalnih digitalnih kola, takozvanih delitelja, takt intervali vremena se mogu povećavati. Na taj način se dobijaju impulsi različitog trajanja i učestanosti, a koriste se u upravljanju svim elementarnim operacijama u centralnom procesoru i ostalim delovima računara.

3.6. ELEMENTARNA ARITMETIČKA KOLA

Pomćenje podataka je jedna od funkcija koju obavljaju digitalna kola u računaru. Ona takođe mogu biti namenjena i obavljanju binarnih aritmetičkih operacija. Na slici 3.30 prikazana je logička mreža poznata pod nazivom polusabirač (**half adder**), njegova simbolička oznaka i tabela istinitosti. Polusabirač je realizovan uz pomoć I, ILI i NE logičkih kola. Ova logička mreža vrši sabiranje dva bita, dve binarne cifre. Rezultat sabiranja binarnih brojeva sadrži dve komponente: zbir $\mathbf{S_i}$ i prenos u sledeći razred $\mathbf{C_i}$.

ulazi		prenos	zbir
$\mathbf{a_i}$	$\mathbf{b_i}$	$\mathbf{C_i}$	$\mathbf{S_i}$
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0



Slika 3.30. Tabela istinitosti, logička mreža polusabirača, blok dijagram

Ulazni podaci $\mathbf{a_i}$, $\mathbf{b_i}$ stupaju na ulaze logičkih kola, koja na osnovu njihovih trenutnih vrednosti generišu signale na svojim izlazima. Kada su oba ulazna signala nula, na izlazu prvog I kola (1) generiše se prenos: $\mathbf{C_i = 0}$. Ista dva ulazna signala dolaze i na ulaz ILI kola (2), koje takođe na svom izlazu daje nulu. NE kolo (3) prihvata bit prenosa i pretvara nulu sa svog ulaza u jedinicu na svom izlazu. Ova jedinica zajedno sa nulom iz ILI kola (2) dolazi na ulaze drugog I kola (4) koje na svom izlazu generiše nulu, signal na izlazu I kola 4 predstavlja zbir $\mathbf{S_i}$ binarnih cifara $\mathbf{a_i}$ i $\mathbf{b_i}$ sa ulaza. Kada je $\mathbf{a_i = 1}$ a $\mathbf{b_i = 0}$, prvo **I** kolo konvertuje ulazne podatke u nula bit prenosa na svom izlazu ($\mathbf{C_i = 0}$). ILI kolo konvertuje te iste ulazne podatke u jedinicu na svom izlazu. Invertor NE prihvata bit prenosa i pretvara ga u

jedinicu, koja zajedno sa jedinicom iz ILI kola dolazi na ulaze drugog I kola. Drugo I kolo na osnovu dve jedinice na svojim ulazima pravi jedinicu na svom izlazu, koja predstavlja zbir $\mathbf{S_i = 1.}$ U trećem slučaju za ulazne podatke $\mathbf{a_i = 0}$ i $\mathbf{b_i = 1}$, postupak obrade i rezultat su isti, tj.: $\mathbf{S_i = 1}$ i $\mathbf{C_i = 0}$. U četvrtom slučaju, treba sabrati dve binarne jedinice, $\mathbf{a_i = 1}$ i $\mathbf{b_i = 1}$. Kada na ulaz prvog I kola dođu dve jedinice, ovo kolo na svom izlazu daje bit prenosa koji je takođe jednak jedinici, $\mathbf{C_i = 1}$. ILI kolo prihvata dve jedinice na svojim ulazima, i na izlazu daje takođe jedinicu. Bit prenosa stupa na ulaz invertora NE, koji od jedinice pravi nulu na svom izlazu. Ova nula zajedno sa jedinicom sa izlaza ILI kola dolazi na ulaze drugog I kola, koje na izlazu daje zbir $\mathbf{S_i = 0}$.
Znači, rezultat je:

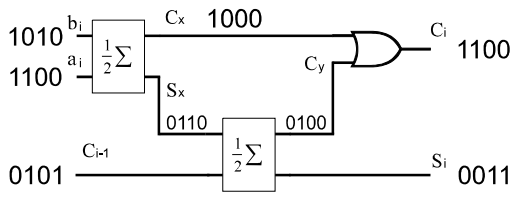
$$\mathbf{S_i = 0} \text{ i } \mathbf{C_i = 1}.$$

Polusabirač je pogodan jedino za sabiranje samo dva bita, ali ne može sabrati i prenos iz prethodnog razreda. No, kod mnogih računara se i za ovakva sabiranja koristi drugačije rešenje. Prenos iz prethodnog razreda se može javiti i kod sabiranja višecifrenih binarnih brojeva. Sabiranje brojeva sa učešćem bita prenosa iz prethodnog razreda obavlja, se uz pomoć digitalne mreže koja se zove potpuni sabirač, ili sumator (**full adder**). Šema potpunog sabirača i njegova tablica istinitosti dati su na slici 3.31.

Kao što se vidi sa slike potpuni sabirač se sastoji od dva polusabirača i jednog logičkog **ILI** kola. Na ulaze jednog polusabirača dolaze binarne cifre $\mathbf{a_i}$ i $\mathbf{b_i}$, i on na svojim izlazima daje delimičan zbir $\mathbf{S_x}$ i prenos $\mathbf{C_x}$. Na ulaze drugog polusabirača dolaze delimični zbir $\mathbf{S_x}$ i prenos iz prethodnog razreda $\mathbf{C_{i-1}}$. Drugi polusabirač na svom izlazu daje potpuni zbir **Si** binarnih cifara $\mathbf{a_i}$ i $\mathbf{b_i}$ i prenosa iz prethodnog razreda $\mathbf{C_{i-1}}$, i delimični prenos $\mathbf{C_y}$.

Konačni prenos u sledeći razred $\mathbf{C_i}$, dobija se na izlazu iz **ILI** kola.

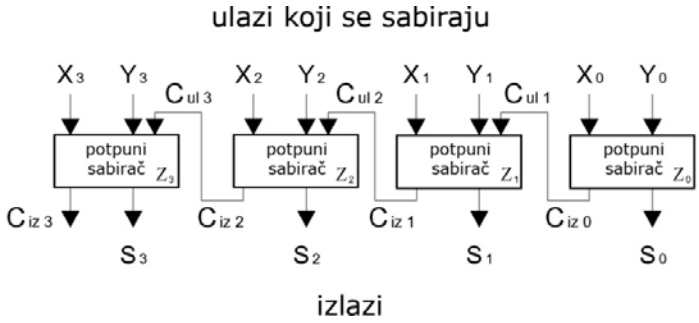
$\mathbf{a_i}$	0	**0**	**0**	0	1	**1**	**1**	1
$\mathbf{b_i}$	0	**0**	**1**	1	0	**0**	**1**	1
$\mathbf{C_{i-1}}$	0	**1**	**0**	1	0	**1**	**0**	1
$\mathbf{S_i}$	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**
$\mathbf{C_i}$	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**1**



Slika 3.31. Tabela itinitosti i blok dijagram potpunog sabirača

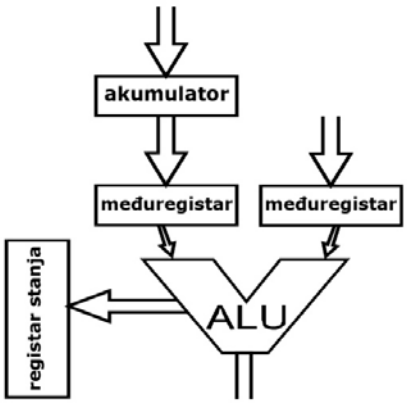
Ovakav proces izračunavanja ponavlja se za sve razrede binarnog broja. Nekada su se zbog visoke cene hardvera pravili i računari koji su imali samo jedan sabirač na čije ulaze su se dovodile binarne cifre i prenos iz prethodnog razreda jedan za drugim, tj serijski. Ovakav sabirač se zvao serijski ili redni sabirač, a sabiranje osmobitnog broja se odvijalo u osam taktova. Danas se za svaki binarni razred koristi poseban sabirač. Tako za sabiranje 32-bitnih brojeva postoji jedan

polusabirač za sabiranje **LSB**, i niz od 31-og potpunog sabirača za sabiranje preostalih botova, ili niz od 32 potpuna sabirača, na čije ulaze istovremeno dolaze svih 32 bita brojeva koji se sabiraju. Ovakvo sabiranje se zove paralelno sabiranje, a deo aritmetičko-logičke jedinice koja to obavlja zove se paralelni sabirač, slika 3.32.



Slika 3.32. Blok dijagram paralelnog četvorobitnog sabirača

Aritmetičko-logička jedinica (**ALU**) obavlja aritmetičke i logičke operacije nad binarnim brojevima i rotiranje. To mogu biti operacije sabiranja, oduzimanja, množenja i deljenja binarnih i BCD brojeva, logičke operacije, razne vrste pomeranja binarnih brojeva ulevo i udesno, i možda još poneka jednostavna operacija, ali obično ne mnogo više od nabrojenog. Pomoću tih osnovnih operacija rešavaju se zadaci postavljeni računaru. Aritmetičko-logička jedinica se obično šematski predstavlja u obliku slova **V**, kao što je prikazano na slici 3.33.



Slika 3.33. Šematski prikaz ALU sa pripadajućim registrima

Pored sabirača, u aritmetičko-logičkoj jedinici se nalaze i mnogi drugi sklopovi. Tu su logičke mreže koje obavljaju operacije **I, ILI**, **iskučivo ILI** i **NE** nad svim ciframa binarnih brojeva istovremeno, zatim pomerački registri, kola za komplementiranje itd. Kod nekih velikih računara u sastav ALU ulaze i sklopovi za BCD aritmetičke operacije, sklopovi za računanje u pokretnom zarezu. Mnogi autori u sastav ALU uključuju i neke registre specijalne namene u kojima se nalaze

operandi i rezultat operacije, to su privremeni registri i akumulatori, kao i jedan specijalni registar u koji se automatski upisuju određeni podaci nakon svake operacije koju izvrši centralni procesor. Taj registar u raznim računarskim sistemima ima drugačije ime, a mi ćemo ga zvati registar stanja.

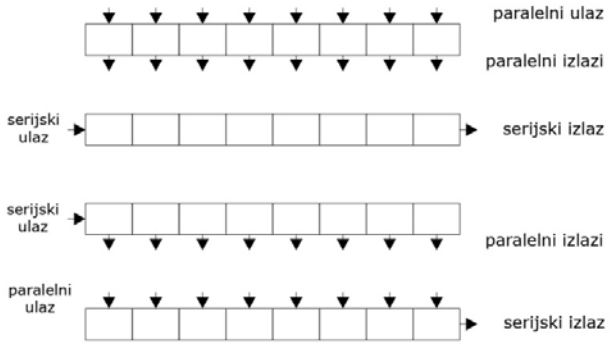
3.7. REGISTRI

Najčešće korišćeni elementi digitalnih uređaja su različite vrste registara. Registar je element koji služi za skladištenje proizvoljnog binarnog broja ograničene dužine. Svaka binarna cifra datog broja čuva se u zasebnom memorijskom elementu, pa je za binarni broj od **n** cifara potrebno **n** binarnih memorijskih ćelija. Registri se primenjuju uglavnom za privremeno memorisanje ili prihvatanje delimičnih ili konačnih rezultata u procesu obrade podataka. Neophodni su na svim mestima gde treba ostvariti vezu između blokova sa različitim brzinama, zatim pri realizovanju aritmetičkih operacija, za pretvaranje serijskog u paralelni kod i obratno, itd. Binarni broj koji se nalazi u registru naziva se sadržaj registra. Za ulaz binarnog broja u registar koristi se termin upisivanje, dok se za izlaz broja iz registra koristi termin čitanje.
Upisivanje i čitanje sadržaja može da se uradi na dva načina:

- paralelno, kada sve cifre binarnog broja ulaze u registar ili izlaze iz njega istovremeno,
- serijski, kada broj ulazi, odnosno izlazi iz registra cifra po cifra (**bit po bit**), počevši od najmlađeg ili najstarijeg razreda.

Kombinacijom opisanih načina ulaza i izlaza informacija razlikujemo različite tipove registara, slika 3.34:

1. registri sa paralelnim ulazom i paralelnim izlazom,
2. registri sa serijskim ulazom i serijskim izlazom,
3. registri sa serijskim ulazom i paralelnim izlazom,
4. registri sa paralelnim ulazom i serijskim izlazom.



Slika 3.34. Osnovne vrste registara

Kod prvog tipa registara upisana informacija ostaje stalno u njemu, i oni se zovu stacionarni registri. Kod ostala tri tipa registara sadržaj registra se pomera kako pristižu novi bitovi, i ukoliko se ne očita na vreme, on može biti i izgubljen. Zbog toga se ovi registri nazivaju dinamički, pomerački ili šift-registri (**shift registers**).
U sastav centralnog procesora ulaze pored ALU i kontrolno-upravljačka jedinica i veći broj registara.

Ovi registri se dele na dve grupe saglasno njihovoj nameni:

- registri opšte namene,
- registri specijalne namene (akumulatori, me|uregistri, registar stawa (status registar), programski brojač, registar instrukcija, registar podataka, adresni registri i niz drugih registara). Na ovom mestu ćemo pomenuti samo neke koji su trenutno od interesa a uloga ostalih će biti objašnjena u okviru obrada u kojima učestvuju.

Akumulator (**accumulator**) je registar u kojem se obično dobijaju (akumuliraju) rezultati različitih operacija s binarnim brojevima. Podaci u akumulator dolaze sa magistrale za podatke, a iz akumulatora se prenose na međuregistar. U sastavu procesora može biti jedan ili više akumulatora.

Međuregistri (**buffers**) privremeno "pamte"-prihvataju podatke, i to jedan (na slici 3.33 levo) podatke iz akumulatora, a drugi (desni) podatke koji dolaze pravo sa magistrale. Takvo privremeno pamćenje podataka potrebno je da bi se po dva podatka mogla dovesti na ulaz aritmetičko-logičke jedinice, koja ih obrađuje u prikladnom momentu. Pored ovih međuregistara u procesoru se nalaze i memorijski adresni registar i prihvatni registar podataka, čiju ulogu ćemo objasniti u okviru memorije. Prema tome, međuregistri imaju samo pomoćnu ulogu, pa se ponekad pri razmatranju prenosa podataka mogu izostaviti.

Registri opšte namene **(R_O-R_N)** postoje u većini računara, sem kod nekih mikroračunara. Broj ovih registara varira od računara do računara, pa ih mogu imati od jednog ili dva do nekoliko desetina. Na tim registrima privremeno se zapisuju različiti podaci, i zato su to registri opšte namene. Oni su registri "pri ruci", pa je pristup do njih jednostavniji i brži nego do registra u memoriji. Vreme pristupa podacima unutar ovih registara je višestruko kraće nego kada se pristupa podacima u memoriji. Stoga se njihovom upotrebom ubrzava rad procesora. No ovi registri su istovremeno vrlo skupi jer se izrađuju u specijalnim poluprovodničkim tehnologijama. Podaci na magistralu dolaze ili iz prihvatnog registra podataka, ili iz registara opšte namene ili iz aritmetičko-logičke jedinice. Podaci sa magistrale idu na registre opšte namene, na akumulator, ili na neki od međuregistara.

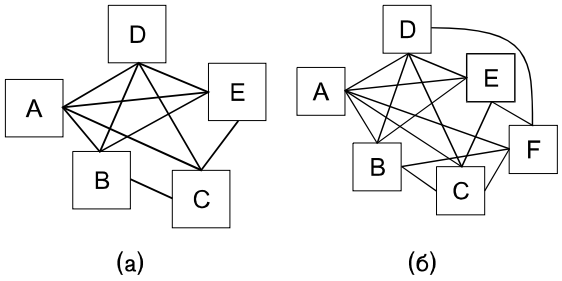
Registar stanja (registar statusa, registar uslova), služi za to da se na njemu prikažu različita stanja koja mogu nastati tokom obrade podataka. U stvari, taj se registar sastoji od niza međusobno nezavisnih flip-flopova takozvanih zastavica (**flags**). Svaka od tih zastavica signalizira neko stanje koje je nastalo tokom obrade

podataka. To mogu biti, na primer ova stanja: *rezultat je jednak nuli, omogućen prekid programa, došlo je do prepunjenja, postoji bit prenosa* itd. Takva stanja mogu uticati na dalje odvijanje rada. Te zastavice omogućuju korisniku (programeru) da proverom stanja određene zastavice usmeri odvijanje programa.

Adresni registri (**address registers**) služe za čuvawe adresa memorijskih lokacija u kojima se nalaze instrukcije (brojač instrukcija), ili adresa lokacija u kojima se nalaze podaci (brojač podataka, pokazivač steka, indeks registri), ili pak sadrže dodatne informacije pomoću kojih se izračunavaju adrese i podataka i instrukcija (bazni i segmentni registri).

3.8. MAGISTRALE

Da bi računar mogao da radi, potrebna je komunikacija između logičkih kola, flip-flopova, pomeračkih registara i drugih komponenti, koje ulaze u sastav centralnog procesora. Komunikacija mora postojati i između centralnog procesora, memorija, ulaznih i izlaznih jedinica, i drugih uređaja koji ulaze u sastav računarskog sistema. Neke od komponenti računara ostvaruju veze sa samo nekoliko drugih sklopova, dok neke druge zahtevaju povezivanje sa velikim brojem drugih delova računarskog sistema. Neki računarski sistemi dopuštaju direktno povezivanje svih komponenti, takozvano povezivanje od tačke do tačke (**point to point**), prikazano na slici 3.35.

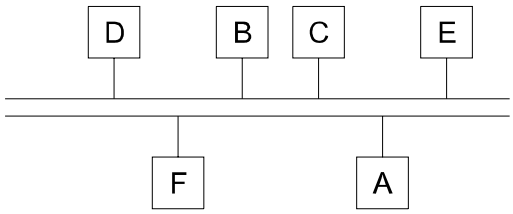


Slika 3.35. Povezivanje od tačke do tačke i dodavanje nove komponente

Ako bi komponente koje treba povezati bili registri sa **n** botova, onda je za povezivanje **p** registra potrebno: (**p-1**). **p**.**n/2** linija veze. Pri dodavanju makar i samo jedne nove komponente treba ostvariti veliki broj novih veza, pa je rekonfiguracija sistema praktično vrlo teško izvodiva. Prednost ove strukture je velika brzina prenosa, jer se istovremeno može ostvariti veći broj međusobnih veza jednovremeno (povezivanje više registara) i pouzdanost, jer se u slučaju otkaza direktne veze, veza između dva registra može uspostaviti preko drugih registara.

Rešenje problema povezivanja više komponenti jeste u korišćenju zajedničkih komunikacionih linija koje se zovu sabirnice ili magistrale (**bus**), kao što je prikazano na slici 3.36.

Grupa linija preko kojih se informacije u binarnom obliku prenose između registara u sastavu centralnog procesora, naziva se interna magistrala. U ovakvoj organizaciji CPU duž jedne magistrale u jedinici vremena, moguće je ostvariti samo jedan prenos, ali je zato broj linija jednak **n,** i ne zavisi od broja registara (i drugih digitalnih sklopova) koji su na nju povezani. Da bi se ostvario prenos između dva registra posredstvom magistrale, mora se putem jedne grupe linija - adresne magistrale (**address bus**), poslati binarni broj koji označava adresu registra koji u tom trenutku sudeluje u prenosu podataka, a čiji podaci se nalaze (ili će se naći) na magistrali podataka (**data bus**). Da bi se sve odvijalo kako treba, brine neka upravljačka jedinica (najčešće u sastavu **CPU**) koja preko upravljačke magistrale (**control bus**), šalje upravljačke signale koji određuju smer prenosa podataka i signale koji usklađuju događaje u vremenu. Iako su ovo po funkciji tri različite magistrale, često se na crtežima prikazuju kao jedna.



Slika 3.36. Generalisana struktura magistrale

Mnogi računari imaju nekoliko raznih magistrala, to su takozvani **multiple-bus** računari. Kod većine takvih računara postoje dva osnovna sistema magistrala. Jedan su interne, tj. unutrašnje magistrale (slika 1.4), koje služe za prenos podataka između glavne memorije i različitih delova centralnog procesora, (ALU, raznih registara i sl.). Drugi čine spoljašnje magistrale koje povezuju centralni procesor sa ostalim delovima računarskog sistema. Korišćenje dve magistrale povećava brzinu rada i omogućava paralelno odvijanje memorijskih i ulazno-izlaznih operacija. Ali unutrašnje i spoljne magistrale su međusobno povezane, jer samo na taj način može se usklađeno odvijati rad procesora i ostalih jedinica koje čine računarski sistem. Povezivanje ovih magistrala obavlja se pomoću ulazno-izlaznih međusklopova (**interfaces**).

Druga vrsta računara ima samo jednu magistralu na koju su povezani svi delovi računarskog sistema. To su takozvani **single-bus** računari (slika 1.3). Prednosti ovakve arhitekture jesu mala cena i jednostavno dodavanje novih periferijskih uređaja. Normalno, to je plaćeno smanjenjem brzine rada računara. Na osnovu ovih razmatranja možemo zaključiti da se jedna magistrala koristi obično kod malih računara koji se zovu mini i mikroračunari, a da veliki računari obično koriste sisteme sa više magistrala.

Iako je na magistralu povezano mnoštvo komponenti, u jednom trenutku vremena, u prenosu podataka učestvuju samo dve. Ostali sklopovi ne učestvuju u prenosu, u

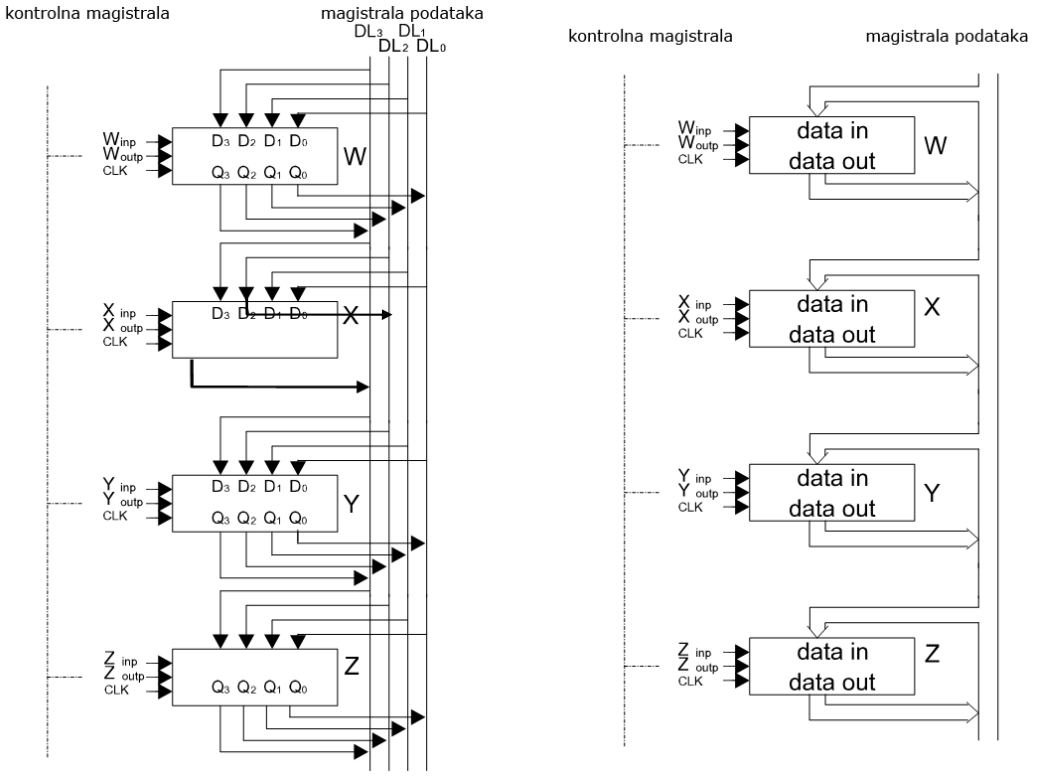
datom trenutku, ali i ne smeju ometati prenos između aktivnih komponenti. Ovo je moguće ostvariti zahvaljujući primeni **buffer** registara i specijalne klase logičkih kola u izradi računarskih sklopova. To su logička kola sa tri stanja, tj. ova kola mogu biti u stanju logičke nule, jedinice i beskonačne impedanse.

Kada su digitalna kola u stanju beskonačne impedanse, ona kao da nisu povezana na magistralu, tj. kao da ne postoje. Prenos podataka duž magistrala dodatno komplikuje činjenica da svi delovi računarskog sistema koji su povezani na magistralu nemaju istu brzinu rada. Neki elektromehanički uređaji su relativno spori, na primer: printeri, terminali, ploteri, tastature i sl. Diskovi i trake su znatno brži od njih, ali kudikamo sporiji od operativne memorije ili procesora. Kako svi ovi uređaji moraju da komuniciraju preko magistrale, neophodno je obezbediti efikasan mehanizam kojim će se premostiti razlike u njihovoj brzini rada.

Rešenje je nađeno u upotrebi međuregistara (**buffer registers**) koji se ugrađuju u ove uređaje da bi privremeno čuvali podatke za vreme prenosa. Pogledajmo to na primeru prenosa jednog karaktera (znaka) iz procesora u printer gde će biti odštampan. Procesor započinje prenos slanjem tog znaka preko magistrale na međuregistar (**buffer**) za printer. Pošto je **buffer** elektronski sklop ovaj prenos zahteva malo vremena. Kada je među-registar napunjen, printer može da počne sa štampanjem bez dodatnih intervencija od strane procesora.

Dakle, procesor i magistrala više nisu potrebni printeru i mogu se koristiti za druge aktivnosti. Printer nastavlja sa štampanjem karaktera koji je zapisan u njegovom **buffer**-u, i nije na raspolaganju za druge zahteve za štampanjem dok ne završi štampanje onog znaka koji se nalazi u njegovom **buffer**-u. Da zaključimo, međuregistar eliminiše razlike u brzini rada procesora i printera i onemogučava da spori uređaji blokiraju rad brzih uređaja, odnosno da procesor čeka dok printer ne završi sa štampanjem.

Ovo omogućava procesoru da se brzo prebacuje sa jednog uređaja na drugi, i da praktično istovremeno upravlja prenosom podataka sa i u više raznih uređaja. Broj magistrala bitno utiče na organizaciju centralnog procesora, a karakteristična su tri tipa organizacije: oko jedne, oko dve i oko tri magistrale. Pre toga pokažimo kako se posredstvom jedne četvorobitne magistrale $\mathbf{DL_0}$ **-** $\mathbf{DL_3}$, na koju su povezana četiri registra **X, Y, Z, W**, vrši prenos podataka između dva registra, slika 3.37 a).



Slika 3.37. Primer a) četvorobitne magistrale i b) i hipotetičke računarske magistrale

Šta se događa na magistrali i koji su upravljački impulsi potrebni da se izvrši prenos podatka iz registra **Z** u registar **X** (koji ima paralelni ulaz i serijski izlaz)**.** Svaka linija podataka sadrži po jedan bit. Neka je u registru **Z** zapamćen broj 0110.

Postupak prenosa je sledeći:

- poslati $\mathbf{Z}_{outp}$ upravljački signal registru Z,
- u sledećem ciklusu takt signala, čita se podatak iz registra Z i šalje se na magistralu,
- podatak je na magistrali i dostupan je svim registrima X, Y, Z i W,
- poslati $\mathbf{X}_{inp}$ upravljački signal na registar X,
- u sledećem ciklusu takta **CLK**, registar X prihvata podatke sa magistrale.

Takt signal, dolazi na ulaz svih registara, odnosno sva digitalna kola u računaru su taktovana. Mada su svi registri povezani na magistralu aktivan je samo onaj koji je dobio odgovarajući upravqački signal **OUTP**, a sva ostala kola su u stanju beskonačne impedanse, tj. njihovi izlazi kao da nisu povezani na magistralu. Signal

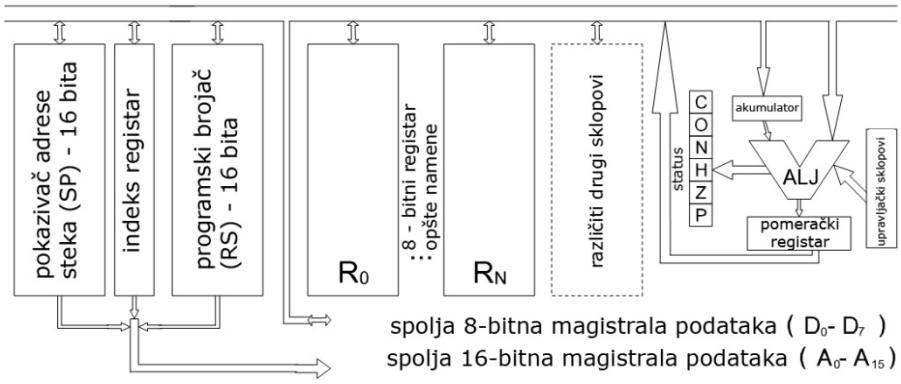
OUTP traje sve do završetka prenosa podataka. Iako su ulazi svih registara povezani na magistralu, podatak će se preneti samo u onaj registar koji dobije upravljački impuls INP. Ponekad je pre upisa novog podatka u registar potrebno prvo izvršiti brisanje starog sadržaja, tj. upisati 0 u sve ćelije registra. Na slici 3.37 (b) prikazana je hipotetička magistrala podataka u računaru, gde se zbog preglednosti ne ucrtavaju linije svakog bita ponaosob, jer bi to bilo krajnje nepraktično (i nerazumljivo) u slučaju većeg broja botova (npr. 16 ili 32 i više).

Bez obzira na to o kojoj se magistrali radi, na nju je priključen veći broj registara, pri čemu svaki od njih u principu može biti ili izvor ili odredište podataka. Pri svakom prenosu učestvuju samo dva registra. Ostali, iako priključeni na magistralu, ne učestvuju u prenosu. U svakom procesoru može se nalaziti jedna ili više magistrala. Svaka magistrala sadrži tri grupe linija: jednosmerne linije za prenos signala adrese - adresna magistrala (adrese memorijskih lokacija ili ulaznih i izlaznih uređaja), i dvosmerne linije za prenos podataka i upravljačkih signala - magistrala podataka i upravljačka magistrala. Budući da broj magistrala znatno utiče na organizaciju procesora i računara, prikazaćemo karakteristične tipove organizacije procesora oko jedne, dve ili tri magistrale.

ORGANIZACIJA PROCESORA OKO JEDNE MAGISTRALE

Pri gradnji računarskih sistema radi se isključivo sa spoljnim magistralama, tj. onim koje se nalaze van procesora. Međutim, pri razmatranju interne organizacije procesora odlučujuću ulogu imaju unutrašnje magistrale, tj. one koje se nalaze u centralnom procesoru, i povezuju različite elemente unutar procesora. Razmotrimo najpre povezanost arhitekture računara sa brojem internih magistrala.

Postoje procesori koji imaju jednu internu magistralu. Takvi procesori imaju najjednostavniju arhitekturu. Primer računara, odnosno njegovog procesora organizovanog oko jedne magistrale prikazan je na slici 3.38. Na slici je, radi jednostavnosti, prikazana samo magistrala za podatke, iako je očigledno da ona može raditi samo zajedno sa internim adresnim i upravljačkim signalima, koji postoje, ali nisu nacrtani jer nas u ovom času zanima samo tok podataka. Prema tome jednom magistralom se vremenski multipleksirano (jedan za drugim) prenose svi podaci. Sa magistralama je povezan određeni broj različitih registara, kao što su akumulator, registri opšte namene $\mathbf{R_0}$ do $\mathbf{R_N}$, međuregistri, registar stanja, pa aritmetičko-logička jedinica. Kada u procesoru postoji samo jedna magistrala, u jednom trenutku vremena se može obavljati samo jedan prenos, jer se na magistrali ne može istovremeno nalaziti više različitih podataka.



Slika 3.38. Organizacija procesora oko jedne magistrale, tipična organizacija mikroprocesora

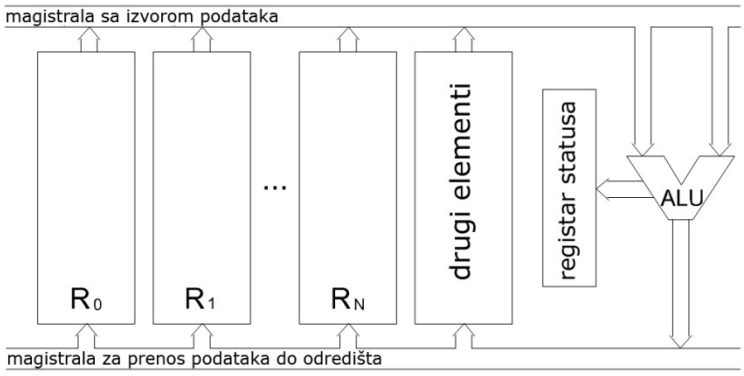
Procesor koji ima samo jednu magistralu podataka radi nešto sporije u poređenju sa sistemima koji imaju dve ili više magistrala koje mogu, između različitih elemenata računara, istovremeno prenositi po dva, pa čak i više podataka.

ORGANIZACIJA PROCESORA OKO DVE I TRI MAGISTRALE

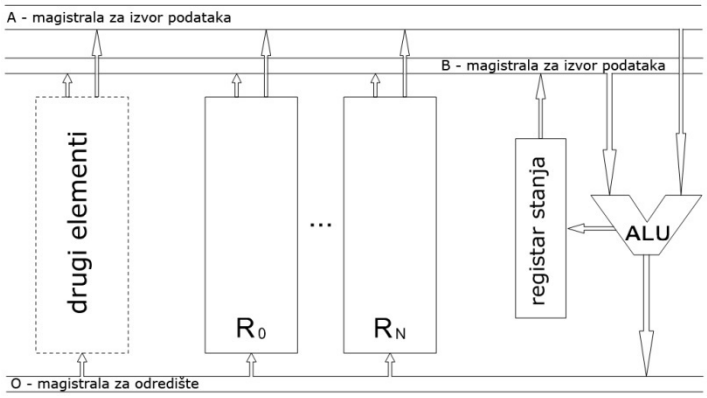
Primer procesora organizovanog oko dve magistrale dat je na slici 3.39. Jedna magistrala služi za podatke koji dolaze na ulaz u **ALU** iz registra opšte namene i drugih elemenata. Rezultati obrade iz **ALU** dolaze na drugu magistralu, preko koje se mogu uputiti na bilo koje odredište. Kad se upotrebe dve magistrale, podaci se mogu istovremeno prenositi po obe magistrale, što ubrzava rad računarskog sistema.

