Danijela Simić Filip Marić Predrag Janičić

Milan Banković Milena Vujošević Janičić Ve-

sna Marinković Mladen Nikolić Mirko Spasić Sa-

na Stojanović Đurđević Ivana Tanasijević

UVOD U INFORMATIKU

Beograd
2025.

Autori:
dr Danijela Simić, docent na Matematičkom fakultetu u Beogradu
dr Filip Marić, redovni profesor na Matematičkom fakultetu u Beogradu
dr Predrag Janičić, redovni profesor na Matematičkom fakultetu u Beogradu
dr Milan Banković, docent na Matematičkom fakultetu u Beogradu
dr Milena Vujošević Janičić, vandredni profesor na Matematičkom fakultetu u Beogradu
dr Mladen Nikolić, vandredni profesor na Matematičkom fakultetu u Beogradu
dr Mirko Spasić, docent na Matematičkom fakultetu u Beogradu
dr Sana Stojanović Đurđević, docent na Matematičkom fakultetu u Beogradu
dr Ivana Tanasijević, docent na Matematičkom fakultetu u Beogradu

UVOD U INFORMATIKU

Izdavač: Matematički fakultet Univerziteta u Beogradu
Studentski trg 16, 11000 Beograd

Recenzenti:
??
??

Obrada teksta i ilustracije: *autori* (osim za slike nabrojane na kraju knjige)
Dizajn korica: *autori*

Sadržaj

Predgovor

Ovo je materijal za predmet Uvod u informatiku na prvoj godini smera *Informatika* na Matematičkom fakultetu Univerziteta u Beogradu) Nadamo da izbor sadržaja i način prezentovanja mogu da budu zanimljivi ne samo studentima, već i svima drugima koje interesuje ova oblast računarstva.

Bićemo zahvalni čitaocima na svim ispravkama, sugestijama i komentarima koje nam pošalju.

- Kratka istorija informatike i informaciono-komunikacionih tehnologija – Filip Marić i Predrag Janičić
- Hardver i softver - Filip Marić, Predrag Janičić i Danijela Simić
- Digitalizacija - Filip Marić, Predrag Janičić i Danijela Simić
- Algoritmi i izračunljivost - Predrag Janičić
- Sistemski softver - Milan Banković
- Računarske mreže - Mirko Spasić
- Programski jezici i prevodioci - Milena Vujošević Janičić
- Proces razvoja softvera - Danijela Simić
- Baze podataka - Ivana Tanasijević
- Matematika i informatika - Filip Marić
- Veštačka inteligencija - Mladen Nikolić i Predrag Janičić
- Računarska grafika - Vesna Marinković

Danijela Simić, Filip Marić, Predrag Janičić, Milan Banković, Milena Vujošević Janičić, Mladen Nikolić, Mirko Spasić, Sana Stojanović Đurđević i Ivana Tanasijević

Beograd, mart 2024.

GLAVA

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.

1.1 Procesor (CPU)

1.1.1 Osnovne komponente procesora

Procesor predstavlja centralnu komponentu svakog računarskog sistema zasnovanog na Fon Nojmanovom modelu arhitekture. Njegova osnovna funkcija je da upravlja tokom instrukcija i izvršava operacije nad podacima, čineći time proces obrade informacija mogućim. U savremenim računarima svi osnovni delovi procesora objedinjeni su u jedinstvenu fizičku celinu — *centralnu procesorsku jedinicu* (engl. *Central Processing Unit, CPU*), koja je implementirana kao mikroprocesor na jednom integrisanom kolu.

Procesor se sastoji od tri osnovna podsistema:

- *aritmetičko-logičke jedinice* (engl. *Arithmetic Logic Unit, ALU*), zadužene za izvođenje aritmetičkih (sabiranje, oduzimanje, množenje, deljenje) i logičkih operacija (konjunkcija, disjunkcija, negacija, poređenje);
- *kontrolne jedinice* (engl. *Control Unit, CU*), koja interpretira i dekodira instrukcije, koordinira izvršavanje i upravlja prenosom podataka između delova procesora i memorije;
- skupa *registara* — malih, vrlo brzih memorijskih ćelija koje privremeno čuvaju operande, adrese i međurezultate obrade. Obično fiksirane širine (8 bitova, 16 bitova, 32 bita, 64 bita).

U najranijim implementacijama komunikacija između ALU i memorije obavljala se preko posebnog registra nazvanog *akumulator*. Savremeni procesori, međutim, raspolažu većim brojem opštih i specijalizovanih registara, čime se značajno smanjuje broj pristupa glavnoj memoriji i ubrzava obrada podataka.

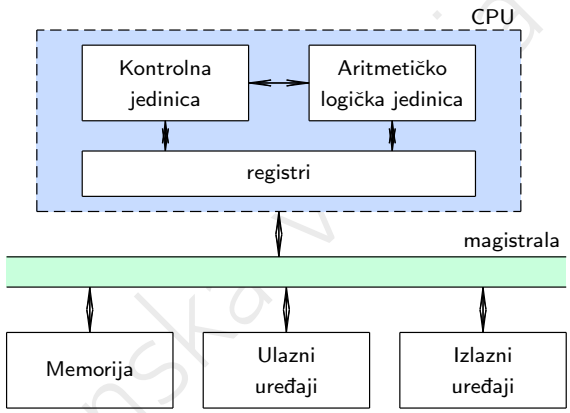
Kontrolna jedinica procesora izvršava tzv. *ciklus instrukcije* koji se sastoji iz tri osnovne faze: *dohvatanja* (engl. *fetch*), *dekodiranja* (engl. *decode*) i *izvršavanja* (engl. *execute*). Tokom svake faze procesor čita instrukciju iz memorije, određuje operaciju koju treba obaviti i sprovodi odgovarajuću aritmetičku ili logičku akciju nad podacima u registrima.

Primer 1.1. Ciklus instrukcije

Neka je u registru `R1` *vrednost 5, a u registru* `R2` *vrednost 3.*
Instrukcija `ADD R1, R2` *znači da se sabira sadržaj registra* `R2` *sa* `R1`*, a rezultat se ponovo upisuje u* `R1`*.*
Kontrolna jedinica redom dohvata, dekodira i prosleđuje ALU instrukciju, koja izvršava sabiranje i ažurira sadržaj registra `R1` *na vrednost 8.*

Brzina rada procesora izražava se najčešće kroz broj instrukcija koje može izvršiti u sekundi, označen kao *MIPS* (engl. *Million Instructions Per Second*), odnosno u broju operacija u pokretnom zarezu po sekundi, *FLOPS* (engl. *Floating Point Operations per Second*). Moderni procesori ostvaruju performanse reda veličine desetina gigaflopsa (10^9 operacija u sekundi), dok se u vrhunskim serverskim i grafičkim čipovima dostižu i teraflopi (10^{12} operacija u sekundi).

Dodatni pokazatelji arhitektonske moći procesora su broj *jezgara* (engl. *cores*), širina reči (32 ili 64 bita) i *radni takt* (engl. *clock rate*), koji se izražava u gigahercima (GHz). Viši takt omogućava brže izvršavanje instrukcija, ali povećava potrošnju energije i disipaciju toplote, zbog čega savremeni procesori teže kompromisu između performansi i energetske efikasnosti.



Slika 1.1: Shema računara Fon Nojmanove arhitekture

1.1.2 Cevovod i tehnike ubrzanja

Cevovod (engl. *pipelining*) predstavlja ključnu tehniku ubrzanja savremenih procesora, zasnovanu na **preklapanju faza obrade instrukcija**. Umesto da svaka instrukcija čeka završetak prethodne, različite faze više instrukcija izvršavaju se istovremeno, čime se značajno povećava *protok* (engl. *throughput*) sistema.

Primer 1.2. Jednostavni cevovod sa pet faza

Savremeni RISC procesori, poput RISC-V ili ARM, često koriste klasičan petostepeni cevovod*:*

- ***IF (Instruction Fetch)*** *– čitanje instrukcije iz memorije;*
- ***ID (Instruction Decode)*** *– dekodiranje i čitanje registara;*
- ***EX (Execute)*** *– aritmetičko-logičke operacije ili izračunavanje adrese;*
- ***MEM (Memory)*** *– pristup podacima u memoriji;*
- ***WB (Write-Back)*** *– upis rezultata u registar.*

Dok jedna instrukcija ulazi u fazu dekodiranja, druga se izvršava, treća pristupa memoriji, a četvrta upisuje rezultat. Time se, idealno, ostvaruje ubrzanje približno broju faza cevovoda.

