Programiranje 1
2. predavanje

Saša Singer

singer@math.hr
web.math.pmf.unizg.hr/~singer

PMF – Matematički odsjek, Zagreb

Sadržaj predavanja

- Uvod u algoritme:
  - Pojam algoritma.
  - Primjeri algoritama.
  - Opis algoritma.
  - Zapis i izvršavanje algoritama.
  - Osnovna svojstva algoritma.
- Principi rada računala:
  - Što je računalo?
  - Instrukcije — operacije i podaci.
  - Osnovni dijelovi računala — ulaz, izlaz, memorija.
  - Izvršni dio računala — procesor.
  - Von Neumannov model računala.

Sadržaj predavanja (nastavak)

- Građa računala:
  - Memorija — bistabil, bit, byte, riječ.
  - Procesor — registri, aritmetičko–logička jedinica, upravljačka jedinica.
  - Ulazna jedinica, izlazna jedinica.

Informacije — odrada

Termini odrade dva predavanja: 18. 1. (ili 21. 12., ovisno o dogovoru) i 25. 1. 2019., su:

- subota, 13. 10., od 12–14 u (003),
- subota, 20. 10., od 10–12 u (003).

Informacije — demonstratori za Prog1 i Prog2

Imamo 6 demonstratora za Prog (ali još nisu službeni). To su:

- Al Depope
- Božidar Grgur Drmić
- Daniela Ivanković
- Sanjin Jurić Fot
- Ena Škopelja
- Patrik Vasung

Najavite se demosima koji dan ranije, na pr. e-mailom.

- Njihove termine i e-mail adrese nađete na službenoj web-stranici kolegija (čim dogovore termine).
- Postavite miša na ime, u prozoru kliknite na E-mail.

Informacije — dodaci ovom predavanju

Na webu imate dva dodatka ovom predavanju:

- "Matematički" model računala — Turingov stroj,
- Stvarni "izgled" računala, s naglaskom na hijerarhijsku građu memorije u modernim računalima (tzv. "cache").

Nije obavezno, ali isplati se pogledati — za "opću kulturu".

Uvod u algoritme

Sadržaj

- Uvod u algoritme:
  - Pojam algoritma.
  - Primjeri algoritama.
  - Opis algoritma.
  - Zapis i izvršavanje algoritama.
  - Osnovna svojstva algoritma.

Pojam algoritma

Što je algoritam? Grubo rečeno:

- Algoritam = metoda, postupak, pravilo za rješenje nekog problema ili dostizanje nekog cilja.

Ovo nije precizna definicija u matematičkom smislu, već samo opis preko drugih, sličnih pojmova, pri čemu je "postupak" najbliži.

- Postupak asocira na konačan niz koraka koje treba napraviti za rješenje nekog problema.
- Metoda se kao izraz često koristi u matematici, ali, obično, uključuje i tzv. beskonačne "postupke" — koji tek na limesu daju rješenje (v. matematička analiza, numerička matematika).

Pojam algoritma (nastavak)

Zašto je bitno znati što je algoritam?!

- Osnovna zadaća: razvoj efikasnih i točnih algoritama.
- Intuitivno je jasno da efikasno znači brzo, a točno da je rješenje blizu "pravom" rješenju. Detaljnije — poslije.

Postoji i precizna matematička definicija pojma algoritam, ali ona nije sasvim jednostavna — potrebna je onima koji će se baviti svojstvima algoritama.

Budući da je cilj kolegija

- naučiti koristiti algoritme za rješavanje raznih problema,

dovoljno je (umjesto definicije)

- opisati osnovna svojstva algoritama.

Primjer algoritma 1

Sva moderna tehnička pomagala imaju upute za uporabu, korištenje, rukovanje, instalaciju, ... ili, kako vam drago.

Primjer. Dojadilo mi je slušanje zaštićenih glazbenih CD-ova na računalu, jer:

- prvo se javi neki odurni "player",
- a nakon toga počne glazba kôdirana u MP3 formatu nevjerojatne nekvalitete od 56 KB u sekundi.

Prvo rješenje: tehničko i nije za javno predavanje :-)

Drugo rješenje: kupio sam priglupu glazbenu CD liniju! Inače se može dogoditi sličan problem kao i na računalu — da stvar ne radi. Recimo, takvi su CD-playeri za automobile.

Primjer algoritma 1 (nastavak)

I gdje je tu algoritam?

Što dolazi uz glazbenu liniju?

- Upute = kako pospajati sve razne dijelove, kablove i ostalo u funkcionalno radeću stvar!

Dakle, dotične upute su:

- algoritam za postizanje cilja = kako iz hrpe dijelova sastaviti radeću glazbenu liniju.

One "druge" upute — upute za uporabu, imaju smisla tek kad završimo s ovima za "instalaciju"!

Primjer algoritma 1 (nastavak)

To može izgledati ovako:

- Kabel A (slijedi sličica) treba izvaditi iz vrećice,
- utaknuti otraga u CD jedinicu u rupu B (vidi sličicu),
- pažljivo stisnuti konektore,
- zavinuti kabel,
- prisloniti ga na stražnju površinu CD-jedinice
- i na kraju ga (ipak) utaknuti u rupu C na pojačalu (nova sličica).
- ...

Sad se sličan postupak ponavlja jedno desetak puta!
Za sat–dva, možda linija i proradi!

Još primjera algoritama

Standardni primjeri algoritama:

- kulinarski recepti (pripravljanje jela),
  - pita od jabuka, muffini, ... (v. prilog pod pauzom),
- recepti za pripravu pića i koktela,
  - "bijeli medvjed":

    1/2 ohlađenog pjenušca (šampanjca),
    1/2 ohlađene votke (dobre, bez metila),
    "uštrcak" maraschina,
    i oprez ... (upozorio sam vas ...),
- uporaba bilo kakvih aparata (izbor operacija, ...),
- rješavanje matematičkih zadataka!

Opis algoritma

Općenito, kako moraju izgledati upute i iz čega se sastoje?

- Opći izgled je nešto poput:

  1. korak
  2. korak
  ⋮
  zadnji korak

Drugim riječima, algoritam se sastoji iz niza koraka koje treba izvršiti da bi se postigao cilj, odnosno, riješio problem.

Tih koraka, naravno, ima konačno mnogo.

- Svaki pojedini korak algoritma je instrukcija ili naredba (može i akcija) koju treba napraviti (izvršiti).

Zapis i izvršavanje algoritama

Uobičajeno, instrukcije se

- pišu jedna ispod druge i
- izvršavaju tim redom kojim su napisane.

Ova "očita" pravila vrijede i u programiranju!

