Glava 1

Hardver i softver

1.1 Hardver savremenih računara

Hardver čine opipljive, fizičke komponente računara. Iako je u osnovi savremenih računarskih sistema i dalje Fon Nojmanova mašina (procesor i memorija), oni se danas ne mogu zamisliti bez niza hardverskih komponenti koje olakšavaju rad sa računarom.

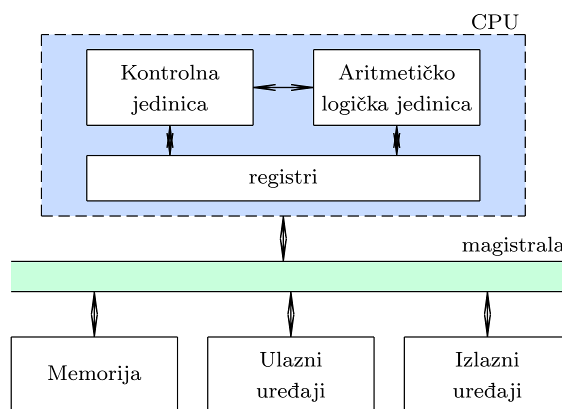
Iako na prvi pogled deluje da se jedan uobičajeni stoni računar sastoji od kućišta, monitora, tastature i miša, ova podela je veoma površna, podložna promenama (već kod prenosnih računara, stvari izgledaju znatno drugačije) i nikako ne ilustruje koncepte bitne za funkcionisanje računara. Mnogo značajnija je podela na osnovu koje računar čine:

- *procesor tj. centralna procesorska jedinica* (engl. *Central Processing Unit, CPU*), koja obrađuje podatke;
- *glavna memorija* (engl. main memory), u kojoj se istovremeno čuvaju i podaci koji se obrađuju i trenutno pokrenuti programi (takođe zapisani binarno, u obliku podataka);
- različiti *periferijski uređaji* ili *ulazno-izlazne jedinice* (engl. *peripherals, input-output devices, IO devices*), kao što su miševi, tastature, ekrani, štampači, diskovi, a koje služe za komunikaciju korisnika sa sistemom i za trajno skladištenje podataka i programa.

Sve nabrojane komponente međusobno su povezane i podaci se tokom rada računara prenose od jedne do druge. Veza između komponenti uspostavlja se hardverskim sklopovima koji se nazivaju *magistrale* (engl. *bus*). Magistrala obuhvata provodnike koji povezuju uređaje, ali i čipove koji kontrolišu protok podataka. Svi periferijski uređaji se sa memorijom, procesorom i magistralama povezuju hardverskim sklopovima koji se nazivaju *kontrolori*. *Matična ploča* (engl. *motherboard*) je štampana ploča na koju se priključuju procesor, memorijski čipovi i svi periferijski uređaji. Na njoj se nalaze čipovi magistrale, a danas i mnogi kontrolori periferijskih uređaja. Osnovu hardvera savremenih računara, dakle, čine sledeće komponente:

Procesori. Procesor je jedna od dve centralne komponente svakog računarskog sistema Fon Nojmanove arhitekture. Svi delovi procesora su danas objedinjeni u zasebnu jedinicu (CPU) realizovanu na pojedinačnom čipu – mikroprocesoru. Procesor se sastoji od *kontrolne jedinice* (engl. *Control Unit*) koja upravlja njegovim radom i *aritmetičko-logičke jedinice* (engl. *Arithmetic Logic Unit*) koja je zadužena za izvođenje aritmetičkih operacija (sabiranje, oduzimanje, množenje, poređenje, ...) i logičkih operacija (konjunkcija, negacija, ...) nad brojevima. Procesor sadrži i određeni, manji broj, *registara* koji privremeno mogu da čuvaju podatke. Registri su obično fiksirane širine (8 bitova, 16 bitova, 32 bita, 64 bita). Komunikacija sa memorijom se ranije vršila isključivo preko specijalizovanog registra koji se nazivao akumulator. Aritmetičko logička jedinica sprovodi operacije nad podacima koji su smešteni u registrima i rezultate ponovo smešta u registre. Kontrolna jedinica procesora čita instrukciju po instrukciju programa zapisanog u memoriji i na osnovu njih određuje sledeću akciju sistema (na primer, izvrši prenos podataka iz procesora na određenu memorijsku adresu, izvrši određenu aritmetičku operaciju nad sadržajem u registrima procesora, uporedi sadržaje dva registra i ukoliko su jednaki izvrši instrukciju koja se nalazi na zadatoj memorijskoj adresi i slično). Brzina procesora meri se u *milionima operacija u sekundi* (engl. *Million Instructions Per Second, MIPS*) tj. pošto su operacije u pokretnom zarezu najzahtevnije, u *broju operacija u pokretnom zarezu u sekundi* (engl. *FLoating Point Operations per Second, FLOPS*). Današnji standardni procesori rade oko 10 GFLOPS (deset milijardi operacija u pokretnom zarezu po sekundi). Današnji procesori mogu da imaju i nekoliko *jezgara* (engl. *core*) koja istovremeno izvršavaju instrukcije i time omogućuju tzv. paralelno izvršavanje.

Elektronska verzija (2024)



Slika 1.1: Shema računara Fon Nojmanove arhitekture

Važne karakteristike procesora danas su broj jezgara (obično 1, 2, 4, 6, 8, pa i više), širina reči (obično 32 bita ili 64 bita) i radni takt (obično nekoliko gigaherca (GHz)) — veći radni takt obično omogućava izvršavanje većeg broja operacija u jedinici vremena.

Za intenzivna računanja sve češće se koriste specijalizovani grafički procesori (engl. *Graphics Processing Unit*, *GPU*). Iako su prvobitno bili namenjeni isključivo za grafičke operacije, kao što su renderovanje 3D scena i obrada piksela, njihova visoko paralelizovana arhitektura omogućava efikasno izvršavanje različitih proračuna nad velikim blokovima podataka. Nasuprot tome, centralni procesori (engl. *Central Processing Unit*, *CPU*) su dizajnirani za efikasno izvođenje sekvencijalnih operacija i optimizovani su za brzo procesiranje jedne ili nekoliko niti (engl. *threads*). Dizajn grafičkih procesora zasnovan je na simetričnoj multiprocesorskoj arhitekturi, podeljenoj na više klastera (engl. *Streaming Multiprocessors*), gde svaki klaster sadrži manji broj jednostavnih jezgara koja dele memoriju i resurse. Glavna razlika između grafičkih i centralnih procesora leži u njihovoj unutrašnjoj arhitekturi: centralni procesori imaju nekoliko složenih jezgara optimizovanih za nisku latenciju, dok grafički procesori imaju stotine ili čak hiljade jednostavnih jezgara koja mogu paralelno izvršavati veliki broj niti. Takva arhitektura omogućava grafičkim procesorima da istovremeno obrađuju veliki broj podataka, što ih čini idealnim za zadatke kao što su grafička obrada, naučne simulacije, kriptografski proračuni, kao i za mašinsko učenje i analizu velikih podataka.