Cevovod povećava broj izvršenih instrukcija u jedinici vremena, ali *ne smanjuje vreme izvršavanja pojedinačne instrukcije*. Štaviše, dodatna kontrolna logika i registri između faza unose mali vremenski trošak (*pipeline overhead*). Zbog toga se u praksi ostvaruje ubrzanje nešto manje od teorijskog maksimuma.

Balansiranje faza

Efikasnost cevovoda zavisi od ravnoteže dužina faza – ukupno vreme ciklusa određeno je najsporijom fazom. Ako je neka faza znatno duža, ona postaje *usko grlo* performansi. Cilj projektanta je da sve faze traju približno isto.

1.1.2.1 Vrste prepreka u cevovodu

Idealno izvršavanje je narušeno pojavom **hazarda**, situacija koje onemogućavaju da sledeća instrukcija započne svoju fazu u predviđenom ciklusu. U praksi se izdvajaju tri osnovne klase:

- **Strukturni hazard** – nastaje kada hardver nema dovoljan broj resursa za istovremeno izvršavanje svih faza (npr. zajednička memorija za instrukcije i podatke).
- **Podatkovni hazard** – javlja se kada jedna instrukcija zavisi od rezultata prethodne (*read-after-write* problem). Rešenja uključuju *prosleđivanje rezultata* (engl. *forwarding*) i *mehanizme zadrške* (*interlocks*).
- **Kontrolni hazard** – posledica grananja: dok procesor ne zna ishod skoka, ne zna ni koju instrukciju sledeću da učita. Savremeni procesori ublažavaju ovaj problem *predikcijom grananja* i *spekulativnim izvršavanjem*.

U idealnim uslovima cevovod može ubrzati izvršavanje do broja svojih faza, ali svaka pojava hazard-a umanjuje taj efekat. Zbog toga su upravo mehanizmi detekcije i ublažavanja hazard-a osnov savremenog dizajna procesora.

Kada su faze pravilno balansirane, a hazardi minimalizovani, cevovod omogućava procesoru da istovremeno izvršava više instrukcija.

1.1.2.2 Paralelizam na nivou instrukcija (ILP)

Paralelizam na nivou instrukcija (engl. *Instruction-Level Parallelism, ILP*) predstavlja sposobnost procesora da izvršava više nezavisnih instrukcija istovremeno, iskorišćavajući preklapanje u cevovodu. Dok cevovod povećava protok obrade, ILP meri koliko instrukcija može zaista biti izvršeno paralelno bez narušavanja ispravnosti programa.

Suština ILP-a leži u pronalaženju instrukcija koje nemaju međusobne zavisnosti. Kada se dve instrukcije ne oslanjaju na iste podatke niti kontrolišu jedna drugu, procesor ih može istovremeno dekodirati i izvršavati. Ograničavajući faktori su *podatkovne*, *imenske* i *kontrolne zavisnosti*, koje određuju redosled instrukcija i sprečavaju prekomerno preklapanje. Ako procesor ne prepozna zavisnosti pravilno, može doći do pogrešnih rezultata ili izuzetaka tokom izvršavanja.

Vrste zavisnosti

- **Podatkovna zavisnost (RAW)** – instrukcija koristi rezultat prethodne, pa mora čekati da se on izračuna.
- **Imenska zavisnost (WAW, WAR)** – dve instrukcije koriste isto ime registra ili memorijsku lokaciju, iako se podaci ne prenose.
- **Kontrolna zavisnost** – određuje redosled izvršavanja u prisustvu grananja.

Kompajleri i procesori zajedno nastoje da povećaju ILP. Savremene arhitekture koriste *dinamičke tehnike* otkrivanja paralelizma u samom hardveru (spekulacija, predikcija grananja), dok kompajleri primenjuju *statičke transformacije* tokom prevođenja koda. Dve osnovne kompajlerske metode su:

- **Promena redosleda nezavisnih instrukcija** (eng. instruction reodering),
- **Proširivanje tela petlje** (eng. loop unrolling) – kako bi se smanjio broj grananja i povećao broj nezavisnih instrukcija koje se mogu izvršavati paralelno.

Primer 1.3. Primer transformacije petlje

Petlja koja sabira dva vektora:

```
\texttt{for (i = 0; i < n; i++)  x[i] = x[i] + y[i];}
```

može se „odmotati" (engl. unroll) tako da se u jednoj iteraciji obrade, recimo, četiri elementa istovremeno. Time se smanjuje učestalost grananja i povećava broj instrukcija koje se paralelno izvršavaju.

U praksi, ILP je ograničen zavisnostima u programu i brojem resursa procesora. Što je više nezavisnih instrukcija u osnovnom bloku koda, to je veći stepen paralelizma koji arhitektura može iskoristiti.

1.1.2.3 Predikcija skokova

Grananja uvode *kontrolne hazarde*: da bi se očuvale kontrolne zavisnosti, cevovod mora da čeka ishod skoka, što stvara zastoje i povećava CPI. *Loop unrolling* smanjuje broj grananja, ali ključno je i *predviđanje* (engl. *branch prediction*), kojim procesor pogađa ishod i *uslovno* izvršava instrukcije duž predviđene putanje na osnovu pretpostavki. Ukoliko se ispostavi da je predviđanje bilo pogrešno, efekti predviđanja se poništavaju, a cevovod se prazni.

Osnovni prediktor. Najjednostavniji oblik predikcije je **2-bitni zasićujući brojač** po svakoj grani (tzv. *(0,2) prediktor*), koji pamti poslednja dva ishoda i prema njima odlučuje da li će naredni skok biti „uzet" ili „neuzet". Ova šema koristi samo lokalne informacije o ponašanju konkretne grane.

Korelacioni prediktori. Tačnost se može povećati ako predviđanje jedne grane zavisi i od *globalne istorije* prethodnih grananja. U opštem slučaju, (m, n)-prediktor koristi m bita globalne istorije da odabere jedan od 2^m prediktora sa n-bitnim brojačima. Tipičan primer je *gshare* prediktor: indeks se dobija `XOR` kombinacijom niskih bitova adrese i globalne istorije, čime se dobija visoka tačnost uz minimalan hardverski trošak.

Turnirski (hibridni) prediktori. Savremeniji turnirski prediktori kombinuju lokalni i globalni pristup. Selektor, takođe zasnovan na 2-bitnom brojaču, adaptivno bira „bolji" prediktor za svaku granu, učeći iz prethodnih uspeha i neuspeha. Ova strategija obezbeđuje stabilnu tačnost pri različitim obrascima grananja.

Označeni hibridi (TAGE). Najtačniji današnji prediktori su **TAGE** (engl. *TAgged GEometric*) prediktori, koji koriste više tabela indeksiranih istorijama različite dužine. Svaka tabela sadrži *tag* koji potvrđuje poklapanje obrasca i dodatno polje korišćenosti koje eliminiše zastarele unose. Na taj način se postiže veća tačnost bez eksponencijalnog rasta memorijskih zahteva.

Povezanost tačnosti predikcije i performansi

Doprinos kontrolnih hazarda ukupnom prosečnom broju ciklusa po instrukciji (eng. Cycles Per Instruction, CPI) može se aproksimirati izrazom:

$$CPI = CPI_{ideal} + P_{mispredict} \cdot Penalty_{branch}$$

gde je $P_{mispredict}$ verovatnoća pogrešne predikcije, a $Penalty_{branch}$ broj ciklusa izgubljenih po promašaju. Kod dubokih cevovoda ova vrednost može dostići i desetine ciklusa.

Dakle, formula kaže: Stvarni CPI raste onoliko koliko često procesor pogreši u predikciji skoka i koliko ga svaka pogrešna odluka košta u ciklusima.

Pri pokretanju programa, prediktor grananja obično se inicijalizuje jednostavnim pravilom, dok tokom rada procesor „nauči" tipične obrasce programa.

Savremene tehnike ubrzanja: višestruko izdavanje i dinamičko raspoređivanje. Savremeni procesori ne ograničavaju se na obradu jedne instrukcije po ciklusu, već primenjuju tzv. *višestruko izdavanje instrukcija* (eng. multiple issue). U takvoj arhitekturi, više nezavisnih instrukcija može se dekodirati i započeti u istom taktu, čime se povećava *propusnost* cevovoda i efikasnije koriste funkcionalne jedinice. Da bi se ovo ostvarilo bez ručnog planiranja od strane kompajlera, procesor koristi *dinamičko raspoređivanje instrukcija* (eng. dynamic scheduling), koje u toku izvršavanja analizira zavisnosti između instrukcija i, kad god je moguće, izvršava ih van redosleda definisanog programom. Time se izbegavaju zastoji, a paralelizam se koristi u meri koju dopušta konkretni tok podataka. U kombinaciji sa predikcijom skokova i promena imena registara, ove tehnike omogućavaju modernim superskalarnim procesorima da u jednom ciklusu započnu i do desetak mikro-operacija, čime se realni CPI približava idealnom, a cevovod održava stalno popunjenim.

