ISTVÁN MATIJEVICS

OSNOVI PROGRAMIRANJA I PROGRAMSKI JEZICI

--- skripta –

VIŠA TEHNIČKA ŠKOLA

SUBOTICA – 2001

SADRŽAJ

PROGRAMSKA PODRŠKA RAČUNARSKIH SISTEMA

Računarski sistem predstavlja jedinstvenu celinu hardvera (fizičke arhitekture) i softvera (programske podrške) koja obezbeđuju konačan broj usluga korisnicima.

Hardver (tvrdotvorina): Fizički deo računarskog sistema koji obuhvata električne/elektronske komponente (npr. uređaje i kola), elektromehaničke komponente (npr. diskovni pogon, štampač itd.), optoelektrične komponente i mehaničke komponente (npr. kućišta).

Pod programskom podrškom računarskih sistema podrazumeva se skup programa računarskog sistema koji omogućavaju njegovo efikasno korišćenje od strane korisnika.

Softver (mekotvorina): Generički termin za one komponente računarsog sistema koje nisu fizičke. Najčešće se odnosi na programe koje izvršava računarski sistem, za razliku od hardvera tog računarskog sistema, i uključuje i simboličke i izvršive oblike takvih programa, Razlika se može povući između sistemskog softvera, koji je esencijalni pratilac hardvera i obezbeđuje efikasnost čitavog računarskog sistema (najčešće isporučuje proizvođač) i aplikacionih programa specifičnih za određenu ulogu.

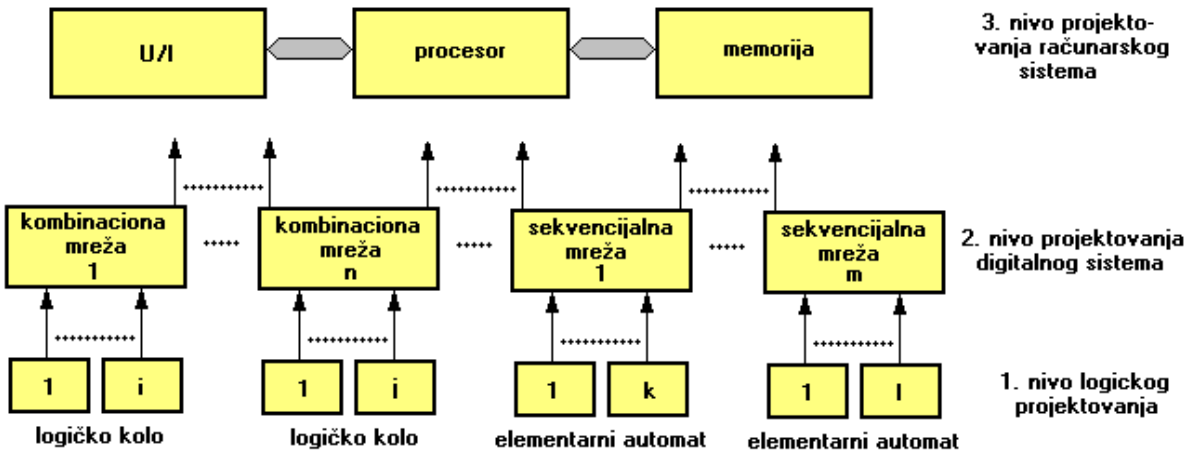
Sa stanovišta razvoja korisnici računarskog sistema se pojavljuju kao:

- korisnici koji razvijaju hardver računara i
- korisnici koji kao osnovni sistem koriste hardver, su projektanti programske podrške namenjene za upravljanje funkcionalnim jedinicama računarskog sistema i informacionom osnovom programske podrške.

Hardver ima tri nivoa strukture (arhitekture) računarskog sistema (slika 1.).

Prvi i drugi nivo fizičke arhitekture se projektuje korišćenjem Boole-ove algebre, teorije automata i mikroprogramskih automata. Treći nivo se projektuje koristeći mikroprogramski jezik (jezik za definisanje razmene informacija između registara), radi definisanja mašinske instrukcije, kao skupa mikroinstrukcija.

Bulovska algebra: Algebra naročito važna u računarstvu. Formalno uzev, to je komplementarna, distributivna mreža. U bulovskoj algebri postoji skup elemenata B koji se sastoji samo iz 0 i 1. Nadalje, postoje dve dijadske operacije koje zovemo I i ILI, postoji i monadna operacija, komplementiranje.



Slika 1.: Nivoi u projektovanju fizičke arhitekture računarskog sistema

Projektanti, koji koriste skup primitivnih operacija (mašinskih instrukcija) i određene tipove i strukture podataka radi definisanja programa koji upravljaju resursima računarskog sistema (operativna memorija, procesor, ulazno-izlazni podsistem i informacije u obliku programa i skupova podataka). U ovoj interpretaciji nad fizičkom arhitekturom trećeg nivoa formira se novi sloj (operativni sistem), a komunikacija sa njim se definiše preko skupa programa, kao primitivnih operacija i precizno definisanih struktura odgovarajućih podataka. Ovaj tip korisnika definiše programera sistema (projektanta operativnog sistema).

Mašinska instrukcija: Naredba centralnoj jedinici računara, kojom se izvršava neka aritmetička, logička ili kontrolna operacija. Mašinske instrukcije su predstavljene u binarnom obliku, tj. u obliku koji centralna jedinica može odmah da obradi.

Operativni sistem (OS): Skup programskih podataka koji zajedno kontrolišu sistemske resurse i procese koji te resurse koriste u okviru jednog računarskog sistema.

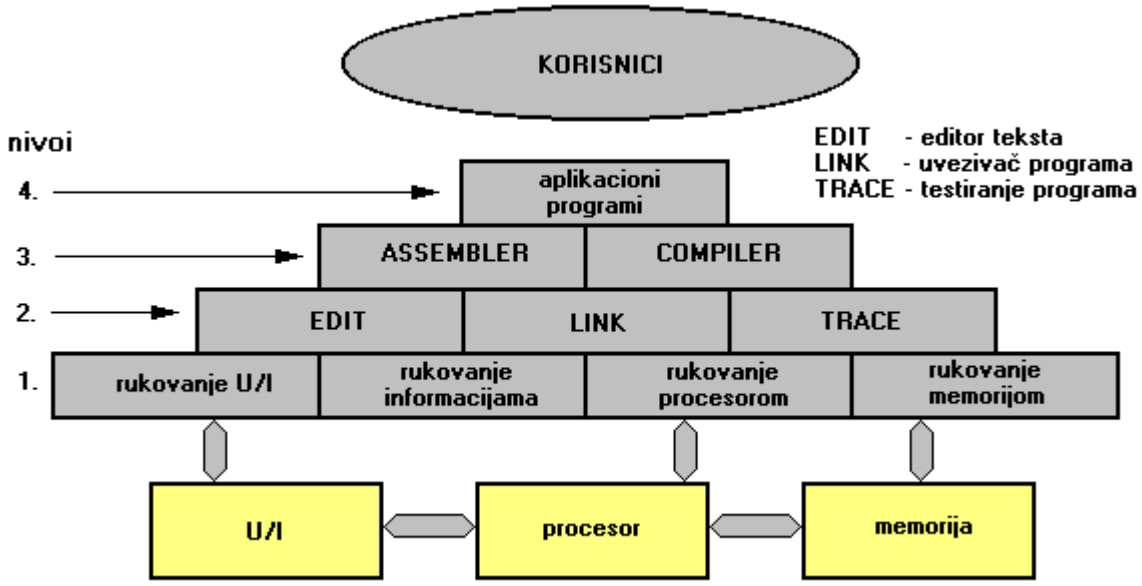
Ukoliko se nad nivoom operativnog sistema definišu programi za komunikaciju korisnika sa računarskim sistemom preko prevodioca sa programskog jezika (asemblera i viših programskih jezika, znači simboličkih programskih jezika) onda se radi još o dve kategorije korisnika.

U prvoj grupi korisnici koriste programe rutine operativnog sistema i njegove strukture podataka radi projektovanja pomoćnih programa preko kojih korisnik, koristeći programski jezik, komunicira sa računarskim sistemom. Ovi programeri su projektanti podsistema.

U drugoj grupi su programeri računarskog sistema. Oni koriste programski jezik radi rešavanja na računarskom sistemu konkretnih programa.

Operativni sistem je deo programske podrške koji obuhvata više programa i odgovarajuće memorijske uređaje za smeštanje informacija.

Programska podrška računarskog sistema je na slici 2.



Slika 2.: Hijerarhijski nivoi programske podrške računarskog sistema

Na nivoima su dati samo primeri programa, različiti računari mogu imati manje ili više od takvih programa.

Nivoi su:

1. najniži nivo, tu su programi koji upravljaju resursima računarskog sistema, kod IBM PC računara ovaj nivo je BIOS (Basic Input Output System),
2. na ovom nivou su programi za punjenje programa sa periferskih jedinica u operativnu memoriju, teksteditori i trace ili debugger programi za otklanjanje grašaka,
3. na ovom nivou su programi za prevođenje izvornog koda (programa) u mašinski kod, asembler i kompajleri viših programskih jezika (Pascal, C itd.),
4. na ovom nivou je skup aplikacionih programa, preko kojih korisnik komunicira sa računarom.

Na trećem nivou razlikujemo:

- prevođenje izvornog koda u mašinski jezik pomoću prevodilaca, gde su faze prevođenja i izvršenja vremenski nezavisne i

Prevodilac (compiler, translator): Svaki program koji prevodi neki drugi program iz jednog oblika u drugi. Prevodilac je program koji prevodi izvorni program, napisan na nekom višem programskom jeziku, u mašinski program.

- interpretaciju izvornog koda, gde su faze prevođenja i izvršenja vremenski zavisne.

Interpretator: Specijalna vrsta prevodioca programskih jezika, koji svaku naredbu jezika zasebno prevodi i odmah izvršava.

OPERATIVNI SISTEMI

UVOD

Prvi računari nisu imali operativne sisteme, međutim, danas se i kod najjednostavnijih mikroračunarskih sistema može uočiti neka vrsta operativnog sistema.

Operativni sistem predstavlja skup modula (programa) koji obezbeđuju jednostavnije korišćenje hardvera i softvera računarskog sistema. Uloga operativnog sistema je da hardverske mogućnosti računarskog sistema učini dostupnim čoveku i omogući mu što udobniji rad, obezbeđujući, pri tom, što efikasnije korišćenje svih resursa računarskog sistema.

Osnovne funkcije operativnih sistema su:

- komunikacija između korisnika i računarskog sistema,
- raspodela resursa računarskog sistema među korisnicima,
- planiranje pristupa zajedničkim skupovima podataka za sve korisnike,
- planiranje pristupa korisnika zajedničkim resursima itd...

Osnovni resursi kojima upravlja operativni sistem su:

- procesor,
- memorije,
- ulazno-izlazni uređaji i
- podaci (informacije).

U toku rada operativni sistem opslužuje:

- sistem-programere,
- programere,
- operatore,
- razne vrste korisnika,
- programe i
- hardverske uređaje.

Ako računarski sistem raspolaže moćnim hardverom, operativni sistem mora da omogućuje iskorišćenje mogućnosti tog harvera. Operativni sistem predstavlja deo softvera koji je najbliži hardveru. Posredstvom operativnog sistema svi ostali programi pristupaju hardveru računara.

Računar u različitim periodima može raditi pod kontrolom različitih operativnih sistema, a, jedan operativni sistem je moguće koristiti na različitim računarima (uz prilagođavanje konkretnom tipu računara). Do sada je napravljen veliki broj operativnih sistema za različite tipove računara. Napraviti dobar operativni sistem nije lak posao, pa su i danas aktuelni operativni sistemi čije su osnove postavljene još pre dvadesetak godina.

PREGLED RAZVOJA OPERATIVNIH SISTEMA

Slično kao i u razvoju računarskih sistema, možemo razlikovati nekoliko generacija u razvoju operativnih sistema. Razvoj operativnih sistema usko je povezan sa

napretkom tehnologije u računarstvu. Prvi računari praktično su bili bez operativnih sistema. Programiralo se na mašinski-orijentisanim jezicima i sve ono što danas rade operativni sistemi, programer je morao sam da isprogramira.

PRVA GENERACIJA OPERATIVNIH SISTEMA

Prvi operativni sistem napravljen za računar IBM-701 početkom pedesetih godina. Operativnih sistemi prve generacije bili su na većim računarima, dok su manji računari i dalje bili bez operativnih sistema. Razvoj prve generacije je potrajao do kraja pedesetih godina. Na računarima se primenjivala paketna obrada podataka. To je postupak u kojem se programi izvršavaju jedan za drugim i dok se izvršava jedan program, ceo računar je njemu na raspolaganju. Programi i podaci su unošeni u računar preko bušenih kartica, koje su slagane u pakete. Dosta je vremena gubljeno prilikom prelaska sa jednog programa na drugi. Prvi operativni sistemi pravljeni su sa ciljem da se skrati vreme prelaska sa jednog programa na drugi, podržavajući pri tom paketnu obradu podataka. Pored redne obrade podataka, za operativne sisteme prve generacije karakteristično je da su posedovali standardne rutine za kontrolu ulaza-izlaza podataka i jednostavan jezik za opis zadatka korisnika.

DRUGA GENERACIJA OPERATIVNIH SISTEMA

Druga generacija operativnih sistema pojavljuje se šezdesetih godina. Kod ovih operativnih sistema je prvi put korišćen multiprogramski režima rada. U ovom režimu nekoliko zadataka se istovremeno nalazi u osnovnoj memoriji računara, a centralni procesor deo svog vremena posvećuje svakom zadatku.

U istim godinama su se pojavili višeprocesorski sistemi kao i operativni sistemi koji su podržavali takav rad.

Višeprocesorski sistem: Računarski sistem, sastavljen iz dve ili više centralnih jedinica, koje koriste zajedničku operativnu memoriju. Višeprocesorski sistemi se koriste za paralelnu obradu podataka.

Znatan broj novih rešenja kod operativnih sistema druge generacije prisutan je i kod savremenih operativnih sistema:

- Rad u razdeljenom vremenu i mogućnost korišćenja terminala za tzv. dijalog sa računarom. Svaki korisnik je u direktnom kontaktu sa računarom preko svog terminala. Zbog velike brzine rada, centralni procesor deli svoje vreme i svakom korisniku posvećuje jedan vremenski interval. Pošto je brzina rada čoveka neuporedivo manja od brzine centralnog procesora, svi korisnici imaju utisak da istovremeno koriste centralni procesor.
- Podrška programima koji ne zavise od imena konkretnih periferiskih uređaja, tj. postizanje sve većih nezavisnosti programa od konkretnog hardvera . Od nastanka prvih računara čovek je nastojao da programiranje učini što manje zavisnim od konkretnog računara. Značajan doprinos u ovoj težnji predstavljaju nova rešenja kod operativnih sistema druge generacije.

- Podrška rada u realmom vremenu . Za operativni sistem se kaže da radi u realnom vremenu ako ima mogućnosti brzog odgovora i kontrole spoljašnjih događaja. Naime, ako se pod kontrolom računara odvija nekakav tehnološki proces.npr. rad nuklearne centrale, korišćenje seizmičkih aparata, let satelita itd.kaže se da računar radi u realnom vremenu. Da bi računar mogao da radi u realnom vremenu, potreban je dovoljno brz hardver, ali i operativni sistem.
- Korišćenje *virtualnih* memorija. Korišćenje ovih memorija započelo je prilikom razvoja operativnih sistema, a stečena znanja su preneta na mikroprocesore.
- Sve veća podrška jezicima višeg nivoa. Pojedini operativni sistemi iz ove generacije omogućavaju testiranje na jeziku višeg nivoa. Osim toga, sada se jezici višeg nivoa koriste i za pisanje samih operativnih sistema.

TREĆA GENERACIJA OPERATIVNIH SISTEMA

Treća generacija operativnih sistema je realizovana 1964. godine na računaru IBM/360. Računar IBM/360 tj. računar opšte namene, kao i operativni sistem OS/360, bili su koncipirani tako da zadovoljavaju sve potrebe korisnika. Ovi operativni sistemi uključuju sve režime rada:

- rednu obradu (paketna obrada podataka),
- multiprogramski rad,
- rad u razdeljenom vremenu i
- rad u realnom vremenu.

Jezik operativnog sistema treće generacije je težak za učenje, pojedini operativni sistemi zauzimali su mnogo memorijskog prostora i sporo su reagovali, što je usporavalo rad celog računarskog sistema.

60-tih godina su se pojavili prve verzije Unix operativnog sistema koji će kasnije dosta uticati na razvoj drugih operativnih sistema.

ČETVRTA GENERACIJA OPERATIVNIH SISTEMA

Od početka sredine sedamdesetih godina su razvijani operativni sistemi 4-te generacije. Danas korišćeni operativni sistemi spadaju u istu generaciju operativnih sistema.

Današnji period karakteriše sve veća povezanost računara i masovna upotreba personalnih računara (mreže, Internet). Za računare povezane u mrežu potrebni su odgovarajući mreži operativni sistemi. Kod takvih operativnih sistema veoma važnu ulogu igra komunikacija (razmena informacija) između više računara, a i zaštita podataka od neovlašćenog korisnika je bitna.

U ovom periodu raste broj korisnika računara, zbog te činjenice operativni sistemi moraju brzo reagovati. Razvijeni su orijentisani operativni sistemi. U cilju olakšanja korišćenja računara mnogi operativni sistemi podržavaju rad sa tzv. "virtualnim mašinama". Korisnik ne mora da poznaje karakteristike konkretnog računara ili mreže na kojima će njegov problem biti rešavan, on pretpostavlja da računar ima određene

funkcionalne mogućnosti i koristi te mogućnosti. Od operativnog sistema zavisi koji hardverski deo će zadužiti da rešava dati zadatak. Na taj način korisnik bira mašinu koja mu odgovara, a operativni sistem ima zadatak da od zamišljene mašine stvori realnu. "Virtualne mašine" su po svojoj koncepciji najčešće veoma bliske realnim.

Operativni sistemi personalnih računara u principu su jednostavniji, mada se neki od njih, po svojim mogućnostima, približavaju operativnim sistemima većih računara, na primer IBM PC računar.

STRUKTURA OPERATIVNIH SISTEMA

Razni računari zahtevaju različito koncipirani operativni sistemi, kod jednih je više naglašena jedna funkcij, kod drugih druga itd. Zajednička osobina im je da svi imaju modularnu strukturu. U predhodnom delu je već navedeno da se kod svakog operativnog sistema mogu prepoznati neke osnovne funkcije. Ove funkcije su kod velikih sistema veoma složene i svaka od njih se može razbiti na niz podfunkcija. Kod pojedinih operativnih sistema neke od ovih podfunkcija toliko su izražene da po značaju prevazilaze i osnovne funkcije.

Moderni operativni sistemi napravljeni su tako da predstavljaju skup modula. Modul je napravljen kao posebna programska celina koja realizuje jednu funkciju (podfunkciju) operativnog sistema. Pojedini moduli su nezavisni i mogu funkcionisati nezavisno jedan od drugog, ali i pozivati jedan drugog, tj. ukazivati na potrebene podatke za početak rada sledećeg modula. Funkcije modula se nadovezuju jedna na drugu, a obično postoji modul koji koordinira akcije svih ostalih.

Modularno programiranje: Tehnika programiranja, kod koje se jedan program deli u više logički nezavisnih celina (modula). Svaki modul se razvija i prevodi nezavisno od drugih. Kod većih programskih paketa, više programera obično razvijaju module, koji čine jedan sistem. Modularno programiranje predstavlja jednu od tehnika strukturnog programiranja.

Broj modula operativnog sistema može biti vrlo veliki i ako bi se svi u isto vreme nalazili u unutrašnjoj memoriji, zauzimali bi mnogo memorijskog prostora. Osim toga, mnogi od njih su veoma retko korišćeni, dok su pojedini potrebni stalno. Moduli koji se često upotrebljavaju, obično su sve vreme prisutni u operativnoj memoriji i oni čine rezidentni deo operativnog sistema. Drugi deo čine moduli koji se pamte na nekoj spoljašnjoj memoriji (obično na tvrdom disku) i oni čine tzv. tranzitni deo operativnog sistema. Po potrebi moduli tranzitnog dela mogu da se pozivaju u operativnu memoriju. Neki od modula su napravljeni tako da se mogu više puta pozivati sa spoljašnje memorije, a da se pri jednom od poziva ne moraju do kraja izvršiti. Ovo se postiže razdvajanjem modula na dva dela: fiksnog i promenjivog. Fiksni deo je u operativnoj memoriji, a promenjivi na spoljašnjoj i može da bude pozvan kada je potrebno.

Prilikom startovanja računara (uključivanja u mrežu napajanja), potrebno je rezidentni deo operativnog sisteama upisati u operativnu memoriju sa neke spoljašnje. Za ove svrhe služi poseban modul, zapamćen u ROM-u računara, koji se naziva punilac operativnog sistema. Kod nekih računara u ROM-u se pamti ceo jedan deo

operativnog sistema, koji ima niz elementarnih funkcija. Ove funkcije se koriste prilikom izgradnje ostalih modula operativnog sistema.

Kod operativnih sistema mogu se razlikovati nekoliko slojeva. U centru je jezgro operativnog sistema, koje čini skup modula za realizaciju elementarnih funkcija. Zatim sledi sledeći sloj koji čine moduli sa funkcijama koje predstavljaju nadgradnju na elementarne funkcije (moduli za upravljanje procesorima i memorijom). Iza ovih slede novi slojevi (sloj modula za upravljanje datotekama) itd. Zahvaljujući modularnoj strukturi, moguće je koncipirati savremene veoma moćne operativne sisteme na ovaj način.

JEZIK OPERATIVNOG SISTEMA

Komunikaciju sa operativnim sistemom korisnik uspostavlja preko jezika operativnog sistema. Jezik operativnog sistema je izgrađen nad:

- skupom komandi,
- skupom propisanih imena i
- skupom raznih poruka.

Većina komandi uključuje određen broj potkomandi i na taj način se razgranava jezik. Po pravilu, jezik operativnog sistema je interpretatorskog tipa. To znači, komande se mogu izvršavati odmah po izdavanju.

Kod većine operativnih sistema prve, druge, pa i treće generacije, nije posvećena posebna pažnja jeziku operativnog sistema. Kod ovih operativnih sistema mahom su se koristili specijalni simboli (kose crte, zvezdice i dr.) i skraćenice engleskih reči. Za ovakav jezik nemože se reći da je posebno udoban za korisnika. Treba imati u vidu da se ovi operativni sistemi upotrebljavaju u periodu kada interaktivni rad nije bio još rasprostranjen i ovakav jezik je prilagođen rednoj (paketnoj) obradi podataka.

Jezici savremenih operativnih sistema su mnogo bogatiji. Pojedini operativni sistemi uključuju i po nekoliko jezika, zavisno od vrste korisnika. Na primer, poseban jezik je rezervisan za sistem-programere, a poseban za programere-korisnike. Komande ovih jezika su obično engleske reči (ili njihove skraćenice). Pojedini operativni sistemi omogućavaju lako prevođenje komandi sa engleskog na neki drugi jezik. Dovoljno je svakoj engleskoj reči odrediti odgovarajuću reč drugog jezika. Pošto se komande izdaju ispisivanjem nekakvog teksta, prirodno je da postoji i neki način za uređivanje tog teksta (pojedine komande sa uključenim potkomandama mogu da budu dugačke). Naime, ukoliko korisnik pogreši u kucanju jednog slova, nije normalno da kuca celu komandu ponovo, već je razumno da postoji jednostavan način za korekciju greške. To znači da operativni sistem treba da uključuje u sebe i neku vrstu editora. U ovakvim slučajevima editovanje se obično realizuje pomoću tzv.funkcionalnih tastera.

Neki jezici operativnog sistema su zasnovani na menijima, tj. skup osnovnih komandi uvek je popisan na ekranu i korisnik bira komandu birajući njeno obeležje ili nešto slično. Normalno je da postoji komanda koja služi za pomoć, tj. pomoću koje se može videti spisak komandi, koje su na raspolaganju korisniku ili pojedinosti vezane za neku komandu.

Kod većine savremenih operativnih sistema korisnik može da izdaje komande operativnom sistemu neposredno ili posredno. Posredno izdavanje komandi obično se vrši zapisivanjem niza komandi operativnog sistema u posebnu datoteku. Pozivanjem te datoteke (ili na neki drugi način), aktivira se posredno izvršavanje komandi. Neposredno izdavanje komandi operativnom sistemu vrši se preko terminala.

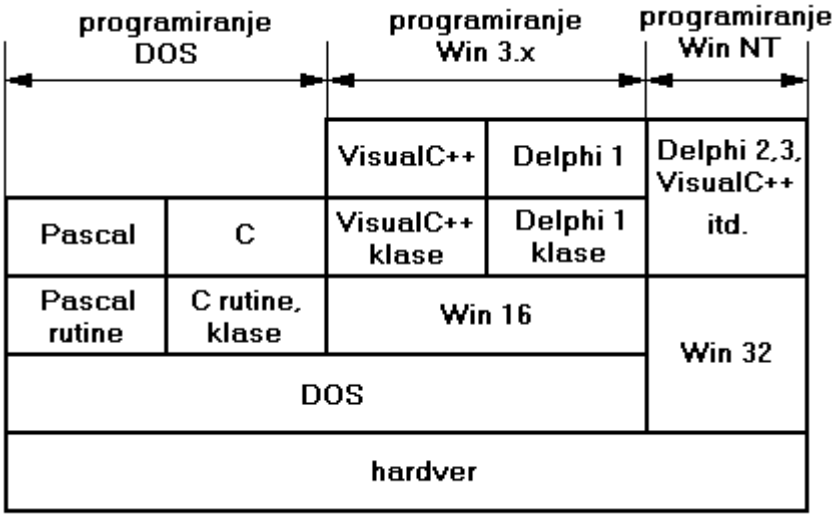
Da bi korisnik mogao neposredno da izda komandu operativnom sistemu, treba da zna da li mu je operativni sistem na raspolaganju. Stoga, svaki operativni sistem ima nekakav znak preko kojeg saopštava korisniku da je spreman za prihvatanje komandi (PROMPT).

OPERATIVNI SISTEMI MIKRORAČUNARA

Operativni sistemi mikroračunara su u principu jednostavni od operativnih sistema većih računara, zato, što su zadaci mikroračunara često specijalizovani i vezani za jednostavne zadatke, kao na primer sistemi za prikupljanje podataka, regulatori, PLC uređaji itd. Ponekad su to samo jednokorisnički orijentisani operativni sistemi a ponekad ovi operativni sistem podržavaju višekorisnički način rada.

Pre nekoliko godina operativni sistem osmobitnih mikroračunara je bio CP/M (Control Program for Microcomputers), danas je to kod IBM PC računara MS (PS) DOS (Microsoft).

Pored DOS operativnog sistema danas na IBM PC personalnih računara koristimo i Windows operativne sisteme, gde glavnu ulogu kod komunikacije igraju miš i ikone. Na slici 3. je dat pregled operativnih sistema IBM PC personalnih računara.



Slika 3: Pregled operativnih sistema IBM PC računara

OPERATIVNI SISTEM -*DOS*

Firma Digital Research je izradila operativni sistem CP/M (Control Program for Minicomputers) koji je korišćen za osam-bitne procesore (Intel 8080 i Zilog Z80). Za 16-bitne procesore (Intel 8086) napisan je operativni sistem 86-DOS koji je izrađen na bazi operativnog sistema CP/M. Veoma slične su:

- komandne strukture,
- organizacija diska,
- imenovanje datoteka,
- imenovanje diskova i
- sintakse komandi.

Operativni sistem DOS prezentovan je javno u septembru 1980. godine.

Kompjuterske kuće IMB i Microsoft su zajedno izradili operativni sistem MS-DOS 1.0 (Microsoft Disk Operating System) koji je prvi put proradio 1981. godine na IBM PC personalnom računaru. U ovom operativnom sistemu komandni procesor je izdvojen i smešten u poseban program sa nazivom COMMAND.COM. Sistem je radio sa jednostranim disketama od 160 KB.

Sledeća verzija DOS-a je MS DOS 1.1. i koristi dvostruku disketu od 320 kB.

Hard disk od 10 MB omogućuje realizaciju novog PC-računara, koji je IBM-XT. Kod ovog računara trebalo je organizovati na drugi način tvrdi disk. Definisan je katalog, koji je hijerarhijski organizovan. Operativni sistem je DOS 2.0, koji je imao mogućnost redirekcije i novi sistem upravljanja datotekama (kao kod UNIX-a). Omogućena je jednostavna podrška raznim ulazno izlaznim uređajima preko fajla CONFIG:SYS. Diskete su od 360 kB.

Tendencija je kod razvoja PC računara povećanje kapaciteta, memorije i diskova i brzine procesora. Na pojačanim verzijama radi OS operativni sistem 2.1 i 2.11.

Sledeća generacija IBM PC računara je PC AT računar (Advanced Tehnology) koji ima novi procesor, Intel 80286. Na ovim verzijama pojavi se potpuno nov operativni sistem OS/ 2, ali se radi i na usavršavanju DOS-a i javlja se nova verzija MS DOS 3.0, koja je obezbeđivala podršku sistemu sa diskom od 20 MB i disketama od 1.2 MB. Rad u mreži je realizovan sa verzijom MS DOS 3.1, a verzija MS DOS 3.2 omogućava rad sa disketama od 3.50" i kapacitetom od 720 KB. Januara 1986. godine je izrađen DOS 3.25.

DOS 3.30 pojavljuje se aprila 1987. godine, gde je povećana brzina i komfornost u radu. Korišćene su diskete od 3.50" kapaciteta 1.44 MB i diskovi veći od 32 MB. Uvedeno je straničenje memorije.

Rapidni razvoj hardverske tehnologije zahteva i brz razvoj operativnih sistema. Realizovani su novi, prijateljski orijentisani (user friendly) grafički interfejsi, koji korisniku olakšavaju komunikaciju sa računarom. Svaka nova verzija dovodi do povećanja performansi sistema i iskorišćenja novih hardverskih mogućnosti.

Stabilan operativni sistem je MS DOS 6.20.

RAD SA OPERATIVNOM SISTEMOM MS DOS

Operativni sistem DOS se koristi za pokretanje sistema, upravljanje hardverskim resursima kojima mašina raspolaže, rad sa njegovim vlastitim programima i

podacima, manipulaciju datotekama i podršku u izvršavanju aplikacijskim programima koje korisnik instalira na svoju mašinu.

Obično korisnik ne poznaje šta operativni sistem radi u pozadini i na koji način pokreće sistem, raspoređuje memoriju, kako vodi organizaciju spoljne memorije, kako upravlja perifernim jedinicama, niti ijednu drugu radnju koju OS izvršava. Korisnik poziva komande koje nudi operativni sistem i druge aplikacije instalirane na mašini.

Druga je situacija, ako korisnik piše programe koji trebaju izvršiti određene zadatke, on u izvesnoj meri mora poznavati strukturu i način rada operativnog sistema kako bi na najoptimalniji način iskoristio sve resurse računara.

Neke komande mogu dovesti do gubitka svih prethodno snimljhenih podataka na diskovima. U početku je preporučljivo striktno se držati neke stručne literature, i ne pokušavati odmah vlastite varijacije isprobati, sve dok se ne nauči suština operativnog sistema i shvate mogućnosti svake komande pojedinačno.

Na neke operacije operativnog sistema korisnik ne može uticati. Nakon uključivanja računara doći će do podizanja sistema, a da korisnik ništa ne doprinese toj radnji. Sa menjanjem sadržaja dva fajla, CONFIG.SYS i AUTOEXEC.BAT možemo uticati na neke operacije. DOS proverava sadržaj ta dva fajla i iz njih učitava niz naredbi koje definišu okruženje u kojem će mašina raditi. Većina ovih parametara definisanja sistema može se i kasnije, u toku rada predefinisati na nove vrednosti.

Resursi na kojima upravlja operativni sistem (hardver i softver) korisnik pristupa preko komandi (naredbi). Naredbe su skup karaktera veličine od jednog do osam karaktera (istih onih koji se koriste za imenovanje datoteka), nakon čega obično ne sledi ništa ili jedan prazan karakter i skup parametara koji definišu način na koji se naredba izvršiti, odnosno nad kojim medijem. Nakon završetka unošenja komande i pridruženih parametara obavezno se mora pritisnuti taster ENTER, da bi OS shvatio da ste završili sa unosom i da želite da se komanda izvrši.

Komande mogu biti:

- interne i
- eksterne.

Interne komande su permanentno prisutne u operativnoj memoriji računara od momenta podizanja sistema do njegovog gašenja.
Eksterne komande se učitavaju sa diska i u memoriji su prisutne samo dok su aktivni programi koje one izvršavaju.

Druga podela komandi operativnog sistema je:

- komande za rad sa katalozima,
- komande za rad sa datotekama,
- komande za predstavljanje parametara sistema i
- pomoćne komande.

PODIZANJE SISTEMA

DOS operativni sistem se nalazi u dve datoteke IO.SYS i MSDOS.SYS (ili IBMIO.SYS i IBMDOS.SYS). Ove datoteke se moraju nalaziti na hard-disku ili disketi, a njihovi atributi su "HSR". (H-hidden S-system R-read only, skriven, sistemski i samo čitljiv). Po uključenju računara dešava se sledeće :

- Inicijalizuje se BIOS ROM (Basic Input Output System), pri čemu se javlja poruka o verziji BIOS-a i vrši testiranje memorije (za to vreme hard disk dostiže nominalnu brzinu okretanja).
- BIOS inicijalizuje kontroler disketne jedinice i testira da li se disketa nalazi unutar drajva (ovo samo u slučaju da je korisnik, ili dizajner sistema odlučio da se to ispitivanje vrši). Ako disketa postoji BIOS čita takozvani but-sektor (boot, nulti sektor) da bi ustanovio da li disketa sadrži DOS. Ako nema DOS-a na disketi javlja to korisniku porukom
    - INVALID BOOT DISKETTE (pogrešna sistematska disketa)
    - Insert any key when ready Pritisni bilo koji taster.
- Ova poruka će se javljati sve dok ne ubacite pravilnu disketu (formatiranu sa operativnim sistemom na istoj) ili dok ne izvadite disketu iz drajva A:,
- Ako BIOS "vidi" da disk jedinica nije spremna, on ispituje da li je DOS smešten na hard disk (što je kod najvećeg broja računara slučaj). Ako nema DOS-a ni na hard disku onda sistem "zaglavi",
- Ako BIOS pronađe DOS (bilo na disketi ili na disku) obavljaju se sledeće operacije:
    - 1)BIOS učitava IO.SYS i predaje mu kontrolu
    - 2)IO.SYS sadrži dva modula:
        - prvi se koristi za pokretanje programa za tastaturu i ekran, serijski kanal i štampač, kao i za upravljanje jedinicom diska,
        - drugi modul (SYSINIT) određuje veličinu memorije, premešta samog sebe na vrh raspoložive memorije (RAM) i zatim učitava datoteku MSDOS.SYS (srce operativnog sistema).
    - 3)MSDOS šalje pozdravnu poruku sa informacijama o verziji DOS-a, inicijalizuje postojeće pokretačke programe, kreira i popunjava interne tabele i radne bafere, kao i prekidne vektore, a zatim vraća kontrolu modulu SYSINIT.
    - 4)Vrši se analiza datoteke CONFING.SYS (ako ista postoji) i podešavanje pojedinih parametara sistema na osnovu informacije sadržanih u toj datoteci.
    - 5)DOS je spreman za rad, sistemskim pozivom selektira interpreter DOS-a, COMMAND.COM, koji izvršava program AUTOEXEC.BAT (ako isti postoji) i šalje prompt (obično skup karaktera poput"C:\").
- Program COMMAND.COM se mora nalaziti na svakoj sistemskoj disketi. I pored toga, on ne spada u operativni sistem. Radi se o izvršnom (.COM) programu koji je sličan svakom drugom izvršnom programu, ali je on zadužen za vrlo bitne operacije. Komandni interpreter je podeljen u tri modula:

- Prvi deo je stalno u memoriji i obrađuje informacije da li je tražen prekid, i da li se javlja kritična greška i, ujedno, obezbeđuje izlazak iz drugih programa i povratak na DOS-ov prompt.
- Drugi modul komandnog interpretera se pojavljuje samo pri prvom upisivanju i odgovoran je za obradu datoteke AUTOEXEC.BAT. Nakon toga nije više potreban, pa se odbacuje iz memorije.
- Treći modul se smešta na sam vrh memorije i u sebi sadrži obrade internih DOS komandi, startovanje programa i obradu datoteka tipa *.BAT. Sa pozicije na kojoj se nalazi lako ga može prebrisati program kojeg je on startovao. Prvi modul se stara da se treći modul ponovo učita sa diska ako se prebrisavanje desilo.

RESETOVANJE SISTEMA

Startovanjem neispravnih ili neispravno instaliranih programa računar se blokira i ne izvršava nikakvu DOS operaciju. Ova situacija se raspoznaje tako što čekate da vam računar vrati kontrolu, a ne dešava se ništa. U takvoj situaciji treba pokušati prekinuti invalid program, istovremenim pritiskanjem tastera CTRL+C ili CTRL+BREAK. U većini slučajeva računar će prekinuti program i ispisati standardni promt (vratiti kontrolu). Ukoliko to ne uspe, mora se resetovati računar. Sa resetovanjem se uništavaju svi podaci smešteni u operativnojoj (RAM) memoriji (programi i podaci na hard-disku i ostalim eksternim memorijama ostaju bezbedni). Postoje dve vrste resetovanja:

- softversko i
- hardversko.

Kod softverskog resetovanja treba istovremeno pritisnuti tri tastera **Ctrl+Alt+Del.** Ovom prilikom disk ne gubi napajanje pa nema opasnosti da dođe do havarije. Ako ovo ne pomogne, sistem se mora hardverski resetovati.

IBM PC računari imaju ugrađen taster "RESET" na svom kućištu, čijim se pritiskom vrši hardversko resetovanje. Dolazi do ponovnog podizanja računarskog sistema isto tako kao kada se računar uključuje. Ako ni ovo ne pomogne, onda računar treba isključiti i sačekati 30-tak sekundi do ponovnog uključenja. Ova pauza je potrebna da bi se hard-disk potpuno umirio. Ovim se izbegavaju havarije sa diskovima i gubitak svih podataka koji se na njima nalaze. Resetovanje treba izbegavati jer je ova operacija suviše radikalna i može dovesti do nezatvorenih ili pogrešno zatvorenih datoteka i gubitaka jednog dela podataka.

LOGIČKA IMENA DISKOVA

Jedinicama spoljne memorije (diskovi, disketne jedinice, CD-ROM i td.) operativni sistem dodeljuje imena, zavisno kako su pripojeni na karticu drajva. Rezervisana imena za diskete su A: i B: (dvotačka je sastavni deo imena). Logička imena C:,D:,E: itd. su rezervisana za hard-diskove i njihove particije. Pri uključenju računara tekuća disk jedinica je ili A: ili C: jer se samo sa ove dve može podići operativni sistem.

Prelazak sa jedne logičke jedinice na drugu vrši se ukucavanjem imena te disk jedinice.

PERIFERIJE

IBM PC računar opremljen je tastaturom, monitorom i spoljašnjom memorijom koja se obično svodi na disk, CD i disketne jedinice. Na IBM PC računar se može priključiti i štampač. Za štampač, bez obzira na način pripajanja na računar, rezervisano je ime PRN. CON- predstavlja konzolu (i monitor i tastaturu), NUL- je fiktivni uređaj koji fizički ne postoji. Kad se računar obraća uređaju NUL-sistem "glumi" da stvarno prenosi podatke, mada će isti biti izgubljeni. Ovo je pogodno za testiranje programa. Sve pomenute reči (A:, B:, C:, PRN:,CON: i neke druge poput COM1:,COM2:,LPT1:,LPT2:...) su rezervisane i ne smeju se koristiti kao imena korisnikovih datoteka ili kataloga. Razlog je u tome što se ime periferije može navesti umesto imena datoteke.

ORGANIZACIJA DISKA

Da bi operativni sistem znao gde se nalaze pojedine vrste informacija, on disk organizuje na način koji mu omogućava lak pristup svakoj lokaciji na disku i praćenje gde se koji podatak nalazi i koliko slobodnog prostora na disku preostaje za buduće podatke koje treba smestiti. Površina hard-diska (diskete) se deli na sektore i tragove. Od prvog traga se smešta deo operativnog sistema (programa) neophodnih za podizanje sistema. Tako je kod formatiziranja diska neophodno navesti opciju kojom će program za formatiranje izvršiti rezervaciju prostora za smeštaj sistemskih datoteka IO.SYS i MSDOS.SYS.

Hard diskovi se mogu deliti na više particija različitog kapaciteta, čiji je ukupni zbir kapaciteta jednak kapacitetu samog diska. Ovo se čini pomoću specijalnog programa FDISK.EXE koji se dobija u paketu sa operativnim sistemom DOS. Svakoj particiji dodeljuje se jedno logičko ime. Ako se disk deli na particije, prva particija je obavezno butabilna (na njoj se nalazi operativni sistem za podizanje i upravljanje računarom) i ona ima logičko ime C:.