Uloga grafičkih procesora u mašinskom učenju postala je ključna zbog rastuće potrebe za izvođenjem intenzivnih numeričkih operacija u modelima dubokih neuronskih mreža. Neuronske mreže sastoje se od više slojeva sa hiljadama, a ponekad i milionima parametara, koje je potrebno paralelno ažurirati tokom obuke. Grafički procesori su idealni za ove zadatke jer omogućavaju paralelno izvršavanje velikog broja operacija sabiranja i množenja matrica, što značajno smanjuje ukupno vreme potrebno za obuku modela. Popularne softverske biblioteke, kao što su *TensorFlow* i *PyTorch*, koriste grafičke procesore za ubrzanje obuke modela, omogućavajući efikasno paralelno računanje pomoću CUDA (engl. *Compute Unified Device Architecture*) ili OpenCL (engl. *Open Computing Language*).

Kompanija NVIDIA razvila je CUDA, programski jezik koji omogućava programerima da pišu kod koji se efikasno izvršava na grafičkim procesorima. Ključna prednost CUDA jezika je njegova sposobnost da mapira zadatke na hiljade paralelnih jezgara unutar grafičkog procesora, maksimizujući iskorišćenost dostupnih resursa. Programeri koriste CUDA jezik za kreiranje malih funkcija (engl. *kernels*) koje se izvršavaju na svakom jezgru paralelno, što značajno ubrzava proračune. Upotreba CUDA jezika u kontekstu mašinskog učenja omogućila je efikasnu implementaciju algoritama kao što su konvolutivne i rekurentne neuronske mreže, čime su grafički procesori postali ključna komponenta u razvoju sistema zasnovanim na modelima veštačke inteligencije.

OpenCL je otvoreni standard za paralelno programiranje koji omogućava razvoj heterogenih aplikacija na različitim uređajima i platformama, čineći ga pogodnim za heterogeno računarstvo – programer definiše uređaje i resurse za izvršavanje zadataka, koji se mogu distribuirati na različite uređaje unutar iste mreže. Ključna prednost OpenCL-a je njegova portabilnost, jer aplikacije napisane u ovom jeziku mogu raditi na različitim platformama bez potrebe za promenama osnovnog koda.

Veštačka inteligencija postaje ključni faktor u unapređenju hardverskog dizajna, posebno u kontekstu projektovanja i optimizacije mikroprocesora i integrisanih kola [**?**]. U tradicionalnim alatima za automatizaciju elektronskog dizajna (eng. *Electronic Design Automation*, *EDA*), dizajniranje čipova je podrazumevalo niz ručnih koraka, uključujući raspoređivanje komponenti, povezivanje (eng. *routing*), analizu kašnjenja i optimizaciju po-

trošnje energije. Sada, korišćenjem tehnika veštačke inteligencije kao što su neuronske mreže i algoritmi zasnovani na dubokom učenju, ovaj proces postaje znatno efikasniji i automatizovaniji.

Na primer, u sklopu optimizacije performansi, modeli veštačke inteligencije mogu da analiziraju milione mogućih konfiguracija čipa kako bi pronašli optimalnu ravnotežu između potrošnje energije i brzine rada procesora, što je posebno važno za modernu *RISC-V arhitekturu*. RISC-V je otvorena arhitektura koja omogućava fleksibilnost u kreiranju sopstvenih skupova instrukcija, a veštačka inteligencija omogućava inženjerima da brzo isprobavaju različite varijante instruktivnih setova i struktura unutar mikroarhitekture. Pored toga, metode veštačke inteligencije se primenjuju u modernim EDA alatima kao što su Cadence, Synopsys, i pomažu u proračunu kašnjenja, predikciji signala i automatskoj sintezi logičkih kola, omogućavajući dizajnerima da brzo generišu optimalne dizajne koji zadovoljavaju tražene performanse i sigurnosne standarde.

Integracija tehnologija veštačke inteligencije sa EDA alatima ne samo da smanjuje ukupno vreme potrebno za projektovanje, već i smanjuje potrošnju energije, dok istovremeno omogućava razvoj kompleksnih i specifičnih arhitektura koje objedinjavanju više funkcionalnosti u jedinstvenom dizajnu. Modeli veštačke inteligencije predviđaju najbolji način povezivanja komponenti kako bi se minimizovala kašnjenja signala i optimizovala prostorna raspodela unutar čipa, što pomaže u ispunjavanju zahtevnih kriterijuma za stalnim povećanjem performansi.

Kvantni računari predstavljaju revoluciju u svetu računarskih tehnologija, zasnivajući se na potpuno drugačijim principima u odnosu na klasične računare. Dok klasični računari koriste bitove kao osnovnu jedinicu informacija i mogu biti u stanju `0` ili `1`, kvantni računari koriste kvantne bitove ili kubitove (engl. *qubits*). Kubiti se razlikuju od klasičnih bitova po tome što mogu biti u superpoziciji stanja (`0` i `1`), što znači da mogu istovremeno postojati kao kombinacija oba stanja, sa određenim verovatnoćama koje se mogu promeniti tokom operacija zahvaljujući pojavi koja se naziva kvantna superpozicija. Ovaj princip omogućava kvantnim računarima da paralelno obrađuju veliku količinu informacija, eksponencijalno povećavajući njihov računski potencijal.

Pored superpozicije, kvantni računari koriste i kvantnu spregu (engl. *entanglement*), još jedan ključni kvantni fenomen. Kada su dva kubita upletena, stanje jednog kubita direktno zavisi od stanja drugog, bez obzira na fizičku udaljenost između njih. Ove osobine omogućavaju kvantnim računarima da rešavaju probleme koje klasični računari, čak i superkompjuteri, ne mogu rešiti u razumnom vremenskom roku.

Superkompjuteri, koji koriste klasične arhitekture sa hiljadama višejezgarnih CPU i GPU jedinica, oslanjaju se na paralelizam za ubrzanje sekvencijalnih algoritama. Iako su superkompjuteri kao što su Summit i Fugaku sposobni da izvršavaju kvadrilione operacija u sekundi, kvantni računari imaju potencijal da prevaziđu ovu sposobnost za specifične zadatke. Na primer, Šorov algoritam iz kriptografije može faktorisati velike brojeve eksponencijalno brže od najboljih klasičnih algoritama, što znači da bi dovoljno razvijen kvantni računar mogao da razbije današnje standarde kriptografije za nekoliko minuta, dok bi klasičnom računaru trebalo milijarde godina. Dok su klasični superkompjuteri ograničeni Morovim zakonom (ograničenje u broju tranzistora koji se mogu smestiti na jedan čip), kvantni računari nemaju takva ograničenja, što omogućava teoretski neograničeno povećanje performansi sa dodavanjem više kubita.