1.1.3 Procesori sa više jezgara i paralelizam

Kako su fizička ograničenja (potrošnja energije, disipacija toplote i složenost pronalaženja dodatnog *instruction-level* paralelizma) usporila dalji rast performansi jednog jezgra, razvoj mikroprocesora prešao je u novu fazu: paralelizam na nivou niti (*Thread-Level Parallelism*, TLP). Umesto povećanja brzine pojedinačne niti, savremeni procesori ostvaruju veći ukupni učinak tako što istovremeno izvršavaju više niti programa – bilo unutar istog procesa, bilo između više procesa.

Za razliku od ILP-a, koji pronalazi nezavisne instrukcije u okviru jedne programske niti, TLP koristi više nezavisnih tokova izvršavanja. Svaka nit poseduje **svoj programski brojač i registre, ali može deliti memorijski prostor sa drugim nitima**. Time se omogućava paralelno izvođenje zadataka, bilo da potiču iz istog programa (npr. delovi petlje), bilo iz različitih aplikacija. Operativni sistem i kompajler zajedno identifikuju i planiraju niti, dok hardver obezbeđuje njihovo istovremeno izvršavanje na različitim jezgrima.

ILP naspram TLP-a

ILP povećava brzinu izvršavanja jedne niti otkrivanjem nezavisnih instrukcija, dok TLP povećava propusnost sistema izvođenjem više niti istovremeno. ILP zahteva kompleksne tehnike predviđanja i dinamičkog raspoređivanja, dok TLP koristi više fizičkih jezgara i niti da ostvari sličan efekat na višem nivou apstrakcije.

Savremeni procesori sa više jezgara (*multicore*) kombinuju ILP i TLP: svako jezgro interno koristi cevovod i višestruko izdavanje instrukcija, dok više jezgara zajedno omogućava paralelno izvršavanje niti. Većina današnjih procesora (Intel, AMD, ARM) su *procesori sa deljenom memorijom* (eng. shared-memory multiprocessori) — više jezgara deli zajedničku fizičku memoriju i pristupa joj kroz hijerarhiju keševa (privatni L1/L2 i deljeni L3).

Dva osnovna pristupa su:

- **Simetrični (SMP)** — sva jezgra imaju jednak pristup memoriji, pa je vreme pristupa uniformno.
- **Distribuirani (NUMA)** — svako jezgro ima lokalnu memoriju sa bržim pristupom, dok pristup memoriji drugih jezgara traje duže.

U realnim sistemima, komunikacija među jezgrima i sinhronizacija niti zahtevaju posebne mehanizme (npr. *koherentnost keš memorije* (eng. cache coherence) i *konzistentnost memorije* (eng. memory consistency)). Iako arhitekture sa više jezgara omogućavaju znatno veće performanse i energetsku efikasnost, ukupno ubrzanje ograničeno je zakonom Amdala – **čak i mali deo koda koji se ne može paralelizovati može postati glavno usko grlo.**

Primer: kombinovanje ILP i TLP

Procesori kao što su **Intel Core**, **AMD Ryzen** i **ARM big.LITTLE** arhitekture kombinuju višestruko izdavanje instrukcija, više cevovoda i više jezgara, uz podršku za više niti po jezgru (*Simultaneous Multithreading* – SMT). Time se na nivou čipa ostvaruje paralelizam od nekoliko desetina nezavisnih niti koje dele zajedničku memoriju i komunikacione magistrale.

1.1.4 Integracija grafičkih procesora i akceleratori

Savremeni procesorski sistemi sve češće kombinuju klasične procesore opšte namene (CPU) sa specijalizovanim jedinicama za paralelno računanje, čineći tzv. *heterogenu arhitekturu*. Najpoznatiji predstavnici ove klase su **grafički procesori** (engl. *Graphics Processing Unit*, *GPU*) i **akceleratori za neuronske mreže** (TPU, NPU, DNN). 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 i zato su se pokazali izuzetno efikasnim i u naučnim, multimedijalnim i aplikacijama gde se koristi mašinsko učenje, gde dominira *paralelizam na nivou podataka* (eng. data-level parallelism, DLP).

Za razliku od CPU-a koji istovremeno izvršava mali broj kompleksnih niti, GPU poseduje stotine ili hiljade jednostavnih jezgara koja istovremeno primenjuju istu instrukciju nad različitim podacima – model poznat kao *SIMD* (Single Instruction, Multiple Data) ili njegova varijanta *SIMT* (Single Instruction, Multiple Threads). Svaka grupa niti, izvršava se sinhrono na skupu paralelnih aritmetičkih jedinica. Na taj način GPU ostvaruje izuzetno visok stepen propusnosti (engl. *throughput*), što ga čini pogodnim za računanja nad velikim nizovima podataka.

Programiranje GPU-a zasniva se na modelima kao što su **CUDA** (NVIDIA) i **OpenCL** (otvoreni standard). U njima programer eksplicitno definiše broj niti i njihovu organizaciju u blokove, dok hardver automatski upravlja njihovim rasporedom i sinhronizacijom. Da bi GPU postigao punu efikasnost, niti u okviru jednog bloka treba da pristupaju povezanim delovima memorije tj. niti u okviru jednog bloka obrađuju uzastopne (susedne) elemente, čime se omogućava efikasno objedinjavanje memorijskih zahteva (engl. memory coalescing).

U savremenim računarima CPU i GPU su povezani interfejsima velike propusnosti (npr. NVLink, PCIe) i mogu deliti jedinstveni adresni prostor zahvaljujući tzv. ujedinjenoj memoriji (engl. *unified memory*). Ovaj pristup omogućava transparentan prelazak podataka između CPU i GPU memorije, olakšavajući programiranje i smanjujući potrebu za ručnim kopiranjem.

Razvoj veštačke inteligencije i dubokih neuronskih mreža doveo je do pojave specijalizovanih procesora namenjenih obradi modela mašinskog učenja. Ovi procesori, poznati kao *akceleratori za duboke neuronske mreže* (engl. *Deep Neural Network Accelerators, DNN accelerators*), projektovani su tako da efikasno izvršavaju veliki broj jednostavnih operacija nad matricama – osnovni oblik računanja u neuronskim mrežama.

Osobine DNN akceleratora

Tipične operacije koje se ponavljaju u neuronskim mrežama su množenje matrica, konvolucije i nelinearne funkcije aktivacije (na primer, ReLU, sigmoid, *tanh*). Zbog toga akceleratori poseduju veliki broj jednostavnih aritmetičkih jedinica (ALU) i brze lokalne memorije koje smanjuju potrebu za pristupom spoljašnjoj memoriji.

Najpoznatiji predstavnik ovih specijalizovanih procesora je Google Tensor Processing Unit (TPU). TPU je dizajniran kao domen-specifičan akcelerator (*Domain-Specific Architecture, DSA*) za fazu *inferencije* – odnosno korišćenje već istreniranih neuronskih mreža. Srž TPU-a čini sistolički niz (engl. *systolic array*) od 256×256 8-bitnih aritmetičkih jedinica koje paralelno izvršavaju operacije množenja i sabiranja.

Ovakav dizajn omogućava postizanje izuzetne propusnosti i energetske efikasnosti, jer se podaci kreću kroz mrežu aritmetičkih jedinica bez stalnog čitanja i

pisanja iz spoljašnje memorije. TPU poseduje i posebne memorijske blokove – Unified Buffer i Weight Memory – koji omogućavaju da se aktivacije i težine modela nalaze što bliže aritmetičkim jedinicama.

Prednosti specijalizovanog dizajna

Zahvaljujući ograničenom domenu primene i jednostavnom modelu izvršavanja, TPU ostvaruje više reda veličine bolji odnos performansi i potrošnje energije u poređenju sa opštim procesorima (CPU) i grafičkim procesorima (GPU) pri izvođenju operacija potrebnih za neuronske mreže.

Pored TPU-a, slične akceleratore razvijaju i druge kompanije (npr. Intel Nervana, NVIDIA Tensor Core, Apple Neural Engine), a svi slede sličnu filozofiju: maksimalna paralelizacija uz minimalne troškove prenosa podataka. Takvi procesori postaju osnovni element savremenih sistema koji kombinuju CPU, GPU i DNN akceleratore u zajedničkom okviru za heterogeno računanje.

