# 1. Uvod

## 1.1. Prvi susret s operacijskim sustavom

### 1.1.1. Zadaci operacijskog sustava

Računalni sustav sastavljen je od procesora, radnog spremnika, vanjskih spremnika i različitih ulazno-izlaznih naprava. Te komponente čine sklopovlje računala. Samo sklopovlje računala (mogli bismo reći: "golo računalo") nije nam od velike koristi ako uz njega ne postoji odgovarajuća programska oprema. Različiti primjenski programi, s pomoću kojih korisnici računala obavljaju sebi korisne zadatke, transformiraju računalni sustav u odgovarajući virtualni stroj. Kao potpora svim tim raznovrsnim programima služi skup osnovnih programa koji omogućuju provođenje radnih zahvata na računalu – izvođenje *operacija računala*. Taj se skup programa naziva *operacijski sustav*. Moglo bi se reći da operacijski sustav "nadoknađuje" sva ograničenja i nedostatke sklopovlja i stvara privid stroja koji je mnogo prikladniji za korištenje. Korisnik ne mora biti niti svjestan detalja provođenja neke operacije koju je on jednostavno zatražio. Operacijski sustav *skriva* od korisnika mnoge njemu nevažne *detalje* izvođenja neke operacije. Jedan je od zadataka operacijskog sustava dakle, *olakšavanje uporabe računala*.

Drugi važan zadatak operacijskog sustava jest organizacija *djelotvornog iskorištavanja svih dijelova* računala. Naime, unutar računala može se istodobno odvijati više poslova. Primjerice, istodobno dok se procesor "brine" o izvođenju jednog niza instrukcija, pristupni sklop pisača može iz glavnog spremnika prenositi sadržaj na pisač. Operacijski sustav se mora pobrinuti da se procesor prebacuje s izvođenja jednog niza instrukcija na drugi i podržati *višeprogramski rad*. Operacijski sustav mora svakom pojedinom programu omogućiti pristup do potrebnih mu datoteka i svih ostalih sredstava. Sva nužna sredstva operacijski sustav mora dodjeljivati pojedinim programima i oduzimati ih od drugih tako da ona budu što je moguće bolje iskorištena. Osim toga, operacijski sustav mora omogućiti ostvarenje komunikacije između računala ako su ona spojena u mrežu.

Sve operacije koje operacijski sustav omogućuje moraju se na neki način pokrenuti. Neke zahtjeve za pokretanje operacija postavlja čovjek – korisnik računala – a neki zahtjevi dolaze neposredno iz programa. Korisnik može, primjerice, zahtijevati da se neki program pokrene ili da se neka datoteka premjesti s diskete na disk ili obrnuto. Primjeri za zahtjeve koji dolaze iz programa jesu zahtjevi za obavljanje ulaznih i izlaznih operacija. Način postavljanja zahtjeva operacijskom sustavu, kao i izgled povratnih poruka operacijskog sustava, mora biti dogovoren. Utvrđeni način takva komuniciranja zovemo sučeljem. Naziv sučelja koristi se, općenito, za čvrsto dogovoreni način uspostavljanja veze između nekih, inače razdvojenih, cjelina.

Operacije operacijskog sustava, dakle, može pokrenuti:

- čovjek preko korisničkog sučelja ili
- program preko sučelja primjenskog programa.

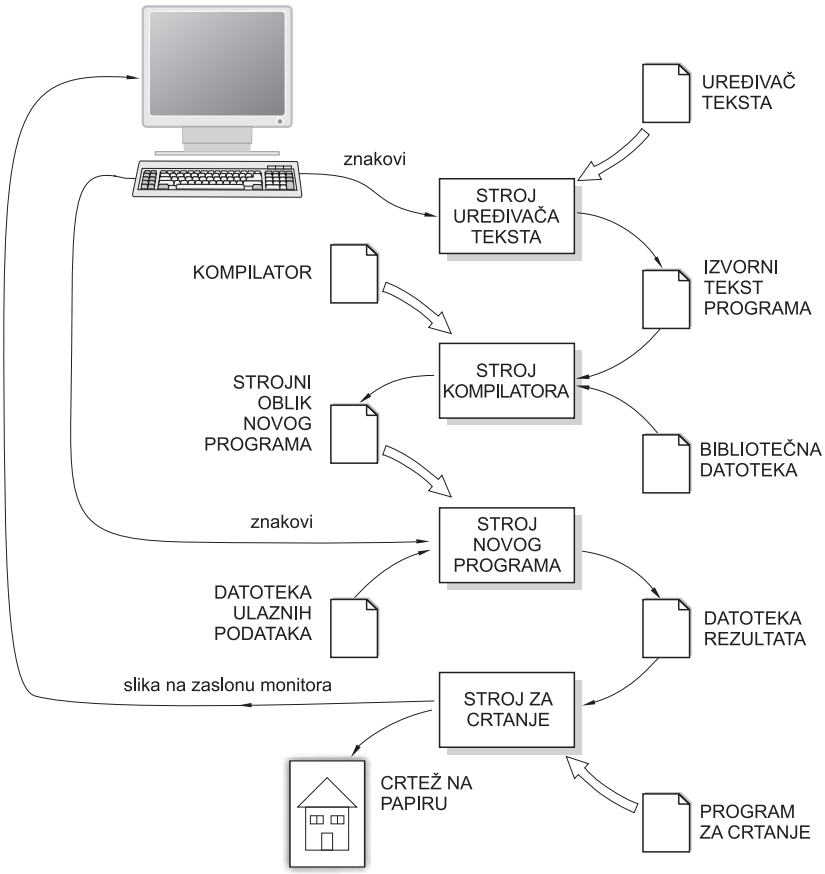
Operacijski sustav preko tih sučelja povratno vraća rezultate zatraženih operacija ili samo informacije o obavljenom zahtjevu. Pojedine operacije operacijskog sustava može pokrenuti čovjek kroz korisničko sučelje prema operacijskom sustavu. Najčešće operacije koje pokreće čovjek odnose se na datoteke: pretražuje se imenik datoteka kako bi se pronašlo željenu datoteku, datoteke se premještaju, kopiraju, preimenuju.

Svoje zahtjeve čovjek postavlja tako da ih upisuje na tipkovnici ili ih odabire na slikovnom sučelju. Već prvi pristup računalu zahtijeva upoznavanje nekoliko osnovnih naredbi operacijskom sustavu. U današnjim operacijskim sustavima prevladava uporaba slikovnog sučelja između čovjeka i stroja. Slikovna sučelja sa sličicama – ikonama – koje simboliziraju pojedine operacije ili objekte olakšavaju čovjeku uporabu računala. Pojedine komande nude se i u pisanom obliku i samo ih treba s pomoću miša odabrati iz ponuđenog izbornika – menija. Isto tako, povratne informacije operacijskog sustava pojavljuju se u vrlo razumljivom slikovnom obliku. Posljednjih se godina izgled slikovnog sučelja pomalo ujednačava i postoje već i neki dogovori koji vode prema standardizaciji slikovnog sučelja. Zaslon monitora tako postaje "prozor"[1] kroz koji se gleda u unutrašnjost računala.

### 1.1.2. Odvijanje tipičnog posla u računalnom sustavu

Napomenimo da se svi programi i svi podaci u računalu trajno čuvaju u obliku datoteka u vanjskim spremnicima. Stoga je pri upoznavanju računala i njegova operacijskog sustava najvažnije upoznati rad s datotekama. Sve što se u računalu događa svodi se na premještanje, mijenjanje i pohranjivanje datoteka. Svaki program koji se pokreće u računalu dolazi iz neke datoteke. Za obavljanje bilo kojeg posla računalom potrebna nam je skupina odgovarajućih programa koji su pohranjeni u obliku datoteka. Ti programi stvaraju radno okruženje za obavljanje tog posla.