Međutim, kvantni računari su još uvek u ranoj fazi razvoja. Jedan od najvećih izazova je očuvanje kvantne koherentnosti, odnosno stabilnosti kvantnih stanja tokom vremena, jer su kubiti veoma osetljivi na spoljašnje smetnje. I najmanje promene u okruženju mogu destabilizovati kvantne operacije. Trenutno najmoćniji kvantni procesori imaju desetine do stotine kubita, ali se očekuje da će budući sistemi sadržati hiljade ili čak milione kubita, što će omogućiti izvođenje kompleksnih kvantnih simulacija i proračuna.

Iako današnji kvantni računari ne mogu zameniti superkompjutere za sve vrste problema, oni već nadmašuju klasične računare u specifičnim zadacima. Na primer, kvantne simulacije molekula i materijala mogu precizno modelovati kvantna svojstva hemijskih spojeva, što nije moguće ni sa najmoćnijim klasičnim računarima. Ove simulacije imaju potencijal da ubrzaju razvoj novih lekova, materijala i optimizaciju hemijskih procesa. Kvantni računari se sve više koriste i za optimizacione probleme, poput pronalaženja optimalnih ruta u velikim transportnim mrežama, rešavanja problema pakovanja i modelovanja finansijskih tržišta.

U budućnosti, kvantni računari imaju potencijal da promene kompletno računarsko polje, omogućavajući rešenja za trenutno nerešive probleme u kriptografiji, optimizaciji i simulacijama fizičkih i bioloških sistema. Kombinacija kvantnih i klasičnih računara, poznata kao kvantno–klasični hibridni sistemi, mogla bi postati standard u industriji, gde klasični računari obrađuju sekvencijalne delove algoritma, dok kvantni računari rešavaju najkompleksnije proračune. Ova kombinacija mogla bi da otključa mogućnosti koje danas smatramo naučnom fantastikom, pružajući potpuno novi nivo računarske moći i inovacija.

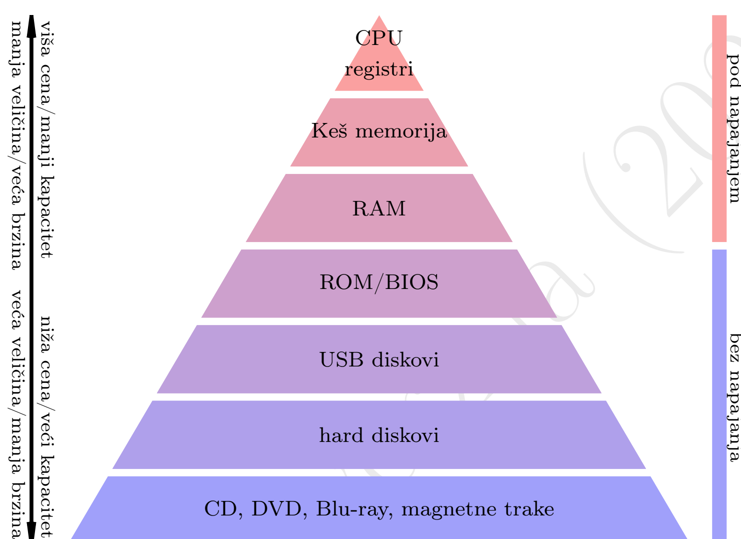
Memorijska hijerarhija. Druga centralna komponenta Fon Nojmanove arhitekture je *glavna memorija* u koju se skladište podaci i programi. Memorija je linearno uređeni niz registara (najčešće bajtova), pri čemu svaki registar ima svoju adresu. Kako se kod ove memorije sadržaju može pristupati u slučajnom redosledu (bez unapred fiksiranog redosleda), ova memorija se često naziva i *memorija sa slobodnim pristupom* (engl. *random*

Elektronska verzija (2024)

access memory, RAM). Osnovni parametri memorija su *kapacitet* (danas obično meren gigabajtima (GB)), *vreme pristupa* koje izražava vreme potrebno da se memorija pripremi za čitanje odnosno upis podataka (danas obično mereno u nanosekundama (ns)), kao i *protok* koji izražava količinu podataka koji se prenose po jedinici merenja (danas obično mereno u GBps).

U savremenim računarskim sistemima, uz glavnu memoriju uspostavlja se čitava *hijerarhija memorija* koje služe da unaprede funkcionisanje sistema. Memorije neposredno vezane za procesor koje se koriste isključivo dok je računar uključen nazivaju se *unutrašnje memorije*, dok se memorije koje se koriste za skladištenje podataka u trenucima kada računar nije uključen nazivaju *spoljne memorije*. Procesor obično nema načina da direktno koristi podatke koji se nalaze u spoljnim memorijama (jer su one znatno sporije od unutrašnjih), već se pre upotrebe svi podaci prebacuju iz spoljnih u unutrašnju memoriju.

Memorijska hijerarhija predstavlja se piramidom. Od njenog vrha ka dnu opadaju kvalitet i brzina memorija, ali zato se smanjuje i cena, pa se kapacitet povećava.



Slika 1.2: Memorijska hijerarhija

Registri procesora predstavljaju najbržu memoriju jer se sve aritmetičke i logičke operacije izvode upravo nad podacima koji se nalaze u njima.

Keš (engl. *cache*) je mala količina brze memorije (nekoliko hiljada puta manjeg kapaciteta od glavne memorije; obično nekoliko megabajta) koja se postavlja između procesora i glavne memorije u cilju ubrzanja rada računara. Keš se uvodi jer su savremeni procesori postali znatno brži od glavnih memorija. Pre pristupa glavnoj memoriji procesor uvek prvo pristupa kešu. Ako traženi podatak tamo postoji, u pitanju je tzv. pogodak keša (engl. cache hit) i podatak se dostavlja procesoru. Ako se podatak ne nalazi u kešu, u pitanju je tzv. promašaj keša (engl. cache miss) i podatak se iz glavne memorije prenosi u keš zajedno sa određenim brojem podataka koji za njim slede (glavni faktor brzine glavne memorije je njeno kašnjenje i praktično je svejedno da li se prenosi jedan ili više podataka jer je vreme prenosa malog broja bajtova mnogo manje od vremena kašnjenja). Motivacija ovog pristupa je u tome što programi često pravilno pristupaju podacima (obično redom kojim su podaci smešteni u memoriji), pa je velika verovatnoća da će se naredni traženi podaci i instrukcije naći u keš-memoriji.

