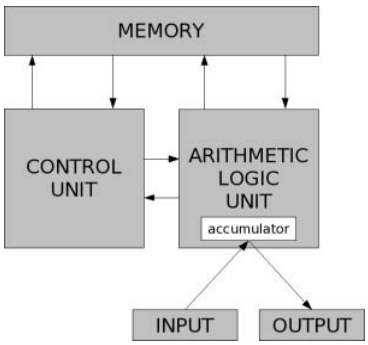
1. Von Neumannov koncept racunala



Utjecaj Von Neumannove arhitekture: Imperativni programski jezici logikom slijede Von Neumannovu arhitekturu. Izvodi se slijed naredbi koje se dohvaćaju iz spremnika. Slijed naredbi mijenja se uvjetnim ili bezuvjetnim skokom. Naredbe mijenjaju sadržaj spremnika. Von Neumann je dokumentirao organizaciju ENIAC-a i zbog tog razloga sva računala koja imaju sličnu organizaciju ili arhitekturu nazivaju se računala sa "von Neumann" arhitekturom.

Odlike von Neuman arhitekture su definirane s četiri svojstva :

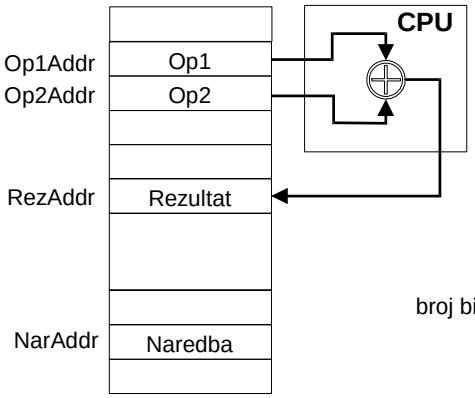
- programi i podaci koriste jednu zajedničko glavno spremiste ili memoriju
- glavno spremište ili memorija je sekvencijalna
- glavno spremište ili memorija je jedno dimenzionalna
- značenje ili kako se primjenjuju podaci nije spremljeno s podacima

Von Neumann računala takoder imaju slijedeće podskupove koje su također prisutne u ostalim računalnim arhitekturama:

- Aritmetičko-logička jedinica ili skraćeno ALU
- Upravljačka jedinica - jedinica koja pretvara naredbe u signale unutar računala
- Spremište ili memorija
- Ulazno/izlazne jedinice - jedinice koje čine sučelje s ljudima ili ostalim dijelovima računala, npr. sekundarnim spremištima (hard disk, floppy disk)

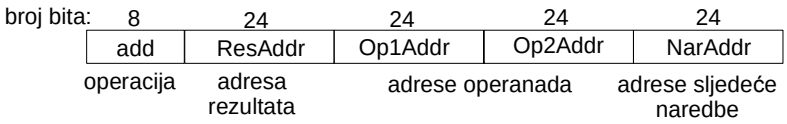
2. Formati naredbi

a) Naredba sa četiri adrese i opcode specificira eksplicitno adrese četiri posljednja podatka (adrese dvaju operanada, rezultata i sljedeće naredbe).



add, Rezultat, Op1, Op2, Naredba

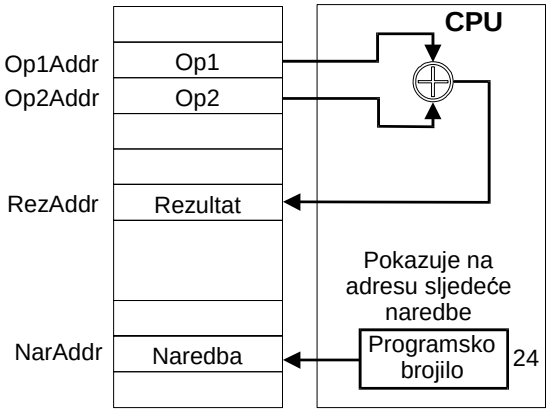
FORMAT NAREDBE



Nema spremnika procesora kojima programer može direktno pristupati. Oba operanda dohvaćaju se direktno iz memorija, a rezultat operacije pohranjuje se direktno natrag u memoriju. Adresa sljedeće naredbe eksplicitno je zadana tekućom naredbom. Ukupna duljina naredbe 13 okteta.