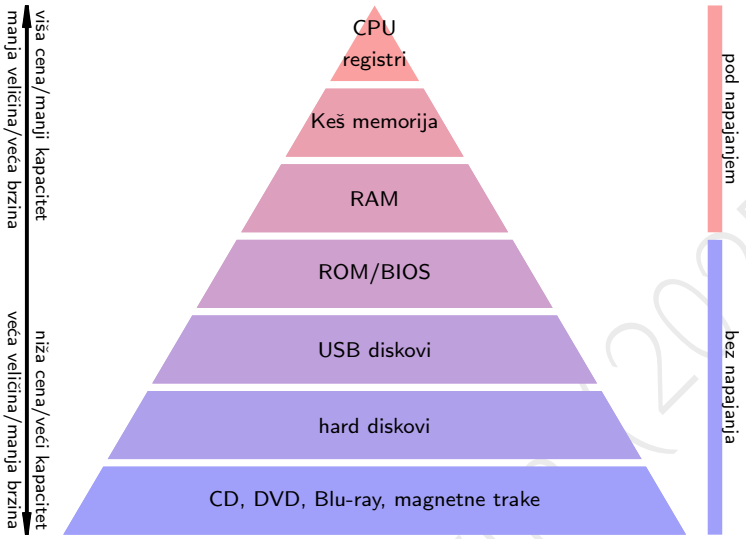
1.2 Memorijska hijerarhija

Druga centralna komponenta Fon Nojmanove arhitekture je *glavna memorija*, u kojoj se čuvaju programi i podaci potrebni procesoru za izvršavanje instrukcija. Sadržaj glavne memorije organizovan je kao linearni niz memorijskih lokacija, odnosno registara (najčešće bajtova), od kojih svaka ima svoju jedinstvenu adresu. Pošto se pristup može vršiti proizvoljnim redosledom, ova memorija se naziva *memorija sa slobodnim pristupom* (engl. *Random Access Memory, RAM*). Osnovni parametri memorija su *kapacitet*, *vreme pristupa* i *protok*.

U savremenim sistemima, razlika u brzini između procesora i glavne memorije postala je toliko izražena da direktan pristup više nije moguć bez ozbiljnog usporenja. Da bi se smanjio ovaj raskorak, uspostavlja se složena *memorijska hijerarhija* koja povezuje memorije različitih brzina, cena i kapaciteta. Na vrhu hijerarhije nalaze se *registri procesora* i *keš memorije* (engl. *cache*), dok su pri dnu sporije, ali kapacitetnije memorije — *glavna memorija*, *SSD uređaji* i druge *trajne memorije*. Procesor direktno komunicira samo s najbržim slojem, dok se podaci iz sporijih nivoa učitavaju prema potrebi.

Princip memorijske hijerarhije

Kako se udaljavamo od procesora, cena memorije po bajtu se smanjuje, kapacitet raste, ali vreme pristupa postaje duže. Hijerarhija memorija balansira ove suprotne zahteve tako da sistem, u proseku, deluje gotovo jednako brzo kao najbrža memorija u lancu.



Slika 1.2: Pregled memorijske hijerarhije

Zid memorije (engl. *memory wall*)

Brzina procesora poslednjih decenija rasla je mnogo brže od brzine glavne memorije. Ova sve veća razlika dovodi do situacije u kojoj procesor većinu vremena provodi čekajući podatke iz memorije – što se naziva „zid memorije“. Keš memorije, predmemoriranje i preuređivanje pristupa podacima nastoje da taj zid ublaže, ali ga ne mogu potpuno eliminisati.

Ova hijerarhijska organizacija čini osnovu svakog modernog računarskog sistema i omogućava efikasno balansiranje između brzine, cene i energetskih zahteva. U narednim odeljcima detaljnije se razmatraju ključni nivoi hijerarhije: registri, keš memorije i glavna memorija.

1.2.1 Registri i keš memorije (L1–L3)

Najviši nivo memorijske hijerarhije čine *registri* – mala, izuzetno brza memorija ugrađena direktno u procesorska jezgra. Predstavljaju najbržu memoriju jer se sve aritmetičke i logičke operacije izvode upravo nad podacima koji se nalaze u njima. Tipično ih ima od desetina do nekoliko stotina, sa latencijom od **jednog takta**.

Latencija je vreme od trenutka kada procesor zatraži podatak/instrukciju do trenutka kada je zaista dobije; meri se u taktovima (eng. clock cycles) ili nanosekundama. Pošto ALU ne može da nastavi bez operanada, niska latencija najbližih nivoa memorije je presudna za performanse.

Odmah ispod registara u memorijskoj hijerarhiji nalazi se *keš memorija* (engl. *cache*), organizovana u više nivoa: `L1`, `L2` i `L3`, koji čuvaju kopije najčešće korišćenih podataka i instrukcija. Keš 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. Cilj keš memorije je da sakrije razliku u brzini između procesora i glavne memorije (RAM-a), čija latencija može biti i stotine puta veća. Tipične vrednosti latencije su:

- `L1`: 2–4 ciklusa,
- `L2`: 10–20 ciklusa,
- `L3`: 30–50 ciklusa,
- Glavna memorija: 100–300 ciklusa.

Primer 1.4. Poređenje veličine i latencije

Nivo	*Tipična veličina*	*Latencija (ciklusi)*	*Primer arhitekture*
L1 (instrukcioni/podatkovni)	*32–64 KB*	*3–4*	*ARM Cortex-A53, Intel*
L2 (privatni)	*256 KB – 1 MB*	*10–15*	*ARM Cortex-A53, Intel*
L3 (deljeni)	*8–32 MB*	*30–50*	*Intel Core i7*

Registri i keš memorije u procesoru najčešće se izrađuju u **SRAM tehnologiji** (eng. Static Random Access Memory). Svaka ćelija SRAM memorije sastoji se od šest tranzistora, bez kondenzatora. Glavne prednosti su veoma brz pristup (reda pikosekundi do nanosekundi), nema potrebe za osvežavanjem (za razliku od DRAM-a), pouzdan i stabilan signal. A glavni nedostaci su mnogo veća površina po bitu (zato što ima 6 tranzistora umesto 1 kao kod DRAM memorije), samim tim je skuplja i energetski zahtevnija po bitu.

Keš memorija je organizovana u *blokove* (ili *linije*) tipične veličine 32–128 bajtova. Svaki blok sadrži podatke koji se često koriste zajedno, što omogućava iskorišćavanje **prostorne lokalnosti**. Kada se blok učita, on se smešta u odgovarajuću *keš liniju*. Adresiranje u keš memoriji zasniva se na podeli adrese na tri polja: `[tag | index | offset]`. Polje `index` određuje u koji deo keša može biti smešten traženi blok, dok `tag` služi za proveru da li se upravo taj blok nalazi u tom delu keša. Polje `offset` označava tačnu poziciju bajta ili reči unutar bloka.

Na osnovu toga razlikujemo:

- **direktno mapirani keš** – svaka adresa ima jedno moguće mesto,

- **n-asocijativni keš** – blok može biti u bilo kom od n mesta unutar jednog dela keša,

- **potpuno asocijativni keš** – blok može biti bilo gde u kešu.

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.

Kada dođe do promašaja (*miss*), keš mora odlučiti koji blok će biti zamenjen novim podacima iz nižeg nivoa memorije. Ovaj izbor sledi određenu **politiku zamene blokova**. Najčešće se koristi politika *najduže nekorišćenog bloka* (engl. *Least Recently Used* — LRU), koja pretpostavlja da će blok koji je najduže bio neaktivan verovatno i ubuduće biti najmanje potreban. Zbog hardverske složenosti potpune implementacije LRU pristupa, u praksi se često primenjuju približne varijante, poput *pseudo-LRU* algoritma, koji donosi gotovo isti efekat uz manji broj registara i logičkih operacija.

Upis podataka u keš takođe može slediti različite **politike upisa**. Naime, keš nije posebna memorija, već brza kopija dela glavne memorije (RAM), čiji je zadatak da ubrza pristup često korišćenim podacima; zato je neophodno održavati konzistentnost između keša i glavne memorije, kako bi svi delovi sistema videli iste vrednosti podataka. Procesor stalno čita i upisuje podatke. Kada upisuje (npr. menja neku promenljivu), ta promena prvo ide u keš, jer je to najbrža memorija i nalazi se odmah uz procesor. Ali, te izmene tada još nisu zapisane u glavnu memoriju (RAM), koja je mnogo sporija. Postavlja se pitanje: kada i kako ažurirati glavnu memoriju da bi svi nivoi imali isti sadržaj?

