1. RAČUNALO OPĆE NAMJENE

1.1. Računalo opće namjene

Digitalno računalo obično se naziva računalo opće namjene. Teorija računarskih znanosti govori da ukoliko digitalno računalo ima *dovoljno* memorije ekvivalentno je ***Turingovom stroju*** (nazvanom po Alanu Turingu, Britanskom računarskom znanstveniku iz 1940 g.). Grubo govoreći Turingov stroj može ukoliko ima dovoljno vremena proračunati sve funkcije koje su izračunjive. Ključne riječi su: *dovoljno memorije* i *dovoljno vremena*. U posljednjih 50-tak godina računala su se razvijala od memorijskih kapaciteta od svega 1024 do 10^{12} bita te brzina izvođenja jedne naredbe do 100 000 i više naredbi u milisekundi. Čak i s ovim performansama veliki broj zadataka ostaje ispod granica mogućnosti današnjih računala. To je npr. 3-D animacija u realnom vremenu, prognoza vremena, simulacija rada složenih fizikalnih sustava, "renderiranje" i sl. Možda će u budućnosti računala biti u mogućnosti rješavati ovakve i složenije probleme u prihvatljivom vremenskom intervalu, ali današnja računala su nažalost daleko od toga.

Odabir odgovarajućeg računala za određenu aplikaciju zahtjeva uzeti u obzir mnoge čimbenike. Današnja računala nalaze se u granicama od malih kontrolera za jednostavne primjene, preko osobnih računala za kućnu upotrebu i uredsko poslovanje pa sve do velikih i snažnih računala i superračunala namijenjena jako zahtjevnim zadacima.

1.1.1 Definicije i konvencije

Tijekom izučavanje problematike iz računarskih znanosti susreću se neke vrlo velike i neke vrlo male veličine. U normalnoj komercijalnoj i inženjerskoj praksi izrazi *kilo* (K) je 10^3, *mega* (M) 10^6, *giga* (G) 10^9 i *tera* (T) 10^{12}. Na sličan način u računarstvu, zbog prirode digitalnih računala, koristi se baza broja 2 pa vrijedi *kilo* (K) je 2^{10}, *mega* (M) 2^{20}, *giga* (G) 2^{30} i *tera* (T) 2^{40}. S brojevnom bazom 2 obično se opisuju memorijski kapaciteti, a s bazom broja 10 frekvencija takta (engl. *clock frequency*). Za male vrijednosti koriste se izrazi *mili* (m) 10^{-3}, *mikro* (μ) 10^{-6}, *nano* (n) 10^{-9}, *piko* (p) 10^{-12}.

Izrazi bit (b), četiri bita (engl. *nibble*), oktet (engl. *byte*) (B) 8 bita, riječ (engl. *word*) (w) 2 okteta, dvostruka riječ (engl. *double word*) (dw) 4 okteta, duga riječ (engl. *long word*) (lw) 8 okteta su izrazi uobičajeni u opisivanju dužine podatka ili naredbe. Također uobičajeno je da se bitovi u riječima označavaju od lijeva prema desno pa je tako prvi lijevi bit bit najvećeg značaja (engl. *most significant bit* skr. msb), a skroz desni bit bit najmanjeg značaja (engl. *last significant bit* skr. lsb)

1.1.2. Različiti pogledi na računalo

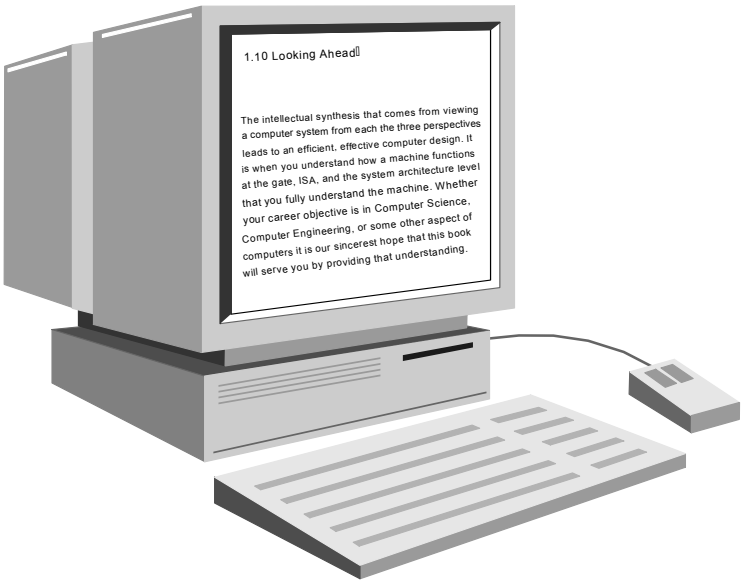
Kako je već napomenuto računalo se može promatrati iz više perspektiva. Različito promatraju, doživljavaju, računalo njegovi krajnji korisnici, različito programeri, a posebno projektanti računala na različitim razinama. Navedeno je razlog da se analiziraju sljedeći pristupi računalu:

- pogled sa stajališta krajnjeg korisnika,
- pogled sa stajališta programera u strojnom ili simboličkom jeziku,

- pogled sa stajališta projektanta arhitekture računala,
- pogled sa stajališta projektanta logičkih sklopova.

1.2 Korisnički pogled na računalo

Pod pojmom korisnik (engl. *user*) podrazumijeva se osoba koja koristi računalo za obavljanje nekog posla. Ta osoba može koristiti računalo za pisanje u Wordu, izradu izvještaja u Excelu, crtanje projekata u AutoCadu, obradu slika u Corel Drawu, pretraživati Intenet pomoću Netscapea ili programirati u nekom višem jeziku (engl. *High Level Language*) FORTRAN, C, Visual Basic, Java i sl. Za ovakve korisnike obično je sklopovlje računala potpuno odvojeno od korisnika operacijskim sustavom i aplikacijskim programom. Korisnik samo primjećuje brzinu kojom se izvodi njegova aplikacija, kapacitet memorije i diska te karakteristike ulazno/izlaznih uređaja koje koristi (tipkovnica, monitor, miš, pisač).



Slika 1.1. *Korisnički pogled na računalo.*

1.3. Pogled na računalo programera u simboličkom jeziku

Programer koji programira u strojnom odnosno simboličkom jeziku izrađuje temeljnu programsku podršku potrebnu kako bi procesor mogao obavljati svoje zadatke. Kako bi se dobio uvid u ovu problematiku potrebno je objasniti sljedeće pojmove:

- Strojni jezik (engl. *Machine language*) je skup temeljnih naredbi koje procesor može izvoditi, a izražen je kao niz 0 i 1.
- Simbolički jezik (engl. *Assembly language*) je alfanumerički ekvivalent strojnog jezika. Naime, programeru je puno lakše koristiti skraćenicu naredbe koje ga podsjeća na operaciju koju procesor mora izvesti nego binarni zapis iste naredbe.
- Program za prevođenje simboličkog jezika (engl. *Assembler*) je program koji translatira (preslikava jedan na jedan) simbolički jezik u strojni jezik procesora. Praktički ovo znači da programiranje u simboličkom jeziku je programiranje u

prirodnom jeziku procesora. Pod pojmom programer podrazumijevati će se osoba koja programira u simboličkom jeziku.