[1] Engleski naziv za prozor – *window* – poslužio je kao naziv operacijskih sustava *Windows*, *Windows 95*, *Windows NT* tvrtke Microsoft, kao i za grafičko korisničko sučelja *X/Windows* za operacijske sustave UNIX.




**Slika 1.1.** Uloga datoteka u odvijanju programa

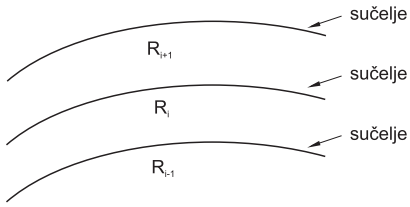
Primjer postupka pripreme jednog programa i njegova korištenja prikazan je na slici 1.1. Datoteke su prikazane kao krugovi, a aktivni programi kao pravokutnici. Neke datoteke koje sadržavaju gotove strojne programe postat će aktiviranjem programi. Na slici je ta pretvorba simbolizirana podebljanom strelicom koja vodi od kruga na pravokutnik. Programi kroz svoje sučelje prihvaćaju ili pojedinačne znakove s tipkovnice ili podatke pripremljene u ulaznim datotekama i proizvode izlazne datoteke. Posao se odvija u nekoliko koraka:

- Najprije se iz datoteke aktivira program za uređivanje teksta. On prihvaća znakove s tipkovnice i pohranjuje ih postupno u datoteku izvornog programa.
- Nakon toga se iz svoje datoteke aktivira kompilator. On prihvaća datoteku izvornog teksta kao ulaznu datoteku i uz dodatno korištenje bibliotečne datoteke izgrađuje strojni oblik programa, koji se opet pohranjuje u datoteku.
- Novostvoreni strojni program se aktivira i postaje program koji prihvaća ulazne podatke s tipkovnica ili iz unaprijed pripremljene podatkovne datoteke.

- Taj program izračunava rezultate i pohranjuje ih u datoteku rezultata.
- Na kraju, može se aktivirati neki program za crtanje koji će kao ulaznu datoteku prihvatiti datoteku rezultata i proizvesti crtež na papiru i rastersku sliku na zaslonu. Iz ovog je prikaza vidljivo kako se računalo pretvara iz jednog virtualnog stroja u drugi jednostavnom zamjenom primjenskih programa. Operacijski sustav mora omogućiti da se te pretvorbe obavljaju što jednostavnije.

## 1.2. Hijerarhijska izgradnja operacijskog sustava

Već se iz ovog kratkog opisa poželjnog ponašanja računalnog sustava naslućuje da su operacije koje mora omogućiti operacijski sustav razmjerno složene. Znamo da se te operacije moraju provoditi na računalnom sklopovlju koje je u stanju izvoditi samo vrlo jednostavne instrukcije. Stoga pri zasnivanju i ostvarenju operacijskih sustava (a i primjenskih programa) treba svladati golemi jaz između mogućnosti sklopovlja koje manipulira, takoreći, pojedinačnim bitovima pa do složenih operacija koje oživotvoruju naše apstraktne zamisli. Svladavanje tog jaza pokušava se ostvariti svojevrsnom hijerarhijskom izgradnjom sustava.




**Slika 1.2.** Razine u hijerarhijskoj izgradnji sustava

Hijerarhijski pristup izgradnje složenih sustava koristi se i u mnogim drugim područjima. Principi stroge izgradnje hijerarhijskog sustava su sljedeći:

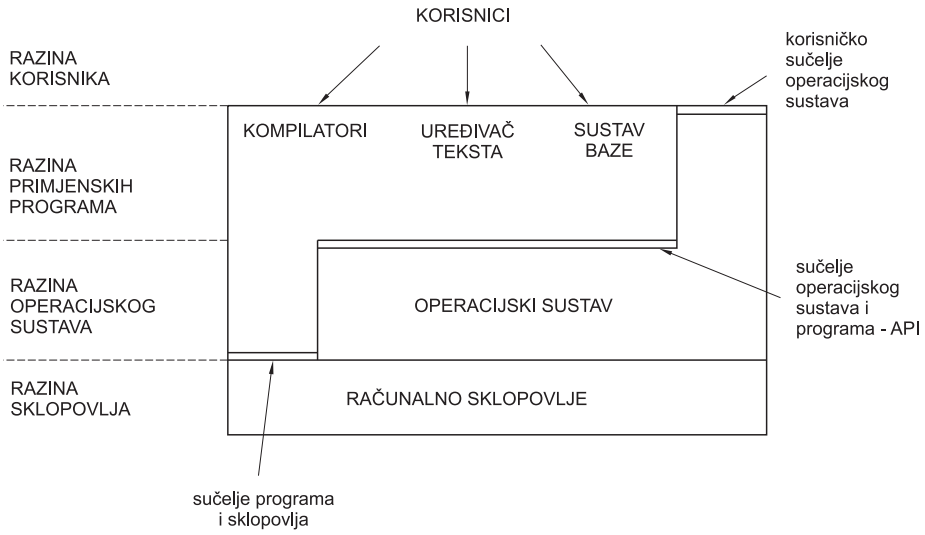
- sustav se izgrađuje po razinama;
- svaka razina sastoji se od objekata i operacija nad tim objektima;
- objekti i operacije neke razine izgrađuju se samo s pomoću objekata i operacija prve neposredne niže razine;
- detalji ostvarenja objekata i operacija pojedine razine skriveni su (a nisu nam ni važni) [2].

Slika 1.2. simbolizira hijerarhijsku izgradnju. Razina $R_i$ ostvarena je samo s pomoću objekata i operacija razine $R_{i-1}$, a razina $R_{i+1}$ samo od objekata i operacija iz razine

[2] Ovdje spominjani objekti i operacije nad njima ne trebaju se do kraja poistovjećivati s objektima u objektno zasnovanom programiranju, iako mnoge sličnosti nisu slučajne.

$R_i$. Razina $R_{i+1}$ "ne vidi" objekte i operacije iz razine $R_{i-1}$. To znači da bismo razinu $R_{i-1}$ mogli i zamijeniti a da to razina $R_{i+1}$ i ne primjeti, tj. da ona kroz svoje sučelje "vidi" jednake objekte i operacije u razini $R_i$. Prema tome, u razini $R_i$ trebalo bi s novom razinom $R_{i-1}$ opet izgraditi jednake objekte i operacije.

U izgradnji operacijskih sustava ne slijede se do kraja principi stroge hijerarhijske izgradnje. Tako se, primjerice, u nekoj razini može koristiti ne samo operacije neposredne niže razine već i daljnjih nižih razina. Takav se pristup može približiti strogoj hijerarhijskoj izgradnji ako se smatra da se neki objekti i operacije prenose nepromijenjeni u više razine. Oni objekti i operacije koji nisu preneseni ostaju za više razine "skriveni", tj. ne mogu se u višim razinama koristiti.




**Slika 1.3.** Hijerarhijska struktura računalnog sustava

Načelo hijerarhijske izgradnje može se uočiti već i na slici 1.3. koja predstavlja grubu slojevitu strukturu sustava. Na računalno sklopovlje, koje čini osnovnu razinu računalnog sustava, oslonjena je razina operacijskog sustava računala. Iznad razine operacijskog sustava nalazi se razina primjenskih programa koji transformiraju računalni sustav u različite virtualne strojeve. Različiti korisnici sa svoje razine kroz sučelja primjenskih programa gledaju "svoj" virtualni stroj. Ti korisnici moraju poznavati samo neke najosnovnije funkcije operacijskog sustava, a mogu do njih pristupiti kroz sučelje operacijskog sustava predviđeno za korisnike. U današnjim se primjenskim programima nastoji što više međusobno ujednačiti sučelja kako bi "horizontalno" premještanje korisnika bilo što jednostavnije. Što više, dio sučelja operacijskog sustava koji je vidljiv s korisničke razine također je sličan sučeljima programa.