Primer organizacije procesora oko tri magistrale prikazan je na slici 3.40. Svaki ulaz u aritmetičko-logičku jedinicu ima svoju magistralu za ulazne podatke. Zbog toga ovde, u principu, ne bi ni bili potrebni međuregistri na ulazu u **ALU**, jer bi se ulazni podaci mogli toj jedinici istovremeno proslediti direktno sa magistrale.

Većina proizvođača ipak stavlja međuregistre, jer su potrebni radi pamćenja ulaznih podataka, ali i iz drugih razloga. Izlazni podaci prenose se na odredište posebnom magistralom. Očito da takva organizacija omogućuje još brži rad računara, jer se istovremeno može prenositi više podataka, svaka magistrala prenosi svoje podatke.



Slika 3.39. Organizacija procesora oko dve magistrale



Slika 3.40. Organizacija procesora oko tri magistrale

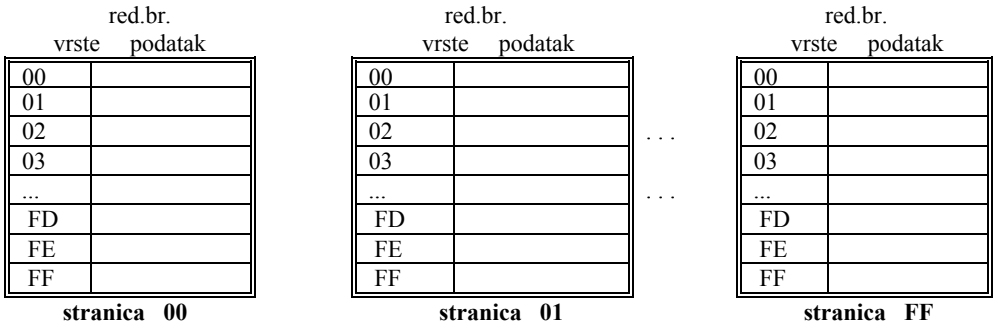
Ovde treba napomenuti da se u mikroračunarima i mikroprocesorima najčešće koristi jednosabirnička arhitektura sa slike 3.38.

3.9. MEMORIJE

Memorija je specijalni hardver namenjen za smeštanje binarnih podataka (tj. upis), s ciljem čuvanja podataka do momenta uzimanja (tj. čitanja) podataka radi dalje obrade. Radnje upisa i čitanja podataka predstavljaju aktivnost koja se zove pristup memoriji (**memory access**). Što je kraće vreme pristupa to su memorije brže. Memorije nisu ništa drugo do određeni, obično vrlo veliki broj registara, povezanih u jednu celinu. U svaki od tih registara može se zapisati jedan binarni podatak (broj), koji ima onoliko binarnih cifara, koliko je duga reč računara. Za razumevanje rada računara od osnovnog značaja je razumevanje načina memorisanja programa i podataka, i načina pronalaženja željenih podataka i instrukcija u unutrašnjoj memoriji računara. Po već opisanoj analogiji automatske i ručne obrade podataka, memoriju treba shvatiti kao svesku, gde je svaka stranica

numerisana. Na svakoj stranici ucrtana je tabela sa velikim brojem vrsta i samo dve kolone, kao na slici 3.41.

Prva kolona sadrži redni broj vrste, a u drugu kolonu se upisuju podaci. Pri tome, u jednoj vrsti, u određenom trenutku, može biti upisan samo jedan podatak. Kada u neko poqe upišemo novi podatak, prethodno memorisani podatak se nepovratno gubi. Naime, da bi upisali novi podatak u tabelu moramo najpre obrisati stari podatak.



Slika 3.41. Korišćenje sveske za skladištenje podataka.

Ako sve stranice poređamo u neprekidan niz (jednu ispod druge), dobijamo memoriju u obliku celovitog bloka kao na slici 3.42. U jednu memorijsku lokaciju upisuje se jedna binarna reč. Broj botova u jednoj reči predstavlja dužinu memorijske reči.

Redni broj vrste i stranice služe za pronalaženje podataka i nazivaju se adresa podatka, ili adresa lokacije podatka. Vrsta u kojoj se nalazi zapisan podatak naziva se lokacija. Sam broj koji je zapisan na nekoj lokaciji predstavlja podatak.

Adresa lokacije podatka, odnosno redni broj vrste u svesci (memoriji), predstavlja fizičku - efektivnu adresu lokacije, i sastoji se od dve komponente:

- rednog broja stranice na kojoj se nalazi podatak i
- rednog broja vrste na stranici (tj. rednog broja vrste u užem smislu), {to predstavlja neku vrstu prividne - virtuelne adrese lokacije.

Ukupan broj vrsta na svim stranicama zajedno, predstavlja maksimalni broj podataka koji se uopšte mogu u jednom trenutku zapisati, i naziva se kapacitet memorije. Kada je adresa lokacije podatka prikazana u obliku rednog broja te vrste, onda je to fizička adresa podatka.

Da bi smo pronašli neki podatak moramo znati (ili nekako izračunati) njegovu fizičku adresu.

0000	1	1	1	0	0	1	0	1	0	0	0	0	0	1	1	1
0001	0	0	0	0	0	1	1	0	1	0	1	0	1	0	0	0
0002	1	0	1	0	1	0	1	0	1	0	0	1	1	1	0	1
...	...															
00FE	0	0	1	0	1	0	1	0	1	1	0	0	1	1	1	1
00FF																
0100																
0101	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0102	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
...																
01FE																
01FF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0200																
0201																
0202																
...																
FFFE	1	1	0	0	0	1	0	1	0	0	1	1	1	1	0	0
FFFF	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0

Slika 3.42 Unutrašnja, glavna memorija računara

Da bi sebi olakšao i ubrzao rad, čovek pri memorisanju i obradi velikog broja podataka primenjuje različite šablone. Tako recimo, podatke određenog tipa, smešta uvek u tačno određene vrste, pa umesto da u program obrade upisujemo sam podatak, upisujemo redni broj vrste (tj. adresu), gde se taj podatak nalazi. Ima još nekoliko razloga zbog kojih se u program obrade ne upisuje sam podatak već njegova adresa.

Prvi razlog je, što se neki podaci tokom vremena često menjaju, (na primer, plata radnika se svakog meseca menja), dok su postupci obrade u jednoj aplikaciji relativno nepromenljivi ili se retko menjaju (na primer, uvek nam treba da izračunamo prosečnu platu zaposlenih i slično). Ali u opštem slučaju podaci su znatno manje promenljivi nego postupci obrade, pa se najčešće jedni te isti podaci obrađuju na više raznih načina u cilju dobijanja različitih informacija.

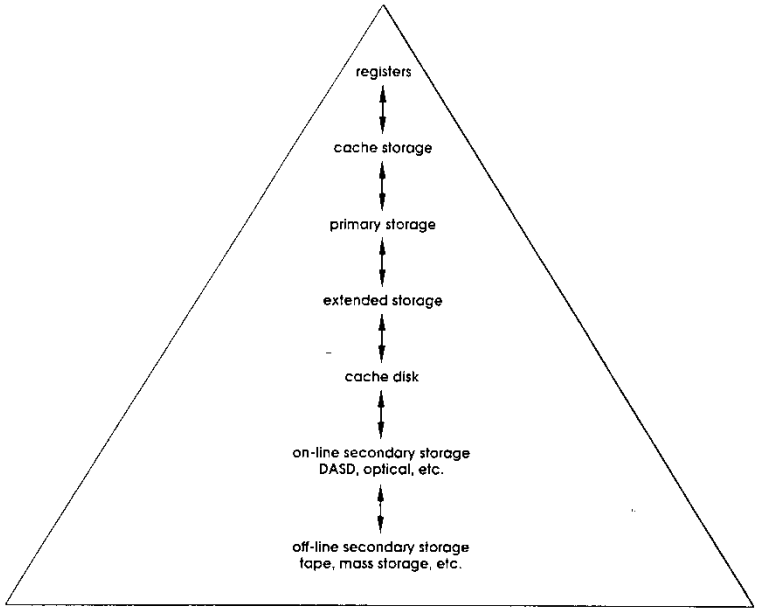
Drugi razlog je što, u postupcima obrade čitave grupe podataka (veliki broj podataka istog tipa), bivaju obrađene na isti način (zaposlenih može biti od nekoliko pojedinaca do nekoliko hiljada, i svi se obrađuju na isti način).

Treći razlog je što različiti podaci mogu imati istu brojnu vrednost, pa bi direktno pisanje podataka u program obrade, taj program učinilo: manje razumljivim, manje preglednim, manje univerzalnim i program bi bio više podložan greškama.

Konačno, samom čoveku koji je neizostavni sastavni deo računarskog sistema, vrlo je teško da duže vreme pamti veliki broj adresa podataka, odnosno mnogo višecifrenih brojeva. Čoveku je zato znatno lakše da podatke sistematisuje po nekom kriterijumu, i da im pridruži neka simbolička imena, koja na neki način

asociraju na njihovu prirodu i način upotrebe (tj. obrade koja se nad tim podatkom vrši). Upotreba simboličkih imena podataka u programu obrade, umesto samih podataka ili njihovih fizičkih adresa, naziva se simboličko adresiranje. Napomenimo da svi programski jezici, izuzev mašinskog jezika, omogućavaju programerima upotrebu simboličkih imena.

Svaki računarski sistem poseduje različite vrste memorijskih ure|aja. Neki su vrlo brzi, a drugi su spori. Uređaji kao što su magnetni i optički diskovi imaju vrlo veliki kapacitet, dok npr. registri sadrže samo jedan bajt ili reč. Kapacitet i vrste memorija variraju od sistema do sistema. Na slici 3.43 je data opšta hijerarhijska šema raspoloživih uređaja za memorisanje.



Slika 3.43. Hijerarhija uređaja za skladištenje podataka

Šema se naziva hijerarhijska jer su uređaji poređani u rastućem redosledu odozdo na gore sa aspekta brzine rada. Susedni memorijski uređaji na hijerarhijskoj šemi u normalnim uslovima rada direktno komuniciraju, tj. prenos podataka ostvaruje se samo između dva susedna nivoa. Brzina prenosa podataka je uvek u obrnutoj srazmeri sa kapacitetom, što znači da je kapacitet utoliko veći što je uređaj dalje od vrha. Vidimo da u računarskim sistemima postoji mnoštvo različitih uređaja čija je osnovna, ali ne i jedina namena pamćenje i čuvanje podataka za kasniju upotrebu.

Među njima postoji sledeća hijerarhija:

- registri (**registers**),
- skrivena memorija (**cache storage**),
- primarna, glavna, operativna memorija (**primary storage**),

- proširena memorija (**extended storage**),
- skriveni disk (**cache disk**)
- stalno aktivne sekundarne memorije (**on-line secondary storage**), magnetni i optički diskovi koji su uključeni u obradu podataka i sastavni su deo računarskog sistema,
- povremeno aktivne sekundarne memorije (**off-line secondary storage**, trake, masovne memorije).

Registri se nalaze u CPU i samim tim su na najvišem nivou hijerarhije. Oni su najbrža i najskuplja memorija u računarskom sistemu, no ujedno i najmanjeg kapaciteta. Imaju različitu namenu, i mogu služiti za pamćenje podataka čiji su namena i postupak obrade unapred definisani u računarskom sistemu (registri specijalne namene), dok neki drugi registri pamte podatke čija upotreba nije unapred određena (registri opšte namene). Izrađeni su u najbržim poluprovodničkim tehnologijama.

Ispod registara, jeftinije i nešto manje brze i većeg kapaciteta, su skrivene memorije, koje služe kao sprega međumemorija (**buffer**) između registara i primarne memorije. U njoj se drže ili kopije najčešće korišćenih podataka (npr.VAX11/780), ili bloka instrukcija koje slede iza tekuće instrukcije (npr. MC 68020). U centralnom procesoru postoji poseban hardver koji vodi računa koji podaci i instrukcije se nalaze u ovoj memoriji, i kada treba pribaviti nove. Tako|e, sistem mora da obezbedi da sve izmene podataka u ovoj memoriji budu preslikane i na originale u primarnoj memoriji.

Primarne memorije, još jeftinije i sporije, ali još većeg kapaciteta, sadrže instrukcije i podatke u tekućoj obradi, a koje mogu pripadati ili operativnom sistemu ili nekom od korisničkih programa. To je glavna, odnosno, operativna memorija, o kojoj ćemo u ovom poglavlju govoriti, i ona je kao i sve prethodne memorije izvedena u poluprovodničkoj tehnologiji.

U mnogim velikim računarima, primarnoj memoriji se dodaje malo sporija, proširena memorija, ogromnog kapaciteta, koja služi za držanje podataka, i programa koji su privremeno izbačeni iz primarne memorije. Brigu o dodeljivanju (alociranju) primarne i proširene memorije vodi deo operativnog sistema koji se zove upravljanje memorijom. Sličnu ulogu, kao proširena memorija, ima i skriveni disk, sa još većim kapacitetom i još manjom brzinom rada, a zadatak mu je još da čuva rezervne kopije podataka.

On-line sekundarne memorije čine tvrdi diskovi, izmenjivi diskovi i optički diskovi, dok **off-line** sekundarne memorije, čine magnetne trake i specijalne masovne memorije koje služe za dugotrajno pamćenje podataka koji su na poseban način organizovani.

GLAVNA MEMORIJA RAČUNARA

Glavna, operativna memorija se može posmatrati kao određeni broj lokacija, koje imaju svoju dužinu (tj. broj botova) i svoju adresu (tj. redni broj). Pomoću adrese pristupa se odre|enoj lokaciji sa ciljem upisa ili čitanja njenog sadržaja. Čitanje podatka iz memorije, znači zapravo kopiranje podatka iz jedne lokacije u memoriji, u neki registar izvan memorije (najčešće u neki registar CPU). Ukupni broj lokacija koje ima memorija (kapacitet), dužina memorijske reči i brzina pristupa, važni su faktori u određivanju veličine i snage računarskog sistema.

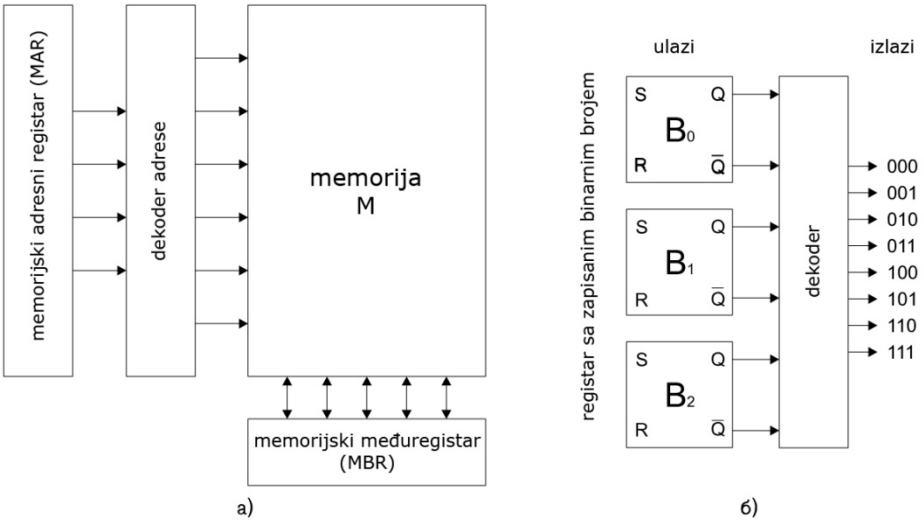
Kapacitet operativne memorije je reda 1,2,4,8,16,32 miliona memorijskih reči, mada su već u upotrebi i operativne memorije sa više od 100 megabajta (Megabyte). Dužina memorijske reči zavisi od vrste računara i varira, od 8, 16 ili 32 bita do više od 100 bita, a najčešće predstavlja umnožak od 8 botova tj. 1 bajta. Jedna memorijska reč obično sadrži 4 bajta, a dupla reč 8 bajta tj. 64 bita, mada to zavisi od tipa i proizvođača.

Kada broj lokacija pomnožimo sa dužinom reči, dobijamo ukupan broj memorijskih elemenata koje sadrži operativna memorija. Ove memorije su se nekada pravile od magnetnih jezgara, a danas se isključivo prave u poluprovodničkoj tehnologiji, koja je znatno brža i nije više tako skupa. Brzina rada memorije je jedan od odlučujućih faktora brzine računara. Ona se može okarakterisati na više načina, a jedan je vreme pristupa, tj. vreme potrebno da se dobije sadržaj neke lokacije nakon postavljanja njene adrese.Po ovom kriterijumu razlikujemo dve vrste memorija: sekvencijalne memorije i memorije sa direktnim pristupom.

Kod sekvencijalnih memorija vreme pristupa zavisi od lokacije gde je podatak memorisan. Glavni predstavnici ovih memorija su: magnetne trake, magnetni mehurići i memorije sa električnim nabojem (CCD), i uglavnom se koriste za realizaciju sekundarnih memorija.

Vreme pristupa kod memorija sa direktnim pristupom je uvek isto, i ne zavisi od lokacije podatka. U ovu grupu memorija spadaju **RAM** (**Random Access Memory**) i **ROM** (**Read Only Memory**) memorije, i danas su to po pravilu poluprovodničke memorije. RAM memorije gube svoj sadržaj nakon isključenja napajanja, ali se u njih podaci mogu i upisivati i iz njih čitati. U ROM memorije se ne mogu ponovo upisivati podaci (već se samo čitaju), ali pri isključewu napajanja ne gube svoj sadržaj. Negde između ove dve grupe, nalaze se razne vrste diskova, koji se u literaturi smatraju, ili memorijama sa direktnim pristupom, ili memorijama sa poludirektnim pristupom. Kod nekih tipova memorije, prilikom čitanja, podatak se izbriše iz memorije, pa ga je potrebno iznova zapisati. Ovaj proces se naziva memorijski ciklus (**memory cycle**), a vreme potrebno za tu operaciju zove se vremenski ciklus (**cycle time**). Ovakve memorije se nazivaju destruktivne memorije.

Takođe postoje dve vrste RAM memorija. Statičke poluprovodničke RAM memorije, koje se prave pomoću flip-flopova, koje zadržavaju svoj sadržaj i posle čitanja (nedestruktivne), sve do ponovnog zapisa, ili isključenja napajanja. Za razliku od njih, dinamičke RAM memorije (**DRAM, Dynamic RAM**) se realizuju kao kapacitivnost **MOS** tranzistora, i kao i svi kondenzatori i ovi vremenom gube naboj, pa se povremeno (na primer, svake mili sekunde) moraju osvežavati, tj. ponovo se upisuju stari sadržaji. Dinamičke memorije su destruktivne, pa se i posle čitanja, mora vršiti ponovni upis pročitanog podatka. Danas dinamičke RAM memorije imaju ogroman kapacitet i brzinu. Naime, već postoje DRAM memorije sa 600 **MBps** (često mega bajta u sekundi), a očekuju se **nDRAM** memorije sa 1,6 **GBps** (gigabajta u sekundi). Zahvaljujući razvoju novih tehnologija RAM memorije više nisu tako skupe (3-10 **$**/**MB**), pa današnji računari koriste operativne memorije vrlo velikog kapaciteta. Danas čak i mikroračunari imaju preko 500 **MB** RAM-a, a kod **mainframe** računara ona se kreće i preko 100 **GB**.



Slika 3.44. Osnovni sastavni delovi memorije

Na slici 3.44. a) prikazana je uprošćena blok šema memorije. Da bi bilo moguće čitanje i upis podataka, u memoriji moraju postojati još neki sklopovi, koji sa samom memorijom čine jedinstvenu funkcionalnu celinu. Jedan od njih je registar u koji se zapisuje adresa lokacije kojoj se pristupa. On se obično naziva memorijski adresni registar (**MAR, memory adress register**).

Drugi registar u koji se privremeno smešta podatak koji se upisuje, ili je pročitan iz lokacije koja je adresirana, je prihvatni registar podataka tzv. memorijski međuregistar (**MBR, memory buffer register**).

Treći sklop je dekoder adrese. Ovaj sklop, na bazi binarnog koda adrese memorijske lokacije, vrši selekciju te memorijske lokacije. To je dekoder tipa jedan od 2^n, gde je **n** broj botova u adresi. Na slici 3.44. b) je prikazan dekoder jedan od 2^3, kojim se na bazi trobitnog ulaza izabira jedan od 8 izlaza. Četvrti sastavni deo svake operativne memorije je upravljački memorijski sklop, kojim se bira vrsta pristupa (upis-čitanje). Ovaj sklop nije prikazan na slici 3.44. jer njegova veza sa ostalim delovima nije tako očigledna. Sklop utiče na smer prenosa podataka između memorijskog elementa i prihvatnog registra podataka.

Kod memorije velikog kapaciteta dekoder, jedan od 2^n, može biti isuviše složen, pa su moguće i drukčije organizovane memorije. Pre razmatranja raznih tipova organizacije RAM i ROM memorija, još jednom moramo istaći da se podaci u računaru pamte u obliku niza binarnih cifara **0** i **1**, odnosno kao binarni brojevi. Memorija služi samo za pamćenje podataka i ovi se mogu čitati i eventualno upisivati, tj. u memoriji se nikada ne vrši nikakva obrada. Obrada se vrši samo u centralnom procesoru.

Programi, tj. nizovi instrukcija imaju isti oblik, kao i podaci, kada se nalaze u računaru. Naime, i instrukcije se tako|e kodiraju pomoću cifara binarnog brojnog sistema, tako da su u računaru takođe predstavljene kao binarni brojevi. **John von Neumann** je došao na ideju da se podaci i programi smeštaju u istu operativnu memoriju. Ovo je tzv. Nojmanova konfiguracija računara. Kada su instrukcije upisane u operativnu memoriju računara, one se mogu tretirati na isti način kao podaci. Na ovaj način je stvorena mogućnost da sam računar izvrši prevođenje programa, napisanog u programskom jeziku koji računar ne razume (na primer, u **FORTRAN**-u), u program ekvivalentan po funkciji, ali napisan u jeziku koji računar razume (npr. u mašinskom jeziku). No, tretiranje instrukcija kao podataka je i opasno, pa operativni sistem računara mora strogo voditi računa o tome gde su u memoriji zapisani podaci, a gde instrukcije, jer centralni procesor vrši obradu instrukcija sasvim drukčije od obrade podataka. To je deo operativnog sistema koji se zove mehanizam zaštite memorije.

RAZLIČITI TIPOVI ORGANIZACIJA MEMORIJE

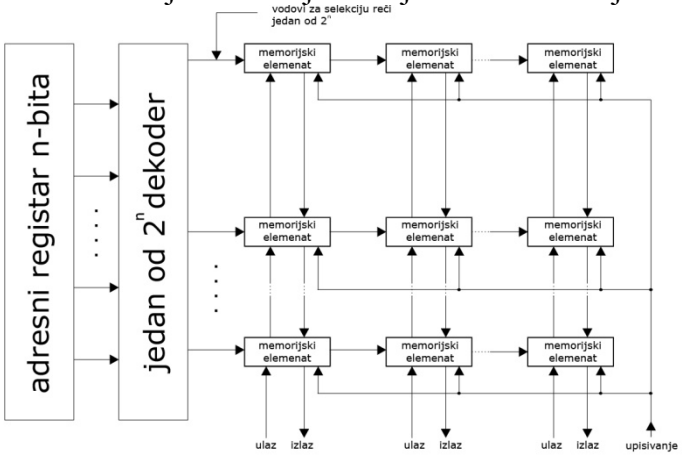
Pamćenje podataka u memorijama uvek se svodi na pamćenje nula i jedinice. To se može ostvariti na različite načine. Osnovni element memorije, nazovimo memorijska ćelija, beleži i pamti jednu nulu ili jedinicu, odnosno jedan binarni bit. Memorije moraju biti u stanju da beleže veliki broj bitova povezanih u bajtove i reči. Da bi se memorija mogla upotrebljavati, mora postojati mogućnost da se podaci upisuju u memoriju i da se čitaju iz nje. To znači da elementarne ćelije za pamćenje pojedinačnih bitova treba na odgovarajući način povezati tako da mogu ispunjavati zadatke koji se od njih traže. Elementarne memorijske ćelije mogu biti povezane na različite načine ili, kako se drukčije kaže, memorije mogu biti organizovane na različite načine. Tako postoje dvodimenzionalne i

trodimenzionalne memorije, i memorije organizovane kao stek. Osim takvih paralelnih organizacija memorija kod kojih se odjednom može upisivati ili čitati cela jedna reč ili bajt i to sa slučajnim pristupom, postoje i serijske memorije kod kojih se podaci kreću jedan iza drugog, pa se bitovima može pristupiti samo određenim redosledom, a ne slučajno. Razmotrićemo karakteristične tipove organizacija memorija, i to prvo dvodimanzionalne, a zatim trodimanzionalne memorije, stek-memorije i serijske pomeračke memorije.

Dvodimenzionalne memorije

Kako izgleda organizacija dvodimenzionalnih memorija prikazano je na slici 3.45. Memorijski elementi (ćelije) pore|ani su jedan do drugog tako da niz takvih horizontalno pore|anih elemenata (jedan red) može da znači jednu reč. Memorijski elementi svrstani u vertikalne kolone čine bitove iste težine u različitim rečima. Tako prva kolona sleva može zapisivati podatke 2^0, druga kolona 2^1, treća 2^2 i poslednja 2^{m-1}, gde je **m** dužina upotrebljene reči.

Adresni registar od **n**-bitova može imati 2^n različitih stanja. Svako takvo stanje dekoder pretvara u stanje **1** samo na jednoj od 2^n izlaznih linija koje služe za selekciju adresirane memorijske lokacije sa koje se čita ili u koju se upisuje.



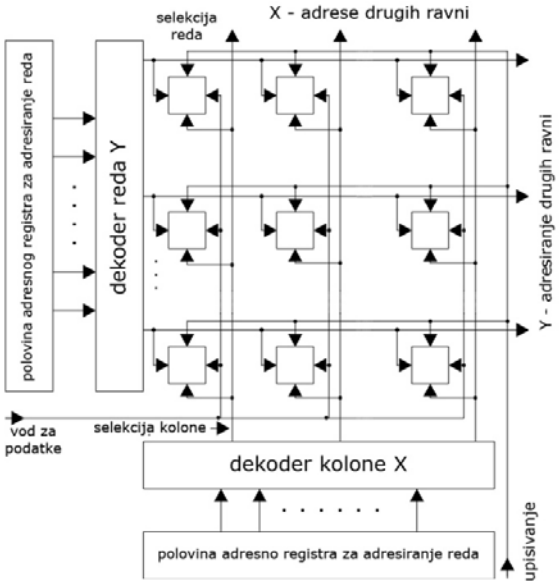
Slika 3.45 Organizacija dvodimenzionalne memorije

U jednom trenutku vremena, linijama za selekciju može se aktivirati samo jedan horizontalni red memorijskih elemenata koji čine jednu reč. Ostali horizontalni redovi nisu aktivni i ne učestvuju u operaciji čitanja ili upisivanja. Kada se generiše odgovarajući signal na liniji za upisivanje, podatak doveden na ulazne linije upisuju se u odgovarajuće ćelije selektovane reči. Pri čitanju stanja odgovarajućih memorijskih elemenata selektovana reč se pojavljuju na izlaznim linijama. Signal za čitanje obično je komplement signala za upisivanje, tj. stanje 1 na vodu za upisivanje omogućuje upis podataka u memorijski element, a pri stanju 0 na toj liniji pojavljuju se podaci na izlaznim linijama. Jednu dimenziju takve memorije

čine adrese lokacija (na slici 3.45 vertikalno), a drugu dimenziju dužina reči (na slici 3.45 horizontalno). Zbog toga je nazivamo dvodimenzionalnom memorijom (a neki to nazivaju i jednodimenzionalnim, linearnim adresiranjem). Međutim, treba istaći da je takva organizacija prilično nepraktična, jer treba imati onoliko selekcionih linija koliko memorija ima reči. Koliko je to linija, dovoljno je reći da za LSI-memorije (memorije visokog stepena integracije, **Large Scale Integration**) od 64**K** bita uz 8-bitne reči, treba više od 8000 linija. Treba reći još i to da se kapaciteti LSI-memorija obično izražavaju u broju bitova, a ne bajtova. Tako LSI-memorija od 8**K** bita može prikazati 1**K** bajt, a ona od 64**K** bita predstavlja 8**K** bajta.

Trodimenzionalne memorije

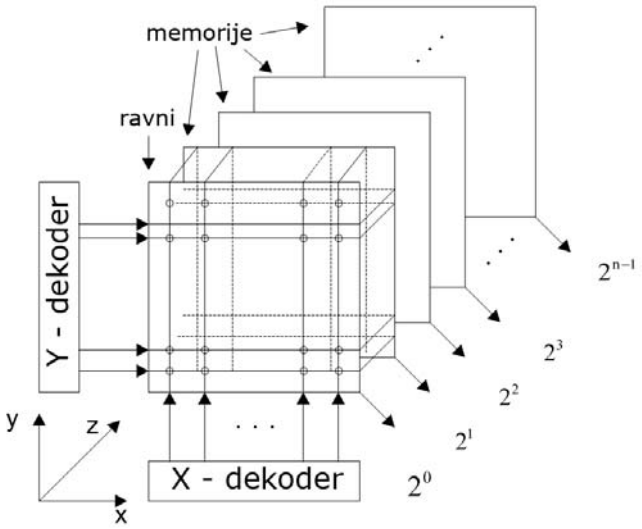
Dvodimenzionalna memorija mora imati veliki broj linija za selekciju reči, odnosno onoliko linija koliko ima reči. To znači da uz 256_{10}, 1024_{10} ili 4096_{10} upotrebqenih reči, mora imati isto toliko selekcionih linija. Toliki broj selekcionih linija predstavlja problem, pa se nastoji da se njihov broj smanji upotrebom trodimenzionalne memorije. To se postiže tako da se selekcija pojedine memorijske ćelije ne izvrši do kraja u posebnom spoljnjem dekoderu, već se deo selekcije izvodi i u samoj memorijskoj ćeliji. Pod spoljnom selekcijom podrazumeva se sve ono što se nalazi izvan memorijskih ćelija, iako može biti na istom čipu.



Slika 3.46. Prikaz jedne ravni trodimenzionalne memorije

Najjednostavniji način takvog adresiranja pojedinih ćelija memorije jeste kad se adresni registar i dekoder podeli na dva dela, obično na dve polovine, jedna za horizontalno adresiranje, a druga za vertikalno. Svaka polovina odabere po jednu adresnu liniju u skladu sa podatkom koji je zapisan na odgovarajućoj horizontalnoj,

odnosno vertikalnoj polovini adresnog registra. Adresira se samo ona memorijska ćelija koja se nalazi na preseku odabrane horizontalne i vertikalne linije. To se izvodi tako da se u svaku memorijsku ćeliju doda **I** kolo koje odabere memorijsku ćeliju samo onda kad su odabrane i horizontalne i vertikalne linije koje dolaze iz spoljnjeg dekodera. Kako se to može izvesti pokazano je za jednu ravan trodimenzionalne memorije na slici 3.47.



Slika 3.47. Organizacija trodimenzionalne memorije

Polovina adrese namenjena je adresiranju reda (**Y=n/2**), a polovina adresiranju kolone (**X=n/2**). Odabere se ćelija koja se nalazi na preseku odabranih linija **X** i **Y**. Samo se u tu memorijsku ćeliju može upisati podatak kad se on pri postojanju signala za upisivanje nalazi na linija za podatke. Isto se tako iz te ćelije može pročitati podatak, i on se takođe pojavljuje na liniji za podatke. Razmotrimo sada koliko dekoderskih linija treba za dvodimenzionalnu, odnosno trodimenzionalnu memoriju. Kod dvodimenzionalne memorije, za **n** adresnih bitova na adresnom registru potrebno je 2^n izlaznih adresnih linija iz jedinstvenog dekodera. Kod trodimenzionalne svaki dekoder ima $2^{n/2}$ linija, a oba zajedno imaju $2 \cdot 2^{n/2} = 2^{n/2+1}$, tj. približno dvostruko manje.

Memoriju koja je ovde nazvana trodimenzionalnom neki nazivaju i memorijom sa dvodimenzionalnim adresiranjem, odnosno memorijom sa koincidentnim adresiranjem. Kod trodimenzionalne memorije adrese čine dve dimenzije, a dužina reči treću dimenziju. Da bi se organizovala takva trodimenzionalna memorija treba onoliko ravni memorije (kao što je ona na slici 3.46), koliko je bitova duga reč koja se pamti.

U ravni trodimenzionalne memorije prikazanoj na slici 3.46, adresira se samo jedna memorijska ćelija. Ona predstavlja jedan bit (npr. nulti bit) adresirane reči, a sve

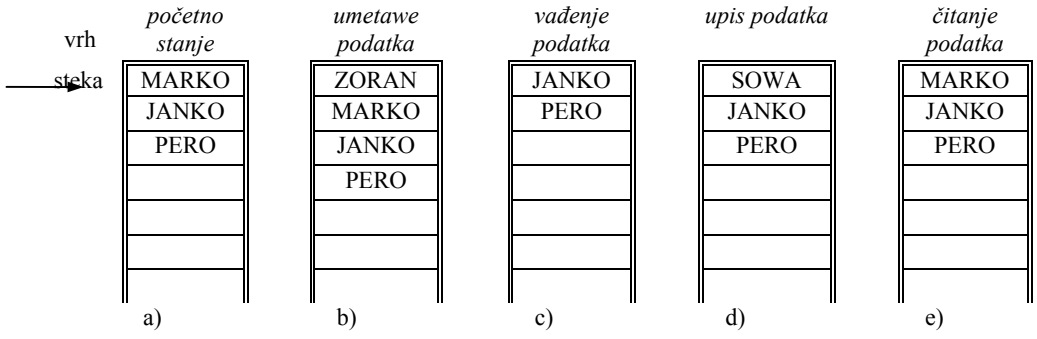
ostale memorijske ćelije u toj ravni predstavqaju po jedan bit različitih reči. To znači da pri takvoj organizaciji memorije u jednoj ravni ima onoliko jednobitnih reči koliko ima bitova u toj ravni. Prema tome, da bi se dobilo više bitova tj. memorijska reč, treba upotrebqavati onoliko ravni memorije koliko su duge reči koje treba pamtiti. Organizacija takve memorije prikazana je na slici 3.47.