MC 68000 Simbolički jezik	**Strojni jezik**
MOVE.W D4,D5	0011 101 000 000 100
ADDI.W #9,D2	0000 000 010 111 100 0000 0000 0000 1001

Tablica prikazuje dvije naredbe procesora MC 68000 u simboličkom i strojnom jeziku. Strojna naredba razbijena je na polja koja određuju značenje pojedinih dijelova naredbe. Analizirajmo prirodu ovih naredbi. Prva naredba MOVE.W D4, D5 je naredba kojom se prebacuje (kopira) 16 bitovna riječ (kod 0011) u spremnik (000) D5 (0101) iz spremnika (000) D4 (100). Pod pojmom spremnik (engl. *register*) podrazumijeva se sklop sa svojstvom da uskladišti (memorira) niz bita. Druga naredba, ADDI.W #9, D2, zbraja (0000) sa sadržajem spremnika (000) D2 (010) 16 bitovnu riječ koja je sastavni dio naredbe(111 100) i pohranjuje rezultat u spremnik D2.

Prvi dio naredbe naziva se operacijski kod (engl. *operation code* skr. opcode), a drugi dio naredbe se odnosi na polje operanada (engl. *operand fields*). Opreracijski kod za prebacivanje podataka iz spremnika u spremnik je 0101, a za zbrajanje 0000.

Ovdje je za napomenuti da svi viši programski jezici (C, FORTRAN, Pascal) su prenosivi s procesora na procesor, dok su strojni odnosno simbolički jezici vezani uz određeni procesor.

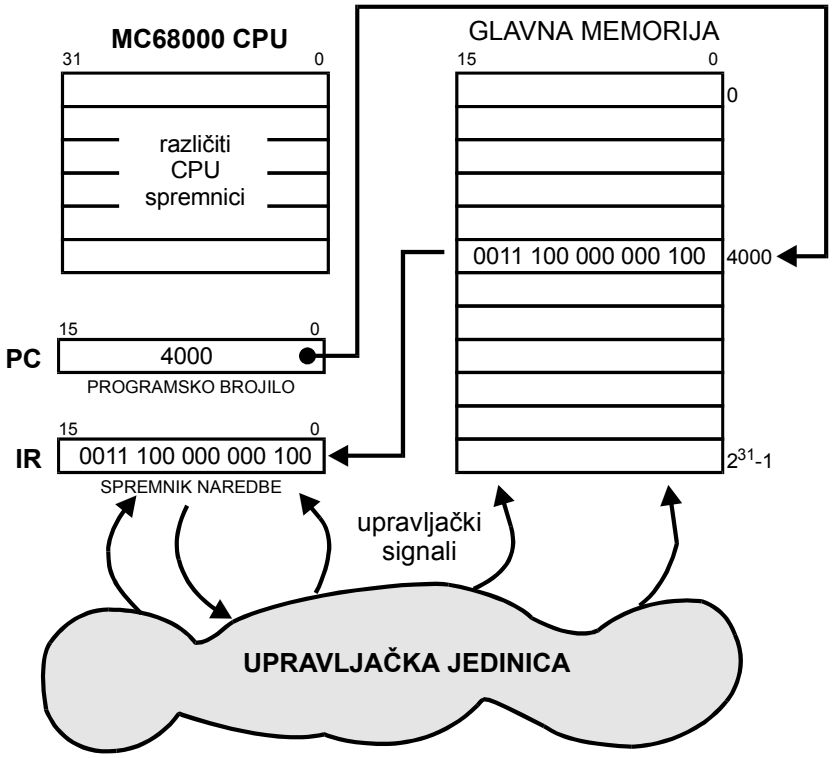
1.3.1. Princip programa pohranjenog u memoriji

Danas praktički svaki procesor ima svoj jedinstven strojni jezik. Ipak svim procesorima zajedničko je svojstvo da program koji izvode kao i podaci nad kojima se obrada izvodi su pohranjeni u memoriji računala. Iako to danas izgleda sasvim prirodno, za napomenuti je do to nije bio slučaj s prvim računalima. Naime prva su se računala programirala direktno sa operatorske konzole prespajanjem ulaza spremnika. Koncept pohrane programa i podataka (engl. *stored program concept*) uveden je sredinom 40tih godina i ostao je temeljni princip rada i današnjih računala.

Naredbe pohranjene u memoriji računala kod većine procesora izvode se sekvencijalno naredba po naredba. Procesor prvo dohvaća (engl. *fetch*) jednu po jednu naredbu iz memorije, privremeno je pohranjuje u poseban spremnik tzv. spremnik naredbe (engl. *instruction register*, skr. IR) a potom CPU izvodi (engl. *execute*) naredbu. Ovakav temeljni princip rada procesora naziva se dohvati i izvedi ciklus (engl. *fetch & execute*). Ovi ciklusi ponavljaju se od trenutka ukapčanja procesora sve do njegovog zaustavljanja.

Za izvođenje ove operacije procesor mora imati informaciju gdje se nalazi sljedeća naredba koju treba izvesti. U tu svrhu koristi se poseban spremnik tzv. programsko brojilo (engl. *program counter*, skr. PC) čiji sadržaj je adresa sljedeće naredbe koju treba dohvati. Ovaj spremnik ponekad se naziva i pokazivač naredbe (engl. *instruction pointer*, skr. IP).

Postupak dohvata naredbe može se objasniti na primjeru procesora MC 68000 kod kojeg su sve naredbe dužine 16 bita pa se nakon što je dohvaćena naredba koju je potrebno izvesti automatski sadržaj programskog brojila poveća za dva (u primjeru iznosi 4002). Cjeloviti ovaj ciklus je pod nadzorom upravljačke logike koja generira odgovarajuće upravljačke signale koji ispravno vremenski vode cikluse dohvati i izvedi.