Primjenski su programi, s jedne strane, neposredno oslonjeni na sklopovlje (izvođenje programa svodi se na izvođenje niza strojnih instrukcija) i, s druge strane, na funkcije pripremljene unutar operacijskog sustava (sustavske funkcije) koje su dohvatljive kroz

njegovo sučelje prema primjenskim programima – API. U neku bismo ruku mogli isto tako reći da skup strojnih instrukcija pojedinog procesora čini sučelje između programa i računalnog sklopovlja. Ne zaboravimo da svi programi, bez obzira na to u kojem su programskom jeziku pripremljeni moraju biti prevedeni u strojni oblik prije nego li započne njihovo izvođenje. *Strojni oblik programa* sastoji se od niza *strojnih instrukcija* koje u sklopovlju računala izazivaju neke osnovne operacije. Jasno je da su i funkcije operacijskog sustava također oslonjene na sklopovlje i da su ostvarene nizom strojnih instrukcija.

## 1.3. Načini izučavanja operacijskih sustava

Pri razmatranju operacijskih sustava treba uzeti u obzir da je način i dubina njihova izučavanja određena ciljevima ljudi razvrstanih u pojedine interesne skupine:

- *Obični korisnici* računala, koji žele koristiti računalo samo kao pomagalo u svom svakidašnjem radu ne moraju mnogo znati ni o računalu, ni o njegovu operacijskom sustavu. Oni moraju moći samo pokrenuti svoje primjenske programe i svladati uporabu tih programa. Osnovna obuka o operacijskom sustavu za takve će korisnike biti vrlo kratka i jednostavna i sastojat će se od upoznavanja sučelja i nekoliko komandi za pokretanje osnovnih operacija operacijskog sustava.
- *Napredni korisnici* računala žele svoje primjenske programe izvoditi na što djelotvorniji način, zadovoljiti neka ograničenja nametnuta načinom njihove uporabe i s nekog određenog stanovišta optimirati njihovo izvođenje. Takvi korisnici moraju detaljnije poznavati svojstva računalne opreme na kojoj se izvode njihovi programi, pa prema tome i svojstva i mogućnosti operacijskog sustava. Upravo kroz operacijski sustav korisnici pristupaju do svih sredstava svog računalnog sustava. Oni moraju dovoljno dobro poznavati osnovne mehanizme kojima se ostvaruju pojedine funkcije operacijskog sustava kako bi s razumijevanjem mogli poduzimati odgovarajuće zahvate na svom sustavu.
- *Programeri primjenskih programa* koji će pripremati nove primjenske programe moraju dobro poznavati sve mogućnosti operacijskog sustava, i to posebice skup operacija koje nudi API, kako bi u svojim programima mogli djelotvorno iskoristiti računalni sustav. Operacijski ih sustav oslobađa mnogih mučnih detalja pristupanja računalnim sredstvima – ti su detalji razriješeni unutar procedura i funkcija koje se pozivaju kroz API.
- *Specijalisti iz područja računarstva* koji će se baviti zasnivanjem, projektiranjem i održavanjem računalnih sustava (pa eventualno i pregradnjama i dogradnjama operacijskih sustava).

Gradivo koje je obrađeno u ovom udžbeniku može biti zanimljivo svim spomenutim interesnim skupinama. Naime, gradivo pojedinih poglavlja izloženo je tako da se kreće

od jednostavnih obrazloženja općih postavki prema sve složenijem i detaljnijem objašnjavanju teme koja se razmatra. Za prvu interesnu skupinu običnih korisnika gradivo je u pravilu malo preopširno, dok će potencijalni specijalisti nakon izučavanja cijelog udžbenika morati potražiti još i dodatnu literaturu. Moglo bi se, dakle, smatrati da je gradivo najbolje prilagođeno dvjema srednjim interesnim skupinama.

Za razmatranje djelovanja računalnog sustava prikladno nam je, barem donekle, slijediti strukturu njegove izgradnje, kako bismo lakše svladali njegovu složenost. Pritom su, u načelu, moguća dva pristupa:

- sustav se može početi razmatrati s "vanjske strane", tako da se pođe od korisničkog sučelja i u objašnjavanju slijedi redom one funkcije nižih razina koje su nužne za razumijevanje nekog podskupa operacija koje omogućuju ostvarenje razmatrane funkcije;
- sustav se može početi razmatrati od najnižih razina i postupno opisivati sve više i više razine.

Prvi je pristup uobičajen pri brzom upotrebnom upoznavanju operacijskog sustava. Svaki korisnik koji želi koristiti računalo mora naučiti neke osnovne komande koje mu omogućuju pokretanje željenog primjenskog programa i pritom ne mora ni znati kako se pojedine operacije provode. Za takav pristup upoznavanju operacijskog sustava najbolje je koristiti priručnike koji ukratko opisuju osnovne komande operacijskog sustava i uzorke njihova korištenja. U današnjim se računalima iz grafičkog sučelja može pokrenuti uslužni program s uputama (engl. *Help*) koji na zaslonu monitora ispisuje osnovna svojstva pojedinih komandi i načine njihove uporabe. Međutim, u takvom se pristupu vrlo često mora prihvatiti neke činjenice bez njihova potpunog razumijevanja, a ostaju nejasni i motivi koji su uzrokovali načine ostvarenja pojedinih funkcija.

Drugi pristup je prikladniji pri izučavanju računalnih sustava u okviru redovitog školovanja kada se bez posebne žurbe može pristupiti sustavnom izučavanju čitavog gradiva. U tom se pristupu mogu postupno objašnjavati pojedine razine sustava upravo onim redoslijedom kako se one i izgrađuju. Pritom se može naglašavati ona osnovna načela izgradnje koja imaju trajniju vrijednost i koja ne ovise o nekom konkretnom programskom rješenju.

Čitatelj koji na takav način pristupi usvajanju gradiva neće moći nakon prvih nekoliko lekcija koristiti računalo, ali će na kraju bolje razumjeti mnoge detalje ponašanja računalnih sustava i moći će se vrlo dobro snaći u raznovrsnim radnim okruženjima. On će naučiti osnovna načela izgradnje složenih programskih sustava i lakše će, i to s razumijevanjem, usvajati novine u izgradnji operacijskih sustava koji su još u razdoblju intenzivnog razvitka.

U ovom je udžbeniku stoga usvojen ovaj drugi pristup izučavanju operacijskih sustava. Polazi se od razine sklopovlja i postupno se izgrađuju sve više i više razine sustava. Pritom se oblikuju neki pojednostavljeni modeli sklopovskih komponenti koji olakšavaju razumijevanje osnovnih mehanizama, ali su dovoljno bliski današnjim ostvarenjima i prema tome omogućuju poimanje stvarnih mogućnosti sustava.



## PITANJA ZA PROVJERU ZNANJA 1

1. Što je operacijski sustav?
2. Koji su osnovni zadaci operacijskog sustava?
3. Navesti osnovne dijelove operacijskog sustava.
4. Što je to sučelje?
5. Što je to sučelje primjenskih programa (API)?