Glavna memorija čuva sve podatke i programe koje procesor izvršava. Mali deo glavne memorije čini ROM (engl. read only memory) – nepromenljiva memorija koja sadrži osnovne programe koji služe za kontrolu određenih komponenata računara (na primer, osnovni ulazno-izlazni sistem BIOS). Znatno veći deo glavne memorije čini RAM – privremena promenljiva memorija sa slobodnim pristupom. Terminološki, podela glavne memorije na ROM i RAM nije najpogodnija jer ove vrste memorije nisu suštinski različite — nepromenljivi deo (ROM) je takođe memorija sa slobodnim pristupom (RAM). Da bi RAM memorija bila što brža, izrađuje se uvek od poluprovodničkih (elektronskih) elemenata. Danas se uglavnom realizuje kao sinhrona dinamička memorija (SDRAM). To znači da se prenos podataka između procesora i memorije vrši u intervalima određenim otkucajima sistemskog sata (često se u jednom otkucaju izvrši nekoliko prenosa). Dinamička memorija je znatno jeftinija i jednostavnija, ali zato sporija od statičke memorije od koje se obično gradi keš.

Spoljne memorije čuvaju podatke trajno – i kada računar ostane bez električnog napajanja. Kao centralna spoljna skladišta podataka uglavnom se koriste *hard diskovi* (engl. *hard disk*) koji čuvaju podatke korišćenjem magnetne tehnologije, a u novije vreme se sve više koriste i *SSD uređaji* (engl. *solid state drive*) koji čuvaju

podatke korišćenjem elektronskih tzv. fleš memorija (engl. flash memory). Kao prenosne spoljne memorije koriste se uglavnom *USB fleš-memorije* (izrađene u sličnoj tehnologiji kao i SSD) i *optički diskovi* (CD, DVD, Blu-ray).

Ulazni uređaji. Osnovni ulazni uređaji današnjih računara su *tastature* i *miševi*. Prenosni računari imaju ugrađenu tastaturu, a umesto miša može se koristiti tzv. *tačped* (engl. *touchpad*). Tastature i miševi se sa računarom povezuju ili kablom (preko PS/2 ili USB priključaka) ili bežično (najčešće korišćenjem BlueTooth veze). Ovo su uglavnom standardizovani uređaji i nema velikih razlika među njima. *Skeneri* sliku sa papira prenose u računar. Princip rada je sličan digitalnom fotografisanju, ali prilagođen slikanju papira.

Izlazni uređaji. Osnovni izlazni uređaji savremenih računara su monitori. Danas dominiraju monitori tankog i ravnog ekrana (engl. flat panel display), zasnovani obično na tehnologiji tečnih kristala (engl. liquid crystal display, LCD) koji su osvetljeni pozadinskim LED osvetljenjem. Ipak, još uvek su ponegde u upotrebi i monitori sa katodnom cevi (engl. cathode ray tube, CRT). Grafički kontrolori koji služe za kontrolu slike koja se prikazuje na monitoru ili projektoru danas su obično integrisani na matičnoj ploči, a ponekad su i na istom čipu sa samim procesorom (engl. Accelerated Processing Unit, APU).

Što se tehnologije štampe tiče, danas su najzastupljeniji laserski štampači i inkdžet štampači (engl. inkjet). Laserski štampači su češće crno-beli, dok su ink-džet štampači obično u boji. Sve su dostupniji i 3D štampači.

1.2 Softver savremenih računara

Softver čine računarski programi i prateći podaci koji određuju izračunavanja koje vrši računar. Na prvim računarima moglo je da se programira samo na *mašinski zavisnim programskim jezicima* — na jezicima specifičnim za konkretnu mašinu na kojoj program treba da se izvršava. Polovinom 1950-ih nastali su prvi *jezici višeg nivoa* i oni su drastično olakšali programiranje. Danas se programi obično pišu u *višim programskim jezicima* a zatim prevode na *mašinski jezik* — jezik razumljiv računaru. Bez obzira na to kako je nastao, da bi mogao da se izvrši na računaru, program mora da budu smešten u memoriju u obliku binarno zapisanih podataka (tj. u obliku niza nula i jedinica) koji opisuju instrukcije koje su neposredno podržane arhitekturom računara. U ovom poglavlju biće prikazani osnovni principa rada računara kroz nekoliko jednostavnih primera programa.

1.2.1 Primeri opisa izračunavanja

Program specifikuje koje operacije treba izvršiti da bi se rešio neki zadatak. Principi rada programa mogu se ilustrovati na primeru nekoliko jednostavnih izračunavanja i instrukcija koje ih opisuju. Ovi opisi izračunavanja dati su u vidu prirodno-jezičkog opisa ali direktno odgovaraju i programima na višim programskim jezicima.

Kao prvi primer, razmotrimo izračunavanje vrednosti $2x + 3$ za datu vrednost x. U programiranju (slično kao i u matematici) podaci se predstavljaju *promenljivama*. Međutim, promenljive u programiranju (za razliku od matematike) vremenom mogu da menjaju svoju vrednost (tada kažemo da im se *dodeljuje nova vrednost*). U programiranju, svakoj promenljivoj pridruženo je (jedno, fiksirano) mesto u memoriji i tokom izvršavanja programa promenljiva može da menja svoju vrednost, tj. sadržaj dodeljenog memorijskog prostora. Ako je promenljiva čija je vrednost ulazni parametar označena sa x, a promenljiva čija je vrednost rezultat izračunavanja označena sa y, onda se pomenuto izračunavanje može opisati sledećim jednostavnim opisom.

```
y := 2*x + 3
```

Simbol `*` označava množenje, `+` sabiranje, a `:=` označava da se promenljivoj sa njene leve strane dodeljuje vrednost izraza sa desne strane.

Kao naredni primer, razmotrimo određivanje većeg od dva data broja. Računari (tj. njihovi procesori) obično imaju instrukcije za poređenje brojeva, ali određivanje vrednosti većeg broja zahteva nekoliko koraka. Pretpostavimo da promenljive x i y sadrže dve brojevne vrednosti, a da promenljiva m treba da dobije vrednost veće od njih. Ovo izračunavanje može da se izrazi sledećim opisom.

```
ako je x >= y onda
    m := x
inače
    m := y
```

Kao malo komplikovaniji primer razmotrimo stepenovanje. Procesori skoro uvek podržavaju instrukcije kojima se izračunava zbir i proizvod dva cela broja, ali stepenovanje obično nije podržano kao elementarna operacija. Složenije operacije se mogu ostvariti korišćenjem jednostavnijih. Na primer, n-ti stepen broja x (tj. vrednost

Elektronska verzija (2024)

x^n) moguće je izračunati uzastopnom primenom množenja: ako se krene od broja 1 i n puta sa pomnoži brojem x, rezultat će biti x^n. Da bi moglo da se osigura da će množenje biti izvršeno tačno n puta, koristi se brojačka promenljiva i koja na početku dobija vrednost 0, a zatim se, prilikom svakog množenja, uvećava sve dok ne dostigne vrednost n. Ovaj postupak možemo predstaviti sledećim opisom.

```
s := 1, i := 0
dok je i < n radi sledeće:
    s := s·x, i := i+1
```

Kada se ovaj postupak primeni na vrednosti $x = 3$ i $n = 2$, izvodi se naredni niz koraka.