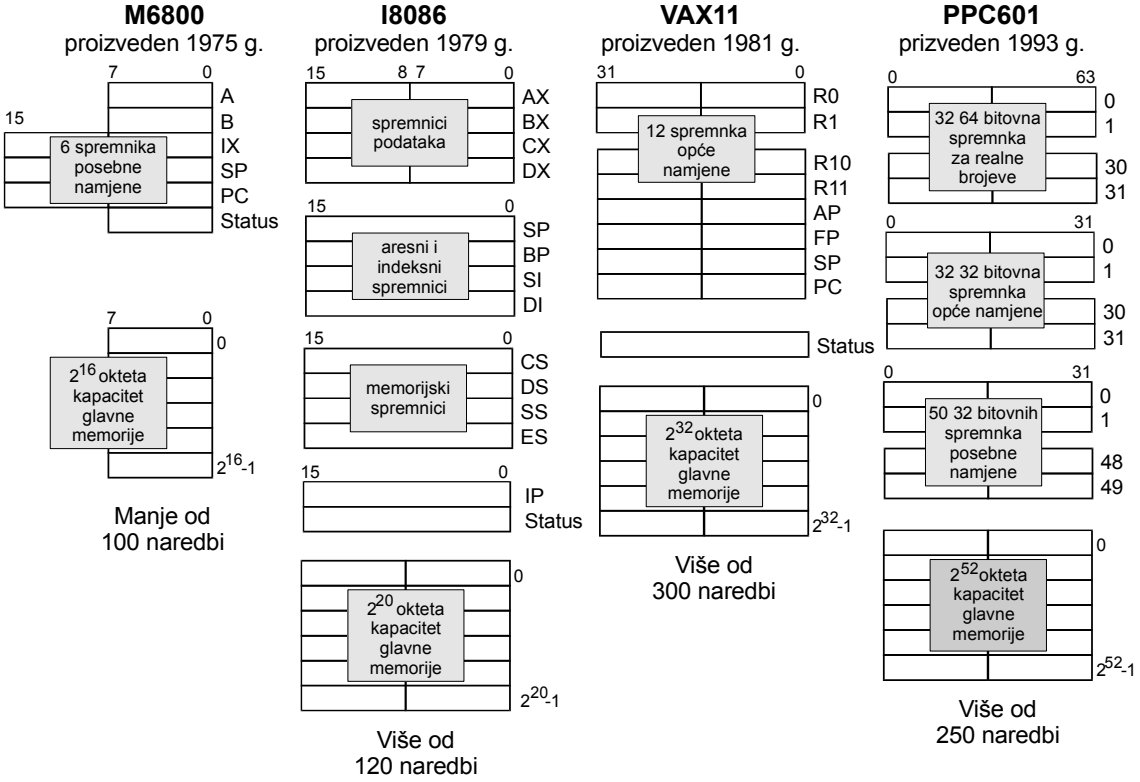
Slika1.2. *Ciklus dohvati i izvedi.*

1.3.2. Programerski model računala, Arhitektura skupa naredbi

Na ovoj razini svakako je interesantno objasniti kako programer vidi računalo. Pri tome važno je definirati da se skup operacije koje procesor može izvesti naziva i **skup naredbi** (engl. *instruction set*). Programer promatra računalo kroz skup naredbi zajedno s resursima računala koje mu stoje na raspolaganju, odnosno koje može koristiti pomoću skupa naredbi. Skup naredbi i resurs računala koje naredbe koriste naziva se **Arhitektura skupa naredbi** (engl. *Instruction Set Architecture*, skr. ISA). ISA obuhvaća skup naredbi procesora, njegovu memoriju i spremnike procesora koje koriste naredbe. Za napomenuti je da procesor može imati osim spremnika koje naredbe koriste i spremnike za privremenu pohranu naredbi, podataka i međurezultata koji su transparentni (nevidljivi) programeru.

Na slici su prikazani programerski modeli triju različitih generacija računala: procesori iz 70tih. g. Motorola 6800 (8 bitovni mikroprocesor) i Intel 8086 (16 bitovni mikroprocesor), VAX11 od Digital Equipment Corporation iz 80tih godina i konačno PowerPC601 od IBM-Motorola-Apple consortium iz 90tih. Za primijetiti je povećanje u broju spremnika i memorijskom kapacitetu. Broj naredbi procesora također se povećavao, ali već kod PowerPC za primijetiti je smanjenje broja naredbi što je posljedica trenda ka računalima sa smanjenim brojem naredbi (engl. *Reduced Instruction Set Computer*, skr. RISC). Ovaj koncep “manje je više” postavio je u 70tim g. David Patterson s Berkeley University of California. Danas je

on temelj mnogih suvremenih računala nasuprot konceptu složenog skupa naredbi (engl. *Complex Instruction Set Computer*, skr. CISC).



Slika 1.3. *Programerski model različitih procesora.*

Bez obzira da li skup naredbi sastoji se od 300 ili 50 naredbi one se u osnovi mogu podijeliti na: naredbe za prebacivanje podataka (*data movement*), aritmetičke i logičke naredbe (*arithmetic and logic*), i naredbe za upravljanje tokom (*control flow*). U tablici su prikazane sve tri skupine naredbi u Cu i VAX simboličkom jeziku.

Tip naredbe	C	VAX simbolički jezik
Prebacivanje podataka	a = b	MOV b, a
Aritmetičke i logičke	b = c + d*e	MPY d, e, b ADD c, d, b
Upravljanje tokom	goto LBL	BR LBL

Kako je za primijetiti neke naredbe u višem jeziku (Cu) preslikavaju se u više naredbi u strojnog (simboličkog) jezika. Omjer preslikavanja je obično 1:4 ili više. Zadatak programa prevodioca je izvršiti preslikavanje iz višeg u strojni jezik na koliko je to moguće optimalan način.

Proizvođači procesora izdaju detaljna uputstva (*manual*) za svoje proizvode u kojima su sadržane informacije o svim sklopovskim detaljima procesora kao i detalji o svakoj naredbi (trajanje, na koje spremnike utječe i sl.).

Izvođenje programa praktički je prijelaz računala iz početnog stanja u konačno stanje. Stanje računala (engl. *machine state*) opisano je sadržajem svih spremnika procesora (dostupnih i nedostupnih programeru) kao i sadržajem svih memorijskih lokacija. Pod pojmom stanje procesora (engl. *processor state*) podrazumijeva se sadržaj svih spremnika procesora, dok pod pojmom stanje memorije (engl. *memory state*) podrazumijeva se sadržaj memorije. Za ispravno izvođenje

programa značajna je ispravna promjena kao i obnavljanje starih stanja računala naročito za vrijeme poziva potprograma i obrade prekida.

Tehnika programiranja podrazumijeva razbijanje programa na manje cjeline koje je jednostavnije programirati i ispitati, te kao takove koristiti u različitim programima. Zato su učinkoviti mehanizmi poziva procedura bitna značajka svakog procesora. Prekidi (*interrupts*) su asinkroni prekidi normalnog izvođenja programa uvjetovani nekim vanjskim događajem ili nekom od programskih pogrešaka u programu. Procesor mora također imati moćne mehanizme za brzu i učinkovitu obradu prekida.

1.3.3. Naredbe procesora i tipovi podataka

Raznolikost tipova podataka kao i struktura podataka prisutnih u višim jezicima izostavljena je na razini strojnog jezika.

Podaci koje obrađuju viši jezici (engl. *High Level Language*, skr. HLL) su nizovi bitova, cjelobrojni i realni brojevi različite preciznosti, logičke varijable, nizovi znakova te različite složene strukture podataka (polja, zapisi). Viši programski jezici imaju ugrađen mehanizam prepoznavanja tipova podataka tako da za vrijeme prevođenja programa (složenog preslikavanja u niz strojnih naredbi) ispituju ispravnu uporabu svakog pojedinog podatka, unaprijed određuju korištenje memorije te dojavljuju (korigiraju) pojedine pogreške u programiranju. Također mnogi viši jezici imaju ugrađene i složene mehanizme optimiranja strojnog koda i korištenja memorije.