Ipak, postoje i instrukcije koje

- mijenjaju standardni redoslijed izvršavanja.

Primjeri:

- ako se dogodi "to i to", onda otiđi na korak "taj i taj", ili
- ponovi "neke korake" sve dok je ispunjen uvjet "taj i taj", ili
- ponovi "neke korake" određeni broj puta.

Primjer — zbrajanje prirodnih brojeva

Postupak zbrajanja prirodnih brojeva "na ruke" (dekadski):

- napisati brojeve jedan ispod drugog, tako da znamenke iste težine budu u istom stupcu, tj. poravnato desno;
- sa zbrajanjem se započinje s desne strane;
- zbrajaju se znamenke u istom stupcu i zbroju se pribraja eventualni prijenos; prijenos je na početku jednak nuli;
- ako je zbroj znamenki u stupcu ≤ 9, onda ga zapisujemo; u protivnom, zapisujemo znamenku jedinica, a znamenka desetica postaje prijenos;
- zatim prelazimo na sljedeći stupac ulijevo ($\leftarrow$);
- postupak se ponavlja sve dok ne zbrojimo znamenke u krajnjem lijevom stupcu i zapišemo taj zbroj (i prijenos).

Vrste instrukcija

Grubo govoreći, postoje dvije vrste instrukcija za kontrolu redoslijeda operacija, tj. izvršavanja ostalih instrukcija:

- Uvjetne = izbor jedne od mogućih alternativa.
- Petlje = ponavljanje nekog bloka naredbi pod kontrolom uvjeta ili brojača.

Ovakve instrukcije su ključne u programiranju.

Razlog?

Bez petlji, programiranje nema smisla. Poanta:

- Program je "kratak", a izvršavanje "traje", tj. dijelovi programa se ponavljaju puno puta.
  (Računalo je "glupo, ali brzo".)

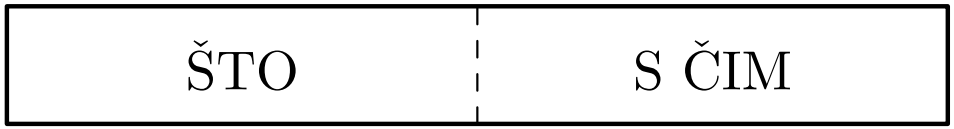
Izgled instrukcije

Sve instrukcije imaju sličan oblik i sastoje se iz 2 dijela:

- što treba napraviti = operacija,
- nad čim to treba izvršiti = objekt nad kojim se obavlja operacija.

Primjer. Izvadi kabel (sličica) iz vrećice.

Isti oblik imaju i instrukcije na računalu:



Primijetite da se ista operacija

- može obavljati na(d) raznim podacima.

Opis algoritma — općenitost

Algoritam bi trebao raditi nad "općenitim" podacima, tj. bitno je samo da se stalno radi o istom postupku.

- Konkretne podatke zadajemo svaki put kad izvršavamo algoritam, tj. možemo ih mijenjati.

Na primjer, puno je bolje napisati algoritam koji nalazi rješenja "opće" kvadratne jednadžbe

$$ax^2 + bx + c = 0,$$

za proizvoljne a, b, $c \in \mathbb{R}$, uz $a \neq 0$, nego onaj koji nalazi rješenja "konkretne" jednadžbe

$$x^2 - 3x + 2 = 0.$$

(Rješenja su, očito, 1 i 2.)

Opis algoritma (nastavak)

U slučaju kvadratne jednadžbe, rješenja su dana formulom:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Primijetite da izračunata rješenja mogu biti kompleksna.

- Što su ovdje instrukcije?

Napomene o točnosti (bolje rješenje — kasnije)!

Rješivost jednadžbi višeg stupnja:

- Do stupnja 4 ide "formulama",
- preko 5 — nema općih formula (jednostavnog algoritma), tj. moramo koristiti približne numeričke metode.

Svojstva algoritma

Dakle, algoritam izvodi neke operacije nad nekim podacima, u obliku niza koraka, i daje neko rješenje.

Gruba skica:

Osnovna svojstva algoritma su:

- ima (ili nema) ulazne podatke,
- ima izlazne podatke,
- završava u konačnom vremenu,
- uvijek je nedvosmisleno definiran,
- mora biti efikasan, tj. završiti u razumnom vremenu.

Ulaz/izlaz

Ulaz:

- Svaki algoritam ima 0 ili više, ali konačno mnogo ulaznih podataka (inače ih ne možemo ni pročitati).

Ako ih ima, njih moramo zadati/izabrati iz neke dozvoljene klase ulaznih objekata.

Algoritmi s 0 ulaza nisu jako česti, ali ima ih (obično, baš u matematici). Opisuju fiksni postupak.

- Recimo: provjeri je li 327 prost broj (i, je li?), ili riješi konkretnu kvadratnu jednadžbu.

Ako algoritam ima više dozvoljenih objekata na ulazu, kažemo da je općenit, jer rješava cijelu klasu problema.

- Recimo: kvadratna jednadžba s općim a, b i c.

Ulaz/izlaz (nastavak)

Izlaz:

- Svaki algoritam mora imati bar jedan izlaz, jer inače nije ostavio trag svog izvršavanja.

To je traženo "rješenje" našeg problema.

Napomena. Postoje programi koji se "stalno vrte", bez izlaza. Njih zovemo procesi. Na primjer, operacijski sustav računala.

Ova prva dva svojstva pričaju o objektima na ulazu i izlazu, ali ništa ne kažu o tome kako se iz jednog dolazi do drugog.

Ostala svojstva nešto govore o "crnoj kutiji" na ranijoj skici

Konačnost

Konačnost:

- Svaki algoritam mora završiti u konačno mnogo koraka, za svaki dozvoljeni ulaz.

U programu uvijek treba provjeriti je li ulaz korektno zadan.

- Na primjer, u kvadratnoj jednadžbi, koeficijent a može biti 0. U tom slučaju, jednadžba nije kvadratna, a u formuli za rješenje pojavit će se dijeljenje s 0!

Savjet. U praksi treba predvidjeti sva moguća korisna ograničenja ulaza i ugraditi ih u program.

- Na primjer, program koji učitava temperaturu T vode u nekoj posudi, trebao bi provjeriti je li $0 \leq T \leq 100$.

Brojevi u prirodi imaju jedinice — ovdje je T u $^\circ$C.

Definiranost i nedvosmislenost

Kad projektiramo algoritam, ne znamo odmah na početku

- od kojih se koraka sastoji čitav postupak za rješenje problema.

Obično se problem rastavi na nekoliko većih cjelina, koje rješavamo pazeći na međuovisnost potproblema.