Operativni sistem koristi dve tabele (također snimljene na disku), registratore u kojima se vode podaci o rasporedu delova datoteka po površini diska. Ove tabele se nazivaju FAT-tabelama (File Allocation Table), od kojih je jedna rezervna tabla i koristi se u slučaju uništavanja originala. Ove tabele se obavezno ažuriraju prilikom svake promene (snimanja ili brisanja, odnosno promene veličine neke datoteke). U FAT tabeli se vodi spisak kataloga, datoteka, tragova i sektora gde su smešteni pojedini delovi datoteka.

DATOTEKE

Datoteke služe za smeštaj podataka i programa. Datoteka (engl. File) je skup informacija srodnog sadržaja. Datoteka se smešta ili u operativnu memoriju (ponekad samo neki njeni delovi) ili na disku, disketi, magnetnoj traci i dr. Svaka datoteka (bilo da je u njoj sadržan program ili podaci) ima svoje ime i proširenje (ekstenziju) imena. U OS DOS imena datoteka se zapisuju po formatu 8.3 (najviše osam karaktera za ime i maksimalno tri za ekstenziju):

IME.EXT

Ime datoteke uglavnom opisuje koja je grupa podataka sadržana u njoj, a ekstenzija ukazuje na vrstu datoteke. Postoje službene ekstenzije imena datoteke koje se po pravilu koriste za označavanje najčešćih tipova datoteka koje DOS prepoznaje. Najvažniji su:

- *.COM i *.EXE fajlovi su izvršni programi, ukucavanjem imena te datoteke i tipke ENTER dolazi do izvršenja programa,
- *.SYS datoteka sadrži parametre okruživanja u kojem računar radi ili neke drajvere za upravljanje perifernim uređajima priključenim na računar,
- *.TXT i *.DOC su tekstualne (ASCII) datoteke i dokumenti,
- *.ASM, *.PAS, *.C...su službene ekstenzije datoteka koje sadrže izvorne programe od kojih programski prevodioci (kompajleri) prave izvršne verzije programa.

Nazivi datoteka i ekstenzije se prave od slova, brojki i specijalnih znakova (“!”, “@”, “#”, “$”, “&”, “-“, “+”,i “-“). Naziv može imati jedan od osam karaktera, a ekstenzija nijedan, jedan, dva ili tri karaktera. Ekstenzija se od naziva odvaja tačkom (npr. PISMO.DOC). Za ekstenziju ne smemo koristiti rezervisana imena COM, EXE, SYS, BAT i td. za imenovanje datoteka (osim ako iste nisu upravo tog tipa).

Informacije u jednoj datoteci su obično struktuirane. Struktuiranost se ogleda u podeli na slogove i polja. Skup više slogova čini entitet. Više entiteta i njihove međusobne relacije čine bazu podataka. Datoteke mogu biti i nestruktuirane (npr. pisma, izveštaji i dr.).

Pre upotrebe bilo koje datoteke, ista se mora učiniti dostupnom. U tom cilju, zavisno od medija na kome je datoteka smeštena, mora se izvršiti standardno ispitivanje i kontrola. Ovo se naziva *otvaranje* datoteke. Na kraju rada sa datotekom ista se mora *zatvoriti*.

Organizacija datoteke predstavlja fizički raspored podataka jedne datoteke na nekom nosiocu podataka (mediju). Ona prvenstveno zavisi od vrste nosioca, a može se podeliti u tri osnovne grupe:

- sekvencijalna,
- direktna i
- indeksna organizacija.

Sekvencijalna organizacija datoteka može se strukturno predstaviti sekvencijalnom linijskom listom u kojoj su elementi slogovi datoteke, a moguće ju je realizovati na svim vrstama medija. Njena prednost u odnosu na druge oblike organizacije je u jednostavnosti i ekonomičnosti zauzimanja memorijskog prostora.

Kod direktne organizacije podataka slogovima se pristupa u memoriji računara preko adresa, koje su rezultat transformacije njihovih ključeva. Direktna organizacija omogućava veliki stepen pristupljivosti i izmenljivosti i može se realizovati samo na adresabilnim memorijskim medijima (operativna memorija, disk, CD i disketa).

Kod indeksne organizacije podataka slogovima datoteke se pristupa pomoću indeksa. Slogovi mogu biti poređani u proizvoljnom poretku. Indeks se sastoji od dva dela, ključa i adrese na kojoj se nalazi odgovarajući slog. Koristeći ovaj tip organizacije može se sortiranjem indeksa po ključu postići pristup slogovima u željenom redosledu, a da pri tom fizička lokacija slogova ostane nepromenjena. Indeksnu datoteku karakteriše proizvoljna raspoređenost slogova fajla na nosiocu podataka.

Pristup datoteci predstavlja način traženja memoriske lokacije radi prenošenja podatake iz memorije u procesor i obrnuto. Ovde razlikujemo tri načina pristupa:

- sekvencijalni,
- direktni i
- pristup preko ključa.

Kod sekvencijalnog pristupa se traženje nekog sloga vrši počev od prvog u nizu. To znači da je vreme pristupa nekom slogu utoliko duže ukoliko je on bliži kraju datoteke.

Kod direktnog pristupa brzina nalaženja nekog sloga ne zavisi od mesta na kojem je isti smešten. U idealnom slučaju vreme pristupa svakom slogu je konstantno (ovaj idealni oblik je moguć samo ako se datoteka nalazi u operativnoj memoriji računara). Kod memorije sa poludirektnim pristupom, kao što su disk, CD i disketa, slogu sortirane sekvencijalne datoteke se pristupa kombinovano, direktno se pristupa cilindru (tragu), a indirektno (pretraživanjem) se nalazi slog na tom tragu.

Kod pristupa preko ključa, slogu se na osnovu vrednosti njegovog ključa moguće pristupiti u raznim vidovima pretraživanja. Pri tome je od značaja da li je datoteka sortirana ili ne, po određenom sortnom polju.

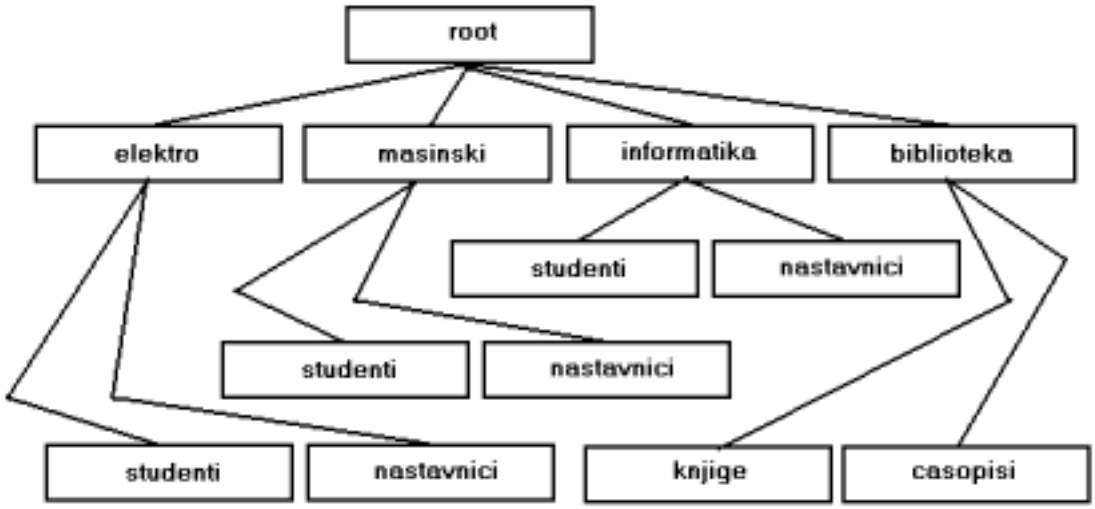
DIREKTORIJI I PODDIREKTORIJI (KATALOZI)

Direktoriji (katalozi) se koriste za potpuno imenovanje i razlikovanje datoteka. U principu, direktoriji predstavljaju skup srodnih datoteka. Veliki kapacitet diska (diskete) nije preporučljivo (a ponekad je i nemoguće) organizovati kao jedan jedini direktoriji. Prvo, pristup datotekama bio bi sporiji, a otežano bi bilo i pronalaženje željene datoteke. Direktoriji se organizuju u hijerarhijsku strukturu tipa stabla (Slika 2.). Poznato nam je već da ime datoteke sadrži dve komponente: naziv i tip. Osim toga postoji i treća komponenta-oznaka diska i direktorija u koji je datoteka zapisana. Dve datoteke A:NELE.TXT i C:NELE.TXT nemaju isto ime zato što se nalaze na različitim medijima (mada njihov sadržaj može čak biti i isti, a ne mora).

Oznaka kataloga se piše između oznake diska i naziva datoteke. Katalog se obično definiše u odnosu na osnovni ROOT (koreni) katalog. Ako hijerarhiju kataloga zamislimo kao stablo, ROOT je koren toga stabla. Prilikom uključenja računara i podizanja sistema ROOT je početni katalog sa kojeg računar polazi, a nalazi se na disku sa koje je sistem podignut (A: ili C:). Osnovni katalog može da sadrži datoteke i druge kataloge. Ti drugi katalozi također mogu da sadrže datoteke i kataloge. Preporučuje se da dubina stabla ne prelazi četiri (radi brzine pretraživanja).

Od kvaliteta kreiranja kataloga zavisi racionalan rad računara. Osnovni katalog ne sme biti prepun. Preporučuje se da *osnovni katalog* sadrži samo datoteke potrebne za

podizanje sistema i uređenja okruženja (konteksta) mašine (npr. IO.SYS, MSDOS.SYS, COMMAND.COM, CONFIG.SYS i AUTOEXEC.BAT). *Drugi nivo* kataloga također ne treba biti prenatrpan. U *treći nivo* treba smestiti veći broj datoteka, pošto četvrti nivo retko postoji.



Slika 4.: Hijerahijska organizacija direktorijuma (stabla)

U DOS-u imamo dva džoker-znaka:

- ”?” znak zamenjuje bilo koji dozvoljeni karakter, a znak
- “*” zamenjuje grupu karaktera u imenu i ekstenziji fajla.

DOS KOMANDE

OZNAKE

disk - logičko ime diska (disketne jedinice) nad kojom se komanda izvršava.

staza - puni put (puno ime) podkataloga na kojem se nalazi datoteka/datoteke na koju/koje se komanda odnosi.

datoteka - je ime datoteke nad kojom se komanda izvršava.

/ - predstavlja oznaku opcije iza koje sledi veliko slovo koje pobliže određuje način izvršenja komande.

n - prirodan pozitivan broj.

[ ] - par srednjih zagrada znači da se sadržaj unutar ovih zagrada može (a ne mora) ukucati. Izostavljanjem ovog sadržaja nećete narušiti sintaksu komande, ali ćete umanjiti neke mogućnosti te komande.

| - specijalni karakter “pipe line” naznačava da se izborom jedne opcije (grupe opcija) isključuje upotreba ostalih opcija i da kombinacija međusobno isključujućih opcija nije moguća.

Prelazak sa tekućeg diska i tekućeg direktorija na tekući direktoriji diska se vrši unošenjem logičkog imena diska/diskete (bez komande) i pritisne se taster ENTER.

KOMANDE ZA RAD SA DISKOVIMA

Komande pišemo velikim a iza komande sledi niz raznih parametara (ako ih komanda ima).

DISKCOPY
DISKCOPY disk 1:disk 2:
Kopira kompletan sadržaj izvorne (disk1:) diskete na odredišnu (disk 2:) disketu.

FDISK
Konfiguriše hard-disk za upotrebu pod MS-DOS-om. Ova komanda deli hard disk na particije koje mogu da se koriste za razne operativne sisteme ili za smeštanje istorodnih grupa podataka.
FDISK[/STATUS]
/STATUS –Prikazuje informaciju o trenutnim participacijama.

FORMAT
Formatira disk ili disketu. Ako su disk ili disketa već formatirani i na njima se nalaze podaci i programi, PONOVNO formatiranje UNIŠTAVA ranije podatke. Vraćanje podataka vrši se pomoću komande UNFORMAT.

SUBST
Pridružuje stazi logičko ime diska. Na ovaj način se fiktivno kreira novi logički disk, koji se u suštini svodi samo na jedan katalog.

SYS
Prenosi operativni sistem na drugi disk ili disketu. Ovom komandom se skrivene datoteke IO.SYS i MSDOS.SYS kopiraju u BOOT sektor formatiranog diska ili diskete.
SYS [disk 1:] [staza] disk 2:
[disk 1:] [staza] Određuje lokaciju sistemskih datoteka.
Disk 2: Određuje disk na koje se datoteke trebaju kopirati.

KOMANDE ZA RAD SA DIREKTORIJIMA

CD (CHDIR)
Prikazuje ime tekućeg direktorija ili menja tekući direktoriji.
CHDIR [disk 1:] [staza]
CHDIR[..]
CD[disk:] [staza]
CD[..]

DELTREE
Briše direktorije i sve poddirektorije i datoteke unutar.
Za brisanje jedne ili više datoteka i direktorija:
DELTREE [/Y] [disk:] staza [[disk:] staza [...]]

/Y Isključuje poruke za potvrđivanjem da li želite brisati poddirektorije.
[disk:] staza Određuje ime direktorija koji želite izbrisati.

MD (MKDIR)
Kreira direktorije. Ime direktorija takodjer može imati format 8:3. Ukupna dužina imena direktorija (staza+ime) nesme preći 63 znaka, računajući i crtu"\" (backslash).
MKDIR [disk:] staza
MD [disk:] staza

RD (RMDIR)
Uklanja (briše) direktoriji.
RMDIR [disk:] staza
RD [disk:] staza

TREE
Prikazuje grafičku strukturu direktorija diska ili staze.
TREE [disk:] [staza] [/F] [/A]
/F Prikazuje imena datoteka u svakom direktoriju.
/A Koristi ASCII umesto proširenih karaktera.

KOMANDE ZA RAD SA DATOTEKAMA

ATTRIB
Prikazuje ili menja atribute datoteka.

+	Postavlja atribut.
-	Briše atribut.
R	Read-only (samo učitavanje) atribut datoteke.
A	Archive (arhivska) atribut datoteke.
S	System (sistemska) atribut datoteke.
H	Hidden (skrivena) atribut datoteke.
/S	Procesira datoteke u svim direktorijima u zadatoj stazi.

COPY
Kopira jednu ili više datoteka na drugu lokaciju.
COPY [/A | /B] izvor [/A|/B] [+ izvor[/A|/B] [+ ...]] [odredište[/A|/B]] [/V] [/Y|/-Y]

Izvor	odredjuje datoteku ili datoteke za kopiranje.
/A	ukazuje na ASCII tekstualnu datoteku.
/B	ukazuje na binarnu datoteku.
Odredište	odredjuje direktoriji ili datoteku za novu(e) datoteku(e).
/V	verifikuje da li su nove datoteke pravilno zapisane.
/Y	isključuje prompt za potvrdjivanje da se želi upisivati preko postojeće odredišne datoteke.
/-Y	uključuje prompt za potvrdjivanje da li se želi upisivati preko postojeće odredišne datoteke.

DEL
Briše jednu ili više datoteka. Ova komanda je prilično opasna jer nesavesno i pogrešnom upotrebom iste može doći do gubitaka podataka.
DEL [disk:] [staza] datoteka [/P]

DIR

Prikazuje listu datoteka i poddirektorija u direktoriju.

DIR[disk:] [staza] [datoteka] [/P] [/W] [/A[[:] atributi]] [/O[[:] sort-red]] [/S] [/B] [/L] [/C[H]

[disk:] [staza] [datoteka] odredjuje disk (katalog), ili datoteka za izlistavanje.

/P	pauzira nakon svakog popunjenog ekrana.
/W	koristi široki format za izlistavanje.
/A	prikazuje datoteke sa zadatim atributima.
Atributi	D Katalozi, R Read-only datoteke, H Skrivene datoteke, S sistematske datoteke, A arnivske datoteke,"-"prefiks znači"ne".
/O	lista sa sortiranim redosledom.
/S	prikazuje datoteke u zadatom direktoriju i sve poddirektorije.
/B	koristi go(bare) format (bez naslova ili skupnog pregleda).
/L	koristi mala slova.
/C[H]	prikazuje stepen kompresije;/CH koristi HOST veličinu alokacione jedinice.

FIND

Pronalazi niz simbola u datoteci ili datotekama.

FIND [/V] [/C] [/N] [/I]"string"[[disk:] [staza]datoteka[...]]

/V	prikazuje sve linije koje ne sadrže zadati string.
/C	prikazuje samo broj linija koje sadrže traženi string.
/N	prikazuje brojeve linija sa prikazanim linijama.
/I	ignoriše da li je veliko ili malo slovo unutar traženog stringa.
	"string" odredjuje tekst string za pretraživanje.

[disk:] [staza] datoteka -odredjuje datoteku ili datoteke za pretraživanje.

Ako staza nije zadata, FIND traži tekst ukucan na promptu ili usmeren sa druge komande.

MOVE

Seli datoteke i preimenuje datoteke i direktorije.

Za preselenje jedne ili više datoteka:

MOVE [/Y | /-Y] [disk:] [staza] datoteka1 [,...] odredište

Za preimenovanje direktorija:

MOVE [/Y | /-Y] [disk:] [staza] direktoriji1 direktoriji2

[disk:] [staza] datoteka1	Određuje lokaciju i ime datoteke ili datoteka koje želi preseliti.
odredište	Određuje novu lokaciju datoteke.
[disk:] [staza] katalog1	Određuje direktorija koji želite preimenovati.
katalog2	Određuje novo ime direktorija.
/Y	Isključuje promptno javljanje za potvrđivanjem kreiranja direktorija ili prepisivanja preko zadatog odredičta
/-Y	Uključuje promptno javljanje za potvrđivanjem kreiranja direktorija ili prepisivanja preko zadatog odredišta.

PRINT
Štampa tekstualnu datoteku dok se za to vreme mogu koristiti ostale MS-DOS komande.
PRINT [/D:uređaj] [/B:kapacitet] [/U:br-taktova1] [/M:br-taktova2] [/S:br/taktova3] [/Q:qkapacitet] [/T] [[disk:] [staza] datoteka [...]][/C][/P]

/D:disk	Određuje uređaj za štampanje.
/B:kapacitet	Postavlja interni kapacitet bafera, u bajtima.
/U:br-taktova1	Čeka zadati maksimalni broj taktova kloka dok printer ne bude na raspolaganju.
/M:br-taktova2	Određuje maksimalni broj taktova kloka za štampanje jednog karaktera.
/S:br-taktova3	Dodeljuje scheduler-u zadati broj taktova kloka za štampanje u pozadini.
/Q:qkapacitet	Određuje maksimalni broj datoteka koje mogu biti u listi čekanja (queue)za štampanje.
/T:	Uklanja sve datoteke iz lisze čekanja za štampanje (print queue).
/C	Odgađa štampanje datoteke u procesu i svih sledećih datoteka.
/P	Dodaje datoteku i ostale koje slede nakon nje u listu čekanja za štampanje.

REN (RENAME)
Promena imena datoteke.
RENAME [disk:] [staza] datoteka1 datoteka2
REN [disk:] [staza] datoteka1 datoteka2

TYPE
Ispisuje na ekranu sadržaj datoteke.

XCOPY
Kopira datoteke (osim skrivenih i sistematskih datoteka) i stablo direktorija.
XCOPY izvor [odredište] [/A | /M] [/D:datum] [/P] [/S] [/E] [/V] [/W]

izvor	Određuje datoteku za kopiranje.
odredište	Određuje lokaciju i(ili) ime nove datoteke.
/A	Kopira datoteke sa atributom "archive", ne menja atribut.
/M	Kopira datoteke sa atributom "archive", isključuje atribut "archive.
/D;datum	Kopira datoteke promenljive na ili nakon zadatog datuma.
/P	Prompt pre kreiranja svake odredišne datoteke.
/S	Kopira direktorije i poddirektorije osim praznih.
/E	Kopira sve poddirektorije, čak i prazne.
/V	Verifikuje svaku novu datoteku.
/W	Prompt da se pritisne neki taster pre kopiranja.
/Y	Isključuje prompt za potvrđivanje da želite prepisati preko postojeće odredišne datoteke.
/-Y	Uključuje prompt za potvrđivanje da želite prepisati preko postojeće odredišne datoteke.

POMOĆNE KOMANDE

CLS
Brisanje ekrana.

DOSSHEEL
Pokreće MS-DOS-ov Šel(shell). Šel je alternativni komandni interpreter, koji može da zameni interpreter COMMAND.COM.

GRAPHICS
Učitava program koji može štampati grafiku.

MEM
Prikazuje količinu zauzete i slobodne memorije u sistemu.

MSAV
Pretražuje računar tražeći poznate viruse.

MCD
Daje detaljne tehničke informacije o računaru.

VER
Prikaz verzije DOS-a.

VSAFE
Vodi računa o pojavi virusa na računaru i prikazuje upozorenjekad pronađe neki.

OPERATIVNI SISTEM *MS-WINDOWS*

- *Microsoft* MS Windows (prozor) operativni sistem namenjen je različitim računarskim sistemima:
- personalnim PC računarima,
- lokalnim i globalnim računarskim mrežama,
- superserver-višeprocesorskim računarima baziranih na RISC procesorima,
- itd.

MS Windows se kao operativni sistem pojavio osamdesetih godina. Većina korisnika računara je u to vreme koristio personalne računare. Najpopularniji operativni sistem je bio MS DOS.

Prve verzije Windows-a su u stvari predstavljale programski sistem koji, ne samo da se oslanja na MS DOS, već predstavlja sistem aplikacija koji se startuje iz samog MS DOS-a sa komandne linije. Novi operativni sistem-Windows trebalo je da predstavlja platformu pod kojom bi mogli da se izvršavaju mnogobrojni programi pisani u MS DOS-u (potpuna DOS kompatibilnost). Pored toga ovi programi su već omogućili i više:

- unificiran grafički korisnički interfejs,
- potpuno korišćenje mogućnosti savremenih procesora,
- efikasnije korišćenje operativne memorije,
- dinamičku razmenu podataka,

- virtuelni memorijski sistem,
- višeprogramski rad,
- podršku za multi mediju itd.

Prva verzija ovog operativnog sistema Windows 3.0. je prodata u preko četiri miliona primeraka. Poboljšane verzije su Windows3.1 i Windows 3.11. Danas je Windows najprodavaniji operativni sistem.
Za različite računar postoje sledeće verzije MS Windows-a:
Windows 3.1 za IBM PC 386, 486, Pentium i za prenosne računare

Windows 95, 98 kao i 2000 predstavlja 32-bitnu verziju operativnog sistema namenjenu računarima sa nešto boljim karakteristikama i sa većim memorijskim zahtevima. Ova verzija predstavlja kompletan operativni sistem koji se definitivno odvaja od MS DOS-a, ali ga i dalje podržava.

Windows NT-operativni sistem je za snažne radne stanice i mrežne servere.

Kod MS Windows operativnog sistema razlikujemo tri modula:
- modul za opštu podršku-kernel,
- grafički modul (GDI),
- korisnički modul-za upravljanje prozorima.

WINDOWS NT

Windows NT (*New Tehnology*), je 32-bitni operativni sistem kreiran da u budućnosti bude osnova za sve PC računare, radne stanice i servere.

Pored osobina prethodnih verzija, ovaj operativni sistem ime sledeće prednosti:
- mogućnost rada na različitim hardverskim platformama,
- rad u računarskim mrežama,
- distribuirana obrada podataka,
- rad sa dosadašnjim DOS i Windows aplikacijama,
- veza sa UNIX aplikacijama,
- OS/2 kompatibilnost,
- visoka stabilnost,
- rad u višeprocesnim sistemima.

Programi pod Windows NT mogu da se izvršavaju u dva osnovna moda:
- kernel mod-kad se izvršava sistemski kod i
- korisnički mod-kad se izvršavaju aplikacije.

API (*Application Programming Interface*) je deo strukture operativnog sistema Windows NT, i predstavlja programske alatke koje omogućavaju razvoj aplikacija nezavisnih od platforme.

APLIKACIONI SOFTVER

Aplikacioni program je specifičan za određenu primenu računara. Aplikacione programe obično razvijaju korisnici računarskog sistema za rešavanje svojih problema. Zbog toga se ovi programi često nazivaju i korisnički programi.

Najvažnije oblasti primene računara, odnosno aplikacionih programa su:

- obrada teksta,
- računska tehnika,
- komunikacija podataka,
- poslovne primene,
- automatizacija tehnoloških procesa,
- naučno tehničke primene,
- primene u obrazovanju,
- veštačka inteligencija itd.

Aplikacione programe možemo podeliti i na sledeći način:

- pojedinačni aplikacini progami,
- programski paketi,
- sistemski programi.

Programski paket sastoji se od više programa i predstavlja skup procedura za rešavanje problema.

Sistemski program predstavlja skup programa čije su komponente usko povezane.

UNIX

OPERATIVNI SISTEM

Operativni sistem je skup programa koji upravljaju resursima računara. Resurse računara možemo podeliti na sklopovske (hardware) i programske (software). Kao primer hardverskih resursa možemo navesti:

- tastature,
- štampače,
- zvučne kartice itd,

dok u programske resurse spada:

- raznovrsna programska podrška koja na računaru izvodi specifične zadatke.
-

Sam operativni sistem (OS) može biti:

- jednokorisnički (single-user) ili
- višekorisnički (multi-user).

Kod jednokorisničkih OS-ova samo jedna osoba može izvršavati samo jedan proces (program ili deo programa). Za jednokorisnički OS koji može izvršavati i više od jednog procesa u istom vremenu kažemo da je višezadatni (multi-tasking).

Nasuprot jednokorisničkim OS-ovima imamo višekorisničke OS-ove kod kojih (kao što im i ime kaže) više korisnika može raditi istovremeno. Višekorisnički OS-ovi su uvek višezadatni (naime svaki korisnik može pokrenuti vlastitu aplikaciju).

Jednokorisnički OS-ovi su DOS, Windowsi 3.xx, dok su Windowsi 95 i NT uz to i višezadatni (Windowsi NT se mogu svrstati i u višekorisničke OS-ove ali ne u doslovnom smislu - ne može biti logirano više korisnika koji mogu pokretati aplikacije na računaru). Glavni predstavnik višekorisničkih sistema je UNIX (u svojim raznim realizacijama).

VERZIJE UNIX-A

Postoji više verzija UNIX-a. Prvi UNIX nastao je daleke 1969 godine kao istraživački proizvod američke firme AT&T Bell Labs. Danas su najvažnije verzije UNIX-a "System V", BSD (nastala 1977 na Kalifornijskom Univerzitetu Berkeley - zadnja verzija BSD4.4). Osim toga postoji Xenix (UNIX mikroračunarska verzija popularnog AT&T UNIX-a verzije 7), POSIX (standard za UNIX slične operativne sisteme). Osim tih verzija danas su poznate (i popularne) sledeće verzije UNIX-a: SunOS, Solaris, SCO UNIX, AIX, HP/UX, ULTRIX i najnoviji Linux.
Ono što će biti prikazano kroz sledeća poglavlja opštenito je za sve verzije UNIX-a, tako da će se usvojena znanja moći koristiti na bilo kojoj od pre navedenih verzija.

RAD NA UNIX RAČUNARU

Da bi uopšte mogli početi raditi na UNIX računaru potrebno je da nam sistem administrator otvori korisnički račun (account). Otvaranjem korisničkog računa administrator definiše korisničko ime (username), dozvole koje ima korisnik na

sistemu, dodeljuje prostor na disku, definiše E-mail adresu i tajnu korisničku lozinku (password).

Postoji više načina putem kojih korisnik može pristupiti do svojih resursa na sistemu:

- Možemo se spojiti preko konzole (konzola je ulazno-izlazna jedinica - "tastatura i ekran" - direktno spojena na UNIX računar i ima povlašćeni status na računaru - neke radnje vezane za operativni sistem mogu se izvesti samo preko konzole - sve one vezane za single user mod rad UNIX-a),
- preko terminala (obično udaljenog) ili emulacije terminala (ukoliko se spajamo sa nekog drugog operativnog sistema). Za svaki od tih načina spajanja na UNIX računar zajednički je proces identifikacije.

PROCES IDENTIFIKACIJE

Da bi računar (UNIX OS) saznao ko je onaj ko se želi spojiti na njega, da li ima ovlašćenja da se spoji i koje resurse da mu dodeli, on provodi prilikom svakog spajanja proces identifikacije (logging in). Proces identifikacije se sastoji u unošenju ličnog korisničkog imena i lozinke. U trenutku kada se želite spojiti na računar (na jedan od pre spomenutih načina) UNIX OS će startati proces identifikacije i napisati vam da od vas traži da unesete korisničko ime (login). Nakon unošenja korisničkog imena od vas se traži da unesete i korisničku lozinku:

UNIX(r) System V Release 4.0 (oliver)

login:**ivica**	(nakon unošenja imena treba pritisnuti ENTER)
password:	(unošenje korisničke lozinke ne vidi se na zaslonu - pritisnuti ENTER)

Ukoliko je korisnička lozinka neispravna računar će ispisati (korisnička imena i lozinke su osetljiva na mala i velika slova tako da treba paziti prilikom unosa):

Login incorrect
login:

A ako je ispravno uneseno korisničko ime i lozinka napisati će pozdravnu poruku i tražiti od korisnika da unese tip terminala (unošenje tipa terminala nije obavezno na drugim računarima).

Last login: Mon Feb 17 15:03:15 from vts.su.ac.yu.
TERM= (vt100) (samo pritisnemo ENTER za potvrdu)

Po uspešnom izvršenju procesa identifikacije, računar će ući u program zvan ljuska (shell). Ljuska interpretira sve komande koje korisnik unosi i opštenito se brine za svu komunikaciju između korisnika i UNIX operativnog sistema. Postoji više vrsti ljuski (razlikuju se po sintaksi koja se upotrebljava) a najpoznatije su: Burne, Koren i C ljuske. U trenutku kada je ljuska spremna da primi korisnikovu naredbu ispisati će na ekranu prompt (znak ili niz znakova kojim ljuska označava da je spremna za unos naredbe).

oliver%	(primer prompta)

Kod Bourn i Korn ljuske obično prompt završava na "$", dok kod C ljuske završava na "%".

UNOŠENJE NAREDBI

Unošenjem naredbi u UNIX ljusci "govorimo" računaru šta i kako da radi. Naredbe označavaju imena UNIX programa. Pritiskom na ENTER ljuska interpretira našu naredbu i izvršava određeni program.

Prvu reč koju unesemo uvek je UNIX naredba. Sve naredbe su osetljive na mala i velika slova, tako da nije svejedno kako se pišu. Sama naredba pored samog imena naredbe može sadržavati i dodatne opcije koje ima naredba (opcije koje se nalaze unutar uglastih zagrada su opcionalne i ne moraju se navesti).

naredba [opcije]

Evo primera nekoliko UNIX naredbi:

oliver% date
Wed Mar 5 11:11:27 MET 1997 (ispisuje trenutno vreme i datum)
oliver% who
ppero pts/3 Mar 5 10:51 (vts.su.ac.yu)
anovak pts/2 Mar 5 10:02 (yunord.net)
srodic pts/4 Mar 5 10:42 (tippnet.co.yu)
mlakic pts/1 Mar 5 10:28 (vts.su.ac.yu)(ispisuje ko trenutno radi na računaru)
oliver% who am i
srodic pts/4 Mar 5 10:42 (yunord.net) (ispisuje ko je logiran na terminalu koji koristimo)

Promenu prompta koji nam ljuska javlja da je spremna da primi našu naredbu možemo promeniti korištenjem naredbe

set prompt='Tvoj rob% '

ZAVRŠETAK RADA

Kada želimo završiti rad na UNIX računaru ne gasimo jednostavno terminal (ili aplikaciju koja emulira terminal - telnet) nego izvršavamo posebnu proceduru (logging out). Da bi pokrenuli proceduru za prestanak rada na UNIX računaru ljusci zadamo naredbu exit (na većini sistema je također važeća naredba i logout) i time smo pokrenuli procedure za završetak rada (po završetku te procedure možemo ugasiti terminal).

KORISNIČKE LOZINKE

Danas kada su skoro svi UNIX računari povezani u jednu veliku mrežu (Internet)

naročito je važan pravilan odabir korisničke lozinke. Korisnička lozinka nikao ne bi smela biti jednostavna. To znači da za lozinku ne koristimo osobna imena, imena nama poznatih osoba, strane reči. Najbolje lozinke su one koje u sebi imaju kombinaciju malih i velikih slova te brojeva. Evo nekoliko primera dobrih i loših korisničkih lozinki:

- ivica -loša lozinka
- windows -loša lozinka
- dv43i3ma -dobra lozinka
- blMK23i4 -dobra lozinka

Da bi promenili lozinku u UNIX ljusci izvodimo komandu **passwd**. Ona od nas traži da unesemo staru lozinku (provera da li smo to stvarno onaj za koga se izdajemo a ne da je naš kolega na trenutak izašao a mi iskoristili njegov uključeni terminal da bi mu promenili lozinku), te dva puta novu (drugi put radi provere da nismo napravili grešku prilikom ukucavanje - unos lozinke ne vidi se na ekranu).

Old password: (unosimo staru lozinku)
New password:
Re-enter new password: (dvaput unosimo novu lozinku)

Lozinka mora imati najmanje 6 znakova (obično osam) od kojih su najmanje dva alfabetski znaci i jedan numerički.

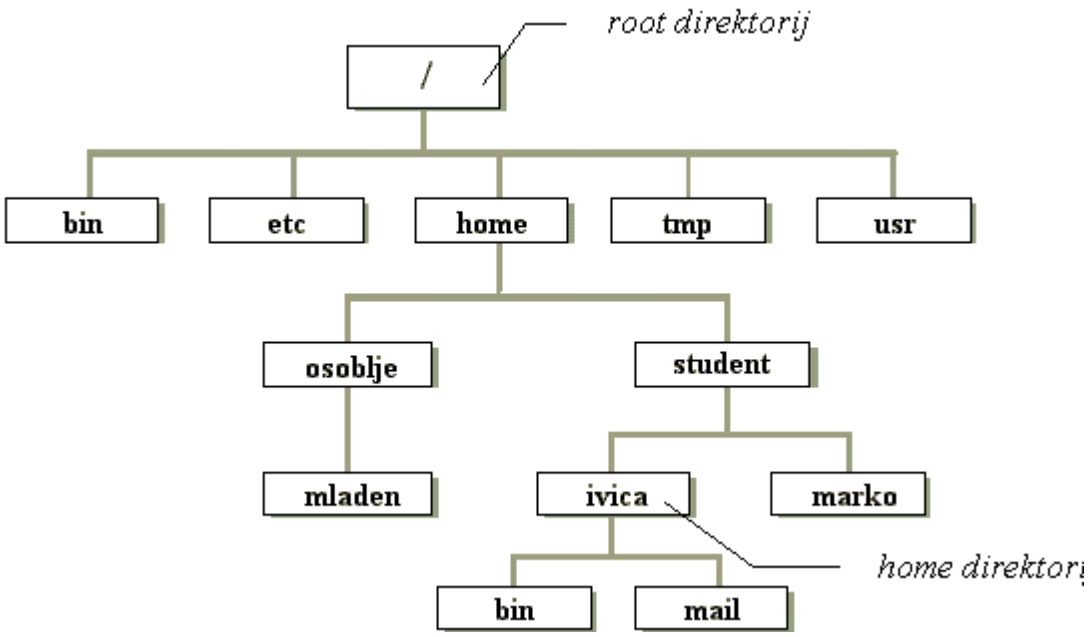
UNIX DATOTEČNI SISTEM

Datoteka je osnovna jedinica pohrane podataka kako na UNIX-u, tako i na mnogim drugim operativnim sistemima. Datoteka može sadržavati bilo kakav oblik podataka:

- običan tekst,
- program,
- sliku,
- zvučni zapis itd. .

Možemo reći da je datoteka sekvencijalni niz podataka, dok sama interpretacija podataka ovisi od programa koji je koriste.

UNIX datoteke su organizovane u direktorije. Direktorij je specijalni deo datoteke u koju operativni sistem sprema podatke o ostalim datotekama (sadržaju tog direktorija). Na Slici 5. možemo videti izgled jednog UNIX datotečnog sistema.



Slika 5.: Izgled jednog UNIX datotečnog sistema

Direktorij koji se nalazi na vrhu datotečnog stabla nazivamo *root direktorij*. Svi drugi direktoriji u datotečnom sistemu su poddirektoriji root direktorija.

HOME DIREKTORIJ

Nakon uspešnog procesa identifikacije (logina) korisnik se nalazi u svom *home direktoriju*. Home direktorij svakog korisnika je jednoznačno određeno mesto u UNIX operativnom sistemu (svaki korisnik ima samo jedan home direktorij) i na tom mestu se nalaze sve inicijalne datoteke potrebne za početnu inicijalizaciju korisnikove ljuske (".login", ".cshrc", ".profile"). Nad svojim home direktorijom UNIX korisnik ima sva ovlaćenja pa tako u njemu može kreirati poddirektorije i spremati datoteke. Neki UNIX sistemi imaju postavljenu kvotu na home direktoriju. To znači da korisnik može spremiti u svoj home direktorij samo određenu količinu podataka definisanu kvotom (ukoliko je kvota 2,5Mb - korisnik može u svoj direktorij staviti najviše 2,5Mb podataka).

RADNI DIREKTORIJ

Radni direktorij je onaj direktorij u kome se trenutno nalazimo. Na početku svakog rada na UNIX operativnom sistemu korisnikov home direktorij je radni direktorij. Tokom rada radni direktorij se može menjati (naredba **cd**). Direktorij u koji se pomaknemo postaje naš radni direktorij.
Sve naredbe koje izvršavamo (ukoliko to drugačije ne odredimo) odnose se na datoteke u našem radnom direktoriju.

DATOTEČNO STABLO

Direktoriji UNIX datotečnog sistema organizovani su u hijerarhijsku strukturu zvanu *datotečno stablo*. Na vrhu te hijerarhijske strukture nalazi se root direktorij koji označavamo sa "/" (**slash**). Na prethodnoj slici (Slika 1.) se mogu videti neki

karakteristični poddirektoriji root direktorija zajednički većini UNIX operativnih sistema:

- u *bin direktoriju* se nalaze sistemski programi,
- u *etc direktoriju* inicijalne datoteke potrebne za rad sistema,
- u *usr direktoriju* korisnički programi, *tmp direktoriju* privremeni podaci i na kraju
- u *home direktoriju* se nalaze korisnički home direktoriji.

Notacija kojom se definše gde se nalazi određena datoteka ili direktorij zove se put (**path**). Put može biti apsolutni ili relativni.

APSOLUTNI PUT

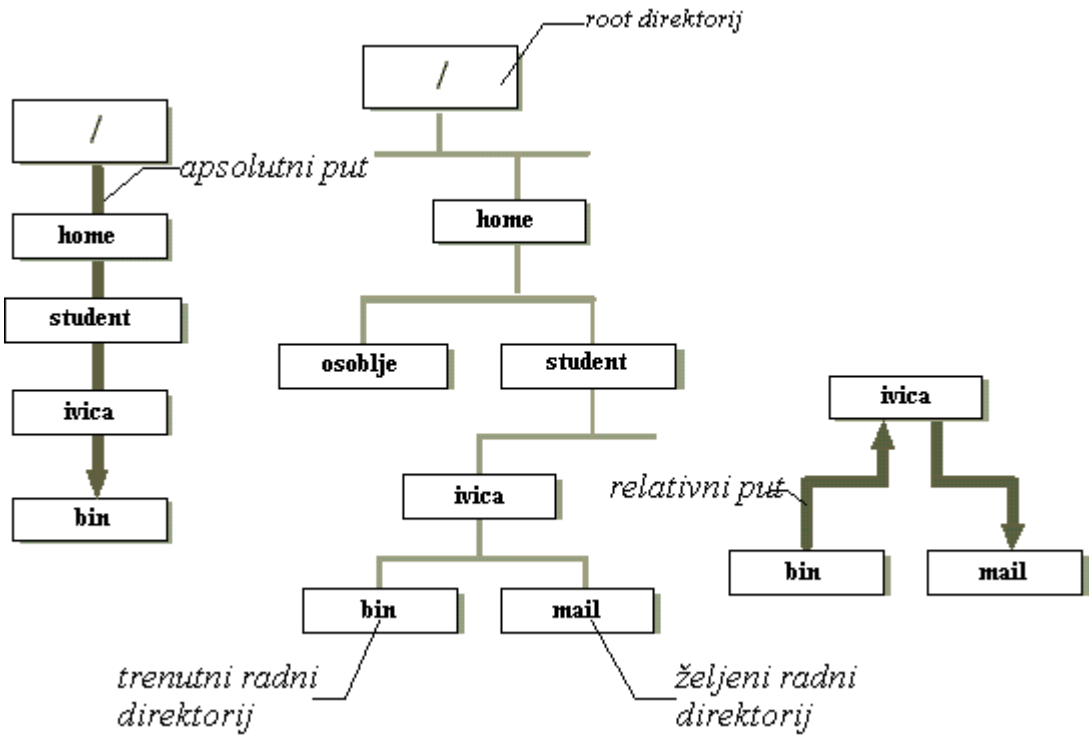
Kao što je rečeno ranije, UNIX datotečni sistem je organizovan kao inverzno stablo sa root direktorijom na vrhu. Apsolutni put govori nam kojim putem moramo proći od root direktorija do direktorija (ili datoteke) do kojeg želimo stići.