Na razini strojnih odnosno simboličkih naredbi ovi mehanizmi ne postoje. Procesor naredbe i podatke vidi kao niz bitova pohranjene u memoriji računala. Dozvoljeni su samo oni tipovi podataka koje može obraditi jedna naredba. Tako npr. prvobitni procesori kao što je Motorola 6800 obradu izvode samo nad 8 odnosno 16 bitovnim cjelobrojnim podacima. Moderniji procesori obrađuju 8, 16 ili 32 a neki i 64 bitovne cjelobrojne veličine kao i realne brojeve određene preciznosti (32 64 ili 80 bita). Nažalost provjera ispravnosti korištenja podataka ne postoji na razini strojnog jezika. Tako npr. 32 bitovna riječ može biti naredba, cjelobrojna veličina, realni broj ili 4 ASCII znaka.

Zadatak programera koji izrađuju programe prevodioce (engl. *Compiler*) je definirati preslikavanje iz višeg u strojni (simbolički) jezik. Ovim zadatkom praktički programeri programa prevodilaca svrstavaju se u skupinu programera u simboličkom jeziku. Funkcija preslikavanja znatno je složenije nego kod simboličkog jezika.

1.3.4. Alati za rad s procesorima

Iako programer u simboličkom jeziku praktički programira procesor na najnižoj razini postoje određeni alati koji mu olakšavaju posao.

1. Program prevodilac (A*ssembler*) T
   1.1. Translatira (preslikava jedan na jedan) simboličke u strojne naredbe.
   1.2. Omogućava programeru da odredi gdje će u memoriji biti pohranjen program a gdje podaci.
   1.3. Translatira simboličke adrese (LBL) u stvarne fizičke memorijske adrese.
   1.4. Ispituje sintaksu primijenjenih naredbi i dojavljuje pogreške kao i nekonzistentnu uporabu varijabli.

2. Program za povezivanje (engl. *Linker*)

   2.1. Povezuje više programskih modula u cjelinu prikladnu za izvođenje. Ovim se omogućava programeru da modularno izrađuje programe, da nezavisno ispituje programske module što značajno olakšava otkrivanje pogrešaka. Ovaj program zadužen je i da osigura razmjenu varijabli između različitih modula.

3. Program za otkrivanje pogrešaka (engl. *Debugger*) ili monitor program.

   3.1. Ovaj program olakšava programeru testiranje programa. Program se može izvoditi korak po korak (naredba po naredba) (*step by step*) uz stalnu provjeru međurezultata (stanja svih spremnika) ili izvođenje dijela programa (*break point*), te izmjenu sadržaja spremnika i memorijskih lokacija tijekom izvođenja programa.

4. Razvojni sustavi

   4.1. Skup programskih i sklopovskih pomagala koje stoje na raspolaganju programeru.

1.3.5. Zašto programirati u simboličkom jeziku

Danas je raširenо razmišljanje da programiranje u strojnom jeziku odumire, ali ipak na nekim područjima ostaje nezamjenjivo i to:

- Projektanti računala moraju dobro poznavati procesore koje koriste, njihov skup naredbi kako bi mogli što optimalnije projektirati čitav sustav. Na ovaj način projektant uočava posebnosti svakog procesora i prilagođava ostale komponente sustava njegovim mogućnostima.
- Programeri koji pišu jezike prevodioce moraju dobro poznavati skup naredbi procesora za koji pišu program prevodilac kako bi prevedeni kod bio ispravan i što djelotvorniji. Današnji procesori također zahtijevaju da programeri programa prevodilaca dobro poznaju i arhitekturu računala (struktura memorije, brza memorija, cjevovodi i sl.) kako bi ispravno i optimalno preveli naredbe iz višeg u strojni jezik.
- Bez obzira na dorađenost modernih jezika prevodilaca, oni ipak ne generiraju u svim slučajevima optimalan kod pa tako kod programiranja kritičnih (što se tiče brzine izvođenja ili zahtijeva na memoriju) odsječaka koda često se koristi simboličkim jezikom.
- Mikrokontroleri su danas postali sastavni dio velikog dijela najobičnijih uređaja (kućanskih aparata, fotografskih aparata, dječjih igračaka, automobila i sl.). Ovi procesori obično su projektirani (optimirani) za posebnu namjenu pa za njih nije isplativo izrađivati programe prevodioce. Također i program koji oni izvode je specifičan i često dosta opterećen različitim komunikacijama s okolinom ili korisnikom.

1.4. Pogled projektanata arhitekture računala

Težište rada projektanta arhitekture računala je projektirati računalo kao cjeloviti sustav s određenim performansama. Njegov zadatak je projektirati sustav s optimalnim performansama u kontekstu zadataka koji mora obavljati. Pri tome je potrebno voditi računa o mnogobrojnim, često jako oprečnim zahtjevima kao što su cijena, veličina, mehanička i termička svojstva, vijek trajanja, utjecaj na vanjske smetnje, dobavljivost komponenata itd.

Projektant arhitekture računala projektira sustav iz skupa blokova (procesor, memorija, ulazno/izlazne jedinice, sabirnice itd.). Njegov zadatak je odabrati optimalan skup blokova i povezati ih na optimalan način (sa stajališta cijene, performansi i sl.). Projektirajući računalo na ovaj način projektant se spušta na razinu blokova koje također mora projektirati na odgovarajući način kako bi ostvario što bolje performanse cjelovitog sustava. Tako npr. mora projektirati dijelove procesora kao što su aritmetičko/logička jedinica, sustav za dohvat i dekodiranje naredbi i sl.

1.4.1. Jednostavna slika računarskog sustava

Na najvišem stupnju apstrakcije računalo se može promatrati kao skup različitih funkcionalnih jedinica (ulazno/izlazne jedinice, memorija i procesor). Svaka funkcionalna jedinica može sadržavati više podsustava, npr. U/I sustav sastoji se od tipkovnice, miša, monitora, a memorija od brze memorije (*Cache*) glavne memorije, RAM diska, magnetskog diska, optičkog diska, magnetske trake i na kraju procesor. Sustav ne mora imati samo jedan procesor, nego obradu može izvoditi više istih ili više različitih procesora (glavni procesor, floating-point procesor, U/I procesor itd.).