`s := 1,`	`i := 0,`	pošto je `i(=0)` manje od `n(=2)`, vrše se dalje operacije
`s := s·x = 1·3 = 3,`	`i := i+1 = 0+1 = 1,`	pošto je `i(=1)` manje od `n(=2)`, vrše se dalje operacije
`s := s·x = 3·3 = 9,`	`i := i+1 = 1+1 = 2,`	pošto `i(=2)` nije manje od `n(=2)`, ne vrše se dalje operacije.

1.2.2 Mašinski programi

Mašinski programi su neposredno vezani za procesor računara na kojem se koriste — procesor je konstruisan tako da može da izvršava određene elementarne naredbe. Ipak, razvoj najvećeg broja procesora usmeren je tako da se isti mašinski programi mogu koristiti na čitavim familijama procesora.

Primitivne instrukcije koje podržava procesor su veoma malobrojne i jednostavne (na primer, postoje samo instrukcije za sabiranje dva broja, konjunkcija bitova, instrukcija skoka i slično) i nije lako kompleksne i apstraktne algoritme izraziti korišćenjem tog uskog skupa elementarnih instrukcija. Ipak, svi zadaci koje računari izvršavaju svode se na ove primitivne instrukcije.

Asemblerski jezici. Asemblerski (ili simbolički) jezici su jezici koji su veoma bliski mašinskom jeziku računara, ali se, umesto korišćenja binarnog sadržaja za zapisivanje instrukcija koriste (mnemotehničke, lako pamtljive) simboličke oznake instrukcija (tj. programi se unose kao tekst). Ovim se, tehnički, olakšava unos programa i programiranje (programer ne mora da direktno manipuliše binarnim sadržajem), pri čemu su sve mane mašinski zavisnog programiranja i dalje prisutne. Kako bi ovako napisan program mogao da se izvršava, neophodno je izvršiti njegovo prevođenje na mašinski jezik (tj. zapisati instrukcije binarnom azbukom) i uneti na odgovarajuće mesto u memoriji. Ovo prevođenje je jednostavno i jednoznačno i vrše ga jezički procesori koji se nazivaju *asembleri*.

Sva izračunavanja u primerima iz poglavlja 1.2.1 su opisana neformalno, kao uputstva čoveku a ne računaru. Da bi se ovako opisana izračunavanja mogla sprovesti na nekom računaru Fon Nojmanove arhitekture neophodno je opisati ih preciznije. Svaka elementarna operacija koju procesor može da izvrši u okviru programa zadaje se *procesorskom instrukcijom* — svaka instrukcija instruiše procesor da izvrši određenu operaciju. Svaki procesor podržava unapred fiksiran, konačan *skup instrukcija* (engl. *instruction set*). Svaki program računara predstavljen je nizom instrukcija i skladišti se u memoriji računara. Naravno, računari se razlikuju (na primer, po tome koliko registara u procesoru imaju, koje instrukcije može da izvrši njihova aritmetičko-logička jedinica, koliko memorije postoji na računaru, itd). Međutim, da bi se objasnili osnovni principi rada računara nije neophodno razmatrati neki konkretan računar, već se može razmatrati neki hipotetički računar. Pretpostavimo da procesor sadrži tri registra označena sa `ax`, `bx` i `cx` i još nekoliko izdvojenih bitova (tzv. zastavica). Dalje, pretpostavimo da procesor može da izvršava naredne *aritmetičke instrukcije* (zapisane ovde u asemblerskom obliku):

- Instrukcija `add ax, bx` označava operaciju sabiranja vrednosti brojeva koji se nalaze u registrima `ax` i `bx`, pri čemu se rezultat sabiranja smešta u registar `ax`. Operacija `add` može se primeniti na bilo koja dva registra.
- Instrukcija `mul ax, bx` označava operaciju množenja vrednosti brojeva koji se nalaze u registrima `ax` i `bx`, pri čemu se rezultat množenja smešta u registar `ax`. Operacija `mul` može se primeniti na bilo koja dva registra.

Elektronska verzija (2024)

- Instrukcija `cmp ax, bx` označava operaciju poređenja vrednosti brojeva koji se nalaze u registrima `ax` i `bx` i rezultat pamti postavljanjem zastavice u procesoru. Operacija `cmp` se može primeniti na bilo koja dva registra.

Program računara je niz instrukcija koje se obično izvršavaju redom, jedna za drugom. Međutim, pošto se javlja potreba da se neke instrukcije ponove veći broj puta ili da se određene instrukcije preskoče, uvode se *instrukcije skoka*. Da bi se moglo specifikovati na koju instrukciju se vrši skok, uvode se *labele* – označena mesta u programu. Pretpostavimo da naš procesor može da izvršava sledeće dve vrste skokova (bezuslovne i uslovne):

- Instrukcija `jmp label`, gde je `label` neka labela u programu, označava bezuslovni skok koji uzrokuje nastavak izvršavanja programa od mesta u programu označenog navedenom labelom.
- Uslovni skokovi prouzrokuju nastavak izvršavanja programa od instrukcije označene navedenom labelom, ali samo ako je neki uslov ispunjen. Ukoliko uslov nije ispunjen, izvršava se naredna instrukcija. U nastavku će se razmatrati samo instrukcija `jge label`, koja uzrokuje uslovni skok na mesto označeno labelom `label` ukoliko je vrednost prethodnog poređenja brojeva bila *veće ili jednako*.

Tokom izvršavanja programa, podaci se nalaze u memoriji i u registrima procesora. S obzirom na to da procesor sve operacije može da izvrši isključivo nad podacima koji se nalaze u njegovim registrima, svaki procesor podržava i *instrukcije prenosa podataka* između memorije i registara procesora (kao i između samih registara). Pretpostavimo da naš procesor podržava sledeću instrukciju ove vrste.

- Instrukcija `mov` označava operaciju prenosa podataka i ima dva parametra — prvi određuje gde se podaci prenose, a drugi koji određuje koji se podaci prenose. Parametar može biti ime registra (što označava da se pristupa podacima u određenom registru), broj u zagradama (što označava da se pristupa podacima u memoriji i to na adresi određenoj brojem u zagradama) ili samo broj (što označava da je podatak baš taj navedeni broj). Na primer, instrukcija `mov ax bx` označava da se sadržaj registra `bx` prepisuje u registar `ax`, instrukcija `mov ax, [10]` označava da se sadržaj iz memorije sa adrese `10` prepisuje u registar `ax`, instrukcija `mov ax, 1` označava da se u registar ax upisuje vrednost `1`, dok instrukcija označava `mov [10], ax` da se sadržaj registra `ax` upisuje u memoriju na adresu `10`.