- Ako su te cjeline prevelike za izravno rješavanje, one se mogu rastavljati na još manje dijelove, ...

Ova metoda zove se metoda postupnog profinjavanja (engl. stepwise refinement).

Dokle treba postupno profinjavati?

- Ovisi o izvršitelju algoritma — koje instrukcije on "prepoznaje", u smislu da ih može ili zna izvršavati.

Definiranost i nedvosmislenost (nastavak)

Definiranost i nedvosmislenost:

- Algoritam se sastoji od niza osnovnih (elementarnih, primitivnih) instrukcija, koje moraju biti jednoznačno i nedvosmisleno definirane za izvršitelja algoritma.

Primjer. Ja volim tropetinsku pitu od jabuka (3/5 jabuka i 2/5 tijesta).

Ako je moja "bolja polovica" kod kuće, onda će moja molba "Napravi mi pitu od jabuka" biti uslišena.

- Za nju je pravljenje pite od jabuka elementarna instrukcija, jer je ona dovoljno moćan izvršitelj.

Dakle, za moj algoritam "kako se dočepati pite" to je elementarna instrukcija.

Definiranost i nedvosmislenost (nastavak)

Naravno, postoji i lošije rješenje (po mene).

Ako ona nije u blizini, a ja ipak želim pitu od jabuka, onda je

- instrukcija "ispeci pitu od jabuka" za mene presložena,
- pa moram zaviriti u kuharicu, gdje su dane "jednostavne" instrukcije za pravljenje pite (vidi prilog).

Kad za rješavanje problema koristimo računalo, potrebno je znati

- što ono "zna" i može napraviti,
- tj. što su elementarne instrukcije — operacije i podaci.

Za zapisivanje tih algoritama postoje jezici ključno određeni "snagom" računala (poput C-a).

Efikasnost

Efikasnost:

- Algoritam mora završiti u razumnom vremenu, što je bitno jači zahjev od konačnosti.
- Recimo, 500 godina nije razumno vrijeme!

A, ima li takvih algoritama? Ima!

- Tzv. NP–potpuni problemi, za koje se ne zna postoje li efikasni algoritmi. (Otvoreni problem "P = NP?".)
- Primjer. Problem trgovačkog putnika (TSP).

Napomena. Postoje problemi za koje ne postoji algoritam za njihovo rješenje — tzv. algoritamski nerješivi problemi.

Primjer. Popločavanje ravnine (skiciraj).

Principi rada računala
(Građa i funkcioniranje)

Sadržaj

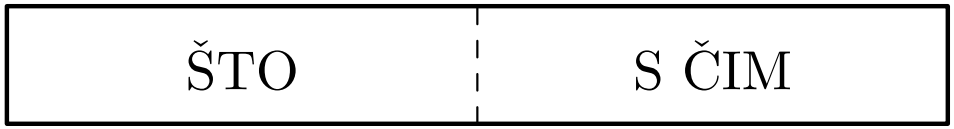
- Principi rada računala:
  - Što je računalo?
  - Instrukcije — operacije i podaci.
  - Osnovni dijelovi računala — ulaz, izlaz, memorija.
  - Izvršni dio računala — procesor.
  - Von Neumannov model računala.
- Dodatak:
  - Turingov stroj — matematički model računala.

Što je računalo?

Što je računalo?

- Računalo = stroj za izvršavanje algoritama.

Prisjetimo se općeg oblika instrukcija:



što = operacija, s čim = podatak ili operand.

Operacije koje računalo "zna" izvršiti su

- osnovne strojne instrukcije.

Podaci s kojima "zna" raditi su

- osnovni ili elementarni podaci.

Građa i funkcioniranje računala

Želja: imati računalo koje dozvoljava, tj. zna izvršiti

- "složene" instrukcije na "složenim" podacima.

Problem je čisto tehnološki: kako to brzo realizirati?

- Zbog toga postoje ograničenja koja diktiraju opći izgled današnjih računala.

Prije opisa, jedno prirodno pitanje:

- Zašto matematičar uopće mora nešto znati o principima rada i građi računala?

Odgovor je jednostavan, a izlazi iz svrhe/cilja:

- Zato da bi ih mogao efikasno koristiti, tj. pisati brze i točne algoritme (ograničenja proizlaze iz fizičke građe).

Ulaz, izlaz, memorija

Budući da algoritam ima ulaz i izlaz, mora to imati i računalo.

- Ulazni dio: čita podatke s nekog medija (ulaz algoritma).
- Izlazni dio: piše podatke na neki medij (izlaz algoritma).

Algoritam izvršava neke instrukcije nad podacima koje je učitao, tj. iz njih pravi nove podatke, koje može više puta koristiti u sljedećim instrukcijama, i tako redom ...

Prema tome, računalo mora moći te podatke negdje spremiti. Taj dio računala je

- memorija — u koju računalo može spremiti neke podatke i iz nje može čitati te podatke.

O tehničkoj realizaciji i izgledu memorije — malo kasnije.

Izvršni dio računala

Naravno, računalo mora imati i onaj "ključni" komad, tzv.

- izvršni dio — koji izvršava instrukcije nad podacima.

Pitanje: A kako "zadajemo" instrukcije, odnosno, algoritam?

Tu se, i u teoriji i praktično, javlja bitna dilema:

- treba li nam poseban stroj za svaki algoritam posebno,
- ili stroj opće namjene, koji izvršava razne algoritme?

Poseban stroj izvršava samo jedan fiksni algoritam koji je "tvrdo" ugrađen u arhitekturu stroja (tzv. "firmware").

- Takav stroj nije besmislen. Ako malo šire gledamo, to su upravljački sklopovi u raznim uređajima.

Računalo opće namjene — spremanje algoritma

Računalo opće namjene je bitno složenije, jer s algoritmima mora postupati slično kao s podacima:

- svaki algoritam mora učitati, spremiti i izvršiti.

Takvo računalo stalno izvršava jedan meta-algoritam (jer se stalno vrti — ne završava izvođenje sve dok ne ugasimo stroj).

- To je ono što obično zovemo operacijski sustav.

Pitanje: Gdje se spremaju algoritmi, odnosno, programi?

Mogli bismo imati posebnu memoriju za programe. Na pr.,

- matematički model računala — Turingov stroj, ima posebni upravljački modul koji "sadrži" algoritam, a "obična" memorija je traka (v. dodatak).

von Neumannov model računala

Međutim, nema potrebe za odvojenim memorijama, pa se

- programi, odnosno, algoritmi spremaju u istu memoriju kao i podaci,
- a upravljački dio računala brine o tome "što je gdje".