- **pisanje-prolazom** (engl. *write-through*) – svaka promena u kešu istovremeno se upisuje i u niži nivo memorije. Ovaj pristup pojednostavljuje održavanje konzistentnosti podataka, ali povećava broj pristupa memoriji.

- **pisanje-na-izbacivanje** (engl. *write-back*) – izmene se najpre upisuju samo u keš, a zatim se, prilikom izbacivanja bloka iz keša, sadržaj ažurira u nižem nivou. Takvi blokovi označavaju se „zastavicom prljavosti" (*dirty bit*). Ova politika smanjuje saobraćaj prema memoriji, ali zahteva složeniju kontrolu da bi se očuvala konzistentnost podataka jer ako dođe do kvara, može se desiti da neki podaci još nisu sinhronizovani.

Primer 1.5

ARM Cortex-A53. *Koristi dvo-nivojski keš: L1 od 32 KB za instrukcije i 32 KB za podatke, te objedinjeni L2 do 2 MB. L1 je virtuelno adresiran, i koristi politiku* pisanje-prolazom *sa alokacijom pri upisu. Tipična kazna promašaja iznosi 10–15 ciklusa ka L2 i oko 100 ciklusa do glavne memorije.*

Intel Core i7. *Sadrži tri nivoa keša: L1 (64 KB ukupno po jezgru), L2 (256 KB po jezgru) i L3 (deljeni, 8–16 MB). Svi nivoi su neblokirajući (*non-blocking*), tj. podržavaju više istovremenih upisa. Politika upisa je* pisanje-na-izbacivanje*. Kod procesora Intel Core i7-6700, latencija pristupa iznosi približno 4 ciklusa za L1, 12 ciklusa za L2, 42 ciklusa za L3 i oko 200 ciklusa za glavnu memoriju, što jasno ilustruje razliku brzina u memorijskoj hijerarhiji.*

Optimizacije keša

Performanse keš memorija unapređuju se nizom hardverskih i softverskih tehnika koje smanjuju vreme pristupa podacima i broj promašaja:

- višestruki nivoi keša(L1 privatni, L2 privatni, L3 deljeni),
- predučitavanje podataka (*prefetching*),
- višestruke banke i paralelni pristup,
- i politika koherentnosti (*cache coherence*).

Višestruki nivoi keša (L1, L2, L3) omogućavaju da se kombinovanjem manjih i bržih keševa sa većima i sporijima postigne ravnoteža između brzine i kapaciteta. Tipično su L1 keševi privatni za svako jezgro, dok je L3 deljeni resurs.

Predučitavanje podataka koristi predviđanje budućih pristupa memoriji na osnovu obrasca izvršavanja programa, pa se podaci dovlače u keš pre nego što ih procesor zaista zatraži, čime se smanjuje broj promašaja.

Višestruke banke i paralelni pristup omogućavaju da keš istovremeno opsluži više zahteva, čime se povećava efektivna propusnost i iskorišćenost procesora.

Politike koherentnosti keša obezbeđuju da svi procesori u višejezgarnim sistemima u svakom trenutku vide iste vrednosti podataka, čime se održava konzistentnost između kopija u različitim keševima.

1.2.2 Glavna memorija, SSD i NVM

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 ra-

čunara (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.

Brza memorija je skupa i energetski „skuplja“ po bitu, dok je spora memorija jeftinija, ali sa većom latencijom. *Hijerarhija memorija* kombinuje više tehnologija tako da sistem deluje *kao veliki i brz*, uz cenu i potrošnju blisku donjim nivoima. Ključ je *lokalnost* (vremenska i prostorna): gornji nivoi (keševi, registri) omogućavaju obezbeđuju brzinu, a donji nivoi (DRAM, SSD) obezbeđuju kapacitet.

DRAM. *Dinamička memorija sa slobodnim pristupom* (engl. *Dynamic Random Access Memory*, **DRAM**) predstavlja osnovnu radnu memoriju savremenih računara. Naziv „dinamička“ potiče od činjenice da se svaka informacija u DRAM čipu čuva kao električni napon u kondenzatoru koji vremenom „curi“ i mora periodično da se *osvežava* (engl. *refresh*). Ova osobina čini DRAM *volatilnom* – njen sadržaj nestaje čim se izgubi napajanje. Ipak, zahvaljujući jednostavnoj strukturi ćelije (samo jedan tranzistor i jedan kondenzator), DRAM postiže izuzetno visoku gustinu podataka i povoljnu cenu po bitu, zbog čega dominira kao glavna memorija sistema.

Osnovne osobine DRAM tehnologije

- **Latencija i propusni opseg.** Pristup jednoj lokaciji traje desetine nanosekundi; nakon inicijalnog čitanja, prenosi se čitav niz susednih reči u tzv. *rafalu* (engl. *burst*), čime se znatno povećava propusni opseg.
- **Osvežavanje i volatilnost.** Svaka ćelija mora se osvežiti nekoliko stotina puta u sekundi. Tokom tog procesa memorijski kontroler privremeno blokira pristupe, što uvodi dodatno kašnjenje i potrošnju energije.
- **Energetski profil.** DRAM troši energiju i dok je neaktivna: deo potrošnje odlazi na *osvežavanje* (tzv. *statika*), a deo na same operacije čitanja i pisanja (*dinamička potrošnja*). Moderni standardi kao što su DDR4 i DDR5 snižavaju radni napon, uvode režime “spavanja” kako bi se smanjila ukupna potrošnja energije.

Savremeni DRAM moduli organizovani su u *blokove* i *redove*, što omogućava da se više pristupa izvršava preklopljeno i time poveća *propusni opseg* memorije. Kod tipičnih desktop procesora, vreme pristupa DRAM-u iznosi oko 60–100 ns, što je nekoliko desetina puta sporije od pristupa kešu drugog nivoa, a više stotina puta sporije od registara procesora. Upravo ta razlika u brzini između procesora i glavne memorije jedan je od glavnih razloga postojanja složene memorijske hijerarhije.

Uloga DRAM-a u hijerarhiji

Glavna memorija je središnji radni prostor računara – ona čuva sve programe i podatke koji su trenutno aktivni. Procesor ne pristupa direktno sporijim uređajima kao što su SSD ili disk, već se svi podaci pre izvršavanja moraju preneti u DRAM. Što je manji broj promašaja u kešu (L3→DRAM), to je niže prosečno vreme pristupa (eng. Average Memory Access Time, AMAT) i manja potrošnja energije.

SSD (Flash memorija) predstavlja **nevolatilno skladište podataka** zasnovano na tehnologiji NAND tranzistora. Za razliku od DRAM-a, SSD čuva sadržaj i bez napajanja, ali mu je **latencija pristupa višestruko veća** – meri se u mikrosekundama. Upis u SSD ne vrši se nad pojedinačnim bajtovima, već nad **stranicama i blokovima**, koji se pre svakog upisa moraju obrisati. Upravo zato je neophodan **kontroler** koji obavlja složene funkcije: *mapiranje* fizičkih adresa, *keširanje* podataka i ravnomerno raspoređivanje ciklusa pisanja kako bi se produžio vek trajanja memorije.

U pogledu potrošnje, SSD je izuzetno štedljiv u mirovanju, dok su operacije pisanja energetski skuplje. Zahvaljujući odsustvu mehaničkih delova, SSD je daleko pouzdaniji i energetski efikasniji od klasičnih tvrdih diskova. Njegova je osnovna uloga **trajno čuvanje podataka**, a ne zamena za DRAM, jer i dalje poseduje veću nasumičnu latenciju i ograničen broj ciklusa upisa.

NVM (nevolatilne memorije srednjeg sloja). Predstavljaju tehnologije koje popunjavaju jaz između brze, ali volatilne DRAM memorije i trajne, ali sporije Flash memorije (SSD). One kombinuju osobine obe grupe – brz pristup u nanosekundama do mikrosekundi i mogućnost trajnog čuvanja podataka bez napajanja. NVM koristi fizičke ili magnetne promene u materijalu koje ostaju i bez napajanja.

Najpoznatiji predstavnici su *Intel Optane* (zasnovan na 3D XPoint tehnologiji), *PCM* (phase-change memory) i *MRAM* (magnetoresistive RAM). Za razliku od NAND Flash-a, mnoge NVM tehnologije podržavaju **adresiranje po bajtu**, što omogućava direktan pristup bez posredovanja datotečnog sistema. Latencije su tipično nekoliko mikrosekundi – znatno niže od SSD-a, ali i dalje veće od DRAM-a.