Uzmimo da nam je trenutni radni direktorij *bin* i da želimo stići u direktorij *mail*. Apsolutni put kojim moramo proći ide od *root* direktorija, preko *home*, *student*, *ivica* i na kraju *mail*. Skraćeno to zapisujemo kao "/home/student/ivica/mail", dakle navodeći celi put koji moramo proći od *root* direktorija do direktorija *mail* (označavanje apsolutnog puta uvek počinje sa "/" - oznaka za root direktorij).

RELATIVNI PUT

Za razliku od apsolutnog puta koji uvek počinje od root direktorija, kod relativnog puta pozicija željenog direktorija se navodi u odnosu na trenutni radni direktorij. Kod relativnog puta trenutni direktorij označavamo sa "." (jednom točkom) a direktorij više (roditeljski direktorij) u hijerarhiji stabla sa ".." (dve tačke).

Pogledajmo nekoliko primera (Slika 6.). Trenutni radni direktorij nam je **student** i želimo doći do dirketorija **bin** (poddirektorija direktorija **ivica**). Relativnim putem kretati ćemo se iz ***student*** (trenutni radni direktorij) u ***ivica*** pa do ***bin*** (željeni direktorij) ili kraće zapisano "**ivica/bin**". Taj put apsolutno zapisan izgledao bi "**/home/student/ivica/bin**". Želimo li sada iz ***bin*** direktorija prebaciti se u ***mail*** direktorij moramo se kretati od ***bin*** direktorija jedan direktorij gore, preko ***ivica*** direktorija, do ***mail*** ili kraće zapisano "**../mail**". Apsolutni put bi bio "**/home/student/ivica/mail**".



Slika 6.: Primeri za apsolutni i relativni put

PROMENA RADNOG DIREKTORIJA

Poznavajući relativni i apsolutni put direktorija sada se možemo kretati kroz datotečni sistem UNIX-a.

Želimo li odrediti u kom direktoriju se trenutno nalazimo koristićemo naredbu **pwd.** Naredba pwd nema argumenata, a kao rezultat nam daje apsolutni put do trenutnog radnog direktorija.

oliver% pwd
/home/student/ivica
oliver%

Sada želimo li promeniti trenutni radni direktorij u /home/student/ivica/bin koristićemo se naredbom **cd**. Naredba cd kao argument ima apsolutni ili relativni put željenog direktorija.

oliver% cd /home/student/ivica/bin
oliver% pwd
/home/student/ivica/bin
oliver%

Nalazimo li se u direktoriju /home/student/ivica/bin i želimo doći do direktorija /home/student/ivica/mail koristeći relativni put kao argument cd naredbe to ćemo izvesti na sledeći način:

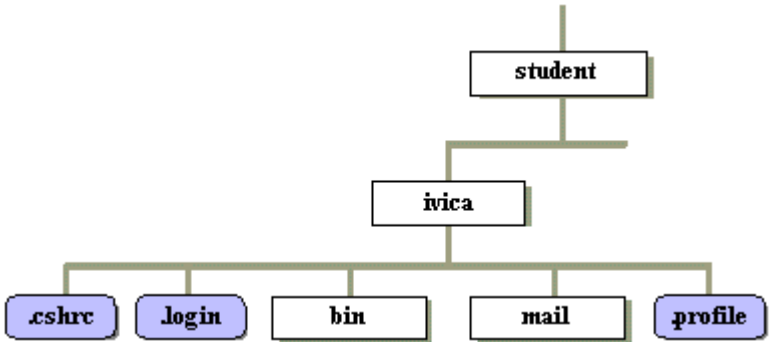
```
oliver% pwd
/home/student/ivica/bin
oliver% cd ../mail
oliver% pwd
/home/student/ivica/mail
oliver%
```

Sa naredbom cd možemo samo promeniti trenutni radni direktorij. Navedemo li kao argument cd naredbe apsolutni ili relativni put do datoteke dobit ćemo poruku o grešci.

```
oliver% cd /etc/hosts
/etc/hosts: Not a directory.
oliver%
```

DATOTEKE U DATOTEČNOM STABLU

Svaki direktorij može sadržavati poddirektorije. Osim poddirektorija direktorij može sadržavati i datoteke. Na Slici 7. možemo videti tipičan izgled jednog korisničkog home direktorija.



Slika 7.: Tipičan izgled jednog korisničkog home direktorija

Da bi videli sadržaj direktorija koristimo naredbu **ls**. Naredba ls će nam dati listu datoteka i poddirektorija radnog direktorija. Ukoliko kao argument ls naredbe navedemo apsolutni ili relativni put dobitćemo sadržaj direktorija zadanog u putu.

IZLISTAVANJE SADRŽAJA DIREKTORIJA

Da bi mogli koristiti naredbu **cd** moramo znati sadržaj direktorija. Sadržaj direktorija možemo dobiti naredbom **ls**. Kada unesemo naredbu **ls** dobijamo popis datoteka i poddirektorija, direktorija koji je trenutno naš radni direktorij. Puna sintaksa naredbe **ls** izgleda ovako:

- **ls** [opcija(e)] [ime-direktorija-ili-datoteke]

Najjednostavnija upotreba je navođenje same naredbe **ls** bez ikakvih parametara. Kao rezultat dobiti ćemo popis datoteka i poddirektorija.

```
oliver% ls
Mail/
```

oliver%

Dodamo li naredbi **ls** opciju *-a* (od engleske riječi all) dobiti ćemo još koju datoteku kao rezultat. Naime opcije *-a* "govori" naredbi ls da prikaže sve datoteke i direktorije (datoteke i direktoriji koji počinju sa "." skriveni su za ls naredbu bez opcije -a).

```
oliver% ls
  .cshrc     .profile
  .login     Mail/
oliver%
```

Kako je ranije objašnjeno "." i ".." su specijalne notacije za radni direktorij (".") a za roditeljski direktorij (".."). Datoteke koje počinju sa "." (tačkom) skrivene su i mogu biti prikazane samo pomoću opcije *-a* (obično su to inicijalne datoteke - datoteka .profile sadrži naredbe koje se automatski izvršavaju od Bourne ili Korn ljuske svaki put kada pristupite sistemu. C ljuska koristi datoteke .login i .cshrc umjesto .profile).

Želimo li dobiti još više informacija o datotekama koristićemo opciju *-l* (od engleske riječi long) i kao rezultat dobiti ispis (Slika 8.):
oliver% ls -al (obe opcije su aktivne -a i -l)

Slika 8.: Rad operacije oliver% ls -al

Tip	Govori nam da li se radi o datoteci (-), direktoriju (d) ili pokazivaču (linku) na drugu datoteku ili direktorij (l).
Dozvola pristupa	Specifikuje tri grupe korisnika (sam vlasnik, grupa kojoj on pripada, i svi ostali korisnici koji ne spadaju u prve dve grupe). Svaka grupa korisnika ima određena ovlašćenja nad datotekom ili direktorijom. Ovlašćenja koja ima određena grupa vidimo po slovima pridruženim određenoj grupi (svaka grupa ima tri polja), a to su r (dozvola čitanja), w (dozvola pisanja) i x (dozvola izvršavanja - kod direktorija dozvola pristupa).
Broj linkova	Pokazuje broj datoteka i direktorija koji su povezani s datotekom.
Vlasnik	Ime vlasnika datoteke ili direktorija (to je uvek

	korisničko ime vlasnika).
Veličina	Veličina datoteke.
Vreme promene	Datum i vreme zadnje promene datoteke.
Ime	Ime datoteke ili direktorija.

Osim ovoga može se dobiti još i grupa kojoj pripada datoteka ili direktorij ako navedemo opciju *-g* (dakle ls -alg). Također kao opciju ls naredbe možemo navesti i apsolutni ili relativni put direktorija čiji sadržaj želimo videti (npr. ls -l /home/vezbe).

DOZVOLE PRISTUPA

Za svaku datoteku ili direktorij na UNIX-u se može postaviti određena dozvola pristupa. UNIX razlikuje tri grupe korisnika:

- vlasnik datoteke,
- korisnici koji pripadaju grupi u kojoj i datoteka,
- te svi ostali (korisnici koji se ne mogu svrstati u jednu od prve dve grupe).

Za svaku od tih grupa postavljaju se dozvole, s time da samo vlasnik datoteke ili direktorija može menjati ovlašćenja.

Postoje tri vrste dozvola:

- dozvola čitanja (kod direktorija listanja),
- dozvola pisanja i
- dozvola izvršavanja (kod direktorija dozvola ulaska u direktorij).

Svaka od tih dozvola se označava slovom:
r (čitanje),
w (pisanje) i
x (izvršavanje-ulazak).

Ako ne postoji dozvola za određenu grupu korisnika onda na tom mestu stoji "-" (dozvole možemo videti sa ls -l).

Pogledajmo par primera ako:
prvo polje se odnosi na vlasnika,
drugo na korisnike koji spadaju u grupu koja ima ovlasti nad datotekom, a
treće na ostale korisnike):

rwx r-- r--	vlasnik ima sve dozvole, grupa i svi ostali dozvolu čitanja
rwx rw- r--	vlasnik ima sve dozvole, grupa dozvolu pisanja i čitanja, ostali dozvolu čitanja
r-x r-x r-x	svi imaju dozvolu čitanja i izvršavanja (ulaska u direktorij)

Promenu dozvole vrši se naredbom **chmod**. Naredba **chmod** kao parametar ima kojoj grupi menjamo ovlašćenja, koja ovlašćenja i nad kojom datotekom. Grupe se označavaju sa:

- *u*-vlasnik,

- *g*-grupa,
- *o*-ostali i
- *a*-svima.

Dozvole mogu biti:

- *r*-čitanje,
- *w*-pisanje i
- *x*-izvršavanje/ulaženje.

Ukoliko dozvolu dodajemo koristimo "+", dok ukoliko je oduzimamo koristimo "-".

Pogledajmo nekoliko primera:

oliver% chmod o-rwx .login	ostalim korisnicima smo oduzeli dozvole čitanja, pisanja i izvršavanja
oliver% chmod o+rwx .login	ostalim korisnicima smo dodali dozvolu čitanja, pisanja i izvršavanja
oliver% chmod u-w .login	vlasniku datoteke smo uduzeli dozvolu pisanja
oliver% chmod u-r .login	vlasniku datoteke smo oduzeli dozvolu čitanja
oliver% chmod u+rw .login	vlasniku datoteke smo dozvolili čitanje i pisanje
oliver% chmod u-x Mail	vlasniku smo zabranili ulazak u direktorij
oliver% chmod u+x Mail	vlasniku smo omogućili ulazak u direktorij

Više o naredbi chmod (a i o ostalim naredbama) možete saznati naredbom **man *chmod*** (kao parametar man naredbe možete navesti bilo koju UNIX naredbu).

PROMENA VLASNIKA I GRUPE KOJOJ PRIPADA DATOTEKA

Želimo li promeniti vlasnika datoteke koristimo se naredbom **chown**. Kao parametar naredbe navodimo novog vlasnika i apsolutni ili relativni put do datoteke ili direktorija:

chown novi-vlasnik put-do-datoteke

U slučaju kada želimo promeniti grupu kojoj pripada datoteka to činimo naredbom **chgrp**. Kao parametar navodimo novu grupu i put do datoteke ili direktorija:

chgrp nova-grupa put-do-datoteke

Pogledajmo to kroz par primera (promene uvek možete videti sa naredbom ls -alg)

oliver% chown ivica .login	ivica postaje vlasnik datoteke .login

oliver% chgrp osoblje .login	grupa osoblje postaje vlasnik nad datotekom .login
oliver% chgrp student .login	grupa student postaje vlasnik nad datotekom .login

Napomena: ukoliko za vlasnika datoteke stavimo drugog korisnika više nismo vlasnici datoteke i ne možemo menjati njene dozvole, vlasnike i grupu kojoj pripada.

METAZNACI

Metaznaci su grupa znakova koji zamenjuju jedan ili više znakova. Metaznakove možemo koristiti kada želimo naredbom uticati na više datoteka ili direktorija. Skup tih metaznakova je u Tabeli 1.

Tabela 1.: Metaznaci	
*	Zamenjuje proizvoljan broj znakova. Uzmemo na primer "lis*", tu "*" zamenjuje proizvoljan broj znakova tako da možemo imati "lista", "lis ta", "lisica", ...
?	Zamenjuje samo jedan znak. Uzmemo na primer "lis?ca", tu "?" zamenjuje jedan znak tako da možemo imati "lisica", "lisbca", "lis ca", "lis1ca", ali ne i "lista"
[...]	Zamenjuje jedan znak sa bilo kojim znakom navedenim u uglastim zagradama. Na primer "lis[a..z]ca", tu "[a..z]" zamenjuje bilo koji znak između a i z tako da možemo imati "lisica", "lisbca", ali ne i "lis1ca".

U praksi metaznakove koristimo kada želimo naredbu primeniti samo na određene datoteke (Tabela 2.).

Tabela 2.: Primeri za korišćenje metaznakova na određene datoteke	
oliver% ls -l /etc/h*	dobiti ćemo sve datoteke u direktoriju etc koje počinju sa h
oliver% ls -l /etc/h?st	dobiti ćemo sve datoteke koje počinju sa h, završavaju sa st, a između imaju samo jedan znak
oliver% ls -l /etc/h??st	dobiti ćemo sve datoteke koje počinju sa h, završavaju sa st a između imaju dva znaka.

ČITANJE SADRŽAJA DATOTEKE

Do sada smo pregledavali direktorije i gledali datoteke spolja, ali nismo mogli videti sadržaj datoteke. Da bi mogli pogledati i unutar "korica" koristimo se nekom od UNIX naredbi za čitanje sadržaja datoteka: **cat** ili **more**.

cat

Naredba **cat** dolazi kao skraćenica od engleske reči **concatenate**, i prvenstveno je

namenjena spajanju datoteka. Također naredbu **cat** možemo iskoristiti i za prikazivanje sadržaja datoteke na ekran. Da bi prikazali sadržaj datoteke naredbom **cat** na standardni izlaz (ekran) koristimo:
cat datoteka(e)

Na primer želimo li videti sadržaj datoteke .login (datoteke čije se naredbe izvršavaju prilikom logiranja na UNIX računar) napisaćemo naredbu:

```
oliver% cat .login
# @(#)Login 2.0 modify by mviljac & azigman
##############################################################
.
.
.
oliver%
```

Naredba **cat** najbolje radi sa kratkim datotekama čiji sadržaj stane na jedan celi ekran. Ako je datoteka predugačka njen sadržaj će vam samo proletiti i nećete uspeti videti ništa osim sadržaja zadnjeg "ekrana". Također se ne možete vratiti na sadržaj prethodnih ekrana. Unesete li **cat** naredbu bez navođenja imena datoteke, naredba **cat** će čitati sadržaj standardnog ulaza (tastatura) i preusmeravati ga na standardni izlaz (ekran). Želimo li prekinuti daljnje izvođenje naredbe **cat** pritisnimo CTRL+D.

more

Želimo li pročitati sadržaj dužih datoteka moramo koristiti naredbu koja nam omogućava da na zaslonu prikazujemo "stranicu" po "stranicu" teksta. Standardni terminalski ekran ima 24 linije teksta. Sintaksa naredbe je:
more datoteka(e)

Naredba **more** nam omogućava da se krećemo po sadržaju datoteke. Mnoge verzije omogućavaju ne samo kretanje unapred po datoteci nego i unatrag. Pozivanjem naredbe **more** na ekranu će nam se prikazati prva "stranica" datoteke.

```
oliver% more .login
# @(#)Login 2.0 modify by mviljac & azigman
##############################################################
.
.
 --More--(17%)
```

Nakon prve stranice dobićemo podatak da smo do sada videli 17% sadržaja dokumenta i da unesemo naredbu za nastavak. Pritisnemo li ENTER dobićemo novu liniju teksta, dok sa SPACE dobijemo novu stranicu teksta. Za pomoć možemo pritisnuti tipku h (od engleske riječi help) i dobićemo ekran sa osnovnim naredbama koje razume **more**. Evo nekih od njih:

SPACE	prikaži sledeću stranicu teksta
RETURN	prikaži sledeću liniju teksta
nf	pomakni se unapred za n stranica
b	pomakni se unatrag za jednu stranicu
nb	pomakni se unatrag za n stranica

/riječ	potraži unapjed reč
?riječ	potraži unatrag reč
CTRL+L	osveži sadržaj ekrana
h	ispiši pomoć
q	završi sa pregledavanjem datoteke pre njenog kraja
v	pozovi vi editor na ovom mestu

STANDARDNI ULAZ I IZLAZ

Mnoge UNIX naredbe učitavaju podatke na ulazu i prikazuju rezultat na izlazu. Opštenito ako ne specifikujemo naredbi datoteku iz koje da čita podatke, korisnička ljuska od nas zahteva da unesemo podatke preko tastature (dokle god ne pritisnemo ENTER). Tastaturu nazivamo standardnim ulazom.

Kao rezultat, naredba nam daje ispis. Taj ispis obično je na ekranu. Ekran nazivamo standardnim izlazom. Svaka naredba, ukoliko se to drugačije ne specifikuje, učitava podatke sa standardnog ulaza (tastature) i prikazuje rezultate na standardnom izlazu (ekranu).

Naredbi možemo promeniti standardni ulaz i standardni izlaz.

PREUSMERIVANJE STANDARDNOG ULAZA I IZLAZA

Preusmerivanje standardnog ulaza i izlaza podrazumeva da naredbi menjamo da joj podaci dolaze sa tastature i rezultati odlaze na ekran. U tu svrhu nam služe operatori ">", "<" i ">>".

Operator ">" nam govori da izlaz naredbe preusmeravamo u datoteku čije se ime nalazi iza operatora:
oliver% ls -l /etc > proba

Ovom naredbom smo ispis rezultata naredbe *ls -l /etc* preusmerili sa standardnog izlaza (ekrana) u datoteku proba (sadržaj datoteke možemo pogledati sa more proba). Dakle kada god želimo rezultat neke naredbe preusmeriti sa standardnog izlaza u datoteku koristimo se operatorom ">". Ukoliko datoteka (u koju preusmeravamo izlaz) postoji njen prethodni sadržaj će biti izgubljen, dok ukoliko ne postoji datoteka će biti kreirana. Želimo li rezultat naredbe samo nadovezati na prethodni sadržaj datoteke onda za preusmeravanje izlaza koristimo operator ">>". Operator ">>" preusmerava ispis umesto na standardni izlaz u datoteku čije se ime nalazi iza operatora s time da ukoliko datoteka postoji ispis će se nastaviti na njenom kraju:
oliver% ls -l /usr/bin >> proba

Identično je i sa preusmerivanjem standardnog ulaza. Ukoliko želimo da nam naredba umesto standardnog ulaza koristi datoteku iza operatora "<" navedemo ime datoteke iz koje nam naredba uzima podatke:
oliver% mail moje_korisničko_ime < proba

ULANČAVANJE

Kao što smo mogli preusmeravati standardni ulaz i standardni izlaz, tako možemo i izlaz jedne naredbe preusmeriti u ulaz druge. Povezivanje dveju naredbi na taj način zove se ulančavanje (pipe). Operator koji označava ulančavanje je "|" i navodi se između dveju naredbi koje spajamo:
oliver% ls -l /etc | more

Ovom naredbom smo preusmerili izlaz naredbe *ls -l /etc* u ulaz naredbe *more*, tako da će nam sada rezultat naredbe *ls -l* biti mogućiti razgledavanje stranicu po stranicu.

Ulančavanjem nismo ograničeni na samo dve naredbe nego možemo ih spojiti i više:
oliver% ls -l /etc | grep a | more

Ovim smo izlaz naredbe *ls -l /etc* preusmerili na ulaz naredbe *grep* (naredba **grep** traži uzorak naveden iza imena naredbe u liniji i ukoliko ga nađe ispisuje tu liniju), da bi zatim izlaz naredbe *grep* preusmerili na ulaz naredbe *more*.

VI EDITOR (OSNOVE)

Želimo li napisati nešto složeniji tekst potreban nam je editor. Editor kojeg srećemo na većini UNIX sistema je vi editor. Zbog toga ćemo se u narednom tekstu osvrnuti na osnovne značajke i naredbe vi editora.

MODALITETI RADA

Vi editor je terminalski tekstualni editor. Prilikom pozivanja možemo koristiti jedan od dva načina:

- **vi [***ime-datoteke***]**
- **vedit [***ime-datoteke***]**

U prvom slučaju pozivamo vi editor nad datotekom ime-datoteke (ukoliko datoteka ime-datoteke ne postoji vi editor će je kreirati - ukoliko korisnik ima dozvolu za pisanje u radnom direktoriju), dok u drugom slučaju pozivamo isti vi editor ali verziju za početnike (lakši je za učenje, ali nema sve opcije aktivne). Ne navedemo li ime datoteke kao parametar prilikom pozivanja vi editora morat ćemo dati ime datoteci prije nego što je snimimo (izađemo iz vi editora sa naredbom ZZ).

Sam vi editor ima dva stanja:

- unošenje naredbi
- unošenje teksta

UNOŠENJE NAREDBI

Kada pozovemo vi editor on se u početku nalazi u stanju unošenja naredbi. Dok se nalazimo u stanju unošenje naredbi možemo zadati neku naredbu vi editoru koju će on izvršiti. Naredbe možemo podeliti u nekoliko grupa:

- naredbe za pomicanje pokazivača (kursora), (Tabela 3.)
- naredbe za prelaz na unos teksta (Tabela 4.)
- naredbe za brisanje i kopiranje teksta (Tabela 5.)
- naredbe za pretraživanje i menjanje teksta (Tabela 6.)

- naredbe za završetak rada s vi editorom (Tabela 7.)

Pogledajmo sada svaku od tih skupina malo detaljnije:

Tabela 3.: Naredbe za pomicanje pokazivača	
l ili strelica u desno	pomiče pokazivač za jedno mesto u desno
h ili strelica u levo	pomiče pokazivač za jedno mesto u levo
j ili strelica za dole	pomiče pokazivač jedan red ispod
k ili strelica za gore	pomiče pokazivač jedan red iznad
0	pomiče pokazivač na početak reda
$	pomiče pokazivač na kraj reda
H	pomiče pokazivač na prvi red na ekranu
M	pomiče pokazivač na srednji red na ekranu
L	pomiče pokazivač na zadnji red na ekranu
[*broj***]G**	pomiče pokazivač na liniju *broj*
Ctrl + f	pomiče jedan ekran unapred
Ctrl + b	pomiče jedan ekran unatrag

Tabela 4.: Naredbe za prelaz na unos teksta	
i	prelaz na unos teksta levo od pokazivača
I	prelaz na unos teksta na početku reda
a	prelaz na unos teksta desno od pokazivača
A	prelaz na unos teksta na kraju reda
o	otvaranje novog reda za unos teksta ispod reda s pokazivačem
O	otvaranje novog reda za unos teksta iznad reda sa pokazivačem
R	unos preko postojećeg teksta

Napomena: za povratak u stanje unošenja naredbi treba pritisnuti tipku *Esc* ili *Ctrl+c*.

Tabela 5.: Naredbe za brisanje i kopiranje teksta	
[*broj***]x**	briše *broj* znakova desno od pokazivača
[*broj***]X**	briše *broj* znakova levo od pokazivača
[*broj***]dd**	briše *broj* redova i sprema ih u registar
[*broj***]yy**	kopira *broj* redova i sprema ih u registar
p	kopira sadržaj registra u novi red ispod reda sa pokazivačem
P	kopira sadržaj registra u novi red iznad reda sa pokazivačem
r*c*	zameni znak pod pokazivačem sa znakom *c*

Tabela 6.: Naredbe za pretraživanje i menjanje teksta	
/*string*	nađi prvo pojavljivanje teksta "*string*" levo od pokazivača
?*string*	nađi prvo pojavljivanje teksta "*string*" desno od pokazivača

n	ponovi pretraživanje
:[*%***]s/***str1***/***str2***/[***g***]**	zameni tekst "*str1*" sa tekstom "*str2*" u celom tekstu

Tabela 7.: Naredbe za završetak rada s vi editorom	
:w	spremi do sada obavljene promene
ZZ	spremi tekst i izađi iz vi editora
:q!	izađi iz vi editora i zaboravi sve promene

Tabela 8.: Još neke korisne naredbe	
u	poništi zadnju naredbu
U	poništi sve promene u redu
.	ponovi zadnju naredbu
:r *ime-datoteke*	učitaj datoteku *ime-datoteke*
Ctrl+l	obnovi sadržaj ekrana
Ctrl+g	napiši broj reda u kome se nalazi pokazivač

Prethodne naredbe se mogu unositi samo u stanju unošenja naredbi!

UNOŠENJE TEKSTA

Kada se vi editor nalazi u stanju unošenja teksta onda sve znakove koje unosimo editor interpretira kao običan tekst. Da bi mogli ponovo uneti neku naredbu moramo izaći iz stanja unošenja teksta i preći u stanje unošenja naredbi. Prelazak se vrši pritiskom na tipku **Esc** ili **Ctrl+c**. Oba načina će nas vratiti u stanje unošenja naredbi i daljnje unošenje teksta editor će shvatiti kao naredbe (dokle god se ne navede naredba za unos teksta koja će izvršiti prelazak u stanje unosa teksta).

UREĐIVANJE DIREKTORIJA

Struktura UNIX datotečnog sistema (stablo) omogućava nam da jednostavno organizujemo svoje podatke. Nakon što smo kreirali datoteku, želimo je ponekad kopirati u drugi direktorij, promeniti joj ime ili dati različita imena istim datotekama. Također možemo željeti kreirati novi direktorij kada krećemo s novim projektom ili jednostavno želimo obrisati zastarele datoteke (direktorije) da nam ne zauzimaju suvišan prostor na disku.

KOPIRANJE DATOTEKA

Često želimo, pre nego što počnemo sa menjanjem neke datoteke, snimiti njenu kopiju radi kasnijeg eventualnog vraćanja njenog starog sadržaja.

cp

Naredbom **cp** pravimo kopiju datoteke u istom ili različitom direktoriju. Naredba cp nema uticaj na originalnu datoteku, pa je stoga to dobar način da izradimo sigurnosnu kopiju pre menjanja datoteke. Sintaksa naredbe je sledeća:
cp stara-datoteka nova-datoteka

gdje je *stara-datoteka* put do originalne datoteke, a *nova-datoteka* put do kopije originalne datoteke. Na primer želimo li kopirati datoteku */etc/hosts* u datoteku *hosts* to ćemo učiniti na sledeći način:
oliver% cp /etc/hosts hosts
oliver%

Također možemo naredbu **cp** koristiti i na sledeći način:
oliver% cp stara-datoteka nova-direktorij
oliver%

Ovom naredbom ćemo napraviti kopiju datoteke *stara-datoteka* (to je put do originalne datoteke) u direktorij *nova-direktorij* bez da promenimo ime datoteke, a gde je nova-direktorij put do direktorija gde ćemo smestiti kopiju datoteke.
Želimo li kopirati više od jedne datoteke istovremeno to možemo postići ili navođenjem metaznakova ili navođenjem više datoteka koje želimo kopirati i na kraju direktorija u koji ih želimo kopirati:
oliver% cp /etc/hosts /etc/asppp.cf .
oliver%

Ovom naredbom kopiramo datoteke *hosts* i *asppp.cf* iz direktorija */etc* u naš trenutni radni direktorij. Ukoliko datoteka odredišna datoteka već postoji onda će nas naredba **cp** pitati da li to stvarno želimo učiniti (Neke druge ljuske automatski obrišu postojeće datoteke bez da pitaju korisnika da li to želi. Da bi to izbegli možemo dodati naredbi *cp* opciju ***-i*** *"cp -i stara-datoteka nova-datoteka"*).

PREIMENOVANJE I POMICANJE DATOTEKA

Ponekad želimo datoteci promeniti ime. Da bi datoteci promenili ime koristimo naredbu **mv**. Naredba **mv** ujedno nam i služi za pomicanje datoteke iz jednog direktorija u drugi. Sintaksa naredbe je sledeća:
mv staro-ime novo-ime

gde je *staro-ime* početno ime datoteke, a *novo-ime* je odredišno ime (ime koje će datoteka imati nakon promene). Ukoliko odredišna datoteka postoji **mv** će nas pitati za dozvolu pre nego što je obriše (Neke druge ljuske automatski obrišu postojeće datoteke bez da pitaju korisnika da li to želi. Da bi to izbegli možemo dodati naredbi **mv** opciju: ***-i****"mv -i staro-ime novo-ime"*).
oliver% mv hosts hosts.old
oliver%

U prethodnom primeru datoteku *hosts* preimenujemo u datoteku *hosts.old*. Pogledamo li sa **ls** sadržaj direktorija videti ćemo da datoteka *hosts* više ne postoji, ali da postoji datoteka *hosts.old*.

Ukoliko kod naredbe **mv** za odredišnu datoteku navedemo put do nekog drugog direktorija datoteku ćemo prebaciti iz postojeći u odredišni direktorij:
oliver% mv hosts.old /tmp

U ovom primeru prebacujemo datoteku *hosts.old* u direktorij *tmp*.

TRAŽENJE DATOTEKE

Ponekad kada imamo puno datoteka ne možemo se setiti u kom direktoriju se tačno nalazi ona koju tražimo. Da bi mogli pretražiti datotečno stablo u potrazi za "izgubljenom" datotekom pomaže nam naredba **find**. Naredbu **find** koristimo tako da navedemo direktorij od kojeg počinjemo pretraživanje, ime datoteke koju tražimo i navođenjem opcije *prin*t kojom kazujemo naredbi **find** da rezultat pretraživanja ispiše na standardni izlaz:

```
oliver% find /home/vezbe -name "vaše-korisničko-ime" -print
/home/vezbe/vaše-korisničko-ime
oliver%
oliver% find / -name "vaše-korisničko-ime" -print
```

U prvom primeru tražimo datoteku ili direktorij sa imenom *"vaše-korisničko-ime"* i pretraživanje počinjemo od direktorija */home/vezbe*. U drugom slučaju pretraživanje počinjemo od *root* direktorija.

KREIRANJE NOVOG DIREKTORIJA

Uobičajeno je da datoteke koje su povezane smeštamo u isti direktorij. Uzmimo na primer da svu elektronsku poštu držimo u direktoriju *mail*, i da želimo ne mešati privremene datoteke sa elektronskom poštom. U tom slučaju ćemo kreirati direktorij *temp* u koji ćemo smestiti privremene datoteke. Naredba kojom kreiramo nove direktorije je **mkdir**:

mkdir ime-novog-direktorija

Kao parametar naredbe **mkdir** navodimo ime novog direktorija:

```
oliver% cd ~
oliver% mkdir temp
oliver%
```

U ovom primeru smo u svom *home* direktoriju kreirali poddirektorij *temp* (možemo proveriti sa naredbom **ls**).

BRISANJE DATOTEKA I DIREKTORIJA

Nakon nekog vremena poželimo obrisati datoteke ili direktorije koji nam više ne trebaju. Periodičnim brisanjem datoteka i direktorija držimo ažurnim svoje datotečno stablo i oslobađamo prostor na disku (veoma važno na sistemima koji imaju postavljenu kvotu). Naredba kojom brišemo datoteke je **rm**. Sintaksa naredbe je jednostavna:

rm ime-datoteke

Kao parametar naredbe **rm** navodimo ime datoteke koju želimo obrisati. Za brisanje direktorija koristimo naredbu **rmdir**, a sintaksa joj je sledeća:

rmdir ime-direktorija

Kao parametar naredbe **rmdir** navodimo ime direktorija kojeg želimo obrisati.

Pogledajmo jedan primer:
```
oliver% cd ~
oliver% mkdir proba
oliver% cp /etc/a* proba
oliver% cd proba
oliver% ls
oliver% rm asppp.cf
oliver% rm a*
oliver% cd ~
oliver% rmdir proba
```

U ovom primeru prvo smo kreirali poddirektorij proba. Zatim smo u njega kopirali sve datoteke iz direktorija */etc* koje počinju sa *a*. Promenili smo radni direktorij u *proba* i pogledali njegov sadržaj. U njemu smo prvo izbrisali datoteku *asppp.cf*, da bi nakon toga izbrisali sve datoteke koje počinju sa *a*. Ponovo smo za radni direktorij izabrali svoj *home* direktorij i onda u njemu obrisali poddirektorij *proba*.

POPRIČAJMO S DRUGIM KORISNICIMA

Kao što smo do sada imali prilike videti UNIX je višekorisnički sistem. To znači da osim nas na UNIX računaru može raditi još korisnika.

KOGA IMA NA RAČUNARU?

Da bi saznali ko je trenutno aktivan na "našem" računaru koristimo se naredbom **who** ili **finger**. Naredbe imaju sledeću sintaksu:
who
finger [ime-korisnika][@ime.računara]

Kao što se vidi naredba **who** je sasvim jednostavna i navodimo je bez parametara. Ona nam daje trenutno aktivne korisnike na računaru na kome smo pokrenuli naredbu.

Pogledajmo na primeru kako se upotrebljava naredba **who** i što daje kao izlaz:
```
oliver% who
elabor pts/0 Apr 16 09:03 (vts.su.ac.yu)
anovak pts/1 Apr 16 09:05 (yunord.net)
mbozic pts/2 Apr 16 09:17 (tippnet.co.yu)
ppetro pts/3 Apr 16 09:22 (vts.su.ac.yu)
```

Kao rezultat naredba who vraća spisak trenutno aktivnih korisnika. Osim njihovih imena dobivamo podatak na kojem virtualnom terminalu rade (pts/x), datum i vreme kada su počeli raditi i mesto (obično računar) sa kojega su se spojili.

Za razliku od who naredba finger daje nešto drugačiji izlaz (u osnovnom obliku):
```
oliver% finger
Login   Name          TTY      Idle    When         Where
elabor Eda Labor      pts/0           Wed 09:03  vts.su.ac.yu
anovak Albert Novak   pts/1      1    Wed 09:05  yunord.net
mbozic Marija Bozic   pts/2           Wed 09:17  tippnet.co.yu
ppetro Petar Petrovic pts/3      1    Wed 09:22  vts.su.ac.yu
```

Također ovde imamo ime trenutno aktivnih korisnika (Login), ime virtualnog terminala na kome rade (TTY), datum i vreme početka rada (When) i od kuda rade (Where). Osim toga imamo i puno ime korisnika (Name) kao i indikator njegove aktivnosti (Idle). Indikator aktivnosti nam govori pre koliko vremena je korisnik zadnji put izvršio neku akciju putem standardnog ulaza (pritisnuo tipku).

Sa naredbom finger možemo dobiti i više informacija o korisniku, tako da iza naredbe navedemo korisničko ime:

```
oliver% finger elabor
Login name: elabor                  In real life: Eda Labor
Directory: /home/student/elabor     Shell: /usr/bin/tcsh
On since Apr 16 09:03:32 on pts/0 from  vts.su.ac.yu
11 seconds Idle Time
Mail last read Wed Apr 16 09:30:29 1997
No Plan.
```

Vidimo da smo sada dobili neke osnovne informacije o korisniku računara. Tu su korisničko ime (Login name) i njegovo puno ime (In real life), put do njegovog home direktorija (Directory) i ljuska koju koristi (Shell). Osim tih podataka imamo i podatke kada se korisnik zadnji put prijavio za rad na računaru kao i kada je zadnji put pročitao svoju poštu.

Za razliku od who s naredbom finger možemo "pogledati" i ko je aktivan na nekom drugom računaru. To činimo tako da iza naredbe finger dodamo ime računara na kome želimo saznati ko je trenutno aktivan:

```
oliver% finger @vts.su.ac.yu
[vts.su.ac.yu]
Login     Name              TTY   Idle   When      Where
jjovan  Jovan Jovanovic - Osno pts/1 1:17 Wed 08:03 vts.su.ac.yu
anovak  Albert Novak          pts/0   23 Wed 09:03 tippnet.co.yu
```

Na nekim računarima, iz sigurnosnih razloga, ne možete dobiti potpun spisak trenutno aktivnih korisnika već morate tražiti podatke o tačno određenom korisniku:

```
oliver% finger jjovan@vts.su.ac.yu
[vts.suac.yu]
Login       Name        TTY    Idle  When     Where
jjovan    Jovan Jovanovic-FET pts/6 19:00 Tue10:25 vts.su.ac.yu
```

ili ukoliko želimo više podataka o korisniku

```
oliver% finger -l jjovan@vts.su.ac.yu
[vts.su.ac.yu]
Login name: jjova n             In real life: Jovan Jovanovic - FET
Directory: /home/staff/jjovan           Shell: /usr/bin/tcsh
On since Apr 15 10:25:08 on pts/6 from vts.su.ac.yu
19 hours Idle Time
No unread mail
No Plan.
```

JEDNOSMERNA KOMUNIKACIJA

Nakon što smo saznali ko je trenutno aktivan na računaru možemo nekome od korisnika (samo ukoliko je aktivan na računaru gde i mi) poslati poruku. U tu svrhu koristimo naredbu **write**. Naredba ima sledeću sintaksu:
write korisničko-ime [broj-terminala]

Korisničko ime obavezno navodimo, dok *broj terminala* samo u slučaju kada je korisnik aktivan na više terminala istovremeno. Nakon što unesemo naredbu sve što kucamo na našem terminalu ide korisniku koga smo naveli u naredbi. Za kraj poruke pritisnemo CTRL+D. Korisniku kome šaljemo poruku pojaviti će se na ekranu:
Message from ime-korisnika on oliver (pts/x) [Wed Apr 16 11:31:38] ...
Dakle podaci od koga je poruka s kog terminala je poslana i u koje vreme.

DVOSMERNA KOMUNIKACIJA

Osim jednosmerne komunikacije (samo šaljemo poruke u jednom smeru) možemo na UNIX sistemima ostvariti i dvosmernu komunikaciju. Za dvosmernu komunikaciju koristimo naredbu **talk**. Sintaksa naredbe je:
talk ime-korisnika[@ime.računara]

Kao *ime korisnika* navodimo korisničko ime osobe s kojom želimo uspostaviti dvosmernu komunikaciju, dok *ime računara* navodimo u slučaju da se ta osoba nalazi na drugom UNIX računaru. Nakon što pokrenemo naredbu ekran nam se podeli u dva dela (u gornjem delu pišemo mi, dok u donjem korisnik koga smo pozvali) isprekidanom linijom.

U gornjem delu nam piše status veze i on je na početku:
[Waiting for your party to respond]

Za to vreme korisniku koga zovemo se javlja poruka na ekranu kojom ga UNIX sistem obaveštava o pozivu za dvosmernu komunikaciju:
Message from Talk_Daemon@oliver at 11:43 ...
talk: connection requested by jjovan@vts.su.ac.yu.
talk: respond with: talk pero@tippnet.co.yu

Ukoliko korisnik odmah ne odgovori onda se menja status na ekranu u:
[Ringing your party again]

a korisniku s kojim želimo uspostaviti dvosmernu komunikaciju ponavlja se poruka
Message from Talk_Daemon@oliver at 11:43 ...
talk: connection requested by jjovan@vts.su.ac.yu.
talk: respond with: talk pero@tippnet.co.yu

Korisnik kojega se poziva odgovara na poruku sa:
oliver% talk ime-korisnike

Nakon toga se i njemu ekran podeli u dva dela odvojena isprekidanom linijom i pojavi se poruka

[Connection established^G^G^G]

Ta ista poruka se pojavi i na ekranu korisnika koji je inicirao vezu. Nakon toga sve što pišemo u gornjoj polovini ekrana vidi korisnik sa druge strane veze u donjoj polovini. Vezu prekidamo pritiskom na CTRL+C.

Ukoliko želimo da nam niko ne može slati poruke (onemogućiti write i talk) možemo izvršiti naredbu:
oliver% mesg n

ili ukoliko želimo omogućiti
oliver% mesg y

ŠTA JE TO E-MAIL?

Do pre nekoliko godina elektronsku poštu (u daljnjem tekstu E-mail) koristio je uski krug ljudi kojima je prvenstveni posao vezan uz računar (sistem inženjeri, programeri, …), dok je danas E-mail nešto bez čega se ne može u suvremenom svetu. Danas je sasvim uobičajeno da na vizit kartici uz adresu i broj telefona imate i svoju E-mail adresu.