Dužina reči predstavlja treću dimenziju memorije, a bitovi jedne memorijske reči nalaze se jedan ispod drugog u raznim ravnima memorije. Pri tome dekodirane adresne linije iz prve ravni prelaze u drugu, zatim treću, itd., pa je, bez obzira na broj ravni, potreban uvek isti broj linija za adresiranje. Podaci se čitaju i upisuju na linijama za podatke u svakoj ravni. Tako se u nultoj ravni upisuje 2^0, u prvoj 2^1, u drugoj 2^2, trećoj 2^3, itd. Na slici 3.47, zbog jednostavnosti i preglednosti, prikazane su samo neke važnije linije, a ostale su izostavljene. Malim kružićem označena je osnovna memorijska ćelija, koja je osim pamćenja binarnog podatka obavlja i dodatnu logičku I-operaciju, jer je adresirana samo onda kada postoji logička jedinica i po liniji (X) i po liniji (Y) koje prolaze kroz kružić.

Memorije organizovane u stek

Dvodimenzionalna i trodimenzionalna organizacija memorija omogućuju slučajan pristup do svakog podatka. To znači da se bilo koji podatak može čitati ili upisivati nezavisno od tog koji se podatak čitao ili upisivao pre toga. Zbog toga se takve memorije i nazivaju memorijima sa slučajnim pristupom, bez obzira na to da li se radi o memorijama **RAM** ili **ROM**.No, postoje i takve organizacije memorija gda se podaci mogu upisivati ili čitati samo po nekom redu, pa to nisu memorije sa slučajnim pristupom. Jedna od takvih je i stog ili stek memorija (**stack**).
Memorija organizovana u stek sastoji se od niza registara, od kojih se svaki sastoji od odre|enog broja memorijskih elemenata (ćelija). Takav niz registara, složenih jedan na drugi kao stog, za razliku od memorija sa slučajnim pristupom, ima dostupan samo onaj registar koji je na vrhu stoga.

Podaci se umeću na vrh stoga, a sa vrha se i vade. Na slici 3.48 a) prikazano je jedno moguće stanje steka.



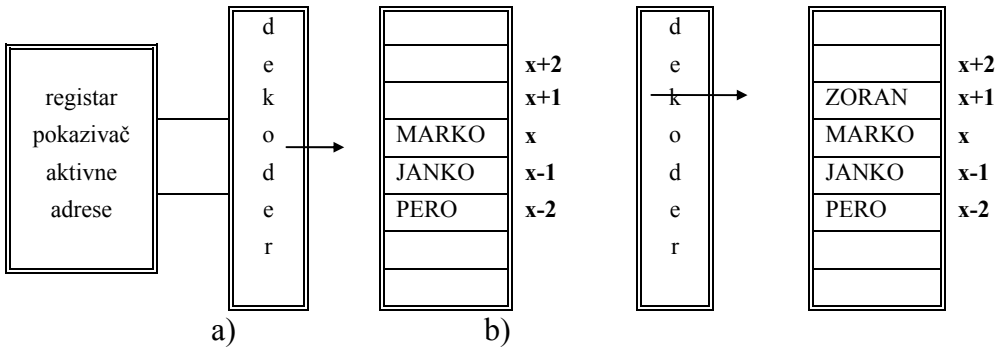
Slika 3.48. Stek memorija i operacije sa stekom

Funkcionisanje steka može se uporediti sa situacijom kad se na radnom stolu na gomilu slažu papiri koje treba rešiti (obraditi). Novi papiri uvek se stavljaju na vrh gomile, a onaj ko rešava te papire uvek uzima papir samo sa vrha gomile. Kad uzme jedan papir i reši ga, premesti ga na drugo mesto, ili baci. Na vrhu steka tada se nađe papir koji je bio ispod prethodnog, itd. Takav način rada zove se "poslednji ušao prvi izlazi", ili **LIFO**, (**Last In First Out**). Ako se umetne nova reč, onda se sve reči koje su pre bile u steku pomaknu za jedan korak naniže, a nova uđe na vrh.

Na slici 3.48. b) vidi se stanje nakon umetanja nove reči ZORAN. Ako se izvadi jedna reč (od početnog stanja prikazanog na slici 3.48 a)), dobije se situacija kao na slici 3.48. c). Pri upisivanju reči (slika 3.48. d)) briše se pređašnja reč MARKO i umesto nje se može upisati, recimo., reč SONJA. Pri upisivanju se podaci u ostalim registrima ne menjaju, tj. reč JANKO i PERO ostaju na svojim pređašnjim mestima. Pri čitanju se pročita reč s vrha steka, dakle, reč MARKO, slika 3.48. e). Čitanje i upisivanje podatka isto je kao u običnoj RAM memoriji računara, dok su operacije umetanja i vađenja podatka svojstvene samo steku i ne postoje kod obične memorije. Obzirom da su podaci zapisani u binarnom obliku i da je u pitanju pomeranje podataka, najjednostavniji način realizacije steka, jeste upotreba pomeračkih registara (**shift register**).

Bit najmanje težine u svim registrima zajedno predstavlja jedan pomerački registar, a sledeći bit je drugi pomerački registar, itd. Pri realizaciji steka na takav način podaci se pomeraju po registrima. Takav način realizacije u načelu je sličan steku prikazanom na slikama 3.48.

Postoji i druga mogućnost realizacije steka, tako da se ne pomera podatak, nego se menja adresa koja odgovara vrhu steka. Takav način realizacije steka prikazan je na slici 3.49.



Slika 3.49 Realizacija steka pomeranjem adrese (softverski stek)

Pri toj realizacije aktivna adresa steka, tj. adresa koja odgovara prethodnom vrhu steka, pomera se pri umetanju, odnosno vađenju podatka.
Ona u pojmovnom smislu i dalje čini vrh steka, mada je bolje da se zove aktivna adresa, jer vrh steka u tom slučaju može biti i na donjoj strani. Ako, dakle, sadašnja aktivna adresa pokazuje lokaciju **x** u kojoj je zapisana reč MARKO, a na nižim

položajima su reči JANKO i PERO, kao na slici 3.49 a), onda će se pri stavljanju nove reči ZORAN (slika 3.49 b)), aktivna adresa najpre pomeriti na položaj **x+1**, pa će se zatim u nju zapisati umetnuta reč. Ako se želi izvaditi reč MARKO, onda se ona pročita i izbriše, a zatim se aktivna adresa pomeri na položaj **x-1**.

MEMORIJA ROM

Memorija **ROM** ili **ROS** (**Read-Only Store**) ispunjava dva zahteva:

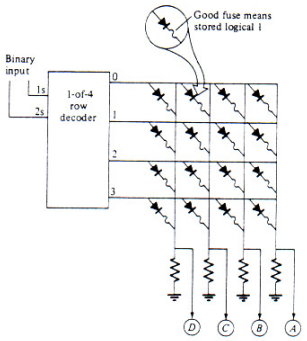
- neizbrisivost (**non-volatibility**),
- nedestruktibilnost-neuništivost sadržaja (**non-destructive readout**).

Neuništivost znači da se ROM pojavljuje kao memorijsko polje čiji je sadržaj, jednom upisan, stalan i ne može se promeniti pod uticajem procesora (operacijom upisivanja). Zbog svojih karakteristika ROM se obično upotrebljava za memorisanje stalnih programa.

U grupu ispisnih memorija, tj. memorija koje se mogu samo čitati, spadaju:

- ROM - isključivo ispisna memorija,
- PROM - programabilna ispisna memorija,
- EPROM - promenqiva programabilna ispisna memorija,
- EEPROM (EAROM)-električno programabilna ispisna memorija.

a) U **ROM** je upisan određeni sadržaj već za vreme izrade memorijskog čipa. Bit-uzorke koji odgovaraju željenom programu korisnik dostavlja proizvođaču u standardnom obliku (npr. MOTOROLA zahteva papirnu traku kao rezultat upotrebe **MC6800** programskog paketa ili heksadekadni kod na bušenoj kartici **IBM**). Proizvođač izrađuje odgovarajuću masku (prema priloženom programu korisnika) i kao zadnji korak u proizvodnji



Slika 3.50. Diodni PROM

čipa ROM vrši metalizaciju, odnosno uspostavlja veze između redova i kolona.

Minimalni broj **ROM**-ova koji se na taj način mogu proizvesti određuje proizvođač, a kreće se oko broja 1000. Vreme pristupa za **ROM** u tehnologiji **MOS** je 500 do 850 ns.

b) **PROM** (**user-programmable read only memory**) je tip ispisne memorije koju može isprogramirati sam korisnik uz pomoć uređaja za programirawe PROM-ova. U memorije PROM spadaju dva tipa programa-bilnih ROM-ova: bipolarno polje dioda i bipolarna tranzistorska konfiguracija. Oba ova tipa primenjuju teniku "pregorljivih veza" (**fusible links**), tj. metalizovanih veza između baze i emitera ili u PN spoju dioda, slika 3.50. Za vreme programiranja uređaj generiše niz impulsa kojim će se izabrane "pregorljive veze" rastopiti i time prekinuti vezu u matrici između kolone i reda. PROM ima kratko vreme pristupa - manje od 100 ns i upotrebljava se kod većine mikroračunara izrađenih u malim serijama za smeštanje programa. Nedostatak PROM-a sličan je nedostatku ROM-a - kada je jednom isprogramiran, ne može se menjati njegov sadržaj.

c) **EPROM** je memorija koju može programirati korisnik uz pomoć (E)PROM-programera, ali se njen sadržaj može izbrisati i zatim ponovo isprogramirati. EPROM se briše osvetljavanjem čipa ultra-ljubičastim zracima u trajanju od pet do deset minuta. Sadržaj svih memorijskih lokacija se briše. Nakon brisanja, EPROM može biti ponovo isprogramiran. Cena EPROM-a je relativno visoka. Vreme pristupa je između 150 do 1200 nanosekundi. Za izradu memorija EPROM upotrebljava se tehnologija MOS. Jedan tip višestruko programibilnog PROM-a je RPROM, koji se briše električki.

d) **EEPROM** ili EAROM (**electrically alterable ROM**). Kao alternativno rešenje problema izbrisivosti naveden je i EEPROM, obično svrstan u razred memorija koje se uglavnom čitaju (**read-mostly memory, RMM**); međutim, EEPROM omogućuje i upisivanja. Operacija upisivanja traži vreme reda veličine milisekunde, dok su operacije čitanja reda mikrosekunde. Očito je da se takve memorije ne mogu upotrebljavati kao klasične memorije sa direktnim pristupom. Memorija EAROM se upotrebljava tamo gde je nužno smeštanje malog broja podataka ili parametara. Ima mali kapacitet, nekoliko desetina **KB** (kilo bajta).

MEMORIJE SA VIŠE MODULA I PREKLAPANJE

Glavna memorija računara najčešće predstavlja kolekciju većeg broja fizički odvojenih modula. Svaki od njih ima svoj memorijski adresni registar (**MAR**) i memorijski bafer registar (**MBR**) tako da je moguće postići da dva i više modula istovremeno izvršava operaciju upisa (**write**) ili čitanja (**read**). Na taj način se može postići znatno skraćenje srednjeg vremena pristupa memorijskim lokacijama.. Da bi se to postiglo potrebno je imati dodatne upravljačke uređaje što poskupčljuje

hardver, pa se ovi sistemi obično koriste u velikim računarima. Moguća su dva pristupa:

- prvi, kada se uzastopne lokacije (reči) nalaze u istom memorijskom modulu. Ovaj metod omogućava, recimo, da jednom modulu pristupa centralni procesor radi upisa ili čitanja podatka (ili instrukcije), dok je drugi modul pod kontrolom, recimo, **DMA** kontrolera i izvrćava prenos podataka u ili iz ulazno-izlaznih jedinica. Za selekciju memorijskog modula koristi se **k** bitova najveće težine (u efektivnoj memorijskoj adresi).
- drugi, kada se uzastopne lokacije (reči) nalaze u uzastopnim (raznim) memorijskim modulima. Obaj metod se naziva preplitanje memorija (**memory interleaving**), i omogućava centralnom procesoru (ili **DMA** kontroleru) da istovremeno pristupi većem broju memorijskih lokacija. Da bi sistem bio vrlo efikasan procesor mora imati mogućnost da "predvidi" koji će mu podatak biti uskoro potreban i da unapred pošalje zahtev memoriji. (**prefetch**). To je slučaj kada se obrađuju nizovi podataka (u uzastopnim lokacijama). Sem toga, potrebno je da broj memorijskih modula bude $\mathbf{r}=2^{k}$. Za selekciju memorijskog modula u kome je željeni podatak koristi se **k** bitova najmanje težine. Ovo može teorijski skratiti vreme pribavljanja podataka za 1/**r** puta (gde je **r** broj memorijskih modula). Ove pogodnosti se gube pri izvršavanju naredbi uslovnog ili bezuslovnog skoka, jer se tada upravljanje prenosi na neku drugu instrukciju kojoj su najčešće potrebni podaci iz nekog drugog dela memorije.

3.10. ZAKLJUČAK

U ovom poglavlju proučene su logičke osnove računara kao i elementarna logička kola koja se koriste u realizaciji računara. Kombinovanjem osnovnih logičkih kola mogu se realizovati različite funkcionalne logičke mreže kao što su: koderi, dekoderi, multiplekseri, demultiplekseri, polisabirači i sabirači.Elementarna logička kola se takođe koriste u realizaciji osnovnih memorijskih sklopova za memorisanje jednog bita, kao što su razne vrste flip-flopova. Za memorisanje podataka dužine reči, koriste se registri koji imaju onoliko flip-flopova koliko reč ima botova. Za upis, i čitanje podataka iz registara koriste se odgovarajući upravljački signali koje generišu upravljačke logičke mreže. Kombinovanjem više flip-flopova prave se i druge digitalne mreže kao što su brojači i pomerački registri.
Za memorisawe većeg broja binarnih podataka koriste se hardverski sklopovi koji se zovu memorije, za čiju realizaciju se mogu koristiti različiti mediji i koje se mogu organizovati na razne načine.ALU i registri unutar centralne procesorske jedinice, povezani su međusobom posredstvom specijalnih linija koje dopuštaju istovremeno prisustvo većeg broja digitalnih kola, i koja se zove interna magistrala. Veza CPU sa memorisjkim jedinicama i ulazno-izlaznim uređajima, tako|e je ostvarena posredstvom magistrala, za čije funkcionisanje je takođe potrebno imati određene upravljačke signale.

3.11. PITANJA

1. Koji su osnovni sastavni delovi centralnog procesora?
2. Šta je logička osnova rada računara?
3. Navesti aksiome Bulove algebre.
4. Na koju osobinu logičkih funkcija ukazuju osnovne teoreme Bulove algebre?
5. Koje su osnovne logičke operacije?
6. Šta je to tablica istinitosti?
7. Nacrtati šemu kojom se realizuju **I**, **ILI** i **NE** operacija pomoću **NILI** kola.
8. Nacrtati šemu kojom se realizuju **I**, **ILI** i **NE** operacija pomoću **NI** kola.
9. Objasniti minimizaciju logičkih funkcija pomoću Karnoovih mapa.
10. Da li stanje na izlazu R/S flip-flopa zavisi samo od trenutne vrednosti signala na S i R ulazu?
11. U čemu je razlika u načinu rada D i R/S flip-flopa?
12. Šta na svom izlazu daje binarni polusabirač?
13. U čemu je razlika između polusabirača i potpunog binarnog sabirača?
14. Koji sklopovi ulaze u sastav aritmetičko-logičke jedinice?
15. Koje su osnovne vrste registara i koja im je namena?
16. Koja je namena osnovnih registara u sastavu CPU?
17. Šta je prednost povezivanja registara preko magistrala?
18. Kako sve mogu biti organizovani procesori?
19. U čemu je razlika između adrese lokacije i njenog sadržaja?
20. Zašto se u programima obrade koriste simboličke adrese tj. simbolička imena podataka, a ne sami podaci?
21. Opisati hijerarhiju uređaja za memorisanje.
22. Šta je sekvencijalni a šta direktni pristup memorijskim lokacijama?
23. Koji su osnovni sastavni delovi glavne memorije računara?
24. Opisati razlike u organizaciji dvodimenzionalnih i trodimenzionalnih memorija.
25. Šta je stek memorija i kako se može realizovati?
26. Šta su ROM memorije, i koje vrste postoje?

3.12. KLJUČNE REČI

- **ALU**, aritmetičko-logička jedinica (**Arithmetic and Logic Unit**)
- Bulova algebra
- **D** flip-flop (**data flip-flop**)
- dinamička RAM memorija (**Dinamic RAM, DRAM**)
- dekoder (**decoder**)
- destruktivna RAM memorija (**DRO, destructive readout**)
- digitalno kolo (**digital circuit**)
- direktni, slučajni pristup (**direct access, random access**)
- dvodimenzionalna memorija
- **ekskluzivno**, **isključivo ILI**, sabiranje po modulu dva (**XOR**)
- fizička adresa, efektivna adresa (**efective address**)
- flip-flop (**flip-flop**)
- **I** operacija, konjunkcija, logičko množenje (**AND**)
- **ILI** operacija, disjunkcija, logičko sabiranje (**OR**)
- integrisano kolo (**IC integrated circuit**)
- koder (**coder**)
- koincidentno adresiranje
- linearno adresiranje
- logička operacija
- logički podatak (**logical data**)
- **LSI,** tehnologija visokog stepena integracije (**large-scale integration, LSI**)
- magistrala, sabirnica (**bus**)
- memorija (**memory, storage**)
- memorijska adresa (**memory address**)
- **NE** operacija, komplementiranje, invertovanje, negacija (**NOT**)
- **NI** operacija,šeferova funkcija **(NAND)**
- **NILI** operacija, Pirsova funkcija (**NOR**)
- polusabirač (**half adder**)
- pomerački registar (**shift register, shifter**)
- preplitanje memorija (**memory interleaving**)
- primarna, glavna, operativna memorija (**primary storage, operating memory**)
- proširena memorija (**extended memory**)
- **R/S** flip-flop (**set-reset flip-flop**)
- RAM memorija (**Random Access Memory**)

- registar (**register**)
- relativna adresa (**relative address**)
- ROM memorija (**read only memory**)
- **ROS** memorija (**Read Only Store**)
- sabirač, potpuni sabirač (**full adder**)
- sekundarna memorija, masovna memorija (**secondary storage, mass memory**)
- sekvencijalni pristup (**sequential access**)
- simboličko adresiranje (**simbolic addressing**)
- skrivena memorija, skriveni disk (**cache memory, cache disk**)
- statička RAM memorija (**Static RAM**)
- trodimenzionalna memorija
- zastavica (**flag)**

4. ARHITEKTURA RAČUNARA

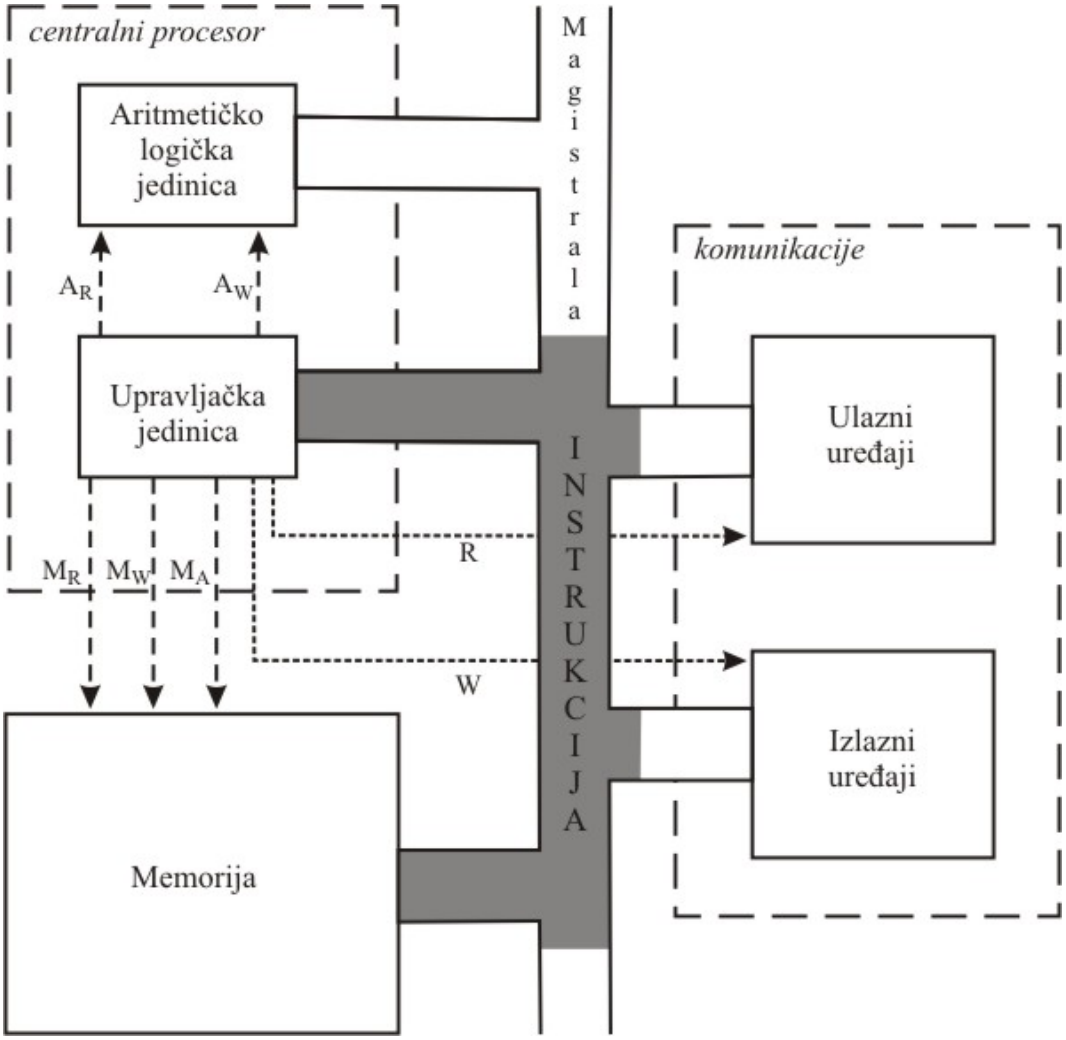
Međusobno povezivanje i koordinacija rada digitalnih kola, magistrala, registara, aritmetičko-logičke jedinice i drugih sklopova u sastavu računara, ostvareni su pomoću programa na mikroprogramskom nivou i na nivou mašinskog jezika, kao što je to prikazano na hijerarhijskom modelu slika 1.5. Ova dva nivoa sadrže osnovne programe koji su sposobni da upravljaju radom digitalnih mreža tako da ove mreže obave određeni koristan rad. Programi na mašinskom jeziku i mikroprogrami, koji koordiniraju rad digitalnih logičkih kola, komponovani su kao nizovi instrukcija od kojih je svaka pojedinačno prikazana kao neki binarni broj. Tokom izvršavanja neke instrukcije, njeni bitovi služe kao ulazni signali u logička kola koja generišu kontrolne i upravljačke signale, izvršavaju aritmetičke ili logičke operacije, ukazuju na lokacije u memoriji itd. Na ovom nivou postoji niz specifičnih detalja koji variraju od računara do računara i od proizvođača do proizvođača. No, generalna ideja je manje više ista. Radi lakšeg razumevanja u razmatranju ovih nivoa, krenućemo obrnutim redosledom, tj. prvo ćemo posmatrati šta se dogaća u računaru na nivou mašinskog jezika, a potom na nivou mikroprograma.

4.1. POJEDNOSTAVLJENA ARHITEKTURA RAČUNARA

Pojam arhitekture računara označava glavne sastavne delove računara i njihovu povezanost u jednu funkcionalnu celinu. Na slici 4.1 prikazane su glavne komponente arhitekture tipičnog računara koje čine: procesor, memorija i komunikacioni sistem.

Ulazne i izlazne jedinice omogućavaju povezivanje računara sa spoljašnjim svetom iz kojeg računar dobija podatke i zahteve za obradu, i kome računar šalje rezultate obrade i informacije. Ovih jedinica može biti više u jednom računarskom sistemu.Memoriju predstavljaju različiti elektronski sklopovi koji se nalaze unutar računara, a u kojima se nalaze programi i podaci koji se upravo obraćuju (tekući, aktuelni program i podaci). Ova memorija se zove operativna ili glavna memorija

računara. Dodatna memorija, koju čine razni registri, nalazi se unutar centralne procesorske jedinice.



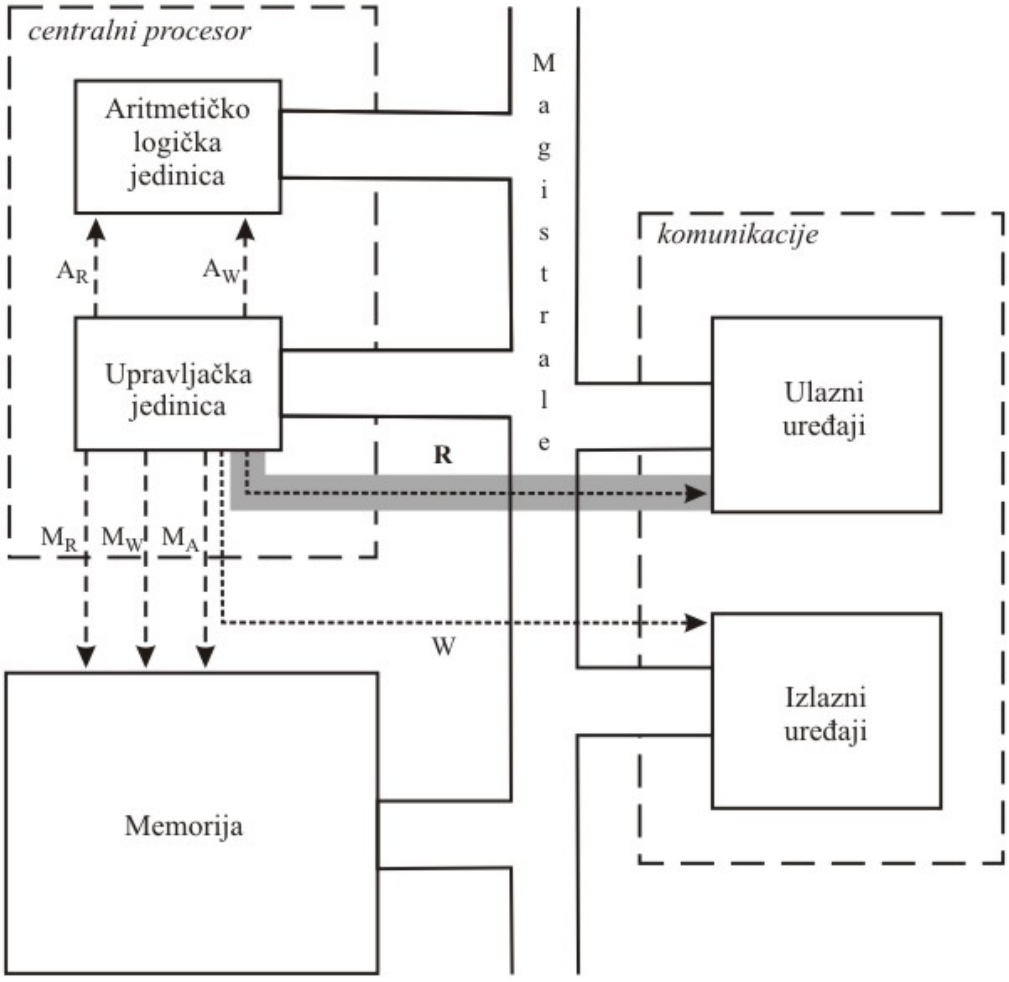
Slika 4.1. Generalisana arhitektura računara

Glavna komponenta arhitekture računara je centralna procesorska jedinica (CPU), koja se sastoji od upravljačke jedinice (**control unit**), i aritmetičko logičke jedinice (ALU). U sastav ove jedinice ulazi i određeni broj registara specijalne i opšte namene koje, za sada, nećemo pominjati, niti koristiti u početnim razmatranjima. Punim linijama na ovoj slici su predstavljene magistrale podataka i adresa, a upravljački signali su prikazani isprekidanim linijama.Centralna procesorska jedinica sadrži digitalna kola koja prihvataju i izvršavaju programske instrukcije. Izvršavanje bilo koje instrukcije ima za posledicu generisanje (od strane

upravljačke jedinice) čitavog niza upravljačkih signala. Ovi signali kontrolišu i sinhronizuju rad svih delova računara.

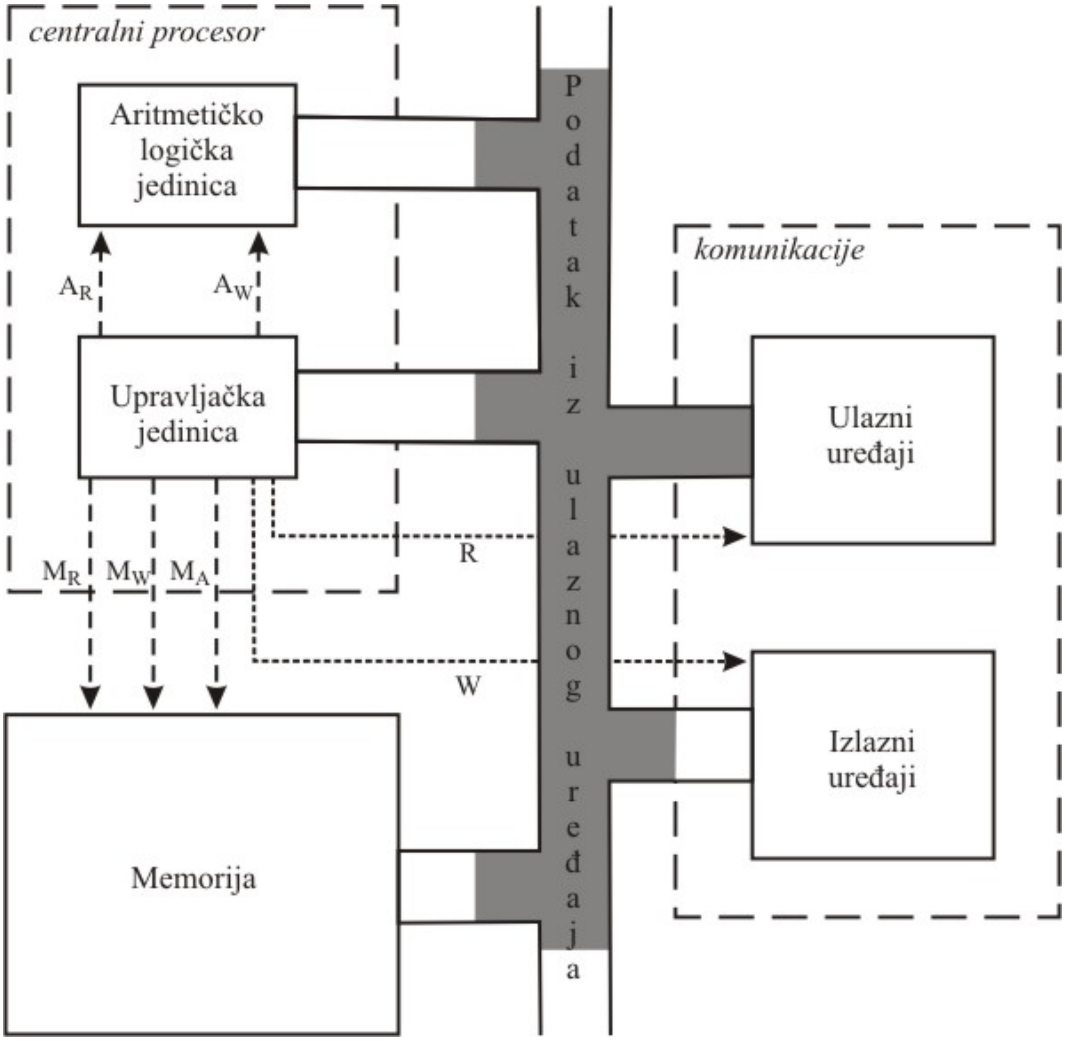
ULAZ PODATAKA

Kada centralnom procesoru treba podatak iz spoljašnjeg sveta, on ulaznoj jedinici pošalje zahtev za unos podatka, odnosno za njegovo zapisivanje na magistralu. Ovaj zahtev se inicijalizuje postavljanjem upravljačkog signala na liniju **R** (slika 4.2), i nakon toga centralni procesor čeka da ta ulazna jedinica odgovori, odnosno izvrši postavljeni zadatak.



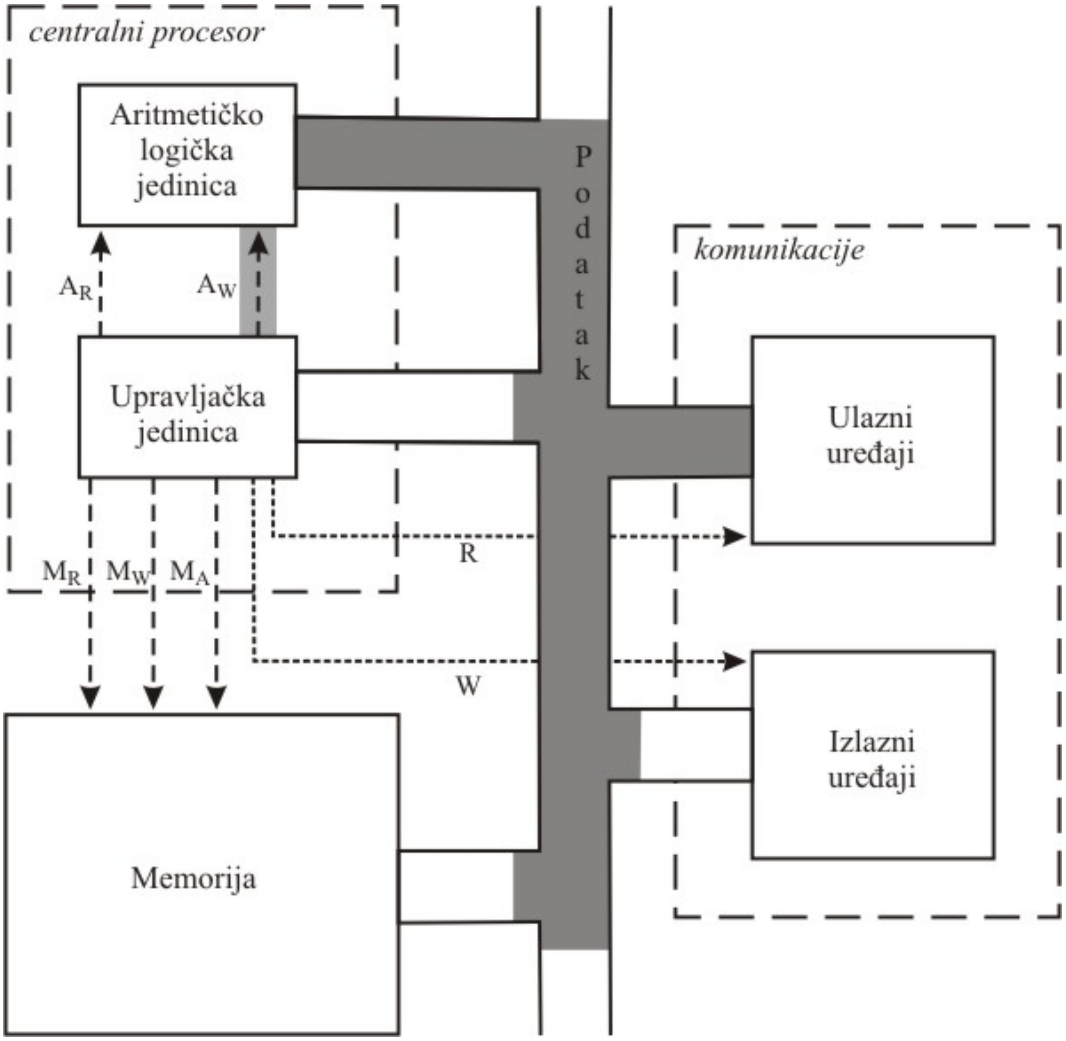
Slika 4.2. Ulazno kolo je aktivirano upravljačkim signalom **R** iz upravljačke jedinice

Pod dejstvom upravljačkog signala R, ulazna jedinica kopira potreban podatak iz nekog svog registra na magistralu podataka, slika 4.3.