Ove memorije mogu raditi u različitim režimima: kao **brzi disk** (blokovski uređaj), kao **proširenje radne memorije**, ili kao **trajna radna memorija** u kojoj podaci preživljavaju i nakon gašenja sistema.

Energetski posmatrano, NVM ne zahteva osvežavanje, pa troši manje energije u stanju mirovanja. Pristupi su skuplji od DRAM-a, ali znatno efikasniji od NAND Flash-a po obavljenoj korisnoj operaciji. Uvođenjem ovog sloja u hijerarhiju memorije, **smanjuje se kazna promašaja ka sporijem SSD-u**, a time i ukupna potrošnja energije po izvršenom zadatku.

Od registara do SSD-a: odnos brzine, kapaciteta i cene

Nivo	Veličina	Latencija	Trajnost	Cena/bit	Na
Registri	~KB	0.2–0.5 ns	ne	visoka	u je
L1/L2/L3 (SRAM)	desetine MB	1–4 / 4–12 / 30–50 ns	ne	srednja	AM
DRAM (glavna)	10–100+ GB	50–100 ns	ne	niža	osv
NVM (Optane/PCM)	GB–desetine GB	0.3–2 μs	da	niža	mo
SSD (NAND)	0.5–10+ TB	50–150 μs	da	niža	sek

Virtuelna memorija. Predstavlja sloj apstrakcije između fizičke memorije računara i programa koji se izvršavaju. Ona omogućava da svaki proces "ima utisak" da raspolaže sopstvenim, neprekidnim prostorom memorije, iako se on u stvarnosti deli između više programa i može biti raspoređen na različite fizičke lokacije. Virtuelnom memorijom upravlja **operativni sistem**, koji po potrebi premešta delove podataka između glavne memorije i trajnog skladišta. Na taj način se postiže veća fleksibilnost, zaštita i efikasnije korišćenje fizičkih resursa. Detaljniji prikaz mehanizama virtuelne memorije biće dat u poglavlju *Sistemski softver*.

1.3 Sistemska magistrala i komunikacija komponenti

Performanse savremenih računarskih sistema ne zavise samo od brzine procesora, već i od načina na koji su **komponente međusobno povezane**. Procesor, memorija i ulazno–izlazni uređaji moraju stalno da razmenjuju podatke, pa je od efikasnosti te komunikacije direktno zavisila i brzina celog sistema.

Magistrala (engl. *bus*) je zajednički komunikacioni kanal preko koga više uređaja može da prenosi *podatke*, *adrese* i *upravljačke signale*. Kada je računar imao samo nekoliko komponenti, takvo deljenje jedne zajedničke linije bilo je jednostavno i ekonomično rešenje. Da bi se izbegli sudari, magistrala koristi mehanizam **arbitraže** kojim se odlučuje koji uređaj u datom trenutku ima pravo prenosa.

Uloga magistrale je, dakle, da obezbedi *zajednički jezik komunikacije* između svih delova računara: procesor putem nje čita podatke iz memorije, piše rezultate nazad ili komunicira sa spoljnim uređajima. Takav princip rada činio je osnovu svih klasičnih računara, zasnovanih na arhitekturi fon Nojmana.

Kako su sistemi postajali složeniji, a broj uređaja rastao, zajednička magistrala postala je **usko grlo**. Više uređaja je moralo da čeka svoj red na prenos, pa su performanse opadale. Savremene arhitekture zato uvode *preklopljene organizacije*, gde svaka komponenta ima sopstvenu vezu prema centralnom delu sistema.

Takva **mreža međupovezivanja** (engl. *interconnection network*) obezbeđuje više paralelnih kanala, veću propusnost i manju latenciju.

Osnovni pojmovi

- **Propusnost** (engl. *bandwidth*) – količina podataka koja se može preneti u jedinici vremena. Zavisna je od *širine magistrale* (broja bitova koji se prenose istovremeno) i od *radnog takta* (učestalosti prenosa). Aproksimativno, za sinhrone paralelne linije:

$$B = \frac{w \cdot f}{8}$$

  gde je w širina (u bitovima), a f radni takt. Na primer: 64-bitna magistrala na 100 MHz daje $\frac{64 \cdot 100 \cdot 10^6}{8} \approx 0{,}8$ GB/s (idealno). Stvarna propusnost je obično manja, jer deo vremena odlazi na upravljačke signale i arbitražu.

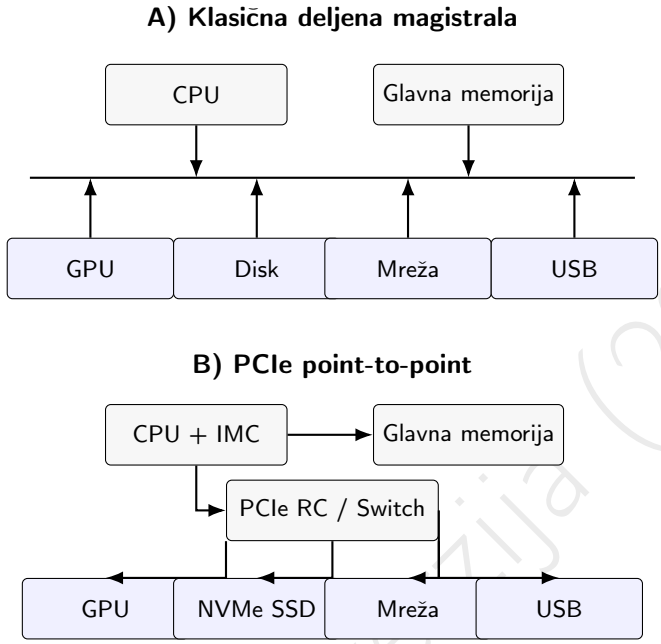
- **Latencija** (engl. *latency*) – vreme potrebno da podatak stigne od izvora do odredišta. Na latenciju utiču kašnjenja u električnim vodovima, vreme obrade u uređajima i eventualno čekanje na pristup magistrali. Dok propusnost pokazuje „koliko", latencija govori „koliko brzo" — i oba parametra moraju biti usklađena za postizanje optimalnih performansi.

- **Arbitraža** – mehanizam kojim se odlučuje koji uređaj u datom trenutku ima pravo da koristi magistralu. U jednostavnim sistemima to može biti *centralni arbitar* (npr. procesor), dok složeniji sistemi koriste *distribuisanu arbitražu* gde svaki uređaj učestvuje u dogovoru o redu pristupa. Cilj arbitraže je da obezbedi pravedan i efikasan prenos podataka bez konflikata i gubitaka.

- **Topologija** – način na koji su komponente fizički povezane. Kod klasične magistrale sve komponente dele jedan zajednički vod, dok savremene arhitekture koriste višestruke, paralelne veze i preklopljene mreže. O različitim topologijama (prsten, zvezda, mreža i dr.) detaljnije će biti reči u posebnom poglavlju o *računarskim mrežama*.

1.3.1 Sistemske magistrala i PCI Express (PCIe)

U starijim računarima komunikacija između procesora, memorije i uređaja obavljala se putem klasične **sistemske magistrale** (npr. standardi *PCI* ili *AGP*). Svi uređaji delili su isti komunikacioni kanal, pa je bilo neophodno da sistem određuje koji uređaj u datom trenutku ima pravo pristupa. Takav pristup je jednostavan, ali postaje ozbiljno ograničenje kada broj uređaja i količina razmenjenih podataka poraste.



Slika 1.3: (A) Klasična deljena magistrala sa arbitražom i jednim zajedničkim kanalom; (B) PCIe point-to-point topologija sa root-kompleksom/switch-em, paralelnim vezama i DMA prenosima.

Savremeni sistemi zato uvode **PCI Express (PCIe)**, naslednika klasične magistrale, koji koristi **direktne digitalne veze između parova uređaja** (eng. point-to-point). Svaka komponenta ima sopstveni kanal ka centralnom delu sistema (procesoru ili čipsetu), pa se prenosi podataka mogu odvijati istovremeno i nezavisno. Time se izbegavaju zagušenja i povećava ukupna propusnost sistema.

Dodatno, savremeni uređaji povezani preko PCIe standarda mogu samostalno da razmenjuju podatke sa glavnom memorijom korišćenjem **direktnog pristupa memoriji (DMA)**. Na taj način procesor ne mora da posreduje u svakom prenosu, čime se oslobađa za druge zadatke i smanjuje ukupno vreme obrade. Ova arhitektura omogućava izuzetno brzu komunikaciju sa uređajima koji zahtevaju visok protok podataka, poput grafičkih procesora, NVMe diskova ili mrežnih adaptera.