Elektronska pošta, za razliku uobičajenog načina dostave, zgodnija je za razmenu informacija. Pre svega tu je "jednostavnost" i brzina dostave takve pošte. Naime kod elektronske pošte nije potrebno čekati danima dok se pošta ne isporuči (u slučaju telegrama sati), već je pošta na odredištu u roku od nekoliko minuta.

Da bi elektronska pošta mogla stići na odredište potrebno ju je adresirati na nekoga i da postoje određeni protokoli koji će izvršiti sam transport informacije sadržane u E-mailu.

E-MAIL ADRESE

Želimo li nekome poslati poštu moramo znati njegovu adresu. Ukoliko šaljemo "običnu" poštu (putem HPT-a) onda moramo na omotnici napisati odredišnu adresu da bi radnici pošte znali kome pismo preusmeriti. Slično je i kod elektronske pošte samo što ovde navodimo adresu virtualnog poštanskog sandučića. Ta adresa se sastoji od imena korisnika kome šaljemo E-mail i imena računara (ili servisa) gde se taj korisnik nalazi:
korisničko-ime@ime-računara

Da bi mogli nekome poslati E-mail dovoljno je poznavati njegovu E-mail adresu.

PROTOKOLI KOJI PRENOSE POŠTU

Da bi računar mogao razmenjivati elektronsku poštu potrebno je da poznaju konvenciju komunikacije koju nazivamo protokol. Dva protokola koja su najvažnija za razmenu elektronske pošte su SMPT (Simple Mail Transfer Protocol) i POP3 (Post Office Protocol). SMPT protokol koriste računari u međusobnom "razgovoru"

prilikom prenosa elektroniske pošte, dok POP3 protokol pretežno koriste razni E-mail čitači putem kojih sa udaljenog mesta pristupamo računaru koji je naš elektronski poštanski poslužitelj (obično je to računar na kome nam je otvoren korisnički račun).
Za razliku od SMPT protokola, POP3 protokol zahteva identifikaciju korisnika pre nego što izvrši bilo kakav prenos elektronske pošte.

E-MAIL KLIJENTI

Da bi poštu mogli predati našem elektronskom poštanskom poslužitelju (E-mail serveru) moramo koristiti neki mail klijent program kojim ćemo mu dostaviti našu elektronsku poštu da bi je on mogao dalje poslati.

Na UNIX računarima takvih klijent programa ima više: mail, elm, pine, … .PC računari imaju također više verzija takvih klijent programa: mail klijent u Internet Exploreru i Netscape-u, Eudora, … .
Svaki od tih programa ima svoje prednosti i mane. Mi ćemo kroz daljnji tekst nešto detaljnije obraditi pine klijent program. On je odabran iz razloga što se nalazi na UNIX računaru tako da ga možemo koristiti u slučajevima kada nismo u mogućnosti slati elektronsku poštu direktno sa PC-a, a ima (za razliku od drugih UNIX programa za slanje i čitanje elektronske pošte) jednostav korisnički interfejs (sa helpom) i dobar tekst editor.

PINE

Pine program startujemo jednostavno tako da napišemo:
oliver% pine

Nakon toga će nam se pojaviti par poruka o tome da li želimo da nam pine kreira poddirektorij sa našim poštanskim fahovima i konfiguracionu datoteku, te na kraju da li želimo da nam pošalje E-mailom upute za korišćenje pine-a. Nakon što smo uspešno prošli inicijalizaciju programa pojavi nam se na ekranu meni sa osnovnim naredbama pine-a:

```
PINE 3.91 MAIN MENU Folder: INBOX 0 Messages
      ? HELP - Get help using Pine
      C COMPOSE MESSAGE - Compose and send a message
      I FOLDER INDEX - View messages in current folder
      L FOLDER LIST - Select a folder to view
      A ADDRESS BOOK - Update address book
      S SETUP - Configure or update Pine
      Q QUIT - Exit the Pine program


Copyright 1989-1994. PINE is a trademark of the University of Washington.
[Folder "INBOX" opened with 0 messages]
? Help P PrevCmd R RelNotes
OTHER CMDS L [ListFldrs] N NextCmd K KBLock
```

Prilikom korišćenja pine-a u svakom trenutku imamo opis naredbi, koje možemo koristiti, na dnu ekrana (znak ^ označava tipku **CTRL** koja mora biti pritisnuta zajedno sa znakom koji sledi iza nje). Sada smo spremni da pošaljemo naš prvi E-mail.

Izaberimo opciju **C** COMPOSE MESSAGE (pritiskom na tipku C ili pomicanjem strelica pa pritiskom na Enter) i pred nama će se pojaviti novi sadržaj u kojem se od nas traži da unesemo ime osobe kojoj šaljemo poruku (To:), ime osobe kojoj želimo poslati kopiju poruke (CC:), put do datoteke koju želimo pridružiti našoj E-mail poruci (Attchmnt:), naslov poruke (Subject:) i na kraju sam tekst poruke.

Od svih ovih polja važno je da upišemo tačnu adresu primaoca poruke, a poželjno je i navesti naslov poruke. Nakon što unesemo sadržaj poruke poruku šaljemo pritiskom na **CRTL+X**.

KAKO PROČITATI PRISTIGLE PORUKE

Želimo li pročitati pristigle (ili pohranjene poruke) izaberemo opciju **L** FOLDER LIST. Ta opcija će nam prikazati postojeće poštanske fahove (u svakom poštanskom fahu mogu se nalaziti poruke, a služe nam da bi lakše mogli razvrstavati i organizovati svoju elektronsku poštu). Poštanski fah u koji nam uvek stiže tek pristigla elektronska pošta (kasnije je možemo razvrstavati u druge fahove ili zauvek obrisati) je INBOX.

Da bi pročitali poruke u INBOX-u selektujemo isti i pritisnimo **enter** (na taj način ćemo dobiti sadržaj INBOX-a). Zatim selektujemo poruku koja nas zanima i pritisnimo opet **enter** da bi dobili njen sadržaj.
Nakon što smo pročitali poruku sa **I** se vraćamo u indeks poruka, a sa **M** u glavni menu.

ZAGLAVLJE

Svaka E-mail poruka sastavljena je iz dva dela, zaglavlja (header) i same poruke (body). Zaglavlje sadrži sve linije teksta od prve linije do prvog praznog reda. Sve iza zaglavlja je poruka.

Zaglavlje ima više linija koje su dodane samoj poruci. Pogledajmo kako izgleda jedno zaglavlje E-mail poruke:
Date: Thu, 9 Jan 1997 13:10:17 -0500 (EST)
From: Jovan Jovanovic <amilas@lagrange.rutgers.edu>
To: Petar Petrovic <ppetro@vts.su.ac.yu>
Subject: Marijina adresa

Iz samog zaglavlja možemo videti kada je poruka poslana (Date), ko ju je poslao (From) kome (To) i koji je naslov poruke (Subject). Iz podatka From znamo adresu pošaljitelja i na tu adresu možemo poslati odgovor.

REPLY I FORWARD E-MAILA?

Ponekad kada od nekoga dobijemo elektronsku poštu želimo na tu poruku odmah odgovoriti ili je preusmeriti nekome drugome.

Reply podrazumeva odgovaranje na poruku s time da sam program iz dolazne poruke izvuče adresu pošaljioca i uzme je kao adresu korisnika kome šaljemo odgovor. Praktično to znači da ne moramo sami upisivati adresu primaoca (kome šaljemo odgovor), već to za nas učini računar. Reply možemo napraviti u programu pine tako

da kada dođemo u direktorij (folder) elektronskih selektiramo poruku na koju želimo odgovoriti i pritisnemo tipku R. Nakon toga pine nas pita da li želimo uključiti originalnu poruku u odgovor (obično se kada odgovaramo na neku poruku uključi jedan njen deo da bi onaj koji je prima znao na šta mu to tačno odgovaramo). Ukoliko želimo uključiti primljenu poruku u odgovor onda odgovorimo sa Y. Dalje nas pine pita da li želimo da poruku pošaljemo samo pošaljiocu ili svima koji su dobili ovu poruku (Reply to all recipients?). Ukoliko želimo da samo onaj koji je autor poruke dobije naš odgovor odgovoriti ćemo sa N (ne) ili ukoliko želimo da poruku dobiju svi oni koji su navedeni u copy listi (CC:) odgovoriti ćemo sa Y (da).

Sa naredbom **forward** (aktiviramo je tako da pritisnemo tipku F kada je selektovana ona poruka u folderu koju želimo nekome proslediti) prosleđujemo selektovanu poruku. Uzmemo da smo dobili poruku na koju mi ne možemo odgovoriti, ali znamo ko može. U tom slučaju bi jednostavno tu poruku prosledili (forward) na adresu korisnika koji može na tu poruku i odgovoriti (eventualno bi dodali poruci neko kratko objašnjenje).

Prosleđivanje poruke aktiviramo sa pritiskom na tipku F (kada nam je u folderu aktivirana poruka koju želimo proslediti. Zatim nam se na ekranu pojavi ekran kao kada šaljemo E-mail s tom razlikom da već imamo tekst i naslov poruke upisan a mi trebamo samo dodati adresu korisnika kome šaljemo poruku (i eventualno izmeniti naslov poruke ili tekst).

POŠTANSKE LISTE

Ponekad grupa ljudi koju vezuje neki zajednički interes povezana je u jednu listu korisnika koju nazivamo poštanska lista. Sa poštanskom listom se komunicira putem mail programa. Kada želimo poslati poruku svim pretplatnicima poštanske liste onda jednostavno navodimo E-mail adresu poštanske liste dok će se sam poslužitelj (server) poštanske liste pobrinuti da je svi njeni pretplatnici dobiju. Kada se želimo pretplatiti (ovde se ne radi o nikakvom stvarnom plaćanju) na poštansku listu onda svoj zahtev šaljemo na E-mail adresu poslužitelja poštanske liste sa zahtevom za pretplatu. Dakle kod poštanskih lista treba razlikovati dve adrese:

- adresa samog poslužitelja poštanske liste listproc@ime-računara
- adresa same liste ime-liste@ime-računara

Na samu poštansku listu pretplata može biti automatska (govorimo o javnoj poštanskoj listi) i tada je ta lista otvorena za sve korisnike ili pretplatu može izvršiti samo vlasnik liste (tada govorimo o privatnoj poštanskoj listi).

Sama lista može biti moderirana ili nemoderirana. Ukoliko je lista moderirana onda sve poruke dolaze prvo do vlasnika poštanske liste te sam vlasnik odlučuje koje poruke mogu doći na listu a koje ne.

Sam poslužitelj poštanske liste razume određene naredbe primljene putem elektronske pošte na njegovu adresu s kojima korisnici mogu izvršiti određene akcije. Prilikom slanja naredbe poslužitelju poštanske liste za adresu navedemo adresu poslužitelja poštanske liste, kao naslov poruke ne navodimo ništa dok kao poruku pišemo naredbe koje razume poslužitelj poštanske liste.

Pogledajmo ukratko koje naredbe razume poslužitelj poštanske liste:

help [naredba]	Navođenjem samo naredbe help dobiti će se osnovna uputstva o naredbama koje razume poslužitelj poštanske liste. Navođenjem dodatnog argumenta ***naredbe*** dobiti će se detaljnija uputstva za specifičnu ***naredbu***.
subscribe <lista> <ime prezime>	Jedini način da se korisnik "pretplati" na listu. Obavezno se mora navesti naziv liste (malim slovima), te ime i prezime korisnika.
unsubsrcibe <lista> signoff <lista>	Ove dve naredbe su dva načina na koji korisnik može maknuti svoje ime s poštanske liste (poništiti pretplatu).
recipients <list> review <list>	Dva načina na koja korisnik može dobiti podatke ko je sve pretplaćen na poštansku listu.
lists	Daje popis poštanskih lista kojima možemo pristupiti
which	Daje popis poštanski lista na koje je korisnik preplaćen.

Evo sada primera kako dobiti osnovne informacije o samim naredbama, postojećim listama i izvršiti pretplatu na listu.

```
help
list
subscribe biks-list Ime Prezime
```

Ako ovo pošaljemo kao poruku poslužitelju poštanske liste on će svaki red shvatiti kao posebnu naredbu. Na **help** će poslati na našu E-mail adresu kratko uputstvo o korišćenju liste, na list će poslati trenutno dostupne liste na poslužitelju, dok će naredba **subscribe** izvršiti pretplatu na poštansku listu biks-list korisnika koji se zove Ime Prezime (kao odgovor će nam poslužitelj poslati poruku o tome da smo se pretplatili na poštansku listu biks-list).

ŠTA JE TO FTP?

FTP je skraćenica od File Transfer Protocol, to jest protokol za prenos datoteka. FTP protokol je namenjen prenošenju datoteka sa/na udaljeni računar (remote computer). Da bi mogli ostvariti sam prenos FTP protokolom moramo imati klijent aplikaciju s jedne strane i poslužitelja (server) s druge strane.

FTP poslužitelj je na UNIX računarima jedan pozadinski proces koji se brine o spajanju klijent aplikacije te prenošenju datoteka sa i na poslužitelja. Prilikom pristupanja ftp poslužitelju on na početku veze vrši identifikaciju korisnika koji se spaja. Pri identifikaciji unosimo svoje korisničko ime i lozinku koju imamo na tom računaru:

```
vts% ftp vts.su.ac.yu
Connected to vts.su.ac.yu.
220 vts FTP server (Version wu-2.4-ci(3) Fri Apr 26 12:35:48 MET DST
1996) ready.
Name (vts.su.ac.yu:ppetrov): vts
331 Password required for vts.
Password:
230 User vts logged in.
```

ftp>

Ukoliko je sam ftp poslužitelj ima konfiguraciju kao javni (**public**) onda mu možemo pristupiti i preko korisničkog imena ftp ili **anonymous**. U tom slučaju kao lozinku navodimo svoju e-mail adresu:

```
vts% ftp ftp.vts.su.ac.yu
Connected to vts.su.ac.yu.
220 vts FTP server (Version wu-2.4-ci(3) Fri Apr 26 12:35:48 MET DST
1996) ready.
Name (vts.su.ac.yu:ppetrov): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
230 Guest login ok, access restrictions apply.
ftp>
```

Na javnom ftp poslužitelju obično se nalaze razni public, freeware i shareware programi. Svi ti javni programi spremljeni su u poddirektoriju pub koji je anonymous korisnicima jedino i dostupan. Osim **pub** poddirektorija tu je još i poddirektorij **incoming** ili **inbox** u koji **anonymous** korisnici mogu spuštati datoteke. Obično se u taj direkotrij može pisati, ali se iz njega ne može čitati već administrator javnog ftp poslužitelja odlučuje koji sadržaj će biti dostupan drugim korisnicima:

```
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
usr
bin
etc
dev
pub
incoming
226 Transfer complete.
98 bytes received in 0.022 seconds (4.3 Kbytes/s)
ftp>
```

Ftp klijent progam možemo pozvati na dva načina; navođenjem samo naredbe ftp ili navođenjem naredbe ftp s imenom ftp poslužitelja kao parametrom:

```
ftp> ftp
  ili
ftp> ftp ftp.vts.su.ac.yu
```

U prvom slučaju će nam se pojaviti ftp prompt (oznaka da je program spreman primiti naše naredbe), dok će u drugom ftp klijent prvo nastojati uspostaviti vezu sa ftp poslužiteljem pa tek nakon identifikacije će se pojaviti ftp prompt.

Nakon toga ftp klijent je spreman prihvatiti naše naredbe. Naredbe koje razumije ftp klijent su sledeće:

account [passwd]	Omogućuje nam da unesemo korisničku lozinku na poslužitelju radi pristupa njegovim resursima.
ascii	Prebacuje se u način prenosa ASCII datoteka

binary	Prebacuje se u način prenosa binarnih datoteka (obično tako prenosimo sve podatke)
bye	Prekidamo vezu sa poslužiteljem i izlazimo iz ftp programa
cd	Mijenjamo radni direktorij na poslužitelju (indentično UNIX naredbi cd)
close	Prekidamo vezu sa poslužiteljem i vraćamo se interpreter naredbi (ftp prompt)
get udaljena-datoteka	Tražimo od poslužitelja datoteku navedenu kao parametar (udaljena-datoteka) i prenosimo je na lokalni računar
hash	Ispisuje znak # za svaki preneseni blok od 8192 byte-a (zavisno od ftp klijent programa ovaj broj može varirati)
lcd	Menjamo radni direktorij na lokalnom računaru
ls	Listamo sadržaj radnog direktorija na poslužitelju
mget udaljene-datoteke	Tražimo od poslužitelja da nam prenese više datoteka i smesti ih u lokalni radni direktorij. Možemo kao parametar navesti više datoteka ili koristiti metaznakove
mput lokalne-datoteke	Tražimo od poslužitelja da od nas preuzme više datoteka i smesti ih u svoj radni direktorij. Možemo kao parametar navesti više datoteka ili koristiti metaznakove
open ime-računara	Uspostavljamo vezu sa ftp poslužiteljem na računaru koje smo naveli kao parametar open naredbe
prompt	Uključuje i isključuje interaktivne upite. Kada koristimo naredbe za rad sa više dokumenata (mget i mput) onda možemo uključiti ili isključiti upit o svakoj datoteci. Naredbom prompt uvek menjamo stanje (ako su interaktivni upiti uključeni prompt naredba će ih isključiti i obrnuto)
put lokalna-datoteka	Tražimo od poslužitelja da preuzme od nas datoteku i smesti je u svoj radni direktorij
pwd	Tražimo od poslužitelja da nam ispiše koji je njegov trenutni radni direktorij
quit	Identično kao i bye
rmdir udaljeni-direktorij	Tražimo od poslužitelja da kreira poddirektorij u svom radnom direktoriju
user ime-korisnika	Ukoliko smo prilikom uspostave veze uneli krivo korisničko ime te nismo prošli proces identifikacije, korišćenjem naredbe user možemo ponovo uneti korisničko ime i lozinku da bi mogli pristupiti do željenih resursa ftp poslužitelja

Pogledajmo sada na primeru kako izgleda prenos datoteke sa poslužitelja i na poslužitelj:

```
vts% ftp ftp.vts.su.ac.yu
Connected to vts.su.ac.yu.
220 vts FTP server (Version wu-2.4-ci(3) Fri Apr 26 12:35:48 MET DST
1996) ready.
Name (ftp.vts.su.ac.yu:ppetrov): vts
331 Password required for vts.
Password:
230 User vts logged in.
```

```
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
.cshrc
.profile
.login
.netscape-preferences
226 Transfer complete.
49 bytes received in 0.0047 seconds (10 Kbytes/s)
ftp> bin
200 Type set to I.
ftp> hash
Hash mark printing on (8192 bytes/hash mark).
ftp> get .login
ftp> get .login
200 PORT command successful.
150 Opening BINARY mode data connection for .login (3188 bytes).
#
226 Transfer complete.
local: .login remote: .login
3188 bytes received in 0.01 seconds (3e+02 Kbytes/s)
ftp> put arj.exe
200 PORT command successful.
150 Opening BINARY mode data connection for arj.exe.
################
226 Transfer complete.
local: arj.exe remote: arj.exe
116260 bytes sent in 0.24 seconds (4.7e+02 Kbytes/s)
ftp>
```

POJAM SOFTVERA

UVOD

Realizacijom, nabavkom računarskog hardvera dobili smo jedno moćno sredstvo, koje može mnogo koristiti, ako se zna kako. To znanje je tzv. računarski softver (programi), niz instrukcija koje "vode" računar pri rešavanju nekog konkretnog zadatka.

Softver se nalazi na nekom memorijskom medijumu (na hard disku, disketi, CD ROM-u) ili u operativnoj memoriji računara (u ROM-u), a pre upotrebe mora se upisati u centralnu memoriju (RAM memoriju) kako bi postao operativan.

Tvrdi disk (hard disk): Masovna memorija sa direktnim pristupom sastavljena iz više diskova, čija je površina namagnetisana. Tvrdi disk je smešten u jednom specijalnom drajvu, koji se obično naziva Viičester-drajv. Pristup podacima, koji se nalaze na tvrdom disku, se vrši preko upisno/čitajućih glava.

Flopi disk (disketa): Vrsta nosioca podataka sa direktnim pristupom. Disketa je elastični disk od plastike, čija je površina namagnetisana. Ona je uvek smeštena u plastični omot, koii ie štiti.

CD - ROM: Optički disk na kome se smeštena informacija ne može izbrisati.

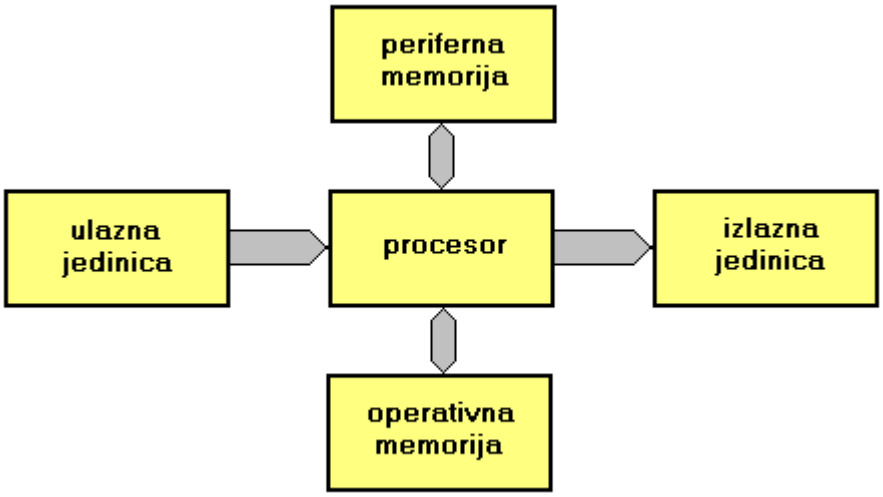
Postoje u principu dva načina da se obezbedi potreban softver za računar. Softver se ili kupuje ili razvija. Teško je reći koji je od ova dva načina bolji. Najverovatnije je kombinacija ova dva rešenja ono pravo rešenje, jer početi razvijati softver od nule, znači biti uvek u velikom zaostatku, a kupovati uvek najnoviji softver, znači ostati uvek u potpunoj zavisnosti od onog koji ga piše i praktično nikada ne znati kako se dotični problem rešava. Normalno, pisanje nekih programa ne treba ni da pada na pamet, kao što su operativni sistemi računara, ili veoma popularni (i jeftini) korisnički programi za pisanje, crtanje itd.

Pisanje softvera je postupak koji se naziva programiranje računara. Program, prema tome, predstavlja niz ciljanih koraka (kojima je do u detalje opisan algoritam, odnosno postupak za rešenje nekog problema) da ih računar razume, pa da mogu biti i izvedeni u računaru, kako bi se došlo do rešenja. Načina na koji softver može biti napisan ima više.

Algoritam: Pojam algoritma je jedan od osnovnih pojmova u informatici. Algoritam je opšti postupak za rešavanje jednog problema (ili klasa problema), koji posle konačnog broja jednoznačnih koraka daje rešenje problema. Osim ovoga, algoritam mora da sadrži tačno uputstvo za svako stanje koje se može pojaviti prilikom obrade zadatog problema. Svaki ispravan program nije ništa drugo do opis jednog algoritma. Prema tome, programski jezici su jezici koji služe za formulisanje algoritama.

MAŠINSKO PROGRAMIRANJE

Mašinsko programiranje je "najniži nivo" programiranja računara, i kao što je to već napomenuto, za ovakav pristup rešavanju programskih problema potrebno je, pored problema koji se rešava, poznavati i arhitekturu procesora i celokupnog hardvera na kojem će se problem raditi. Praktično to znači da treba poznavati osnovne instrukcije za manipulisanje podacima na relaciji **ulaz/izlaz ⇔ procesor ⇔ memorija** (slika 1.), a one su specifične za svaki procesor i nazivaju se jednim imenom mašinske instrukcije.



Slika 1.: Blok-šema računara

ASEMBLER

Zbog nepreglednosti programa napisanih mašinskim instrukcijama, uvodi se skraćeni način njihovog pisanja, tzv. mnemonički (simbolički) ili asemblerski kod, kratko asembler.

Mnemonički kod (simbolički kod): Ovaj atribut se dodeljuje pojmovima, koji služe kao podrška ljudskom pamćenju. Na primer, u većini asemblera, operacija sabiranja se obeležava simbolom ADD (engleski: sabiranje). Time programer odmah zna o kojoj je operaciji reč. Osim operacija, i podaci mogu imati simbolička imena (simboličko adresiranje).

Da bi se savladala tehnika programiranja na nivou asemblera, potrebno je poznavati unutrašnju logiku računara, pa učenje osnova ovakvoga programiranja traje nešto duže, a programi napisani na ovaj način nisu ni pregledni ni jednostavni za korigovanje. Jedina prednost im je brzo izvođenje, jer se u mašini izvode direktno.

VIŠI PROGRAMSKI JEZICI

Programiranje računara pomoću asemblera ima i jedan veliki nedostatak. Univerzalnost (nezavisnost od tipa računara na kome se radi), pa i atraktivnost rada nije takva da bi se računarima obezbedilo široko tržište i veliki broj korisnika. Te činjenice su bili svesni i veliki proizvođači računara, pa su u borbi za bolji plasman odvajali velika sredstva za istraživanje metoda za jednostavnije i korisniku "bliže" programiranje. Kao rezultat tih istraživanja krajem pedesetih godina pojavio se prvi viši programski jezik kojim je bilo omogućeno programiranje nezavisno od tipa računara. Programski jezik je bio namenjen za opštu upotrebu, i to, pre svega, za inženjersko-tehničke probleme.

Skup instrukcija: Skup svih instrukcija jednog računara (mašinski jezik), odnosno naredbi jednog programskog jezika.

Izvorni program: Program napisan na nekom simboličkom ili višem programskom jeziku. Karakteristika izvornog programa je da se on ne može direktno izvršiti na računaru, već se pre toga mora prevesti u objektni program korišćenjem odgovarajućeg prevodioca. Programski jezik na kome je izvorni program napisan nazivamo izvorni programski jezik. Većina proizvođača ne isporučuje svoje izvorne programe korisnicima, već samo odgovarajuće izvršne programe.

Ideja na kojoj je baziran viši programski jezik je veoma jednostavna. Ako već u računaru postoji operativna memorija u kojoj se mogu pamtiti podaci ili instrukcije, zašto ne iskoristiti tu osobinu memorije i popamtiti, na primer niz asemblerskih instrukcija kojima se sabiraju, množe, oduzimaju brojevi, računaju logaritmi itd. Svi ti "skupovi" instrukcija "pamte" se pod pogodnim imenima i mi samo, pozivajući se na ta imena (skup instrukcija jezika), pišemo program, odnosno određujemo tok računarskih i logičkih operacija u računaru. Tako se program napisan na

višemprogramskom jeziku prvo u mašini "prevede" objektni kod pa na mašinski jezik i tek onda, kada ga mašina razume, znači ako nema grešaka, on se izvršava.

Objektni program: Rezultat prevođenja jednog izvornog programa korišćenjem asemblera ili nekog prevodioca. Objektni program je predstavljen nizom mašinskih instrukcija. Neke objektne programe je moguće odmah izvršiti, dok se drugi, pre izvršenja, moraju povezati sa drugim objektnim programima.

Izvršni program: Oblik programa koji omogućuje njegovo direktno izvršenje na jednom računaru.

OBJEKTNO ORIJENTISANI JEZICI

Objektno orijentisana arhitektura je arhitektura u kojoj je sve predstavljeno kao objekat (procesi, tekst, U/I-operacije itd.). Objekti su srukture podataka u memoriji kojima manipuliše celokupan sistem (hardver i softver), oni daju visokorazinski opis koji uzima u obzir visokorazinski korisnički interfejs. Objekti imaju razne deskriptore – imena, ukazivače i obeležja. Ovi deskriptori takođe sadrže informaciju o tipu objekta i opis mogućnosti datog objekta. Sistemi objektivne arhitekture se mogu smatrati proširenjem ili generalizacijom sistema kvalifikativne arhitekture, pa se i na njima mogu temeljiti zaštita i računarska bezbednost.
Objektno orijentisani programski jezik je programski jezik za objektno orijentisani programski sistem. U jednom takvom sistemu se koncept procedure i podataka, realizovan u konvencionalnim programskim sistemima, zamenjuje konceptima objekata i poruka: objekt predstavlja pakovanje informacija i opis za njihovu manipulaciju, a poruka je specifikacija jedne od manipulacija objektom. Za razliku od procedura koje opisuju kako se izvodi manipulacija, poruka specifikuje šta pošiljalac želi da se uradi, a primalac određuje šta je se tačno dogoditi.

INSTRUKCIJE RAČUNARA

S opšteg stanovišta računara, suština svakog digitalnog izračunavanja je da se kao rezultat dobije rešenje u obliku funkcije:

$$X_1 = F_1(x_1, x_2 .... x_n)$$
$$Y_2 = F_2(x_1, x_2 .... x_n) \qquad 1.1$$

za elemente date na ulazu $x_1, x_2 ....$. U oba slučaja ulazne i izlazne reči računara predstavljaće skup **0** i **1**, koje su na odgovarajući način smeštene u registrima računara i memorijskih ćelija. Reči mogu biti predstavljene različitim skupom brojeva i alfanumeričkih nizova, koji opisuju problem logičkim stanjima.

Računar: Uređaj ili sistem koji je u stanju da obavlja neku sekvenciju operacija na jasno i eksplicitno definisan način. Te operacije su često numerička izračunavanja ili manipulacije podacima, ali obuhvataju takođe i ulaz/izlaz operacije u okviru sekvencije mogu da zavise i od određenih podatkovnih vrednosti. Definicija sekvencije se ziove program. Računar ima ili uskladišteni program ili ožičeni program.

Tražena rešenja mogu biti veoma dugačka i izuzetno komplikovana. Stoga moraju biti razbijena u elementarne matematičke relacije koje su definisane skupom instrukcija računara. Takav sređen niz elementarnih instrukcija, utvrđen radi sprovođenja određene operacije računara, naziva se program.

Instrukcija: Opis operacije koju treba da izvede računar. Sastoji se iz iskaza operacije koja treba da se izvede i neke metode specificiranja operanada (ili njihovih lokacija) i odredišta rezultata operacije. Instrukcije se često dele u klase kao što su aritmetičke, logičke, U/I itd. Mogu, ali ne moraju biti fiksne dužine.

Program: Skup iskaza koji se može dati kao celina jednom računarskom sistemu i koristiti za usmeravanje ponašanja sistema. Proceduralni program precizno definiše proceduru koju računarski sistem treba da sledi da bi se dobio traženi rezultat. Nasuprot tome, neproceduralni program specificira ograničenja koja proizvedeni rezultati moraju zadovoljiti ali ne daje proceduru dolaska do rezultata, nju bira sam računar.

Da bismo mogli da utvrdimo koji računar može ekonomičnije i efikasnije da obavi određeni zadatak, tj. da reši problem, najsigurnije je ako se napiše program na asembler jeziku i uporedi:

- broj potrebnih memorijskih reči da bi se program stavio u memoriju i
- broj ciklusa računara da bi se taj prgram izvršio.

Ciklus izvršavanja je najbolje merilo brzine rada računara. Stvarno vreme - t_r - izvršavanja predstavlja proizvod iz broja ciklusa - n_c - i vremena trajanja ciklusa - t_c :

$$t_r = n_c \cdot t_c \qquad 1.2$$

S gledišta pouzdanosti rada računara povoljnije je da se program izvršava s manjim brojem ciklusa, a uvećanim vremenom trajanja ciklusa.

Na taj način možemo konstatovati da jedino asemblerski jezik omogućava stvarno poređenje efikasnosti različitih računara. Putem programa s višim jezicima teško je, pa čak i nemoguće, da se oceni broj procesa vraćanja operacija, prekidanja procesa, memorijskih lokacija i drugih faktora bitnih za efikasno izvršavanje programa. U takvom slučaju je potrebno istovremeno uključiti u analizu i kompilator (compiler – prevodilac) programa kao i računar.

Asemblerski jezik: Notacija kojom se programi u mašinskom kodu predstavljaju u čitljivom obliku. Asemblerski jezik dozvoljava programeru da koristi alfabetske kodove operacija sa mnemoničkim značenjem, da sam bira simbolička imana za mašinske i memorijske registre i kako je njemu najzgodnije specificira adresirajuće sheme. Takođe dozvoljava upotrebu različitih brojnih osnova za numeričke konstante, a korisniku omogućava da pripoji obeležja programskim linijama tako da ostali delovi programa pomoću simbola mogu upućivati na te linije.

PODELA INSTRUKCIJA

Svaki računar ima određeni broj instrukcija koje koristi, a koji zavisi od kapaciteta i vrste računara, odnosno njegove namene. Prilikom izrade programa mi smo ograničeni tim unapred definisanim skupom.

Sve instrukcije dele se prema tome kakav saobraćaj podataka unutar računara izazivaju. Tako imamo podelu prema delovima računara unutar kojih se kreću podaci usled dejstva instrukcije ili prema tome kakav zadatak obavlja koja instrukcija (Tabela 9).

Tabela 9: Podela instrukcija prema delovima računara

instrukcije	**opis instrukcija**
memorijske instrukcije	odnose se na tok podataka u vezi s memorijom (memory reference instructions)
registarske instrukcije	odnose se na tok podataka između i u vezi registara (register reference instructions)
ulazno-izlazne instrukcije	odnose se na kretanje podataka od periferne jedinice ka centralnoj jedinici i obrnuto (input-output reference instructions)

Instrukcije možemo još svrstati u sledeće grupe (tabela 10.):

Tabela 10.: Podela instukcija prema zadacima

instrukcije	**opis instrukcija**
aritmetičke i logičke instrukcije	definišu operaciju između dva operanda, određujući istovremeno mesto izvora i smeštaja podataka, mesto smeštaja gde se, u stvari, nalazio drugi operand postaje i mesto rezultata tako da menja sadržaj
instrukcije za izmeštanje podataka	koriste se za transfer podataka iz memorije u radni registar – akumulator i nazad, ovom instrukcijom određen je izvoz podataka, smer kretanja i odredišno mesto
upravljačke instrukcije	koriste se za donošenje odluke na bazi statusa u toku procesa, ispitivanjem statusa dovode do grananja kojim se program dalje usmerava
operacijske instrukcije	se odnose na samo jedan operand, obično onaj u akumulatoru, stoga se koriste za brisanje sadržaja akumulatora, rotiranje, komplementiranje, pomeranje i slično
instrukcije za uvođenje podprograma	služe da se ostvari grananje procesa s glavnog programa na podprogram i povratak nazad na glavni

MIKROPROCESOR INTEL 8086

Šesnaestobitni mikroprocesor Intel 8086 je kompatibilan na nivou zbirnog jezika sa svojim osambitnim prethodnicima (Intel 8080 i 8085). Intel 8086 ima dodatne procesne sposobnosti:

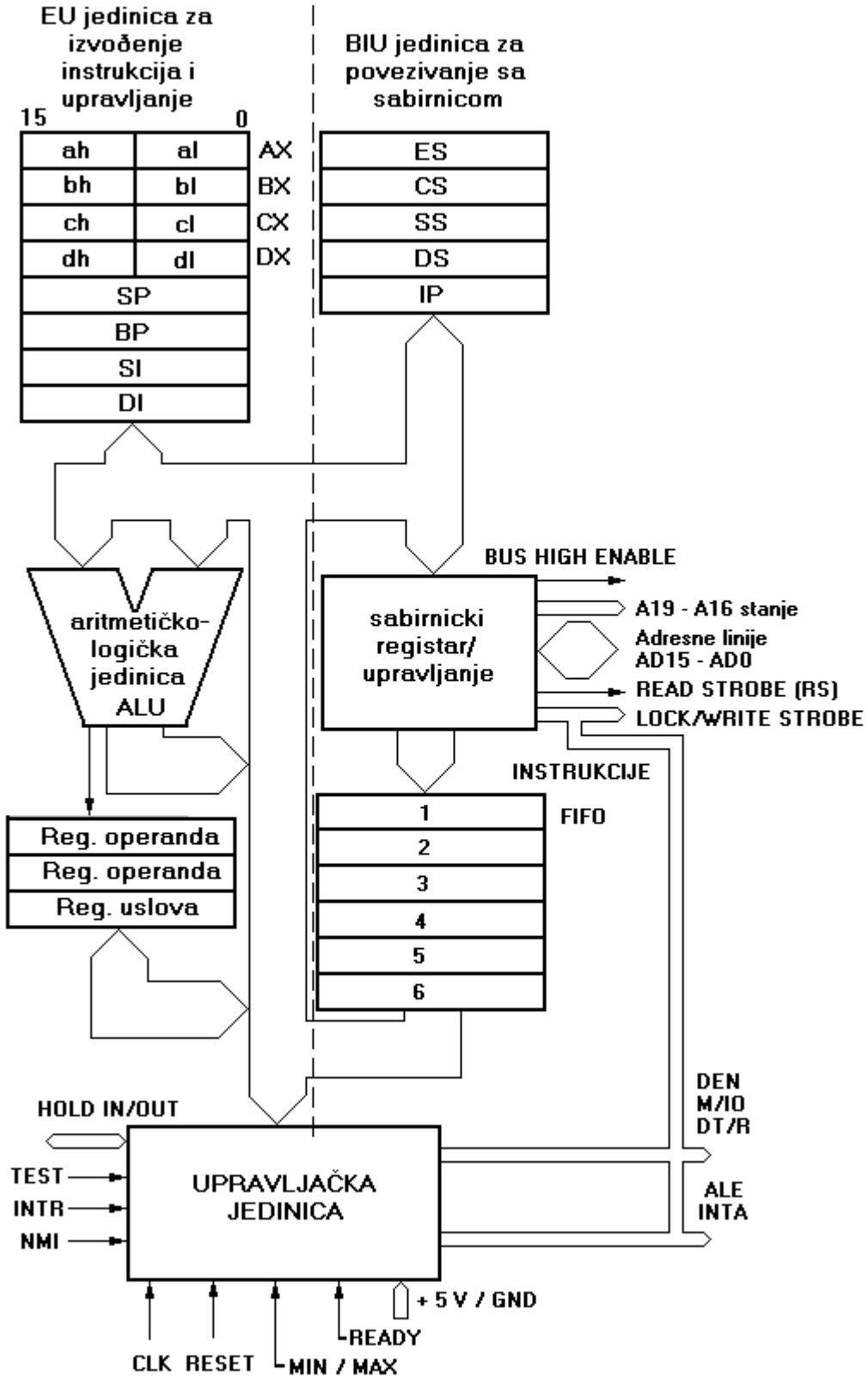
- 8-bitnu i 16-bitnu aritmetiku (uključujući množenje i deljenje),
- efikasne operacije za rukovanje bitovima,
- skup instrukcija za izvođenje operacija za rukovanje nizovima bajtova i nizovima reči (byte, word, string),
- dinamičku relokatibilnost programa i
- direkto adresibilan prostor od 1 M bajta i U/I prostor od 64 kB adresibilnih

Blok šema mikroprocesora I8086 je na slici 9. Funkcionalno mikroprocesor I8086 može se podeliti na dva dela:

- jedinicu za izvođenje instrukcija i upravljanje (EU – Execution Unit) i
- jedinicu za povezivanje sa sabirnicom (BIU – Bus Interface Unit).

U jedinici za izvođenje i upravljanje EU nalaze se četiri šesnaestobitna reistra, koji su adresibilni i kao osambitni registri (ah, al, bh, bl, ch, cl, dh i dl), dva šesnaestobitna pokazivača (SP – stack pointer i BP – pokazivač baze), dva indeksna registra (izvorni indeksni registar SI i odredišni indeksni registar DI), te šesnaestobitna aritmetička-logička jedinica sa dva šesnaestobitna registra operanda nedostupna programeru. Indikatorski registar (registar uslova) je također u sklopu jedinice EU i ima sledeće zastavice:

- AF (Auxiliary Carry) – pomoćni prenos,
- CF (Carry) – prenos,
- DF (Direction) – smer rukovanja na nizu / auto-inkrementiranje ili auto-dekrementiranje,
- IF (Interrupt Enable) – omogući prekid,
- OF (Overflow) – pretek,
- PF (Parity) – parnost,
- SF (Sign) – predznak i
- TF (Trap) – zastavica zamke, postavlja procesor u stanje izvođenja programa korak po korak.