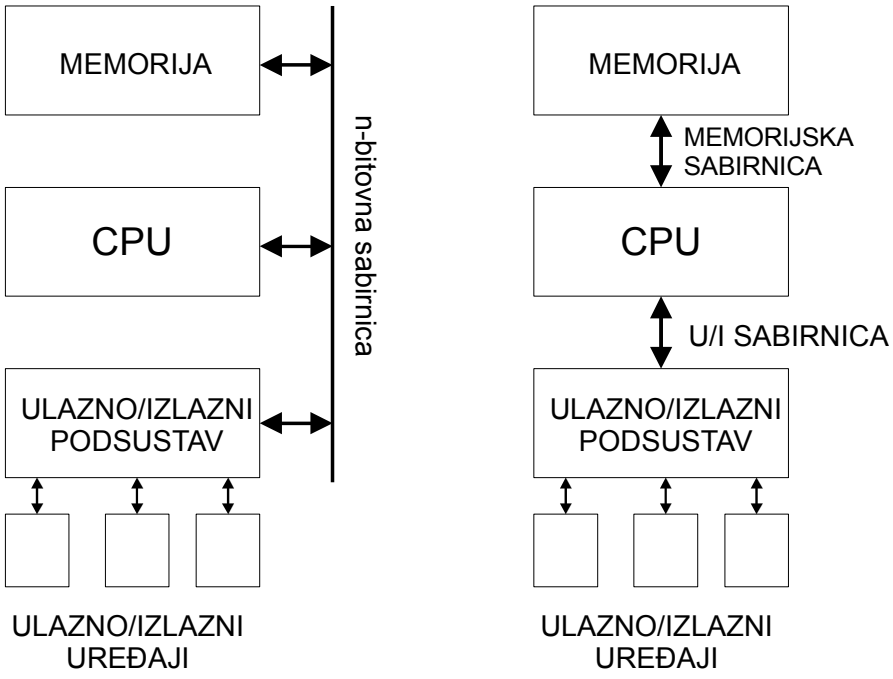
Svi dijelovi sustava dostupni su i upravljani preko ISA. To je i razlog da se projektant arhitekture računala prvenstveno mora pozabaviti projektiranjem arhitekture skupa naredbi. ISA stvara sučelje između programa i dijelova računala (sklopovlja) na kojima se obrada izvodi.

Procesor radi na principu da dohvati i izvedi. Iz tog razloga performanse računale jednako ovise o brzini kojom procesor izvodi naredbu, a i o brzini odziva memorije. Ukoliko se računalo proširi i dodatnim funkcionalnim blokovima (npr. U/I uređajima) tada i oni imaju doprinos performansama sustava. Na performanse računala utječe još jedan čimbenik koji se često zanemaruje a to je način međusobnog povezivanja funkcionalnih jedinica. Ovo vrijedi na razini sustava (računala) i na razini podsustava (CPU).

Funkcionalne jedinice povezani su spojnim vodovima tzv. sabirnicama (engl. *buses*), a obično više jedinica dijeli zajedničke međuveze. Ovim pristupom više funkcionalnih jedinica vremenski dijeli spojne putove za međusobnu komunikaciju. Svojstvo vremenskog multipleksiranja spojnih vodova postavlja i posebne zahtjeve na izradu jedinica koje ih koriste.

Preko sustava sabirnica prenose se podaci (podaci + naredbe) i različiti upravljački signali. One mogu biti serijske (prijenos se odvija bit po bit) ili paralelne (više bita istovremeno).

Na slici su pokazani primjeri jedno- i dvo-sabirničkog računala.



a) jednosabirnički sustav **b) dvosabirnički sustav**

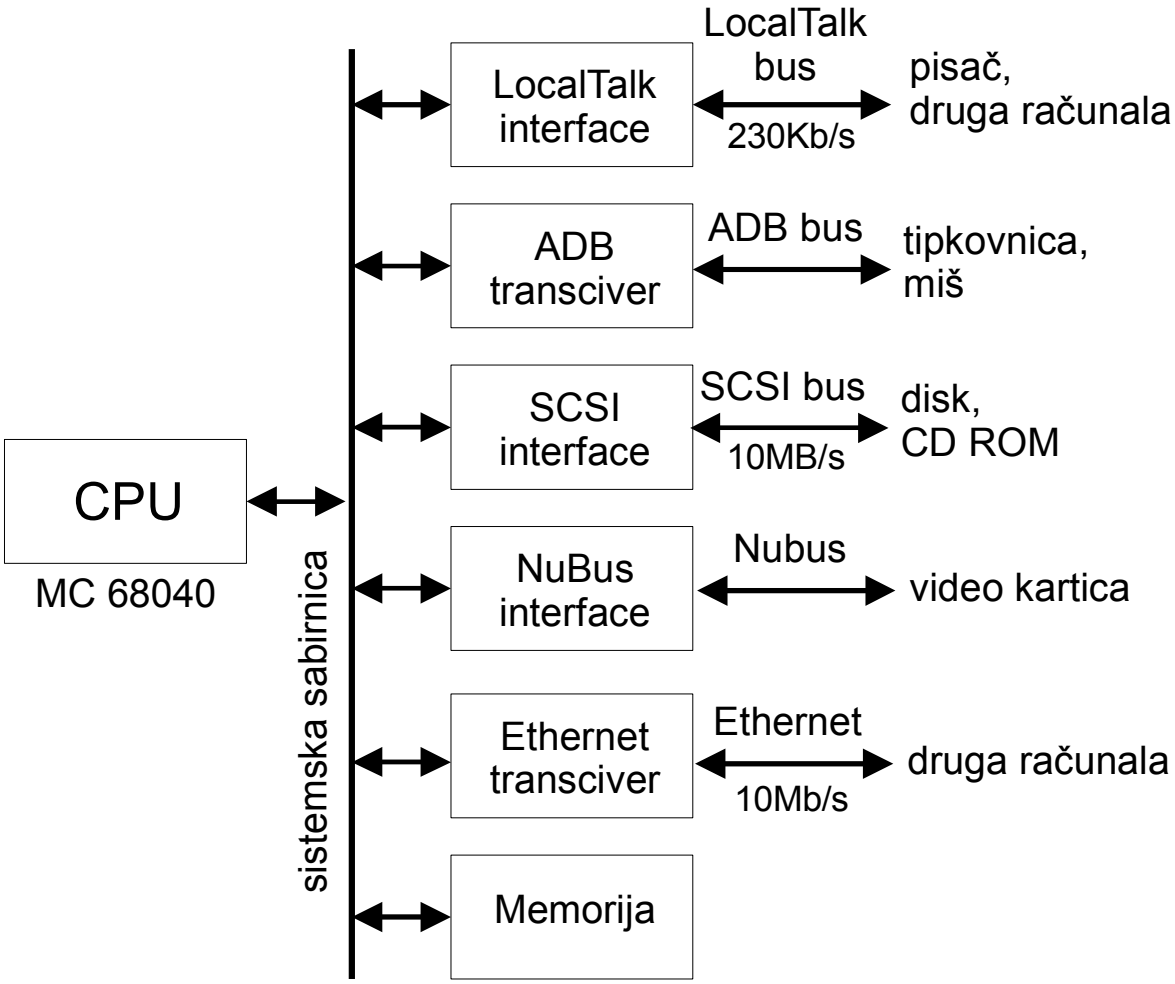
Slika 1.4. *Jednostavna jedno- i dvo-sabirnička arhitektura.*

Kod jedno-sabirničkog sustava procesor pristupa memoriji i ulazno/izlaznim uređajima preko zajedničke sabirnice. Ovakvo rješenje je relativno jednostavno. Procesor aktiviranjem određenih signala preko sabirnice određuje da li pristupa memoriji ili U/I uređajima. Također moguć je i mehanizam da U/I uređaji direktno komuniciraju s memorijom uz uvjet da je procesor odspojen (električki) od sabirnice.