# 2. Model jednostavnog računala

## 2.1. Von Neumannov model računala i načini njegova ostvarenja

### 2.1.1. Funkcijski model računala

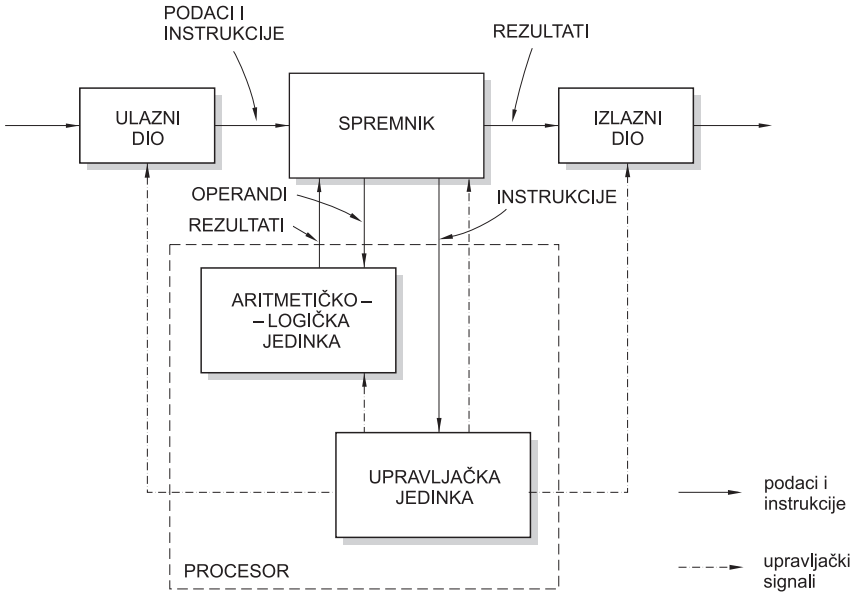
Današnji računalni sustavi zasnivaju se još uvijek pretežito na koncepcijskom modelu koji je još 1945. godine opisao John von Neumann.

Von Neumannov model utvrđuje da svako računalo mora imati sljedeće dijelove:

- *ulazni dio* preko kojeg se u spremnik unose iz okoline podaci i instrukcije programa;
- *izlazni dio* preko kojeg se u okolini prenose rezultati programa;
- *radni* ili *glavni spremnik* u koji se pohranjuju svi podaci i instrukcije programa uneseni izvana, kao i rezultati djelovanja instrukcija;
- *aritmetičko-logičku jedinku* koja može izvoditi instrukcijama zadane aritmetičke i logičke operacije;
- *upravljačku jedinku* koja dohvaća instrukcije iz spremnika, dekodira ih i na temelju toga upravlja aritmetičko-logičkom jedinkom, te ulaznim i izlaznim dijelovima.

Slika 2.1. ilustrira međusobnu povezanost svih tih dijelova. Na slici su označeni tokovi podataka, instrukcija i upravljačkih signala.

Središnji dio računala je *spremnik*. U njega se slijevaju svi podaci i instrukcije koje se unose u računalo preko *ulaznog dijela*, te svi rezultati operacija iz aritmetičko-logičke jedinke. Preko izlaznog dijela rezultati izračunavanja prenose se u okolinu. Iz spremnika *upravljačka jedinka* dohvaća instrukcije i na temelju njih upravlja preostalim dijelovima računala. Upravljačka jedinka određuje koju će operaciju izvesti *aritmetičko-logička jedinka*. Upravljačka jedinka i aritmetičko-logička jedinka spregnute su današnjim računalima u jednu cjelinu i, dodatno, s jednim skupom registara čine *procesor*.



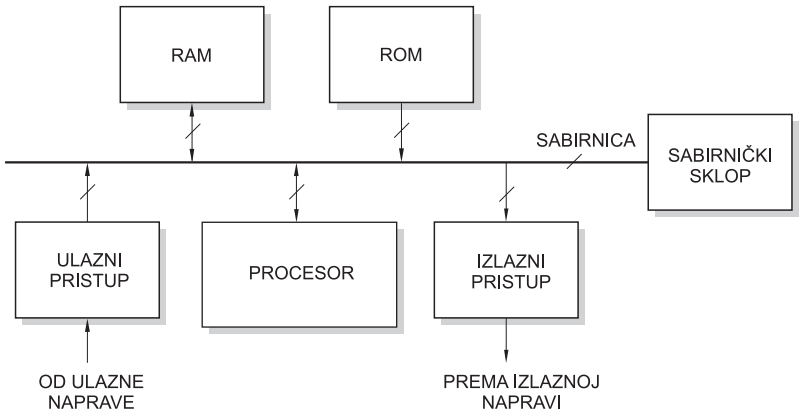

**Slika 2.1.** Funkcijski Von Neumannov model računala

## 2.1.2. Sabirnička građa računala

U funkcijskom prikazu računala na slici 2.1. vidljivo je kako su pojedini dijelovi računala međusobno povezani. Svaka od crta koja predstavlja tok podataka, instrukcija ili upravljačkih signala sastoji se od većeg broja vodiča preko kojih se prenose električki signali kojima se prenose bitovi. Za prijenos svakog bita potreban nam je stoga jedan vodič. Svaka crta u funkcijskom prikazu računala predstavlja prospojni put kojim se istovremeno prenosi potrebni broj bitova. Takvo isprepleteno međusobno povezivanje pojedinih dijelova računala nespretno je i stoga je osmišljen *sabirnički sustav* za njihovo povezivanje. Sabirnica[1] je jedan zajednički snop vodiča na koji su spojeni svi dijelovi računala. Osim vodiča sabirnica ima i svoj sabirnički elektronički sklop, koji pomaže pri ostvarivanju veza.

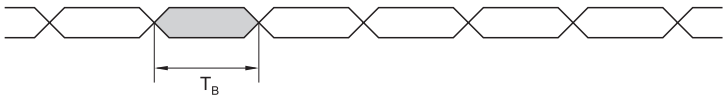
Računalo sa sabirničkim povezivanjem dijelova prikazano je na slici 2.2. Sve potrebne razmjene podataka, instrukcija i upravljačkih signala obavljaju se preko zajedničkih vodiča. Kosa crtica preko crte koja predstavlja sabirnicu označava da je riječ o snopu vodiča. Jasno je da se preko zajedničkih vodiča ne može uspostavljati više istovremenih veza već se mogu obavljati samo pojedinačni prijenosi. Sabirnica se stoga mora naizmjence – s podjelom vremena (engl. *time share*) – koristiti za ostvarenje potrebnih veza između dijelova računala.

[1] U engleskom jeziku je sabirnica dobila naziv *bus*, što je izvorni naziv za autobus koji sabire i razvozi putnike.




**Slika 2.2.** Sklopovlje računala povezano sabirnicom

Vrijeme se na sabirnici dijeli na sabirničke periode ili *sabirničke cikluse*,[2] kao što je prikazano na slici 2.3. U jednom sabirničkom ciklusu trajanja $T_B$ ostvaruje se jedna veza, tj. prijenos jednog sadržaja. Primjerice, sabirnica koja bi imala ciklus trajanja $T_B = 100$ ns omogućila bi da se u jednoj sekundi obavi 10 milijuna prijenosa sadržaja. Ako bi se u jednom ciklusu prenosio samo po jedan bajt, onda bi se preko sabirnice moglo prenijeti 10 MB u sekundi.




**Slika 2.3.** Podjela vremena na sabirničke cikluse

### 2.1.3. Radni ili središnji spremnik računala

Ponovimo još jedanput da je središnji dio svakog računala spremnik. S obzirom na to da u računalnim sustavima postoje raznovrsni spremnici, za spremnik Von Neumannova modela koristi se naziv radni spremnik ili središnji spremnik. *Radni spremnik* dobio je svoj naziv zbog toga što pri izvođenju programa u njemu moraju biti smještene i instrukcije programa i podaci na koje te instrukcije djeluju. Radni spremnik je, isto tako, prolazno odredište i ishodište svih informacija koje kolaju između svih ostalih dijelova računala.