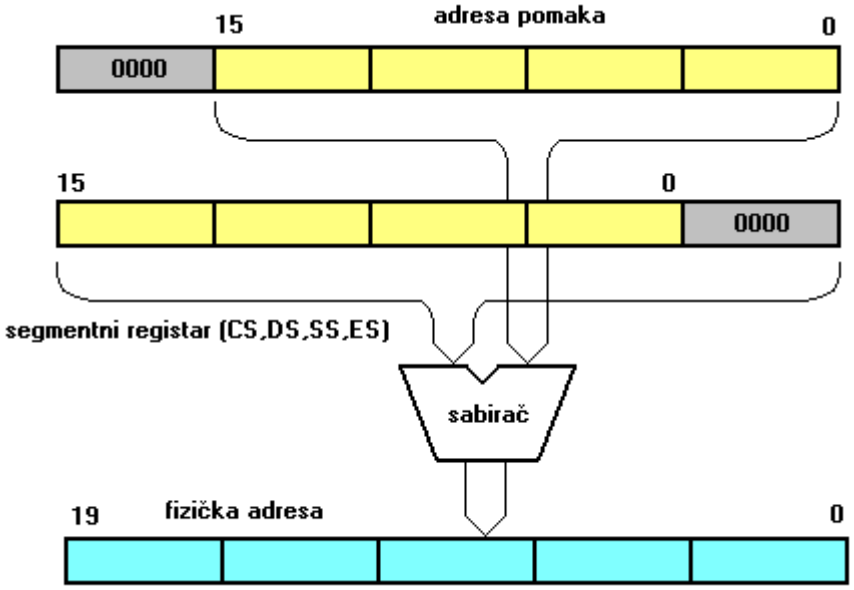
Slika 9. : Funkcionalna blokšema mikroprocesora I8086

Pre opisa jedinice za povezivanje sa sabirnicom (BIU) potrebno je opisati organizaciju memorijskog prostora za mikroračunar na osnovi mikroprocesora I8086.

Adresibilni memorijski prostor se sastoji od 1 M bajta direktno adresabilnih lokacija. Budući da mikroprocesor I8086 ima 16-bitnu aritmetičku jedinicu rukuje sa šesnaestobitnim podacima, potreban je dodatni mehanizam za izračunavanje adresa u 1 M bajtu memorijskog prostora. Memorija mikroračunara na osnovi I8086 podeljena je na segmente, od kojih svaki ima maksumalnu veličinu od 64 k bajta. Svaki segment započinje na adresi koja je deljiva sa šesnaest. U svakom trenutku sadržaji četiri takvih segmenata su direktno adresibilni. Segmenti se mogu prekrivati. Ti segmenti su:

- CS – tekući kodni segment,
- DS – tekući segment podataka,
- SS – tekući segment stack-a i
- ES – posebni segment.

Šesnaest značajnijih bitova adrese svakog tekućeg segmenta sadržani su u 16-bitnim segmentima-registrima, koji su elementi jedinice BIU. Slika 10 prikazuje generisanje adrese uz pomoć segmentnog registra. Na primer, sadržaj kodnog registra CS određuje tekući kodni segment, adresa instrukcije je relativna u odnosu na sadržaj CS i sadržaj IP (brojač instrukcije), koji je pomak u odnosu na CS.



Slika 10.: Generisanje adresa uz pomoć segmentnog registra

Dvadesetobitna fizička adresa dobijena je sabiranjem šesnaestobitne adrese pomaka i šesnaestobitne adrese segmenta sa 4 pridodana najmanje značajna bita koja su jednaka nuli. Funkcije segmentnih registara u jedinici BIU su sledeće: sadržaj registra CS definiše tekući kodni segment. Sve instrukcije se čitaju u odnosu na CS, a sadržaj pokazivača instrukcije IP (Instruction Pointer) služi kao ofset. Sadržaj registra DS definiše tekući segment podataka. Svako pozivanje podataka relativan je u odnosu na sadržaj registra DS,

Stog (stack):
1.LIFO stog. Linearna lista u kojoj se svi pristupi, umetanja i odstranjenja odvijaju na jednom kraju liste, koji se zove vrh (top). To podrazumeva pristup na bazi poslednji unutra – prvi napolje: element koji je poslednji umetnut u listu, prvi će biti odstranjen.
2. FIFO stog. Kada je prvi umetnuti element u linearnu listu ujedno i prvi element koji se odstranjuje, reč je o potisnom stogu, poznatijem kao red čekanja.

izuzev onih koji se koriste registrima BP (Base Pointer) i SP (Stack Pointer).

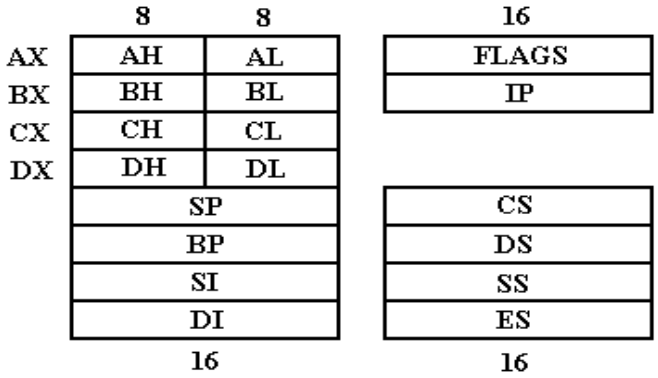
Sadržaj registra SS (Stack Segment) definiše tekući segment stack-a (stoga). Svako pozivanje podataka koje se koriste registrima SP ili BP relativan je u odnosu na sadržaj registra SS (tu su uključene operacije pohranjivanja u stog, uzimanja iz stoga, pozivanje, prekidi i operacija povratka iz potprograma).

Sadržaj registra ES (Extra Segment) definiše tekući posebni segment. U ovom segmentu je video memorija.

Programi, koji se menjaju, odnosno ne rukuju segmentnim registrima su dinamičko relokatibilni. Takav program može biti prekinut, memorisan u memoriju na novim lokacijama i ponovo upućen sa novim sadržajima segmentnih registara.

U sastavu jedinice za povezivanje sa sabirnicom nalazi se struktura FIFO od šest registara za privremeno memorisanje i prosleđivanje učitanih instrukcija. Učitane instrukcije čekaju u repu i ulaze u upravljačku jedinicu. Na taj način se povećava brzina rada procesora.

Slika 11 prikazuje programski model mikroprocesora I8086, koji se sastoji od skupa registara opšte namene (AX, BX, CX i DX), skupa registara pokazivača i indeksnih registara (SP, BP, SI i DI), skupa segmentnih registara (CS, DS, SS i ES), pokazivača instrukcija (IP) i registara uslova (FLAGS).



Slika 11.: Programski model mikroprocesora I8086

Skup instrukcija sastoji se od 97 osnovnih tipova instrukcija koje omogućuju rukovanje bitovima, rečima i nizovima. Skup instrukcija može biti podeljen na šest podskupova:

- instrukcije za prenos podataka,
- aritmetičke instrukcije,
- logičke instrukcije,
- instrukcije za rukovanje nizovima,
- instrukcije za prenos upravljanja i
- insrukcije za upravljanje procesorom.

Mikroprocesor I8086 ima bogatiji izbor načina adresiranja u odnosu na svoje osambitne prethodnike. Instrukcije koje zahtevaju dva operanda omogućuju da jedan od njih bude smešten ili u registru ili u memoriji, odnosno da drugi operand bude u registru, ili da bude konstanta. Sve operacije sa dva operanda (izuzev množenja, deljenja i operacija na nizovima) omogučuju da se jedan operand pojavljuje u

instrukcijskoj reči kao usputni podatak. Operande u memoriji moguće je adresirati direktno sa 16-bitnim pomakom ili indirektno sa dodatnim baznim registrom (registar BX ili registarski pokazivač baze BP služe kao bazni registri) ili/i indeksnim registrom (SI ili DI) i uz to sa 8-bitnim ili 16-bitnim pomakom.

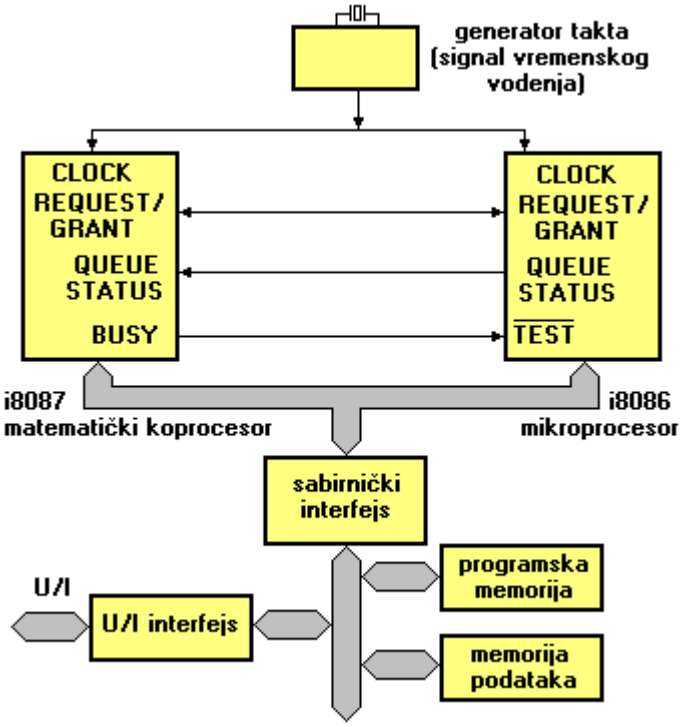
Prekidnu strukturu čine dve vrste prekida:

- nemaskirajući prekid i
- maskirajući prekid.

Prekidom se prenosi upravljanje na novu lokaciju u novom kodnom segmentu. Tablica prekidnih vektora sadrži 256 elemenata i smeštena je u memoriji. Svaki element se sastoji od četiri bajta i sadrži adresu pomaka i adresu kodnog segmenta programa za posluživanje prekida. Svaki element odgovara jednom tipu prekida, koji su numerisani od 0 do 255.

U sastavu mikroprocesora I8086 nalazi se i matematički koprocesor I8087, koji izvodi kompleksne matematičke operacije u aritmetici sa pokretnom tačkom (sabiranje, oduzimanje, množenje, deljenje, izračunavanje drugog korena itd.), logičke operacije, izračunavanje trigonometrijskih, hiperboličnih i logaritamskih funkcija.

Slika 12. prikazuje način povezivanja mikroračunara na osnovi i8086 i matematičkog koprocesora I8087 u snažan računar.

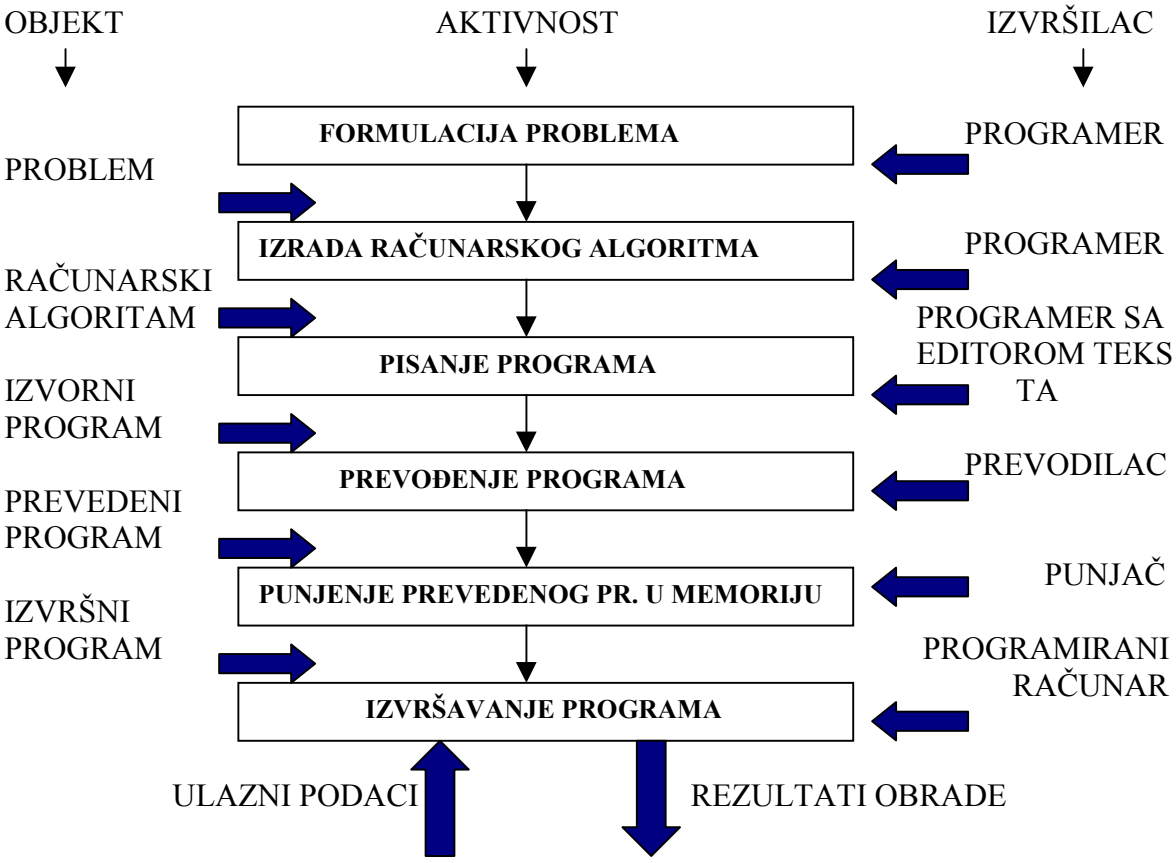


Slika 12.: Povezivanje mikroprocesora I8086 s matematičkim koprocesorom I8087

TEHNIKA PISANJA PROGRAMA NA ASEMBLERU

Program za mikroračunar se stvara u nekoliko sekvencionalnih faza. Proces prevođenja (asembliranja) predstavlja samo jednu fazu u računarskoj obradi podataka. Za slučaj programiranja na simboličkom (mnemoničkom) jeziku ceo proces izrade i izvršavanja programa prikazan je na slici 1.Ovaj proces započinje formulacijom računarski rešljivog problema, a završava se izvršnim mašinskim programom pomoću kojega se na osnovu skupa ulaznih podataka dolazi do skupa rezultata obrade. Treba uočiti da u toku proizvodnje izvršnog programa programeri koriste tri fundamentalne softverske alatke:

- editor teksta,
- prevodilac (asembler) i
- punjač.



Slika 1.: Proces izrade računarskih programa – osnovne aktivnosti i izvršioci

ANALIZA PROBLEMA

Sem uobičajenih aktivnosti u analizi problema neophodno je zbog ograničenosti mogućnosti računara razmotriti sledeće:

- veličinu i vrstu numeričkih podataka (mogu se obavljati samo celobrojne operacije sa brojevima apsolutne vrednosti – bajt, reč, ili postoji aritmetika u pokretnom zarezu),
- potrebne aritmetičke operacije (mogu se direktno realizovati samo osnovne dve, ili četiri) i

- predvidljivi broj koraka (ograničen je kapacitet operativne ili glavne memorije – programske memorije i memorije podataka).

IZRADA GRUBOG DIJAGRAMA TOKA

U ovoj fazi se postavlja koncepcija rešenja vodeći računa o elementima definisanih u analizi problema. Faza u mnogočemu je istovetna sa postavljanjem dijagrama toka za više programske jezike (**C**, **PASCAL** itd.).

Dijagram toka: metoda predstavljanja logike jednog programa. Dijagram toka koristi normirane geometrijske slike (pravougaonik, romboid i sl.) za predstavljanje pojedinih logičkih koraka jednog programa, kao i njihov redosled.

PLAN OPERATIVNE ILI GLAVNE MEMORIJE

Za sve promenljive koje se koriste u programu mora se odrediti memorijska lokacija. Za promenljive se moraju predviti lokacije (0. lokacija ≤ PROMENLJIVA ≤ zadnja lokacija).

Promenljiva: U programiranju, simboličko ime neke memorijske lokacije čiju vrednost je moguće menjati.

IZRADA DETALJNOG DIJAGRAMA TOKA

U ovoj fazi se detaljno razradi koncepcija postavljena u tački 'IZRADA GRUBOG DIJAGRAMA TOKA' u saglasnosti sa planom operativne memorije iz tačke 'PLAN OPERATIVNE ILI GLAVNE MEMORIJE' i mogućnostima računara. Dijagram toka mora predvideti sve izmene sadržaja registra i memorijskih pozicija.

TESTIRANJE DIJAGRAMA TOKA

Za kontrolu ispravnosti postavljenog dijagrama toka treba uzeti mali broj test podataka nad kojima se predviđene operacije lako realizuju i rezultati su poznati. Pri tom treba paziti da test podaci obuhvate sve predviđene mogućnosti.

Testiranje: Provera pravilnog rada jednog programa korišćenjem izabranih podataka (podaci za testiranje). Potpuni test jednog programa (tj. test sa svim mogućim podacima za testiranje) nije izvodljiv, zbog ogromnog broja mogućih programskih puteva i ulaznih podataka. Zbog toga izbor podataka za testiranje igra veoma važnu ulogu u pouzdanosti jednog testa.

ZAPIS UNOŠENJE PROGRAMA U MNEMONIČKOM KODU

Uz pomoć ekranskog tekst editora treba upisati program u datoteku, odnosno u glavnu memoriju.

Editor: Program, koji uz pomoć periferijskih uređaja (najčešće tastature i ekrana), omogućuje unošenje i menjanje informacije. Unesena informacija se obično memoriše u datoteke. Ako je informacija data u obliku teksta, govorimo o editoru za obradu teksta. U računarstvu se često javljaju editori koji su specijalno pravljeni za programiranje u nekom programskom jeziku. Takvi editori se nazivaju jezički editori.

IZRADA DOKUMENTACIJE

Dokumentacija o programu mora sadržati sledeće:

- opis problema,
- grubi dijagram toka,
- plan operativne memorije,
- detaljan dijagram toka,
- program u mnemoničkom kodu i
- uputstvo za korišćenje programa.

Programska dokumentacija: Skup svih pisanih tekstova koji se odnose na jedan program ili programski sistem. Programska dokumentacija predstavlja važan deo programiranja i neophodno je potrebna kod primene programa. U zavisnosti od osoba kojima je dokumentacija namenjena, razlikujemo sistemsku i korisničku dokumentaciju.

Uputstvo za korišćenje programa mora obuhvatiti način zadavanja i broj ulaznih parametara kao i formu saopštavanja rezultata.

Sistemska dokumentacija: opisuje interni rad i konstrukciju programa. Ovaj deo dokumentacije je namenjen programerima, koji se staraju o održavanju programa.

Korisnička dokumentacija: namenjena je korisniku i ona opisuje način korišćenja i mogućnosti programa.

INTEL 8086 ASEMBLER

Pitanje je, zašto učiti i koristiti asembler, kad višeprogramski jezici, odnosno objektno orijentisani jezici programeru daju moćan razvojni sistem? Činjenica je, da ovi jezici ne mogu optimalno iskoristiti resurse hardvera. U manjim i jednostavnijim sistemima operativna memorija ima svega nekoliko kilobajtni kapacitet, a memorija za podatke često nekoliko desetina bajtova. Kad se radi o sistemima, gde su odgovori u realnom vremenu, odnosno kad se radi o sistemima na primer za upravljanje, regulaciju, merenje itd., onda vreme reakcije mikroračunarskog sistema je kritično. Kod ozbiljnijih sistema, na primer kod PC računara isto može da se pojavi zahtev za brze odgovore, na primer kod programa za igre. Sledeći zadatak pokazuje neke dobre osobine asemblera.

Zadatak: Na IBM PC računaru treba ispisati na monitor brojeve od 0 do 65535. Za realizaciju zadatka korišćeni su jezici:

- asembler I8086,
- Pascal (Borland Pascal 6.0) i
- Borland Turbo C 2.0.

Dužine tih programa, odnosno potrebno vreme rada tih programa se nalaze u tabeli br. 11. Ovi rezultati su zavisni i od konkretne konfiguracije računara, verzije prevodilaca, operativnog sistema, video kartice. itd., ali tendencije su očigledne.

Tabela 11.: Upoređivanje programskih jezika		
Programski jezik	**dužina programa [bajt]**	**vreme rada [sec]**
Borland Turbo C 2.0	**8994**	**2.4**
Borland Pascal 6.0	**3888**	**0.5**
Asembler	**80**	**0.15**

Iz ove analize sledi, da sa asemblerom možemo postići kraće vreme odgovora u odnosu na bilo koji višiprogramski jezik, a i program je kraći. Međutim, onu problematiku, gde veoma efikasno koristimo mogućnosti višiprogramskog jezika i realizujemo pomoću tih programskih jezika. Rad sa matricama veoma je jednostavan u Pascalu, za rad sa bazama podataka razvijeni su dBase programski jezici, za veštačku inteligenciju Prolog itd.. LINK program će napraviti jedinstveni izvršni kod od objektnog koda asemblera, Pascala, Prologa itd.

PROCESORI INTEL I80286, I80386, I80486, PENTIUM, PENTIUM II I PENTIUM III

Novije tehnologije uvek daju mogućnost za povećanje takt frekvencije, a istovremeno i za kompleksnije hardversko rešenje, na primer za veću dužinu reči procesora. Firma Intel kod razvoja sledeće generacije procesora ima filozofiju, da prethodne rezultate ne treba odbaciti, nego treba integrisati u novija rešenja, ovo postižu tako, što su nove generacije mikroprocesora kompatibilni sa prethodnim procesorima.

Za početak dovoljno je naučiti asembler i8086, a ako neko želi da iskoristi mogućnosti Pentiuma, samo treba da dodaje instrukcije ovog procesora.

JEDNOSTAVNI i8086 ASEMBLERSKI PROGRAMI

Ako kreiramo u DOS-u jedan poddirektorijum sa nazivom ASS i radimo u ovom direktorijumu, onda pomoću jedne poznate DOS naredbe možemo u mašinskom kodu napisati program za ispisivanje bilo kojeg karaktera na monitor (ekran) računara. Iz DOS-a znamo, da svaki fajl pored naziva (ime fajla) ima i jednu ekstenziju, iz koje ekstenzije neki programi prepoznaju tip fajla. Na primer, *.pas je Pascal program, *.C je C program itd. Izvršni programi su (koji mogu odmah raditi):

- ***.exe** i
- ***.com**.

DOS (disk operativni sistem): Operativni sistem koji upravlja prenosom podataka od računara ka disk-jedinici i obratno. DOS je uvek sastavni deo softvera računara.

Direktorijum (katalog): Spisak svih datoteka koje se nalaze na jednom nosiocu podataka sa direktnim pristupom (disketa, tvrdi disk, CD-ROM, drugi računar u mreži). Operativni sistem upravlja katalogom jednog nosioca podataka preko sistema za upravljanje datotekama.

Upisivanjem:
copy con prvi.exe ENTER

DOS naredbe kopiramo sa konzole (tastature) brojeve (mašinske instrukcije) direktno u **prvi.exe** fajl, koji fajl će biti snimljen na **HD** (Hard Disk – tvrdi disk) u radni direktorijum.

Brojevi programa su sledeći:
180, 2, 178, 49, 205, 33, 205, 32.

Upisivanje svakog broja je pomoću **ALT** tastera i broja, a na kraju posle otpuštanja tastera na ekranu će se pojaviti **ASCII** karakter broja, koji karakter ponekad ima čudan izgled. Posle upisivanja zednjeg broja sa **CTRL Z** znakom izlazimo iz **copy** programa, a istovremeno će i **fajl** biti snimljen u **radni direktorijum**.

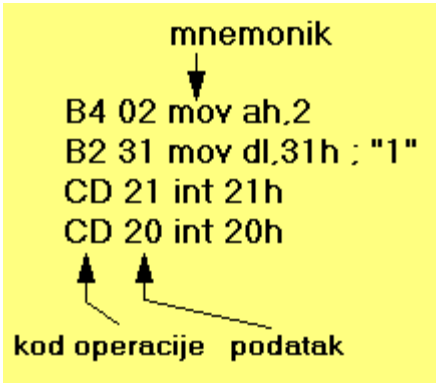
Sa **dir DOS** naredbom možemo izlistati sadržaj radnog direktorijuma, gde će se pojaviti **prvi.exe** fajl. Jednostavno će izvršiti računar program, ako ukucamo **prvi.exe ENTER.** Ako smo sve ispravno radili, na monitoru će biti ispisan broj **1**.

Moramo biti veoma oprezni kod programiranja u mašinskom kodu, jer već i greška od 1 bita može da blokira računarski sistem, ili će prouzrokovati nepoželjene akcije računara.

Na slici 13. je listing istog programa, ali sad pored brojeva su i mnemonički kodovi naznačeni, odnosno nazivi registara. Kod programiranja u mašinskom kodu, a isto i u asembler programiranju najčešće koristimo heksadecimalni kodni sistem za brojeve. Ako preračunavamo **180d** u heksadecimalni oblik, dobćemo **B4h**, **2d** u heksadecimalni **02h** itd. Slovo na kraju je oznaka za brojni sistem:

- b – binarni,
- d – decimalni i

- h – heksadecimalni.



Slika 13.: Ispisivanje broja 1 na monitor računara

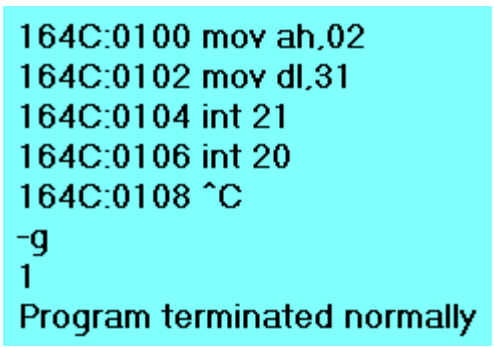
PROGRAMSKI PAKET **DEBUG**

Operativni sistem **DOS IBM PC** računara sadrži jedan ugrađeni programski paket za razvoj programa u asembleru. Program možemo pokrenuti sa ukucivanjem **DEBUG** naziva, nakon čega na monitoru će se pojaviti **'-' PROMPT** (znak spremnosti). Ovde možemo ukucati naredbe (jedno slovo). Ako kucamo '?' upitnik, dobićemo listu sa postojećim naredbama i eventualnim parametrima.

Naredba **'a'** je za aktiviranje veoma jednostavnog asemblera, u kojoj opciji **DEBUG** šalje na ekran segmentnu adresu i ofset adresu operativne memorije, odnosno mesto programa u memoriji. Pored ove adrese možemo ukucati naredbe mikroprocesora **i8086** u mnemoničkom kodu i operande, koji mogu biti registri procesora, ili brojevi, a brojeve isključivo u heksadecimalnom obliku (bez **h** sufiksa). Pošto ne možemo raditi sa simboličnim adresama i promenljivama, ovaj asembler nije pravi asembler (prevodilac). Prelazak na sledeći asemblerski red je sa **ENTER** tasterom, gde opet možemo u mnemoničkom kodu zadati naredbu. Postupak nastavimo do zadnje operacije, a na kraju programa sa **CTRL C** znakom, napušta sistem asemblera i ponovo je u **DEBUG** modu.

Valja napomenuti, da kod upisivanja mašinskih kodova pomoću **copy** naredbe DOS operativni sistem šalje program na tvrdi disk, dok upisivanje preko DEBUG-a je praktično asembliranje, a prevedeni program će biti direktno upisan u operativnu memoriju, a pošto je izvršni kod, može momentalno da radi.

Prethodno napisani postupak je na Slici 14.



Slika 14.: Upisivanje programa pomoću '**a**' (asembliranje opcije DEBUG-a u operativnu memoriju i izvršenje programa sa '**g**' naredbom

Na slici **164C** je adresa kod segmenta, a ovaj broj će odrediti operativni sistem, kod drugog računara neki drugi broj će biti ispisan. Adresa lokacije u kod segmentu je **0100h**, ovo znači da početak programa nije od nulte lokacije. Memorija od **0000h** do **00FFh** je zauzeta, tu su neki parametri sistema. Posle adrese **0100h** je sledeća adresa na slici **0102h**, zato, što prva instrukcija **mov ah,02** zahteva dva bajta, prvi bajt je kod operacije (**mov**), a drugi bajt je jednobajtni operand (**02**). Slučajno, u ovom programu sve operacije su dvobajtne. Na adresi **164C:0108 CTRL C** je naredba asembleru, da nema više instrukcija i treba da vrati komandu u **DEBUG**.

Posle unošenja programa i pojave '-' **PROMPT**-a **DEBUG** je spreman da prihvata sledeću komandu, a to je u ovom slućaju (slika 2.) '**g**', odnosno **go**, naradba, da **DEBUG** izvrši program. Zadatak programa je bio štampanje broja **1**. Nakon štampe **DEBUG** šalje poruku korisniku:

'Program terminated normally',

značі uspešno je završen program. Programer mora obezbediti pravilan izlazak iz programa, da bi **DEBUG** mogao korektno da nastavi rad. U suprotnom, računar će nekontrolisano raditi, i najverovatnije se blokirati. Istina, da kod **Windows multitasking** (multiprogramski) operativnog sistema **DOS** režim je samo jedan **task** (zadatak), pa jednostavno kod greške u **DOS**-u **Windows** će normalno nastaviti rad dalje.

Multiprogramski rad: Način rada jednog računarskog sistema, kod koga centralna jedinica naizgled istovremeno obrađuje više procesa.

Ako imamo program u operativnoj memoriji i želimo ga analizirati naići ćemo na poteškoće, jer čitanje programa u mašinskom kodu je praktično nemoguće. U ovom slučaju program **disasembler** vrati mašinski kod u mnemonički program, šalje na ekran **listing** (listu) programa. Aktiviranje disasemblera u **DEBUG** programskom paketu je sa '**u**' naredbom. Na slici 3 je disasembliran naš prvi program. Pošto se ne zna kraj programa, disasembler izlista jedan ekran, a ako je program kraći, posle zadnje instrukcije pojaviće se deo memorije sa mnemoničkim kodovima, koje instrukcije nisu elementi programa (na slici 15 se to ne vidi). Ti podaci mogu biti delovi prethodnih programa.

```
-u
164C:0100 B402          mov ah,02
164C:0102 B231          mov dl,31
164C:0104 CD21          int 21
164C:0106 CD20          int 20
164C:0108
```

Slika 15.: Rezultat disasembliranja programa **prvi**

ASCII (American Standard Code for Information Interchange): Skraćenica za američki standardni kod za razmenu informacije. Ovaj sedmobitni kod uspostavlja jednoznačnu vezu između slova, cifara, specijalnih i kontrolnih znakova, sa jedne strane, i kombinacije binarnih brojeva dužine sedam bitova, sa druge strane. U **ASCII**-kodu je moguće kodiranje 128 različitih znakova.

U programu programske linije imaju sledeća značenja:

- **mov ah,02** upisivanje broja **02** u značajniji bajt akumulatora **ax**,

- **mov dl,31** upisivanje **ASCII** koda broja **1** u manje značajniji deo **dx** registara,
- **int 21** aktiviranje ugrađene rutine operativnog sistema za ispisivanje jednog karaktera na ekran, ako je prethodno komanda **02** upisana u **ah** i ako je **ASCII** kod karaktera upisan u **dl** registar.
- **int 20** aktiviranje ugrađene rutine operativnog sistema, koja rutina će obezbediti ispravno vraćanje upravljanja u onaj program, odakle je pozvan potprogram (u ovom slučaju program je pozvan iz **DEBUG**-a).

'd' komanda **DEBUG**-a izlista deo sadržaja operativne memorije u heksadecimalnom i **ASCII** obliku. Ako ćemo izlistati naš program, onda od tih podataka samo prvih osam bajtova pripadaju našem programu, ostali podaci nemaju nikakav značaj za nas. Kod **ASCII** vrednosti '**...1.!**' broj **1** je kod našeg broja **1**, koji smo mi upisali u **dl** registar. '**.**' (tačka) će se pojaviti u slučaju, ako kod nema odgovarajući karakter, na primer ako je instrukcija ekrana (povrat kolica, novi red itd.).

U Tabeli 12. su ASCII kodovi. Ako iskoristimo i bit D7 za kodiranje, onda praktično još 128 kodova možemo definisati. U ovoj drugoj tabeli su specijalni karakteri, slova drugih (neengleskih) jezika itd.

Tabela 12: ASCII kodovi

	000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	**NUL**	**DLE**	**blank**	**0**	**@**	**P**	**'**	**p**
0001	**SOH**	**DC1**	**!**	**1**	**A**	**Q**	**a**	**q**
0010	**STX**	**DC2**	**"**	**2**	**B**	**R**	**b**	**r**
0011	**ETX**	**DC3**	**#**	**3**	**C**	**S**	**c**	**s**
0100	**EDT**	**DC4**	**$**	**4**	**D**	**T**	**d**	**t**
0101	**ENQ**	**NAK**	**%**	**5**	**E**	**U**	**e**	**u**
0110	**ACK**	**SYN**	**&**	**6**	**F**	**V**	**f**	**v**
0111	**BEL**	**ETB**	**'**	**7**	**G**	**W**	**g**	**w**
1000	**BS**	**CAN**	**(**	**8**	**H**	**X**	**h**	**x**
1001	**HT**	**EM**	**)**	**9**	**I**	**Y**	**i**	**y**
1010	**LF**	**SUB**	*****	**:**	**J**	**Z**	**j**	**z**
1011	**VT**	**ESC**	**+**	**;**	**K**	**[**	**k**	**{**
1100	**FF**	**FS**	**,**	**<**	**L**	****	**l**	**|**
1101	**CR**	**GS**	**-**	**=**	**M**	**]**	**m**	**}**
1110	**SO**	**RS**	**.**	**>**	**N**	**^**	**n**	**~**
1111	**SI**	**US**	**/**	**?**	**O**	**_**	**o**	**DEL**

'r' komanda **DEBUG**-a je za ispisivanje sadržaja registara procesora i **indikatorskih bitova** (zastavica).

'p' komanda **DEBUG**-a je za izvršenje programa korak po korak, nasuprot komande **g**, a istovremeno realizuje i '**r**' komandu.

Ako je zadatak štampanje slova '**A**', onda ne moramo ponovo pisati program, odnosno kompletno upisati u operativnu memoriju, nego sa **u** naredbom (**disasembler**)

izlistamo program, potražimo red, gde je štampanje broja **1** i na ovu poziciju unosimo novu instrukciju. Iz listinga vidi se da red **0102h** sadrži traženu instrukciju (**mov dl,31**), a na ovu poziciju treba upisati **mov dl,41**, jer iz **ASCII** tabele kod **A** karaktera je broj **0100 0001b**, odnosno **41h**, ili **65d**. Dole vidim kako treba to realizovati. Opet koristimo '**a'** komandu, ali je sad adresa konkretna, a **CTRL C** služi za izlazak iz asemblera.

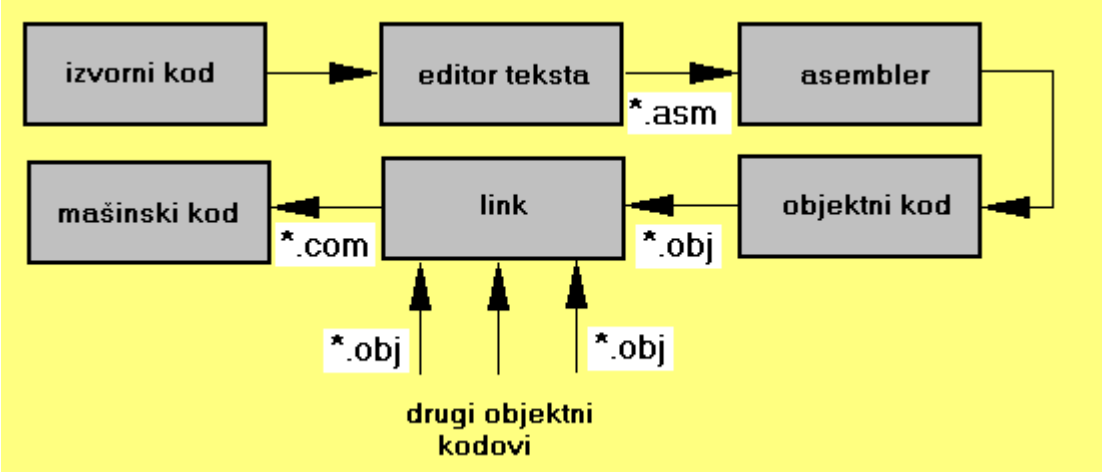
```
-a 0102
146C:0102 mov dl,41
146C:0104 ^C
-g
A
Program terminated normally
```

Slika 16.: Upisivanje novog reda u program

Istina da programski paket DEBUG omogućuje rad u operativnoj memoriji, programiranje U/I perifernih jedinica, ali mogućnosti programa su veoma ograničeni i praktično nije ga moguće efikasno koristiti u razvoju asemblerskih programa. Za ozbiljniji rad moramo izabrati pravi i moćan asembler i linker. Asembleri su u nekim operacijama slični, prema tome ako neko nauči koristiti asembler, prelazak na drugu verziju nije težak posao.

PISANJE, ASEMBLIRANJE I IZVRŠENJE ASEMBLERSKIH PROGRAMA

Za realizaciju programa su potrebni neki programski paketi, a faze realizacije mašinskog koda su prikazane na slici 17.



Slika 17.: Faze realizacije programa i pomoćni programski paketi

Izvorni program, odnosno onaj program koji sadrži mnemoničke kodove odgovarajućeg procesora i instrukcije korišćenog asemblera pomoću jednog **ASCII** editora teksta biće upisan u ***.asm** fajl i memorisan na tvrdom disku. Skraćenica **asm** označuje tip fajla. Editor teksta može da bude bilo koji **ASCII** editor, na primer

***.asm:** '*' (zvezdica) u operativnom sistemu DOS (kao i unekim drugim operativnim sistemima označuje ima (naziv) datoteke. Prvi znak je slovo, a ostali karakteri mogu biti ili slova, ili broievi.

Norton Commander sadrži kod funkcionalnog tastera F4 takvu opciju. Editor teksta Word ne daje ASCII fajl, pa prema tome ne koristimo ga za unošenje izvornog programa.

***.asm** datoteku moramo asemblirati, znači provoditi u mašinski jezik. Postoji takav asembler, koji može direktno iz **asm** fajla da napravi mašinski kod, međutim bolje je u prvom koraku prevoditi u ***.obj** (objektni fajl), zatim pomoću programskog paketa **link** povezati sa ostalim ***.obj** fajlovima (ako postoje). Na takav način bilo koji programski jezik možemo koristiti za razvoj delova programa, pa sa **link** programom od tih ***.obj** datoteka napraviti zajednički mašinski, odnosno **izvršni program**.

Za asembliranje koristimo **TASM** asembler, koji se program inače zove **Turbo Assembler**. Treba ukucati u **DOS**-u ime asemblera i ime asm programa:

tasm prvi.exe ENTER.

Asembleri obično imaju dva prolaza (two pass assembler):

- u prvom prolazu na osnovu simboličkih imena (naziva), promenljive i tipa promenljive ispunjuju **tabelu simbola**,
- a u drugom prolazu realizuju prevod mnemoničkih simbola u **mašinski** (binarni) **kod** i ispunjavaju sa brojevima tabelu simbola, tako realizajući ***.obj** datoteku.

Dobijeni objektni fajl je standardni i kompatibilan sa ostalim **obj** fajlovima kod drugih programskih jezika (C, Pascal, Prolog Dbase itd). Ova činjenica obezbeđuje povezivanje različitih programskih jezika u izvršni kod (samo treba kontinualno napisati imena tih programa, između dva programa je prazno mesto (blank). Za taj zadatak se koristi program **link**. Posle ukucivanja imena programa link program će pitati ime *.obj programa, zatim sledeći program itd., pa traži od nas naziv *.obj programa. Ako je odgovor **ENTER**, onda i obj fajl će imati isti naziv kao **asm** fajl. Na kraju link kreira ***.exe** izvršni fajl.

Šta je, ako smo napravili sintaktičku grešku? Već asembler signalizira, da ne može da analizira izvorni program, pa šalje poruku (poruke) korisniku, gde određuje grešku i poziciju te greške u listingu (koji red). Posle ispravke postupak prevođenja treba opet ponoviti.

STRUKTURA ***.exe*** I ***.com*** IZVRŠNIH PROGRAMA

Na Slici 18.a. je struktura ***.exe***, a na Slici 1.b. je struktura ***.com*** programa. Dužina ***.exe*** programa nije limitirana, dok ***.com*** program ne može da bude duži od jednog segmenta (a to je **64 kB**).

*.EXE

```
Kod     Segment
        assume
        CS:Kod,DS:Podatak,SS:Stack
Start:  .
        .
        .
Kod     Ends
Podatak Segment
        .
        .
        .
Podatak Ends
Stack   Segment
        .
        .
        .
Stack   Ends
        End     Start
```

a)

*.COM

```
Kod     Segment
        assume  CS:Kod,DS:Kod

        Org     100h
Start   .
        .
        .
Kod     Ends
        End     Start
```

b)

Slika 18.: Struktura ***.exe*** i ***.com*** programa

U listingu za početak segmenta se koristi izraz **Segment**, a ispred toga **labela**, koja je ime programa, a za označavanje kraja segmenta se koristi **Ends** izraz.

Naredba **assume** upisuje segmentne adrese u segmentne registre. Nije obavezno korišćenje **SS** registara. Kod ***.com*** programa početak **kod segmenta** i početak **segmenta za podatke** može da bude isto. **Org** je naredba **asemblera** kod ***.com*** programa i služi za određivanje početka programa. Ne smemo koristiti adrese od **0000h** do **00FFh**, jer tamo su sistemske promenljive.