1.3.2 Čipset i kontroleri

Savremene matične ploče više nisu zasnovane na jednoj centralnoj magistrali, već na nizu *specijalizovanih kontrolera* koji povezuju procesor, memoriju i uređaje. Ulogu koordinacije ima **čipset**, odnosno skup logičkih čipova koji upravljaju komunikacijom između procesora, glavne memorije, grafičkog podsistema, skladišta podataka i perifernih uređaja.

U ranijim generacijama arhitekture razlikovala su se dva osnovna dela čipseta: *severni most* (engl. northbridge), koji je povezivao procesor i memoriju, i *južni most* (engl. southbridge), koji je služio za sporije I/O uređaje (USB, SATA, PCI). Danas su te funkcije objedinjene: memorijski kontroler i PCIe konekcije često su integrisani direktno u procesor, dok čipset preuzima samo pomoćne funkcije – povezivanje dodatnih uređaja, mrežnih interfejsa i upravljanje napajanjem.

1.4 Ulazno–izlazni podsistem

Ulazno–izlazni (*I/O*) podsistem predstavlja ključnu komponentu savremenih računarskih sistema, jer omogućava komunikaciju između procesora, memorije i spoljašnjih uređaja. Razvoj interneta i servisa koji se oslanjaju na masovno skladištenje i mrežnu povezanost pomerio je fokus arhitekture računara sa *računanja* ka *komunikaciji i skladištenju informacija*.

I/O podsistem

I/O podsistem obuhvata uređaje (diskove, tastature, mrežne adaptere, grafičke kartice), interfejse i kontrolere koji upravljaju prenosom podataka između uređaja i glavne memorije.

I/O uređaji su fizički elementi sistema koji obavljaju ulazne ili izlazne operacije. Zbog raznolikosti brzina, formata podataka i protokola komunikacije, svaki uređaj poseduje odgovarajući **kontroler** – specijalizovani procesor koji posreduje između uređaja i magistrale sistema.

Tipično, procesor ne komunicira direktno sa uređajem, već šalje komande kontroleru. Kontroler prevodi te komande u niz električnih signala prilagođenih uređaju, nadgleda izvršenje i obaveštava procesor putem prekida kada je operacija završena (više o prekidima u sekciji *Sistemski softver*).

Primer: Disk kontroler prevodi zahteve operativnog sistema (npr. čitanje sektora 12345) u konkretne mehaničke operacije – pozicioniranje glave i čitanje odgovarajuće staze. Moderni kontroleri uključuju i keš memoriju i algoritme za optimizaciju redosleda komandi.

I/O sistem se često modeluje kao hijerarhija slojeva: na vrhu su aplikacije i operativni sistem, ispod njih drajveri i kontroleri, a na dnu fizički uređaji. Ovakva slojevita organizacija omogućava apstrakciju hardvera i modularnost u projektovanju.

1.4.0.1 Direktan pristup memoriji (DMA)

Da bi se smanjilo opterećenje procesora tokom prenosa podataka, koristi se mehanizam **direktnog pristupa memoriji** (eng. Direct Memory Access – DMA). Kod standardnog I/O prenosa procesor bi morao da pročita svaki bajt sa uređaja i upiše ga u memoriju, što je neefikasno. DMA kontroler preuzima ovu ulogu: on samostalno kopira blokove podataka između uređaja i memorije, dok procesor nastavlja sa izvršavanjem drugih instrukcija.

Kada DMA prenos završi, kontroler generiše prekid da bi obavestio procesor da su podaci spremni. Ovakav pristup značajno povećava propusnost sistema i omogućava preklapanje I/O i računanja.

> DMA omogućava istovremeno izvršavanje procesorskih operacija i prenosa podataka, čime se postiže veća iskorišćenost magistrale i manja latencija I/O operacija.

1.4.0.2 Standardi i interfejsi: USB, PCIe, NVMe

Zahvaljujući standardizovanim I/O interfejsima, uređaji različitih proizvođača mogu da rade zajedno i da ostvaruju visok protok podataka.

- **USB (Universal Serial Bus)** je serijski standard koji povezuje periferne uređaje male i srednje brzine (tastature, miševe, kamere, memorijske stikove). Moderni standardi kao USB 3.2 i USB4 dostižu brzine prenosa i do 40 Gb/s i podržavaju napajanje uređaja.

- **PCI Express (PCIe)** je glavni interfejs za visoko-propusne uređaje (grafičke kartice, SSD diskove, mrežne adaptere). PCIe koristi višekanalnu arhitekturu koja omogućava paralelne prenose podataka sa malom latencijom.

- **NVMe (Non-Volatile Memory Express)** je protokol razvijen specijalno za SSD uređaje zasnovane na flash memoriji. Za razliku od SATA ili SCSI protokola, NVMe koristi PCIe magistralu i omogućava hiljade paralelnih redova komandi, čime dramatično smanjuje latenciju pristupa podacima.

Kombinacijom PCIe i NVMe standarda postignute su performanse koje omogućavaju višestruko brži pristup u odnosu na tradicionalne magnetne diskove – latencije su reda mikrosekundi i propusnost je veća od 5 GB/s po uređaju.

Pouzdanost informacija se ostvaruje čuvanjem podataka u više kopija ili uz dodatne kontrolne podatke koji omogućavaju oporavak ako dođe do greške, kao kod RAID organizacija diskova, i upotrebom nevolatilnih medijuma (SSD, NVMe). Operativni sistem upravlja redovima komandi i prioritizacijom pristupa, dok DMA i kontroleri obezbeđuju efikasan fizički prenos.

Savremeni I/O podsistem nije samo skup perifernih uređaja, već složena arhitektura koja povezuje procesor, memoriju, magistralu i spoljašnji svet kroz standardizovane, paralelne i pouzdane komunikacione protokole.

1.5 Integracija hardverskih podsistema

Računarski sistem funkcioniše kao celina zahvaljujući složenoj integraciji komponenti na **matičnoj ploči** (engl. motherboard). Ona predstavlja fizičku osnovu sistema i obezbeđuje električno napajanje, komunikacione puteve i sinhronizaciju između procesora, memorije, grafičkog podsistema i ulazno–izlaznih uređaja.

Centralnu ulogu u koordinaciji komponenti ima **čipset**, koji se istorijski sastojao od *severnog* i *južnog mosta*. Savremeni procesori preuzeli su deo funkcionalnosti čipseta: kontroler memorije i PCIe interfejs sada su integrisani direktno u CPU, dok čipset na matičnoj ploči (često nazvan *Platform Controller Hub, PCH*) upravlja perifernim uređajima, mrežnim i USB kontrolerima.

Moderni računari ne pokreću se odmah operativnim sistemom, već prolaze kroz proces inicijalizacije kojim upravlja ugrađeni programski kod, **firmware** ugrađen u matičnu ploču. U starijim sistemima to je bio **BIOS** (*Basic Input/Output System*), dok je u novim sistemima zamenjen naprednijim **UEFI firmware** (*Unified Extensible Firmware Interface*).

UEFI

UEFI predstavlja savremeni sloj između hardvera i operativnog sistema: on prepoznaje komponente, vrši testiranje, učitava učitavača operativnog sistema (eng. boot loader) i omogućava sigurno pokretanje sistema (eng. *Secure Boot*).

BIOS je monolitan i zatvoren sistem – skup rutinskih procedura u ROM-u, čiji je zadatak bio da izvrši POST (*Power-On Self Test*) i pokrene operativni sistem. Korisnik i programeri nisu mogli da mu dodaju funkcionalnosti: nije imao drajvere koje bi se lako ažurirali, niti podršku za mrežu, grafički interfejs ili velike diskove (više od 2 TB).

Kod klasičnih računara iz 80-ih i 90-ih, BIOS je bio upisan u ROM čip direktno na matičnoj ploči. To znači:

- sadržaj BIOS-a je fizički „urezan“ u čip,
- ažuriranje nije bilo moguće ili je zahtevalo fizičku zamenu čipa,
- kod je bio vrlo mali (obično 64 – 512 KB) i pisan u čistom asembleru.

UEFI je modularan i programabilan – može da koristi mrežne drajvere, grafički interfejs, pa čak i jednostavne aplikacije pre nego što se operativni sistem pokrene. Na taj način on obezbeđuje veću fleksibilnost i sigurnost u odnosu na tradicionalni BIOS.