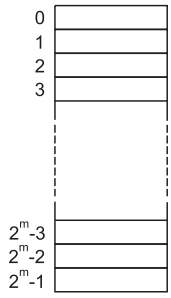
U današnjim su računalima spremnici organizirani tako da se sastoje od bajtova koji se sastoje od osam bitova. Za označavanje bajta služi kratica B. Za bajt se može koristiti i naziv *oktet*[3] ili *osmorka* bitova, ali se time uglavnom podrazumijeva bilo kakva nakupina

[2] Ciklus dolazi od grčkog *kyklos* – krug, kolo, okruglo – a znači neku pojavu koja se redovito ponavlja. Slično značenje ima i naziv perioda, koji ćemo koristiti u opisu rada procesora, ali je prikladnije za sabirničku periodu koristiti naziv ciklus da bi se istaknula razlika prema periodi takta procesora.

[3] U francuskom se za bajt upotrebljava naziv *octet*.

od osam bitova. Isto tako, za nakupinu od četiri bita koristi se naziv *četvorka* bitova ili *kvartet*[4]. Tako se može reći da jedan bajt sadržava jedan oktet ili dva kvarteta bitova, odnosno jednu osmorku ili dvije četvorke bitova.

Do svakog bajta u spremniku može se neposredno pristupiti. Svaki bajt dobiva svoj redni broj. Taj je broj čvrsto povezan s tim bajtom i s pomoću njega se bajt odabire. Stoga se taj broj naziva *adresom* bajta.




**Slika 2.4.** Adrese bajtova

Adrese bajtova unutar spremnika izražavaju se brojevima zapisanima u binarnom obliku. Ako se za zapisivanje adrese koristi $m$ bitova, tada se može zapisati $2^m$ različitih adresa, a adrese se kreću u granicama od 0 do $2^{m-1}$. S brojem $m$ određena je veličina tzv. *adresnog prostora*.

**PRIMJER 2.1.**

Podsjetimo se da se veličina spremnika izražava kao cjelobrojni višekratnik od $2^{10}$ ili $2^{20}$. Naime, pokazuje se da je:

$$2^{10} = 1024 \cong 1000 = 10^3,$$

odnosno:

$$2^{10} \cong 10^3.$$

Kako se za $10^3 = 1000$ koristi kratica k (pisana malim slovom i čita se "kilo"), to je za $2^{10} = 1024$ upotrijebljena kratica K (pisana velikim slovom i čita se "ka"). Tako je, primjerice, uz $m = 16$ moguće adresirati spremnik veličine:

$$2^{16}\,\mathrm{B} = 2^6 \times 2^{10}\,\mathrm{B} = 2^6 \times 1\,\mathrm{KB} = 64\,\mathrm{KB}.$$

Sljedeća veća jedinica za veličinu spremnika je MB (čita se: "megabajt"). Pritom je:

$$1\,\mathrm{MB} = 2^{20}\,\mathrm{B} = 2^{10} \times 2^{10}\,\mathrm{B} = 1024 \times 1\,\mathrm{KB} = 1024 \times 1024\,\mathrm{B} = 1048576\,\mathrm{B},$$

što je približno jednako jedan milijun, odnosno 1 000 000 ili $10^6$, pa se stoga i koristi prefiks *mega*.

[4] U engleskom se za kvartet koristi naziv *nibble* – što znači mali zalogaj.

Isto tako definira se i jedinica GB (čita se: "gigabajt") koja iznosi:

$$1\,\mathrm{GB} = 2^{30}\,\mathrm{B} = 2^{10} \times 2^{20}\,\mathrm{B} = 1024 \times 1\,\mathrm{MB} = 1024 \times 1024 \times 1\,\mathrm{KB}$$
$$= 1024 \times 1024 \times 1024\,\mathrm{B} = 1\,048\,576\,\mathrm{B} = 1\,073\,741\,824\,\mathrm{B},$$

što je približno jednako jednoj milijardi, odnosno $1\,000\,000\,000$ ili $10^9$, pa se stoga i koristi prefiks *giga*.

Tako, primjerice, adrese s $m = 32$ bita mogu adresirati adresni prostor veličine:

$$2^{32}\,\mathrm{B} = 2^2 \times 2^{30}\,\mathrm{B} = 4 \times 1\,\mathrm{GB} = 4\,\mathrm{GB}.$$

### 2.1.4. Rudimentarno računalo, radni spremnik

Za opis nekih osnovnih svojstava računala možemo razmatrati model računala koji se sastoji samo od *procesora* i *radnog spremnika*. Takvo nam rudimentarno računalo može poslužiti za modeliranje ponašanja računalnog sustava, pri čemu pretpostavljamo:

- da je u radnom spremniku smješten strojni oblik programa kao niz instrukcija pohranjenih u uzastopne spremničke lokacije;
- da se u radnom spremniku nalaze i svi ulazni podaci koje će program dohvaćati tijekom izvođenja;
- da će program rezultate koje bude proizvodio tijekom izvođenja pohranjivati u za to predviđene lokacije radnog spremnika.

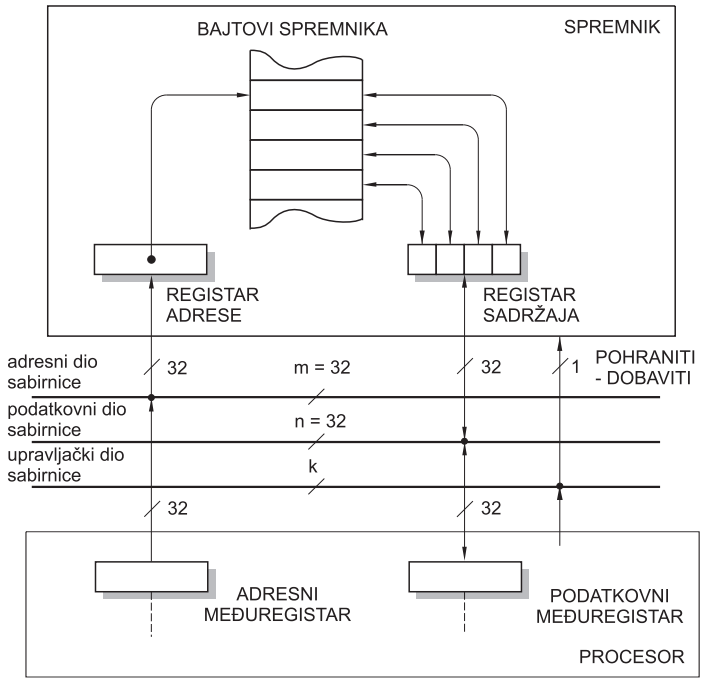
Ovakvo računalo ne bi imalo nikakvu praktičnu vrijednost, jer ne bi imalo veze s okolnim svijetom, ali dobro može modelirati računalni sustav u razdoblju između unošenja ulaznih podataka i prenošenja rezultata izračunavanja u vanjski svijet.

Pogledajmo malo podrobnije kako bi takvo računalo moglo djelovati. Spremnik i procesor u ovakvom bi rudimentarnom računalu mogli biti neposredno povezani, ali je zbog kasnijeg proširenja ovog modela prikladno razmotriti povezivanje preko sabirnice, kao što prikazuje slika 2.5. Sabirnica je ovdje prikazana detaljnije i vidi se da ona ima tri dijela:

- adresni dio sabirnice s $m$ vodiča;
- podatkovni dio s $n$ vodiča;
- upravljački dio s $k$ vodiča.

Na slici je pretpostavljeno da je $m = 32$ i $n = 32$, tj. da koristimo tridesetdvobitovnu spremničku adresu, te da se preko sabirnice istovremeno može prenijeti četiri bajta. Govorimo da je *širina* pristupa do spremnika četiri bajta ili 32 bita.[5]

[5] Danas se u osobnim računalima najčešće koristi tridesetdvobitovna arhitektura s $m = 32$ i $n = 32$, a u snažnijim računalnim sustavima šezdesetčetverobitovna arhitektura s $m = 64$ i $n = 64$. U šezdesetčetverobitnoj arhitekturi može se adresirati adresni prostor veličine $2^{64}\,\mathrm{B} = 2^4 \times 2^{60}\,\mathrm{B} \approx 16 \times 10^{18}\,\mathrm{B} = 16\,\mathrm{EB}$ (čita se: "eksabajt"), a širina pristupa omogućuje istovremeni prijenos osam bajtova.