Poboljšanje sveukupnih svojstava postiže se odvajanjem pristupa procesora memoriji i U/I uređajima. Ovim je moguće paralelno izvođenje U/I operacija i pristupa memoriji.

Stvarni sustavi su značajno složeniji od pokazanih primjera. Većina općenamjenskih računala ima više sabirničkih sustava preko kojih pristupa unutarnjim funkcionalnim jedinicama (CPU, memoriji, U/I uređajima disku, video memoriji), te ostvaruje komunikaciju s vanjskim svijetom (druga računala).

Na slici je prikazan primjer računala Apple Quadra 950. Sustav je projektiran za brzi procesor Motorola MC 68040.



Slika 1.5. *Sustav sabirnica računala Apple Quadra 950.*

LocalTalk sabirnica je serijska sabirnica brzina rada 230 Kb/s i koristi se za relativno sporu komunikaciju između računala i pisača ili drugih računala.

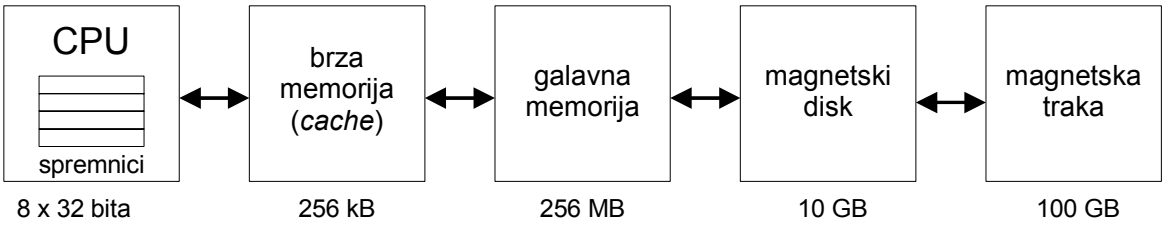
ADB je spora serijska sabirnica (projektirana od Apple corp.) namijenjena komunikaciji sa sporim ulazno/izlaznim uređajima kao što je miš, tastatura.

SCSI sabirnica (engl. *Small Computer System Interface*, skr. SCSI) je 16 bitovna paralelna sabirnica preko koje se prenose podaci brzinom od nekoliko MB/s. SCSI je praktički standardizirana sabirnica što podrazumijeva da je strogo definirano značenje svake linije, naponski nivoi i vrijeme odziva (*timing*). Na ovu sabirnicu spajaju se brzi U/I uređaji kao što je magnetski disk, CD ROM.

NuBus je paralelni priključak (*slot*) na matičnoj ploči računala na koji se prikapča video kartica, proširena memorija, grafički procesor i različite druge kartice posebne namjene. Osigurava brzu komunikaciju između procesora i navedenih uređaja.

Ethernet međusklop je komercijalni standard za prikapčanje računala u lokalnu mrežu računala do udaljenosti od oko 300m. Brzine rada su 10 Mb/s preko 100 Mb/s dok se danas već govori i o Gb/s Ethernetu.

Projektant arhitekture vidi memorijski sustav kao hijerarhijsku organizaciju s brzom skupom memorijom malog kapaciteta te sporijim ali jeftinijim memorijskim sklopovima većeg kapaciteta (slika 1.6.).



Slika 1.6. *Hijerarhijska struktura memorije.*

Informacije se prebacuju između različitih memorijskih razina, a projektant mora osigurati mehanizme da odgovarajuća informacija bude prisutna na određenoj razini u danom trenutku.

U/I ili periferni uređaji ostvaruju vezu računala s vanjskim svijetom. Zato oni zajedno sa sustavom veza, sabirnica koje povezuju dijelove računala značajni su za cjelokupno funkcioniranje računala. Tako projektant mora dobro proučiti koje će komponente integrirati u računarski sustav jer o izboru diskova, memorije, sustava sabirnica te procesora značajno ovise sveukupne performanse sustava. Prilikom projektiranja projektant arhitekture mora dobro proučiti sve karakteristike svih komponenata sustava kako bi mogao pronaći kompromise između mnogih oprečnih zahtjeva (cijena, brzina izvođenja, potrošnja i sl.).

Projektant arhitekture računala prilikom projektiranja koristi se različitim alatima. Za vrijeme prvobitne faze projektiranja značajno pomažu programi za simuliranja rada računala tzv. simulatori. Simulator je program koji simulira rad računala kako bi se ispitale njegove performanse prije njegove realizacije.

Emulatori su naprotiv sklopovi kojima se sklopovski simulira rad računala. Ovakav pristup daje pouzdanije rezultate i bolji uvid u rad sustava ali je i znatno skuplji i u mnogim funkcijama ograničen.

Kada se realizira prototip računala primjenom niza ispitnih programa procjenjuje se njegova funkcionalnost. Ovakvi programi kojima se ispituju karakteristike sustava nazivaju se *benchmark* programi.

Navedenim pomagalima te analizom toka podataka moguće je otkriti uska grla (*bottleneck*) u radu sustava koja se naknadno mogu otkloniti odgovarajućim zahvatima. Na ovaj način postiže se usklađenost dijelova računala kako neki od dijelova (podsustavi) ne bi predstavljali usko grlo radu računala. Posljednji korak u postupku projektiranja predstavlja odabir tehnologije kojom se realiziraju pojedini dijelovi računala (npr. procesor ECL ili HCMOS ili TTL, slično memorija itd.)

Naravno u cjelokupnom procesu projektiranja, projektant arhitekture mora biti u stalnom kontaktu s programerima (koji moraju pisati programe za računalo) i projektantima logičkih sklopova (koji moraju implementirati projekt računala). Naravno u cijelom ovom postupku potrebno je na neki način i formalizirati opise koji se koriste u ovom procesu komunikacije među različitim razinama. Sam opisni jezik ne daje obično dovoljnu razinu preciznosti. Zato se i uvodi jezik koji je sličan programskom jeziku, a koji formalno opisuje aktivnosti u računalu i omogućava kasnije jednostavnu implementaciju opisanih funkcija. Ovaj jezik naziva se **Opis prijenosa podataka između spremnika** (*Register Transfer Notation*).