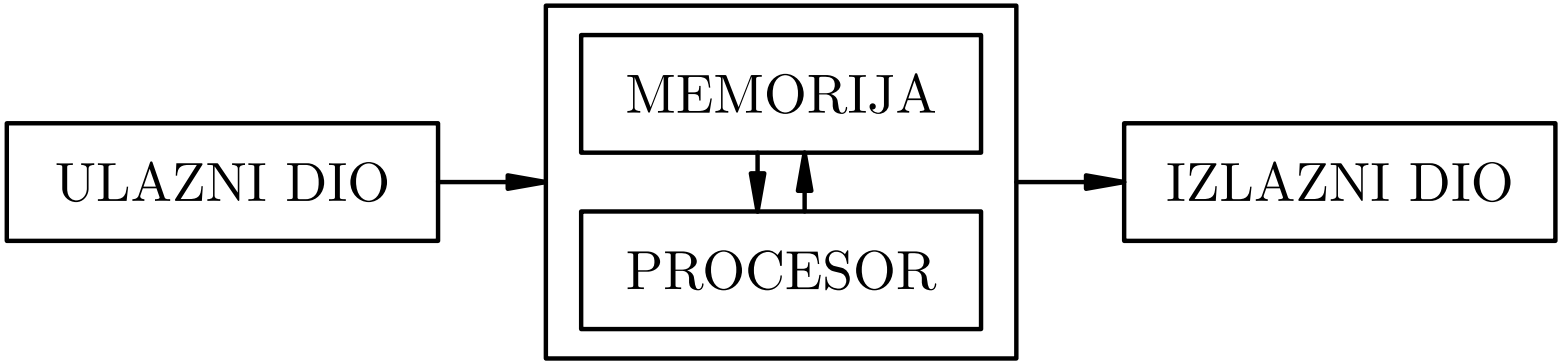
To je ključna stvar von Neumannovog modela računala.

- Ideja je nastala početkom četrdesetih godina prošlog stoljeća, a objavljena je tek nakon 2. svjetskog rata.
- Za vrijeme rata poslužila je kao podloga za gradnju prvih računala opće namjene.
  - Svrha: računanje balističkih tablica (posebno za brodove), i jasno je da je bila strogo čuvana tajna.

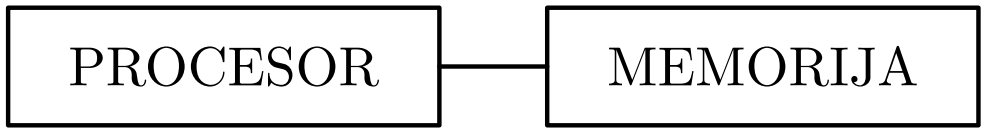
Sva moderna računala bazirana su na tom modelu.

von Neumannov model računala (nastavak)

Sasvim općenito, ako izvršni dio računala zovemo standardnim imenom procesor, onda shematski model računala izgleda ovako:



Slika bitnog dijela računala je:



Kako izgleda i funkcionira taj dio — više u nastavku.

Građa računala
(Osnovi dijelovi računala)

Sadržaj

- Građa računala:
  - Memorija — bistabil, bit, byte, riječ.
  - Procesor — registri, aritmetičko–logička jedinica, upravljačka jedinica.
  - Ulazna jedinica, izlazna jedinica.

Memorija — bistabil i bit

Memorija računala sastoji se od osnovnih elemenata koje zovemo bistabili.

- Bistabil može biti u (jednom od) 2 stabilna stanja (BI = 2).

Stabilno stanje? Ako je element u jednom od stanja, on će ostati u tom stanju sve dok ne uložimo energiju da se to stanje promijeni u drugo stanje.

Matematički rečeno, količina informacije koju možemo spremiti (pohraniti) u takvom elementu je

$$1 \text{ bit} = 1 \text{ binarna znamenka}.$$

Zato se ta dva stanja uobičajeno i označavaju binarnim znamenkama 0 i 1.

Memorija — malo povijesti

Nekad, u doba ranih računala (1960-ih) bistabil se realizirao pomoću feritnih jezgrica.

- Feritne jezgrice sastojale su se od sitnih željeznih prstenova kroz koje je prolazila žica.
- Puštanje struje u jednom ili drugom smjeru rezultiralo je magnetizacijom te jezgrice u jednom od 2 smjera.

Danas se memorija izrađuje od sitnih tranzistora koji rade kao elektronički prekidači, po principu

- ima struje — nema struje,

tj. opet imaju 2 stanja.

Sutra . . . Tko zna?

Memorija — bitovi i logičke operacije

Osnovne logičke operacije ne, i, ili na pojedinim bitovima realiziraju se tzv. logičkim sklopovima.

Uočite da logičke operacije "rade" isto kao i aritmetičke operacije na binarnim znamenkama 0, 1:

- ne — komplement, ili $0 \leftrightarrow 1$, ili $1 -$ operand,
- i — množenje,
- ili — zbrajanje.

Kad bitove organiziramo u veće cjeline (na primjer, dogovor prikaza cijelih brojeva binarnim znamenkama),

- pomoću takvih logičkih sklopova mogu se realizirati i osnovne aritmetičke operacije na brojevima.

(Zbrajač i slični "elektronički sklopovi".)

Memorija — zašto bitovi?

Čisto tehnički, tu postoje 2 bitna ograničenja:

- Nemoguće je napraviti brzi stabilni element koji bi imao više od 2 stabilna stanja.
  - Bilo je nekih pokušaja s 3,
  - a sve je počelo mehanički — s 10 (Pascal, Babbage).

  No, to je presporo. Zato su računala "binarna".

- Brzina svjetlosti je, trenutno, fundamentalno ograničenje brzine računala.
  - Minijaturizacija — 45 nm, 32 nm, 22 nm, 14 nm, ... tehnologije, uglavnom, služi povećanju brzine.

  Tu smo stigli blizu granice, s današnjom tehnologijom.

Memorija — "usko grlo" računala

Brzina svjetlosti diktira

- brzinu upravljanja i izvršavanja operacija u računalu.

Bitna je veličina prekidača/tranzistora kroz koje prolazi struja.

- Logički sklopovi u procesoru su još relativno brzi.
  - Na primjer, standardni PC procesori rade na frekvencijama od preko 3 GHz.

  Međutim, najveća frekvencija je oko 4 GHz i tu stoji već neko vrijeme. IBM Power procesori idu do 5 GHz.
- Daljnji napredak u snazi procesora ne ide ubrzanjem, nego paralelizacijom (Dual core, Quad core, ...).