**Slika 2.5.** Procesor i spremnik povezani na sabirnicu

Procesor je, također, povezan na sabirnicu preko registara. Nazvat ćemo ih međuregistrima prema sabirnici ili, kraće, samo *međuregistrima*. Iz *adresnog međuregistra* postavlja se adresa na adresni dio sabirnice, a *podatkovni međuregistar* ima "dvosmjerno" djelovanje: podaci se mogu prenositi iz procesora u spremniku ili iz spremnika u procesor. Adresa koju procesor postavlja na sabirnicu adresira prvi od četiri bajta koji će biti dobavljeni iz spremnika (u 64-bitovnoj arhitekturi to je adresa prvog od osam bitova) ili pohranjeni u spremnik. Pokazalo se praktičnim da se sadržaji koji imaju više od osam bitova (višekratnik od osam) pohranjuju s *poravnatim adresama* (adrese su cjelobrojni višekratnici od broja bajtova potrebnih za pohranjivanje sadržaja).

U jednom spremničkom ciklusu može se obaviti jedno pohranjivanje ("pisanje") u spremnik ili jedno dobavljanje sadržaja ("čitanje") iz spremnika[6]. U *ciklusu pohranjivanja* događa se sljedeće:

- *procesor* na adresni dio sabirnice postavlja adresu iz svog adresnog međuregistra, signal pohranjivanja na priključak POHRANITI-DOBAVITI, preko upravljačkog dijela sabirnice, te postavlja podatak iz svog podatkovnog registra na podatkovni dio sabirnice;
- *spremnik* prihvaća adresu u svoj registar adrese, prihvaća podatak u registar sadržaja i pohranjuje sadržaj u bajtove spremnika.

[6] U engleskom se jeziku za operacije premještanja sadržaja koriste nazivi *store* – pohraniti ili *write* – pisati, te *load* – nakrcati (misli se na podatkovni međuregistar, odnosno neki drugi od registara procesora) ili *read* – čitati. U uporabi je i naziv *move* – premjestiti, uz koji se naznačuje ishodište i odredište premještanog sadržaja.

U *ciklusu dobavljanja* podatak se iz spremnika u procesor prenosi na sljedeći način:

- *procesor* na adresni dio sabirnice postavlja adresu iz svog adresnog međuregistra i signal dobavljanja na priključak POHRANITI-DOBAVITI, preko upravljačkog dijela sabirnice;
- *spremnik* prihvaća sa sabirnice adresu u svoj registar adrese, prebacuje iz bajtova spremnika sadržaje u registar sadržaja, te postavlja sadržaj na podatkovni dio sabirnice;
- *procesor* prihvaća u svoj podatkovni registar sadržaj s podatkovnog dijela sabirnice.

Već smo spomenuli da spremnički ciklus ima neko konačno trajanje $T_B$. To se vrijeme ne može skratiti ispod neke donje granice zbog fizikalnih ograničenja sklopovlja i vodiča. Kao što smo već spomenuli, uz $T_B = 100$ ns moguće je obaviti 10 milijuna prijenosa. No, ako istovremeno prenosimo četiri bajta, onda se može prenijeti 40 MB u sekundi, a uz širinu prijenosa od osam bajtova brzina prijenosa iznosila bi 80 MB u sekundi.

Ne zaboravimo da su svi dijelovi računala u jednosabirničkom sustavu povezani preko jedne sabirnice. Sve razmjene sadržaja obavljaju se preko te sabirnice, pa se sabirnički ciklusi moraju dijeliti svim sudionicima u "prometu" preko sabirnice, te ona može postati prometno "usko grlo" i ograničiti brzinu rada računala. Vidjet ćemo kako se taj problem donekle ublažava uporabom priručnog spremnika nakon što razmotrimo funkcijski opis procesora.

### 2.1.5. Rudimentarno računalo – procesor

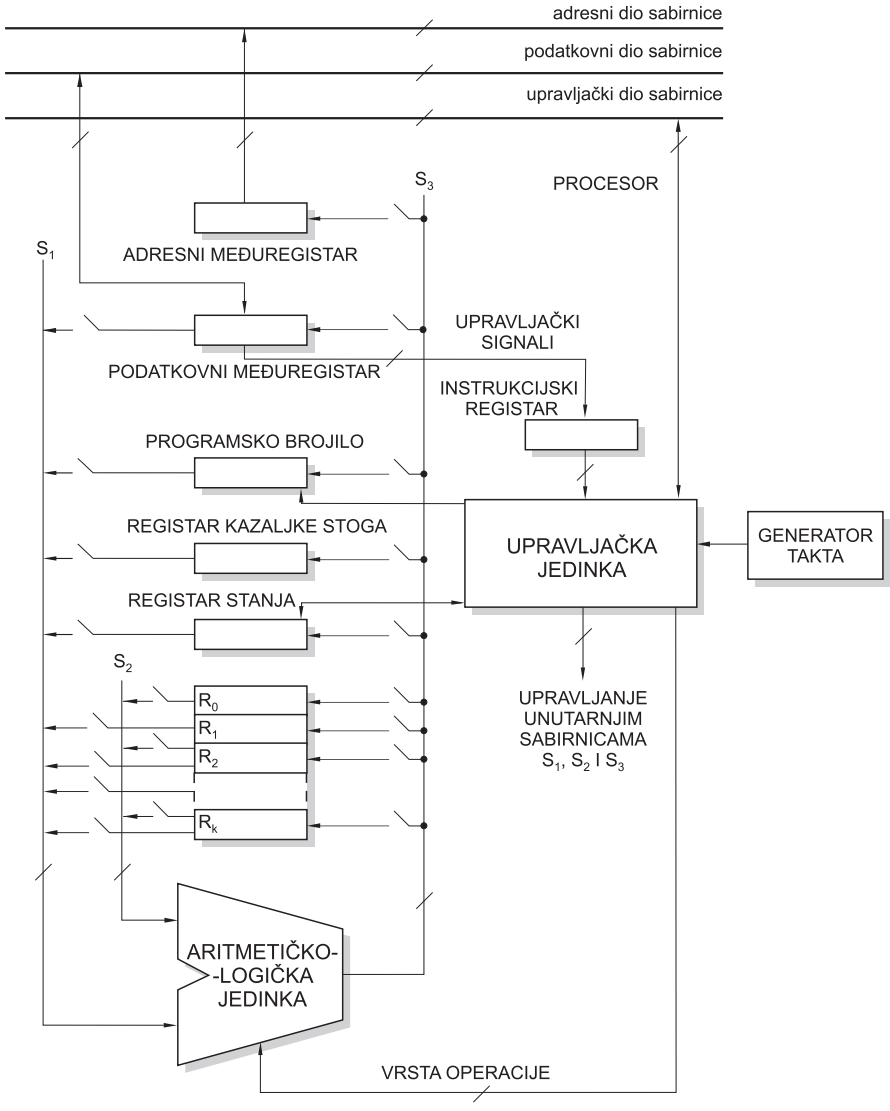
Treba posebno naglasiti da radni spremnik računala (što vrijedi i za ostale spremnike računalnog sustava) služi samo za pohranu nizova bitova. Interpretacija značenja tih nizova bitova događa se izvan spremnika. U rudimentarnom računalu sadržaji spremničkih lokacija zbivat će se u procesoru (interpretacija sadržaja moguća je i u drugim dijelovima računala – primjerice, na zaslonu monitora nizovi bitova pretvaraju se u izgled pojedinog slova). Unutar procesora se sadržaji koji se dobavljaju iz procesora interpretiraju dvojako:

- kao instrukcije strojnog programa računala;
- kao osnovni tipovi podataka.