BIOS (Basic Input Output System) softver kod **IBM PC** personalnih računara sadrži niz potprograma, koji su za razmenu podataka između procesora i memorije, odnosno procesora i perifernih jefinica. Korisno je ugrađivanje tih potprograma u naše programe. U programima uspešno možemo koristiti **int 21** potprogram, koji obezbeđuje korektan izlazak iz korisničkog programa. Parametar u ovom slučaju je **4C00h**, a treba ga upisati u **ax** registar. Ovaj izlazak obezbeđuje povratak u glavni program bez poruke.

1. Zadatak: Napisati korisnički program, koji korektno vraća upravljanje operativnom sistemu i ne šalje nikakvu poruku korisniku

Listing programa je:

```
;*********************************************************************
Prog01      Segment                         ;definicija segmenta
            assume CS:Prog01,DS:Prog01      ;pocetak CS I DS segmenta
;*********************************************************************
Start:      mov   ax,Prog01                 ;pocetak na DS
            mov   ds,ax
;*********************************************************************
            mov   ax,4c00h                  ;nazad u DOS
            int   21h
;*********************************************************************
Prog01      Ends                            ;kraj segmenta
            End   Start
;*********************************************************************
```

Analiza programa:

- Asembler ne prevodi tekst od '**;**' znaka do kraja reda, ovde može programer da piše komentar, (komentar je koristan za dokumentaciju).
- U ovom programu tri puta imamo **mov** instrukciju, mada formalno su iste, i imaju isti zadatak, prenos podataka, realizuju različite prenose. U Tabeli 13. vidimo nekoliko primera za instrukciju **mov**.

Tabela 13.: Neke mov instrukcije		
mov	**ax,1839**	u akumulator **ax** će biti upisan broj 1839
mov	**ax,3Bh**	u akumulator **ax** će biti upisan broj 3Bh
mov	**al,45**	u akumulator **al** će biti upisan broj 45
mov	**ax,bx**	u akumulator **ax** će biti upisan sadržaj **bx** registara
mov	**al,bh**	u akumulator **al** će biti upisan sadržaj **bh** registara
mov	**ax,labela**	u akumulator **ax** će biti upisana vrednost labele

Veoma veliki broj kombinacija postoji kod korišćenja instrukcija, ove kombinacije treba naučiti, da bismo mogli efikasno iskoristiti mogućnosti asemblera. Neke od tih su:

- posle mnemoničkog koda barem jedan parametar je registar,
- u prvi operand (odredište) ulazi sadržaj iz drugog operanda (izvor),
- sadržaj drugog operanda nakon operacije ostaje nepromenjen,
- ako je podatak u memoriji, onda između **[]** znaka je adresa lokacije, odakle podatak će biti upisan u prvi operand,
- ako je operand određen sa labelom, onda obavezno treba odrediti tip (**byte, word**),
- dimenzije operanada moraju biti iste (kod instrukcije mov ax,0B3h u **ax** će biti upisan **00B3h**).

Realizacija programa:

- Iz **Windows**-a prelazimo u **DOS** operativni sistem, kreiramo poddirektorijum (ako već ne postoji) i ovde u **Norton Commander**-u sa **SHIFT F4** funkcijom otvorimo praznu datoteku (fajl) sa imenom **prog01.asm**.
- Ukucamo gornji program.
- Aktiviramo asembler, parametar asemblera je ime fajla: **tasm prog01.asm**.

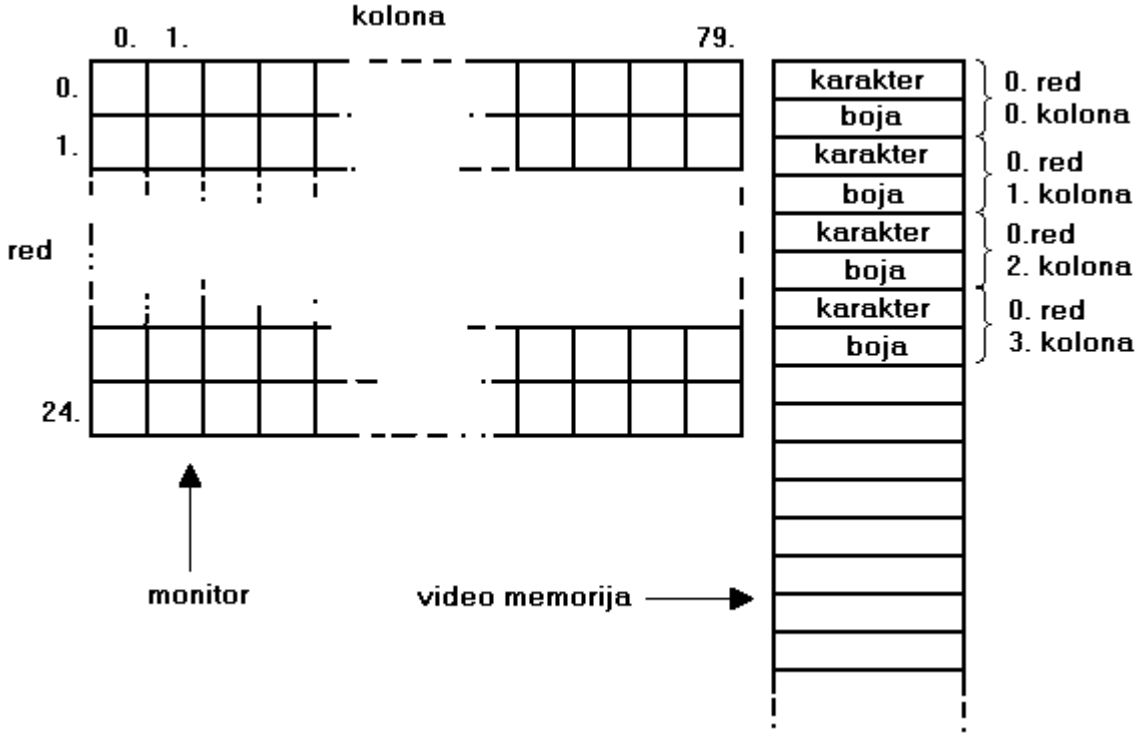
- Ako postoji greška, sa editorom otklonimo je i ponovimo prethodnu tačku.
- Pošto smo dobili **prog01.obj** ukucamo **link** i **ENTER**, i odgovorimo na pitanja, ukucamo **prog01.obj** i **ENTER**, a na ostala tri pitanja odgovorimo sa **ENTER**-om.
- Ako link program ne javlja grešku, uspešno je preveden program u izvršni kod sa imenom **prog01.exe**.
- Ukucamo **prog01.exe**, opet ćemo dobiti u **DOS**-u **PROMPT**.

2. Zadatak: Na monitor (ekran) u **7**. red i **13**. kolonu treba ispisati slovo **X**.

Analiza programa:

Kod **IBM PC** personalnih računara postoji video memorija od adrese **0B800h**, ova memorija se nalazi u operativnoj memoriji (**ES** - ekstra segment). Kod ovog primera nama treba tekstualni režim te memorije, gde možemo upisati **ASCII karaktere**. **IBM PC** ima **hardver** i **softver** za periodično ispisivanje sadržaja video memorije na monitor. Na slici 19. je prikazana organizacija monitora, odnosno video memorije. Vidi se, da za jednu poziciju u video memoriji dva bajta koristi sistem:

- prvi bajt je kod karaktera (**ASCII** kod),
- a drugi bajt služi za određivanje boja karaktera (u Tabeli 14 su dati kodovi boja).



Slika 19. Organizacija slike monitora i video memorije

Jednostavno možemo izračunati poziciju u video memoriji gde treba upisati kod karaktera i boju, ako znamo u koju poziciju ekrana želimo nešto upisati:

adresa = 80 * 2 * red + 2 * kolona (1.3.)

gde su:

- adresa – adresa u operativnoj memoriji,
- red - mesto na ekranu, koji red (od 0 do 24 – ukupno 25) i
- kolona – mesto na ekranu, koja kolona (od 0 do 79 – ukupno 80).

Ako želimo ispisati karakter u 7. red i 13-u kolonu, onda je adresa 1146d = 047ah, odnosno **0b800h:47ah**.

Tabela 14.: kodovi boje

bit	značenje	bit	značenje
0.	plava boja karaktera	4.	plava boja pozadine
1.	zelena boja karaktera	5.	zelena boja pozadine
2.	crvena boja karaktera	6.	crvena boja pozadine
3.	intenzitet boja karaktera	7.	periodično/kontinualno ispisivanje

Listing programa je:

```
;*********************************************************************
Prog02      Segment                         ;definicija segmenta
            assume CS:Prog02,DS:Prog02      ;pocetak CS i DS segmenta
;*********************************************************************
Start:      mov   ax,Prog02                 ;pocetak na DS
            mov   ds,ax
;*********************************************************************
      mov   ds,ax
      mov   ax,0b800h               ;adresa video memorije u es
registar
      mov   es,ax
;*********************************************************************
      mov   di,1146                 ;ofset adreasa u di
;*********************************************************************
      mov   al,"X"                  ;slovo X u al registar
      mov   ah,7                    ;crna pozadina, beli karakter
      mov   es:[di],ax              ;slovo u video memoriju
;*********************************************************************
            mov   ax,4c00h                  ;nazad u DOS
            int   21h
;*********************************************************************
Prog02      Ends                            ;kraj segmenta
            End   Start
;*********************************************************************
```

3. Zadatak: Ispisati jedan karakter na monitor, pozicija karaktera je data sa x i y koordinatama, program treba da izračuna adresu u video memoriji.

Listing programa je:

```
;*********************************************************************
Prog03      Segment                         ;definicija segmenta
            assume CS:Prog03,DS:Prog03      ;pocetak CS i DS segmenta
;*********************************************************************
Start:mov   ax,Prog03                       ;pocetak na DS
      mov   ds,ax
;*********************************************************************
      mov   ds,ax
      mov   ax,0b800h                       ;adresa video memorije u es
                                            ;registar
```

```
      mov   es,ax
;*****************************************************************
      mov   al,byte ptr [Y]               ;y
      mov   ah,0
      mov   bl,160                        ;0. red 80 * 2 bajta
      mul   bl                            ;ax = al * bl
      mov   di,ax                         ;upisati u di
;*****************************************************************
      mov   al,byte ptr [X]               ;x
      mov   ah,0
      mov   bl,2                          ;x * 2
      mul   bl
      add   di,ax                   ;pozicija karaktera u memoriji
;*****************************************************************
      mov   al,byte ptr [KARAKT]          ;ASCII kod karaktera
      mov   ah,byte ptr [BOJA]            ;kod boje
;*****************************************************************
      mov   es:[di],ax                    ;upisivanje karaktera
                                          ;u memoriju
;*****************************************************************
      mov   ax,4c00h                      ;nazad u DOS
      int   21h
;*****************************************************************
X:    db    40
Y:    db    12
KARAKT:db   "X"
BOJA: db    7
;*****************************************************************
Prog03      Ends                        ;kraj segmenta
            End   Start
;*****************************************************************
```

Analiza programa:

Za upisivanje podataka (broja) u memoriju služi **db** (define byte) **direktiva** (instrukcija asemblera). Na početak reda dolazai **labela**, **simboličko ime** broja, zatim direktiva **db** i kao parametar vrednost. Čitanje iz te memorije je jednostavno, samo treba se pozivati na promenljivu i označiti tip broja (na primer **byte**). U programu **prog03** na primer **x** pozicija je opisana sa **X: db 40** redom.

Sa instrukcijom **mul** vršimo množenje dva operanda, na primer **al** sa **bl**, a pošto su ovi operandi osmobitni, rezultat je šesnaestobitni, biće upisan u registar **ax**. Množenje registara **ax** sa **bx** daje tridesetodvobitni rezultat, koji će biti upisan u **dx** i **ax** registar.

4. Zadatak: Program treba da štampa tekst iz memorije. Dužina teksta je proizvoljna, kraj teksta je označena sa **ASCII** karakterom, koji ne može da bude ni slovo ni broj, na primer **0**.

Listing programa je:

```
;*****************************************************************
Prog04Segment                       ;definicija segmenta
      assume CS:Prog04,DS:Prog04    ;pocetak CS i DS segmenta
;*****************************************************************
Start:mov   ax,Prog04               ;pocetak na DS
      mov   ds,ax
```

```
;*****************************************************************
      mov   ds,ax
      mov   ax,0b800h         ;adresa video memorije u es registar
      mov   es,ax
;*****************************************************************
      mov   al,byte ptr [Y]               ;y
      mov   ah,0
      mov   bl,160                        ;0. red 80 * 2 bajt
      mul   bl                            ;ax = al * bl
      mov   di,ax                         ;upisati u di
;*****************************************************************
      mov   al,byte ptr [X]               ;x
      mov   ah,0
      mov   bl,2                          ;x * 2
      mul   bl
      add   di,ax                   ;pozicija karaktera u memoriji
;*****************************************************************
      mov   ah,byte ptr [BOJA]            ;kod boje
            mov   si,offset TEKST         ;adresa teksta u si
;*****************************************************************
cikl: mov   al,[si]                       ;podatak u al registar
(karakter)
      cmp   al,0                          ;kraj teksta?
      jz    kraj                          ;da, kraj
      mov   es:[di],ax                    ;karakter u video memoriju
      add   di,2                          ;sledeca pozicija u
memoriji
      inc   si                            ;sledeci karakter
      jmp   cikl                          ;stampati dalje
;*****************************************************************
kraj: mov   ax,4c00h                      ;nazad u DOS
            int   21h
;*****************************************************************
X:          db    30                      ;pozicija teksta
Y:          db    12
TEKST:      db    "VISA TEHNICKA SKOLA - SUBOTICA",0
BOJA: db    7
;*****************************************************************
Prog04      Ends                        ;kraj segmenta
            End   Start
;*****************************************************************
```

Analiza program:

Labela (label) je posebna oznaka jedne programske naredbe (programskog reda). Labela omogućava uspostavljanje veze između označene programske naredbe i nekog drugog dela programa. Ova veza se obično uspostavlja naredbom skoka, odnosno naredbom pozivanja potprograma (rutine). U asembleru posle labele mora da bude znak '**:**' (dvotačka).

Za upoređivanje dve vrednosti koriste se operacije **cmp**. Ove instrukcije zavisno od rezultata upoređivanja određuju indikatorske bitove (C, Z i O), setuju ili resetuju ih. Te

Indikator (flag - zastavica): Indikatori su univerzalno primenljivi kontrolni bitovi, koji se često koriste u mikroprocesorima. Oni pokazuju postojanje, odnosno nepostojanje određenog stanja sistema, ispunjenje, odnosno neispunjenje određenog uslova, kao i specijalne događaje u sistemu. Menjanje vrednosti indikatora može biti prouzrokovano određenim instrukcijama unutar sistema ili nekim spoljnim događajem.

istrukcije su veome slične instrukcijama oduzimanja (**sub**), sa razlikom da operandi ostaju nepromenjeni. Operacije transfera podataka (**mov**) ne određuju vrednosti zastavica, naime kod transfera podataka ne koristi se aritmetičko-logička jedinica, nasuprot tome **cmp** radi preko **ALU** jedinice.
U programu za testiranje rezultata upoređivanja konkretno se koristi **Z** (zero) bit.

Najvažniji indikatori su:

- **indikator nule** (Zero flag) je specijalizovani indikator koji služi za ispitivanje vrednosti smeštene u akumulator, u slučaju da je, posle neke operacije, vrednost svih bitova u akumulatoru jednaka nuli, indikator nule će imati logičku vrednost jedan (**Z = 1**), a ako bar jedan bit u akumulatoru ima vrednost jedinicu (**(A)** $\neq$ **0**), indikator nule će imati logičku vrednost nulu (**Z = 0**),
- **indikator prenosa** (Carry flag) je specijalizovani indikator, koji registruje prekoračenje nastalo u akumulatoru, u slučaju da rezultat neke aritmetičke operacije nije moguće smestiti u akumulator (zbog prekoračenja, odnosno negativnog prekoračenja), indikator prenosa će dobiti logićku vrednost jedan (**C = 1**).

5. Zadatak: Program treba da realizije logičku **I** funkciju između dva osmobitna broja, zatim treba da ispiše brojeve i rezultat. Program mora da briše ekran. Za realizaciju ponovljenih delova programa treba koristiti potprogram (ispisivanje teksta, brojeva).

Listing programa je:

```
;*********************************************************************
Prog05      Segment                         ;definicija segmenta
            assume CS:Prog05,DS:Prog05      ;pocetak CS i DS segmenta
;*********************************************************************
Start:mov   ax,Prog05                       ;pocetak na DS
      mov   ds,ax
;*********************************************************************
      mov   ax,0b800h               ;adresa video memorije u es
      registar
      mov   es,ax
;*********************************************************************
      mov   ax,3                            ;brisanje ekrana
      int   10h
;*********************************************************************
      mov   di,0*160                        ;pozicija 1. teksta (0,0)
      mov   si,offset TEKST1                ;pocetna adresa TEKST1
      call  tekstisp                        ;ispisivanje teksta
;*********************************************************************
      mov   bl,byte ptr [BR1]               ;prvi broj
      call  brojisp                         ;ispisivanje broja
;*********************************************************************
      mov   di,1*160                        ; pozicija 2. teksta (1,0)
      mov   si,offset TEKST2                ; pocetna adresa TEKST2
      call  tekstisp                        ;ispisivanje teksta
;*********************************************************************
      mov   bl,byte ptr [BR2]               ;drugi broj
      call  brojisp                         ;ispisivanje broja
;*********************************************************************
      mov   di,2*160                        ;pozicija teksta 'REZULTAT'
```

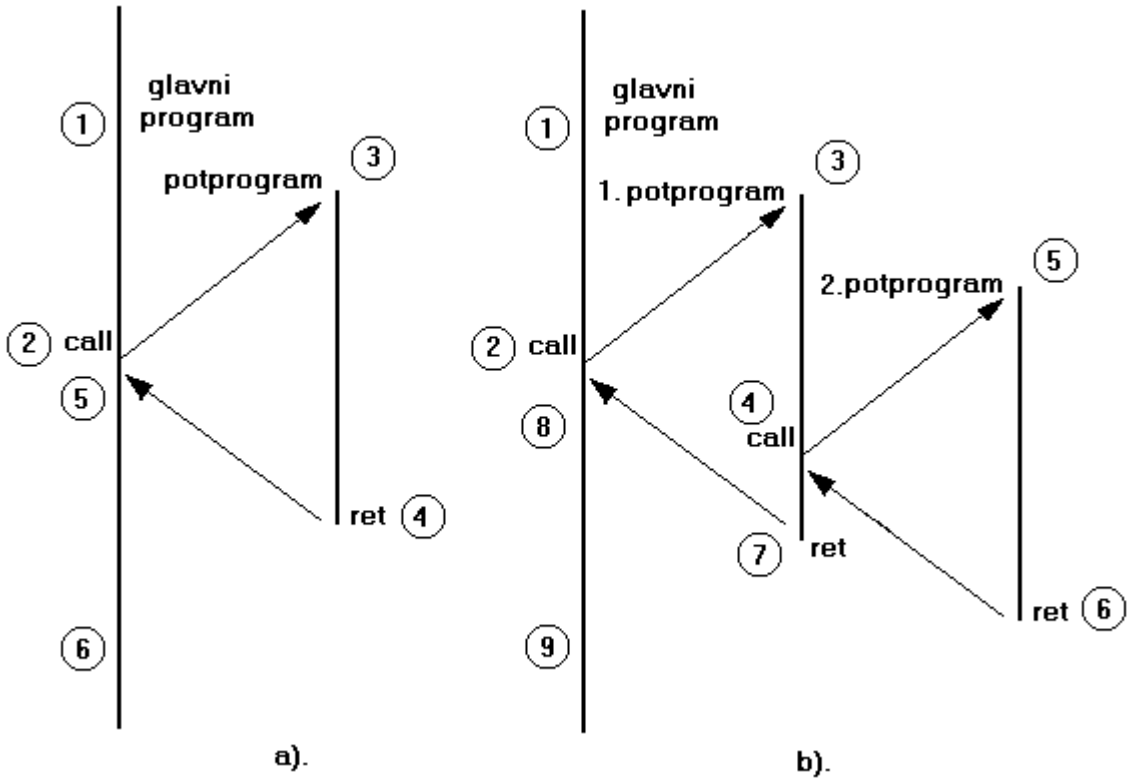
```
      mov   si,offset TEKST3              ;pocetna adresa TEKST3
      call  tekstisp                      ;ispisivanje teksta
;*********************************************************************
      mov   bl,byte ptr [BR1]             ;prvi broj
      and   bl,byte ptr [BR2]             ;BR1 AND BR2
      call  brojisp                       ;ispisivanje broja
;*********************************************************************
      mov   ax,0                          ;cekanje na taster
      int   16h
;*********************************************************************
kraj: mov   ax,4c00h                      ;nazad u DOS
      int   21h
;*********************************************************************
;#####################################################################
tekstisp    Proc                    ;rutina za ispisivanje teksta
      mov   cx,16                   ;tekst ima ukupno 16 karaktera
      mov   ah,15                   ;crna pozadina, beli karakter
;*********************************************************************
cikl1:mov   al,[si]                       ;karakter u al
      mov   es:[di],ax                    ;karakter u video memoriju
      add   di,2                          ;sledeca pozicija
      inc   si                            ;sledeci karakter
      loop  cikl1                   ;cx=cx-1 i skok ako je cx<>0
      ret                                 ;povratak u glavni program
szovir      Endp                          ;kraj potprograma
;#####################################################################
;*********************************************************************
;#####################################################################
brojisp     Proc                          ;potprogram za ispisivanje
                                          ;broja
      add   di,6                          ;+ 3 pozicije
      mov   cx,8                          ;broj je osmobitni
      mov   ah,15                   ;crna pozadina, beli karakter
;*********************************************************************
cikl2:mov   al,48                         ;ASCII kod nule (0)
      shl   bl,1                          ;CY <== d7 bit
      jnc   ki2                           ;ako 0, skok na izl2
      mov   al,49                         ;ASCII kod jedinice (1)
;*********************************************************************
izl2:mov    es:[di],ax                    ;broj u video memoriju
      add   di,2                          ;sledeca pozicija
      loop  cikl2                   ;cx=cx-1 i sko ako je cx <>0
      ret                                 ;povratak u glavni program
brjisp      Endp                          ;kraj potprograma
;#####################################################################
;*********************************************************************
TEKST1:     db    "1. BAJT       :"
TEKST2:     db    "2. BAJT       :"
TEKST3:     db    "REZULTAT      :"
BR1:        db    01011101b
BR2:        db    10101011b
;*********************************************************************
Prog05      Ends                          ;kraj segmenta
      End   Start
;*********************************************************************
```

Analiza programa:

U ovom primeru možemo analizirati pisanje i korišćenje potprograma. U programskim jezicima, kao i u asembleru programski deo koji za sebe predstavlja nezavisnu celinu,

ali se ne može zasebno izvršiti. U programu **prog05** na primer ispisivanje teksta, ili ispisivanje broja su potprogrami. Potprogram se uvek poziva od nekog drugog programa, to može da bude glavni program (Slika 20. a).) ili neki drugi potprogram (Slika 20. b).), ili potprogram može da zove samog sebe, ovo je rekurzija. U asembleru pozivanje potprograma se vrši sa naredbom **call** i navođenjem imena potprograma. Predaja i preuzimanje parametara je preko registara, odnosno promenljive koje su u memoriji. Kod viših programskih jezika predaja parametara je sa simboličkim imenima.

Rekurzivnost: Postupak, kod koga svaki korak u radu koristi rezultate prethodnih koraka. U programiranju ovaj pojam označava obično one potprograme, koji mogu da pozivaju sami sebe.



Slika 20.: a). Glavni program zove potprogram i b). Glavni program zove 1. potprogram, a 1. potprogram zove 2. potprogram

Posle izvršavanja potprograma upravljanje potprogramom preuzima pozivni program, u asembleru se to postiže sa **ret** naredbom na kraju potprograma.

Potprogrami se najčešće koriste za programiranje nekog programskog dela koji se više puta ponavlja. Rastavljanje programa u veći broj potprograma daje pregledniji program.

Registar **cx** je projektovan tako, da sa **loop** instrukcijom jednostavno možemo koristiti kao brojača. Pre početka petlje unosimo sa **mov** operacijom početnu vrednost brojača, koju vrednost na kraju petlje **loop** instrukcija dekrementira (**cx = cx – 1**) i istovremeno i testira, da li je vrednost **0**, ako nije onda izvršava skok na naznačenu poziciju, inače (**cx = 0**) napušta petlju i od sledeće tačke nastavi rad. Instrukcija **loop** je iz grupe instrukcije “uslovan skok”.

Ako je parametar **0** kod int **10h** potprograma onda sistem prelazi u **80*25** tekstualni rad.

Potprogram **int 16h** je koristan program iz **BIOS**-a, ako želimo da program čeka sve do pritiska tastera. Kad je pritisnut bilo koji taster, onda potprogram sačuva **ASCII** kod tastera u **ah** registru. U ovom primeru nismo isčitali tu vrednost.

6. Zadatak: Treba ispisati vreme iz računara memorije na ekran.

Program je:

```
;*****************************************************************
Prog06      Segment                         ;definicija segmenta
            assume CS:Prog06,DS:Prog06      ;pocetak CS i DS segmenta
;*****************************************************************
Start:mov   ax,Prog06               ;pocetak na DS
      mov   ds,ax
;*****************************************************************
      mov   ax,0b800h           ;adresa video memorije u es registar
      mov   es,ax
;*****************************************************************
cikl: mov   ax,0200h    ;citanje vremena iz sistemskog casovnika
      int   1ah
;*****************************************************************
      call  vrisp               ;ispisivanje vremena
;*****************************************************************
      mov   ax,0100h            ;taster?
      int   16h
;*****************************************************************
      jz    cikl                ;ne, nazad
;*****************************************************************
      mov   ax,0                ;da, iscitati vrednost
      int   16h
;*****************************************************************
      mov   ax,4c00h                        ;nazad u DOS
      int   21h
;*****************************************************************
;#################################################################
vrisp Proc                      ;potrogram za  ispisivanje vremena
;*****************************************************************
      mov   di,0                ;monitor, levo gore
      mov   ah,15               ;crna pozadina, beli karakter
;*****************************************************************
      mov   bx,cx               ;bx <== sati, minuti
      call  isp1                ;sati na ekran
;*****************************************************************
      inc   di                  ;mesto za separatora
      inc   di
;*****************************************************************
      call  isp1                ;minuti na ekran
      inc   di                  ;mesto za separatora
      inc   di
;*****************************************************************
      mov   bh,dh               ;bh <== sekunde
      call  isp1                ;sekunde na ekran
;*****************************************************************
      mov   al,"-"              ;separator
      mov   es:[di-6],ax        ;na odgovarajuce pozicije
```

```
      mov   es:[di-12],ax
;*************************************************************
      ret                     ;povratak u glavni program
vrisp Endp                    ;kraj potprogram
;#############################################################
;*************************************************************
;#############################################################
isp1  Proc                    ;potprogram
;*************************************************************
      mov   cx,2              ;ispisivanje 2 cifre
cikl1:push  cx
      mov   cx,4              ;BCD broj (4 bit)
      mov   al,0
;*************************************************************
cikl2:shl   bx,1              ;Znacajniji deo bx registara (BCD broj)
      rcl   al,1              ;
      loop  cikl2
;*************************************************************
      or    al,30h            ;bin ==> ASCII
      mov   es:[di],ax        ;u video memoriju
      inc   di                ;sledeca pozicija
      inc   di
      pop   cx                ;druga cifra
      loop  cikl1
;*************************************************************
      ret                     ;nazad u program odakle je je pozvan
ir1   Endp                    ;kraj potprograma
;#############################################################
;*************************************************************
Prog06      Ends              ;kraj sgmenta
      End   Start
;*************************************************************
```

Program koristi ugrađeni sat **IBM PC** računara, sa potprogramom **int 1ah** (parametar **0200h**) isčitamo vreme iz **CMOS (SETUP)** memorije. Brojevi su prikazani u **BCD** brojnom sistemu, ovako samo treba dodati vrednost **30h**, za pretvaranje **BCD** broja u **ASCII** kod, u kom obliku već ih možemo direktno koristiti. Ovu operaciju možemo realizovati i pomoću **ILI** logičke operacice, **or al,30h**.

7. Zadatak: Treba napisati program, koji posle brisanja ekrana ispiše jedan meni, sledećem sadržajem:

- **PRVI**
- **DRUGI**
- **TREĆI**
- **ČETVRTI**
- **PETI**

Nakon toga u pozadini prvog reda treba da se pojavi zelena traka (pozadina teksta). Zelenu traku sa strelicama dole/gore proizvoljno je moguće namestiti na izabrani tekst. Sa pritiskom tastera **ENTER** treba da se pojavi tekst sa crvenom pozadinom:

- **IZABRAN xxxxxxx MENI** (xxxxxxx = odgovarajući tekst iz menija) i nakon toga tekst
- **NOVI IZBOR ? (d/n)**.

Sa odgovorom **d** (da) ponovo treba da se pojavi meni, a sa odgovorom **n** (ne) napustićemo program. Ni jedan drugi karakter nema nikakav efekat.

Listing programa je:

```
;*********************************************************************
Prog07      Segment                         ;definicija segmenta
            assume CS:Prog07,DS:Prog07      ;pocetak CS i DS segmenta
;*********************************************************************
Start:mov   ax,Prog07                       ;pocetak na DS
      mov   ds,ax
;*********************************************************************
      mov   ax,0b800h                       ;adresa video memorije u
      mov   es,ax                           ;es registar
;*********************************************************************
      mov   ax,3                            ;brisanje ekrana
      int   10h
;*********************************************************************
      call  write1                  ;meni na ekran
lab1: call  line              ;zelena traka u pozadini prvog reda
;*********************************************************************
lab2: mov   ax,0100h                ;taster?
      int   16h
;*********************************************************************
      jz    lab2                    ;ne, cekati
;*********************************************************************
      mov   ax,0                    ;da
      int   16h
      cmp   ah,1ch                  ;ENTER?
      jz    ent                     ;da, nastaviti tamo
;*********************************************************************
      cmp   ah,48h                  ;^? (gore?)
      jnz   lab3                    ;ne, proveriti sta je
;*********************************************************************
      cmp   byte ptr [POINTER],1    ;POINTER = 1 ?,
      jz    lab2
;*********************************************************************
      dec   byte ptr [POINTER]      ;POINTER - 1
      jmp   lab1                    ;cekati opet na taster
;*********************************************************************
lab3: cmp   ah,50h                  ;dole ?
      jnz   lab2                    ;ne, taster
;*********************************************************************
      cmp   byte ptr [POINTER],5    ;POINTER jos ne pokazuje na
      jz    lab2              ;5. poziciju, inkrementirati
;*********************************************************************
      inc   byte ptr [POINTER]
      jmp   lab1                    ;taster
;*********************************************************************
ent:  mov   cl,byte ptr [POINTER]   ;u cl vrednost POINTER-a
      mov   si,offset ADDRESS       ;u bs adresa
;*********************************************************************
ent1: mov   bx,[si]
      add   si,2
      loop  ent1
      jmp   bx                      ;skok na osnovu bx registara
;*********************************************************************
ent2: mov   si,offset QUEST         ;pitanje na ekran
      call  write2
;*********************************************************************
ent3: mov   ax,0100h                ;da? ne?
      int   16h
      jz    ent3
```

```
;*********************************************************************
      mov   ax,0
      int   16h
;*********************************************************************
      cmp   ah,31h                  ;ne, izlaziti iz programa
      jz    ent4
;*********************************************************************
      cmp   ah,17h                  ;neki drugi taster, dalje
testirati
      jnz   ent3
;*********************************************************************
      jmp   start                   ;da, na pocetak
;*********************************************************************
ent4: mov   ax,4c00h                ;nazad u DOS
      int   21h
;*********************************************************************
pr1:  mov   si,offset TXT1          ;ispisivanje teksta
      call  write2
      jmp   ent2
;*********************************************************************
pr2:  mov   si,offset TXT2          ;ispisivanje teksta
      call  write2
      jmp   ent2
;*********************************************************************
pr3:  mov   si,offset TXT3          ;ispisivanje teksta
      call  write2
      jmp   ent2
;*********************************************************************
pr4:  mov   si,offset TXT4          ;ispisivanje teksta
      call  write2
      jmp   ent2
;*********************************************************************
pr5:  mov   si,offset TXT5          ;ispisivanje teksta
      call  write2
      jmp   ent2
;*********************************************************************
;#####################################################################
line  Proc                          ;potprogram trake
      mov   al,byte ptr [POINT2]  ;prethodna vrednost pokazivaca u al
      mov   bl,160                  ;duzina u bl
      mov   ah,0
;*********************************************************************
      mul   bl                      ;pocetna adresa pokazivaca
      add   ax,1499
      mov   di,ax
      mov   cx,21
      mov   al,7                    ;crna pozadina, beli karakter
;*********************************************************************
line1:mov   es:[di],al              ;boja
      add   di,2
      loop  line1
      mov   al,byte ptr [POINTER]   ;kao prethodno, ali nova pozicija
      mov   byte ptr [POINT2],al    ;pojacana pozadina
      mov   bl,160
      mov   ah,0
      mul   bl
      add   ax,1499
      mov   di,ax
      mov   cx,21
      mov   al,47
;*********************************************************************
```

```
line2:mov   es:[di],al
      add   di,2
      loop  line2
      ret                           ;nazad
line  Endp                          ;kraj potprograma
;#########################################################################
;*************************************************************************
;#########################################################################
write1Proc                          ;potprogram za ispisivanje teksta
      mov   si,offset CHOICE        ;0 ==> di+160 (novi red)
      mov   ah,7                    ;255 ==> kraj teksta I zlaz
      mov   di,1660
;*************************************************************************
wr1:  push  di
wr2:  mov   al,[si]
      inc   si
      cmp   al,0
      jz    wr3
      cmp   al,0ffh
      jz    wr4
      mov   es:[di],ax
      add   di,2
      jmp   wr2
;*************************************************************************
wr3:  pop   di
      add   di,160
      jmp   wr1
wr4:  pop   di
;*************************************************************************
      ret                           ;alprogram vege
write1Endp                          ;alprogram vege
;#########################################################################
;*************************************************************************
;#########################################################################
write2Proc                          ;ispisivanje teksta
      mov   al,[si]                 ;izracunavanje koordinate
      mov   ah,0
      shl   ax,1
      mov   di,ax
      mov   al,[si+1]
      mov   bl,160
      mul   bl
      add   di,ax
      add   si,2
      mov   ah,12
;*************************************************************************
wr21: mov   al,[si]
      inc   si
      cmp   al,0
      jz    wr22
      mov   es:[di],ax
      add   di,2
      jmp   wr21
;*************************************************************************
wr22: ret
write2Endp
;#########################################################################
;*************************************************************************
POINTER:db  1
POINT2:db   1
;*************************************************************************
```

```
CHOICE:db   "PRVI",0
       db   "DRUGI",0
       db   "TRECI",0
       db   "CETVRTI",0
       db   "PETI",255
;*********************************************************************
TXT1: db    22,5,"IZABRAN PRVI MENI",0
TXT2: db    21,5,"IZABRAN DRUGI MENI ",0
TXT3: db    21,5,"IZABRAN TRECI MENI ",0
TXT4: db    21,5,"IZABRAN CETVRTI MENI ",0
TXT5: db    21,5,"ZABRAN PETI MENI",0
;*********************************************************************
QUEST:db    28,20,"NOVI IZBOR ? (d/n)",0
;*********************************************************************
ADDRESS:dw  offset pr1
       dw   offset pr2
       dw   offset pr3
       dw   offset pr4
       dw   offset pr5
;*********************************************************************
Prog07Ends                          ;kraj segmenta
       End   Start
;*********************************************************************
```

VEŠTAČKA INTELIGENCIJA I EKSPERTNI SISTEMI

VEŠTAČKA INTELIGENCIJA

Delatnost čoveka se ispoljava u vidu obavljanja fizičkog i umnog rada tj. ostvarivanju transformacije energije i informacije. Fizički rad se pretežno ispoljava u vidu transformacije energije i za čoveka predstavlja zamornu, tegobnu i dosadnu aktivnost. Procesi realizacije fizičkog rada su relativno jednostavni. Ostvaruju se u skladu sa opštim zakonima prirode. Efekti ovog rada su merljivi. Razvoj tehnike i automatizacije proizvodnje realizacijom odgovarajućih mehanizama i automata dovela je do bitnog olakšavanja na području obavljanja fizičkog rada.

Umni rad čoveka se manifestuje u vidu dobijanja, obrade i kreiranja informacije. Procesi transformacije informacija u ljudskom mozgu se odvijaju prema za sad malo poznatim zakonitostima. Čoveku je poznat polazni i konačni set informacija a ne i postupak transformacija informacija. U okviru nauke o obradi informacija sve intenzivnije se istražuju algoritmi i metodologije obrade informacija u ljudskom mozgu.

Otkrivene zakonitosti se svrstavaju u novu naučnu oblast informatike tkzv. "nauku o veštačkoj inteligenciji. Veštačka inteligencija teži ka rešavanju primenom računara takvih problema koje traže metodologiju umnog rada tj. inteligenciju. Zadaci, koji spadaju u ovu oblast, obično ne sadrže u potpunosti razjašnjenje mehanizama za njihovo rešavanje, već dolazi do izražaja pokušaj, a njega usmerava čovekovo stručno znanje i intuicija.

Veštačka inteligencija je grana računarske tehnike, koja se bavi razvijanjem inteligentnih računarskih sistema. Ovi sistemi su hardverski/softverski sistemi, koji su sposobni na čovečji način rešiti komplikovane zadatke:
uče iz prethodnih iskustava, samostalno rešavaju probleme, i/ili međuvremeno komuniciraju sa okruženjem. Tumače saopštenja koja su data na prirodnim jezicima, ili su grafički prikazani. Računarski sistemi, koji su sposobni da funkcionišu slično kao čovek stručnjak, su ekspertni sistemi U ovu grupu sistema svrstavaju se danas svi sistemi koji uz bilo koju hardversku konfiguraciju rešavaju probleme metodologijama koje je dala naučna oblast veštačke inteligencije.

Današnji najrazvijeniji ekspertni sistemi koji rešavaju probleme uskog srtučnog znanja još nisu tako savršeni da na bilo koji način budu konkurentni čoveku. Popularna štampa i razne softverske kuće već naširoko koriste termin "ekspertni sistemi" za svoje softverske proizvode, često neopravdano, pošto umesto ovog termina ne postoji adekvetniji izraz.

RAZVOJ VEŠTAČKE INTELIGENCIJE

Inteligencijom čoveka bavili su se već u starom veku. Grci, koji su ispitivali tajanstvenosti čovečjeg razmišljanja, već pre 3000 godina, stvorili su sisteme logike, sintakse i teorije znanja, koje oblasti su i danas značajne.

Krajem drugog svetskog rata, engleski i američki istraživači su realizovali računar. Naučnik Neumann je odredio princip rada i arhitekturu računara. Njegov model (Slika 21) je pretrpeo više modifikacija ali su poštovani sledeći principi:

- računar je uređaj opšte namene i poseduje potpuno automatsko izvođenje programa,
- računar, osim podataka za računanje (ulazne vrednosti, granične vrednosti, tabele, funkcija), memoriše međurezultate i rezultate računanja,
- računar je sposoban za pohranjivanje toka naredbi (programa),
- naredbe su u računaru svedene na numerički kod, tako da se podaci i naredbe pohranjuju na isti način u istoj jedinici, ta jedinica se naziva memorijom,
- računar poseduje jedinicu koja obavlja osnovne aritmetičke operacije, to je aritmetičko-logička jedinica,
- ima jedinicu koja "razume" naredbe i upravlja tokom izvršavanja programa, to je upravljačka jedinica i
- računar komunicira sa okolinom, jedinica, koja omogućava komunikaciju čoveka i računara, naziva se ulazno-izlaznom jedinicom.

Slika 21: Neumann-ov model računara

Još pre pojave računara postojala su dva pravca u nauci koji su se bavili osnovama veštačke inteligencije:

- formalna logika - koja se bavi osnovama simboličke obrade zasnovane na matematičkoj logici i teoriji rekurziviteta i
- kognitivna psihologija koja se bavi psihologijom razmišljanja i prepoznavanja i istražuje ponašanje i sposobnost čoveka u rešavanju problema.

Objedinjavanjem ova dva pravca nauke uz sprezanje sa informatikom započeta su savremena istraživanja na području veštačke inteligencije. Ova doba može se staviti na sredinu 1950-tih godina. Da bi se postigao zajednički cilj, razvijen je jezik LISP. Njegov autor je bio John McCarthy. Po mnogima, on je prvi upotrebio izraz "Veštačka Inteligencija" "Artifical Intelligence".