Sa ovakvim procesorom na raspolaganju, izračunavanje vrednosti $2x + 3$ može se ostvariti na sledeći način. Pretpostavimo da se ulazni podatak (broj x) nalazi u glavnoj memoriji i to na adresi `10`, a da rezultat y treba smestiti na adresu `11` (ovo su sasvim proizvoljno odabrane adrese). Izračunavanje se onda može opisati sledećim programom (nizom instrukcija).

```
mov ax, [10]
mov bx, 2
mul ax, bx
mov bx, 3
add ax, bx
mov [11], ax
```

Instrukcija `mov ax, [10]` prepisuje vrednost promenljive x (iz memorije sa adrese `10`) u registar `ax`. Instrukcija `mov bx, 2` upisuje vrednost `2` u registar `bx`. Nakon instrukcije `mul ax, bx` vrši se množenje i registar `ax` sadrži vrednost $2x$. Instrukcija `mov bx, 3` upisuje vrednost `3` u registar `bx`, nakon instrukcije `add ax, bx` se vrši sabiranje i u registru `ax` se nalazi tražena vrednost $2x + 3$. Na kraju se ta vrednost instrukcijom `mov [11], ax` upisuje u memoriju na adresu `11`.

Određivanje većeg od dva broja može se ostvariti na sledeći način. Pretpostavimo da se ulazni podaci nalaze u glavnoj memoriji i to broj x na adresi `10`, broj y na adresi `11`, dok rezultat m treba smestiti na adresu `12`. Program (niz instrukcija) kojima može da se odredi maksimum je sledeći:

```
mov ax, [10]
mov bx, [11]
cmp ax, bx
jge vecix
mov [12], bx
jmp kraj
vecix:
```

Elektronska verzija (2024)

```
mov[12], ax
kraj:
```

Nakon prenosa vrednosti oba broja u registre procesora (instrukcijama `mov ax, [10]` i `mov bx, [11]`), vrši se njihovo poređenje (instrukcija `cmp ax, bx`). Ukoliko je broj x veći od ili jednak broju y prelazi se na mesto označeno labelom `vecix` (instrukcijom `jge vecix`) i na mesto rezultata upisuje se vrednost promenljive x (instrukcijom `mov[12], ax`). Ukoliko uslov skoka `jge` nije ispunjen (ako x nije veće ili jednako y), na mesto rezultata upisuje se vrednost promenljive y (instrukcijom `mov [12], bx`) i bezuslovno se skače na kraj programa (instrukcijom `jmp kraj`) (da bi se preskočilo izvršavanje instrukcije koja na mesto rezultata upisuje vrednost promenljive x).

Izračunavanje stepena može se ostvariti na sledeći način. Pretpostavimo da se ulazni podaci nalaze u glavnoj memoriji i to broj x na adresi `10`, a broj n na adresi `11`, i da konačan rezultat treba da bude smešten u memoriju i to na adresu `12`. Pretpostavimo da će pomoćne promenljive s i i koje se koriste u postupku biti smeštene sve vreme u procesoru, i to promenljiva s u registru `ax`, a promenljiva i u registru `bx`. Pošto postoji još samo jedan registar (`cx`), u njega će naizmenično biti smeštane vrednosti promenljivih n i x, kao i konstanta 1 koja se sabira sa promenljivom i. Niz instrukcija kojim opisani hipotetički računar može da izračuna stepen je sledeći:

```
  mov ax, 1
  mov bx, 0
petlja:
  mov cx, [11]
  cmp bx, cx
  jge kraj
  mov cx, [10]
  mul ax, cx
  mov cx, 1
  add bx, cx
  jmp petlja
kraj:
  mov [12], ax
```

Ilustrujmo izvršavanje ovog programa na izračunavanju vrednosti 3^2. Inicijalna konfiguracija je takva da se na adresi `10` u memoriji nalazi vrednost $x = 3$, na adresi `11` vrednost $n = 2$. Početna konfiguracija (tj. vrednosti memorijskih lokacija i registara) može da se predstavi na sledeći način:

```
10: 3    ax: ?
11: 2    bx: ?
12: ?    cx: ?
```

Nakon izvršavanja prve dve instrukcije (`mov ax, 1` i `mov bx, 0`), postavlja se vrednost registara `ax` i `bx` i prelazi se u sledeću konfiguraciju:

```
10: 3    ax: 1
11: 2    bx: 0
12: ?    cx: ?
```

Sledeća instrukcija (`mov cx, [11]`) kopira vrednost `2` sa adrese `11` u registar `cx`:

```
10: 3    ax: 1
11: 2    bx: 0
12: ?    cx: 2
```

Vrši se poređenje sa registrom `bx` (`cmp bx, cx`) i kako uslov skoka (`jge kraj`) nije ispunjen (vrednost `0` u `bx` nije veća ili jednaka od vrednosti `2` u `cx`), nastavlja se dalje. Nakon kopiranja vrednosti `3` sa adrese `10` u registar `cx` (instrukcijom `mov cx, [10]`), vrši se množenje vrednosti u registrima `ax` i `cx` (instrukcijom `mul ax, cx`) i dolazi se u sledeću konfiguraciju:

```
10: 3    ax: 3
11: 2    bx: 0
12: ?    cx: 3
```

Nakon toga, u `cx` se upisuje `1` (instrukcijom `mov cx, 1`) i vrši se sabiranje vrednosti registara `bx` i `cx` (instrukcijom `add bx, cx`) čime se vrednost u registru `bx` uvećava za `1`.

```
10: 3    ax: 3
11: 2    bx: 1
12: ?    cx: 1
```

Bezuslovni skok (`jmp petlja`) ponovo vraća kontrolu na početak petlje, nakon čega se u `cx` opet prepisuje vrednost `2` sa adrese `11` (`mov cx, [11]`). Vrši se poređenje sa registrom `bx` (`cmp bx, cx`) i kako uslov skoka (`jge kraj`) nije ispunjen (vrednost `1` u `bx` nije veća ili jednaka vrednosti `2` u `cx`), nastavlja se dalje. Nakon još jednog množenja i sabiranja dolazi se do konfiguracije:

```
10: 3    ax: 9
11: 2    bx: 2
12: ?    cx: 1
```

Bezuslovni skok ponovo vraća kontrolu na početak petlje, nakon čega se u `cx` opet prepisuje vrednost `2` sa adrese `11`. Vrši se poređenje sa registrom `bx`, no, ovaj put je uslov skoka ispunjen (vrednost `2` u `bx` je veća ili jednaka vrednosti `2` u `cx`) i skače se na mesto označeno labelom `kraj`, gde se poslednjom instrukcijom (`mov [12], ax`) konačna vrednost iz registra `ax` kopira u memoriju na dogovorenu adresu `12`, čime se stiže u završnu konfiguraciju:

```
10: 3     ax: 9
11: 2     bx: 2
12: 9     cx: 1
```