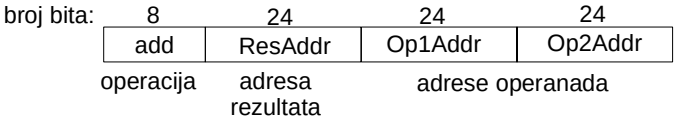
b) Naredba s tri adrese i opcode. Programsko brojilom (PC) je posebni spremnik procesora koji pokazuje na adresu sljedeće naredbe.

Naredba i dalje sadrži adrese operanada i adresu rezultata.

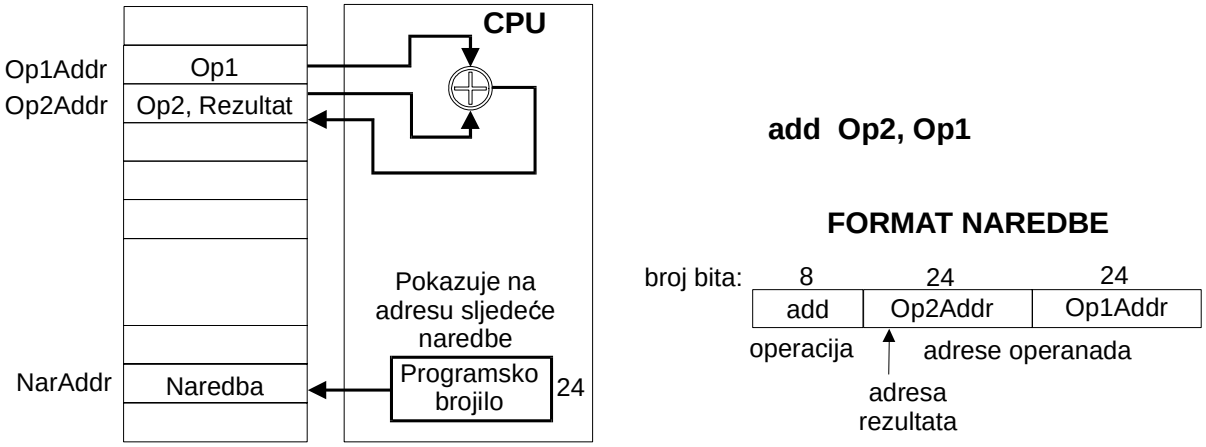


add, Rezultat, Op1, Op2

FORMAT NAREDBE

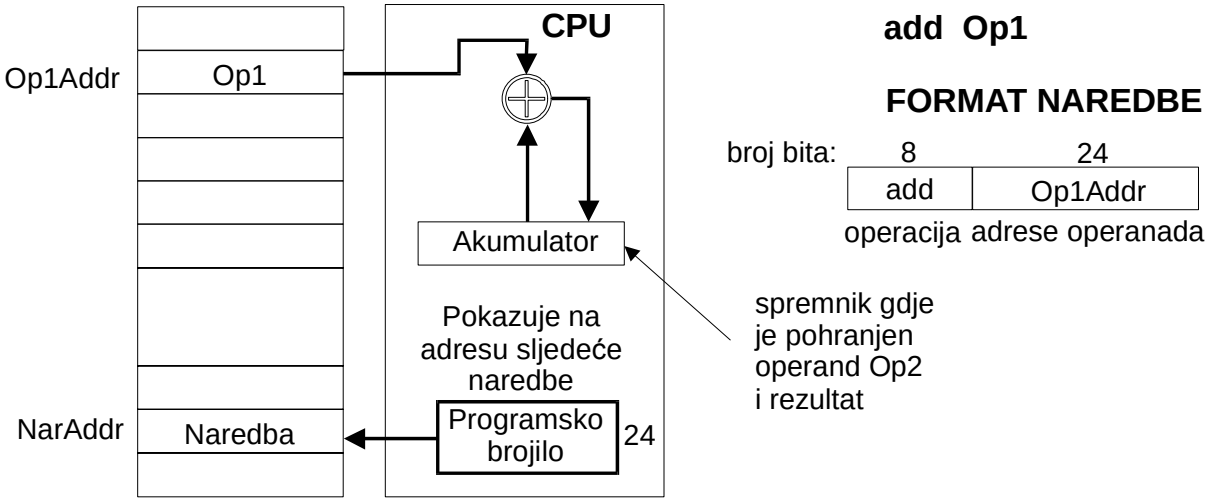


c) Naredba s dvije adrese i opcode. Smanjenje duljine naredbe može se postići ako se rezultat operacije pohrani na mjesto drugog operanda. Ovim je potrebno unutar naredbe specificirati samo operacijski kod i adrese prvog i drugog operanda. U ovom slučaju dužina naredbe se smanjuje na 7 okteta, a izvođenje na 6 pristupa memoriji.



d) Naredba s jednom adresom i opcode, procesor temeljen na akumulatoru.