Samo zato još uvijek vrijedi tzv. Mooreov zakon ($2\times$ u ≈ 2 g.).

Memorija — "usko grlo" računala (nastavak)

Kod memorija, situacija je još kompliciranija, jer je bistabilu potrebno neko vrijeme za promjenu stanja iz jednog u drugo.

- To vrijeme je ključno usko grlo arhitekture svih modernih računala.
  - Na primjer, jako brze memorije rade na frekvencijama od preko 2 GHz (i to razno–raznim trikovima),

  ali, zasad, nisu stigle ni blizu 3 GHz.
- Ova razlika u brzini procesora i memorije rješava se na dva načina:
  - paralelizacijom — podjelom na više "chipova",
  - hijerarhijskom građom cijele memorije u računalu.

Memorija — organizacija i izgled

Slično Turingovom stroju, osnovni elementi memorije su bitovi, ali memorija:

- nije linearna (poput trake), već je pravokutna (2-dim.),
- i nije beskonačna, nego konačna (pa nije potrebno imati prazni simbol koji označava kraj trake).

Zašto pravokutna organizacija memorije? Funkcionalno,

- 1 bit je premala količina informacije za smislenu obradu.

Zbog toga se bitovi organiziraju

- u veće cjeline — s fiksnim brojem bitova.

Svaku takvu cjelinu zovemo riječ memorije, a broj bitova u riječi je duljina riječi.

Memorija — organizacija i izgled (nastavak)

Svrha: Riječ je osnovna cjelina za smisleni podatak.

- Takve veće cjeline prikazuju potrebne vrste podataka i na njima, kao cjelinama, možemo izvršavati pripadne operacije i to brzo — na nivou arhitekture računala.

Koliko bitova čini jednu riječ?

- Značenje pojma riječ se mijenjalo razvojem računala. Prikaz te povijesti — malo kasnije.

Nekad je riječ bila cjelina bitova predviđena za prikaz cijelih brojeva, ili (još bolje), za prikaz strojnih instrukcija i adresa.

Danas, gotovo univerzalno, riječ je cjelina bitova predviđena za prikaz jednog znaka, tj. 1 riječ = 1 Byte = 8 bitova.

Memorija — organizacija i izgled (nastavak)

Memorija je linearni niz riječi, a svaka riječ

- ima svoju adresu, tj. poziciju ili mjesto u nizu.

Slična organizacija vrijedi i za strukturu podataka koju zovemo niz ili polje:

- to je konačni uređeni niz podataka istog tipa, a pristup pojedinim podacima je moguć preko indeksa u nizu.

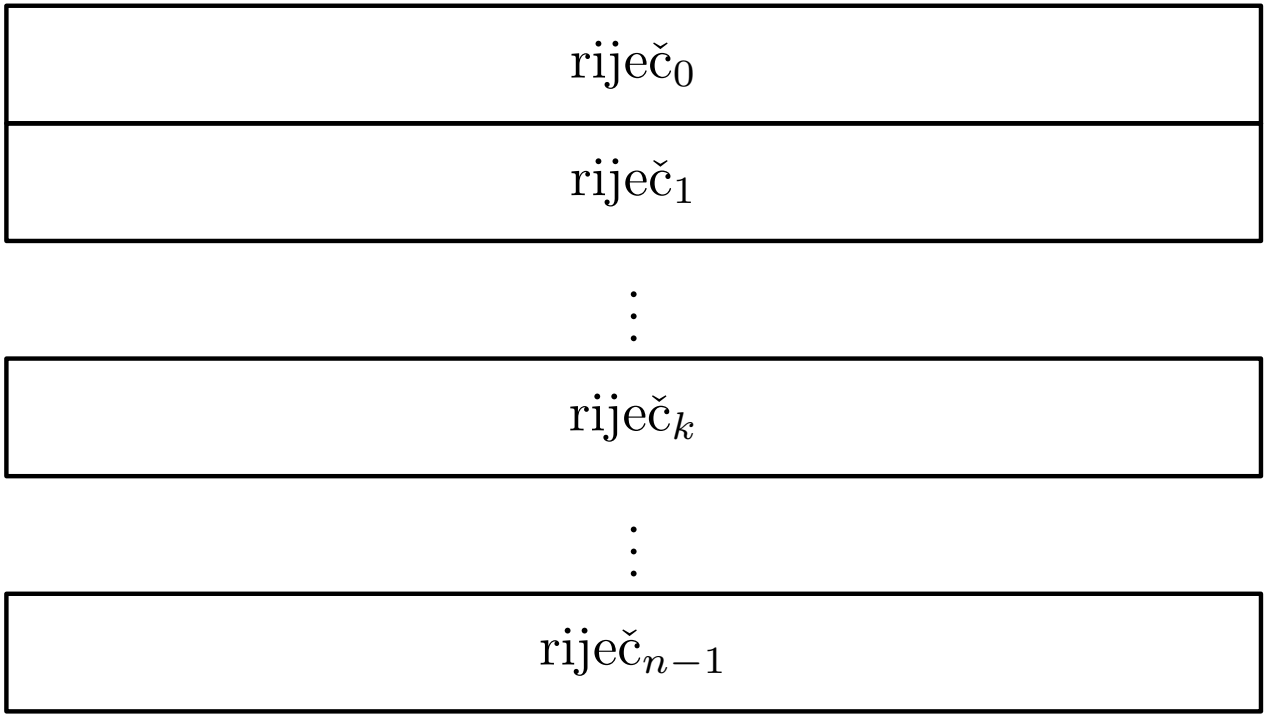
Matematički gledano, niz od n članova je uređena n-torka $x_1, x_2, \ldots, x_n$.

Razlika obzirom na matematičku definiciju:

- brojanje pozicije ne počinje s 1, nego s 0, jer
- adresa neke riječi u nizu označava koliko je ta riječ "odmaknuta" od početne riječi. Isti princip je u C-u!

Memorija — organizacija i izgled (nastavak)

Skica memorije s n riječi (odnosno, n byteova) izgleda ovako:



Kažemo da se riječ$_k$ (odnosno, byte$_k$) nalazi na k-tom mjestu, ili da se nalazi na adresi k.

Memorija — adresa i sadržaj

Dakle, svaki podatak u memoriji računala ima 2 bitna dijela:

- adresu — mjesto gdje je spremljen,
- sadržaj — vrijednost podatka spremljenog na toj adresi, tj. kako se interpretiraju pripadni bitovi.

Pristup svakom podatku ide preko adrese tog podatka, tj. preko njegove pozicije u memoriji. Obično još kažemo da

- adresa "pokazuje" na taj podatak u memoriji.