Slika 4.3. Nakon što je ulazno kolo dobilo upravljački signal, ono postavlja podatak na magistralu

Kada je podatak postavljen na magistralu (vidi sliku 4.3.), on je na raspolaganju svim komponentama računara, tj. dostupan je i memoriji, i upravljačkoj jedinici, i aritmetičko-logičkoj jedinici, i bilo kojoj izlaznoj jedinici, ali samo jedna od njih može stvarno da ga preuzme. **Koja?** To zavisi od toga koji upravljački signal će generisati upravljačka jedinica (**M_W , A_W** ili **R**).Ako podatak sa magistrale preuzima aritmetičko-logička jedinica, onda će biti generisan signal **A_W** (slika 4.4), dok će memorija i izlazna jedinica ostati neaktivne. Slovo (i indeks) **W**, predstavlja skraćenicu od reči **write**, a označava operaciju upisivanja podatka. Slovo **R** (i indeks) predstavlja skraćenicu od reči **read**, što označava čitanje, dok indeks **A** označava da su u pitanju signali adrese.

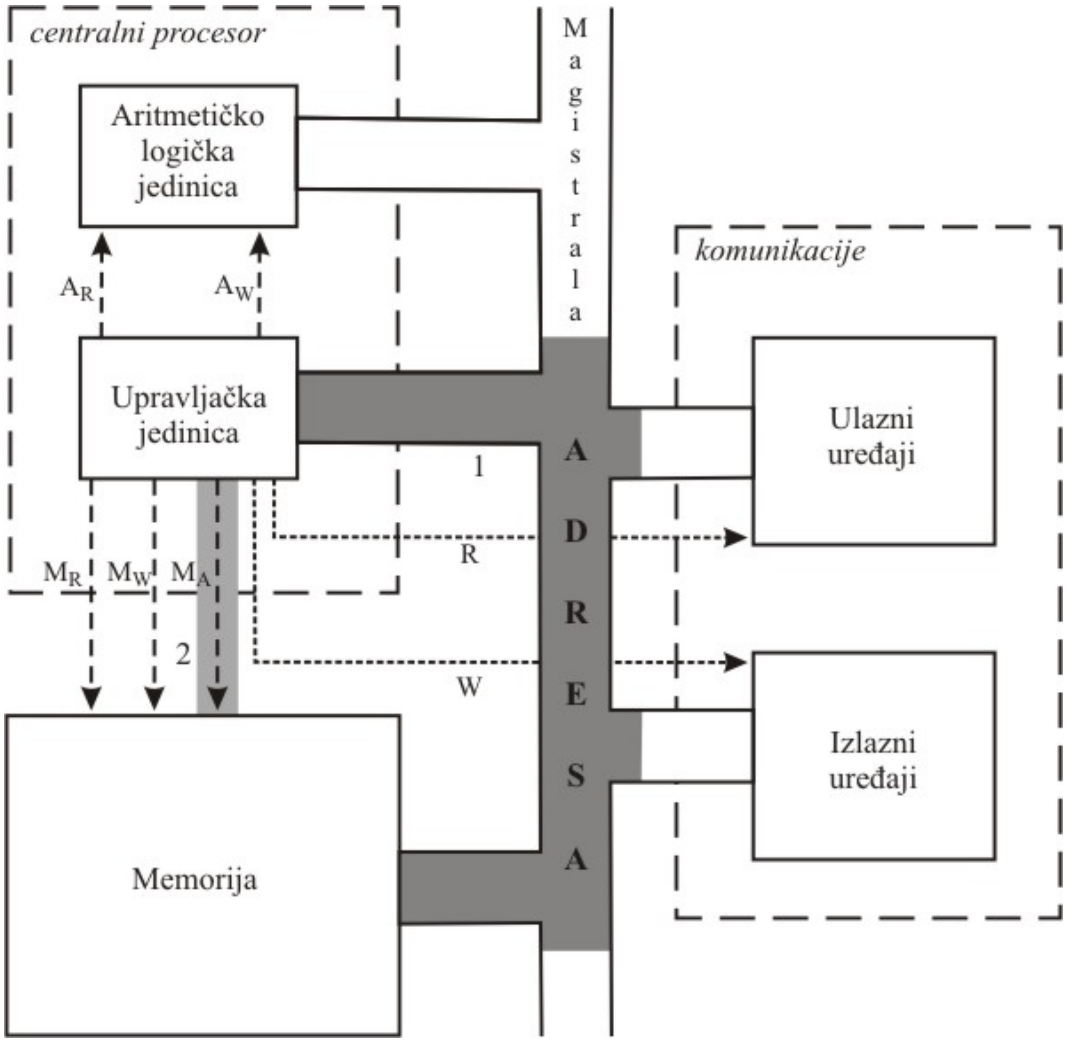


Slika 4.4. Upravljačka jedinica određuje odredište podatka, i šalje upravljački signal odgovarajućoj jedinici, u ovom slučaju je to aritmetičko-logička jedinica

PRISTUP MEMORIJI

Smeštanje podataka u memoriju je znatno složenije od smeštanja podataka u aritmetičko-logičku jedinicu. Memorija sadrži veliki broj memorijskih lokacija (nekoliko miliona), zbog čega moramo tačno odrediti koju ćemo lokaciju koristiti za smeštanje podatka.

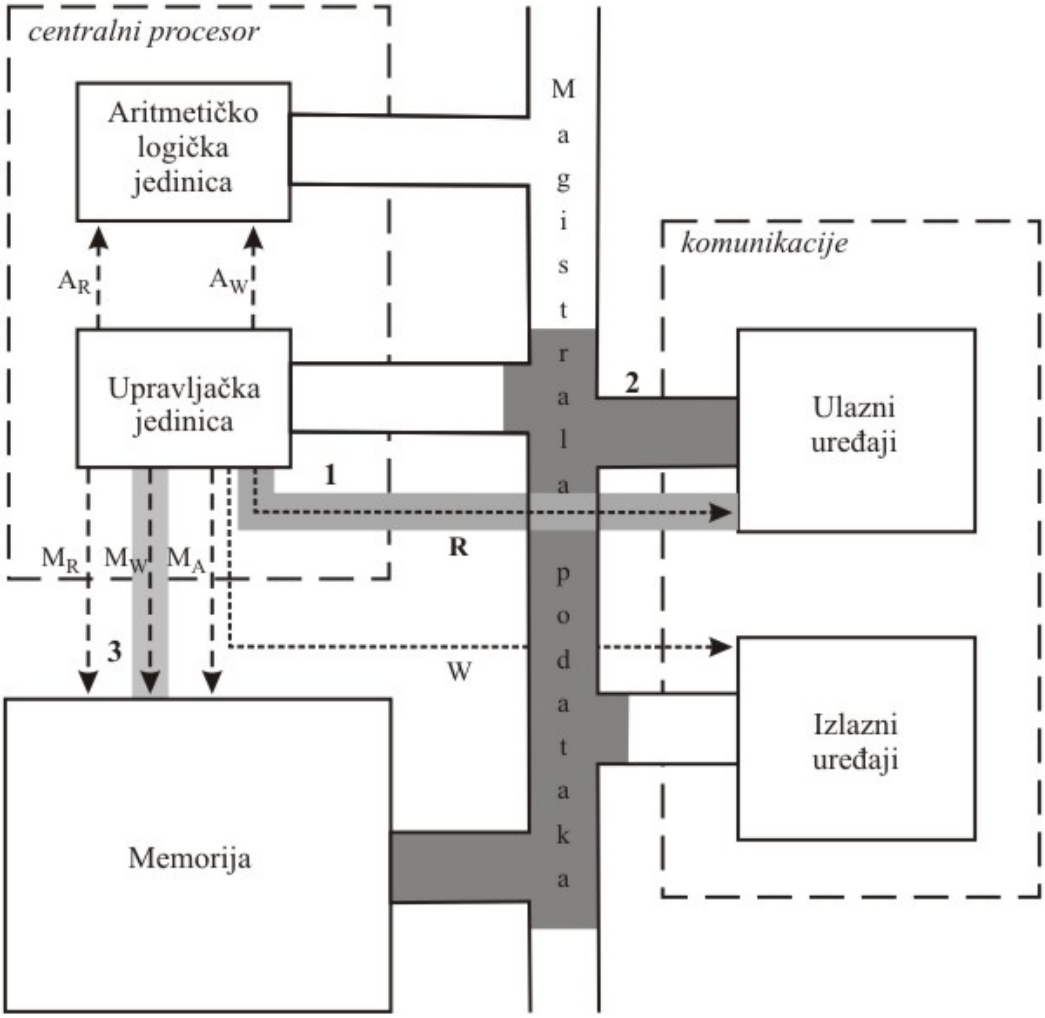
Ovaj problem rešava upravljačka jedinica tako što prema memoriji pošalje adresu memorijske lokacije (preko adresne magistrale), a posredstvom upravljačkog signala $\mathbf{M_A}$ obavesti memoriju da prihvati adresu (slika 4.5.). Proces čitanja podataka sa ulaza i smeštanja u memoriju zahteva nekoliko koordiniranih aktivnosti.



Slika 4.5. Pošto glavna memorija sadrži mnogo potencijalnih lokacija za smeštanje podatka, upravljačka jedinica mora da pošalje potrebnu adresu na magistralu i saopšti memoriji da je preuzme ($\mathbf{M}_A$)

Upravljačka jedinica mora, najpre, da pošalje memoriji adresu željene lokacije (slika 4.5.), i da sačeka potrebno vreme da omogući memoriji da prihvati adresu, a zatim aktivira ulaznu jedinicu pomoću linije **R**.

Treba uvek imati u vidu činjenicu da se sve operacije odvijaju u nekom konačnom intervalu vremena, bez obzira što su računari izuzetno brzi, i da se sve operacije obavljaju sukcesivno, tj. jedna za drugom. Ulazna jedinica, pod dejstvom upravljačkog signala na liniji **R,** izvršava postavljeni zadatak tako što postavlja podatak na magistralu podataka.



Slika 4.6. Pošto memorija već "zna" koju adresu (lokaciju) treba koristiti, upravljačka jedinica naređuje ulaznoj jedinici da pošalje podatak na magistralu (signal **R**), i naređuje memoriji da preuzme podatak sa magistrale (signal $\mathbf{M_W}$)

Memoriji se tada upućuje zahtev, preko linije $\mathbf{M_W}$, da prihvati podatak koji je ulazna jedinica postavila na magistralu i da ga smesti na izabranu lokaciju (slika 4.6). Sve ove aktivnosti su koordinirane u vremenu da bi se obezbedilo pouzdano kopiranje podataka sa neke od ulaznih jedinica u određenu lokaciju u operativnoj memoriji.Podaci se iz memorije najčešće prenose (kopiraju) u aritmetičko-logičku jedinicu, odnosno u neki od registara koji su joj pridruženi (akumulatori itd.). Kao i kod ulaza podataka, za prenos podataka u memoriju potrebno je izvršiti nekoliko koordiniranih aktivnosti.

Prvo, upravljačka jedinica postavlja na adresnu magistralu adresu memorijske lokacije u kojoj je zapisan podatak, a zatim šalje upravljački signal ka memoriji da bi ona prihvatila adresu. **Drugo**, upravljačka jedinica šalje memoriji, preko

kontrolne linije $\mathbf{M_R}$, upravljački signal kojim se kopira sadržaj izabrane memorijske lokacije na magistralu, odnosno, čita se podatak iz selektovane lokacije, Kada se podaci nađu na magistrali, upravljačka jedinica aktivira kontrolnu liniju $\mathbf{A_W}$, kojom naređuje aritmetičko-logičkoj jedinici da preuzme podatak sa magistrale.

Ne smemo zaboraviti da **ALU** sadrži u sebi nekoliko različitih registara, pa su samim tim potrebni još neki kontrolni signali da bi se tačno odredio registar u **ALU** koji prihvata podatak.

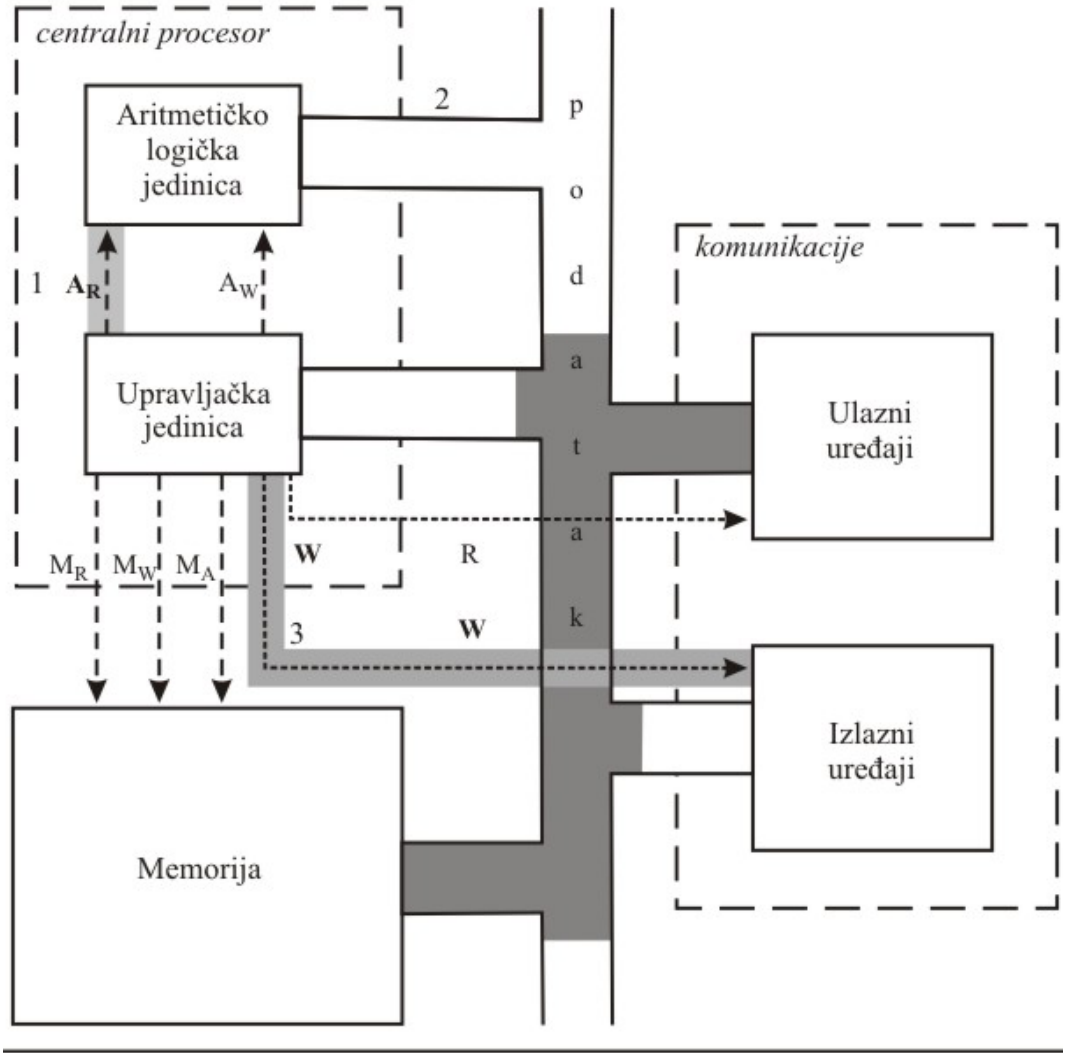
Prebacivanje podatka iz **ALU** u neku lokaciju memorije obavlja se na sličan način. Upravljačka jedinica prvo postavlja na adresnu magistralu adresu memorijske lokacije, zatim upravljački signal $\mathbf{A_W}$ da bi **ALU** postavila podatak na magistralu podataka. Nakon toga se šalje upravljački signal $\mathbf{M_W}$ koji kazuje memoriji da prihvati podatke sa magistrale.

IZLAZ PODATAKA

Podatak koji se šalje spoljašnjem svetu mora biti kopiran na magistralu bilo iz aritmetičko-logičke jedinice (korišćenjem upravljačke linije $\mathbf{A_R}$ kao na slici 4.7), bilo iz memorije (korišćenjem upravljačke linije $\mathbf{M_R}$ kao na slici 4.8).

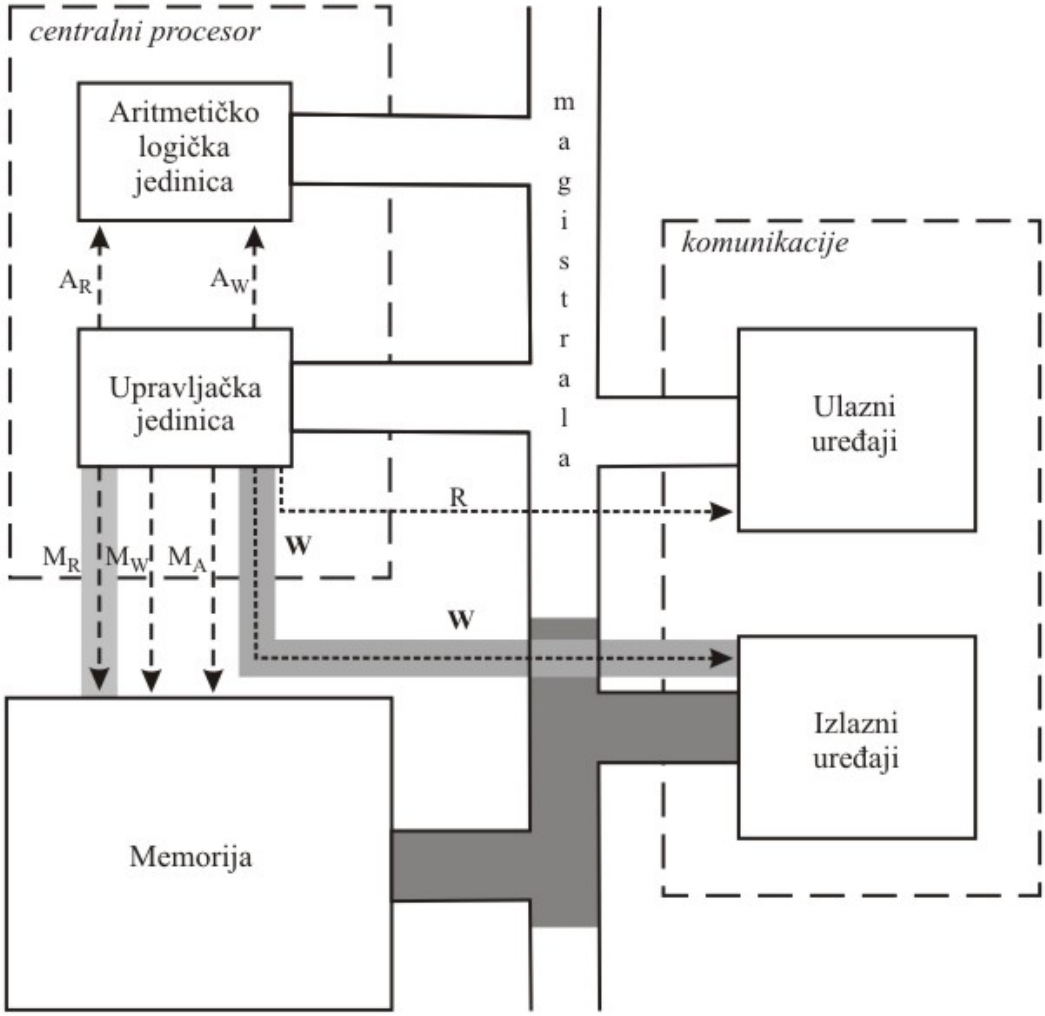
Aktiviranje upravljačke linije **W** dopušta izlaznoj jedinici da prihvati podatak sa magistrale. Ako se izlaznom uređaju šalje podatak iz memorije, upravljačka logika prethodno mora da obezbedi adresu memorijske lokacije i da je aktivira pomoću signala $\mathbf{M_A}$. Kada upravljačka jedinica pošalje memoriji signal za čitanje (preko kontrolne linije $\mathbf{M_R}$), sadržaj izabrane lokacije (tj. podatak) prenosi se na magistralu podataka.

Na kraju procesa prenosa podatka iz memorije u izlazni uređaj upravljačka logika mora da generiše signal **W** nakon čega izlazni uređaj preuzima podatak sa magistrale.



Slika 4.7. Podatak može biti kopiran iz aritmetičko-logičke jedinice na izlaznu jedinicu tako što se prvo naredi **ALU** da pošalje podatak na magistralu (signal $\mathbf{A}_R$), a potom se aktivira izlazni uređaj (signal **W**)

Broj ulaznih i izlaznih uređaja takođe može biti veliki, pa se i za izbor jednog od njih koriste dodatni kontrolni signali, tj. upravljačka logika mora da generiše (izračuna) potrebne adrese i za svaki periferijski uređaj ponaosob.

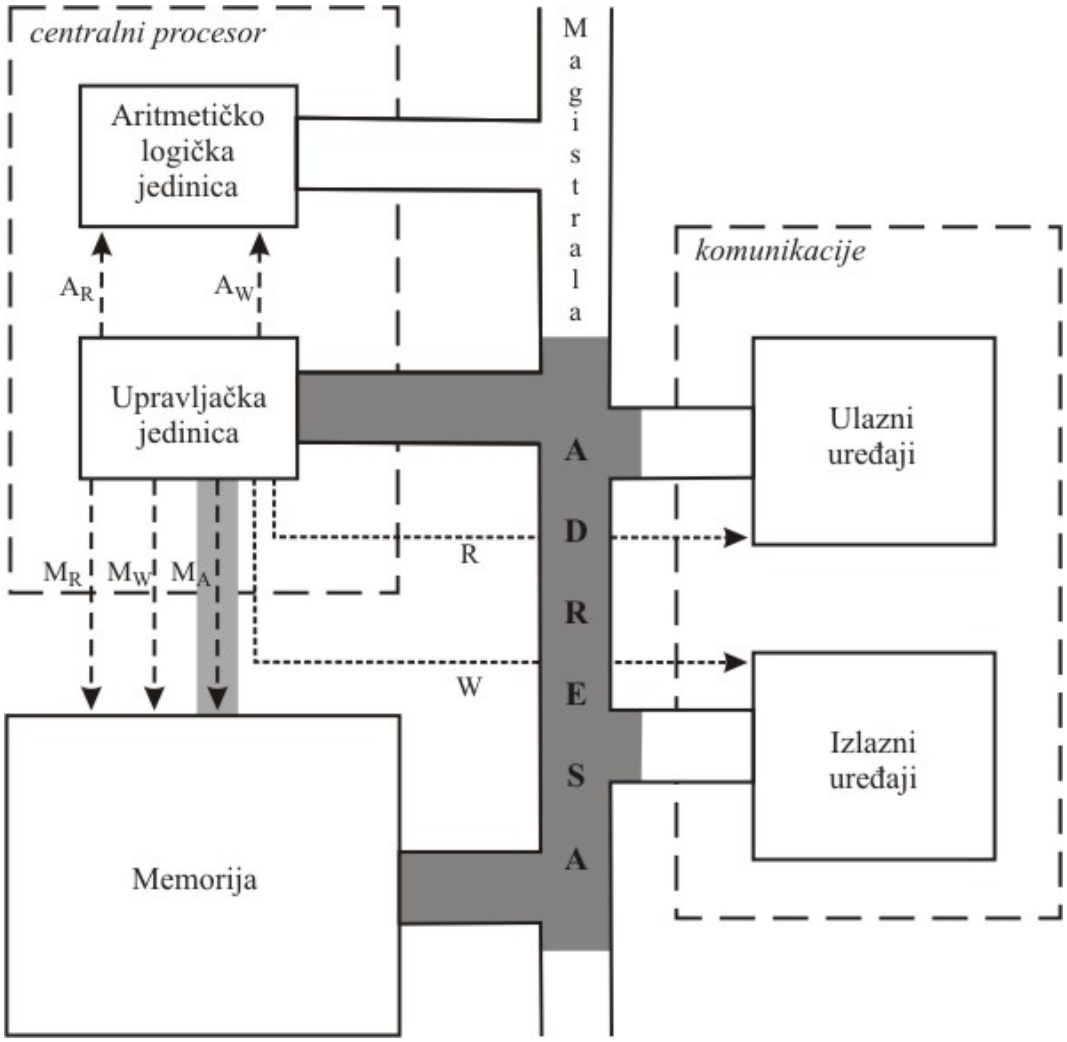


Slika 4.8. Podatak može biti prenet iz memorije na izlaznu jedinicu aktiviranjem signala (linije $\mathbf{M}_R$) za čitanje memorije i aktiviranjem izlaznog kola (signal **W**). (Memorijska lokacija je prethodno izabrana pomoću adrese i signala $\mathbf{M}_A$)

PRIBAVLJANJE INSTRUKCIJA

Glavni problem za jedan računar je sledeći: kako može upravljačka jedinica da zna koje upravljačke signale i kada treba da generiše? Ova informacija se dobija iz niza instrukcija koje su smeštene u memoriji. Upravljačka jedinica mora da pribavi instrukciju iz memorije pre nego što pod njenim dejstvom počne da generiše upravljačke signale.

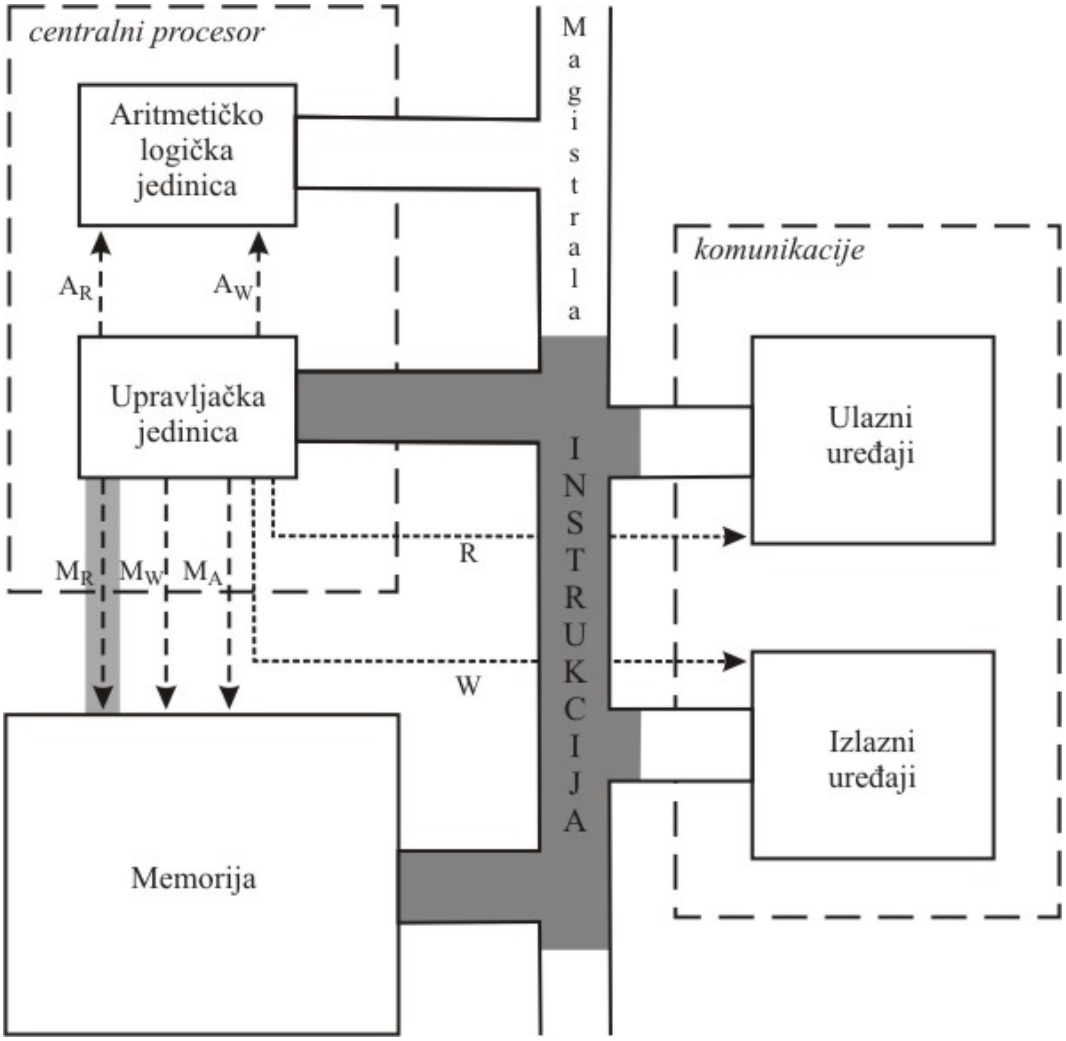
Tokom pribavljanja instrukcije, upravljačka jedinica postavlja adresu memorijske lokacije u kojoj je zapisana potrebna instrukcija. Kada je adresa postavljena na adresni deo magistrale upravljačka jedinica pomoću signala $\mathbf{M}_A$ " kazuje" memoriji da treba da je prihvati (slika 4.9).



Slika 4.9. Da bi pribavila instrukciju iz memorije,upravljačka jedinica mora memoriji da pošalje adresu lokacije u kojoj se nalazi instrukcija

Upravljačka jedinica zatim šalje memoriji upravljački signal $\mathbf{M_R}$, nakon kojeg se čita sadržaj selektovane (adresirane) lokacije i postavlja se na magistralu. Instrukcija koja je postavljena na magistralu (slika 4.10), dostupna je svim delovima računara (ali svi su neaktivni), ali automatski je preuzima upravljačka jedinica.

Za razliku od podataka čiji tokovi kroz računar mogu biti različiti, instrukcije se iz memorije kopiraju isključivo u upravljačku jedinicu.

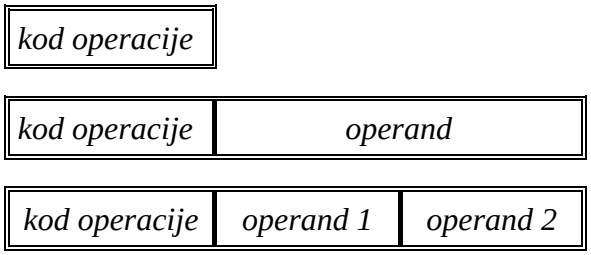


Slika 4.10. Nakon postavljanja adrese željene lokacije, upravljačka jedinica zahteva od memorije da kopira sadržaj te lokacije (instrukciju) na magistralu

4.2. INSTRUKCIJE U MAŠINSKOM JEZIKU

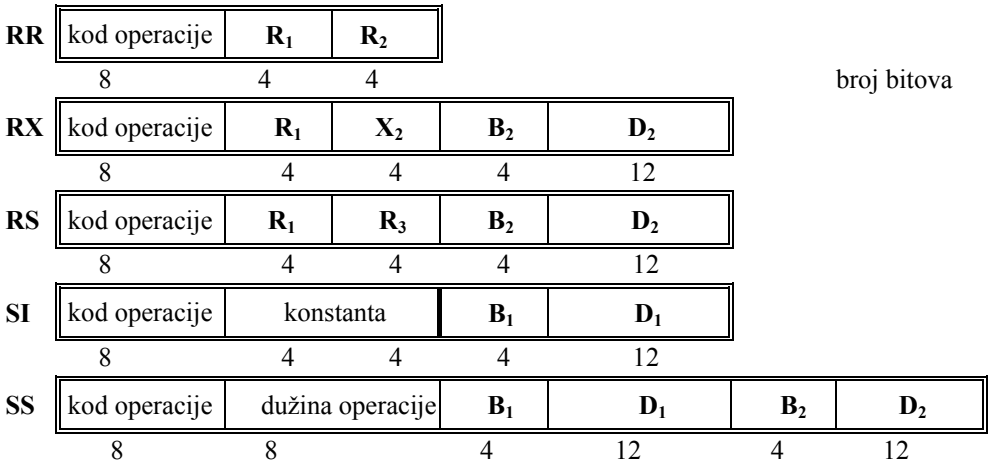
Skoro svi računari rade na principu koncepta memorisanog programa, koji je razvio **John von Neumann** sa svojim timom sredinom 1940. godine. Ovaj koncept dopušta da jedna memorijska lokacija može da sadrži ili podatak ili instrukciju. U ovakvim računarima instrukcije se predstavljaju kao nizovi bita, koji sadrže binarno kodiranu informaciju, koja se odnosi na željenu operaciju.

Na slici 4.11 prikazani su neki od mogućih oblika instrukcije, tj. formata instrukcije.



Slika 4.11. Tipični formati instrukcija

Kod operacije tj. polje obrade, ukazuje na željenu akciju koja treba da se izvrši pod dejstvom te instrukcije. Ta akcija može biti: kopiranje podatka iz memorije u aritmetičko-logičku jedinicu, sabiranje, neka logička operacija itd. Posle koda operacije najčešće sledi jedno ili više polja koja se zovu operandi. Ova polja mogu da zadaju registre (u CPU), ili adresu lokacije u memoriji koja se koristi tokom izvršavanja instrukcije. Na slici 4.12 prikazani su glavni formati instrukcija koje koriste računari IBM 370.