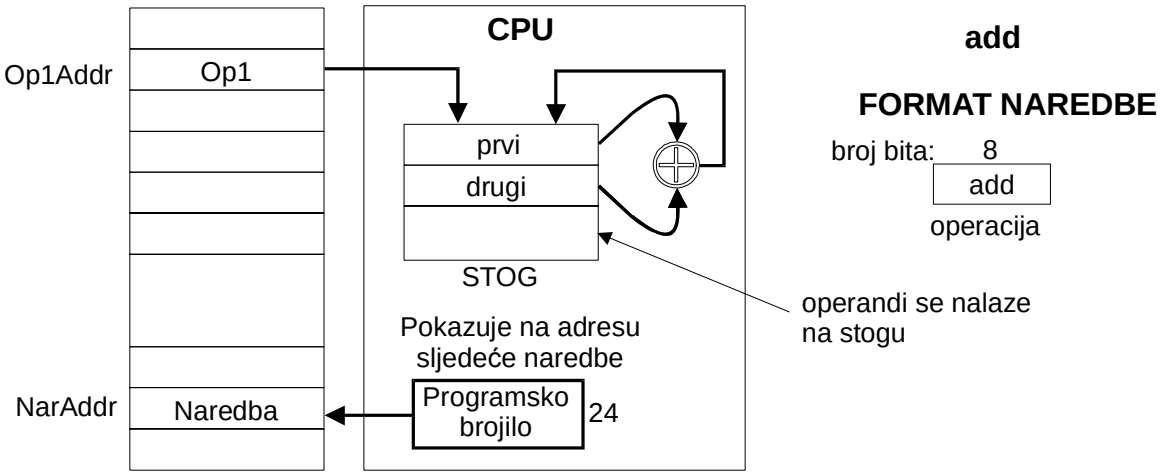
U akumulator se pohraniti jedan operand, a po obavljanju operacije rezultat.



Ovakav procesor zahtijeva posebne naredbe kojima se operand upisuje u akumulator (**load**), te naredba kojom se rezultat naredbe pohranjuje natrag u memoriju (**store**).

e) Naredba s nula adresa i samim opcode, procesor temeljen na stogu.

U okviru naredbe nije potrebno specificirati niti adrese operanada niti adresu rezultata. Operandi se stavljaju na stog posebnom naredbom **push**. Aritmetička jedinica očitava operande sa stoga i stavlja rezultat operacije na stog. Dovoljno je unutar naredbe specificirati operacijski kod.



Nedostatak ovakvih procesora je što se operandi uvijek moraju nalaziti na prve dvije lokacije stoga. Sama naredba za zbrajanje ima najmanje moguću dužinu i ne zahtijeva pristup memoriji, ali za izvođenje zbrajanja potrebne su dvije **push** naredbe, **add** naredba i jedna **pop** naredba. **push** i **pop** naredba mora uz operacijski kod sadržavati i adresu operanda.

3. Tipovi naredbi

Jednostavan procesor sa smanjenim skupom naredbi ima:

- 32 32 bitna spremnika opće namjene,
- programsko brojilo PC
- spremnik naredbe IR.

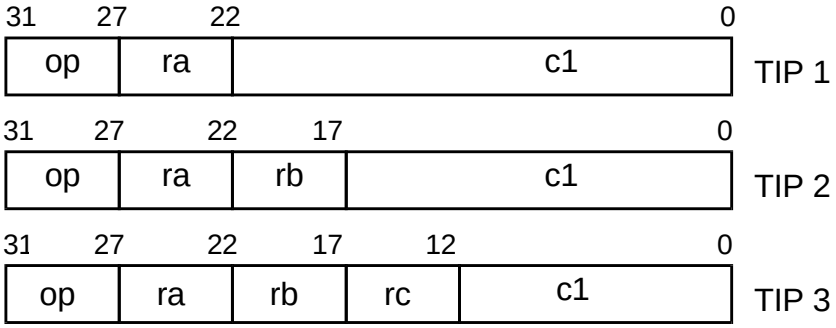
Memoriji može pristupati samo preko naredbi **load** i **store**.