Procesor adresira instrukcije i podatke na bitno različite načine. Instrukcije se dohvaćaju s adrese na koju pokazuje posebni registar procesora koji nazivamo programskim brojilom. Podaci se adresiraju tijekom izvođenja instrukcije, i to na temelju informacije o smještaju podataka sadržanih u instrukciji. Dio instrukcije koji definira operaciju koja će se izvoditi određuje i način interpretacije niza bitova koji će biti dobavljeni iz spremnika. Prema tome, procesorom je određen tip podataka. Za svaki tip podataka propisani su načini zapisivanja. Na toj strojnoj razini računala dogovoreni su oblici *osnovnih tipova podataka*, možemo reći i *osnovni objekti*. Za njih su na strojnoj razini ostvarene operacije u elektroničkom sklopovlju računala, a u strojnom jeziku postoje i instrukcije koje izazivaju obavljanje tih osnovnih operacija. Svi ostali složeniji tipovi podataka, potrebni za opise

složenijih objekata, moraju se oblikovati od tih osnovnih tipova, a operacije za te objekte izgrađuju se programskim funkcijama.

U Von Neumannovu modelu računala procesor čine aritmetičko-logička jedinka i upravljačka jedinka, te skup registara. Mi ćemo ovdje razmotriti pojednostavljeno djelovanje procesora, i to na vrlo pojednostavljenom modelu koji prikazuje slika 2.6. Stvarni su procesori mnogo složeniji od ovog modela, a i međusobno se prilično razlikuju, ali se njihova djelovanja mogu shvatiti na osnovi razmatranja ovog jednostavnog modela.




**Slika 2.6.** Pojednostavljeni model procesora

Pretpostavit ćemo da su brojevi bitova adrese i podataka jednaki kao što smo pretpostavili i na slici 2.5. U tom slučaju su svi registri jednake duljine i svi prospojni putovi imaju jednak broj vodiča, pa to pojednostavljuje razmatranje.

Osnovna svojstva i ponašanje procesora određeni su *skupom registara* i *skupom instrukcija* koje procesor može obaviti. Registri služe za pohranjivanje svih informacijskih sadržaja koji ulaze i izlaze iz procesora i u njemu se transformiraju. Skup instrukcija određen je izvedbom aritmetičko-logičke i upravljačke jedinke procesora.

Opišimo najprije pojedine registre i njihovu ulogu u radu procesora:

- *Adresni međuregistar* već je opisan u prethodnom odjeljku. On služi za adresiranje spremnika ali i za adresiranje ostalih dijelova računala.
- *Podatkovni međuregistar* je posrednik za razmjenu sadržaja između procesora i ostalih dijelova računala. Svi podaci koji se žele prenijeti iz procesora na sabirnicu prolaze kroz podatkovni međuregistar.
- U *instrukcijski registar* prenosi se instrukcija dobavljena iz spremnika. Bitovi instrukcije dovode se na upravljačku jedinku koja iz njih ustanovljuje koju operaciju procesor treba obaviti.
- *Programsko brojilo* (engl. *Program Counter* (PC)) je registar koji sadržava adresu instrukcije koju sljedeću treba obaviti. Njega upravljačka jedinka automatski povećava tako da se instrukcije pohranjene u spremniku izvode jedna iza druge.
- *Registar kazaljke stoga* (engl. *Stack Pointer* (SP)) služi za poseban način adresiranja spremnika. U spremniku se rezervira posebna skupina uzastopnih bajtova i zapiše u registar kazaljke stoga najvišu adresu te skupine. S pomoću tog registra ostvaruje se pohranjivanje podataka na stog.
- Bitovi *registra stanja* (engl. *Status Register* ili *Condition Code Register* – registar uvjeta, *Flag Register* – registar zastavica) služe za zapisivanje različitih zastavica koje označavaju ispravnost ili neispravnost rezultata i operacija ili neke druge pojave. Vrijednosti tih zastavica jesu uvjeti na temelju kojih upravljački sklop određuje daljnji tijek odvijanja programa.
- Skupina *općih registara* (imenovanih s $R_0$ do $R_K$) služi za pohranjivanje operanada i rezultata, te za pripremanje adresa budućih pristupa do spremnika.

Registri su povezani međusobno i s ostalim dijelovima procesora, posebice s aritmetičko-logičkom jedinkom. U ovom pojednostavljenom modelu procesora predviđa se povezivanje s pomoću *unutarnjih sabirnica* $S_1$, $S_2$ i $S_3$. Preklopke na crtama koje povezuju registre na sabirnicu simboliziraju povezanost registra na sabirnicu. Zamišljene preklopke otvara i zatvara upravljački sklop prema potrebi.

*Upravljačka jedinka* (engl. *Control Unit*) upravlja radom cijelog procesora. Ona dekodira instrukciju dovedenu u instrukcijski registar i na temelju toga povezuje potrebne registre na unutarnje sabirnice, određuje aritmetičko-logičkoj jedinki vrstu operacije koju treba obaviti i automatski prelazi s instrukcije na instrukciju.

Upravljačka jedinka, a time i svi ostali dijelovi računala rade u ritmu koji određuje generator takta. *Generator takta* generira impulse koji pobuđuju elektroničke sklopove. U današnjim procesorima frekvencija generatora ritma kreće se u granicama od nekoliko stotina MHz pa do nekoliko GHz. (Podsjetimo se da, primjerice, frekvencija generatora takta od 1 GHz određuje da je perioda pobudnih impulsa $T_S = 1$ ns.) Sve pojave unutar procesora sinkronizirane[7] su s generatorom takta. Generator takta, dakle, određuje brzinu rada procesora.

Konačno, *aritmetičko-logička jedinka* (engl. *Arithmetic-Logic Unit*) obavlja, kao što joj to i ime kaže, aritmetičke i logičke operacije s operandima koji se u našem modelu procesora dovode na njezin ulaz preko sabirnica $S_1$ i $S_2$ i rezultat postavlja na sabirnicu $S_3$. Upravljačka jedinka koja je dekodirala instrukciju preko upravljačkih vodiča određuje aritmetičko-logičkoj jedinki operaciju koju ona mora obaviti. Isto tako, upravljačka jedinka iz instrukcije ustanovljuje odakle dolaze operandi i gdje treba pohraniti rezultate, pa u skladu s tim povezuje na unutarnje sabirnice pojedine od registara, tj. zatvara odgovarajuće preklopke prema sabirnicama $S_1$, $S_2$ i $S_3$. U našem pojednostavljenom modelu jedinka može i "propustiti" bez promjene sadržaj iz jednog od registara povezanog na sabirnicu $S_1$ do sabirnice $S_3$ kako bi sadržaji mogli prenositi iz jednog u drugi registar.

Nakon što smo ukratko upoznali sastavne dijelove procesora ustanovimo kako on radi. Pretpostavljamo da su u spremniku pohranjene instrukcije programa, i to onim redoslijedom kako ih treba izvoditi. Taj niz instrukcija zovemo *strojnim programom*. U programsko brojilo (PC) mora se postaviti adresa prve instrukcije strojnog programa.

Procesor se može promatrati kao automat koji nakon uključivanja trajno izvodi instrukciju za instrukcijom strojnog programa. Ponašanje sklopovlja procesora pri izvođenju svake instrukcije može se opisati na sljedeći način[8]:

```
ponavljati {
     dohvatiti iz spremnika instrukciju na koju pokazuje programsko brojilo;
     dekodirati instrukciju, odrediti operaciju koju treba izvesti;
     povećati sadržaj programskog brojila tako da pokazuje na sljedeću instrukciju;
     odrediti odakle dolaze operandi i gdje se pohranjuje rezultat;
     operande dovesti na aritmetičko-logičku jedinku, izvesti zadanu operaciju;
     pohraniti rezultat u odredište;
}
dok je (procesor uključen);
```

Procesor je, dakle, automatski izvoditelj programa koji obavlja instrukcije onim redom kojim su one smještene u spremniku. Ako taj redoslijed izvođenja treba narušiti, tj. iz-

[7] Sinkronizirati dolazi od grčkog *syn* – s i *chronos* – vrijeme i znači: vremenski uskladiti.