Slika 4.12. Glavni formati instrukcija za računare **IBM** serija **370**

Polja sa oznakom $\mathbf{R_X}$ **(**$\mathbf{R_1}$**,**$\mathbf{R_2}$**,...)** ukazuju na registre u CPU u kojima se nalaze podaci, ili će biti smešten rezultat. Polja $\mathbf{B_X}$ ukazuju na bazne registre, a $\mathbf{D_X}$ na pomeraje (**displacement**) koji se koriste u postupku izračunavanja adrese lokacije. Polja $\mathbf{X_i}$ **(**$\mathbf{X_1}$ **,** $\mathbf{X_2}$ **,...)** ukazuju na indeks-registar koji se koristi pri adresiranju. Uočimo da je polje koda operacije veličine **8** bita, što znači da može postojati 256 različitih kodova operacija.

Polja registara su četvorobitna, što znači da se može zadati jedan od 16 registara. Između polja kod operacije i adresnog dela instrukcije ponekad se umeću dodatni

biti koji ukazuju na specijalne metode adresiranja, (kao na slici 4.13), gde je prikazan format instrukcije računara **Unisys 1100**.

opcode	**mod**	**reg.**	**reg.**	**h**	**i**	**operand**
6	4	4	4	1	1	16

Slika 4.13. Format instrukcija računara **Unisys 1100**

Računari VAX imaju u svom spisku i takve instrukcije, koje imaju veliki broj operanada kao što je to prikazano na slici 4.14.

kod operacije (1 ili 2 bajta)
specifikator operanda 1
specifikator operanda 2
specifikator operanda 3
. .
specifikator operanda N

Slika. 4.14. Opšti format instrukcije **VAX** računara

Kada govorimo o formatu instrukcije očigledno je da svaki računar ima svoj format, ili čak više različitih formata, pa se ne može izvršiti nikakva generalizacija, no, uz određena apstrahovanja, moguće je zaključiti sledeće: sve instrukcije sadrže polje koda operacije - **opcode**.

Što se tiče polja adrese operanda, postoji više vrsta instrukcija:

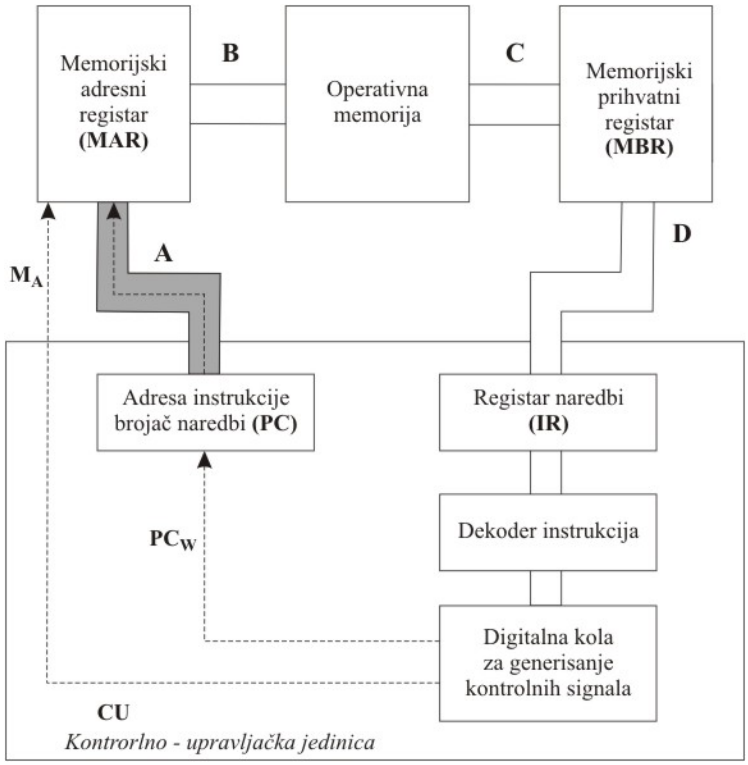
- nulaadresne, bez polja operanda,
- jednoadresne, sa jednim poljem operanda,
- dvoadresne, sa dva polja operanada,
- troadresne, sa tri polja operanada,
- četvoroadresne, sa četiri polja operanada.

Od broja operanada, odnosno adresnosti instrukcija, u mnogome zavisi efikasnost obrade podataka pomoću računara. Dugačke naredbe (višeadresne instrukcije), zahtevaju upotrebu dugačkih memorijskih reči, ili se pribavljaju u više pristupa memoriji. Programi su pregledniji, čitljiviji i kraći, tj. zauzimaju manji broj memorijskih lokacija. Naizgled, upotreba višeadresnih instrukcija povećava

efikasnost obrade. No, u stvarnosti to nije slučaj, pa su se četvoroadresni računari koristili samo u početnoj, eksperimentalnoj fazi razvoja računara. Već prvi računar prve generacije **EDVAC,** koji je počeo sa radom 1951. godine, bio je jednoadresni, a imao je sledeći format instrukcije: **opcode A** .
Kada se kaže jednoadresna, onda se misli isključivo na adresiranje memorijskih lokacija. Adrese registara u centralnom procesoru, kao i implicitne adrese sadržane u kodu operacije ne određuju adresnost instrukcije.

4.3. FAZA PRIBAVLJANJA MAŠINSKIH INSTRUKCIJA

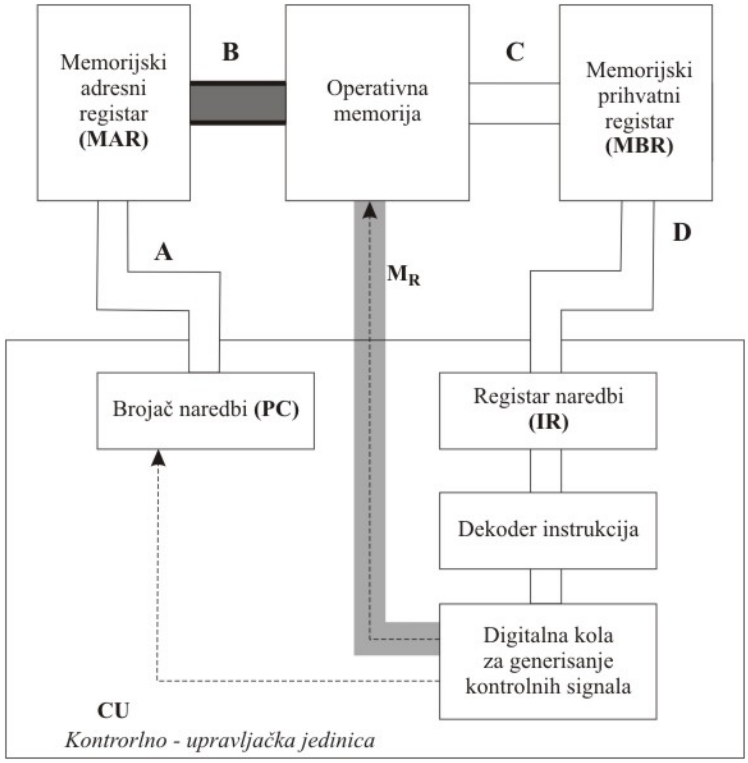
Nezavisno od veličine i formata instrukcije, svaka instrukcija mora najpre biti kopirana (pribavljena) iz memorije u upravljačku jedinicu, a tek posle toga otpočinje njeno izvršavanje i ona preuzima kontrolu nad procesorom. Posebne logičke mreže, u upravljačkoj jedinici, koriste se da dekodiraju i izvrše instrukciju. Niz aktivnosti koje su potrebne da se instrukcija prenese iz memorije u centralni procesor, nazivamo faza pribavljanja instrukcije. Za ova razmatranja koristićemo nešto složeniji model centralnog procesora koji je prikazan na slici 4.15.



Slika 4.15. Adresa lokacije u kojoj se nalazi sledeća instrukcija zapisana je u brojaču naredbi. Adresa mora biti preneta u memorijski adresni registar pre nego što se pristupi lokaciji i iz nje se pročita instrukcija.

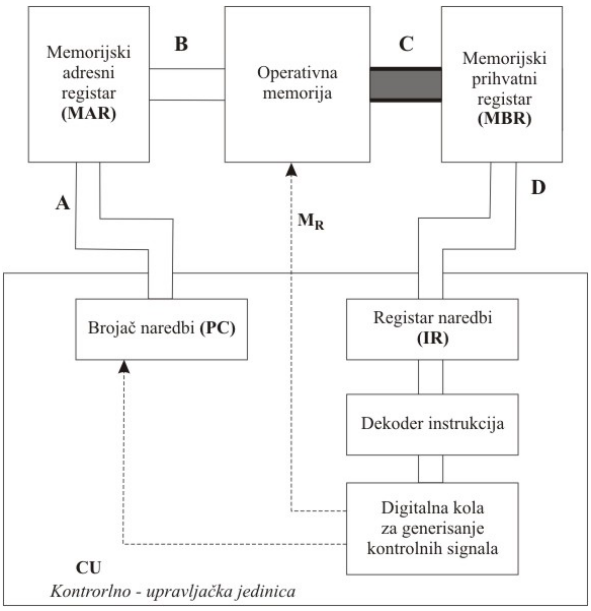
Adresa sledeće naredbe, koja će biti izvršena, nalazi se zapisana u registru specijalne namene koji se zove brojač naredbi ili programski brojač (**program counter, PC**). Proces pribavljanja instrukcije započinje kopiranjem sadržaja programskog brojača u memorijski adresni registar (**MAR**) preko linija označenih sa A (kao na slici 4.15). (Da bi se to ostvarilo najpre **CU** generiše signal $\mathbf{PC_W}$, da se prenese sadržaj programskog brojača na magistralu **A**, a zatim **CU** generiše već poznati signal $\mathbf{M_A}$, koji naređuje memoriji, tj., memorijskom adresnom registru da prihvati adresu. Ove aktivnosti se stalno ponavljaju pri pribavljanju, i zato ih više nećemo pominjati.)

Adresa koja se nalazi u **MAR**-u, prolazi kroz logičku mrežu za dekodiranje tipa 1 od 2^n, i generiše se signal koji aktivira samo jednu od mogućih 2^n lokacija u memoriji. Stvarni pristup memoriji događa se pod kontrolom signala za čitanje ($\mathbf{M_R}$) koji postavlja upravljačka jedinica (vidi sliku 4.16).



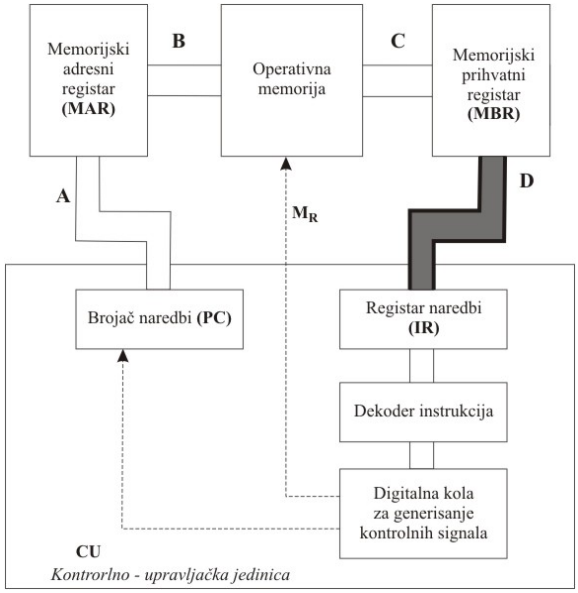
Slika 4.16. Postavljanje signala $\mathbf{M_R}$ " kazuje " memoriji da kopira na magistralu **C** podatak zapamćen na adresi na koju pokazuje **MAR**

Nakon prijema signala za čitanje, podatak (koji ovog puta predstavlja instrukciju) se iz izabrane memorijske lokacije kopira, preko magistrale **C,** u drugi specijalni registar, memorijski međuregistar ili prihvatni registar memorije (**memory buffer register, MBR**), kao što se vidi na slici 4.17.



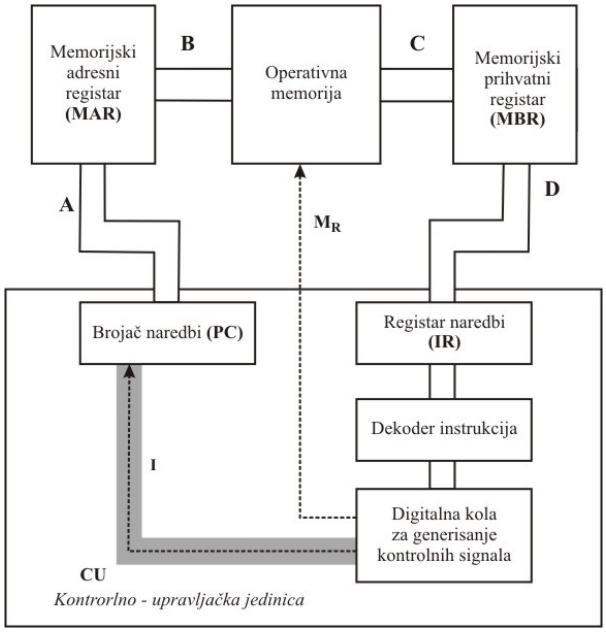
Slika 4.17. Podatak koji je pročitan iz memorije kopira se u **MBR**

Podaci koji se nalaze u **MBR**-u prebacuju se preko magistrale **D** (slika 4.18) u specijalni registar takozvani **registar instrukcija** ili registar naredbi (**instruction register, IR**). Iz registra naredbi, deo instrukcije koji čini kod operacije, prebacuje se u logičku mrežu koja se zove dekoder instrukcija, gde se najpre proverava ispravnost koda, a zatim se kod dekodira, a kasnije generiše niz upravljačkih signala, koji obezbeđuju izvođenje one operacije koja je zadata u kodu.



Slika 4.18. Tokom faze pribavljanja sadržaj **MBR** se kopira u registar naredbi (**IR**)

Poslednji korak u toku faze pribavljanja instrukcije predstavlja inkrementiranje programskog brojača (slika 4.19) pomoću upravljačkog signala **I**. Novi sadržaj programskog brojača je jednak starom sadržaju uvećanom za jedan (**PC = [PC] + 1**).



Slika 4.19. Brojač naredbi (**Program Counter, PC**) se inkrementira na kraju ciklusa pribavljanja instrukcije

Ovaj korak postavlja novu vrednost u programski brojač, tako da on sada sadrži adresu nove lokacije u memoriji, koja neposredno sledi iza lokacije koja sadrži instrukciju (kod operacije instrukcije) koja je upravo pribavljena. Na toj lokaciji se najčešće nalazi sledeća instrukcija. Ali, kod nekih računara koji imaju kratke memorijske reči (naročito kod osmobitnih i šesnaestobitnih mikroračunara) na sledećoj memorijskoj lokacije se često nalazi operand instrukcije koja je upravo pribavljena. Iz ovoga neposredno proističe pretpostavka da su operandi (podaci, njihove adrese ili instrukcije), koje će biti korišćene neposredno po pribavljanju instrukcije, smešteni u memoriji na lokaciji odmah iza tekuće instrukcije.

Ova pretpostavka se zasniva na činjenicama da operandi neposredno slede iza koda operacije, i da se većina instrukcija izvršava sekvencijalno, odnosno jedna za drugom, onim redosledom kojim su smeštene u memoriju (odnosno onim redosledom kojim su napisane u programu).

Sva ova razmatranja su aproksimativna, jer instrukcije se među sobom puno razlikuju od računara do računara. U jednom računaru postoje različite instrukcije po dužini, po mestu gde su operandi (memorija ili registri), po broju operanada, po

načinima adresiranja itd. Zato ćemo se ograničiti na razmatranje jednoadresnih instrukcija (i računara), jer su one najrasprostranjenije.

Adresnost naredbi bitno utiče na efikasnost obrade podataka u računaru, a meri se sledećim parametrima:

- vremenom potrebnim za izvođenje programa (meri se ukupnim brojem memorijskih ciklusa potrebnih za izvršavanje svih instrukcija programa),
- memorijskim prostorom potrebnim za zapisivanje programa.

Četvoroadresne naredbe, kojima se opisuju binarne aritmetičke i logičke operacije, sadrže: kod operacije, adrese prvog i drugog operanda, adresu rezultata i adresu sledeće naredbe.

Ali, ako se naredbe smeštaju u memoriju onim redosledom kojim se izvršavaju, adresa sledeće naredbe nije neophodna, već se uvodi specijalni hardver, odnosno brojač naredbi (**program counter, PC**). Brojač naredbi je registar specijalne namene koji u sebi uvek sadrži adresu sledeće instrukcije, koja tek treba da se pribavi i izvrši. Na ovaj način se potrebni broj adresa smanjio, tj. dovoljne su troadresne instrukcije. Pošto se pri izvo|enju programa ponekad mora odstupiti od sekvencijalnog redosleda izvođenja instrukcija (tj. sledeća instrukcija nije na sledećoj memorijskoj lokaciji), moraju se uvesti specijalne instrukcije za kontrolu toka programa, tj. instrukcije skoka (**jump**) ili grananja (**branch**).

Dalje pojednostavljenje se postiže uvođenjem ograničenja tako da se rezultat binarnih operacija uvek privremeno smešta u neki registar **CPU**, najčešće u akumulator. No, da bi se dobijeni rezultat sačuvao za dalju obradu, neophodno ga je preneti u neku lokaciju u operativnoj memoriji, odnosno, neophodno je imati i naredbu za prenos sadržaja akumulatora u memorijsku lokaciju (**store, move**).

Da bi bila moguća upotreba samo jednoadresnih naredbi uvode se dodatna ograničenja u pogledu smeštanja operanada i rezultata obrade. Naime, kod binarnih operacija jedan od operanada je uvek u jednom od akumulatora, pa je neophodno proširiti spisak naredbi uvođenjem naredbi za punjenje akumulatora, tj. za prenos sadržaja iz neke memorijske lokacije u akumulator (**load, move**).

Da bi koristili naredbe bez adresa, svi operandi (tj. podaci) moraju biti u tačno određenim registrima centralnog procesora, a njihove adrese su sadržane u samom kodu operacije. Da bi se operandi smestili u određene registre, set instrukcija računarskog sistema mora se proširiti instrukcijama za prenos podataka između dva registra u centralnom procesoru.

Na kraju, možemo zaključiti sledeće: smanjenje adresnosti instrukcija postiže se uvođenjem novih hardverskih komponenti u centralni procesor, novih instrukcija i određenih ograničenja u pogledu smeštanja instrukcija, operanada i razultata operacija.

4.4. FAZA IZVRŠAVANJA MAŠINSKIH INSTRUKCIJA

Nakon što je instrukcija smeštena u upravljačku jedinicu i u programski brojač postavljena nova adresa memorijske lokacije, faza pribavljanja je završena. Kontrolu nad upravljačkom jedinicom preuzima pribavljena instrukcija, i otpočinje faza izvršenja.

Da bi lakše shvatili ovu fazu, posmatrajmo je na primeru izvođenja jedne elementarne operacije sabiranja dva broja:

sum = num1 + num2 .

Prevedeno na govorni jezik ova naredba kazuje sledeće: brojnu vrednost (na primer: 05_{10}) zapamćenu u memorijskoj lokaciji sa simboličkom adresom **num1,** sabrati sa brojem (na primer: 07_{10}) zapamćenim u lokaciji sa simboličkom adresom **num2** i rezultat smestiti u lokaciju sa simboličkom adresom **sum**.

Većina računara smešta podatke u memoriju, ali logičke, aritmetičke i druge operacije izvodi, po pravilu, nad sadržajima registra u centralnom procesoru, tačnije u aritmetičko-logičkoj jedinici. To odmah ima za posledicu da napred navedena naredba ne bi mogla da se izvede neposredno, već u nekoliko koraka (tj. kao niz nekoliko mašinskih instrukcija).

Najpre, treba podatke prebaciti iz memorije u registre centralnog procesora, zatim izvršiti sabiranje, a potom rezultat vratiti natrag u memoriju. Stvarni niz mašinskih instrukcija koje rešavaju navedeni problem varira od računara do računara. Jedna od mogućnosti je rešenje pomoću tri mašinske instrukcije.

Prva instrukcija ima kod operacije koji ukazuje na kopiranje podatka iz memorije u registar (npr. akumulator u ALU ili slično). Ovo podseća na postupak punjenja akumulatora, pa se simbolički ova operacija može nazvati punjenje (**load**). Ova instrukcija mora ukazivati na memorijsku lokaciju gde je smešten podatak, a koju zovemo **num1.**

Druga instrukcija ima kod operacije koji kazuje da sadržaj druge memorijske lokacije (koju zovemo **num2**) treba dodati (sabrati-**add**) broju u registru (koji je korišćen u prethodnoj instrukciji), a rezultat ponovo vratiti u taj registar. I ova instrukcija sadrži ukazatelj na podatak smešten u memoriji.

Treća instrukcija kazuje da rezultat treba preneti iz registra u centralnom procesoru, u neku lokaciju u memoriji i zapamtiti ga (**store**). Kao i u prethodnim, i u ovoj instrukciji moramo na neki način ukazati na adresu željene memorijske lokacije, a u ovom slučaju to je lokacija simboličkim imenom **sum**.

Neka je promenljiva **num1** smeštena u memoriji na lokaciji $\mathbf{26}_{10}$. Onda je njena binarna adresa $\mathbf{00011010}_2$, odnosno $\mathbf{1A}_{(16)}$ u heksadecimalnom brojnom sistemu. Po

analogiji, **num2** je smešten na sledećoj memorijskoj lokaciji čija je adresa $\mathbf{1B}_{(16)}$, a **sum** je na lokaciji sa adresom $\mathbf{1C}_{(16)}$.

Pretpostavimo, takođe, da su sve instrukcije iste dužine, tj. 2 bajta. U prvom bajtu je kod operacije, a u drugom adresa operanda. Pretpostavimo i to da su ove instrukcije smeštene u memoriju jedna za drugom počev od lokacije sa adresom $\mathbf{100}_{(16)}$, kao što je to prikazano na slici 4.20 b). Izgled programa na asembleru dat je na slici 4.20 a):

adresa	**sadržaj**
100:	**load data**
101:	**1A**
102:	**add data**
103:	**1B**
104:	**store data**
105:	**1C**

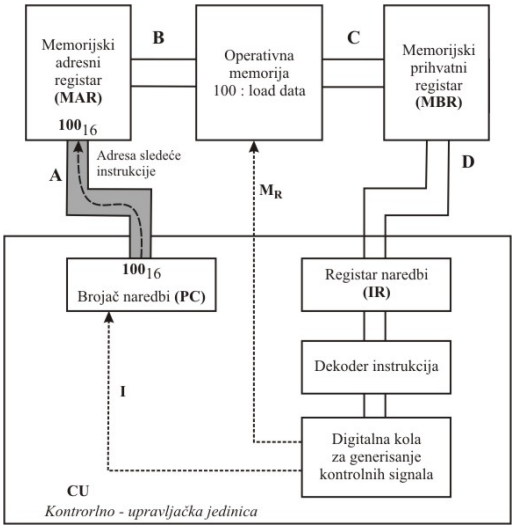
a) mašinski program

podatak	simbolička adresa	fizička adresa	memorija
05	num1	1A	00000101
07	num2	1B	00000111
?	sum	1C	-
		. . .	. . .
		100	load data
		101	00011010
		102	add data
		103	00011011
		104	store data
		105	00011100
		106	. . .

b) stanje memorije

Slika 4.20. Sabiranje dva broja

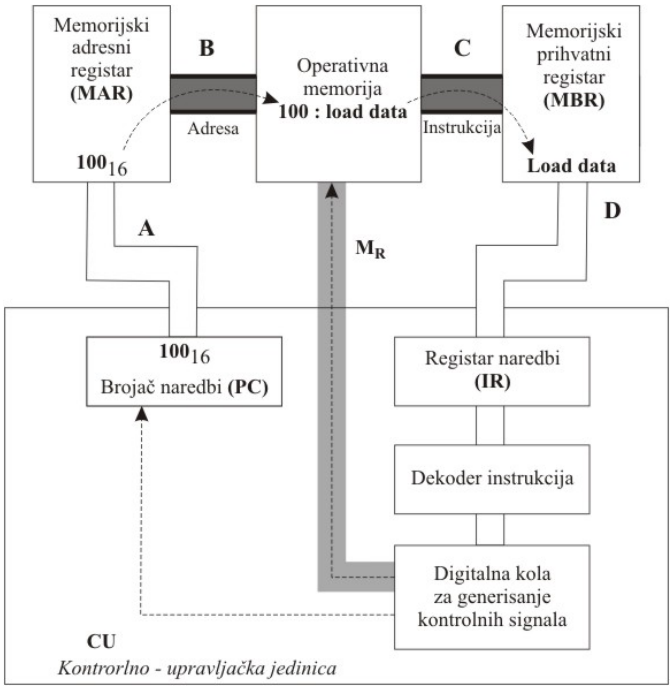
Uočimo da smo zbog preglednosti i razumljivosti koristili simbolička imena operacija (inače, u memoriji, one su kodirane kao osmobitni nizovi **0** i **1,** tj. prikazane su kao binarni brojevi).



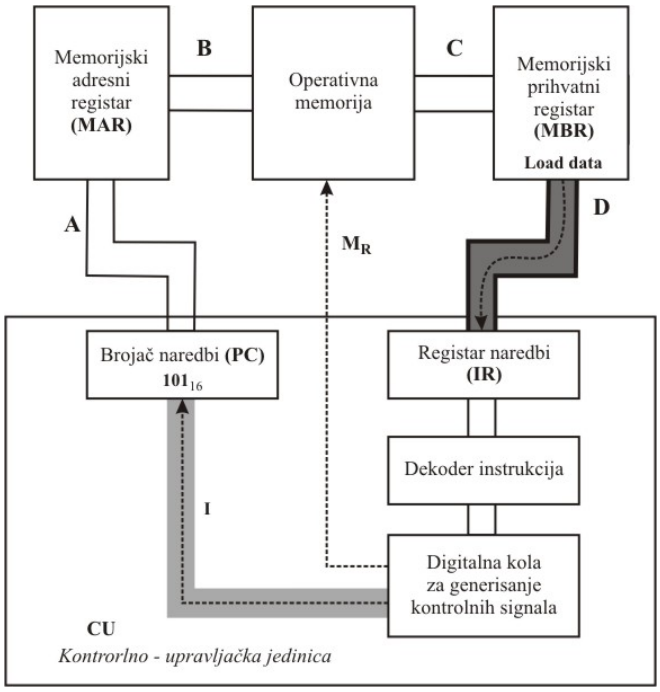
Slika 4.21. Pribavljanje instrukcije sa lokacije $\mathbf{100}_{16}$, najpre se sadržaj programskog brojača prenosi u memorijski adresni registar (**MAR**)

Prvi korak u izvođenju ovog niza instrukcija je otpočinjanje faze pribavljanja prve instrukcije. U brojač naredbi se upiše broj $\mathbf{100}_{(16)}$, i otpočinje faza pribavljanja. Po završetku faze pribavljanja, u registru naredbi (**IR**) nalazi se kod operacije (**load data**), a programski brojač se inkrementira na $\mathbf{101}_{(16)}$. Šta se tokom pribavljanja instrukcije zbivalo u računaru prikazano je na slikama 4.21, 4.22 i 4.23.

Tokom dekodiranja ove naredbe, upravljačka logička kola utvrdila su da treba sadržaj jedne memorijske lokacije prekopirati u jedan od registara CPU. U kodu operacije (**load**) takođe je specificirano da polje operanda sadrži adresu podatka. Da bi podatak mogao biti kopiran u neki registar centralnog procesora, njegova adresa mora biti prethodno određena. Brojač naredbi, čiji je sadržaj uvećan za **1** na kraju ciklusa pribavljanja, sadrži adresu memorijske lokacije u kojoj je smešten operand.



Slika 4.22. Kada je upravljački signal za čitanje postavljen (**R**), u memoriji se pronalazi instrukcija na adresi na koju ukazuje **MAR**, i prenosi je u **MBR**



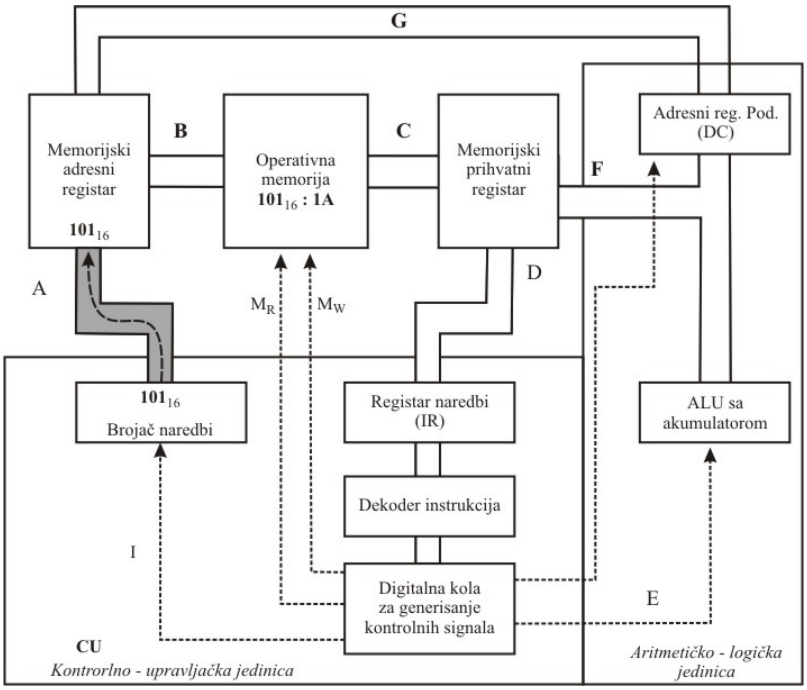
Slika 4.23. Tokom poslednje faze pribavljanja instrukcije, kod operacije se prenosi iz **MBR** u registar instrukcija (**IR**), a programski brojač se inkrementira.

Proces pribavljanja operanda zahteva od upravljačke jedinice sledeće aktivnosti:

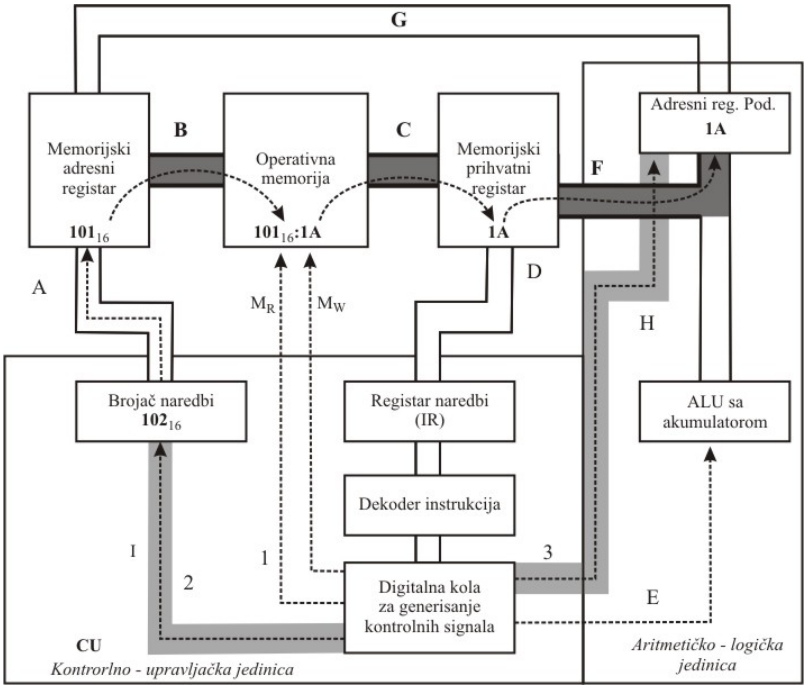
- Kopiranje sadržaja programskog brojača (**PC**) preko adresne magistrale (**A**), u memorijski adresni registar (**MAR**) - slika 4.24.
- Postavljanje signala za čitanje iz memorije ($\mathbf{M_R}$ na slici 4.25).
- Prebacivanje operanda iz memorije pomoću bafer registra (**MBR**) u registar adrese podataka (**data counter, DC**), preko linija **C** i **F** na slici 4.25.
- Inkrementiranje sadržaja brojača naredbi, signal **I** na slici 4.25.

Ponovimo još jednom: podatak koji se nalazi u **MBR** nije broj koji treba preneti u **ALU** ili registre, već adresa tog broja.

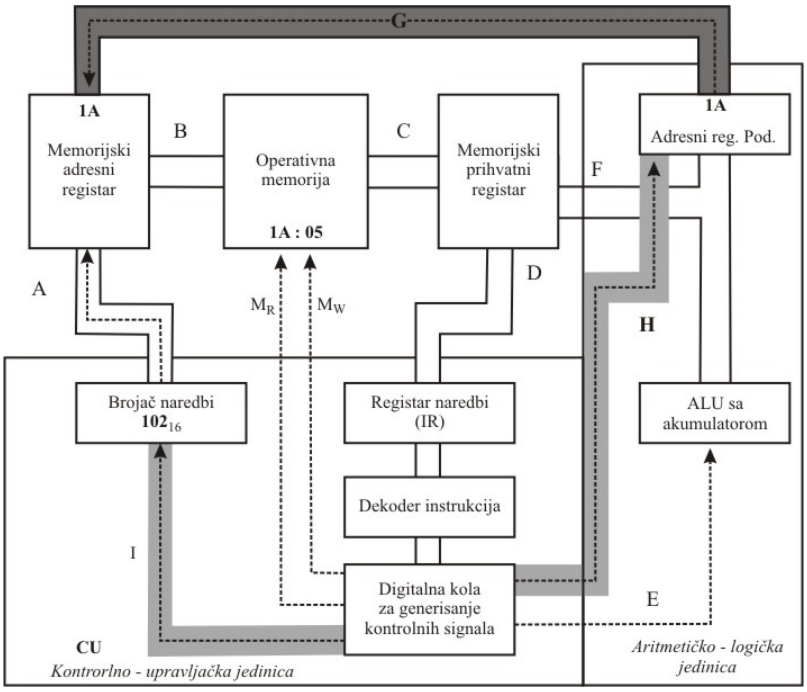
Koristeći tu adresu, upravljačka jedinica će konačno preneti podatak, no za kompletiranje te aktivnosti potrebno je izvršiti još jedan niz elementarnih operacija.