Efektivna memorijska adresa je duljine 32 bita.

Razlikuju se tri tipa naredbi:

- TIP 1 koji uz operacijski kod ima specificiran jedan spremnik i konstantu,
- TIP 2 koji uz operacijski kod ima specificirana dva spremnika i konstantu,
- TIP 3 koji uz operacijski kod ima specificirana tri spremnika i konstantu.

Pretpostavlja se da će procesor izvoditi svega 32 naredbe pa je za operacijski kod potrebo 5 bitova.



4. Adresni modovi

a) **Neposredno adresiranje**, (*immediate addressing*) koristi se da bi se pristupilo konstanti koja je sastavni dio naredbe. Koristi se za dobavu operanda naredbe, a ne može se koristiti za pohranu rezultata.

b) **Direktno adresiranje** (*direct addressing*) je kada je adresa operanda sastavni dio naredbe. Ovaj način adresiranja može se koristiti i za dohvat operanda i pohranu rezultata.

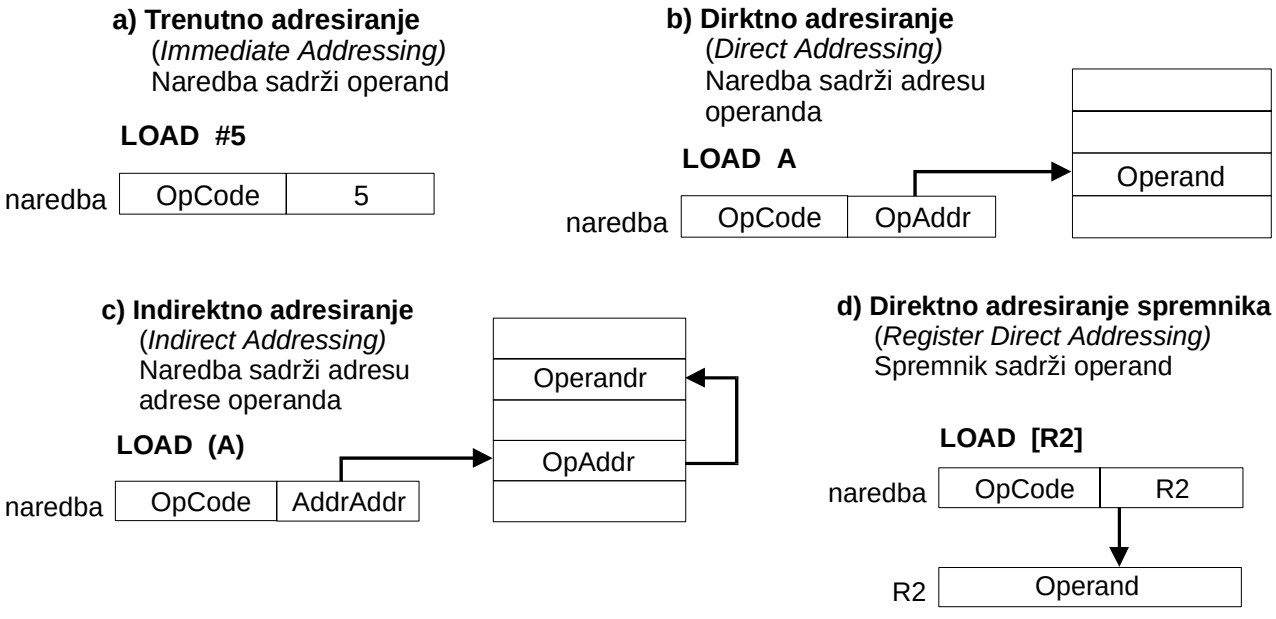
c) **Indirektno adresiranje** (*indirect addressing*), konstanta koja je sastavni dio naredbe je adresa memorijske lokacije na kojoj je upisana adresa memorijske lokacije s koje se operand namjerava očitati ili na koju se rezultat želi upisati. Ovim modom implementiraju se pokazivači. Za manipulaciju s podatkom potrebna su dva pristupa memoriji, prvo dohvat pokazivača tj. adrese gdje je pohranjen podatak s kojim se manipulira, a zatim dohvat operanda ili upis rezultata u memoriju.