Ovo je ključno za razumijevanje rada računala:

- računalo vidi bilo koji podatak kao "sadržaj spremljen na određenoj adresi".

Memorija — pristup podacima preko adrese

Da bismo nešto spremili ili pročitali kao sadržaj lokacije u memoriji, moramo imati dvije osnovne instrukcije:

- spremi podatak na adresu "tu i tu",
- pročitaj podatak s adrese "te i te".

Netrivijalna posljedica:

- memorijske adrese su, također, podaci, tj.
- adresa podatka je ključni dio instrukcije koja nešto radi s podacima.

Memorija — adresni prostor

Veličina "adresnog" prostora = broj bitova predviđen za spremanje adresa (na pr., u sklopu instrukcija).

Ako imamo m bitova za spremanje adresa, onda možemo prikazati točno 2^m različitih adresa: od 0 do $2^m - 1$.

To određuje i maksimalnu količinu memorije, jer

- više od toga ne možemo adresirati!

Na primjer, ako je $m = 32$, onda je maksimalna količina memorije koju možemo adresirati

$$2^{32} = 4\,294\,967\,296 \text{ byteova},$$

odnosno, $4\,\mathrm{GiB}$. Zato se prešlo na veći adresni prostor.

Adrese se standardno "pišu" u heksadecimalnom sustavu.

Memorija — ispravno pisanje jedinica za količinu

Međunarodni sustav jedinica SI ima standardne prefikse za faktore koji su potencije broja 10. Na primjer,

- k ("kilo") označava faktor 10^3, a M ("mega") faktor 10^6.

Zbog $2^{10} = 1024 \approx 10^3$ imamo kaos u svijetu računala.

U prosincu 1998. godine Međunarodna elektrotehnička komisija (IEC) donijela je standardna imena i prefikse za faktore koji su potencije broja 2. Skraćena tablica je:

Faktor	Ime	Simbol	Porijeklo	Izvod
2^{10}	kibi	Ki	kilobinarni: $(2^{10})^1$	kilo: $(10^3)^1$
2^{20}	mebi	Mi	megabinarni: $(2^{10})^2$	mega: $(10^3)^2$
2^{30}	gibi	Gi	gigabinarni: $(2^{10})^3$	giga: $(10^3)^3$
2^{40}	tebi	Ti	terabinarni: $(2^{10})^4$	tera: $(10^3)^4$

Memorija — ispravno pisanje jedinica (primjeri)

Primjeri upotrebe ovih prefiksa i usporedba sa SI prefiksima:

Količina	Oznaka i značenje
jedan **kibibit**	$1\,\mathrm{Kibit} = 2^{10}\,\mathrm{bit} = 1024\,\mathrm{bit}$
jedan **kilobit**	$1\,\mathrm{kbit} = 10^{3}\,\mathrm{bit} = 1000\,\mathrm{bit}$
jedan **mebibyte**	$1\,\mathrm{MiB} = 2^{20}\,\mathrm{B} = 1\,048\,576\,\mathrm{B}$
jedan **megabyte**	$1\,\mathrm{MB} = 10^{6}\,\mathrm{B} = 1\,000\,000\,\mathrm{B}$
jedan **gibibyte**	$1\,\mathrm{GiB} = 2^{30}\,\mathrm{B} = 1\,073\,741\,824\,\mathrm{B}$
jedan **gigabyte**	$1\,\mathrm{GB} = 10^{9}\,\mathrm{B} = 1\,000\,000\,000\,\mathrm{B}$

Više o jedinicama, prefiksima, konstantama i sl. — na adresi

`http://physics.nist.gov/cuu/Units/`

Malo povijesti — byte i spremanje znakova

Danas je opće prihvaćeno da je

- 1 byte ("bajt") = 8 bitova.

Oznaka jedinice za byte je B, i to nije standardna SI jedinica.

Povijesno, pojam "byte" je nastao kao naziv za

- prostor za spremanje jednog znaka teksta.

Jasno je da su znakovi jedan od osnovnih tipova podataka koje treba spremiti u memoriju — za "humani" ulaz/izlaz (nije lijepo pisati i čitati nizove nula i jedinica).

- Znak se sprema u nekom kôdu, koji se prikazuje bitovima.

Prvi standardi za kôdiranje znakova pojavili su se istovremeno s prvim računalima.

Malo povijesti — razni kôdovi za znakove

Najpoznatiji standardi:

- EBCDIC — "asketskih" 6 bitova, samo velika slova,
- ASCII — 7-bitni standard s velikim i malim slovima,

No, u ta vremena, stvarno se trošilo 8 bitova na znak, kao četvrtina 32-bitne riječi. Tako je nastao byte, a zatim i

- 8-bitni ISO/ASCII — standard za dodatnih 128 znakova koji su potrebni za odgovarajuće jezike i razlikuje se od jezika do jezika (tzv. "ISO–nešto" character set).

Danas se sve više koristi Unicode. Osim "običnih" znakova,

- sadrži posebne znakove za razne jezike i pisma, kao i hrpu standardnih matematičkih simbola.

Zbog toga, uobičajeno, troši 2 bytea za jedan znak.

Malo povijesti — riječ i byte

Nekad je riječ zaista bila najmanja cjelina koju se moglo direktno adresirati. Recimo:

- IBM 1130 je imao 16-bitne riječi,
- mnogi strojevi (na pr., IBM 360) su imali 32-bitne riječi,
- Univac 11xx (xx = 06 ili 10) je imao 36-bitne riječi,
- CDC Cyber standardno su imali 60 ili 64-bitne riječi.

Nakon toga su se pojavila mikroračunala. Njihova je memorija bila sagrađena po principu 1 riječ = 1 byte, tj. byte je postao najmanja cjelina koju procesor može direktno adresirati.

Usput, i pripadni procesor je bio 8-bitni, a takva je bila i veza između procesora i memorije, tzv. sabirnica ili magistrala.

Iz tih mikroračunala su nastala i prva osobna računala!

Malo povijesti — riječ i byte (nastavak)

U modernim osobnim računalima, jedan byte je i dalje osnovna cjelina koja se može adresirati. Međutim, stvarna arhitektura računala koristi mnogo dulje cjeline ("riječi"):

- IA-16 — 16-bitne instrukcije i adrese (otišlo u povijest),
- IA-32 ili x86 — 32-bitne instrukcije i adrese (povijest),
- AMD64 ili x64 — to je IA–32 arhitektura, ali su adrese 64-bitne (vrijedi za većinu današnjih osobnih računala),
- IA-64 — Itanium procesori, sve je 64-bitno,
- a radi se i na 128-bitnom standardu za velika računala.