Mašinski jezik. Fon Nojmanova arhitektura podrazumeva da se i sam program (niz instrukcija) nalazi u glavnoj memoriji prilikom njegovog izvršavanja. Potrebno je svaki program (poput tri navedena) predstaviti nizom nula i jedinica, na način „razumljiv" procesoru — na mašinskom jeziku. Na primer, moguće je da su binarni kodovi za instrukcije uvedeni na sledeći način:

`mov`	`001`
`add`	`010`
`mul`	`011`
`cmp`	`100`
`jge`	`101`
`jmp`	`110`

Takođe, pošto neke instrukcije primaju podatke različite vrste (neposredno navedeni brojevi, registri, apsolutne memorijske adrese), uvedeni su posebni kodovi za svaki od različitih vidova adresiranja. Na primer:

neposredno	`00`
registarsko	`01`
apsolutno	`10`

Pretpostavimo da registar `ax` ima oznaku `00`, registar `bx` ima oznaku `01`, a registar `cx` oznaku `10`. Pretpostavimo i da su sve adrese osmobitne. Pod navedenim pretpostavkama, instrukcija `mov [10], ax` se, u ovom hipotetičkom mašinskom jeziku, može kodirati kao `001 10 01 00010000 00`. Kôd `001` dolazi od instrukcije `mov`, zatim slede `10` i `01` koji ukazuju da prvi argument predstavlja memorijsku adresu, a drugi oznaku registra, za čim sledi memorijska adresa $(10)_{16}$ binarno kodirana sa `00010000` i na kraju oznaka `00` registra `ax`. Na sličan način, celokupan prikazani mašinski kôd navedenog asemblerskog programa koji izračunava $2x + 3$ je moguće binarno kodirati kao:

```
001 01 10 00 00010000   // mov ax, [10]
001 01 00 01 00000010   // mov bx, 2
011 00 01               // mul ax, bx
001 01 00 01 00000011   // mov bx, 3
010 00 01               // add ax, bx
001 10 01 00010001 00   // mov [11], ax
```

Između prikazanog asemblerskog i mašinskog programa postoji veoma direktna i jednoznačna korespondencija (u oba smera) tj. na osnovu datog mašinskog koda moguće je jednoznačno rekonstruisati asemblerski kôd.

Specifični hardver koji čini kontrolnu jedinicu procesora dekodira jednu po jednu instrukciju i izvršava akciju zadatu tom instrukcijom. Kod realnih procesora, broj instrukcija i načini adresiranja su mnogo bogatiji a prilikom pisanja programa potrebno je uzeti u obzir mnoge aspekte na koje se u navedenim jednostavnim primerima nije obraćala pažnja. Ipak, mašinske i asemblerske instrukcije stvarnih procesora veoma su slične navedenim hipotetičkim instrukcijama.

1.2.3 Klasifikacija savremenog softvera

Računarski programi veoma su složeni. Hardver računara sačinjen je od elektronskih kola koja mogu da izvrše samo elementarne operacije i, da bi računar mogao da obavi i najjednostavniji zadatak zanimljiv korisniku, neophodno je da se taj zadatak razloži na mnoštvo elementarnih operacija. Napredak računara ne bi bio moguć ako bi programeri morali svaki program da opisuju i razlažu do krajnjeg nivoa elementarnih instrukcija. Zato je poželjno da programeri naredbe računaru mogu zadavati na što apstraktnijem nivou. Računarski sistemi i softver se grade slojevito i svaki naredni sloj oslanja se na funkcionalnost koju mu nudi sloj ispod njega. U skladu sa tim, softver savremenih računara se obično deli na *sistemski* i *aplikativni*. Osnovni zadatak sistemskog softvera je da posreduje između hardvera i aplikativnog softvera koji krajnji korisnici koriste. Granica između sistemskog i aplikativnog softvera nije kruta i postoje programi za koje se može smatrati da pripadaju obema grupama (na primer, editori teksta).

Sistemski softver. Sistemski softver je softver čija je uloga da kontroliše hardver i pruža usluge aplikativnom softveru. Najznačajniji skup sistemskog softvera, danas prisutan na skoro svim računarima, čini *operativni sistem* (OS). Pored OS, sistemski softver sačinjavaju i različiti *uslužni programi*: editori teksta, alat za programiranje (prevodioci, dibageri, profajleri, integrisana okruženja) i slično.

Korisnici OS često identifikuju sa izgledom ekrana tj. sa programom koji koriste da bi pokrenuli svoje aplikacije i organizovali dokumente. Međutim, ovaj deo sistema koji se naziva *korisnički interfejs* (engl. *user interface – UI*) ili *školjka* (engl. *shell*) samo je tanak sloj na vrhu operativnog sistema i OS je mnogo više od onoga što krajnji korisnici vide. Najveći i najznačajni deo OS naziva se *jezgro* (engl. *kernel*). Osim što kontroliše i apstrahuje hardver, operativni sistem tj. njegovo jezgro sinhronizuje rad više programa, raspoređuje procesorsko vreme i memoriju, brine o sistemu datoteka na spoljašnjim memorijama itd. Najznačanjni operativni sistemi danas su Microsoft Windows, sistemi zasnovani na Linux jezgru (na primer, Ubuntu, RedHat, Fedora, Suse) i Mac OS X.

OS upravlja svim resursima računara (procesorom, memorijom, periferijskim uređajima) i stavlja ih na raspolaganje aplikativnim programima. OS je u veoma tesnoj vezi sa hardverom računara i veliki deo zadataka se izrvršava uz direktnu podršku specijalizovanog hardvera namenjenog isključivo izvršavanju OS. Nekada se hardver i operativni sistem smatraju jedinstvenom celinom i umesto podele na hardver i softver razmatra se podela na sistem (hardver i OS) i na aplikativni softver.

Aplikativni softver. Aplikativni softver je softver koji krajnji korisnici računara direktno koriste u svojim svakodnevnim aktivnostima. To su pre svega pregledači Veba, zatim klijenti elektronske pošte, kancelarijski softver (programi za kucanje teksta, izradu slajd-prezentacija, tabelarna izračunavanja), video igre, multimedijalni softver (programi za reprodukciju i obradu slika, zvuka i video-sadržaja) itd.