Razvoj ove oblasti tekao je prema sledećem:

- period od 1955 do 1960, karakteristično je za početak istraživanja VI, prva konferencija je organizovana u Dartmouth-u. Izradjeni su sistemi PERCEPTRON i GPS.
- između 1961-70, pojavili su se prvi osnovni jezici VI LISP i PROLOG. Arhitektura ovih višeprogramskih jezika se bazira na matematičkoj logici, a rešavaju probleme nealgoritamski manipulacijom simbolima. Prvi sistemi su bili izgradjeni u definisanim oblastima, sa konačnim brojem pravila. Medjutim, veliki i složeni sistemi nisu se mogli dobro

simulirati ni sa složenim strategijama traženja. Programi se nisu mogli približiti intuiciji stručnjaka. Složene sisteme je bilo lakše savladati, ako se prezentuje sam zadatak, i dobro struktuirano se prikaže znanje o koracima za njegovo rešavanje. Aktiviranje potrebnog dela programa vrši se jednostavnim mehanizmom zavisnim od problema.

- u periodu od 1971 do 1980, pojavljuju se sistemi na bazi znanja ("knowledge based") koji zahtevaju drugačiji pristup organizacije programa. Odvaja se deo sa bazom znanja stručnjaka odredjenog domena, od mehanizma koji to aktivira u zavisnosti od ukazane situacije. Većina tih sistema pruža prijatan rad i komunikaciju sa korisnicima, koji nisu specijalisti računarske tehnike. Bilo ko može pratiti rad sistema, a na zahtev korisnika sistem daje obrazloženje za izvršena pravila, i to na jeziku koji
- je bliže čoveku. Ove komponente sistema zajedno sa delom za popunjavanje baze znanja i za njeno održavanje dobro služe kao okvirni sistemi, kao prazne školjke (shell). Poznati sistemi za osnovu prezentaciju znanja su MYCIN, HERSAY II, a za osnovu tehnologije znanja su EMYCIN, GUIDON.
- u periodu od 1981 do 1990, pojavljuju se na tržištu i primenjuju se hardveri za VI, i softveri za obradu govornog jezika, prepoznavanje i generisanje glasa, mašinsko vidjenje, mašinsko učenje i neuralne mreže.
- od 1991, javljaju se računarski sistemi 5. i 6. generacije. U Japanu su radili na 5. generaciji, na principu logičke obrade (PROLOG). 6-ta generacija računarskih sistema se gradi na principu neuralne mreže. One su dinamički sistemi sa velikom paralelnošću, sa kojima je moguće u većoj meri se približiti čovekovu inteligentnu obradu informacija. Računari 6-te generacije građeni su na bazi pozitivnih dostignuća svih prethodnih generacija računara.

Od 1989 tehnologija zasnovana na principu baze znanja razvija se u dva pravca:

- tehnologija na bazi znanja čvršće se integriše sa
- tradicionalnim tehnikama, kao što su baze podataka, i sa novijim tehnikama, kao što je hipertekst i razne tehnike VI,
- umesto razvoja sredstva za opštu namenu, razvijaju se i uvode sredstva koja su orijentisana prema primeni, tj. prema tipu problema.

POJAM VEŠTAČKE INTELIGENCIJE

Pojam veštačke inteligencije uveo je John McCarthy 1956 godine. Tako je nazvao svoja istraživanja. Iako naziv nije najpogodniji on je ipak i dan danas u upotrebi. Ova nauka pod inteligencijom podrazumeva one principe inteligentnog ponašanja, u koje spada sposobnost komunikacije, razvoja, prilagođavanja, osetljivost prema problemima, sposobnost rešavanja problema, ciljno orijentisana delatnost, samostalni izbor cilja. Ovi pojmovi nam ukazuju, da se tu ne radi o takvoj inteligenciji, o kojoj se govori u slučaju čoveka ili životinje. U slučaju životinje podrazumeva se još i stvaranje unutrašnje slike o svetu, manipulacija sa tim simbolima koji stvaraju tu sliku, preko njih se manifestuje težnja da se promeni "svet", sposobnost za prilagodjavanje i učenje, koja pretpostavlja predstavu prošlosti i budućnosti, komunikaciju putem simbola, i podrazumeva se njihovo zajedničko prisustvo, i dodaje se još i "namera", kao i pojam

"osećaj", a kod čoveka još i samorefleksija, sposobnost za raspoznavanje i prevazilaženje sopstvenih ograničenja.

Računarska tehnika u nekim oblastima prevazilazi efikasnost čovekovog uma, u brzini, tačnosti a pruža i druge prednosti. Računar, na ovim područjima može dati iznenadjujuće i uspešne rezultate. Najrasprostranjena definicija VI je ona koju je dao Marvin Minsky:
"veštačka inteligencija je ona nauka, koja osposobi računar za obavljanje takvih radnji, koje se obeleže kao inteligentne, kada ih čovek izvršava".

PODRUČJA ISTRAŽIVANJA VEŠTAČKE INTELIGENCIJE

Oblasti istraživanja VI su sledeće:

- dokaz teoreme, ova oblast ispituje na koji način se može dokazati jedna hipoteza uz dato znanje pomoću dokaznih metoda i pravila. Ponekad je potrebno razbiti hipotezu na pomoćne hipoteze, gde izbor delova teoreme može biti zadatak koji zahteva veliko iskustvo. Najteže je čovekovo iskustvo pretvoriti u znanje koje može računar da koristi.
- prepoznavanje uzorka. Kada su objekti, situacije ili događaji opisani izvesnim numeričkim promenljivama, koje promenljive liče na određene uzorke i iz brojčano izražene daljine zaključuju npr. istovetan, blizu istovetan, verovatno istovetan, neistovetan itd.,
- heuristički postupci. Ovi postupci najčeše savetuju opšte, celishodne, fleksibilne, u praksi opravdane metode rešenja nasuprot onim rešenjima, koji su egzaktni, teorijski osnovani, i koji obećaju optimalna rešenja, ali samo uz strogo definisane uslove. Heuristička rešenja igraju važnu ulogu u ekspertnim sistemima, jer egzaktna rešenja nisu uvek dostupna, ili ne bi funkcionisale u slučaju da fali jedan ili više podataka.
- postupci traženja. Najčeše opisuju razne mogućnosti kako obilaziti grafom reprezentovani domen problema. Svaka heuristika, za ovaj problem, osigurava rešenje koje je blizu optimalnom rešenju.
- prezentacija znanja. Najizraženiji pravac VI. Činjenica je da čovek pamti svoja znanja na sasvim drugačiji način od računara. Pitanje je na koji način prikazati razno stručno znanje koja bi računar uspešno obradio i prikazao na nekom programskom jeziku.
- obrada simbola. Pored pisanog materijala obično treba raditi i sa zvučnim signalima kao i sa slikama. Kod zvučnog signala treba prepoznati signale, zatim izvršiti transformaciju i izvršiti dekodiranje istih. Prepoznavanje jedne slike, počinje na osnovnom nivou sa identifikacijom linija, kriva, tačke, i uglova. Postupno, po odredjenoj hierarhiji se sastavlja slika. Prepoznavanje dinamične slike sastoji se iz više nivoa identifikacije, i tako se formira celina.
- kombinatorni i taktički zadaci. Neki zadaci su suviše dugotrajni (ako bi isprobali svaku kombinaciju), čak i na najbržem računaru, zato se koriste razne metode za skraćivanje, razne heuristike. Složenost zadatka, koji spada u ovu grupu, eksponencijalno zavisi od vrednosti nekih parametara.
- manipulacija neizvesnostima. Postupci se koriste u rešenjima gde su informacije nepotpune, nesigurne ili pune šumova. Postoji više izvora nesigurnosti, prema tome i njihovo rukovanje, procena uticaja je

raznovrsna u zavisnosti od osobine i drugih okolnosti problema. Neke metode rešenja su bazirana na matematičkim modelima npr. fuzzy-logika, ili drugi modeli.

PODRUČJA PRIMENE VEŠTAČKE INTELIGENCIJE

VI, koja se razvija relativno brzo, već naširoko se koristi u sledćim aplikacijama:

- inteligentno rukovanje bazom podataka. U ovim sistemima korisniku je omogućen pristup ne samo onim podacima, do kojih se može doći direktno, nego i onim izvedenim podacima do kojih se dolazi nakon analize direktno dobijenih podataka. To je moguće sa korisničkim modelima, koji opisuju razne potrebe korisnika, očekivanja i mogućnosti.
- robotika. Nauka i tehnologija za razvoj i upotrebu "inteligentnih" sredstava. Obavlja se fizički, konstruktivni posao sa izvesnim stepenom prilagodjavanja. Značajna je sposobnost identifikacija pozicije objekta i okolnosti da bi se izabrala odgovarajuća metoda za postizanje unapred datog cilja. Iz osnovnih radnji sastavlja se plan aktivnosti, koji su potrebni za prelaz iz jednog stanja robota u drugo. Ima veliki uticaj na dalji razvoj industrije.
- obrada prirodnih i veštačkih jezika. Vrši se prevod, ili priprema za prevod sa jednog jezika na drugi.
- inteligentna komunikacija sa računarom. Tokom rada računara čovek očekuje jasnu komunikaciju u vezi zadatka.
- razumevanje i generisanje živog jezika. Sa skromnom količinom reći i jednostavnom gramatikom je ovo moguće na nekim moćnijim računarima. Računar, sem značenja pojedinačnih reči, mora da poznaje i relacije izmedju njih, gramatiku jezika i pojmove
- pridružene objektima, kao i relacije medju rečenicama. Iz toga proizilaze tri funkcije jezičke analize:
    - leksička analiza (analiza reči),
    - sintaktička analiza (analiza mesta reči u rečenici) i
    - semantička (analiza značenja rečenice u kontekstu i kao zasebne celine)
- automatsko programiranje. Programski zadatak, koji je napisan na nekom specifičnom jeziku (npr. predikatum-kalkulus), prevede na neki programski jezik, ili na izvršni kod. Program, koji to izvršava mora poznavati sintaksu izvornog (na kome je data specifikacija) i izvršnog jezika, kao i moguće strukture podataka, konvencije za rukovanje input-outputom i ekranom .
- mašinsko učenje. Razvijeni su neki značajni programi učenja na osnovu saveta i kazivanja, na osnovu primera posredstvom induktivnog zaključivanja - i na osnovu otkrića.
- sistemi zasnovani na znanju. Tu spadaju ekspertni sistemi, koji rešavaju zadatke simboličke prirode, sposobni su da obrazlažu svoja ponašanja i zaključke, sposobni su da se usavršavaju, da rešavaju značajne, teške i složene probleme koji uključuju i neizvesnost.

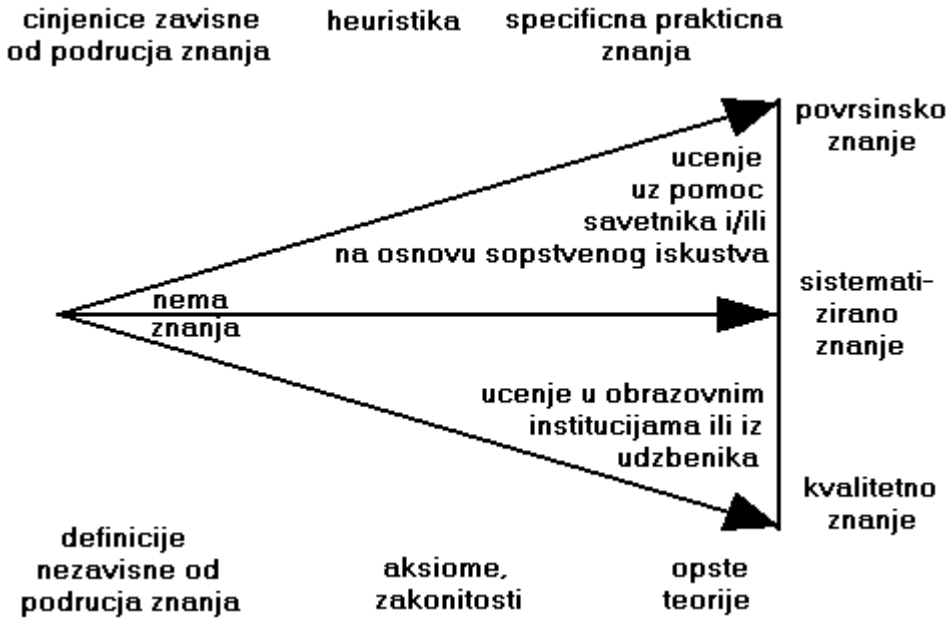
EKSPERTNI SISTEMI

Izraz, ekspertni sistemi potiče od činjenice, da su takvi sistemi nastali tako što su od priznatih eksperata u nekoj oblasti, intervjuisanjem i drugim postupcima doznavana i "zahvatana", a potom organizovana njihova znanja. Izraz ekspertni sistemi (u daljem tekstu: ES), se najčešće primenjuje na programe koji se koriste znanjima radi

simuliranja ponašanja čoveka-eksperta, zapravo čije funkcionisanje ima neke oblike ponašanja čoveka eksperta. Takvom čoveku je svojstveno da poseduje veliku količinu specijalizovanih znanja i informacija uključujući i neke retke činjenice - potrebnih za rešavanje specifičnih problema u odredjenoj oblasti, ali da sem toga, poseduje neka, pretežno lična, pravila domišljatog rezonovanja i nagadjanja, što mu omogućava efikasno analiziranje i uspešno i brzo rešavanje tih problema. On ima moć učenja iz iskustva, sticanja opštih znanja, rekonceptualizacije, rezonovanja po analogiji, transfera znanja iz jednog područja u drugo, ima fleksibilnost u menjanju pristupa da bi se pronašao onaj koji je prikladan za rešavanje datog problema. Osim toga, ES omogućava dijagnosticiranje problema, zna da preporučuje alternative i rešenja, obrazlaže i racionalizuje svoje dijagnoze i preporuke, kao i da "uči" iz prethodnih iskustava dodajući bazi znanja nove elemente do kojih dolazi tokom rešavanja problema.

KOMPARATIVNA ANALIZA LJUDSKOG I MAŠINSKOG ZNANJA

Procesi stvaranja i memorisanja ljudskog znanja su veoma složeni i praktično neotkriveni. Kategorizacija se može obaviti samo sa stanovišta spoljnih efekata a ne i sa stanovišta internih mehanizama. Na slici 2 data je jedna kategorizacija ljudskog znanja uz način sticanja istih kojoj se može upotrebiti za komparativnu analizu ljudskog znanja i mašinskog znanja ugrađenih u razne tipove ekspertnih sistema.



Slika 2: Način sticanja i kategorizacije ljudskog znanja

Čovek stiče znanje ili na bazi sopstvenog ili tuđeg iskustva posmatranjem, zaključivanjem tj. delovanjem u praksi. Ovaj vid znanja može biti često veoma efikasan za rešavanje nekih praktičnih problema ali ima ograničen domen efikasnosti. Znatno efikasnije se stiče znanje uz sistematski mehanizam usvajanja znanja učenjem u obrazovnim ustanovama ili iz udžbenika. Bez obzira na nedostatke obrazovnih procesa za sada se ovaj vid sticanja znanja pokazao efikasnijim i delotvornijim. U Tabeli 1. date su osnovne karakteristike ljudskog znanja i znanja ekspertnih sistema prve generacije.

Tabela 1: Osnovne karakteristike ljudskog znanja i znanja ekspertnih sistema prve generacije	
stručna znanja čoveka	stručna znanja zasnovana na primeni veštačke inteligencije
prolazno	stalno
teško se prenosi, reprodukuje, širi se obučavanjem	lako se prenosi i reprodukuje
teško se dokumentuje	lako se dokumentuje
nije uvek dosledno, često je čak i labilno, podložno je emocijama	uvek je dosledno
skupo a često postaje i nedostižno	može se pribaviti "jeftino" i uvek stoji na raspolaganju
kreativno, inovativno	nekreativno, bez duše
adaptivno, obučava se	zna samo ono što su ugradili u bazu znanja
interaktivno je sa okolinom preko čula čoveka	komunikacija sa okolinom je ograničena na interaktivni sistem
procesi mišljenja su promenljivi, bogati	operiše unutrašnjim simbolima
problem razmatra sa više aspekata, proces mišljenja je dinamičan i adaptivan, operiše višenivoskim modelima	problem razmatra sa ograničenjima tj. samo sa pozicija ugrađenih aspekata
bazira se na zdravom razumu	ima samo tehničko znanje, treba očekivati samo predlog, savet a ne i rešenje problema

Pri formiranju ekspertnih sistema čovek je težio da u sistemu upravljanja znanja i donošenju odluka oponaša čoveka. Ekspertni sistemi prve generacije koriste nestandardne metode gradnje znanja. U toku rada ne proširuju bazu znanja na osnovu sopstvenih iskustava tj. ne uče iz sopstvenog iskustva tj. imaju samo površinske efekte eksperta.

Ekspertni sistemi druge generacije teže ka primeni metodologije mašinskog učenja tj. ka proširivanju baze znanja u hodu. Deluju u pravcu integracije površinskog (praktičnog) i kvalitetnog znanja.

TIPOVI I PODRUČJA PRIMENE EKSPERTNIH SISTEMA

Do današnjeg dana je razvijen veći broj ekspernih sistema koji uspešno mogu rešavati neke probleme. Osnovna područja uspešne primene su sledeća:

- sistemi za objašnjavanje, (interpretaciju) - ova kategorija sadrži nadzor, razumevanje jezika, analizu lika, tumačenje hemijske strukture, tumačenje signala, i više tipova inteligentne analize, ovi tipovi sistema, objašnjavaju podatke, na taj način, što im daju simboličko značenje za opis situacije ili stanja sistema,
- prognostički sistemi - sadrže prognozu vremena, prognozu demografskih kretanja, prognozu u prometu, procenu žetve i vojna predvidjanja, prognostički sistemi su tipično dinamični sistemi sa vrednostima parametara, koji odgovaraju datoj situaciji, zaključci, koji proizilaze iz modela čine osnovu za predvidjanja, ignorisanjem

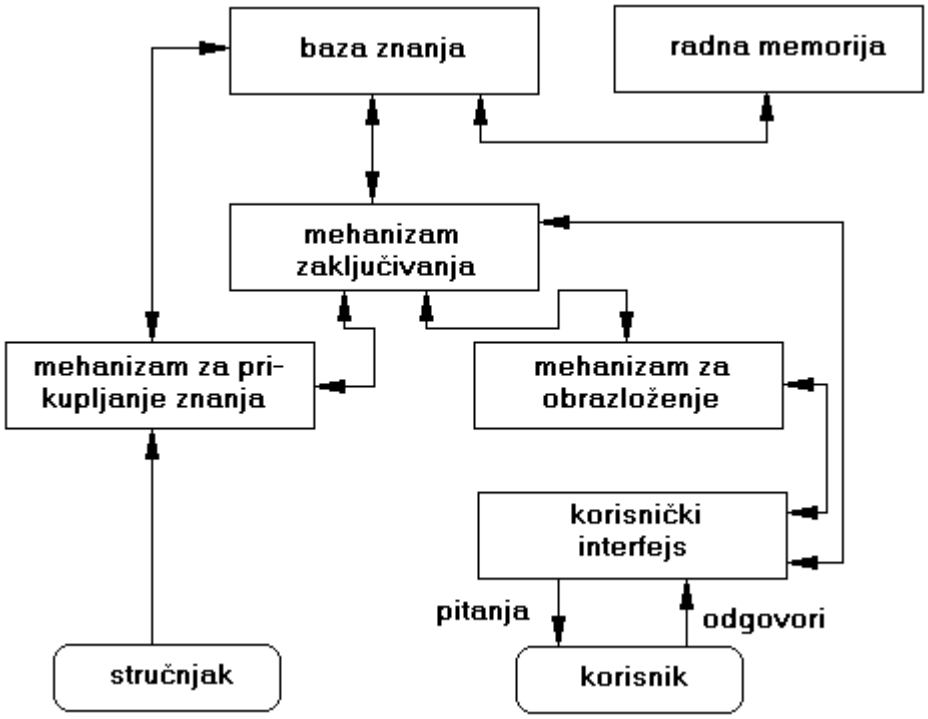
verovatne procene, prognostički sistemi mogu generisati veliki broj mogućih scenarija,

- dijagnostički sistemi - ova kategorija, izmedju ostalog sadrži dijagnozu iz medicine, elektronike, mehanike i softversku dijagnozu, tipično, ovi sistemi, posmatrano nepravilno ponašanje vezuju sa istaknutim uzrokom, koristeći jednu ili više tehnika, kod nekih metoda, ključna je tabela veza izmedju ponašanja i dijagnoze, drugi metodi kombinuju znanje o dizajnu sistema sa znanjem o potencijalnim greškama dizajnu, implementaciju ili komponente za generisanje moguće pogrešne funkcije dosledno opažaju,
- sistemi planiranja - sadrže automatsko programiranje, kao i planiranje kretanja robota, odvijanja projekta, puta, komunikacije, eksperimente, i rešavanje problema vojnih planiranja itd., sistemi za planiranje koriste modele o ponašanju okruženja, da bi mogli doneti zaključke efektima ponašanja okruženja,
- sistemi monitoringa - upoređuje se posmatranje o ponašanju sistema sa osobinom za koji se smatra da je kritičan za uspešan rezultat plana, ove kritične osobine, ili povredljive situacije, mogu prouzrokovati pogrešan tok plana, uopšte, sistemi monitoringa, identifikuju povredljive osobine na dva načina:
    - prema jednom načinu, tip kritične situacije odgovara jednom prihvatljivom uslovu, čija povreda bi uništila racionalnost plana,
    - druga vrsta kritične situacije je kada neki potencijalni efekti plana povredjuju planiranu strukturu, oni odgovaraju pogrešnim funkcijama u predvidjenim situacijama, postoji više sistema monitoringa sa računarskom podrškom, za planiranje nuklearne energije, vazdušnog prometa, bolesti, regulacije, i druge,
- debaging sistem - ovi sistemi, oslanjaju se na planiranje, dizajniranje, i predvidjanje mogućnosti za kreiranje specifikacije ili savete za korekciju utvrđenog problema, debaging sistem sa računarskom podrškom, postoji za računarsko programiranje, čak i u formi inteligene baze znanja i tekst editora, ali se ne klasifikuju kao ekspertni sistemi,
- sistemi obnavljanja, ovi sistemi obuhvataju sisteme debaginga, sisteme planiranja i izvršne sposobnost,. računarom podržavani sistemi ove vrste, nalaze se u područjima:
- distributivnih mreža, avijacije i u održavanju računara, kao i u drugima,
- sistemi instrukcije - ovi sistemi sadrže dijagnostičke i debaging podsisteme koji daju instrukcije istraživaču, učeniku, ili bilo kom korisniku, ovi sistemi tipično kreću sa konstrukcijom hipotetičkog opisa korisnikovog znanja, koji objašnjava korisnikovu reakciju, kasnije se ispita slabost korisničkovog znanja i identifikuju se odgovarajuće korekcije, najzad se planira dvosmerni tutorijal, koji je namenjen za povećanje znanje
- korisnika,
- kontrolni sistemi - ovi sistemi upravljaju celo ponašanje sistema, da bi to umeli, kontrolni sistemi moraju više puta tumačiti postojeće stanje, predvideti budućnost, ispitivati uzroke predvidjenih problema, formulisati korektivne planove, i nadzirati njihovo izvršenje da bi se postigao uspeh, problemi, koji su snabdeveni kontrolnim sistemom, su kontrola vazdušnog prometa, menadžment poslovanja, menadžment

bitke itd., tehnologija inženjeringa znanja obrađuje više kontrolnih problema, koji se teško mogu rešiti klasičnim matematičkim pristupom na tradicionalni način.

ARHITEKTURA EKSPERTNIH SISTEMA

Uspeh ekspertnih sistema može se tražiti u činjenici da se umesto napornog i sistematskog pretraživanja po bazi znanja uvodi heuristika tj. optimalizirajući mehanizmi stečeni na bazi čovekovih empirijskih iskustava. Baza znanja je odvojena od mehanizama za zaključivanje i izvršenja. Svaki ES poseduje odredjene karakteristike i specifičnosti koje kod drugih ES nema. Ipak, u suštini svi imaju istu konstrukciju, konfiguraciju i organizaciju sastavljenu od baze znanja, mehanizma zaključivanja, podsistema za obrazlaganje ponašanja i zaključaka, podsistema za uzimanje i ažuriranje znanja i korisničkog interfejsa. Slika 22. prikazuje arhitekturu ekspertnog sistema.



Slika 22.: Arhitektura ekspertnog sistema

Baza znanja: Bazu znanja čini osnovni skup znanja tj. bazične činjenice, pravila i heuristika. Pravilo je uslovni iskaz koji odredjuje akciju u slučaju zadovoljenja nekog posebnog uslova. Pravila se uopšteno baziraju na stručnom znanju za rešavanje problema, koji su razvili eksperti, radeći više godina na odredjenoj problematici. Postoji i znanje sistema o samoj sebi, tj. izvedeno znanje iz datih pravila, ili metaznanje. Znanje pohranjeno u bazi čini jedan ES sposobnim da se ponaša kao ekspert.

Mehanizam zaključivanja i kontrole. Ovaj deo simulira deduktivne misaone procese čoveka stručnjaka. Polazeći od baze znanja i korisnikovog ulaza u ES, procedura zaključivanja je metoda razmišljanja uz pomoć koje se prepoznaju i uparuju uzorci

znanja s podacima. Ugrađene strategije zavise od metoda koje su se koristile za prezentaciju znanja. Tu spadaju unapred olančavanje, unazad olančavanje, kao unutrašnji nadzor sistema. Ovaj mehanizam uključuje i strategije zaključivanja i kontrole za manipulisanje znanjem radi dolaženja do novih informacija i znanja.

Podsistem za uzimanje i ažuriranje znanja. Baza znanja treba da sadži sva potrebna znanja iz posmatranog domena u svakom trenutku vremena. Da bi se proširenja i izmene kojem znanje u jednoj oblasti podleže mogli pratiti i pohranjivati u bazu znanja, potrebno je omogućiti njegovo manuelno ili automatsko ažuriranje. Ovu aktivnost izvršava ovaj podsistem. Manuelno ažuriranje vrši inženjer znanja ili domen-ekspert, a automatsko ažuriranje podrazumeva mašinsko učenje, odnosno generalizaciju iskustva u formi novog znanja.

Podsistem obrazlaganja ponašanja i zaključaka. Ekspertni sistemi u svom radu nastoje oponašati čoveka-eksperta, koji osim što rešava složene probleme, može i obrazložiti svoje odluke. Obrazlaganje zaključaka i ponašanja se sastoji od identifikovanja koraka u procesu rezonovanja i provere njihove ispravnosti. Obrazlaganje zaključaka i ponašanja spada u domen razumevanja prirodnog jezika, budući da sistem formuliše objašnjenja na način razumljiv korisniku, u skladu sa nivoom ekspertize korisnika. U postojećim ES, ova komponenta je svedena na izlistavanje pravila aktiviranih u procesu izvršavanja programa i provere njihove ispravnosti u okviru ovog podsistema.

Korisnički interfejs. Uloga ovog dela je u prihvatanju informacija koje zadaje korisnik i njihovom prevodjenju u oblik razumljiv za sistem, kao i u transformaciji podataka koje pruža sistem u oblik razumljiv za korisnika. Ova komunikacija korisnika sa računarom treba da se odvija na jeziku što sličnijem prirodnom, pogotovo što računar zamenjuje čoveka-eksperta, tako da se često ugradjuju sistemi za obradu prirodnog jezika u okviru interfejsa.

SREDSTVA I JEZICI EKSPERTNIH SISTEMA

Složenost izrade i rada ekspertnih sistema zahteva i razvoj obimnih i složenih programa. Izbor tehnike predstavljanja i rukovanja znanjem, razvoj mehanizama zaključivanja, kontrole, pružanja dodatnih informacija itd. je bitno olakšeno ukoliko postoji posebno sredstvo za razvoj ES.

Aktivnosti oko razvoja ekspertnih sistema mogu se razvrstati u sledeće kategorije:

- programski jezici za razvoj ES,
- jezici inženjeringa znanja,
- programska podrška izgradnje sistema i
- olakšice izgradnje ES.

Programski jezici mogu biti problemsko-orijentisani, kao što su FORTRAN, PASCAL, C, ili jezici koji služe za manipulisanje simbolima, kao što su LISP i PROLOG. Zbog svojih osobina pogodnih za izgradnju ES, najčešće su korišćeni programski jezici LISP, PROLOG i OPS5. U poslednje vreme, sve više se šire ES, koji su pisani na tradicionalnim jezicima kao što su PASCAL i jezik C. Svi ovi jezici pružaju veliku slobodu pri izgradnji ES, ali ne pomažu pri izboru načina predstavljanja znanja ili pristupu bazi znanja.

Za razliku od programskih jezika opšte namene, ovi su programski jezici na najvišem nivou i namenjeni su isključivo konstruisanju ES. Ovi programi su jezici za inženjering znanja i pružaju posebne olakšice pri radu, ali su manje fleksibilni sa aspekta predstavljanja i manipulisanja znanjem. Ovi jezici su inteligentan alat za razvoj ES. Sastoje se od jezika za izgradnju ES kojima je pridodato široko razvojno okruženje. Ukoliko je ovom programu potrebno dodati samo znanje specifično za odabrani domen, tj. domensko znanje, da bi on prerastao u ES, reč je o jeziku ogoljenog tipa. Takvi jezici se nazivaju ljušturama ES. Ljuske pored gotove strukture, pružaju i niz olakšica za brzi i jednostavan razvoj ES, ali im nedostaje opštost i fleksibilnost u rukovanju problemima koji ne spadaju u unapred odredjenu klasu, za koju su namenjeni. Nazivi nekih ljuski su EXSYS Professional, KAS, GURU, M1, NEXPERT-OBJECT, LEVEL5.

Programsku podršku izgradnje ekspertnog sistema čini skup programa, koji pomažu ili pri sticanju i predstavljanju znanja domenskog eksperta ili pri konstrukciji ES. Programi koji povezuju znanje eksperta sa bazom znanja rade na principu interakcije računara i korisnika. Korisnik je u mogućnosti da iskaže nova pravila pomoću redukovanog skupa engleskih reči, koje sistem analizira, daje predloge u zavisnosti od potpunosti i konzistentnosti pravila i pomaže pri otkrivanju grešaka. Programi koji pomažu pri dizajnu i izgradnji ES, nude inženjeru znanja odredjeni skup koncepata, koje ovaj koristi kao blokove za gradnju: spaja ih po potrebi u delove ES. Svaka komponenta je skup funkcija nekog programskog jezika i podržava ulančavanje unapred, unazad, ili arhitekturu table. Ova grupa programa je moćno oruđe u rukama iskusnog inženjera znanja koji dobro poznaje jezik kojim su ovakvi programi napisani.

Olakšice izgradnje ES, u formi pomoćnih alata pri programiranju, takođe se mogu uvrstiti u sredstva za izradu ES. Ovi alati su debugger, editor baze znanja, ugradjeni U/I mehanizmi, mehanizam za davanje objašnjenja. Pri izboru sredstva za izradu ES, moramo biti sigurni da je odredjeni alat odgovarajući s obzirom na tip problema koji rešavamo. Problemi se javljaju kada ni jedan od postojećih alata ne zadovoljava naše potrebe u potpunosti. Jedno od rešenja za nastali problem jeste kreiranje novog alata, sa jedinstvenom prezentacijom znanja, mehanizmom zaključivanja i tehnikama kontrole, napisan u nekom opštem jeziku veštačke inteligencije. Sredstva i jezici ES se mogu koristiti za mašinsko prihvatanje znanja eksperta, ali to znanje prvo mora se dobiti od čoveka eksperta.

STICANJE ZNANJA ZA EKSPERTNE SISTEME

Rešenja odredjenih problema, dobijena ES -om će u onoj meri biti zadovoljavajuća, koliko je ekspertsko znanje adekvatno ugradjeno u sistem. Pogrešno tumačeno, dvosmisleno ili parcijalno obuhvaćeno znanje su direktni preduslovi za loše funkcionisanje ES. Potrebno je u odabranom domenu crpiti znanja od eksperta da bi se postiglo što potpuniji i tačniji opis znanja i da se predstavi način na koji se ekspert koristi svojim znanjem. To je zadatak inženjera znanja ili projektanta baze znanja. Crpljenje znanja od eksperta je složen zadatak. Odvija se pomoću nekih od sledećih tehnika:

- ispitivanje eksperata - od eksperta se traži da se priseti šta je činio, ili da zamisli kako bi postupao pri rešavanju odredjenog problema. Od raznih oblika, jedan oblik je deskripcija, kada se inženjer znanja upoznaje sa bazičnim informacijama i strukturom posmatranog domena putem opisa

koje daje čovek-ekspert. Predstavljaju se "idealni slučajevi". Drugi oblik daje sveobuhvatnije znanje o domenu, a to je introspekcija. Inženjer znanja ispituje eksperta u cilju razjašnjavanja sopstvenih nedoumica i radi iznalaženja znanja iz posmatranog domena. Metode za ispitivanje eksperta su:

- intervju,
- repertoarske rešetke,
- skale procenjivanja,
- tehnika kritičnih dogadjaja,
- uparivanje karakteristika i odluka, razlikovanje ciljeva,
- reklasifikacija,
- analiza odlučivanja.

- posmatranje eksperta na delu - ovaj metod je usredsredjen na ono što ekspert čini u aktuelnoj situaciji. Posmatranje eksperta na delu, omogućava inženjeru znanja da zapazi mnoge bitne detalje koji deskripcijom ostaju neuočeni. Ova tehnika pretpostavlja da je inženjer znanja upoznat sa osnovnim domenskim znanjem, zato se ona kombinuje sa prethodno navedenim metodama ili proučavanjem pisanih izvora. Tokom rešavanja problema, na zahtev posmatrača, ekspert može glasno razmišljati i tako dati potrebna razjašnjenja i obrazloženja svojih odluka, postupaka i radnog ponašanja.

PREDSTAVLJANJA ZNANJA I ZAKLJUČIVANJA

Ekspertni sistem je računarski program, koji rešava složene probleme realnog sveta, koji zahtevaju obimnu ljudsku eksperetizu. Ovo stručno znanje je uglavnom heurističke, praktične prirode, nesigurno je i nije uvek zakonito i dosledno.

Da bi došao do rešenja, ES oponaša proces ljudskog rezonovanja, primenjujući specifično znanje i zaključivanje. Da bi ES mogao pristupiti rešavanju problema, u njega mora biti uključeno razumevanje osnovne procedure, što je uopšteni oblik ekspertize, zajednički za većinu domena.

Domensko znanje eksperta mora se prevesti u oblik pogodan za kreiranje ES. To se vrši odredjenim tehnikama. Inženjeri znanja analiziraju znanje eksperta i na odredjeni način to znanje predstavljaju u softverskom okruženju koje obuhvata ES, kao relevantnu informaciju.

Ukoliko arhitekturu ES predstavljamo na način kako je prikazano na slici 3., prema kojem se on sastoji od:

- baze znanja,
- mehanizma za zaključivanja i kontrole,
- podsistema za uzimanje i ažuriranje znanja,
- podsistema za obrazlaganja ponašanja i zaključaka,
- korisničkog interfejsa,

tada možemo reći da ćemo se zadržati na prvoj tačci i obraditi strategije za predstavljanje znanja uz poseban osvrt na mehanizme zaključivanja.

OSNOVI INŽINJERINGA ZNANJA

Za opis svoje discipline, istraživači, na polju ES, odomaćili su izraz "inženjer znanja", koji kombinuje naučne tehnološke i metodološke elemente. Kao princip inženjera znanja, drži se da stručno obavljanje nekog posla, retko odgovara strogo algoritmičkom procesu, uz to, takvo obavljanje posla, sposobno je za obradu na računaru. Izvedeno, jasno izrečeno, i na računaru obradjeno znanje stručnjaka, čini ključni zadatak na ovom području. Zbog velikog značaja ove problematike, razmotriće se ljudsko znanje i iskustvo.

ZNANJE I ISKUSTVO

Uopšteno shvaćeno, znanje sadrži opis, uzajamne veze i procedure nekih domena interesa. Opisi u bazi znanja, koja identifikuju i razlikuju objekte i klase, su rečenice na nekim jezicima. Elementarni sastojci ovih jezika su jednostavne karakteristike ili koncepcije. Opis sistema uglavnom sadrži pravila i procedure za primenu i interpretaciju opisa u specifičnim aplikacijama. Baza znanja takođe sadrži posebnu vrstu opisa, koje su poznate kao uzajamne veze. Oni izražavaju zavisnosti i asocijacije izmedju stavaka u bazi znanja. Takve tipične uzajamne veze opisuju pojmovne, definicijske i empirijske asocijacije. S druge strane, procedure određuju operacije za izvršenje kada se pokuša dati zaključak, ili rešiti problem.

U praksi, znanje se ne pojavljuje u nekim strogim formama, koje se čisto kvalifikuje u te abstraktne kategorije. Njegov položaj je kao neka besmislena i nerazjašnjena pojava, znanje, koje na neki način omogućuje čoveku stručnjaka da reši teške probleme. Ima više formi. Forma se često sastoji iz empirijske asocijacije. Doktori i geolozi, na primer, poseduju znanje iz više takvih formi asocijacija, u kojoj su slični uzroci za posmatrane činjenice. Posledica takvog znanja je da stručnjaci kombinuju heurističke metode sa verovatnim, pogrešnim i nesigurnim podacima i zaključcima. Mnogi stručnjaci poseduju drugačiju formu znanja, i to u formi koncepcije, ograničavanja i regulacije, a te operacije su odlučujuće na njihovim područjima. Oni takodje mogu iskoristiti uzročne modele njihovih studija, kao šeme zaključivanja koje koriste te modele za predvidjanje, dijagnosticiranje, planiranje ili za analizu situacije. Ukratko, znanje se sastoji iz:

- simboličkog opisa, koji opisuje definisanu i empirijsku međuzavisnost u datom domenu i
- procedure za manipulaciju tih opisa.

Za razumevanje ekspertnih izvodjenja, pomaže ako se razmotri razlika između znanja i iskustva. Iskustveno izvođenje nekog zadataka često sadrži mnoge karakteristike, koje nisu prisutne u dobro informisanom ali neiskustvenom poslu, kao što su velika brzina, smanjenje grešaka, smanjenje kognitivnog tereta (zahtevane pažnje) i prilagodjavanje vitalnost pri razvoju. Takve karakteristike se održavaju i u znanju i u inženjeringu. Iskustvo znači imati pravo znanje i koristiti ga efikasno.

ORGANIZACIJA ZNANJA I ZAKLJUČIVANJA EKSPERTNIH SITEMA

Inženjer znanja, kad počinje svoj rad na izgradnji ekspertnog sistema, teži prvo da izvuče znanje eksperata i tek onda ga organizuje u sistemu za efektivno izvršavanje.

Neke tehnike sticanja znanja za ES, navedeni su u poglavlju ‘Sticanje znanja za ekspertne sisteme’. Ovo stečeno znanje se organizuje u bazi znanja ES na sledeće načine:

- kao produkcioni sistem,
- kao struktuirani objekti, tipa:
  - semantičke mreže,
  - trojke objekat-atribut-vrednost,
  - frejmovi,
- upotrebom predikatske logike.

PREDSTAVLJANJE ZNANJA PRODUKCIONIM PRAVILIMA I SISTEM ZAKLJUČIVANJA

Procedura je takva metoda, u kojoj se korak po korak postiže neki specifični rezultat. Produkcioni sistemi se sastoje od:

- globalnog memorijskog područja,
- skupa produkcionih pravila i
- interpretera koji ispituje trenutna stanja i izvršava pravila.

Globalni memorijski elementi su memorijska područja, koje se koriste za validno stanje posmatranog okruženja. Namenjeno je otkrivanju stanja posmatranog sistema, i sastoji se od niza memorijskih elemenata. Svaki elemenat sadrži opis jednog objekta, koji je bitan za posmatrani problem. Ovaj opis je u obliku simboličkog identifikatora. Svakom opisu se udružuje par atribut-vrednost. Svaki takav par opisuje trenutna, aktuelna stanja i vrednosti pridruženih atributa.

Skup produkcionih pravila je tehnika predstavljanja znanja, koja ukazuje na odnos izmedju preduslova i posledica. Ima sledeći oblik:

IF uslov **THEN** akcija.

Interpreter je potreban za prepoznavanje i izvršavanje pravila. On uporedjuje preduslov pravila sa stanjem memorijskih elemenata i u slučaju podudarnosti aktivira pravilo. Proces se završava uspešno dolaskom do rešenja ili neuspešno usled nepostojanja pravila koje bi se moglo aktivirati.

ZAKLJUČIVANJE

Sukcesivnim aktiviranjem više pravila, nastaje lanac zaključivanja. U zavisnosti od načina aktiviranja pravila, dobijamo dva različita mehanizma:

- ulančavanje unapred to je strategija vodjena podacima, jer korisnik zna uslove a traga za posledicama,
- ulančavanje unazad to je strategija vodjena ciljem, jer su poznati simptomi, rezultati, posledice, a traga se za uzrocima koji su do njih doveli.