Slika 4.24. Prvi korak u fazi izvođenja instrukcije prenosa je prebacivanje sadržaja **PC**, tj. adrese operanda u **MBR**



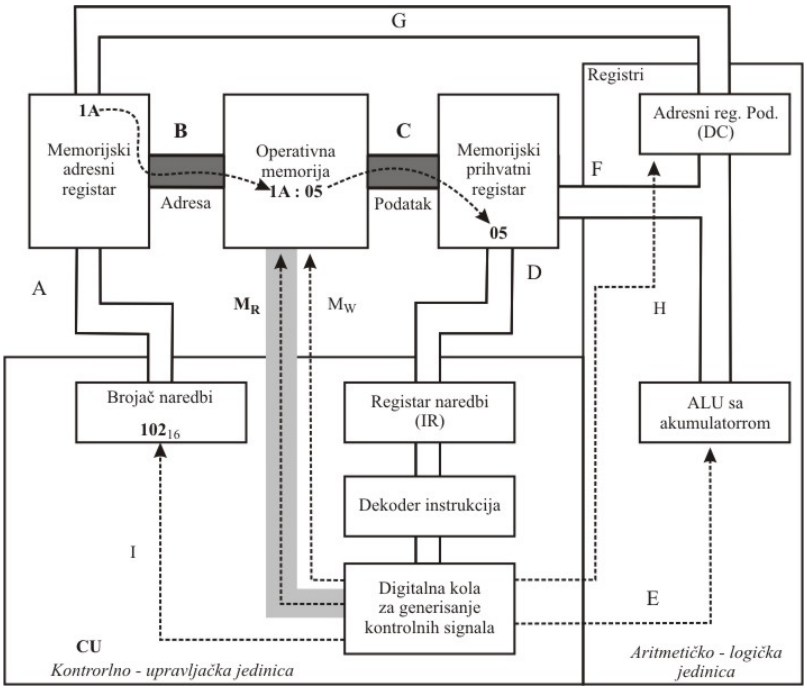
Slika 4.25. Signalom $\mathbf{M_R}$ se čita operand i prebacuje se preko magistrale **C, MBR** i magistrale **F** u brojač podataka. Poslednji korak je inkrementiranje **PC**.



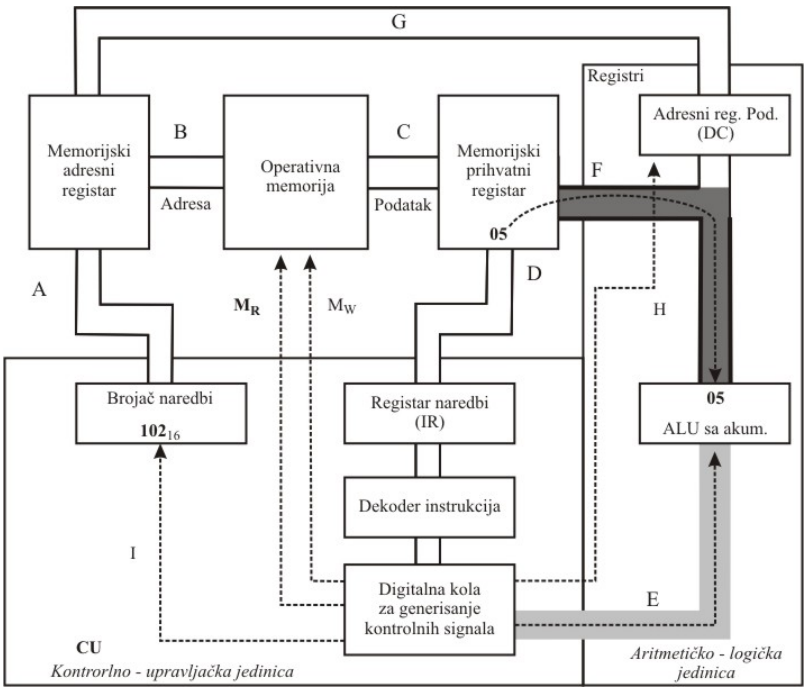
Slika 4.26. Operand je ustvari adresa podatka, pa se iz adresnog registra podataka, preko magistrale **G,** šalje u memorijski adresni registar **MAR**

- Preneti sadržaj brojača podataka (DC) u memorijski adresni registar (MAR) preko adresne magistrale (G na slici 4.26).
- Poslati novi zahtev za čitanje iz memorije (MR na slici 4.27).
- Prihvatiti podatak u MBR (preko linije C, slika 4.27).
- Reći jednom od registara u procesoru da preuzme podatak (kontrolni signal H na slici 4.28).
- Kopirati podatak preko magistrale F u jedan od registara (to je najčešće akumulator, slika 4.28).

Nakon što su sve ove operacije obavljene, faza izvršenja prve instrukcije je završena, i upravljačka jedinica se vraća u režim pribavljanja instrukcije. Programski brojač ukazuje na sledeću instrukciju, tj. na njen prvi deo. (kod operacije) koji je smešten na lokaciji sa adresom $\mathbf{102}_{(16)}$. Adresa iz **PC** se kopira u **MAR** i aktivira se zahtev za čitanje iz memorije. Instrukcija **add data** se (iz lokacije $\mathbf{102}_{(16)}$ u memoriji), kopira u **MBR** i automatski se, preko magistrale **D**, prenosi u registar **IR**. Faza pribavljanja se završava inkrementiranjem brojača podataka na $\mathbf{103}_{(16)}$.

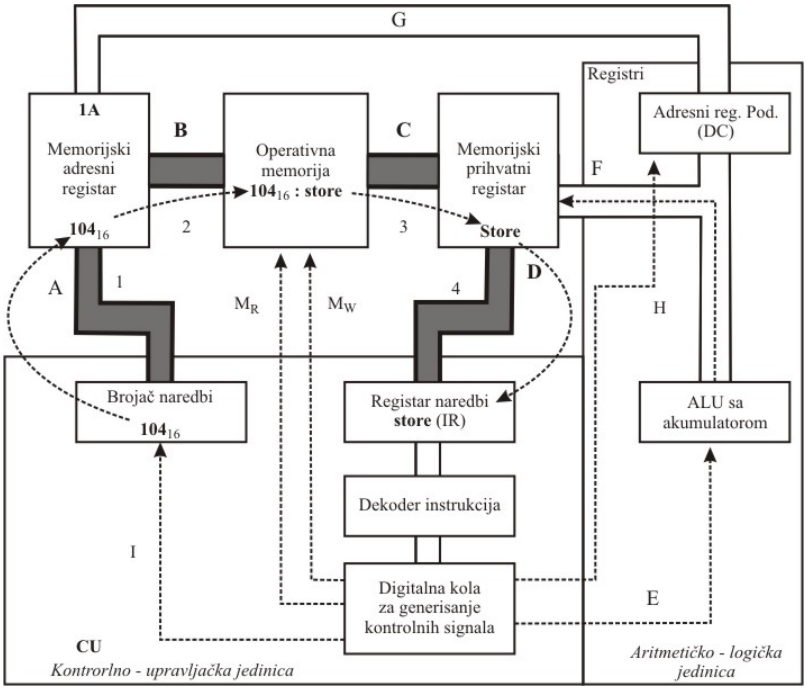


Slika 4.27. Kopira se podatak 05 iz memorije (sa adrese na koju ukazuje operand) u memorijski registar podataka **MBR**



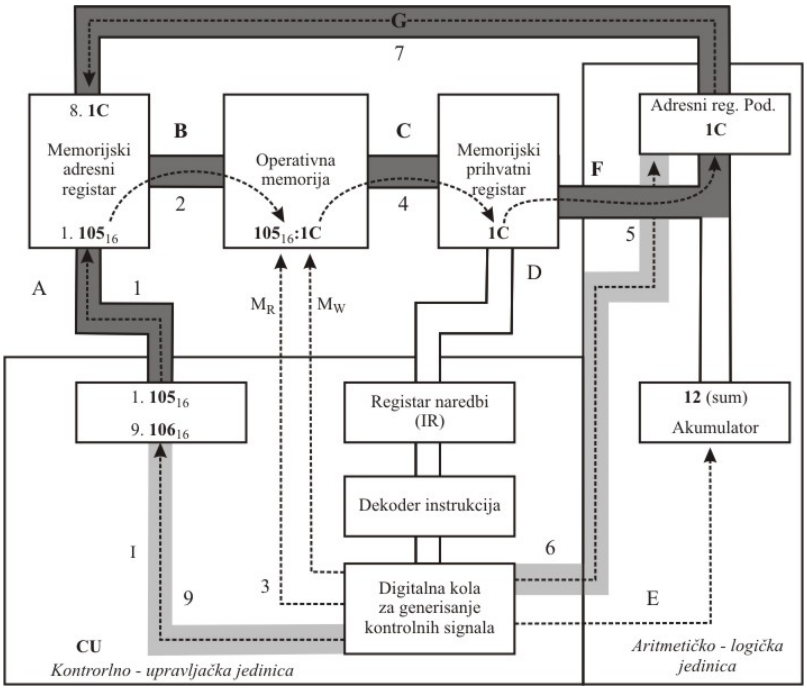
Slika 4.28. Podatak se prenosi iz **MBR** u neki registar u **CPU,** odnosno u akumulator

Uočimo da je faza pribavljanja instrukcije **add** identična fazi pribavljanja instrukcije **load,** prikazanoj na slici 4.21, 4.22 i 4.23. Tokom faze izvršenja instrukcije **add**, upravljačka jedinica mora da pribavi operand i upotrebi ga kao adresu; zatim mora da pristupi memorijskoj lokaciji sa tom adresom i njen sadržaj prebaci u **MBR** i inkrementira **PC** na $\mathbf{104}_{(16)}$, (slično kao u instrukciji **load**). No nakon toga, preko upravljačkih signala **E**, kazuje ALU da prihvati podatak sa magistrale **F** i doda ga sadržaju izabranog registra, tj. izvrši sabiranje, i rezultat sačuva u tom istom registru. Nakon pribavljanja koda operacije programski brojač se inkrementira na $\mathbf{103}_{(16)}$, a nakon pribavljanja operanda inkrementira se na $\mathbf{104}_{(16)}$. Nakon izvršenja instrukcije sabiranja otpočinje faza pribavljanja sledeće instrukcije: prebacuje se sadržaj iz lokacije $\mathbf{104}_{(16)}$ (a to je instrukcija **store**) u registar naredbi (**IR**), i inkrementira brojač naredbi na $\mathbf{105}_{(16)}$, slika 4.29.

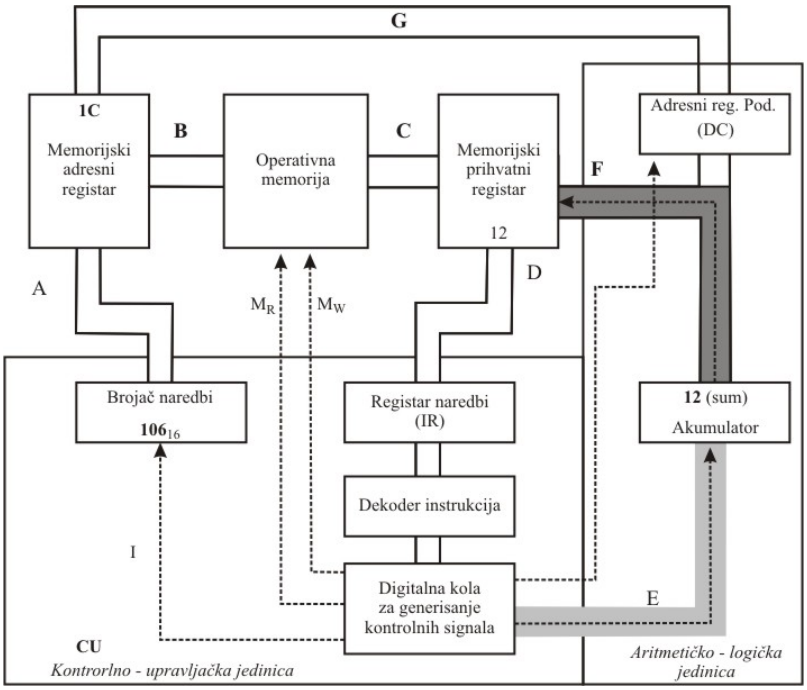


Slika 4.29. Prikaz faze pribavljanja instrukcije **store** sa lokacije $\mathbf{104}_{16}$

Prvi koraci izvršenja instrukcije **store**, su isti kao kod prethodne dve instrukcije. Nakon pribavljanja instrukcije, pristupa se operandu i **PC** se inkrementira na $\mathbf{106}_{(16)}$. Operand se postavlja u **MAR** (preko magistrale **G** slika 4.30), kao adresa lokacije u kojoj će biti smešten sadržaj akumulatora. Zatim se šalje upravljački signal **E** ka akumulatoru, i sadržaj akumulatora se prenosi u **MBR** preko magistrale **F** (slika 4.31).

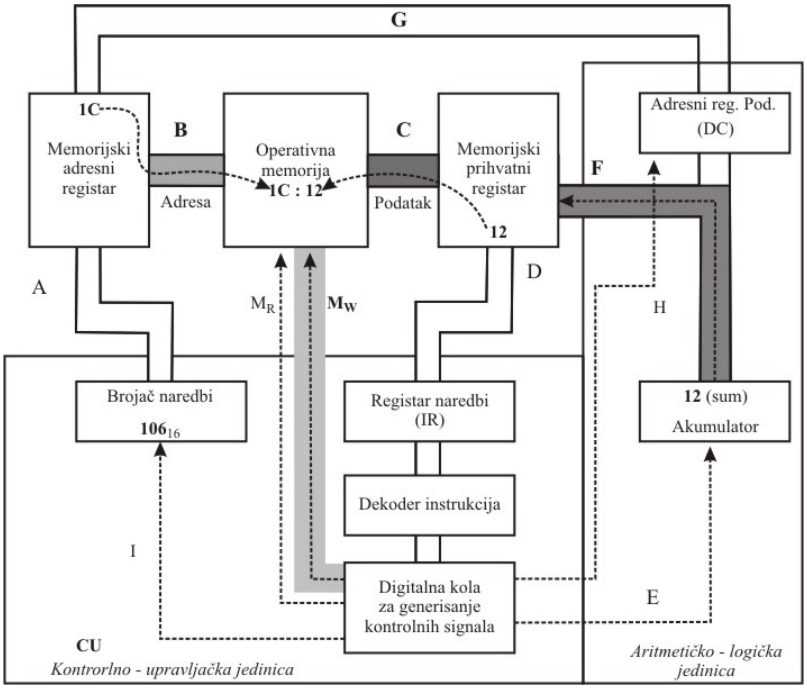


Slika 4.30. Objedinjeni prikaz prenosa operanda preko brojača podataka u **MAR**



Slika 4.31. Prenos podatka iz registra u CPU (iz akumulatora) u **MBR**

Konačno-upravljačka jedinica šalje komandu u memoriju ($\mathbf{M_W}$ **,** kontrolni signal za upisivanje sadržaja u memorijsku lokaciju), a sadržaj memorijskog prihvatnog registra (**MBR**) se smešta na lokaciju čija se adresa nalazi u memorijskom adresnom registru (**MAR**) (vidi sliku 4.32).



Slika 4.32. Slanjem upravljačkog signala **W** (write), vrši se upisivanje podatka iz **MBR** na selektovanu lokaciju u memoriji (na koju ukazuje adresa u **MAR**)

Da još jednom istaknemo: upravljačka jedinica ima nekoliko režima rada, ali dva su bazična:

a) stanje pribavljanja instrukcije (**fetch**), kada se upravljački signali generišu automatski. Kao što se vidi iz opisa pribavljanja instrukcija **load, store** i **add**, ovi upravljački signali su uvek isti, nezavisno od vrste instrukcije koja se pribavlja.

b) stanje izvršavanja instrukcije (**execution**), kada se upravljački signali generišu pod dejstvom koda operacije instrukcije. Za svaku instrukciju imamo drugačiji skup kontrolno-upravljačkih signala. Svaka mašinska instrukcija se realizuje preko drugačijeg niza elementarnih operacija (mikro operacija).

4.5. MIKROPROGRAMSKA ORGANIZACIJA UPRAVLJAČKIH JEDINICA

Da bi logičke mreže, koje ulaze u sastav svih delova računara, mogle da obavljaju svoju funkciju, neophodni su im različiti upravljački signali koje generiše upravljačka jedinica. Postoje dve vrste upravljačkih signala: sinhronizacioni i funkcijski. Sinhronizacioni signali su obično impulsnog oblika i služe za određivanje različitih vremenskih intervala i usklađivanje raznih etapa u radu računara.

Funkcijskim signalima se selektuju različite funkcije na bazi skupa sinhronizacionih signala i skupa signala dobijenih iz registara u procesoru, ili iz spoljašnjeg okruženja upravljačke jedinice (pa čak i iz spoljašnjeg sveta).

Upravljački signali se mogu generisati na dva načina, pa imamo dve vrste organizovanja upravljačke jedinice: hardversku i mikroprogramsku.

Kod hardverske organizacije, upravljački signali se generišu pomoću posebnih digitalnih mreža. Ovakve upravljačke jedinice su brže, ali često i vrlo složene, a nisu ni fleksibilne, niti dostupne korisniku da ih modifikuje po svojim potrebama. Ovakvu organizaciju imaju i **RISC** procesori (**reduced instruction set computers**).

Kod mikroprogramskog ili firmverskog generisanja, upravljački signali su memorisani u posebnoj memoriji (unutar upravljačke jedinice), koja se zove mikroprogramska memorija, i iz nje se čitaju kada je to potrebno. Ovakav upravljački organ je sporiji, jer se generisanje upravljačkih signala obavlja pomoću posebnih programa, mikroprograma, koji se sastoje od mikroinstrukcija. Za svaku mašinsku instrukciju postoji poseban niz mikroinstrukcija pomoću kojih se generišu upravljački signali. Ovakvu organizaciju imaju i **CISC** procesori (**complete instruction set computers**).

Da bi se izvršio neki mikroprogram, potrebno je, najpre, generisati početnu adresu mikroprograma, i to se radi na bazi koda operacije i nekih spoljašnih uslova. Naredbe mikroprograma se obično odvijaju jedna za drugom onim redosledom kojim su zapisane u mikroprogramskoj memoriji. I ovde se koristi poseban registar, kao programski brojač, čiji se sadržaj posle svake mikroinstrukcije inkrementira.

Kao i kod korisničkih programa, i ovde se može odstupiti od normalnog redosleda izvršavanja mikroinstrukcija, tj. mogući su takozvani mikroprogramski skokovi pod dejstvom raznih uzroka. Na hijerarhijskom modelu računarskog sistema, vidimo da su digitalna logička kola obuhvaćena mikrokodom, a nivo mikrokoda je obuhvaćen mašinskim jezikom. Pri razmatranju faza pribavljanja i izvršenja mašinske instrukcije, implicitno smo pretpostavili da je svaki kod operacije direktno implementiran u hardver. Mnogi računari stvarno tako i rade, ali znatan je

broj i onih koji, između mašinskog jezika i digitalnih kola, imaju jedan dodatni niži nivo nazvan mikroprogramski nivo ili nivo mikrokodiranja.

Osnovna ideja, koja se skriva iza mikrokodiranja, je u tome što se mnoge stvari koje se događaju na nivou mašinskih instrukcija često ponavljaju: kopiranje adrese iz **PC** u **MAR**, kopiranje podatka iz memorijske lokacije u **MBR**, izračunavanje adrese podatka, itd. Obzirom da se ove aktivnosti ponavljaju pri izvođenju svake elementarne instrukcije (mašinske instrukcije), to znači da su ove aktivnosti još jednostavnije od samih mašinskih instrukcija i da, u stvari, one predstavljaju elementarne operacije.

Na osnovu ovakvog pristupa možemo uobičajene mašinske naredbe, tretirati kao makroinstrukcije koje se izvršavaju kao nizovi malih koraka. Mikroinstrukcije koje implementiraju ove korake zapamćene su unutar CPU u specijalnoj memoriji koja se zove mikroprogramska upravljačka memorija. Kod nekih sistema, ova memorija je tipa ROM i naziva se firmver (**firmware**). Njen sadržaj popunjava proizvođač CPU-a. Kod nekih drugih sistema, korisniku se dopušta da menja sadržaj ove memorije, i da sam kreira mikroprograme prema svojim potrebama, tj. da vrši mikroprogramiranje.

Dužina reči procesora (npr. 8 bita kod mnogih mikroračunara) nema direktne veze sa dužinom reči mikroinstrukcije (npr, 18 bita, kod mikroprocesora Intel 8080, slika 4.33). Mikroinstrukcija, upravljačka reč, iz mikroprogramske memorije, preko kontrolne logike (dekodera mikroinstrukcija), generiše upravljačke signale koji se upućuju u nezavisne upravljačke tačke u svim delovima CPU i računara, koji omogućavaju izvođenje mikrooperacija.

a		**b**			**c**				**d**				**e**			**f**	
0	0	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0

Slika 4.33. Odnos mikroinstrukcije i mašinske instrukcije

Odnos (makro) instrukcija - mikroinstrukcija je prikazan preko primera izvođenja mašinske instrukcije **COMA** (komplementiraj sadržaj akumulatora) za procesor **Intel 8080**.

Izvođenje mašinske instrukcije **COMA** izaziva sledeće mikrooperacije, koje se izvršavaju u posebnim intervalima vremena:

1. takt: mikroinstrukcija za prenos sadržaja akumulatora preko interne magistrale u **ALU** u sklop za komplementiranje,

2. takt: mikroinstrukcija za aktiviranje logičke mreže za komplementiranje,

3. takt: mikroinstrukcija za prenos komplementiranog sadržaja preko interne magistrale u akumulator.

Iz ovog primera vidimo da se mikroprogram za mašinsku instrukciju **COMA** sastoji od tri 18-bitne mikroinstrukcije. Mikroinstrukcije za procesor **Intel 8080** imaju dužinu od 18 bita koji su grupisani u 6 polja (slika 4.33), koja imaju sledeće značenje:

- polje određuje internu mikrooperaciju u upravljačkoj jedinici,
- određuje logičku mrežu u aritmetičko-logičkoj jedinici (npr. 100-sklop za komplemetiranje, ili 111-ALU nije aktivan),
- određuje odredište podataka (npr. 0100 je akumulator , 0010 je sklop za komplementiranje),
- određuje registar ili deo ALU koji predstavlja izvor podataka,
- određuje informaciju o sledećoj adresi mikroinstrukcije (npr. 000 povećaj adresu mikroprograma za 1, tj. inkrementiraj mikroprogramski brojač **MPC** za jedan),
- određuje spoljašnji uslov koji učestvuje u generisanju sledeće adrese mikroinstrukcije (npr. 01 izbor bita prenosa).

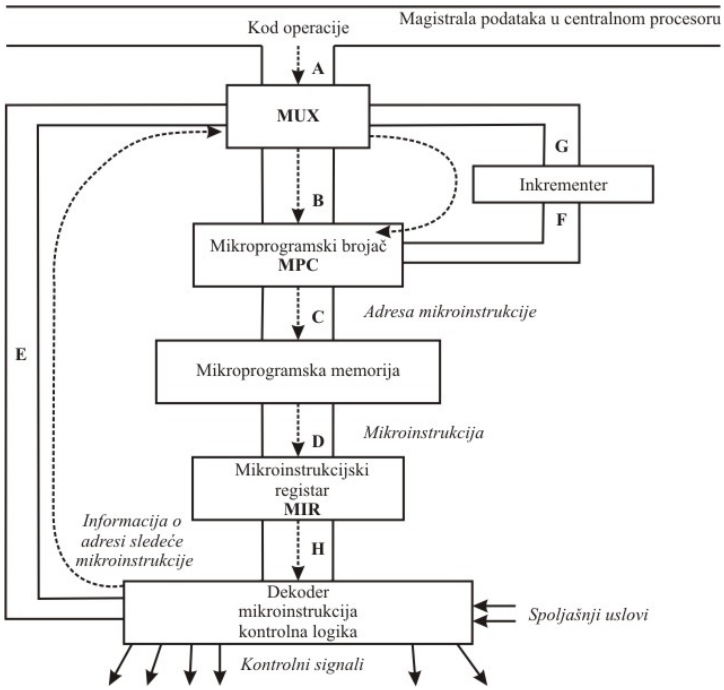
Mikroprogrami su smešteni u brzoj internoj memoriji (ROM, PLA), sa vremenom pristupa od nekoliko nanosekundi. **PLA** (**programable logic array**) je specijalna logička mreža, koja je realizovana u obliku čipa, a u koju se može upisati proizvoljna logička funkcija. U ovakvo polje korisnik može sam zapisati mikroprogram za svoj računar.

FAZA PRIBAVLJANJA MIKROINSTRUKCIJA

Pri izvođenju mikroinstrukcija postoji čitav niz koraka nalik na one pri izvođenju mašinskih instrukcija. Sekvenca počinje kada kod operacije mašinske instrukcije, iz registra naredbi (**IR**), stupi u deo upravljačke jedinice koji vrši dekodiranje koda operacije. Ovaj sklop se naziva generator adrese mikroprograma (slika 4.34).

Instrukcijski kod sadržan u instrukcijskom registru, uslovi (spoljašnji i unutrašnji-interni, kao rezultat izvođenja ranije mikroinstrukcije), kao i informacija o sledećoj adresi mikroprograma, generišu stvarnu adresu mikroprograma u bloku koji se naziva generator adrese mikroprograma (koji može biti poput multipleksera (kao na slici 4.34), ili specijalni sklop poznat pod imenom **microprogram sequencer** (kao na slici 4.35).

Mikroinstrukcija je reč zapisana na adresiranoj lokaciji mikroprogramske memorije.



Slika 4.34. Kod operacije mašinske instrukcije je početna adresa mikroprograma

Kod operacije može da se dekodira, najčešće tako, da odmah predstavlja početnu adresu niza mikroinstrukcija u upravljačkoj memoriji. U tu svrhu se može, umesto dekodera koda operacije, koristiti i multiplekser kojim se izabira jedna od nekoliko mogućih adresa.

Početna adresa, koja je određena na osnovu koda operacije mašinske naredbe, stupa u mikroprogramski brojač (**MPC**), koji ima istu namenu kao brojač naredbi (**PC**), ali istovremeno služi i kao memorijski adresni registar. Adresa sadržana u **MPC**, preko linija **B**, dolazi u mikroprogramsku memoriju iz koje se čita sadržaj i mikroinstrukcija se kopira u mikroinstrukcijski registar (**MIR**). I ovaj registar ima dvojaku ulogu, tj. ujedno predstavlja registar naredbi i memorijski bafer registar. Dvojaka uloga **MIR** i **MPC** je omogućena time što ova memorija sadrži samo instrukcije, pa nema potrebe da se razdvajaju podaci od naredbi kao u slučaju operativne memorije. Na kraju faze pribavljanja (dok se mikroinstrukcija izvršava),

treba generisati adresu sledeće mikro-instrukcije u mikroprogramskoj memoriji. To se može učiniti ili inkrementiranjem sadržaja **MPC,** ili na bazi nekog podatka sadržanog u tekućoj instrukciji ili na bazi spoljašnjih uslova koji se u generator adrese prenose takođe preko linija **E**.

Svaka mikroinstrukcija sadrži najmanje jedno polje za izračunavanje adrese sledeće instrukcije koja se vraća nazad u generator adrese pod kontrolom signala generisanih tokom faze izvršenja mikroinstrukcije.

FAZA IZVRŠENJA MIKROINSTRUKCIJE

Kada mikroinstrukcija dođe u **MIR**, počinje faza izvršenja. Kao i kod mašinskih naredbi, mikroinstrukcija se dekodira i digitalne logičke mreže generišu upravljačke signale. Pod kontrolom ovih impulsa rade svi sastavni delovi računara. Pomoću njih se selektuju registri u CPU koji učestvuju u prenosu podataka i adresa. Ako u CPU ima 16 registara, treba nam najmanje 4 linije za njihovo selektovanje. U neke od ovih registara su zapisane neke često korišćene konstante (npr. 1,0,-1). Neki upravljački signali se koriste za selektovanje pojedinih logičkih mreža u ALU koje obavljaju zasebne elementarne operacije. Ako ALU ima četiri funkcije treba nam dve linije za izbor elementarne operacije, a za ALU sa 8 funkcija tri linije.

Postoji još jedan način modifikovanja instrukcija, koji se takođe naziva mikroprogramiranje, mada to nije u punom smislu te reči. No, ako pod mikroprogramiranjem podrazumevamo mogućnost korisnika da sam kreira i modifikuje osnovne instrukcije, onda i ovo može biti mikroprogramiranje. Ovaj postupak se sastoji u spajanju dve ili više osnovnih instrukcija u jednu, pri čemu se ona i izvodi kao jedna instrukcija.

Neka recimo treba izvesti jednu za drugom dve instrukcije:

clear (brisanje sadržaja akumulatora)
clear C (brisanje bita prenosa, **carry**).

Ako se one kodiraju u binarnom obliku kao nizovi šesnaest **0** i **1**:

clear $0\ 111\ 010\ 000\ 000\ 000_{(2)} = 072000_{(8)}$
clear C $0\ 111\ 001\ 000\ 000\ 000_{(2)} = 071000_{(8)}$

Kodovi ovih instrukcija se razlikuju samo u dva bita, pa ako izvršimo logičko sabiranje (**ILI** operaciju), spajanjem ovih instrukcija dobijamo:

clear clear C $0\ 111\ 011\ 000\ 000\ 000_{(2)} = 073000_{(8)}$.

Pri izvođenju ove nove instrukcije izbriše se i akumulator i bit prenosa u samo jednom instrukcijskom ciklusu. Na ovaj način se i ubrzava rad, (štedi se vreme) i štedi se memorijski prostor.

4.6. HARDVERSKA ORGANIZACIJA UPRAVLJAČKIH JEDINICA

Kod **RISC** računara instrukcije su direktno ugrađene u hardver i samim tim mogu da se izvršavaju znatno brže, jer mažinske instrukcije ponekad zahtevaju izvođenje desetak ili čak stotinu mikroinstrukcija. Računari sa arhitekturom nazvanom **RISC** (**reduced instruction set computers**), generalno imaju manji broj instrukcija nego oni mikroprogramirani.

Druga razlika između ove dve koncepcije realizacije, odnosi se na implementaciju viših programskih jezika. Mikroprogramirani računari dopuštaju kreiranje mikroinstrukcija koje mogu direktno realizovati neku funkciju viših jezika. Kod ovih računara sam korisnik može programirati, tj. upisivati nizove mikroinstrukcija u mikroprogramsku memoriju. Na taj način sam korisnik dobija mogućnost da oblikuje specifičan skup instrukcija orijentisan ka odgovarajućoj primeni računara. Promena skupa instrukcija se izvodi vrlo jednostavno, zamenom mikroprogramske memorije, tj. **ROM**-a, pa se može uz jedan računar koristiti više setova instrukcija, zavisno od zadataka koje treba rešavati. Efikasnost obrade mnogo zavisi od skupa instrukcija. **RISC** računari, s druge strane, zahtevaju od jezičkih prevodilaca da generišu niz instrukcija u koje se ova funkcija implementira.
To znači da mikroprogramirani računari izvršavaju mnoge funkcije na nivou hardvera (ili **firmware**-a), dok **RISC** računari imaju rešenje u softveru. **RISC** računari imaju fiksnu dužinu instrukcija, pa je pribavljanje instrukcija brže. **RISC** računari imaju ograničen broj načina adresiranja, pa je i izračunavanje adrese brže, ali, programiranje je onda složenije.
Cena kojom se plaća ova štednja u broju naredbi i načina adresiranja ogleda se u povećanim problemima u pisanju i prevođenju programa sa viših jezika u mašinski. No, korisnike to ne treba previše da brine, jer su poslednjih godina napravljeni translatori (prevodioci) za mnoge više jezike.

4.7. ARHITEKTURA TEKUĆE TRAKE I PARALELNA OBRADA

Od prvog računara do današnjih dana, projektanti hardvera se trude da naprave što brže računarske sisteme. Jedan pristup rešenju tog problema je zasnovan na činjenici da se mnoge operacije mogu razdeliti na jednostavnije obrade-podoperacije, a zatim se nekoliko podoperacija može obavljati istovremeno.

Izvođenje svake instrukcije najgrublje se može podeliti u dve faze: pribavljanje i izvršavanje. Neki procesori (**Intel**) mogu za vreme izvršavanja prve instrukcije pribaviti drugu instrukciju. Za vreme izvršavanja druge se pribavi treća i tako redom. Ovaj postupak se zove **preklapanje faza**, a vreme potrebno za pribavljanje instrukcija praktično nestaje.

Osnovu za tehniku preklapanja faza predstavlja činjenica da se neke instrukcije u vreme faze izvršavanja ne obraćaju memoriji (na primer, instrukcije za **Intel** procesore: **DECA** dekrementiraj akumulator, **COMA** komplementiraj akumulator).

Takve instrukcije upotrebljavaju samo internu magistralu podataka, akumulator i aritmetičko-logičku jedinicu, dok su adresni registri (brojač naredbi, brojač podataka i sl.) i adresna magistrala slobodni. Kako adresni podsistem ne učestvuje u izvršavanju tekuće instrukcije, on se može upotrebiti za pribavljanje sledeće instrukcije. Procesori **Intel 8088** i familija procesora **8086** imaju dve nezavisne jedinice:

- jedinicu za izvođenje instrukcija i upravljanje **(execution unit, EU**)
- jedinicu za povezivanje sa magistralom (**bus interface unit, BIU**), koja u svom sastavu ima i skup registara zvani **red** (**queue**) koji u sebi sadrži nekoliko sledećih instrukcija programa.

Ali, moguće je još veće ubrzavanje rada procesora. Posmatrajmo opet niz mašinskih instrukcija kojima se rešava problem sabiranja dva broja:

sum = num1 + num2 .

ADRESA	SADRŽAJ	ZNAČENJE SADRŽAJA LOKACIJE
100:	**load data**	prenesi iz memorije u akumulator podatak
101:	**$1A_{(16)}$**	sa lokacije 1A
102:	**add data**	saberi sadržaj akumulatora sa podatkom
103:	**$1B_{(16)}$**	sa lokacije 1B
104:	**store data**	prenesi iz akumulatora podatak u memoriju
105:	**$1C_{(16)}$**	na lokaciju sa adresom 1C

Ako se pak izvođenje svake od instrukcija podeli na četiri koraka, potprocesa: pribavljanje koda operacije, dekodiranje, pribavljanje operanda, izvršavanje. Ako bilo koji potproces zahteva jedan vremenski interval, onda se ovaj mali niz obavlja u **12** taktova u standardnom izvođenju.