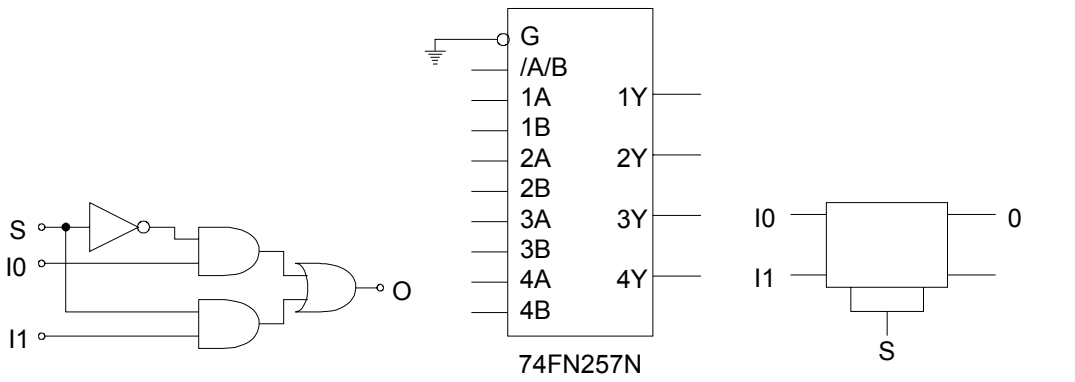
1.5. Pristup računalu projektanta logičkih sklopova

Projektant logičkih sklopova ima zadatak projektirati logičke sklopove kojima će se realizirati funkcije koje je predvidio projektant arhitekture računala. Naravno ove dvije razine projektiranja usko su povezane pa je uobičajeno u praksi da je projektant arhitekture ujedno i projektant logičkih sklopova, odnosno da ove dvije funkcije izvodi jedan te isti tim stručnjaka. Stroga podjela ovih funkcija imala bi u praksi mnoge negativnosti i mogla bi rezultirati npr. zahtjevima projektanta arhitekture takvim da se ne mogu realizirati na razini logičkih sklopova ili bi ta realizacija bila neprihvatljivo skupa. Zato su neophodne stalne korekcije i konzultacije između projektanata na ovim dvjema razinama.

1.5.1. Domena implementacije

Domena implementacije je skup svih sklopova koji stoje na raspolaganju projektantu logičkih sklopova da realizira postavljene mu zadatke. Tako u ovu domenu spadaju VSLI sklopovi realizirani bilo u TTL, ECL, CMOS, HCMOS, odnosno PLAs ili optički prekidači i sl. Ovi sklopovi mogu biti povezani bilo na tiskanoj pločici ili na siliciju kao jedinstveni integrirani sklop ili povezani kablovima.

Na nižoj razini implementacije projektant se u načelu ne mora opterećivati odabirom tehnologije izvedbe logičkih sklopova, nego je dostatna i analiza na razini logičkih funkcija i povezivanje apstraktnih logičkih sklopova. Naravno na višoj razini način realizacije logičkih sklopova potrebno je uzeti u obzir. Na slici 1.7. prikazan je apstraktni prikaza 2-1 multipleksera, kao i dvije različite razine implementacije ovog sklopa, Fast Schotky TTL i optički prekidač. Naravno svaki projektant na razini logičkih sklopova uvijek uz logičku funkciju (apstraktnu) ima na umu i praktičnu izvedbu sklopa.



Slika 1.7. *Različite domene implementacije 2-1 multipleksora.*

1.5.2. Značenje domene implementacije

Domena implementacije postaje naročito značajna u trenutku prijelaza s apstraktne razine logičkih vrata na konkretnu implementaciju. U ovom trenutku osim realizacije zadane funkcije postaju značajne i posebnosti sklopovlja kojima se realiziraju navedene funkcije: njihova brzina, pogonske sposobnosti (*Fanout Fanin*) potrošnja, gustoća pakovanja, povezivanje s ostalim logičkim sklopovima itd.

1.5.3. Razlika između klasičnog projektiranja logičkih sklopova i projektiranja logičkih sklopova računala

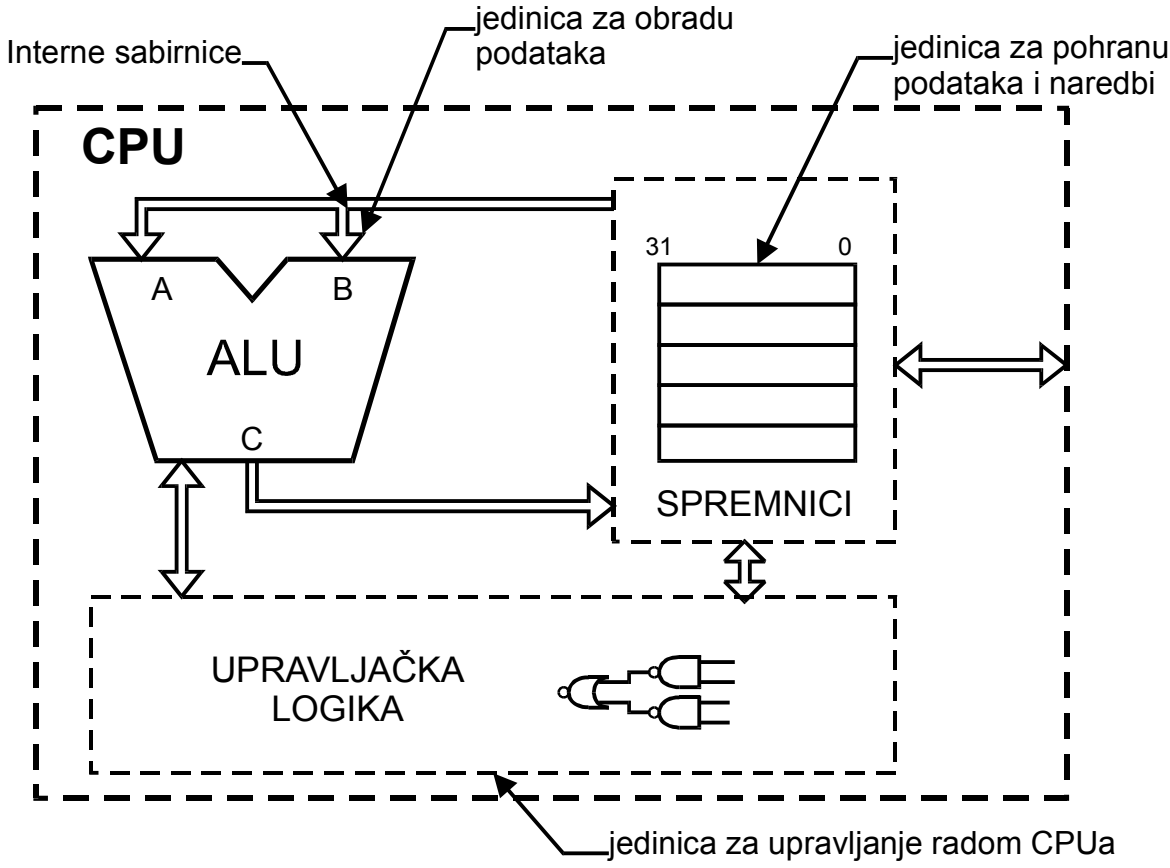
Kod projektiranja klasičnih logičkih sklopova korištene su poznate tehnike minimizacije ali koje nisu primjenjive na procesor jer je on presložen za ovakve postupke. Također računalo ili procesor se sastoji od čitavog niza modula koji imaju manje ili više međusobne povezanosti tako da se mogu zasebno projektirati uz uvažavanje posebnosti modula s kojima se povezuju.