Stvarno, pojam "riječ" više nema puno smisla!

Dogovor oko termina — byte, riječ, jedinice

Dogovor oko termina i jedinica — da ne bude "kaosa":

- jedan byte = veličina prostora za spremanje jednog "običnog" znaka, odnosno, standardnog tipa `char` u C-u;
- jedna riječ = veličina prostora za spremanje jednog cijelog broja, odnosno, standardnog tipa `int` u C-u.

  To znači da je: 1 riječ = 4 B = 32 bita.

Nitko vam neće ozbiljno zamjeriti ako iskoristite da je

- 1 KB = 1024 B.

Samo budite svjesni da je to krivo, iako mnogi to često rade. Uostalom,

- teško ću izreći "imam osam gibibajta RAM-a" :-)

Tipovi podataka u računalu

Zasad nismo rekli koji osnovni tipovi podataka postoje u računalu — za nas kao korisnike.

Za računanje koristimo brojeve raznih vrsta:

- nenegativne cijele brojeve (bez predznaka), tj. podskup od $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$,
- cijele brojeve s predznakom (i negativni uključeni), tj. podskup od $\mathbb{Z}$,
- brojeve s pomičnim zarezom (engl. floating–point), tj. podskup od $\mathbb{R}$.

Za ulaz–izlaz koristimo: znakove.

Svi ostali tipovi, uglavnom, se svode na ove osnovne tipove.

Tipovi podataka u računalu (nastavak)

Veza arhitekture računala i tipova podataka ide tako daleko da se i najjednostavniji tip podataka

- logički ili Booleov tip, koji ima samo dvije vrijednosti: laž i istina (oznake F/T, $\bot/\top$),
- prikazuje preko cijelih brojeva, i to: laž $= 0$, istina $= 1$.

Osim ovih korisničkih tipova, trebamo još 2 stvari — bitne za rad samog računala:

- adresa — to je tip podataka sličan nenegativnim cijelim brojevima (stvarno, prikazi su im isti).
- instrukcije — koje se nekako kôdiraju bitovima, ovisno o arhitekturi procesora (nećemo ulaziti u to).

Procesor

Po von Neumannovom modelu, instrukcije/programi se, također, pamte u memoriji.

Zato procesor mora imati (bar) 2 bitna "radna" dijela:

- izvršni dio, tzv. aritmetičko–logičku jedinicu — naziv dolazi od tipičnih operacija koje ona izvršava nad korisničkim tipovima podataka,
- upravljački dio — koji obavlja sve osnovne poslove vezane za instrukcije, tj. brine se za:
  - dohvat instrukcija iz memorije (engl. fetch),
  - njihovu interpretaciju (engl. decode) i
  - njihovo izvršavanje (engl. execute).

Procesor (nastavak)

Dakle, upravljački dio upravlja radom aritmetičko–logičke jedinice prema instrukcijama.

Osim toga, procesor ima i skup registara. To je radna memorija procesora za spremanje:

- podataka nad kojima se izvršavaju instrukcije i rezultata tih operacija,
- instrukcija (odnosno, dijelova instrukcija) koje se izvršavaju.

Naime, operacije u procesoru mogu se napraviti

- samo na operandima koji su prebačeni iz memorije u registre procesora.

Procesor (nastavak)

Zašto je organizacija baš takva?

- Procesor i memorija su fizički odvojeni (razni "čipovi")
- i komuniciraju preko "kanala" (magistrala ili sabirnica).

Za izvršavanje bilo koje instrukcije,

- prvo instrukciju treba "dovući" iz memorije u procesor.

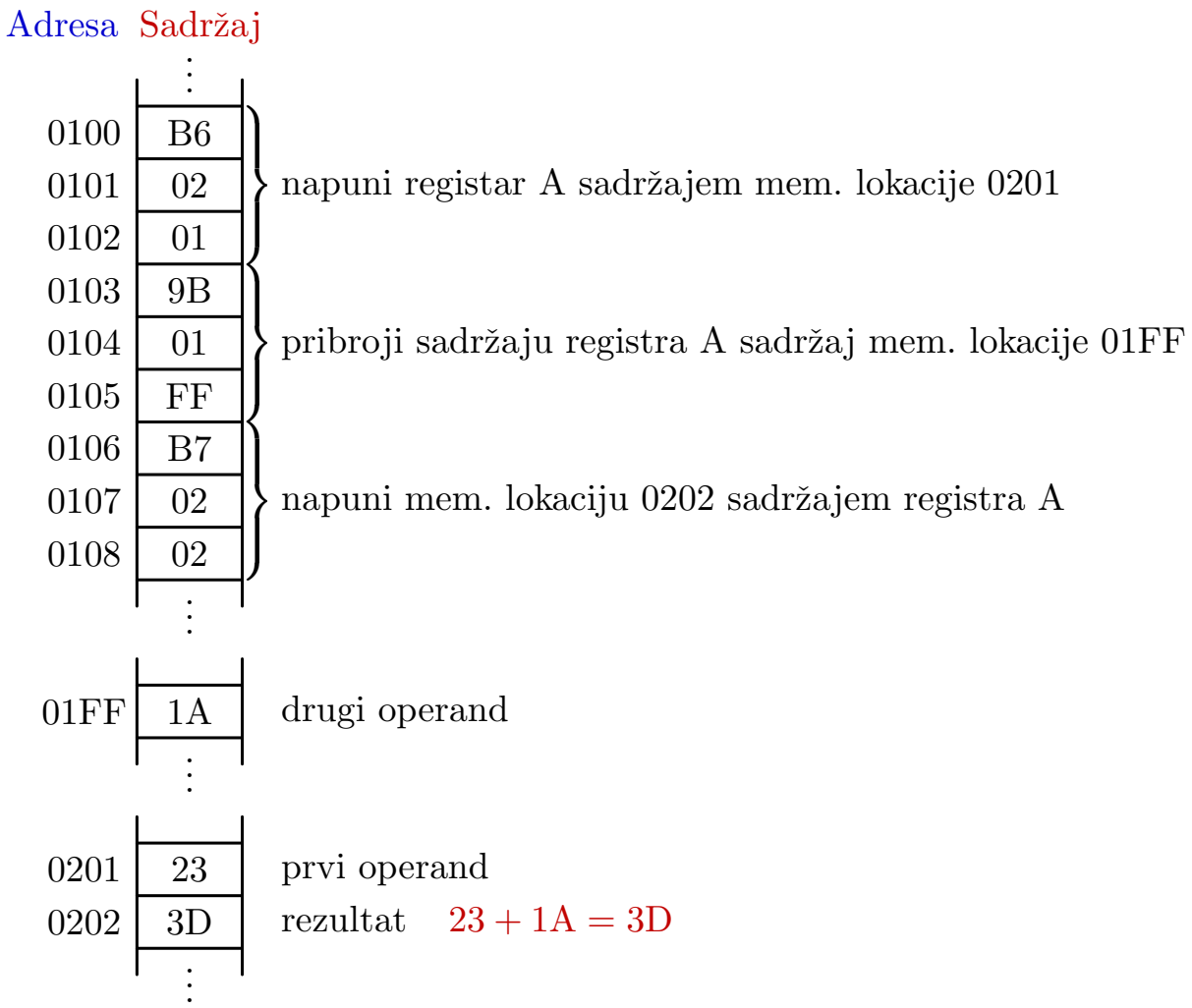
Potpuno isto vrijedi i za podatke!