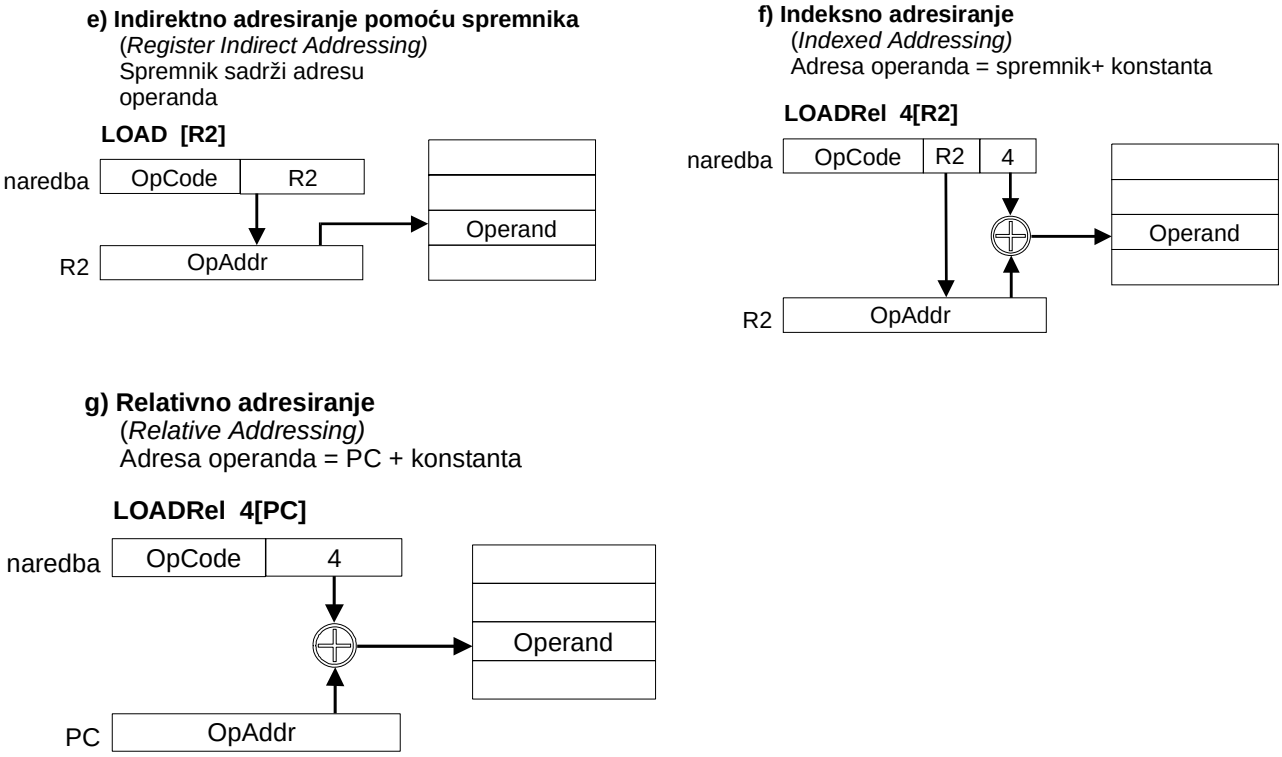
d) **Direktno adresiranje spremnika** (*register direct mode*), podatak je upisan ili se upisuje u spremnik čija adresa je sastavni dio naredbe.

e) **Indirektno adresiranje pomoću spremnika** (*register indirect mode*), adresa operanda pohranjena je u spremniku čija adresa je sastavni dio naredbe. Ovaj adresni mod koristi se obično za sekvencijalni pristup elementima polja pohranjenim u memoriji. Početna adresa polja pohranjuje se u spremnik te se nakon svakog pristupa elementu polja sadržaj spremnika poveća za dužinu elementa.

f) **Indeksno adresiranje** (*indexed mode*). Adresa se dobiva zbrajanjem konstante koja je obično sastavni dio naredbe i sadržaja spremnika adresiranog naredbom. Konstanta koja je sastavni dio naredbe obično se naziva **baza**, a sadržaj spremnika **pomak** ili **offset**. To je razlog da se ponekad ovaj način adresiranja naziva i **posmačno** ili **bazno adresiranje**.

g) **Relativno adresiranje** (*relative addressing mode*) je sličan indeksnom adresiranju uz razliku da se konstanti (bazi) koja je sastavni dio naredbe pridodaje sadržaj programskog brojila te sa na taj način formira efektivna adresa.