Sama priroda digitalnih računala takva je da se razlikuju tokovi podataka (*data path*) i upravljački tokovi (*control path*). Projektiranje tokova podataka podrazumijeva projektiranje sklopova za uskladištenje podataka (spremnici) i njihovih međuveza. Upravljački tokovi su signali koji upravljaju tokom informacija za potrebe toka podataka.

Dok projektanti klasičnih logičkih sklopova koriste NI, NILI vrata i D flip-flopove, projektant logičkih sklopova računala koristi multipleksere, dekodere, spremnike. Ovim je značajno doprineseno jednostavnosti procesa projektiranja računala jer se skupine logičkih vrata i flip-flopova jednostavno zamjenjuju crnim kutijama koji imaju zahtjeve na ulaze i ponašanja na izlazu. Tako se na višoj razini može aritmetička jedinica zamijeniti crnom kutijom koja ima određene ulaze i izlaze s određenim ponašanjem.

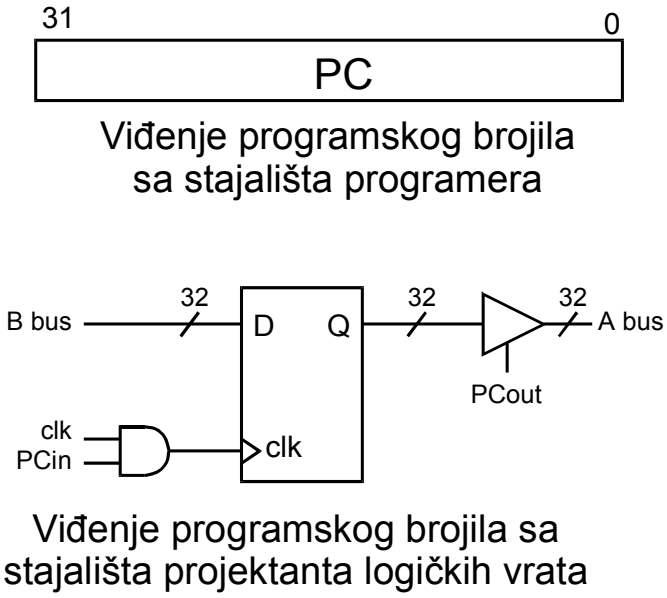
1.5.4. Prikaz CPU

Na slici 1.8. prikazani su osnovni moduli CPU-a.



Slika 1.8. Osnovni moduli CPU-a.

Pogled projektanta logičkih sklopova je znatno dublji od onoga što vidi programer. Programer vidi samo spremnike i naredbe dok projektant logičkih sklopova mora uzeti u razmatranje i sve spremnike za privremenu pohranu podataka, ALU, upravljačku logiku itd. Najvažniji segment na koji projektant logičkih sklopova mora obratiti pažnju su mehanizmi kojima se upravljaju procesorom. Ovaj proces započima s programskim brojilom koji sadrži adresu naredbe koju treba dohvatiti iz memorije i pohraniti je u spremnik naredbe. Sljedeća slika prikazuje kako programsko brojilo vidi programer a kako projektant logičkih sklopova. Programsko brojilo realizirano je kao 32 D bistabila. Izlazi bistabila dovode se na A sabirnicu preko međusklopa s tri stanja (*buffer*) kada je narinut signal PC_{out}. S dtruge strane sadržaj s B sabirnice upisuje se u programsko brojilo na signal PC_{in} sinkroniziran sa sistemskim taktom. Ovdje je vidljiv spoj između toka podataka i upravljačkih signala.



Slika 1.9. *Različiti pogledi na programsko brojilo.*

1.6. Povijesni pregled razvoja računala

Charles Babbage (1792-1871) projektirao je prvi mehanički stroj za računanje koje zbog složenosti nije nikad realizirano.

George Boole (1815-1864) razvio je matematički aparat nazvan po njemu koji je temelj računarske aritmetike.

Claude Shannon s MIT je u ranim 30-tim godinama ovog stoljeća postavio je temelje matematičke teorije diskretnih sustava, diskretne obrade signala i teorije komunikacija.

Prvo relejno računalo napravio je George Stibitz iz Bell Telephone Lab, a paralelno u Njemačkoj i Konrad Zuse. Ova računala imala su memoriju i bila su programabilna

1. **generacija digitalnih računala** (1946-59) koristila je elektronske cijevi, releje i živine linije za kašnjenje. Prvo ovakvo računalo bilo je ENIAC (*Electronic Numerical Integrator & Computer*). Sastojalo se od nešto više od 18.000

elektronskih cijevi i 1.500 elektromagnetskih releja. Izvodio je oko 5.000 cjelobrojnih zbrajanja u sekundi, a programirao se sklopovskim prespajanjem. Za reprogramirati računalo (prespajanje žica) bio je potreban i cijeli dan. Koncept programa pohranjenog u memoriji razvio je John von Neumann. Prva računala praktički nisu imala ni strojni jezik.

2. **generacija digitalnih računala** (1959 – 1964) realizirana su diskretnim poluvodičkim komponentama. Tranzistori su pouzdaniji, znatno mannji sa smanjenom potrošnjom i brži od elektronskih cijevi. Tadašnja računala u stanju su izvesti naredbu za nekoliko μs, a memorijski kapaciteti su narasli na nekoliko KB.
3. **generacija digitalnih računala** (1964 – 1975) realizirana su integriranim sklopovima malog i srednjeg stupnja integracije. Predstavnik ove generacije računala bili su IBM 360/370 32 bitovna velika i srednja računala memorijskog kapaciteta od 64K do 16M. Kod ovih računala uvedeni su i magnetski diskovi koji omogućavaju izvođenje istovremeno više programa na jednom računalu. Cijena ovih računala kretala se od 100.000 $ do nekoliko miliona $. S druge strane DEC se tada pojavio sa svojim 8 i 16 bitovnim miniračunalima s memorijskim kapacitetom od 4K uz cijenu od ispod 20.000 $.
4. **generacija digitalnih računala** (1975 do danas). Intel i Texas Instruments polovinom osamdesetih godina proizvodili su i prodavali u malim količinama četri bitovne procesora ali prava ekspanzija procesora u jednom integriranom krugu može se smatrati pojavom Intelovog 8 bitovnog procesora 8080 i Motorolinog procesora 6800. Oba procesora imala su 8 bitovni akumulator, radili su s 8 bitovnim podacima i imali su 16 bitovnu adresnu sabirnicu. Od tog trenutka procesori se strahovito brzo razvijaju tako da su danas već dostupni 64 bitovni procesori koji izvode naredbe unutar nekoliko ns. Trend ovog razvoja se nastavlja.
5. **generacija digitalnih računala** obično se smatraju paralelna računala s posebnim arhitekturama.