Kod ulančavanja unapred se traga za novim informacijama, koji su potrebni za rešavanje problema. Mehanizam zaključivanja nastoji da na osnovu postojećih podataka u memoriji, dođe do željenog zaključka. Ako nema dovoljno podataka za aktiviranje nekog pravila, kontrola se vraća korisničkom interfejsu, u suprotnom se

aktivira pravilo čiji uslovni deo odgovara zadatim uslovima. Korisnički interfejs igra veliku ulogu u prikupljanju potrebnih informacija, putem kojeg se korisniku, u vidu opcija menija prikazanog na ekranu, pruža mogućnost izbora stanja koje najviše odgovara stvarnoj situaciji. Korisnik izborom jedne ili više ponudjenih opcija, omogućava dodatnu informaciju potrebnu za rešavanje problema, koju mehanizam zaključivanja prebacuje u radno područje memorije.

Ulančavanje unazad. Kod većih sistema, čiji broj pravila prelazi red veličina od nekoliko stotina pravila, pri olančavanju unapred se može desiti da su aktivirana mnoga pravila koja iako imaju odgovarajuće posledice, nisu relevantna za samo rešenje problema. Da bi izbegli aktiviranje suvišnih pravila, koja su nepotrebna za izvođenje određenog zaključka, a da bi se i vreme uštedelo, efikasnije je primeniti ulančavanje unazad. Kod ovog mehanizma za zaključivanja, sistem polazi od onoga što želi dokazati, te se samo ona pravila izvršavaju koja su relevantna za posmatrani cilj.

PREDSTAVLJANJE ZNANJA PUTEM STRUKTUIRANIH OBJEKATA

Ova metoda predstavljanja znanja bazira se na objektno-orijentisanom modelovanju podataka, koji se obrađuju objektno-orijentisanim programiranjem.

OBJEKTNO-ORIJENTISAN PRISTUP MODELOVANJU PODATAKA

Postoje dva nova puta kojima se krenulo u modelovanju podataka. Jedan vodi ka proširenom relacionom modelu podataka, a drugi ka objektno-orijentisanoj tehnologiji baza znanja i podataka.

Kroz projektovanje arhitekture softvera, projektant se sukobljava sa izborom strukture koja može biti bazirana na funkciji ili na podacima. Funkcionalni pristup je nezamenljiv u kreiranju inicijalne verzije softverskih proizvoda, medjutim, nije pogodan za trajan proces kontinualnih promena i adaptacija. Radi kontinualnosti je potrebno vršiti promene u modulima, a ne u celokupnoj strukturi sistema. Tako je nastao pristup razvoju softverskih proizvoda preko podataka, tzv. objektno-orijentisana tehnologija. Svet je modeliran od objekata, uredjaja, poslova i pogodan je za organizaciju modela putem računarske prezentacije tih objekata. U apstraktnoj ili fizičkoj realnosti, objekat postoji tamo, gde se može odrediti.

Objekat je apstrakcija koja predstavlja bilo kakvu posebnost uočenu u realnom sistemu, deo sistema koji se po nekim svojstvima razlikuje od ostalih. Ta svojstva po kojima se objekti uočavaju, predstavljaju zapravo odnos koji objekat ima prema drugim objektima. Uočavanjem vrsta svojstava objekata i tipova njihovih odnosa, omogućuje se struktuiranje realnog sistema i formiranje njegovog modela.
Postoji više načina predstavljanja podataka u bazi znanja putem struktuiranih objekata. To su:

- semantičke mreže,
- trojke objekat-atribut-vrednost i
- frejmovi.

Jedna od najstarijih načina predstavljanja znanja u oblasti veštačke inteligencije su semantičke mreže. Semantička mreža je sredstvo reprezentovanja znanja u vidu

obeleženog usmerenog grafa. Svako teme grafa predstavlja koncept, a svako obeležje relaciju između koncepata. Teme grafa sačinjava kolekcija objekata koji se nazivaju čvorovima. Obično su i veze i čvorovi imenovani. Postoje dobre i loše mreže, koje su prezentacija takvog znanja, koja u većoj ili manjoj meri podržava karakteristike ljudskog rezonovanja. Da bi se ocenila "dobrota" semantičke mreže, uvodi se mera konceptualne razdaljine, (tj. semantičke različitosti izmedju dva koncepta predstavljenih čvorovima ili skupom čvorova u mreži), a vezama tipa "jeste" se dozvoljava da istovremeno označavaju i semantičku udaljenost (udaljenost čvorova u semantičkoj mreži) i konceptualnu razdaljinu.

Trojka objekat-atribut-vrednost. Pod objektom se podrazumevaju fizički entiteti ili konceptualni entiteti. Atributi su opšte osobine ili svojstva pridruženi objektima. Vrednost odredjuje prirodu atributa u odredjenoj situaciji. ES u fazi sticanja znanja pokušava saznati vrednosti raznih atributa važnih za pojedine situacije. Kako je koja vrednost dosegnuta, sistem je pohranjuje u radnu memoriju. Ovaj postupak je zamenjivanje.

Predstavljanje znanja frejmovima i zaključivanja. Frejmovi omogućavaju još jedan način predstavljanja podataka i relacija. Pravila pojednostavljuju odnos "pretpostavka i posledica" izmedju pojmova, predmeta ili dogadjaja. Frejmovi prvenstveno izražavaju bitne komponente pojedinih pojmova, predmeta ili dogadjaja.

Ideju frejmova je razvio Marvin Minsky. Po njemu, frejm je mreža čvorova i relacija, organizovanih, gde čvorovi na vrhu predstavljanju uopštene koncepte (objekte), a niži čvorovi specifične delove tih koncepata. Svaki objekat predstavljen čvorom je opisan skupom atributa i vrednostima tih atributa, pri čemu se atributi nazivaju slotovi i nose informacije o datom objektu frejma. Na ovaj način je objekat povezan sa skupom činjenica, pravila, procedura, podrazumevanih (default) vrednosti, displej instrukcija, koje ga nedvosmisleno konstituišu. Pridružene procedure su proceduralna reprezentacija u okruženju koje je deskriptivnog karaktera, odnosno medju tvrdnjama koje su istinite, a predstavljaju znanje o nekom objektu, mogu se naći eksplicitno navedeni skupovi instrukcija čiju tačnost moramo ispitati.

Frejmovi i trojke objekat-atribut-vrednost mogu da se posmatraju kao specifični slučajevi semantičkih mreža, a isto tako mogu predstavljati njihove delove, odnosno pojedini čvorovi i veze u mreži mogu biti predstavljeni putem frejmova ili trojki O-A-V. Kako između semantičkih mreža, frejmova i trojki O-A-V postoji visok stepen srodnosti, jedno te isti podaci u bazi znanja mogu biti predstavljeni na ma koji od ova tri načina.

PREDSTAVLJANJE ZNANJA PREDIKATSKIM RAČUNOM

Predikatski račun osim odnosa između iskaza uzima u obzir i odnose unutar iskaza. Jezik predikatskog računa sadrži sve elemente jezika iskaznog računa (iskazne promenjive: p,q,r,...; logičke konstante: $\cup, \cap, \equiv, \supset$ ...; i zagrade: ( ), a pored njih individualne termine (x, y, z,...), predikatske termine (F, G, H) i kvantifikatore ($\forall, \exists$).

Individualne termine sačinjavaju individualne konstante i individualne promenljive.

" Kanarinac ima perje ",

" x ima perje ",
" x ima y "

Predikatske termine sačinjavaju predikatske konstante i predikatske promenljive.

F(x) znači " individua x ima svojstvo F ",
F(x,y) znači " x je u odnosu sa y "

Iskazna funkcija je rečenica koja postaje iskaz kada se individualne promenljive zamene individualnim konstantama. Ona je logička formula, koja sadrži promenljive izvan oblasti dejstva kvantifikatora (izvan zagrada koje slede odmah iza kvantifikatora), tj. slobodne promenljive.

Naučno znanje se stiče, povezuje u celinu, dokazuje, proverava i razvija zaključivanjem. Iskazi se mogu prihvatiti kao rečenice kojima se izražava znanje, ako se zasnivaju na adekvatnom i značajnom svedočanstvu koje se takođe navodi u vidu iskaza. Zaključivanje je uspostavljanje takvog odnosa medju iskazima posredstvom kojeg se na osnovu poznate istinitosne vrednosti nekog iskaza procenjuje istinitosna vrednost nekog drugog iskaza. Zaključivanje se može shvatiti i kao proces izvodjenja jednog iskaza ili suda iz drugih iskaza.

Za obradu podataka iskazni račun je dovoljan, ali okruženje ekspertnih sistema zahteva znatno širi skup logičkih alata. Ne može se zadovoljiti samo istinitosnim vrednostima određenih tvrđenja koja su uključena u bazu znanja, već je potrebno iskazati i utvrditi prisustvo relacija izmedju objekata u bazi znanja.

U gornjem primeru može se uvesti predikat **"ima"** u oznaci **I**, kanarinac označiti sa **k**, a perje sa **p**. Gornji iskaz u formi simboličke predikatske logike je **I(k,p)**. Tvrdnja **"x je kanarinac"** se po analogiji sa prethodnim primerom može predstaviti kao **K(x)**, gde je **K** unaran predikat, sa semantikom **"je kanarinac"**.

Okruženje ekspertnih sistema zahteva razvoj proširenog skupa pravila zaključivanja koji će omogućiti izvođenje zaključaka iz izraza predstavljenih simboličkom predikatskom logikom.

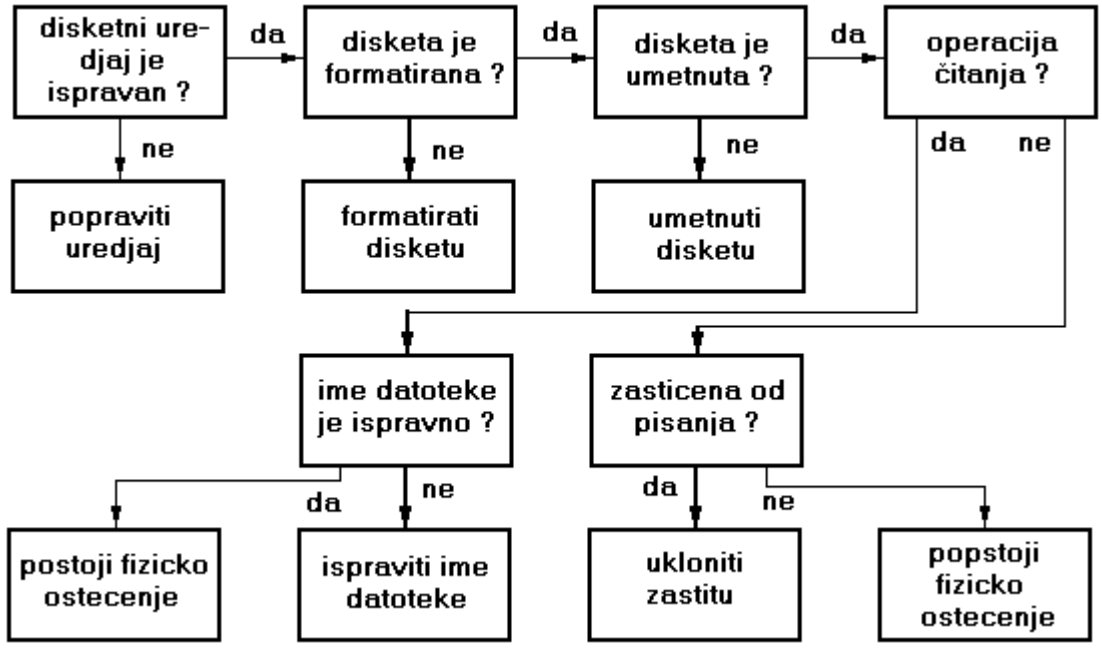
POMOĆNE TEHNIKE PREDSTAVLJANJA ZNANJA

Ovi načini predstavljanja znanja se upotrebljavaju kao dodatni, pomoćni alati kada je baza znanja suviše \ kompleksna, ili se posmatrani domen sastoji iz mnoštva informacija kojima se služimo na različite načine u postupku dolaženja do rešenja. Tabele odlučivanja mogu da posluže kao ulaz za semantičke mreže ili kao ulaz za stabla odlučivanja pomoću kojih će se formulisati produkciona pravila.

Tabele odlučivanja. Pomoću njih se na efikasan način može obuhvatiti i sortirati veliki obim podataka i opisati znanje, zbog toga što pregledno prikazuju mnogobrojne kombinacije opcija i karakteristika posmatranog sistema.

Tabela 15.: Tabela odlučivanja		PRAVILA
		1,2,3,4....N-1,N
1 2 . . K	Kriterijumi uslova počinju sa "ako"	Oznake ispunjenosti uslova
1 2 . . M	Aktivnosti koje treba preduzeti, počinju sa "tada"	Oznake koje aktivnosti treba izvršiti

Stablo odlučivanja. Može se shvatiti kao hijerarhijska semantička mreža, povezana nizom pravila, koja objedinjuje strategije pretraživanja sa relacijama među znanjem. Potezi mreže su odluke, a čvorovi su ciljevi. Mreža se pretražuje od korena ka listovima, s leva na desno. Svi krajnji čvorovi, listovi stabla, su delovi krajnjeg cilja. Na slici 23 dato je stablo odlučivanja za slučaj ispravnosti ili kvara disketne jedinice.



Slika 23: Stablo odlučivanja

DEKLARATIVNI JEZIK PROLOG

UVOD

PROLOG je deklarativni programski jezik koji je namenjen logičkom programiranju. Naziv PROLOG je skraćenica od engleskih reči PRO(gramming) in LOG(ic). PROLOG je *jezik veštačke inteligencije.* PROLOG se razlikuje od ostalih proceduralnih višeprogramskih jezika.

Veštačka inteligencija (Artificial intelligence – AI): Oblast informatike, koja se razvila pslednjih godina pod uticajem lingvistike i psihologije. Zadatak veštačke inteligencije je rešavanje opštih problema vezanih za ljudsku inteligenciju korišćenjem računara. U ove probleme spadaju: obrada ljudskog govora, stvaranje ekspertnih sistema, prepoznavanje oblika itd.

ISTORIJA

Autor PROLOGA je Francuz Alen Colmerauer. Prva verzija jezika pojavila se 1971god. u Marseju.

Godine 1977 Warren je u Edinburgu napravio interpretator i kompajler za PROLOG. Japanci naučnici su početkom osamdesetih godina objavili da će njihovi računari pete generacije biti zasnovani na PROLOGU. PROLOG je zamišljen kao mašinski jezik japanskih računara pete generacije. Od tada je napisano mnogo ekspertnih sistema na PROLOG. Zajedno sa LISP-om, PROLOG postaje najpoznatiji jezik veštačke inteligencije.

Zbog nepostojanja zvaničnig standarda postoje razne verzije PROLOG-a, kao što su:

- mikro-PROLOG,
- turbo-PROLOG
- m-PROLOG, itd.

KLASIFIKACIJA PROGRAMSKIH JEZIKA I PROLOG

Postoje razne klasifikacije programskih jezika koje zavise od različitih kriterijuma. Najčešće se kao kriterijum koristi stepen zavisnosti od računara (hardvera).

Ako se kao kriterijum izabere način rešavanja problema, onda sve više programske jezike možemo podeliti u dve klase:

- proceduralne (imperativne)
- deskriptivne (opisne).

Prilikom programiranja na proceduralnom programskom jeziku, programer preko algoritama (procedura) saopštava računaru KAKO da reši problem. Većina poznatih programskih jezika (kao što su: FORTRAN, COBOL, BASIC, Pascal, Ada, itd) pripada klasi proceduralnih programskih jezika.

Za razliku od proceduralnih jezika, prilikom programiranja na deskriptivnom jeziku, programer opisuje ŠTA program treba da radi. U ovom slučaju glavni posao oko načina rešavanja zadatka treba da uradi interpretator (izvršilac) programa. PROLOG je jedan od najpoznatijih predstavnika klase deskriptivnih jezika.

OSNOVE PROLOG-A

Teorijsku osnovu PROLOG-a čini predikatski račun prvog reda. Međutim, postupak nalaženja rešenja u PROLOG-u zasniva se na principu rezolucije za automatsko dokazivanje teorema koji je pronašao J. Robinson 1965. godine. U stvari, radi se o jednoj modifikaciji principa rezolucije za tzv. Hornove disjunkte. Naime, osnovni elementi PROLOG-a nazivaju se Hornovi disjunkti (klauzule) i predstavljaju rečenice oblika:

AKO c1, c2, … cn **ONDA** c.

Ovo se može zapisati i ovako:

c ako c1, c2, …, cn.

ili u običajenoj notaciji gde se umesto reči ako koristi spacijalni simbol

:- (tzv. vrat-simbol jer spaja glavu predikata c sa telom: c1, c2, …, cn)

c:-c1, c2, …, cn.

Predikati koji nemaju telo nazivaju se **činjenice** u PROLOG-u. Predikati bez glave nazivaju se **ciljevi** ili **upiti**.

U opštem slučaju, u matematičkoj logici, pogodbene rečenice mogu imati oblik:

AKO p1, p2, …, pn **ONDA** c1, c2, …, ck.

Ovde je reč o klauzuli koja ima više glava i takve klauzule se koriste u PROLOG-u. Ako klauzula ima samo jednu glavu ili nema glavu, tada se naziva Hornova klauzula.

Princip rezolucije u PROLOG-u primenjuje se na Hornove klauzule. Sve činjenice u PROLOG-u tretiraju se kao apsolutne istine tj. aksiome.

Primena metoda rezolucije sastoji se u usaglašavanju postavljenog cilja (upita) sa činjenicama, tj. bazom podataka. Naime, ako je postavljen cilj, PROLOG nastoji da ispuni taj cilj u skladu sa bazom činjenica i pravila. Ako PROLOG u svom nastojanju uspe da ispuni cilj, daje pozitivan odgovor (‘YES’), u suprotnom negativan (‘NO’).

O PROGRAMIRANJU U PROLOGU

NAČIN REŠAVANJA PROBLEMA U PROLOG-U

U Hornovim klauzulama **c1, c2, …, cn** su klauzule (rečenice) koje mogu sadržati konstante i promenljive. Elementi **c1, c2, …, cn** nazivaju se potciljevi. Glavni cilj

može uspeti (biti ispunjen) ako uspe svaki potcilj u okviru glavnog cilja. Prema tome, glavni cilj:

:- c1, c2, …, cn

(koji ćemo označavati i na sledeći način: **?-c1, c2, …, cn**) predstavlja konjunkciju potciljeva: **c1, c2, …, cn**. Svaki potcilj može se shvatiti kao poziv procedure. Procedure se zadaju preko predikata sa odgovarajućim imenom i odgovarajućim brojem argumenata.

Potciljevi se ispunjavaju u procesu nalaženja rešenja, tj. u procesu izračunavanja. Proces rešavanja može da se razgrana različitim putevima i da generiše više različitih rešenja.

U procesu ispunjavanja (zadovoljavanja) cilja mogu nastati tri slučaja:

- proces traženja se završava i daje određeno rešenje, u ovom slučaju reč je o uspešnom ispunjenju cilja,
- proces traženja se završava neuspešno i ne daje rezultat, tj. cilj nije ispunjen,
- proces traženja se ne okončava i ne daje nikakav rezultat.

U prvom i drugom slučaju reč je o završenom izračunavanju, a u trećem slučaju o nezavršenom.

Za svaki cilj postoji potpun prostor izračunavanja koji može da se razmatra kao skup svih puteva duž kojih se može tražiti. Duž nekog od tih puteva može se uspešno okončati traženje, duž nekog traženja se završava neuspešno, a duž nekih traženja može da se ne završi. Potpun prostor izračunavanja određen je ciljem i pravilima za izračunavanje.

PROLOG ispunjava gavni cilj tako što ispunjava svaki potcilj u okviru cilja. Standardni način zadovoljavanja potciljeva realizuje se počev od prvog levog potcilja. Ukoliko se ispuni prvi levi potcilj, PROLOG nastoji da ispuni prvi potcilj iza njega itd. Ako u procesu ispunjavanja neki potcilj ne može biti ispunjen, PROLOG se vraća na prethodni potcilj i nastoji da ga ispuni za neke druge vrednosti. Taj postupak se nastavlja sve dok ne budu iscrpljene sve mogućnosti. Postupak traženja se vraćanjem (backtracking) omogućava nalaženje svih rešenja iz skupa mogućih rešenja i karakterističan je za PROLOG.

Navedimo primer na kojem će biti konkretno objašnjeni prethodno pomenuti pojmovi.

Neka je dat sledeći PROLOG-program:

```
Vozac(pavle).
Vozac(marko).
Vozac(X):- dobar_vozac(X).

Dobar_vozac(X):- ne_pije(X), vozac_autobusa(X).
Dobar_vozac(janko).

Ne_pije(dusan).
```

Ne_pije(petar).

Vozac_autobusa(dusan).
Vozac_autobusa(ranko).
Vozac_autobusa(petar).

Navedeni program se sastoji iz Hornovih klauzula sa glavom koje predstavljaju činjenice i predikate. Slovom **X** je označena jedna promenljiva, a ostala imena, koja ne počinju velikim slovom su konstante.

Upit:

?- vozac(X).

predstavlja Hornov disjunkt bez glave. PROLOG-sistem nastoji da ispuni ovaj cilj nalažeći konkretne vrednosti za promenljivu X tako da cilj bude usaglašen sa skupom činjenica.

Potpun prostor izračunavanja može da se predstavi stablom pretraživanja. Stablo pretraživanja je konačno, proces nalaženja rešenja ne može neograničeno produžiti. Razlikujemo listove označene sa **[]** koji predsatvljaju čvorove uspeha i listove označene sa **?-** koji predstavljaju čvorove neuspeha.

Iz navedenog stabla pretraživanja mogu se sagledati svi putevi duž kojih postoji mogućnost da zadati cilj bude ispunjen. PROLOG, na zahtev korisnika može da pronađe sva rešenja koja ispunjavaju zadati cilj. Za razliku od PROLOG-a, kod proceduralnih jezika nije uvek jednostavno naći sva rešenja.

Prilikom traženja rešenja, PROLOG nastoji da ispuni cilj usaglašavajući ga sa bazom podataka počev od vrha ka dnu. To znači, da PROLOG započinje sa traženjem rešenja počev od prvog levog čvora u stablu pretraživanja, a zatim nastavlja obilazak stabla pretraživanja nalazeći sva rešenja. Ako u procesu traženja naiđe na čvor uspeha, PROLOG izdaje rešenje, u suprotnom ne obaveštava da je došao do čvora neuspeha. Način rada PROLOG-interpretatora može se delimično sagledati iz redosleda kojim izdaje rešenja. Testiranjem prethodnog programa dobijamo:

- **?- vozac(X).**
- **X = pavle ->;**
- **X = marko ->;**
- **X = dusan ->;**
- **X = petar ->;**
- **X = janko ->;**
- **no**

Proces traženja odvija se kretanjem od korena stabla ka listovima, ali i od listova ka vrhu. Kretanje od listova ka vrhu predstavlja traženje sa vraćanjem.

NAČIN PROGRAMIRANJA

Deklarativnost jezika i način programiranja u PROLOGU omogućavaju da problem koji se programom rešava bude iznesen (zapisan) u jasnijoj eksplicitnoj formi nego što je kod proceduralno orjentisanih jezika. Naime PROLOG program je mnogo manje opterećen "*tehničkim detaljima"* koja se odnose na opis procedure rešavanja problema. Zbog toga se često naglašava da sam proces programiranja u PROLOG-u pridonosi razumevanju problema (i pronalaženju optimalnog rešenja) neuporedivo više nego što je to kod proceduralno orjentisanih jezika, ali to nipošto ne znači da bi pisanju programa trebalo pristupiti pre nego što se problem u celosti razmotri i definiše. Način formulisanja problema i izbor metode njegova rešavanja nedvojbeno zavisi o sredstvima koja nam za to stoje na raspolaganju – dakle o jeziku i sistemu. Međutim aprioritetno vezivanje problema za programski jezik ujedno znači i usmeravanje pažnje s problema na formu njegovog programskog zapisa, što najčešće rezultira suboptimalnom i/ili nepotpunom rešavanju.

Pravilan pristup rešavanju problema karakteriše primena metode ***ODOZGO PREMA DOLE***. Problem se najpre postavja u opštim terminima a zatim sledi postepena razrada njegovih pojedinih komponenti. Tako razrađen problem zapisuje se u programskom jeziku. Tada deklarativnost jezika i modularnost programa omogućavaju da sam proces programiranja pridonese boljem razumevanju a time i optimalnom rešavanju problema.

Dobrim osobinama PROLOG programa smatramo one osobine koje su nužne za programiranje uopšte. Među njima su najpre:

- ispravnost programa
- efikasnost programa,
- preglednost,
- modularnost,
- dokumentiranost.

Ispravnost programa proverava se kao i obično testiranjem iako pažljivo čitanje može pružiti bolji uvid u PROLOG program nego što pruža testiranje.

Efikasnost programa može se povećati poboljšanjem algoritma rešavanja problema. S druge strane poznavanje proceduralne interpretacije programa omogućuje da se algoritam zapiše na najprikladniji način što takođe može uveliko povećati efikasnost njegova izvođenja. Važna je i primena rezova kojima se sprečavaju nepotrebna računanja. Valja pri tome znati da rez redovno ograničava prostor primenljivosti programa.

Dobro oblikovani PROLOG program obično je mnogo čitljiviji od programa pisanih u proceduralnim jezicima. Način oblikovanja programa uveliko zavisi o tome što programer smatra preglednim programom.

Prolog pruža mogućnosti da se program napiše na pregledan i modularan način.

Izbor reči argumenata čini da takav program bude i samokomentarisan.

SINTAKSA I SEMANTIKA PROLOGA

Abecedu PROLOG programa, to jest skup znakova jezika čine:

- **Velika slova:** *A,B,…,Z.*
- **Mala slova :** *a,b,…,z.*
- **Brojevi :** *0,1,2,…,9.*
- **Posebni znakovi:** **,+,-,/,[,…* .

Sintaksički niz znakova nazivamo termom. Termove delimo na:

- **konstantne**, koji mogu biti
  - atomi i
  - brojevi,
- **varijable, i**
- **strukture**

Konstante i varijable zovemo jednim imenom "*JEDNOSTAVNI TERMOVI* ", dok strukture nazivamo " *SLOŽENI TERMOVI* ".

KONSTANTE:

One se dele na atome i brojeve. Atom nazivamo korisnički niz znakova koji udovoljava jednom od sledećih pravila:

- Atom je niz slova brojki i posebnih znakova, koji počinju malim slovom abecede na primer:
  - *a*
  - *jasna*
  - *x_*
  - *x_y_drugi.*
- Atom je niz posebnih znakova poput:
  - *= =>*
  - *<= =*
  - *……..*
  - *:_*
  - *,*
  - ****
  - *;*
- Atom je niz znakova dat u znacima navođenja:
  - *'ovo je atom'*
  - *'+*!!k/n'*

U brojeve kao drugu vrstu konstanti spadaju realni brojevi poput:

- 123,
- –133,
- -0.154 itd.

VARIJABLE:

Varijable predstavljamo nizom slova, brojeva, i posebnim znakovima koje počinju velikim slovom ili posebnim znakom '_'. Varijable su na primer sledeći nizovi:

A
AA
_A
_a
_123
_*visa*_skola
VISA_SKOLA

Ali ne i VISA SKOLA, jer su to dve varijable.

U okviru ovog poglavlja proučićemo zadatke na osnovu kojih možemo upoznati osnovne principe rada PROLOG interpretatora (koji su ranije opisani), unifikacija i traženje sa vraćanjem, kao i neke osnovne ugrađene predikate i operatore za kontrolu toka programa i ulaza/izlaza.

Dajemo kratak pregled osnovnih operatora i predikata:

- **,** - tretira se kao '**AND**', određuje konjunkciju ciljeva,
- **;** - tretira se kao '**OR**', određuje disjunkciju ciljeva, izvršava se s leva nadesno i da bi uspeo mora biti ispunjen cilj bar sa jedne strane operatora,
- **!** - operator odsecanja (rez), cilj koji uspeva kada se do njega dođe pri izvršavanju 'nadole', a propada kada na njega naiđemo pri traženju sa vraćanjem, a takođe propada i ceo predikat u okviru koga se rez operator našao,
- **true** - cilj koji uvek uspeva,
- **false** - cilj koji uvek propada, pogodan je kada želimo traženje sa vraćanjem,
- **repeat** - kada se naiđe na njega uspeva, i pri traženju sa vraćanjem uspeva, realizovan je tako da omogući formiranje ciklusa pri izvršavanju programa.
- **not(P)** - ako P uspe, tada not(P) propada, a ako P propadne tada not(P) uspeva,
- **read(P)** - omogućava unošenje podataka,
- **write(P)** - omogućava izdavanje podataka,
- **nl** - prelazi u novi red prilikom izdavanja podataka.

Zadatak 1.

Glavni gradovi

Data je baza znanja o glavnim gradovima. Napisati PROLOG program za odredjivanje glavnog grada zemlje na osonvu date baze ako se u programu zadaje ime države.

Resenje:

/**********
**/

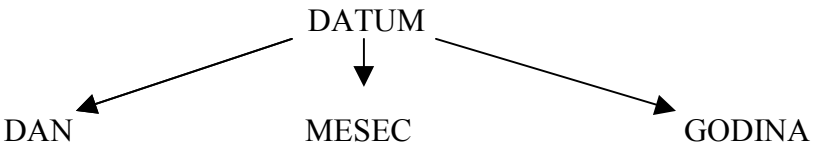
```
/*
*/
/*                                                    GLAVNI GRADOVI
*/
/*
*/
/*****************************************************************
**/

        glavni (srbija,beograd).
        glavni (nemacka,berlin).
        glavni (engleska,london).
        glavni (kina,peking).
        glavni (japan,tokio).
        glavni (makedonija,skoplje).

        glavnigrad  :- nl,nl,
                write('Unesite ime drzave i zavrsite tackom: '),
                read(R),
                glavni(R,G),nl,
                write('Glavni grad izabrane zemlje je:  '),
                write(G).
/*****************************************************************
```

STRUKTURE

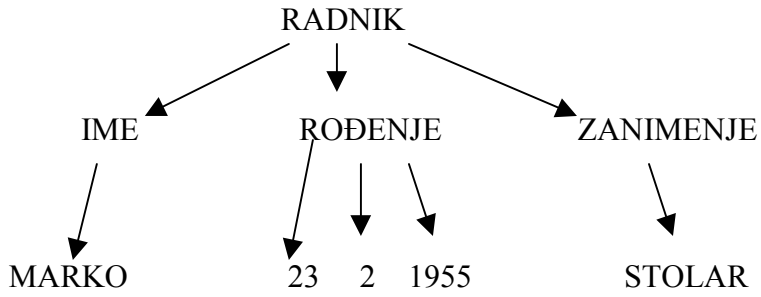
Struktura je složena term jezika koji se formira vezivanjem nekoliko jednostavnih termova. Struktura u jednom cilju vezuje funktor.

Opšta formula strukture je:
Funktor(ARGUMENT_1,…, ARGUMENT_n).

Funktor je atom u argumentu su termovi koji mogu biti jednostavni ili složeni. Struktura se označava kao na primer: **datum(Dan, Mesec, Godina)** grafički je prikazana kao:



Radnik (ime (Miroslav), rodjen (23.02.1955), zanimanje (stolar)).



STABLO PRETRAŽIVANJA

Stablo pretraživanja je formalno sredstvo pomoću kojeg prikazujemo sve moguće načine zadovoljavanja postojećeg cilja. Opštu definiciju stabla pretraživanja prilagodili smo ovde aktuelnim principima izbora implementiranim u standardnom sistemu PROLOG.

Cilj postavljen programu može se zadovoljiti na više načina. Pretraživanje ide prvo levo i zatim u dubinu.

STRUKTURE PODATAKA

Strukture koje se često javljaju u PROLOG-u su:

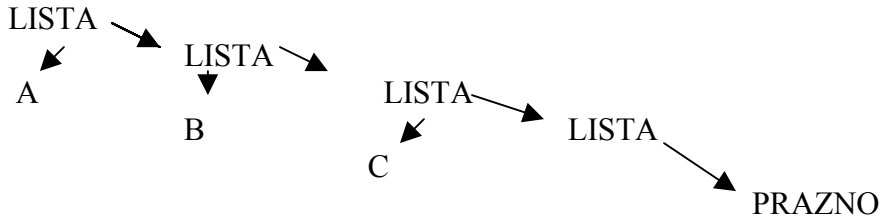
- **lista i**
- **binarno stablo.**

PROLOG program može biti namenjen predstavljanju znanja ili procesiranju znanja. U prvom slučaju program je baza znanja, u drugom slučaju program tom bazom upravlja. Reč je samo o dva načina posmatranja programa, jer u PROLOG-u nema formalne razlike izmedju podataka i programa.

LISTA

Lista je uređen niz elemenata. Pod uređenošću podrazumevamo da je redosled elemenata liste određen. Lista može sadržati i više jednakih elemenata. Listu možemo shvatiti kao binarnu strukturu. Za funktor te strukture odaberemo atom “lista”. Prvi argument strukture neka bude prvi element liste a drugi argument neka bude lista koja sadrži preostale elemente. Prvi element takve binarne strukture nazivamo **glavom** liste a drugi **repom** liste.

Lista (a, lista (c, prazno)).



Eksplicitnim pretstavljanjem liste kao binarne strukture lista brzo postaje nepregledna. Zato je u PROLOG-u uvedena posebna notacija za tu strukturu. Prema toj notaciji nepraznu listu pretstavljamo navođenjem njenih elemenata zatvorenih u uglaste zagrade, dok je prazna lista pretstavljena simbolom "[]".

U listi [a, b, c] u PROLOG-u a je glava liste, a [b, c] je rep liste.

Operacije sa listama su:

- provera pripada li neki element listi,
- provera pripadanja jedne liste drugoj,
- izdvajanje iz date liste neku drugu listu koja sadrži iste elemente ali u obrnutom redosledu,
- sortiranje liste,
- upis u listu,
- brisanje liste,
- brisanje jednog elementa date liste,

BINARNO STABLO

Binarno stablo pretstavljamo strukturom sledećeg oblika stabla (Element, Levo, Desno). Pri tome je drugi argument levo podstablo čvora. Prazno stablo možemo označiti praznim simbolom različitim od ostalih simbola koji već imaju neko posebno zaduženje.

Dužinu najdužeg lanca u stablu nazivamo i **dubinom stabla**.
Dubina praznog stabla je 0, dužina stabla koji ima samo jedan koren je 1.

OSNOVNE OPERACIJE SA BINARNIM STABLOM

Element jeste element binarnog stabla ako je:

- koren tog stabla,
- element njegovog levog podstabla,
- element njegovog desnog podstabla.

Brisanje elementa binarnog sistema:

- element koji se briše je lista stabla, to jest oba njegova potomka su prazna stabla, obrisani element zamenjujemo praznim stablom,
- element koji se briše ima samo jedno neprazno podstablo, neprazno podstablo dolazi na mesto stabla čiji je koren izbrisan,
- oba podstabla elementa koji se briše neprazna su,

- element koji se pokušava brisati nije sadržan u datom stablu, stablo ostaje nepromenjeno.

ARITMETIČKE I LOGIČKE OPERACIJE

Prolog je namenjen pre svega procesiranju struktura odnosno numeričkom procesiranju. Zato je i izbor predikata namenjen izvođenju računskih operacija relativno skroman. Aritmetičke izraze formiramo pomoću predikata +, -, *; / i mod:

- zbir x+y,
- razlika x-y,
- proizvod x*y,
- delenje x/y i
- delenje po modulu x mod y.

LOGICKE OPERACIJE

Jednako x= : =y razlicito x= \ =y vece od x > y jednako ili vece od x > = y manje od x < y jednako ili manje od x = < y

RUKOVANJE SA DATOTEKAMA

PROLOG učitava podatke s aktuelnog ulaza a odgovore ispisuje na aktuelni izlaz, dok to drukčije ne definišemo. Da bi to bilo moguće, program se najpre mora uneti u radnu memoriju PROLOG-sistema. Učitava se celokupni sadržaj datoteke u radnu datoteku PROLOG-interpretera.

PRAĆENJE RADA PROGRAMA

Standardni predikat praćenja rada programa je **trace**.
Cilj trace postavljamo sistemu pre postavljanja cilja čiji pokušaj zadovoljavanja želimo pratiti. Taj cilj uzrokuje da sistem (na ekranu) pokazuje:

- svaki cilj, počevši od polaznog, koji se pokušava zadovoljiti u sklopu pokušaja zadovoljavanja polaznog cilja i
- rezultat pokušaja zadovoljenja pojedinog cilja, ako pokušaj završi sa uspehom, onda su to vrednosti argumenata za koje je postavljeni cilj zadovoljen, u suprotnom, rezultatom smatramo informaciju o neuspehu zadovoljenja posmatranog cilja.

Pokazivanjem koraka u pokušaju zadovoljenja cilja sistem pokazuje tok pretraživanja.

PREDIKAT ODSECANJE I TRAŽENJE REŠENJA

U PROLOG-u postoji mogućnost da se spreči pretraživanje obilaskom celog stabla pretraživanja. U te svrhe koristi se sistemski predikat odsečanja koji se najčešće obeležava znakom uzvika '!'. Pomoću ovog predikata na stablu pretraživanja odsečaju se pojedine grane, a na taj način se sprečava traženje rešenja u potpunom prostoru

izračunavanja. Korišćenjem ovog predikata narušava se deklarativno svojstvo PROLOG jezika. Da bismo sagledali efekte predikata odsečanja, modifikujemo prethodno navedeni program na sledeći način:

```
Vozac_djaka(pavle).
Vozac_djaka(marko).
Vozac_djaka(X):- dobar_vozac(X).

Dobar vozac(X):- ne_pije(X), !, vozac_autobusa(X).
/* Modifikovan red */
Dobar_vozac(janko).

Ne_pije(dusan).
Ne_pije(petar).

Vozac_autobusa(dusan)
Vozac_autobusa(ranko).
Vozac_autobusa(petar).
```

Testiranjem dobijamo sledeci skup resenja:

```
?- vozac_djaka(X).
X = pavle  ->;
X = marko  ->;
X = dusan   ->;
no
```

Može se uočiti da su određene grane na stablu pretraživanja odsečene i da su odstranjeni određeni putevi za nalaženje rešenja.

ELEMENTARNI ZADACI SA LISTAMA

Osnovu rešenja mnogih problema u PROLOGU čini rad sa listama. Lista je struktura podataka koja predstavlja sekvencu elemenata koji mogu biti različitog tipa (atomi, brojevi, promenljive, liste…) i pišu se u srednjim zagradama.

Primer:

[] - prazna lista.
[a] - lista sa jednim elementom.
[a| X] - lista sa glavom **a** i repom **X** .

(X je promenljiva koja predstavlja listu koja cini preostali deo liste [a|X]. Ako je data lista [a, b, c, d], koja se pri radu programa unifikuje sa [a|X], tada se X unifikuje sa [b,c,d])

Veliki broj problema koji zahteva upotrebu slozenijih struktura podataka resava se svodjenjem na elementarne predikate za rad sa listama. Neke od tih, osnovnih, predikata proucicemo kroz zadatke u ovom poglavlju.

Primer: Član liste

Napisati PROLOG program za odredjivanje da li je dati element član liste.

- realizovati ga tako, da daje odgovor da li je dati element član liste, i daje sve elemente liste,
- isključivo proveravati da li je dati element član liste.

Posle pronađenog elementa program se "zamrzava" (onemogućava dalju pretragu pri eventualnom traženju sa vraćanjem u okviru složenih zadataka).

Rešenja:

ČLANOVI LISTE

```
clan(X,[X|_]).
clan(X,[_|Y])  :- clan(X,Y).
```

Poziva se sa: clan(element,lista).

Test-primer:

```
?- clan(3,[2,4,3,1]).

   yes

?- clan(2,[1,4,6,4]).

   no

?- clan(X,[2,3,4,1]).

   X = 2 ->;
   X = 3 ->;
   X = 4 ->;
   X = 1 ->;
   no
```

LITERATURA

- Berković, Ivana: Elementi veštačke inteligencije, Zrenjanin
- Damjanović, Boško: Principi programiranja, Nauka, Beograd
- Dujmović, Jozo: Programski jezici i metode programiranja, Naučna knjiga, Beograd
- Kovačević, Vladimir: Programska podrška računarskih sistema, FTN, Novi Sad
- Mišić, Vojislav: IBM PC u 25 lekcija, Tehnička knjiga, Beograd
- Petković, Dušan: Mali leksikon mikroračunarskih izraza, Savremena administracija, Beograd
- Ristanović, Dejan: Mašinsko programiranje, Tehnička knjiga, Beograd
- Ristić, Živan, Balaban, Neđo, Bošnjak, Zita: Ekspertni sistemi, Savremena administracija, Beograd
- --: Oksfordski rečnik računarstva