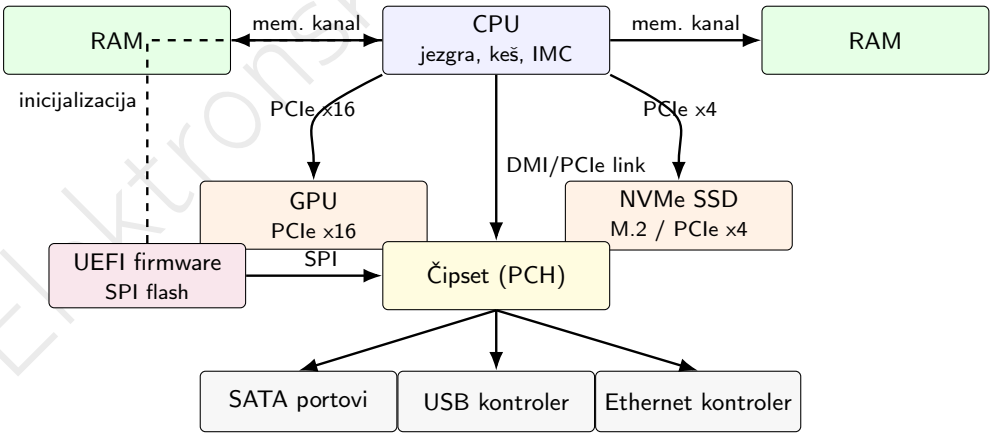
UEFI se i dalje fizički nalazi na matičnoj ploči, ali ne više u ROM-u, već u malom fleš čipu (SPI flash ROM) koji se može prepisivati. To znači:

- UEFI je i dalje „ugrađen“ – deo ploče – ali se može ažurirati softverski,
- čip je tipično veličine 8 – 16 MB,
- proizvođač (ili korisnik) može izvršiti *firmware update* (tzv. „flešovanje“ BIOS-a/UEFI-ja).

Kada se računar uključi:

- Procesor počinje da izvršava kod iz te male flash memorije (adresira se direktno).
- UEFI pokreće osnovne servise, učitava drajvere za PCIe, SATA, USB itd.
- Zatim pronalazi i pokreće učitavač operativnog sistema (eng. boot loader) sa diska (ili mreže).

Integracija svih ovih slojeva – fizičkih, logičkih i upravljačkih – predstavlja osnovu savremenog računarskog sistema. Operativni sistem preuzima kontrolu tek nakon što UEFI završi inicijalizaciju i preda mu izvršavanje, čime počinje rad softverskog sloja sistema.



Slika 1.4: Minimalna šema integracije: CPU sa integrisanim memorijskim kontrolerom (IMC), RAM, PCIe veze ka GPU/NVMe, veza ka PCH (DMI/PCIe), periferni kontroleri i UEFI firmware u SPI flash memoriji.

1.6 Trendovi u savremenim arhitekturama

Energetska potrošnja danas predstavlja jedan od ključnih izazova u dizajnu savremenih računarskih sistema. Snaga se mora dovesti i ravnomerno raspodeliti unutar čipa, a zatim efikasno odvesti u vidu toplote. Razlikuju se tri fundamentalna parametra: *vršna snaga* (engl. *peak power*), koja označava maksimalno opterećenje napajanja; *termička projektna snaga* (*Thermal Design Power, TDP*), koja definiše zahteve sistema hlađenja; i *srednja snaga*, koja zavisi od tipičnog opterećenja. Ipak, za realnu procenu efikasnosti presudniji je pojam *energije po zadatku* – proizvod prosečne snage i vremena izvršavanja. U praksi, procesor koji troši više snage može biti energetski povoljniji ako posao obavi znatno brže. Pošto su vrednosti napona poslednjih godina praktično „zakočene" na oko 1 V, rast radnog takta se usporio, a fokus je premešten na energetsku efikasnost i paralelizam.

Tehnike upravljanja energijom

Najvažnije savremene tehnike uključuju:

- *isključivanje takta* neaktivnim jedinicama (eng. clock gating),
- *dinamičko prilagođavanje napona i frekvencije* (eng. Dynamic Voltage and Frequency Scaling, DVFS),
- *isključivanje napajanja* neaktivnim delovima čipa (*power gating*), kao i strategiju *trka-do-zaustavljanja* (eng. race-to-halt), kojom se zadatak izvršava maksimalnom brzinom, a zatim sistem prelazi u stanje mirovanja radi uštede energije.

Značajan udeo ukupne potrošnje dolazi i iz *statičke energije* izazvane curenjem struje kroz tranzistore, naročito u velikim keš memorijama tipa SRAM. Zbog toga savremeni čipovi uvode višestruke energetske domene, koji omogućavaju potpuno isključivanje delova procesora kada nisu u upotrebi.

SoC (*System-on-Chip*) arhitekture integrišu procesorska jezgra, grafiku, memorijski kontroler, mrežni interfejs i druge komponente u jedinstveni čip. Ovaj pristup smanjuje kašnjenja i potrošnju energije, jer se komunikacija odvija unutar čipa, bez potrebe za spoljnim magistralama.

Pored logičke potrošnje, presudnu ulogu ima i *memorijska hijerarhija*: pristup glavnoj memoriji (DRAM) troši i do 10^4 puta više energije nego operacija sabiranja u aritmetičko–logičkoj jedinici. Ova razlika objašnjava zašto je lokalna povezanost podataka i upotreba bafera ključna za energetski efikasne arhitekture.

Paradigma energetske efikasnosti

Sa stanovišta projektovanja, glavni cilj postaje *minimizacija energije po zadatku* (eng. energy per task) i *maksimizacija performansi po vatu* (engl. performance per watt). Kako više nije moguće sve tranzistore istovremeno držati aktivnim bez prekoračenja termičkog limita, značajan deo čipa ostaje neaktivan – fenomen poznat kao *tamni silicijum* (engl. dark silicon). Ovo je dovelo do porasta upotrebe *domen-specifičnih akceleratora* kao što su TPU, NPU i Tensor Cores, koji omogućavaju visok stepen paralelizma uz minimalnu potrošnju energije.

U celini, dalji razvoj procesora sve manje zavisi od povećanja takta, a sve više od *heterogenosti, specijalizacije* i *štedljive upotrebe podataka*. Energetska efikasnost je postala novo merilo napretka, a ne samo brzina izvršavanja.

Pitanja i zadaci za vežbu

Pitanje 1.1. *Koje tri osnovne komponente čine tipičan računar prema Fon Nojmanovom modelu?*

Pitanje 1.2. *Koje su osnovne funkcije aritmetičko–logičke jedinice (ALU)?*

Pitanje 1.3. *Koje su faze ciklusa instrukcije koje sprovodi kontrolna jedinica?*

Pitanje 1.4. *Šta je osnovna ideja cevovoda (pipelining) i šta mu je glavni cilj?*

Pitanje 1.5. *Koje tri vrste hazard-a mogu da se pojave u cevovodu?*

Pitanje 1.6. *Šta označava pojam* Instruction-Level Parallelism (ILP)*?*

Pitanje 1.7. *Kako* predikcija skokova *doprinosi smanjenju broja ciklusa po instrukciji (CPI)?*

Pitanje 1.8. *U čemu se razlikuju ILP i TLP pristupi paralelizmu?*

Pitanje 1.9. *Koja je osnovna razlika između CPU i GPU arhitekture u pogledu paralelizma?*

Pitanje 1.10. *Koja je uloga keš memorije u hijerarhiji memorija?*

Pitanje 1.11. *Koja je razlika između politika upisa pisanje-prolazom (engl.* write-through*) i pisanje-na-izbacivanje (engl.* write-back*)?*

Pitanje 1.12. *Šta je osnovna funkcija sistemske magistrale?*

Pitanje 1.13. *Koje su prednosti PCI Express (PCIe) u odnosu na klasičnu zajedničku magistralu?*

Pitanje 1.14. *Šta je zadatak čipseta (PCH) u savremenim računarima?*

Pitanje 1.15. *Koja je osnovna razlika između BIOS-a i UEFI firmware-a?*

Pitanje 1.16. *Gde se fizički nalazi UEFI kod i kako se ažurira?*

Pitanje 1.17. *Šta znači pojam* Direct Memory Access (DMA) *i čemu služi?*

Pitanje 1.18. *Koja je funkcija NVMe protokola u odnosu na starije SATA interfejse?*

Pitanje 1.19. *Šta je osnovna ideja* System-on-Chip (SoC) *arhitekture?*

Pitanje 1.20. *Koja su tri osnovna parametra energetske potrošnje čipa (vršna snaga, TDP, srednja snaga)?*

Pitanje 1.21. *Navedi dve tehnike upravljanja energijom korišćene u savremenim procesorima.*

Pitanje 1.22. *Šta je* tamni silicijum *(dark silicon) i kako utiče na dizajn procesora?*