Slika 2.9. *Različiti adresni modovi.*

5. CISC – RISC, razlika

CISC

Kod CISC procesora postoje dodatne varijacije adresnih modova. CISC procesori nisu rezultat određene filozofije u pristupu projektiranja ralunala. Oni su rezultat težnji projektanata da u procesoru inkorporiraju što više mogućnosti npr. Adresnih modova, tipova naredbi i sl.

CISC je engleska kratica za Complex Instruction Set Computer i ona označava računarsku arhitekturu čija je filozofija gradnje ta da uvrsti što je moguće više naredbi na mikro razini - to jest na razini centralne jedinice (CPU).

CISC arhitektura ima nekoliko prednosti:

- naredbe su izvedene na razini *elektronike* u centralnoj jedinici ili u mikro programu
- izvršavanje naredbi je brzo

Nedostaci CISC arhitekture su:

- zbog izvođenja naredbi na mikro razini CPU postaje složeniji i skuplji za proizvodnju
- zbog složenosti elektronskih krugova moguće su veće greške tokom izrade, a kada su greške izvedene u tvari njih je skoro nemoguće ispraviti
- teže je proizvesti nove generacije mikroprocesora
- neke ugrađene naredbe su možda neefikasne zbog dostupnosti novih algoritama ili zbog nemarnosti dizajnera
- mali broj naredbi od cijelog dostupnog skupa naredbi se koristi (80/20 pravilo)

RISC

RISC procesori opisuju se stanjem procesora koje obuhvaća veći skup spremnika opće namjene i nekoliko spremnika posebne namjene. Kod stvarnih procesora postoje i dodatni spremnici koji određuju stanje procesora kao što su npr. spremnik stanja procesora, spremnik prekida, U/I spremnici, itd.
Noviji RISC procesori naprotiv ograničavaju da se operandi moraju nalaziti u spremnicima opće namjene tako da je prije izvođenja aritmetičke ili logičke operacije potrebno dopremiti operande iz memorije u spremnike kao i pohraniti rezultat iz spremnika u memoriju.
Suvremeni procesori, naročito RISC temeljeni su na naredbama s 1 | adrese. Sve aritmetičke i logičke naredbe kao i grananja koriste isključivo spremnike, dok je pristup memoriji ograničen **load** i **store** naredbama

RISC je kratica za Reduced Instruction Set Computer ili tip središnje jedinice (procesora) sa smanjenim skupom naredbi. Filozofija RISC-a svodi se na:

- stvaranje procesora s manjim opsegom naredbi
- povećanjem broja registara dostupnim CPU
- stavljanjem cache memorija na CPU
- korištenje tzv. *pipelining*-a koji omogućuje izvršavanje više naredbi unutar jednog otkucaja unutarnjeg sata CPU-a

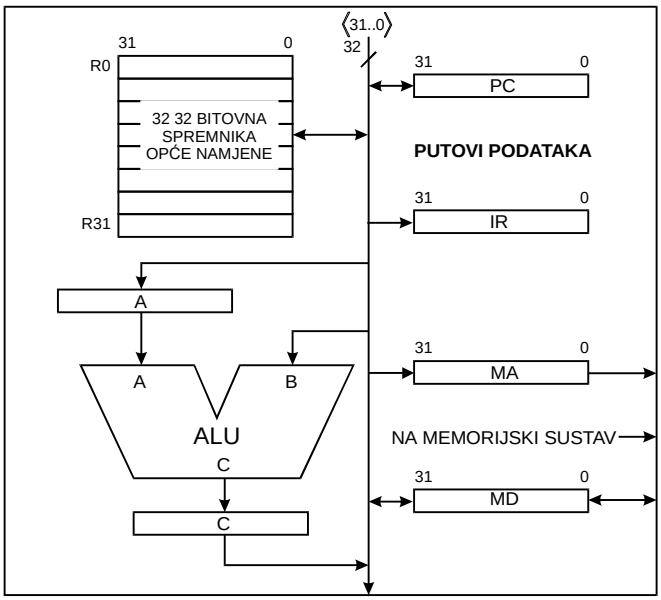
6. Postupak projektiranja RISC procesora