Osnovne instrukcije za baratanje podacima:

- LOAD REG, adr — "napuni" registar "REG" sadržajem s adrese "adr",
- STORE REG, adr — "spremi" podatak iz registra "REG" na adresu "adr" (promjena sadržaja memorije).

Program za zbrajanje dva broja

Program:



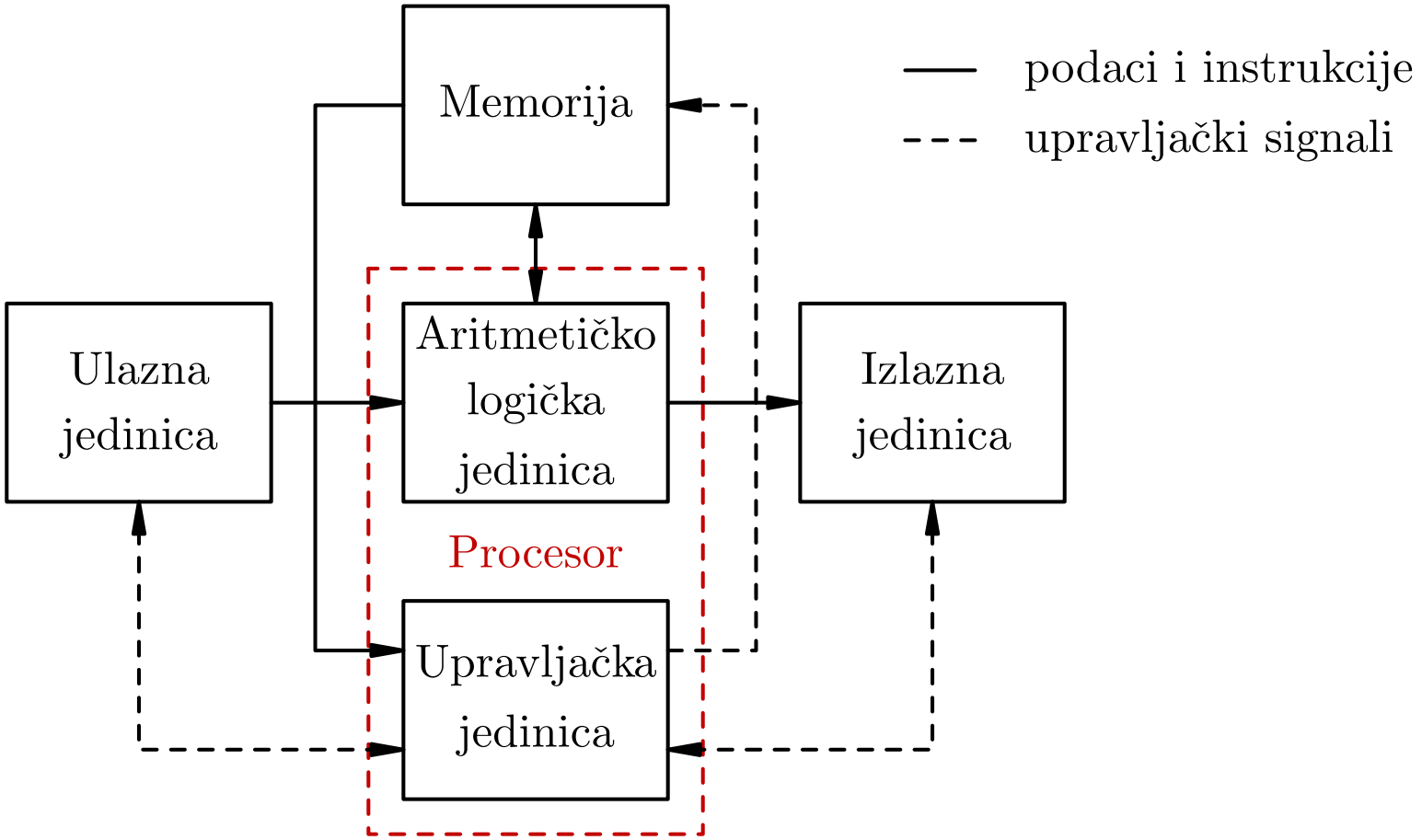
Ulazne i izlazne jedinice

Svako računalo, osim memorije i procesora, mora imati još i

- ulaznu jedinicu — koja podatke iz vanjskog svijeta pretvara u binarni oblik,
- izlaznu jedinicu — koja rezultate iz binarnog oblika pretvara u oblik razumljiv korisniku, ili ih pohranjuje za daljnju obradu.

Shema računala

Shematski, ovako izgledaju osnovni dijelovi računala:



Dodatna literatura — floating–point aritmetika

Ako želite saznati još ponešto o floating–point prikazu brojeva i aritmetici, pogledajte/potražite članak:

- David Goldberg, What Every Computer Scientist Should Know About Floating–Point Arithmetic,
  ACM Computing Surveys, Vol. 23, No. 1, March 1991,
  pp. 5–48.

Ovo je "ozbiljan" matematički članak — ima i teorema!

Postoji i prošireno izdanje, objavljeno kao

- Appendix D, Numerical Computation Guide,
  Sun Microsystems, Inc., July 2001.

Zbog "copyrighta", ovo nije na mom webu, ali možete dobiti, ako želite.

Dodatna literatura — memorija računala

Ako želite saznati više o detaljima funkcioniranja memorije modernih računala, pogledajte članak:

- Ulrich Drepper, What Every Programmer Should Know About Memory, 2007.

Kompletna (prva) verzija članka može se naći na adresi

`http://www.akkadia.org/drepper/cpumemory.pdf`

Članak možete i čitati na webu. Početak je na adresi

`http://lwn.net/Articles/250967/`

a pri dnu su linkovi na ostala poglavlja.