[8] Ovaj opis slijedi pravila pisanja uvedenih u programskom jeziku C. Svaka "instrukcija", zaključena sa znakom točka-zarez označava neku osnovnu aktivnost mikroelektroničkog sklopovlja. Cijeli prolaz kroz petlju obavlja se tijekom izvođenja jedne instrukcije. Na jednak način opisivat ćemo i mnogo složenije aktivnosti operacijskog sustava gdje će nam pojedine "instrukcije" obilježavati izvođenje cijelih programskih odsječka koji se mogu sastojati od nekoliko stotina ili nekoliko tisuća instrukcija.

vesti neke "skokove" pri izvođenju programa, onda se unutar instrukcije mora prisilno promijeniti sadržaj programskog brojila.

Iz gornjeg je opisa vidljivo da se izvođenje instrukcije može podijeliti u tri faze:

- U prvoj fazi *dohvata instrukcije* (engl. *fetch*) događa se sljedeće:
  - — sadržaj programskog brojila prebacuje se u adresni međuregistar;
  - — upravljačka jedinka pokreće aktivnost dohvata instrukcije iz spremnika;
  - — instrukcija dolazi u podatkovni međuregistar i iz njega u instrukcijski registar.
- U drugoj fazi *dekodiranja instrukcije* (engl. *decode*) upravljačka jedinka:
  - — utvrđuje na temelju dijela bitova instrukcije (koje nazivamo *operacijskim kodom*) operaciju koju treba provesti;
  - — povećava sadržaj programskog brojila tako da pokazuje na sljedeću instrukciju;
  - — šalje upravljačke signale aritmetičko-logičkoj jedinki kako bi ona "znala" koju operaciju treba obaviti;
  - — na temelju dijela bitova instrukcije (koje nazivamo *adresnim dijelom instrukcije*) ustanovljuje iz kojih registara dolaze operandi i gdje treba pohraniti rezultat, te "zatvara" odgovarajuće preklopke na unutarnjim sabirnicama;
  - — ako adresni dio instrukcije određuje da operand dolazi iz spremnika, onda se adresa operanda prebacuje u adresni međuregistar i pokreće dobavljanje operanda iz spremnika, pa će se taj operand naći u podatkovnom međuspremniku i iz njega dovesti u aritmetičko-logičku jedinku.
- U trećoj se fazi *obavljanja operacije* (engl. *execute*) događa sljedeće:
  - — aritmetičko-logička jedinka obavlja zadanu operaciju i pohranjuje rezultat preko sabirnice $S_3$ u odredište;
  - — vrijednosti pojedinih zastavica koje ovise o dobivenom rezultatu pohranjuju se u registar stanja.

### 2.1.6. Brzina rada procesora, priručni spremnik

Gornji opis ponašanja procesora ukazuje da izvođenje jedne instrukcije traje određeno vrijeme. S obzirom na to da je rad procesora sinkroniziran s generatorom takta, trajanje instrukcije je cjelobrojni višekratnik periode takta. Prema tome, isti procesor može raditi različitim brzinama ako se promijeni frekvencija generatora takta. Za svaki je procesor određena gornja granica te frekvencije, no odabrana frekvencija rada ovisi i o okolnom sklopovlju računala (osobito o mogućoj brzini prijenosa podataka preko sabirnice).

Često se brzinsko svojstvo procesora iskazuje brojem instrukcija koje može izvesti u sekundi. To ovisi o tome koliko perioda takta procesor troši za izvođenje instrukcija. S obzirom na to da različite instrukcije mogu "potrošiti" različit broj perioda takta uzima se neki prosječni broj perioda odgovarajuće mješavine različitih instrukcija. Primjerice,

ako je perioda takta $T_S = 10$ ns i za izvođenje instrukcija procesor troši 5 perioda, prosječno trajanje jedne instrukcije je 50 ns, tj. u jednoj sekundi može se izvesti 20 milijuna instrukcija. Kaže se da je računalna moć procesora 20 MIPS (milijuna instrukcija po sekundi)[9].

Međutim, brzina rada računala ne ovisi samo o brzini procesora, već i o ostalom računalnom sklopovlju. S obzirom na to da se instrukcije i podaci dobavljaju iz spremnika, instrukcije se ne mogu dobavljati brže nego što to dopušta sabirnica. U jednom sabirničkom ciklusu može se dobaviti samo onoliko bajtova koliko to dopušta širina pristupa. Tako bi sabirnica sa sabirničkim ciklusom od $T_B = 100$ ns dopuštala izvođenje 10 milijuna instrukcija u sekundi, iako procesor ima moć od 20 MIPS. Štoviše, ustanovili smo da neke instrukcije mogu izazvati i višekratni pristup do spremnika. Time se brzina rada i dalje smanjuje. Sabirnica sa svojom ograničenom brzinom prijenosa bitno ograničava mogućnosti procesora.

Graditelji računala nastoje na razne načine ubrzati rad računala. Ovdje nije mjesto razmatranju takvih poboljšanja. Zainteresirani čitatelj upućuje se na literaturu u kojoj su takva arhitektonska poboljšanja temeljito opisana[10]. Međutim jedno od poboljšanja toliko je rasprostranjeno da ga se i u ovom pojednostavljenom prikazu ne može preskočiti. Naime, neposredno uz procesor dodaje se jedan manji spremnik (on čak može biti izveden na istom procesorskom čipu), tzv. priručni spremnik (engl. *cache* – spremište, tajno skrovište), koji ima dopuštenu brzinu pristupa sukladnu brzini rada procesora. Instrukcije i podaci koji se nalaze u tom spremniku dostupni su, dakle, u istom vremenu kao i registri procesora (u nekim arhitekturnim rješenjima predviđeni su razdvojeni priručni spremnici za instrukcije i podatke). Tada se procesor u radu može približiti svojim maksimalnim brzinskim mogućnostima. Priručni spremnik može imati samo ograničenu veličinu i posebnim se mehanizmima treba osigurati da se kopija sadržaja onog dijela glavnog spremnika koji se upravo koristi u njega premjesti. Tijekom izvođenja programa razni dijelovi glavnog spremnika premještaju se u priručni spremnik i obrnuto, iz priručnog spremnika u glavni. Sve se to događa "skriveno" (otuda i engleski naziv za priručni spremnik). Strojni program i način adresiranja ne moraju se mijenjati u odnosu na izvođenje bez priručnog spremnika. Jedina vidljiva razlika je u trajanju izvođenja programa. S obzirom na to da je glavnina mehanizama za djelovanje sustava riješena sklopovski, u ovom se udžbeniku time nećemo posebno baviti.

### 2.1.7. Instrukcijski skup procesora

Izvedba aritmetičko-logičke jedinke i upravljačke jedinke određuje *instrukcijski skup* (engl. *Instruction Set*) nekog procesora. Već smo rekli da upravljačka jedinka mora dekodirati instrukcije i na temelju njihova sadržaja narediti aritmetičko-logičkoj jedinki koju operaciju treba obaviti te povezati potrebne registre na unutarnje sabirnice i postavljati upravljačke signale ostalim dijelovima računala. Aritmetičko-logička jedinka mora moći provesti zadanu operaciju.

[9] Izvorno je MIPS kratica od engleskog *Milion Instructions per Second.*
[10] Vidi: S. Ribarić, Napredne arhitukture RISC i CISC procesora, Školska knjiga, Zagreb, 1995.