Programer ne bi trebalo da misli o konkretnim detaljima hardvera, tj. poželjno je da postoji određena *apstrakcija hardvera*. Na primer, mnogo je pogodnije ako programer umesto da mora da kaže „Neka se zavrti ploča diska, neka se glava pozicionira na određenu poziciju, neka se zatim tu upiše određeni bajt itd." može da kaže „Neka se u datu datoteku na disku upiše određeni tekst". OS je taj koji se brine o svim detaljima, dok se programer (tačnije, aplikacije koje on isprogramira), kada god mu je potrebno obraća sistemu da mu tu uslugu pruži. Konkretni detalji hardvera poznati su u okviru operativnog sistema i komande koje programer zadaje izvršavaju se uzimajući u obzir ove specifičnosti. Operativni sistem, dakle, programeru pruža skup funkcija koje on može da koristi da bi postigao željenu funkcionalnost hardvera, sakrivajući pritom konkretne hardverske detalje. Ovaj skup funkcija naziva se *programski interfejs za pisanje aplikacija*[1] (engl. *Aplication Programming Interface, API*). Funkcije se nazivaju i *sistemski pozivi* (jer se OS poziva da izvrši određeni zadatak). Programer nema mogućnost direktnog pristupa hardveru i jedini način da se pristupi hardveru je preko sistemskih poziva. Ovim se osigurava određena bezbednost celog sistema.

Postoji više nivoa na kojima se može realizovati neka funkcionalnost. Programer aplikacije je na vrhu hijerarhije i on može da koristi funkcionalnost koju mu pruža programski jezik koji koristi i *biblioteke* tog jezika. Izvršivi programi često koriste funkcionalnost specijalne *rantajm biblioteke* (engl. *runtime library*) koja koristi

[1] Ovaj termin se ne koristi samo u okviru operativnih sistema, već i u širem kontekstu, da označi skup funkcija kroz koji jedan programski sistem koristi drugi programski sistem.

Elektronska verzija (2024)

funkcionalnost operativnog sistema (preko sistemskih poziva), a zatim operativni sistem koristi funkcionalnost samog hardvera.

Pitanja i zadaci za vežbu

Pitanje 1.1. *Nabrojati osnovne periode u razvoju računara i navesti njihove osnovne karakteristike i predstavnike.*

Pitanje 1.2. *Ko je i u kom veku konstruisao prvu mehaničku spravu na kojoj je bilo moguće sabirati prirodne brojeve, a ko je i u kom veku konstruisao prvu mehaničku spravu na kojoj je bilo moguće sabirati i množiti prirodne brojeve?*

Pitanje 1.3. *Kojoj spravi koja se koristi u današnjem svetu najviše odgovaraju Paskalove i Lajbnicove sprave?*

Pitanje 1.4. *Kakva je veza između tkačkih razboja i računara s početka XIX veka?*

Pitanje 1.5. *Koji je značaj Čarlsa Bebidža za razvoj računarstva i programiranja? U kom veku je on dizajnirao svoje računske mašine? Kako se one zovu i koja od njih je trebalo da bude programabilna? Ko se smatra prvim programerom?*

Pitanje 1.6. *Na koji način je Herman Holerit doprineo izvršavanju popisa stanovnika u SAD 1890? Kako su bili čuvani podaci sa tog popisa? Koja čuvena kompanija je nastala iz kompanije koju je Holerit osnovao?*

Pitanje 1.7. *Kada su nastali prvi elektronski računari? Nabrojati nekoliko najznačajnijih.*

Pitanje 1.8. *Na koji način je programiran računar ENIAC, a na koji računar EDVAC?*

Pitanje 1.9. *Koje su osnovne komponente računara Fon Nojmanove arhitekture? Šta se skladišti u memoriju računara Fon Nojmanove arhitekture? Gde se vrši obrada podataka u okviru računara Fon Nojmanove arhitekture? Od kada su računari zasnovani na Fon Nojmanovoj arhitekturi?*

Pitanje 1.10. *Šta su to računari sa skladištenim programom? Šta je to hardver a šta softver?*

Pitanje 1.11. *Šta su procesorske instrukcije? Navesti nekoliko primera.*

Pitanje 1.12. *Koji su uobičajeni delovi procesora? Da li se u okviru samog procesora nalazi određena količina memorije za smeštanje podataka? Kako se ona naziva?*

Pitanje 1.13. *Ukratko opisati osnovne elektronske komponente svake generacije računara savremenih elektronskih računara? Šta su bile osnovne elektronske komponente prve generacije elektronskih računara? Od koje generacije računara se koriste mikroprocesori? Koji tipovi računara se koriste u okviru III generacije?*

Pitanje 1.14. *U kojoj deceniji dolazi do pojave računara za kućnu upotrebu? Koji je najprodavaniji model kompanije Commodore? Da li je IBM proizvodio računare za kućnu upotrebu? Koji komercijalni kućni računar prvi uvodi grafički korisnički interfejs i miša?*

Pitanje 1.15. *Koja serija Intelovih procesora je bila dominantna u PC računarima 1980-ih i 1990-ih godina?*

Pitanje 1.16. *Šta je to tehnološka konvergencija? Šta su to tableti, a šta „pametni telefoni“?*

Pitanje 1.17. *Koje su osnovne komponente savremenog računara? Šta je memorijska hijerarhija? Zašto se uvodi keš-memorija? Koje su danas najkorišćenije spoljne memorije?*

Pitanje 1.18. *U koju grupu jezika spadaju mašinski jezici i asemblerski jezici?*

Pitanje 1.19. *Da li je kôd na nekom mašinskom jeziku prenosiv sa jednog na sve druge računare? Da li asembler zavisi od mašine na kojoj se koristi?*

Pitanje 1.20. *Ukoliko je raspoloživ asemblerski kôd nekog programa, da li je moguće jednoznačno konstruisati odgovarajući mašinski kôd? Ukoliko je raspoloživ mašinski kôd nekog programa, da li je moguće jednoznačno konstruisati odgovarajući asemblerski kôd?*

Zadatak 1.1. *Na opisanom asemblerskom jeziku opisati izračunavanje vrednosti izraza* `x := x*y + y + 3`. *Generisati i mašinski kôd za napisani program.*

Zadatak 1.2. *Na opisanom asemblerskom jeziku opisati izračunavanje:*

```
ako je (x < 0)
   y := 3*x;
inace
   x := 3*y;
```

Zadatak 1.3. *Na opisanom asemblerskom jeziku opisati izračunavanje:*

```
dok je (x <= 0) radi
   x := x + 2*y + 3;
```

Zadatak 1.4. *Na opisanom asemblerskom jeziku opisati izračunavanje kojim se izračunava* $[\sqrt{x}]$*, pri čemu se* x *nalazi na adresi 100, a rezultat smešta na adresu 200.* $\surd$

Pitanje 1.21. *Koji su osnovni razlozi slojevite organizacije softvera? Šta je sistemski, a šta aplikativni softver?*

Pitanje 1.22. *Koji su osnovni zadaci operativnog sistema? Šta su sistemski pozivi? Koji su operativni sistemi danas najkorišćeniji?*

Elektronska verzija (20