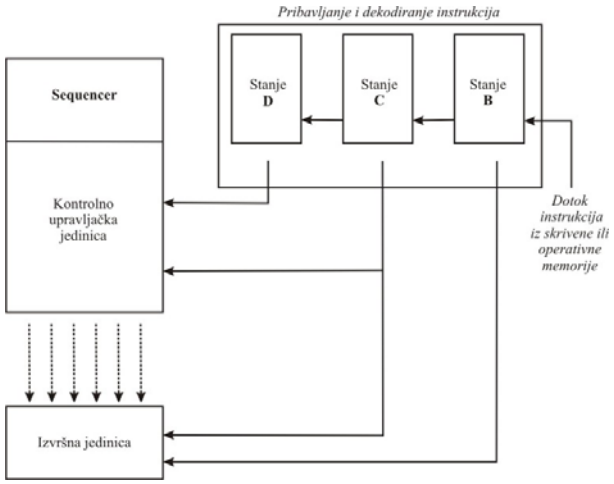
Ako se svaki od potprocesa može obaviti odvojeno, onda se ovaj niz instrukcija (program) može obaviti kao sledeći niz koraka:

1. Pribavljanje **load** instrukcije,
2. Dekodiranje **load** instrukcije,
   Pribavljanje **add** instrukcije.

3. Pribavljanje operanda za **load** instrukciju,
   Dekodiranje **add** instrukcije,
   Pribavljanje **store** instrukcije.
4. Kopiranje podatka za **load** instrukciju,
   Pribavljanje operanda za **add** instrukciju,
   Dekodiranje **store** instrukcije,
   Pribavljanje instrukcije na adresi **106**$_{(16)}$,
5. Sabiranje, tj. ivršavanje instrukcije **add**,
   Pribavljanje operanda za **store** instrukciju,
   Dekodiranje instrukcije na **106**$_{(16)}$,
   Pribavljanje instrukcije na **108**$_{(16)}$,
6. Smeštanje rezultata, tj. izvođenje instrukcije **store sum**.

. . .

Korišćenjem preklapanja nezavisnih obrada u potkoracima, potrebno ukupno vreme za izvođenje programa skraćuje se sa **12** na **6** intervala vremena, odnosno obrada je obavljena za polovinu vremena. Teorijski ova ušteda vremena može biti i veća, tj. može se svesti na četvrtinu prvobitnog vremena. Ovaj način obrade se zove arhitektura tekuće trake (**pipelining**). Pri ovom postupku proces obrade se deli na niz potprocesa koji se izvršavaju sa preklapanjem u autonomnim jedinicama za obradu.



Slika 4.35. Procesor **MC 68020** obrađuje podatke na tekućoj traci

Računar sa arhitekturom tekuće trake, koji deli proces na četiri autonomna potprocesa, može, teoretski, da četvorostruko poveća brzinu obrade. To je, međutim, nerealno očekivanje, pre svega zbog upotrebe zajedničkih magistrala i instrukcija skoka i grananja.

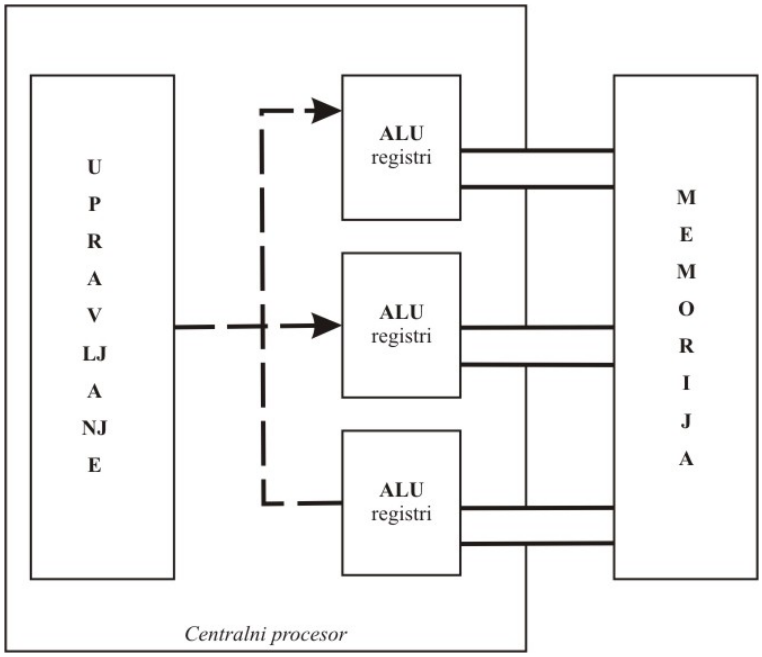
Na slici 4.35 prikazana je arhitektura tipa tekuće trake koju ima **Motorola MC 68020** procesor. Ovaj procesor (mikroprocesor) provodi instrukciju u toku izvršenja kroz tri stanja **B, C** i **D.** Stanje **D** daje upravljačkoj jedinici potpuno dekodiranu instrukciju. U toku izvršenja instrukcije, podatak iz stanja **C** može da posluži kao konstanta ili kao proširenje koda operacije.

Arhitekturu tekuće trake posedovao je i računar **CDC STAR-100**, koji je imao dva paralelna procesora za 64-bitne operande u pokretnom zarezu. Procesori su radili paralelno i obrađivali dva skupa 32-bitnih operanada, jer je svaki imao po dve linije tekuće trake koje su davale rezultat obrade svakih 40 nanosekundi. Ovaj sistem je dakle bio sposoban da obavi 10^8 32-bitnih operacija u pokretnom zarezu u sekundi (**100 megaflops, millions of floating-point operations per second**).

Ovaj računar je koristio memorijske reči od 512 bita. Memorija kapaciteta 4 megabajta, napravljena je od feritnih jezgara sa vremenom pristupa od 1.28 mikrosekundi, i bila je podeljena na 32 modula (kapaciteta 2048 reči) kojima se nezavisno pristupalo. Obzirom da je procesor svakih 40 ns obrađivao 128 bita, to je u jednom memorijskom ciklusu preko magistrale prenošeno 128·4·32=16384 bita.

4.8. MULTIPROCESORSKI SISTEMI

Jedan drugi pristup povećanja perfomansi podrazumeva korišćenje nekoliko procesora u isto vreme, ili paralelno. Računari sa paralelnim procesiranjem koriste od dve do nekoliko stotina jedinica za obradu koje rade jednovremeno.

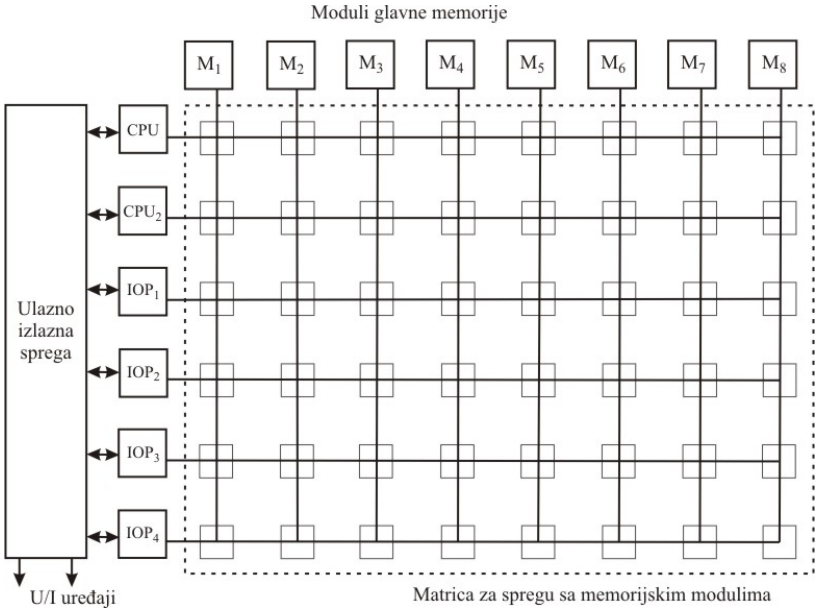


Slika 4.36. Jednostavna konfiguracija paralelnog procesora

Na slici 4.36 prikazana je jedna moguća konfiguracija paralelnog procesora sa tri ALU jedinice i sa tri skupa registra koji koriste zajedničku memoriju i upravljačku jedinicu. Ovakva konfiguracija poznata je pod imenom bit-odrezak (**bit-sliced**) arhitektura. Svaka ALU jedinica obrađuje po jedan deo podatka, pa se kombinovanjem nekoliko jedinica praktično može realizovati računar sa proizvoljnom dužinom reči. Ako svaka od ALU na slici 4.36 obrađuje po 4 bita, onda ovaj procesor ima reč od 12 bita. Na sličan način je realizovan računar **IBM 370**, **Honeywell Level 6**, **Univac 1100**, **CDC 6600** sa deset izvršnih jedinica kao i eksperimetalni računar **ILLIAC IV** (**Burroughs Corporation**). **ILLIAC IV** je imao jednu kontrolnu jedinicu i 64 procesne jedinice.

Svaka procesna jedinica sadrži sopstvenu procesnu memoriju i aritmetičko-logičku jedinicu koja je, pored uobičajenih operacija, obavljala i instrukcije nad 64-bitnim brojevima u pokretnom zarezu. Svaka procesna memorija je imala 2048 reči od 64 bita. Računarski sistemi mogu imati i više od jednog centralnog procesora. Postoje dve vrste ovakvih sistema: multiprocesorski sistemi i računarske mreže. Kad jedan računar sadrži dva ili više centralnih procesora, onda se on zove se multiprocesorski sistemi. Ovakvi sistemi su predodređeni za multiprocesiranje (istovremeno izvršavanje dve ili više instrukcija istog programa), ali i za multiprogramski rad(paralelno odvijanje više programa).

Na slici 4.37. prikazan je višeprocesorski sistem sa dva centralna procesora i sa četiri periferijska procesora, od kojih svaki ima određeni stepen autonomije. Primer na slici 4.37 predstavlja konfiguraciju računara **Burroughs B5000** i **B5500**.



Slika 4.37. Višeprocesorska konfiguracija računara **Burroughs B5000**

Glavna memorija je podeljena na osam delova, tj. modula, kojima se može nezavisno pristupati. Memorije su povezane sa procesorima pomoću sprežne mreže koja dopušta istovremeni pristup za svaki procesor ponaosob jednom (različitom) modulu memorije. Međutim, povećanje broja procesora ne dovodi obavezno do proporcionalnog povećanja efikasnosti obrade. To je pre svega posledica činjenice da algoritmi najčešće predviđaju obavljanje neke obrade korak po korak. Znatno veće povećanje efiksnosti postiže se kod takozvanih rekurzivnih postupak obrade. Poseban tip višeprocesorskog sistema čine sistemi sa distribuiranom obradom (prilog B), tj. računarske mreže, koje predstavljaju računarski sistem koji u svom radu objedinjuje više desetina, pa i više stotina nezavisnih računara (najčešće mikroračunara), koji su povezani komunikacionom mrežom. Ovakvi sistemi su pogodni za multiprogramski rad i paralelnu obradu.

4.9. ZAKLJUČAK

Nezavisno od toga kako su instrukcije implementirane, mnogi koraci zahtevaju izvođenje vrlo prostih operacija. Svaka instrukcija mora biti pribavljena iz memorije i dekodirana. Izvršenje ma koje instrukcije zahteva generisanje i vremensku kontrolu velikog broja signala koji upravljaju radom digitalnih logičkih kola koja čine hardver računara.

Izvođenje instrukcija često se može posmatrati na dva nivoa. Viši nivo, vidljiv za obične programere, je nivo uobičajenog mašinskog jezika. Mnogi računari (CISC) imaju i niži nivo, nivo mikrokodiranja, koji koristi nizove mikroinstrukcija (mikroprograme) za izvođenje svake mašinske naredbe. Druga vrsta procesora (RISC) ima za svaku mašinsku instrukciju zasebnu logičku mrežu koja omogućava izvršavanje instrukcije u jednom taktu, što znatno ubrzava rad računara.

Arhitektura tekuće trake i raznih vidova paralelne obrade, omogućavaju istovremeno izvođenje dve ili više instrukcija. Moderni računari su sposobni da izvršavaju širok spektar aritmetičkih i logičkih funkcija. Oni, takođe, koriste brojne tehnike za lociranje, tj. izračunavanje apsolutne adrese podataka i instrukcija u memoriji.

4.10. PITANJA

1. Koji su osnovni delovi generalisane arhitekture računara?
2. Koje korake treba izvršiti da bi se podaci izneli iz računara?
3. U čemu se razlikuje faza pribavljanja instrukcija od pribavljanja podatka?
4. Iz kojih delova se sastoji mašinska instrukcija (formati instrukcija)?
5. Kako se vrši pribavljanje mašinskih instrukcija?
6. Koja ograničenja i hardver omogućavaju smanjenje adresnosti naredbi?
7. Napisati program za sabiranje dva broja za jednoadresnu mašinu.
8. Opisati korake koji čine fazu pribavljanja mašinske instrukcije, i da li zavise od vrste instrukcije?.
9. Opisati korake koji čine fazu izvršavanja instrukcije **load**.
10. U čemu se razlikuju faze izvršavanja instrukcija load i **add**?
11. Opisati korake koji čine fazu izvršavanja instrukcije **store**.
12. Šta su mikrooperacije i mikroinstrukcije?
13. Opisati faze pribavljanja i izvršavanja mikroinstrukcije.
14. Šta su RISC procesori, koje su im prednosti i mane?
15. Kako se ubrzava rad računara, i koje vrste paralelne obrade postoje?
16. Šta predstavlja multiprocesiranje a šta multiprogramiranje?

4.11. KLJUČNE REČI

- adresna magistrala (**address bus**)
- akumulator (**accumulator**)
- arhitektura računara,
- faza pribavljanja (**fetch cycle**)
- faza izvođenja **(execution cycle**)
- firmver (**firmware**)
- glavna, operativna memorija (**main storage, operating memory**)
- instrukcija, naredba (**instruction**)
- kod operacije (**opcode**)
- magistrala podataka (**data bus**)
- memorijski adresni registar (**MAR**)
- memorijski bafer registar (**MBR**)
- mikroinstrukcija (**microinstruction**)
- mikroinstrukcijska memorija (**microinstruction storage**)
- mikroprogram (**microprogram**)
- operand (**operand**)
- paralelna obrada (**paralel processing**)
- programski brojač (**PC**)
- registar naredbi (**IR**),
- **RISC (reduced instruction set computer)**
- tekuća traka, cevovod, (**pipeline**)
- upravljačka jedinica (**control unit**)
- preklapanje faza
- distribuirana obrada (**distributed computing**)
- **CISC** (complete instruction set computers)

5. VRSTE NAREDBI I NAČINI ADRESIRANJA

Ako digitalne mreže posmatramo kao osnovne sastavne jedinice računarskog sistema, onda programe možemo posmatrati kao sredstvo koje te sastavne delove oživljava i povezuje u jednu funkcionalnu celinu. Računar izvršava programe tehnikom cikličnih ponavljanja faza pribavljanja i izvršenja instrukcija.

Centralni procesor je deo računarskog sistema čiji je jedan od osnovnih zadataka da pribavi i dekodira pribavljenu instrukciju, i generiše odgovarajući niz upravljačkih signala potrebnih za izvođenje te instrukcije. Ovi signali upravljaju prenosom podataka preko magistrala, nadziru rad aritmetičko-logičke jedinice, pobuđuju odgovarajuće registre i sastavne delove računara itd. Dakle, CPU generiše i raspoređuje takt i upravljačke signale kojima se usklađuju sve aktivnosti ne samo računara, već i svih drugih delova računarskog sistema.

Glavni sastavni delovi CPU-a su: registri, aritmetičko-logička jedinica i upravljačka jedinica koji su međusobno povezani pomoću magistrala.

Logičke mreže koje ulaze u sastav centralnog procesora određuju skup operacija koje mogu biti neposredno izvedene. Ove operacije se mogu grupisati po različitim kriterijumima, pa samim tim postoji više različitih podela za jedan isti skup instrukcija.

Drugu važnu stvar predstavljaju načini za određivanje apsolutnih adresa podataka i instrukcija u memoriji. Veliki broj načina adresiranja (**addressing modes**) je razvijen i ugrađen u razne računare. Tipovi načina adresiranja, koji su na raspolaganju za dati računar, direktno utiču na implementaciju funkcija viših programskih jezika u računar.

Digitalne mreže, koje su neophodne za implementaciju različitih naredbi i načina adresiranja, nalaze se u centralnoj procesorskoj jedinici (**CPU**). CPU je odgovorna za pribavljanje, dekodiranje i izvršenje instrukcija, kao i za usklađivanje prenosa podataka u i iz ulazno-izlaznih jedinica. Informacije koje ulaze u, i izlaze iz CPU-a, putuju duž magistrala. Mnogi računari imaju odvojene magistrale za sve glavne tipove informacija (za podatke, za adrese i upravljačke signale). Moguća su i

drugačija rešenja. Razmatranje sastavnih delova i funkcije centralnog procesora počećemo od registara koji ulaze u njegov sastav.

5.1. REGISTRI

U opisu faze pribavljanja instrukcija (i operanada) susreli smo se sa sledećim registrima:
- *programskim brojačem, brojačem naredbi (**PC**),*
- *memorijskim adresnim registrom (**MAR**),*
- *memorijskim bafer registrom (međuregistar za prihvat podataka **MBR**),*
- *registrom naredbi, instrukcijskim registrom (**IR**).*

Programski brojač (**PC**) sadrži adresu memorijske lokacije u kojoj je zapisana sledeća instrukcija koja će biti pribavljena u narednom ciklusu. Programski brojač se inkrementira tokom faze pribavljanja, ali i u toku pristupanja operandima za vreme faze izvršenja. Sadržaj ovog registra se, takođe, menja u toku izvršenja naredbi skoka i preskoka, zatim za vreme obrade prekida i pri pozivanju potprograma.

Memorijski adresni registar (**MAR**) sadrži adresu memorijske lokacije kojoj se pristupa. U računaru obično postoje dva ovakva registra: jedan u sastavu CPU, a drugi u sastavu memorije, a među sobom su povezani adresnom magistralom.

Brojač podataka je adresni registar podataka (**DC**), a sadrži u sebi adresu memorijske lokacije u kojoj se nalazi operand ili podatak.

Registar za prihvat podataka (**data buffer register**) služi za privremeno memorisanje informacija koje ulaze ili izlaze iz CPU. Registri slične namene nalazi se i u memoriji, (**MBR**), a i u svim ulazno-izlaznim sklopovima, i istovremeno služe za povezivanje svih sastavnih delova računara na magistralu podataka.

Registar naredbi (**IR**) prihvata kod operacije iz instrukcije koja je pribavljena, i koja će sledeća biti izvršena, tj. na osnovu koje će upravljačka jedinica generisati vremenske i upravljačke signale.
Većina aritmetičkih i logičkih operacija koristi registre poznate pod nazivom **akumulatori**. Neki procesori imaju jedan (**Intel** 8080 i nadalje), a neki više akumulatora (**Motorola** 6800 ima dva akumulatora, a **Unisys** 1100 ih ima 16). Oni imaju dvojaku ulogu, koriste se za privremeno prihvatanje jednog od ulaznih podataka u ALU a, takođe, i za prihvatanje rezultata obrade, odnosno izlaza iz ALU. Akumulatori imaju središnju ulogu u prenosu podataka u, i iz centralnog procesora. Kako se akumulatori nalaze u CPU (uz samu ALU, a po mišljenju mnogih autora akumulator je registar u sastavu ALU), to je vreme pristupa podacima u akumulatorima izuzetno kratko, pa više akumulatora omogućava veću brzinu obrade podataka. Kod računara organizovanih oko jedne magistrale, na jedan od ulaza ALU uvek se postavlja akumulator koji služi da zajedno sa privremenim registrima odvoji ulaz ALU od izlaza **ALU**. Ovo je potrebno da ne bi

došlo do neželjenog uticaja izlaza **ALU** na ulaz (**critical race**), jer su i ulaz i izlaz **ALU** na istoj magistrali.

BROJAČI I POMERAČKI REGISTRI

Brojači (**counters**) su digitalne mreže čiji se sadržaj, pod dejstvom impulsa, može menjati u određenom redosledu, tako da se može interpretirati kao niz sukcesivnih brojeva. Kako niz brojeva može monotono da raste ili opada, to i brojači mogu da broje unapred i unazad (inkrementiranje, dekrementiranje). Najčešće se koriste brojači koji broje u binarnom i dekadnom brojnom sistemu.

Pomerački registar (**shifter**) je skup memorijskih kola koja su povezana tako da memorisani podatak može da se pomera od jednog do drugog memorijskog kola. Pomeranje se može vršiti od bita manje težine na bit veće težine (**ulevo**) ili sa bita veće težine na bit manje težine (udesno).

POKAZIVAČ STEKA

Pokazivač steka (**stack pointer**) sadrži adresu dela memorijskog prostora poznatog pod imenom stek, a koji se koristi za privremeno smeštanje podataka ili adresa. Najčešće su to sadržaji nekih registara iz **CPU**. Sadržaj ovog registra se automatski uvećava i umanjuje, tj. inkrementira i dekrementira pri izvođenju nekih operacija sa stek memorijom. Stek memorija i ovaj registar koriste se prilikom obrade prekida, kod operacija sa potprogramima i za čuvanje specijalne grupe podataka takvih da se prvo koristi poslednji upisani podatak (**LIFO - Last In First Out**).

INDEKS REGISTRI

Indeks registri se najčešće koriste pri izračunavanju fizičke adrese memorijskih lokacija, kada se sadržaju nekog od ovih registara dodaje sadržaj iz polja operanda u instrukciji. Ovakav pristup memoriji se zove indeksno adresiranje. Ovi registri se često koriste za kontrolu programskih petlji. U ove registre se mogu upisivati podaci, a takođe se njihov sadržaj može inkrementirati i dekrementirati.

BAZNI I SEGMENTNI REGISTRI

Neki računari imaju specijalne bazne registre, dok računari zasnovani na intelovim mikroprocesorima (**Intel** serija **8086** i nadalje), imaju niz registara koji se zovu segmentni registri. I bazni i segmentni registri se koriste pri izračunavanju memorijskih adresa instrukcija i podataka. Brojač naredbi, brojač podataka, pokazivač steka, indeks registri, bazni i segmentni registri čine grupu adresnih registara, tj. služe za izračunavanje adresa.

REGISTRI OPŠTE NAMENE

Procesori **Intel 8086** i nadalje imaju grupu od četiri registra za podatke (AX, BX, CX, DX koji se sastoje iz dva dela: gornjeg H i donjeg L), slika 5.1, i svaki od njih se može upotrebiti pri izvršavanju aritmetičkih i logičkih operacija. **IBM 370** ima 16 registara opšte namene. Oni se mogu koristiti i kao akumulatori, i kao indeks registri, ili kao bazni registri. Računari **Unisys 1100** imaju četiri registra koji se mogu koristiti kao indeks registri ili kao akumulatori.

Višenamenski registri omogućavaju programeru veliku fleksibilnost u rešavanju problema. U njima se privremeno zapisuju različiti podaci, a pristupa im se znatno brže i jednostavnije nego memoriji.

Ovi registri znatno ubrzavaju rad računara: **Prvo**, vreme pristupa tih registara je izuzetno kratko, i iznosi desetak nanosekundi, pa čak i manje. **Drugo**, instrukcije su kraće, a kod organizacije oko magistrale, adresa registra je deo proširenog koda operacije, tj. nema polja operanda.

No, ponekad se ovi registri organizuju kao mala memorija, i onda se ona zove **scratchpad store** (pribeleška), memorija za privremeno čuvanje. U ovakvoj organizaciji, registar se bira pomoću adrese, ali su i ove instrukcije kraće od memorijskih.

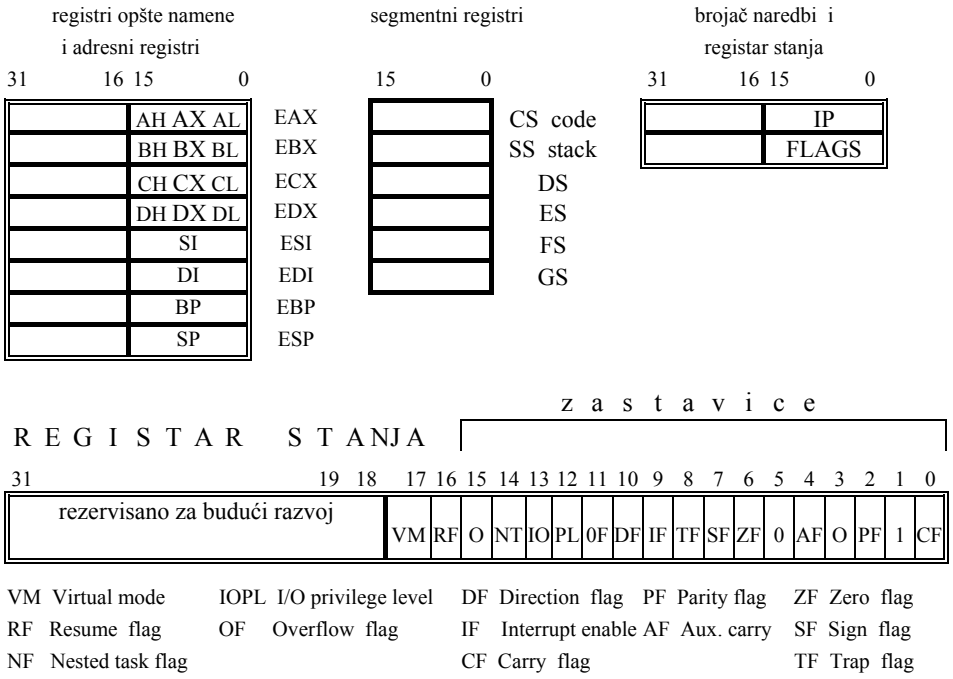
REGISTRI STANJA PROCESORA I REGISTRI STANJA PROGRAMA

Registar stawa (**SR, Status Register**) ili registar kodova uslova (**Condition Code, CCR**) je poseban registar kod koga pojedinačni bitovi prikazuju različita stanjlja koja mogu nastati u toku obrade podataka. Ovi bitovi se nazivaju zastavice (**flags**).

Sve operacije koje koriste **ALU** mogu postavljati neke od zastavica da bi označile da je izračunavanje dalo, na primer nulu ili negativan rezultat. Neke druge zastavice mogu ukazivati da se pojavio jedan bit viška na kraju registra (prenos, **carry**), ili da je rezultat suviše veliki da bi stao u registar (prepunjenje, **overflow**), ili da je pak rezultat (broj u pokretnom zarezu) suviše mali (potkoračenje, **underflow**).

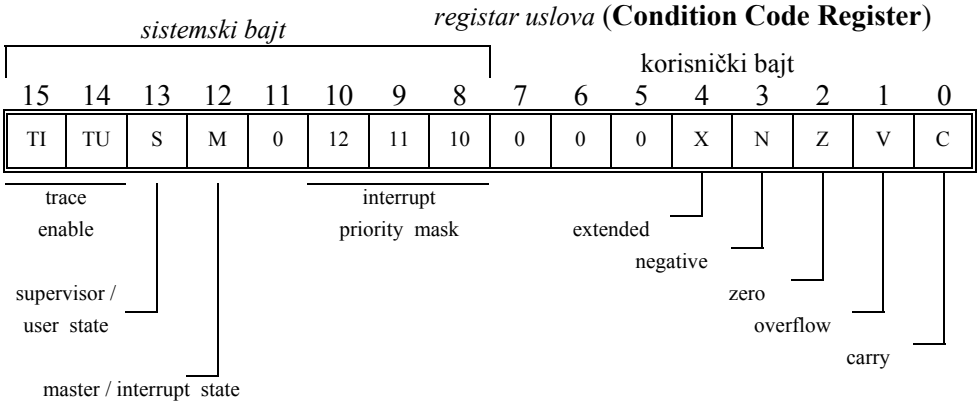
Ta stanja mogu uticati na dalje odvijanje programa, ili rada računara, jer programer može proverom stanja određene zastavice da usmeri odvijanje programa, i promeni redosled izvršavanja instrukcija.

Ne postoje standardi šta sve i u kom obliku sadrži registar koji opisuje stanje procesora i programa. Slika 5.1. pokazuje registre opšte namene i adresne registre, segmentne registre i registar stanja (registar zastavica, **flag register**) u **Intel 80386** mikroprocesoru.



Slika 5.1. Registri opšte namene i registar stanja procesora Intel **80386**

Na slici 5.2. prikazan je registar stanja za **Motorola MC 68020** mikroprocesor..



Slika 5.2. Polja u registru stanja mikroprocesora **Motorola MC68020**

Ovaj registar je podeljen na korisnički i sistemski bajt. Kod nekih računara se postojanje uslova prikazuje u kodiranom obliku (kao kod **IBM 360/ 370**), a kod nekih je za svaki uslov rezervisana posebna zastavica. Pomenućemo neke od zastavica na osnovu čijeg sadržaja se najčešće kontroliše tok programa, tj. redosled izvođenja instrukcija.

Zastavica **C** (**carry**) koja označava prenos, element je registra stanja (ili registra uslova, **condition code register, CCR**), i ima dve funkcije:

- *sadrži bit prenosa nakon izvođenja aritmetičkih operacija (prenos iz bita najveće težine, **MSB**),*
- *zastavica **C** se upotrebljava kao pomoćni bit pri operacijama pomeranja, ili rotiranja sadržaja akumulatora.*

Zastavica **V** (**overflow**) je bit koji označava da je došlo do prekoračenja vrednosti (ako se radi o 8-bitnom procesoru **V = 1,** označava da je rezultat operacije po apsolutnoj vrednosti prekoračio granicu od 127).

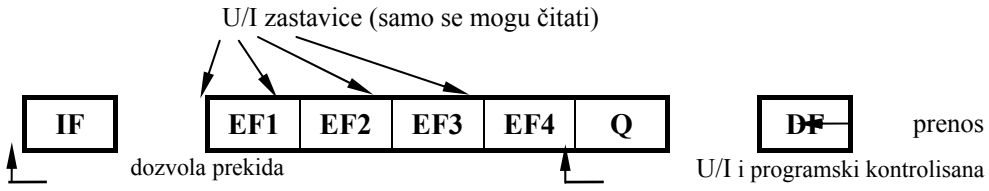
Zastavica **N** (**negative**) se upotrebljava za indikaciju negativnog rezultata nakon izvršenja aritmetičke operacije, i još nekih instrukcija.

Zastavica **Z** (**zero**) je bit koji javlja da je rezultat aritmetičke operacije **0**. Takođe se upotrebljava i za logičke operacije, npr. upoređivanje, gde se, ukoliko su dva operanda jednaka, postavlja zastavica (**Z = 1**).

Zastavica **H** (**nibble**) označava prenos iz donjeg polubajta u gornji polubajt (tetrada), i upotrebljava se u **BCD** operacijama, obzirom da se u kodu **BCD** upotrebljavaju četiri bita za prikaz jedne dekadne cifre.

Zastavica **P** (**parity**) je bit parnosti i nalazi se u registru uslova kod procesora **Intel 8080** za ispitivanje tačnosti prenosa podataka. Većina procesora nema bit **P** u registru stanja, pošto je funkcija generisanja i ispitivanja parnosti preneta u nadležnost programabilne ulazno-izlazne jedinice ili ulazno-izlaznog kanala.

Zastavica **I** (**interrupt enable**) je bit prekida i obično se upotrebljava u specijalnim programima za obradu iznenadnih situacija koje su nastale u računarskom sistemu ili van njega. Bit **I** se postavlja u slučaju zabrane nekih vrsta prekida (**I=1**). Zastavica **I** se kod procesora naziva prekidna maska, jer se u slučaju da je **I=1**, i ako se pojavi zahtev za prekid nižeg prioriteta, prekid tekućeg programa se neće dogoditi.



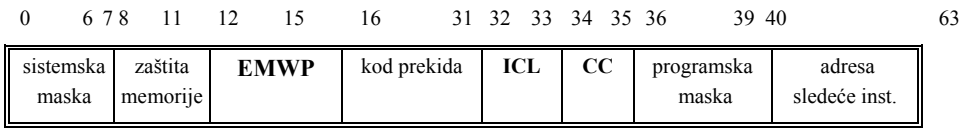
Slika 5.3. Reč stanja procesora za mikroprocessor **CDP 1802**

Na slici 5.3. prikazan je sličan registar koji se nalazio u **COSMAC CDP 1802** mikroprocesoru. On nema klasičan registar stanja već ima sedam zastavica stanja specifične namene: **DF** je zastavica prenosa. **IE** je zastavica za dozvolu ili maskiranje prekida, zastavica **Q** koju postavljaju spoljašnja logička kola (ali je menjaju i neke instrukcije) i takozvane **U/I** zastavice **EF1, EF2, EF3, EF4** koje isključivo postavljaju spoljašnja logička kola i mogu se samo čitati. Poslednjih pet zastavica se koristi za realizaciju instrukcija uslovnog skoka tj. grananja.

Računari sa procesorom **MC 68020**, kao i većina modernih računara, sadrže u memoriji, u isto vreme, deo operativnog sistema i jedan ili više programa. Centralni procesor (**CPU)** i operativni sistem, moraju uvek znati gde se svaki od tekućih korisničkih programa nalazi, gde se nalaze podaci, i koja instrukcija se upravo izvršava.

CPU mora, takođe, sačuvati trag o specijalnim uslovima koji mogu uticati na tekuću ili buduću instrukciju. Ove informacije se, kod nekih procesora, pamte u registru stanja programa (**Program Status Word, PSW**), ili u registru stanja procesora (**Processor Status Register, PSR**), koji su sasvim slični registru stanja (**Status Register, SR**).

No, kao što smo već napomenuli, mnogi računari ne postavljaju direktno zastavice, već kodiraju stanje procesora i programa. U tom slučaju ti kodovi moraju sadržati i informacije koje nam daju pomenute zastavice. Na slici 5.4 prikazana je dupla reč stanja programa (**PSW** od 64 bita), za računare IBM 360 / 370 serije.



Slika 5.4 Registar stanja programa za računare **IBM** serije 360 / 370

Bitovi od 0 do 7 su sistemska maska za kontrolu raznih **I/O** prekida, i označavaju da li centralni procesor može prihvatiti prekid sa nekog kanala. Bitovi 0 do 5, ako su uključeni (postavljeni na 1), označavaju da je sa kanala 0 do 5 dopušten prekid.

Bit 7, ako je uključen, označava da su dopušteni spoljni prekidi. Ako je uklju~en bit 6, svi kanali mogu signalizirati prekide. Ako su ulazno-izlazni prekidi maskirani, hardver automatski zadržava prekid za kasnije, kada se maska skine i prekidi dopuste.

Bitovi od 8 do 11 predstavljaju ključ za zaštitu memorije, i koriste se za kontrolu pristupa raznim delovima memorije. Polje je dugačko 4 bita. U to polje upisana je binarna šifra (broj) koji se mora prilikom svakog pristupa u memoriju uporediti s binarnim brojem koji u memoriji postoji. Polje zaštitnog ključa služi za zaštitu pojedinih delova operativne memorije od nedopuštenog pristupa.

Naime, kod ovog sistema memorija je podeljena u blokove od 2048 bajta i svaki takav blok ima pridruženu šifru dužine 4 bita. Ako se, dakle, ključ u **PSW**-u i ključ u memoriji podudaraju, korisnikovom programu je dozvoljen pristup u taj deo memorije.

Pristup se, takođe, može selektivno regulisati, tako da je u neke blokove memorije moguće upisivati podatke i čitati ih, dok za druge blokove može biti dopušteno samo čitanje sadržaja.

Bit 12 (**E**) kod računara IBM 360/370 označava da je računar u proširenom režimu rada (**extended mode**). Bit **M** (13) i bitovi za programsku masku (36-39), pokazuju vrstu grešaka koje mogu da budu zanemarene. Bit **W** označava da je procesor u stanju izvođenja (kada je **W=0**), ili u stanju zastoja (**W=1**).

Bit **P** označava da li je procesor u stanju izvođenja problemskog programa (**P=0**), ili izvodi neki od programa iz sastava operativnog sistema (**P=1**), odnosno bit **15** ukazuje da je računar u korisničkom režimu (**problem state**), ili u privilegovanom-supervizor režimu rada (**supervisor mode**). Bitovi 32, 33 (**ILC**) zovu se kod dužine instrukcije, i pokazuju veličinu (dužinu) poslednje mašinske naredbe izraženu u polurečima (16 bita = 2 bajta = 1 polureč).

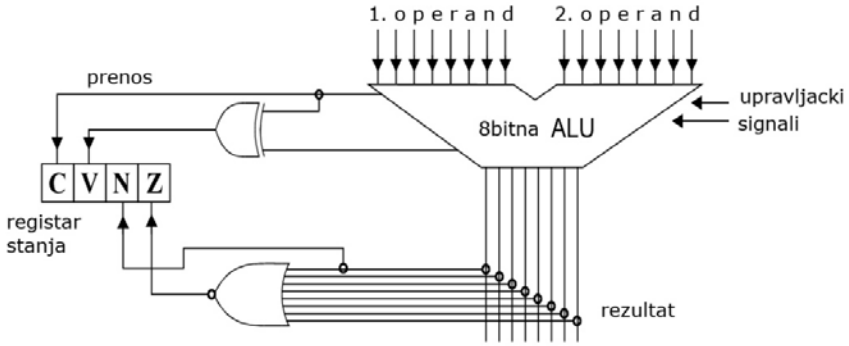
Polje 34, 35 (**CC**) je tzv. kod uslova, ili indikator stanja nakon izvođenja nekih instrukcija, i ukazuje da li je rezultat prethodne aritmetičke operacije bio jednak nuli, negativan, pozitivan ili suviše veliki. Kod računara IBM, iz serije 360/370, ne postavljaju se direktno zastavice za svaki uslov posebno, već se pojava određenih uslova zbirno kodira.

PSW registar se nalazi u centralnom procesoru i služi za kontrolu redosleda izvođenja instrukcija i kompletnu indikaciju stanja sistema u odnosu na program koji se trenutno izvodi. Naime, bitovi od 40 do 63 sadrže adresu sledeće instrukcije koja treba da se izvrši, odnosno imaju ulogu brojača naredbi.

Taj deo **PSW** registra kod **IBM** računara ima istu ulogu kao brojač naredbi **(Program Counter, PC**), ili instrukcijski adresni registar (**IAS**) kod nekih drugih računara.

5.2. ARITMETIČKO - LOGIČKA JEDINICA

Aritmetičko - logička jedinica je višefunkcijski digitalni sklop. **ALU** je centralna komponenta u fazi izvođenja instrukcija u računaru. Obično ima dva ulaza na koje se dovode operandi **A** i **B**, i jedan izlaz gde se dobija rezultat (slika 5.5).



Slika 5.5 Aritmetičko-logička jedinica **ALU**

Posmatrana na nivou mašinskog jezika, **ALU** obavlja mnoštvo funkcija pod kontrolom niza upravljačkih signala. Sem neposrednog izračunavanja rezultata, **ALU** na izlazu generiše i informacije o stanjima koja se smeštaju u registar stanja. Neke operacije sem operanada zahtevaju, kao ulazni podatak, i bit prenosa (tipično za obradu višecifarskih dekadnih i višerečnih binarnih brojeva).

Aritmetičko-logička jedinica u svom sastavu ima digitalna kola za: komplementiranje, pomeranje, inkrementiranje, dekrementiranje, sabiranje, itd. **ALU** obavlja aritmetičke i logičke operacije nad binarnim brojevima, a takođe i njihovo rotiranje (pomeranje). Broj operacija koje se neposredno izvode (tj. za koje postoje posebna digitalna kola) nije veliki.

Neki autori u sastav **ALU** uključuju i akumulator i registar stanja, zbog njihove funkcionalne povezanosti. Oni u sastav aritmetičko-logičke jedinice uključuju i registar podataka (**Data Register, DR**), kao i privremene registre.Sem u instrukcijama koje u sebi sadrže neku aritmetičku ili logičku operaciju, aritmetičko-logička jedinica se koristi i pri različitim postupcima izračunavanja adresa podataka i instrukcija.U narednom delu razmotrićemo instrukcije koje su tipične za većinu računara, kao i najčešće načine adresiranja.

5.3. TIPOVI MAŠINSKIH INSTRUKCIJA

Skup mašinskih instrukcija definiše osnovne operacije koje računar može da izvede. Instrukcije se mogu klasifikovati po različitim kriterijumima, a jedna od mogućih podela je bazirana na tipu operacije koja se izvodi primenom date instrukcije.

Tako se instrukcije računara mogu svrstati u tri grupe:

1. *aritmetičko-logičke instrukcije,*
2. *instrukcije za prenos podataka,*
3. *instrukcije za kontrolu toka programa, tj. upravljačke instrukcije.*

ARITMETIČKE I LOGIČKE INSTRUKCIJE

Ove instrukcije se dele u dve grupe, zavisno od toga koliko operanada sadrže:

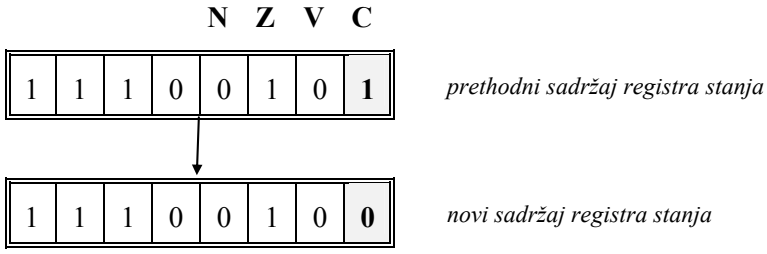
1. *unarne, sa jednim operandom (**monadic instructions**),*
2. *binarne, sa dva operanda (**diadic instructions**).*

Unarne instrukcije

Tipične unarne instrukcije su: postavljanje (**set**), brisanje (**clear**), komplementiranje (**complement**), negacija (**negate**), inkrementiranje (**increment**), dekrementiranje (**decrement**), pomeranje (**shift**) i rotacija (**rotate**).

Set i **Clear** instrukcije se koriste za promenu, tj. za postavljanje željenog sadržaja pojedinih bitova u registar stanja, čitavih registara u **CPU** i memorijskih lokacija. Procesor može imati instrukciju za brisanje sadržaja čitavog registra (sadržaj nula), ili će vršiti brisanje (**reset**) pojedinih zastavica: **clear carry**, **clear overflow** itd. Normalno, na raspolaganju su i instrukcije za postavljanje jedinice u pojedine zastavice: **set carry**, **set decimal mode**, **set interrupt** itd.

Na slici 5.6. je prikazan efekat instrukcije **CLC** za procesor **Motorola MC 6800** (**clear carry**) na promenu sadržaja dela registra stanja, tj. na zastavicu za prenos.

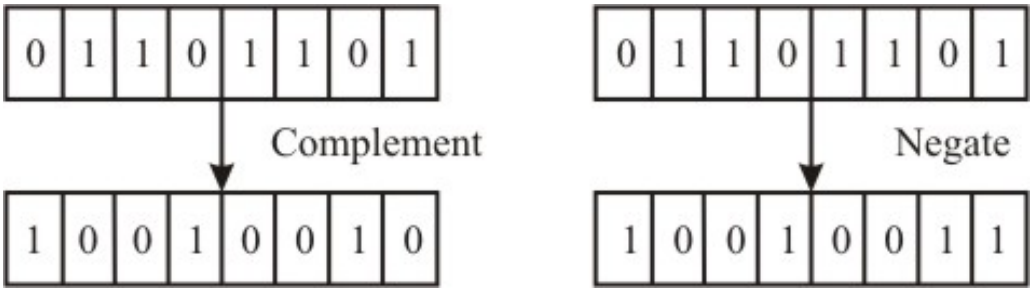


Slika 5.6. Uticaj instrukcije **clear carry** na registar stanja

Uočimo da je ova instrukcija uticala samo na sadržaj bita za prenos, dok su sadržaji ostalih zastavica ostali nepromenjeni.

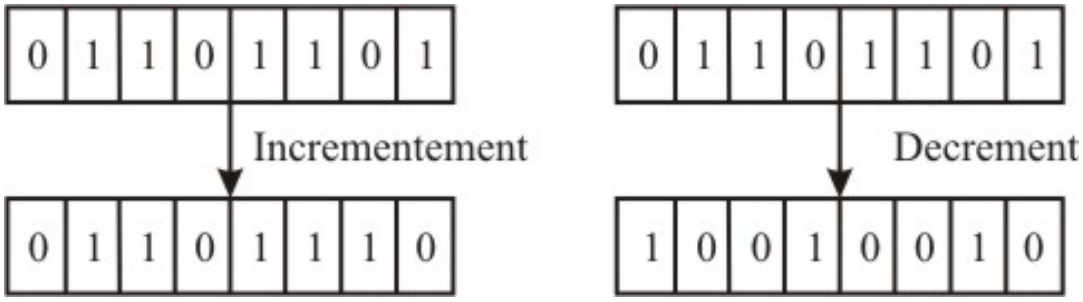
Napomenimo da neki mikroprocesori automatski vrše sabiranje sa učešćem bita za prenos. Da ne bi slučajni stari sadržaj ovog bita izazvao pojavu netačnog rezultata, to ova instrukcija, po pravilu, prethodi instrukciji za sabiranje.

Komplementiranje i negacija su instrukcije koje se najčešće odnose na sadržaj celog registra, a sličnog su dejstva. Naime, procesor **Intel 8080** nema instrukciju tipa **clear carry** već se brisanje zastavice za prenos vrši pomoću instrukcije **CMC - complement carry**. Komplementiranje registra menja sve nula bitove u jedinice, a jedinice u nule, a kao rezultat se dobija prvi komplement (inverzni kod) broja. Računarski sistemi koji rade u drugom komplementu imaju i instrukciju negacije, koja kao rezultat daje drugi komplement broja (slika 5.7.).



Slika 5.7. Instrukcije komplementiranja i negacije u računaru sa drugim komplementom

Inkrementiranje i dekrementiranje su instrukcije za povećanje i smanjenje njihovog operanda za jedan (slika 5.8.). U nekim računarima, ove instrukcije se primenjuju samo na registre koji se nalaze u **CPU**, dok se kod drugih mogu primeniti i na sadržaje memorijskih lokacija.



Slika 5.8. Instrukcije povećanja i smanjenja sadržaja registra za jedan

Pomeranje i rotiranje su instrukcije koje pomeraju bitove unutar jednog registra, u jednom ili drugom smeru. Mikroprocesori obično pomeraju sadržaj registra za jednu poziciju u jednom taktu, dok veliki računari mogu u jednom taktu pomeriti sadržaj za više pozicija. Ove operacije se vrše najčešće nad sadržajem akumulatora.

Pretpostavimo da akumulator sadrži sledeće bitove:

(10011001)

Pomeranjem sadržaja ovog registra za jedno mesto levo, dobija se sledeći rezultat:

1 (0011001**?**).

Uočimo da je krajnje levi bit (**MSB**) izvan registra, i da je krajnje desni bit (**LSB**) nedefinisane vrednosti.

Mnogi računari popunjavaju ispražnjene pozicije nulama. Bit koji na levom kraju izlazi, biće za neke računare izgubljen, a kod drugih će biti prihvaćen kao bit prenosa.

Posmatrajmo šta se događa pri pomeranju sadržaja registra za jedno mesto udesno. Neka je početni sadržaj akumulatora (01110110), posle pomeranja udesno biće

(**?**0111011)**0**. **LSB** će biti ili izgubljen, ili će se upisati u zastavicu prenosa (u ovom primeru zastavica prenosa je **0**). Kod pomeranja udesno ostaje nedefinisan krajnje levi bit (**MSB**). Jedno od rešenja je da se uvek u ovaj bit upisuje nula, pa bi onda novi sadržaj akumulatora bio (**0**0111011). Ovakav pomeraj, kada se u **MSB** upisuje nula, naziva se logički pomeraj udesno.

Pomeranje sadržaja registra ima niz značajnih primena. Prva je ispitivanje sadržaja bita u registru. Ako nam treba da saznamo šta se nalazi u trećem bitu sa leve strane, jednostavno pomerimo sadržaj registra za tri mesta ulevo, i bit čiji sadr`aj istražujemo biće smešten u zastavicu prenosa, a potom se ispita sadržaj ove zastavice, (slika 5.9).

prenos	registar	
(**?**)	(10101101)	osnovni sadržaj
(**1**)	(01011010)	prvi pomeraj
(**0**)	(10110100)	drugi pomeraj
(**1**)	(01101000)	treći pomeraj

Slika 5.9 Korišćenje pomeranja za ispitivanje sadržaja bita

Druga primena instrukcije pomeranja je za množenje i deljenje broja sa brojem 2^n, slika 5.10. Pomeranje sadržaja registra za jedno mesto ulevo ima isto svojstvo kao množenje sadržaja registra sa 2 (2^1). Pomeranje za dva mesta ulevo množi osnovni broj sa 4 (2^2), a pomeranje za **n** mesta ulevo množi sadržaj registra sa 2^n.

Pomeranje udesno ima suprotan efekat, tj. odgovara deljenju sadržaja registra sa 2 ili 2^n, gde je **n** broj mesta za koliko se pomera sadržaj. Ovo je celobrojno deljenje, bez ostatka.

Tačnije, ostatak nije sačuvan u registru, ali je informacija o postojanju ostatka sačuvana u zastavici za prenos.

(00000101) = 5		(00010100) = 20
(00001010) = 10	pomeranje za 1 mesto	(00001010) = 10 (20:2)
(00010100) = 20	pomeranje za 2 mesta	(00000101) = 5 (20:4)
(00101000) = 40	pomeranje za 3 mesta	(00000010) = 2 (20:8)
(a) pomeranje ulevo		(b) pomeranje udesno

Slika 5.10 Množenje i deljenje sa 2^n pomoću pomeranja binarnog broja za **n** mesta ulevo i udesno

No, posmatrajmo šta bi se desilo kad bi postupak deljenja sa 2 pomoću pomeranja udesno, primenili na broj (11110011). Posle pomeranja udesno na opisani način, u **MSB** se upisuje nula, dobićemo broj (01111001).

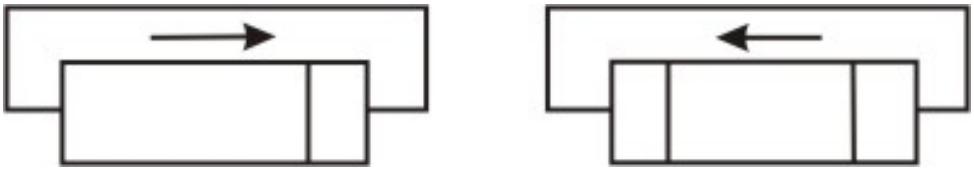
Ako je ovo računar koji radi u prvom komplementu, onda je početni broj bio negativan, tačnije to je $-12_{(10)}$. Deljenjem sa 2 trebalo bi da dobijemo broj $-6_{(10)}$. Međutim, nakon pomeranja dobijen je broj koji je pozitivan i jednak je $121_{(10)}$.

Da bi prevazišli ovaj problem, mnogi računari imaju i drugu vrstu pomeraja udesno, takozvani aritmetički pomeraj. Kod ovog pomeraja u **MSB** (bit znaka), pri pomeranju udesno, upisuje se njegova prethodna vrednost, tako da se zadržava znak broja. Ako je osnovni broj bio pozitivan, onda je **MSB** nula, i nakon pomeranja udesno, opet se upisuje nula.

Ako je broj bio negativan, pri pomeranju udesno, u krajnje levi bit treba upisati jedinicu, tj. i rezultat će biti negativan. Ako primenimo ovo pravilo, nakon pomeranja registra (11110011) udesno, dobijamo (11111001), što je u prvom komplementu broj $-6_{(10)}$, pa je rezultat tačan.

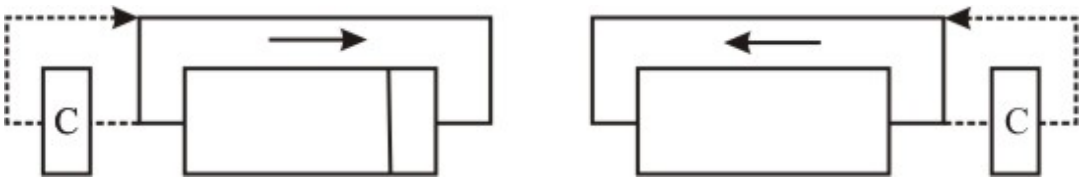
Neki procesori imaju i aritmetički i logički pomeraj. Takvi su i **Motorola MC 68000**, **Intel 8086** (i kasniji tipovi ovih familija procesora). **VAX** računari imaju pak samo aritmetičke pomeraje, a smer pomeranja zavisi od znaka (pomeranje udesno za šest mesta označava se sa **-6**, a ulevo sa **+6**). Mikroprocesor **RCA COSMAC 1802** ima samo logički pomeraj.

Rotiranje binarnog broja je slično pomeranju, samo se bit koji napušta registar upisuje na mesto ispražnjenog (nedefinisanog) bita, na drugom kraju registra, slika 5.11.



Slika 5.11 Tipična rotacija sadržaja registara

Rotiranje sadržaja registra (11001100), za jedno mesto ulevo, daje rezultat (10011001), (jedinica sa levog kraja upisana je na desnom kraju). Rotiranje sadržaja registra (00001111) za jedno mesto udesno, daje rezultat (10000111). I rotiranje se, kao i pomeranje, koristi za ispitivanje sadržaja nekog bita. Tada se bit koji napušta registar upisuje u zastavicu prenosa i ujedno vraća na drugi kraj registra, kao na slici 5.12. Kod nekih procesora (**Motorola MC6800**) i sam bit prenosa učestvuje u rotaciji (isprekidane linije na slici 5.12.).



Slika 5.12. Rotacija sadržaja registra sa pamćenjem prenosa

Kod nekih procesora se u procesu pomeranja i rotacije umesto zastavice za prenos, koristi takozvani **link** registar. **Link** je jednobitni registar i ima istu funkciju kao zastavica za prenos.

Binarne instrukcije

Binarne instrukcije imaju dva operanda, od kojih je najčešće bar jedan u nekom od akumulatora (ili drugim registrima u **CPU**), dok je drugi u nekom od registara memorije. Ove operacije su znatno brže i kompaktnije, ako su oba operanda u registrima unutar centralnog procesora. Operacije koje se vrše nad operandima mogu biti aritmetičke ili logičke, pa postoje i dve grupe ovih instrukcija.

5.4. BINARNE LOGIČKE INSTRUKCIJE

Ove instrukcije izvršavaju familiju operacija I, ILI, i ekskluzivno ILI (**AND, OR, XOR**). Logičke instrukcije obrađuju sve bitove u registru, ali se primenjuju na svaki par bita ponaosob, i za svaki par generišu poseban rezultat, koji ne utiče na rezultat u drugim bitima - ovde nema prenosa između dva susedna razreda u registru. Ako registar **A** sadrži niz $11010011_{(2)}$, a registar **B** niz $01001110_{(2)}$, onda su rezultati logičkih operacija i operacije sabiranja dati na slici 5.13.

	A AND B	**A OR B**	**A XOR B**	**A + B**
A	**11010011**	**11010011**	**11010011**	**11010011**
B	**01001110**	**01001110**	**01001110**	**01001110**
prenos			**11 011110**	
rezultat	**01000010**	**11011111**	**1 0011101**	**100100001**

Slika 5.13 Odnos logičkih operacija nad sadržajem registra i sabiranja binarnih brojeva

Rezultat instrukcije **AND** ima jedinice samo u onim bitima gde su oba bita i u **A** i u **B** jedinica. Instrukcija **OR** daje u rezultatu jedinicu, u svim bitima gde je jedinica bilo u **A** bilo u **B**, a **XOR** daje u rezultatu jedinicu, ako je taj bit jedinica, samo u **A** ili samo u **B**. Uočimo da ni jedan rezultat logičkih operacija nije ni nalik na zbir binarnih brojeva koji čak ima devet cifara u rezultatu.

Operacija **AND** se može koristiti za izdvajanje sadržaja dela registra. Pretpostavimo, na primer, da treba da izdvojimo donju cifru BCD broja iz akumulatora, čiji je sadržaj $10010101_{(2)}$. Ovo se može izvesti tako što napravimo masku koja sadrži jedinicu u bitima koji nas interesuju (npr. $00001111_{(2)}$ je maska za donju tetradu), i izvršimo operaciju **AND** maske i sadržaja akumulatora, slika 5.14(a).

		10010101	*originalni sadržaj akumulatora*
akumulator	10010101	01011001	*sadržaj posle rotacije za 4 mesta*
maska	00001111	00001111	maska
AND	00000101	00001001	**AND**

a) izdvajanje donje tetrade (b) izdvajanje gornje tetrade rotiranjem

Slika 5.14. Korišćenje **AND** operacije za izdvajanje **BCD** cifara iz registra

Obzirom da operacija **AND** daje jedinice samo na pozicijama gde oba operanda imaju jedinice, to rezultat kopira samo donji polubajt (tetradu) akumulatora, jer je maska u gornjem polubajtu nula, pa je samim tim i u rezultatu taj polubajt nula. Proces pravljenja maski i njenog logičkog množenja sa registrom može se kombinovati sa rotacijom i pomeranjem. Na slici 5.14(b) je prikazano izdvajanje gornjeg polubajta, pomoću maske za donji polu-bajt, i uz pomoć rotacije akumulatora za 4 mesta. Na sličan način može se izdvojiti bilo koji bit, postavljanjem odgovarajuće maske (sve nule sem na poziciji od interesa gde se postavlja jedinica).

XOR operacija ima takođe razne primene, a jedna od njih je i način za upisivanje broja nula u registar (**ekskluzivno ILI** sadržaja registra sa samim sobom), no najčešće se koristi za komplementiranje sadržaja željenih bitova. **XOR** se koristi za invertovanje nekog bita, pomoću maske gde su svi bitovi nula, samo je jedinica na poziciji od interesa. Ako hoćemo da invertujemo sadržaje bitova $\mathbf{b}_4$ i $\mathbf{b}_5$ koristimo masku 00110000, koja na mestima koja invertujemo ima jedinice a ostalo su nule, slika 5.15.

	10010111	originalni sadržaj registra
XOR	00110000	maska
	10100111	rezultat

Slika 5.15. Invertovanje bitova pomoću **XOR** instrukcije

Operacija **OR** se može koristiti za pakovanje podataka u jednu celinu. Neka registar **B** sadrži broj $00000100_{(2)}$, tj $(4_{(10)})$, a **C** registar sadrži broj $00001001_{(2)}$ $(9_{(10)})$. Ovo su dve BCD cifre koje hoćemo da spojimo u jedan BCD broj $49_{(10)}$ u registru **A**. Da bismo to postigli treba izvršiti sledeći niz operacija:

- izvršiti operaciju **clear A**, kojom se u akumulator upisuje nula,
- izvršiti operaciju **A OR B**, u registru **A** dobijamo **0000100**,
- pomeriti sadržaj registra **A** za četiri mesta ulevo; dobijemo **01000000** u **A**,
- izvršiti operaciju **A OR C;** u registru **A** dobijemo konačni rezultat, tj. niz **01001001 BCD** odnosno $49_{(10)}$.

Ovaj postupak se zove pakovanje **BCD** cifara, jer se **BCD** brojevi pamte po principu **1** cifra **1** bajt, u takozvanom raspakovanom obliku, a obrađuju se u pakovanom obliku tj. **2** cifre **u 1**-om bajtu.

Pogledajmo sada jedan mogući primer upotrebe računara i logičkih operacija za upravljanje nekim proizvodnim procesom. Neka se jedan 16-to bitni registar koristi za upravljanje (uključivanje i isključivanje) 16 prekidača: S_0, S_1, . . . , S_{14}, S_{15} (prekidač-**switch**). Ovi prekidači se mogu koristiti za uključivanje-isključivanje: svetiljki, motora, mašina, grejalica i sl. Stanje svakog prekidača može se opisati jednim bitom: jedinica znači da je prekidač uključen, a nula da je isključen. Neka su u nekom slučajnom trenutku vremena uključeni samo prekidači S_1, S_7, S_{12} i S_{14}. Ovom stanju prekidača odgovara sledeći sadržaj jednog registra, na primer akumulatora:

A_{cc}	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

Početno stanje akumulatora

1. Selektivno uključenje pojedinih prekidača, na primer $S_8 \div S_{11}$ bez promene stanja ostalih prekidača može se izvršiti logičkom operacijom ILI (**OR**) između sadržaja akumulatora i maske koja ima jedinice na mestima $M_8 \div M_{11}$ a u ostalim razredima su nule:

M	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Maska za uključivanje prekidača $S_8 \div S_{11}$

Treba izvršiti operaciju ILI nad sadržajima akumulatora i registra za masku:

	0	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
	0	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
OR	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

Početno stanje akumulatora

Maska za uključivanje prekidača $S_8 \div S_{11}$

Novo stanje akumulatora

2. Selektivno isključenje pojedinih prekidača, na primer $S_7 \div S_{10}$ bez promene stanja ostalih prekidača može se izvršiti logičkom operacijom I (**AND**) između sadržaja akumulatora i maske koja ima nule na mestima $M_7 \div M_{10}$ a u ostalim razredima su jedinice:

	0	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
	1	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
AND	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

Staro stanje akumulatora

Maska za isključivanje prekidača $S_7 \div S_{10}$

Novo stanje akumulatora

3. Selektivno invertovanje stanja pojedinih prekidača, na primer $S_4 \div S_{11}$ bez promene stanja ostalih prekidača može se izvršiti logičkom operacijom ekskluzivno ILI (**XOR**) između sadržaja akumulatora i maske koja ima jedinice na mestima $M_4 \div M_{11}$ a u ostalim razredima su nule:

	0	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
	0	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**
XOR	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**0**

Staro stanje akumulatora

Maska za invertovanje prekidača $S_4 \div S_{11}$

Novo stanje akumulatora

4. Invertovanje stanja svih prekidača može se izvršiti logičkom operacijom ekskluzivno ILI (**XOR**) sadržaja akumulatora i maske koja ima jedinice u svim razredima:

	0	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**0**
	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
XOR	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**

Staro stanje akumulatora
Maska za invertovanje svih prekidača
Novo stanje akumulatora

5.5. ARITMETIČKE BINARNE INSTRUKCIJE

Aritmetičke (binarne) instrukcije mehanizuju osnovne aritmetičke operacije (a pre svega sabiranje), nad nizovima nula i jedinica, koji predstavljaju binarne kodove brojeva. S obzirom da postoji više vrsta brojeva, čak i unutar jednog računara, onda je moguć i veliki broj različitih oblika ovih instrukcija. Ono što je sasvim izvesno jeste da svi računari imaju instrukciju za sabiranje dva cela pozitivna binarna broja. Računari koji rade u prvom i drugom komplementu, najčešće oduzimaju brojeve praveći komplement umanjioca, i pri tome vrše sabiranje. **Unisys 1100** računari, na primer, imaju instrukciju **add negative** da bi zamenili oduzimanje.

Vrednost brojeva koji mogu biti sabrani jednom instrukcijom, prirodno je ograničena veličinom registra, tj. osmobitna mašina sabira u jednom intervalu vremena osam bita, 16-bitna sabira 16 bita odjednom. Sabiranje većih brojeva zahteva niz instrukcija i korišćenje zastavice za prenos. Na primer, sabiranje heksadecimalnih brojeva $\mathbf{0DE2}_{(16)}$ i $\mathbf{1D95}_{(16)}$ u osmobitnom računaru zahteva dve instrukcije.

Prvo se saberu bajti manje težine ($\mathbf{E2}_{16} + \mathbf{95}_{16}$), i dobije se delimična suma $\mathbf{177}_{(16)}$. Dve zadnje cifre čine zbir **77** koji se pamti, a jedinica predstavlja prenos u sledeću instrukciju gde se sabiraju bajti veće težine, tj. $\mathbf{0D}_{16} + \mathbf{1D}_{16} + \mathbf{1}$, i dobija se $\mathbf{2B}_{(16)}$.

Konačan rezultat je $\mathbf{2B77}_{(16)}$, a dobija se spajanjem dva parcijalna zbira.

Mnogi računari mogu da sabiraju **BCD** brojeve, kao što sabiraju prirodno kodirane binarne brojeve. Neki procesori (i mikroprocesori) imaju specijalne instrukcije za operacije nad **BCD** brojevima, dok drugi koriste zastavice u registru stanja, da promene značenje uobičajene instrukcije sabiranja i oduzimanja (**6502** procesor), i pređu na **BCD** aritmetiku. **MC68020** računari imaju posebne instrukcije za sabiranje i oduzimanje binarnih brojeva (**ADD** i **SUB**), a posebne za **BCD** brojeve (**ABCD-Add Decimal with Extend** i **SBCD-Subtract Decimal with Extend**). **VAX** računari imaju specijalne instrukcije za sabiranje pakovanih **BCD** brojeva, binarnih brojeva i brojeva u pokretnom zarezu.

Većina osmobitnih mikroprocesora nema instrukcije za množenje i deljenje, već ih realizuje programski. Šesnaestobitni mikroprocesori imaju instrukcije za celobrojno množenje i deljenje. Veliki računari imaju instrukcije za celobrojno i **BCD** množenje i deljenje, a takođe i za izvođenje aritmetičkih operacija nad brojevima u

pokretnom zarezu. Veliki broj mikroprocesora za aritmetiku u pokretnom zarezu koristi specijalne matematičke procesore, koprocesore.

Veliki računari imaju pun set instrukcija u pokretnom zarezu, uključujući sabiranje, oduzimanje, množenje i deljenje. Oni, takođe, imaju i instrukcije za konverziju celih brojeva u brojeve u pokretnom zarezu, i obratno, instrukcije za normalizaciju brojeva u pokretnom zarezu itd. **VAX** računari imaju više od 50 instrukcija koje podržavaju konverziju brojeva iz jednog oblika u drugi.

LITERATURA

[1] Alexandridis, N.A., “Microprocessor Systems-Architecture and Engineering”, Infortech State of the Art Report on Microeletronics, Pergamon Infotech, series 8, no. 2, 1980.

[2] Alexandridis, N. A., “Microprocessor System Design Concepts”, Computer Science Press, Rockvile, 1984.

[3] Crook, C., “Microcomputer Architecture-A Survey Report”, Infortech State of the Art Report on Microelectronics, Pergamon Infotech, series 8, no. 2, 1980.

[4] Davis, S., “Microprocessors”, EDN, August 5, 1979.

[5] Hayes J. P., “Computer Architecture and Organization”, McGraw-Hill International Book Company, 1983.

[6] Hilburn J.L. and E. Teja, “ As You Get to Know the 8086, Use Your 8-bit Experience”, EDN, January 20, 1979.

[7] Intel Corporation, 8080 Microcomputer System User`s Manual, Santa Clara, CA, September 1975.

[8] Jauković M. “Uvod u informacione sisteme”, Tehnička knjiga, Beograd, 1992.

[9] Johnson, R.C., “32-bit Microprocessors Inherit Mainframe Features”, Electronics, February 24, 1981.

[10] Kvaternik R. “Uvod u operativne sisteme”, Informator, Zagreb, 1988.

[11] LeMair, I., and R. Nobis, “ Complex Systems are Simple to Design (with the MC68000)”, Electronic Design, 18, September 1, 1978.

[12] Luce T. “Computer Hardware, System Software, and Architecture” Mitchell Publishing, Inc. Watsonville, CA , 1989.

[13] Madnick S. and Donavan J., “Operating Systems”, McGraw-Hill, New York, 1974.

[14] Mijalković M. "Programiranje MSC96 serije mikrokontrolera", Viša elektrotehnička škola, Beograd, 2002,

[15] Motorola, Incorporated, 16-Bit Micrprocessing Unit, Austin, TX, 1980.

[16] Nikolić M & Nikolić M. “Disk operativni sistem MS DOS 5.0” Tehnička knjiga, Beograd, avgust 1992.

[17] Osborne, A. “An Introduction to Microcomputers, Volume II, Some Real Products”, Berkeley, CA: Adam Osborne and Associates, Inc., 1976.

[18] Ribarić S. “Arhitektura mikroprocesora”, Tehnička knjiga, Zagreb, 1985.

[19] Smiljanić G. “Osnove digitalnih računala”, Školska knjiga Zagreb, 1978.

[20] Stanković M. i Stanković M. “Računarski sistemi”, Zavod za udžbenike i nastavna sredstva, Beograd, 1990.

[21] Stojković V. i Tošić D. “Programski sistemi I deo”, Naučna knjiga, Beograd, 1979.

[22] Stritter, E., and T. Gunter, “ A Microprocessor Architecture for a Changing World: The Motorola 68000”, Computer, February 1979.

[23] Velašević D. “Uvod u sistemsko programiranje I”, Naučna knjiga, Beograd, 1988.

[24] Vujičić V. “Uvod u C jezik”, Univerzitet u Beogradu, Institut za nuklearne nauke “Boris Kidrič” Vinča, 1988.