a) Jednako trajanje izvođenja svih naredbi. Ovaj jednostavan koncept praktički je najvažniji temelj u projektiranju RISC-a. Prve definicije RISCa podrazumijevale su da se sve naredbe izvode u jednom ciklusu. Kako vrijeme izvođenja ovisi o propusnosti procesora, a ne o vrmenu izvođenja pojedine naredbe, izvođenje jedne naredbe po ciklusu je opravdan projektantski cilj.
b) Jednaka dužina svih naredbi. Ako se izvodi jedna naredba po ciklusu i ako su naredbe jednostavne tada je logično da su sve naredbe i fiksne dužine, obično jedna riječ. Ta riječ specificira sve o naredbi, operacijski kod, gdje su smješteni operandi, gdje pohraniti rezultat i gdje se nalazi sljedeća naredba
c) Pristup memoriji preko load i store naredbi. Ako je naredba dužine jedne rijeći, te kako dohvat operanada iz memorije zahtijeva dodatno vrijeme, projektanti RISC računala ograničavaju se na izvođenje svih operacija nad operandima pohranjenim u spremnicima procesora. Pristup operandima u memoriji ograničen je na operacije njihova dohvata, load, i pohrane, store. Minimizacija broja pristupa memoriji od strane procesora, reduciranje opterećenja vanjske sabirnice, minimizacija kašnjenja pri dohvatu sljedeće naredbe.
d) Jednostavni adresni modovi. Složeni adresni modovi zahtijevaju više taktova budući da se mora provesti više aritmetičkih operacija. RISC procesori obično su građeni na samo dva adresna moda, indirektno pomoću spremnika i indeksno.
e) Manje jednostavnih operacija. Jednostavne naredbe impliciraju kraće vrijeme izvođenja (manji broj taktova) budući da je za njihovu realizaciju potreban znatno manji broj radnji.
f) Odgođeni pristupi memoriji i grananja. Kako bi se osiguralo da naredba koja slijedi neku drugu naredbu ne bi ovisila o njenom rezultatu uvodi se posebna naredba nop

koja ništa ne radi. Ona samo zauzima mjesto u nizu naredbi i troši vrijeme kako bi se omogućilo prethodnoj narredbi da završi i dostavi razultat koji mogu koristiti sljedeće naredbe.

g) Podhvat naredbe i spekulativno izvođenje. Kada su sve naredbe jednake dužine jednostavnije je analizirati naredbu kako uđe u cjevovod te zaključiti da li se radi o naredbi aritmetičkoj ili logičkoj naredbi, pristupu memoriji ili grananjima. Ukoliko se zaključi da se radi posljednjima može se odmah pristupiti dohvatu operanda ili naredbe koja slijedi ukoliko dođe do grananja. Ovim pristupom praktički ne dolazi do okašnjenja zbog grananja ili pristupa memoriji. Procesor ima više jedinica koje mogu istovremeno obavljati dijelove obrade naredbi, a kako je moguće znati adresu mogućeg grananja unaprijed, procesor započima obradu naredbe koja ej razultat grananja bez obzira da li je zadovoljen uvjet grananja. Rezultat dijela njene obrade će se odbaciti ukoliko se pokaže da nije potrebno izvesti grananja.
h) Značenje programa prevodioca. RISCovi imaju kraće naredbe koje izvode samo jednostavnije operacije. Razbijanje složenih naredbi CISCa na niz jednostavnih naredbi uspostavlja veću ovisnost između naredbi. Kako bi se što bolje iskoristile karakteristike RISCova potrebno je da se izvođenje naredbi izvodi i izvan redosljeda određenog programom kad god je to moguće. Ovaj zahtjev ozbiljno opterećuje programera jer on ili ona ne mora samo paziti na ispravnost programa nego i da kod bude optimiziran za određeni procesor u smislu da u potpunosti iskoristi karakteristike njegove arhitekture i sklopovlja.

7. Na arhitekturi prikazanoj na slici objasnite izvođenje naredbe add R2, R3, R6 te navedite koje upravljačke signale je potrebno generirati.



Apstraktan opis operacije: R2 ← R3 + R6

Konkretan opis operacije: A ← R6 ; sadržaj spremnika R6 se upisuje u privremeni spremnik A

C ← R3 + A ; postavljanje drugog operanda (R3) na sabirnicu te pohrana rezultata zbrajanja u izlazni spremnik C

R2 ← C sadržaj spremnika C se prebacuje u odredišni spremnik R2

Potrebno je generirati slijedeće upravljače signale:

$R6_{out}$, A_{in}, $R3_{out}$, C_{in}, C_{out}, $R2_{in}$

8. Programski odsječak prevedite u simbolički (asemblerski) oblik

int i, j, k;
k=i + j – 20;

```
LD      R1, adresa od i        ;   dovođenje varijable i iz memorije u registar

LD      R2,  adresa od j       ;   dovođenje varijable j iz memorije u registar

ADD     R3, R2, R1             ;   zbrajanje varijabli

ADDI    R3, R3,  -20           ;   pribrajanje konstante

ST      R3, adresa od k        ;   spremanje rezultata na adresu od k
```

9. Programski odsječak prevedite u simbolički (asemblerski) oblik

```
int i, j, k;
if (i > 0)
        k=i + j;
```

```
LD        R1, adresa od  i

LD        R2,  adresa od  j

ADDI      R∅ , R1,   ∅

LDA       R3,  adresa_grananja

BRMI      R3,  R∅

ADD       R4,  R1, R2

ST        R4, adresa od k
```

10. Prve tri simboličke naredbe iz prethodnog zadatka prevedite u strojni oblik.

Prve tri simboličke naredbe glase:

LD R1, adresa od i

LD R2, adresa od j

ADDI R∅, R1, ∅

	OPCODE	
LD	1	00001
ADDI	15	01111

&i =1
&j=2

Binarni oblik:

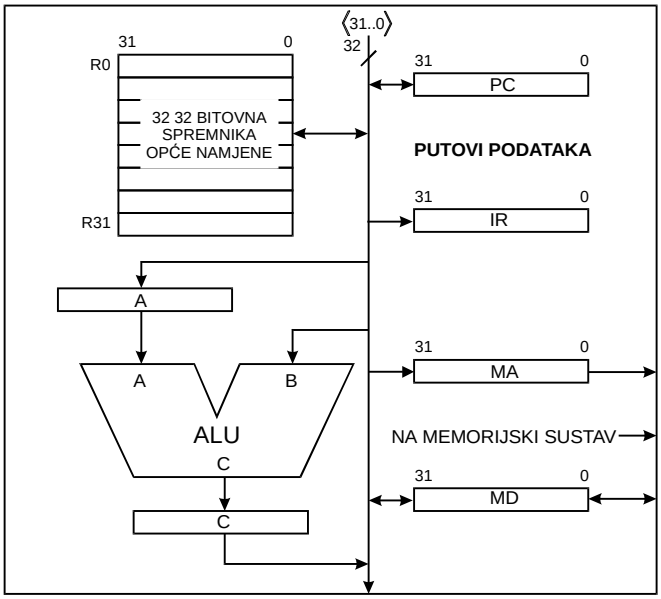
00001	00001	00000	00000000000000001
LD	R1	0	&i =1
00001	00010	00000	00000000000000010
LD	R2	0	&j =2
01111	00000	00001	00000000000000000
ADDI	R0	R1	konst = 0

Heksadecimalni oblik: 0 8 4 0 0 0 0 1 H
0 8 8 0 0 0 0 2 H
7 8 0 2 0 0 0 0 H

11. Objasnite koncept računala temeljenog na akumulatoru.

Procesori temeljeni na akumulatoru imaju ograničen broj akumulatora, obično samo jedan, zajedno sa spremnicima za pohranu adresa. Naziv akumulator potječe od činjenice da ovaj spremnik služi za pohranu jednog operanda aritmetičke ili logičke operacije, te kao spremnik u koji se po izvođenju operacije pohranjuje njen rezultat. Tako on služi za akumulaciju podataka te mu odatle potječe i naziv. U vrijeme prvih procesora kada je memorija bila skupa ovakva računala su bila naročito popularna zato jer je uvijek jedan operand bio u akumulatoru te je u naredbi bilo potrebno specificirati samo memorijsku lokaciju drugog operanda. Rezultat operacije se uvjek pohranjivao u akumulator. Zato se ovakvi procesori nazivaju procesori s jednom adresom. Naredbe ovakvih procesora zato su bili znatno kraće (dva okteta) i program je zauzimao manje memorije. Ovakav pristup ima bitnih ograničenja kod proračuna aritmetičkih i logičkih operacija te zahtjeva učestalo premještanje podataka iz memorije u akumulator i rezultata iz akumulatora u memoriju. Zbog opterećenja sabirnice koja povezuje glavnu memoriju i procesor ovakvo rješenje napušteno je kod modernijih procesora.

12. Na arhitekturi prikazanoj na slici objasnite izvođenje naredbe add R2, R3, R6 te navedite koje upravljačke signale je potrebno generirati.



Apstraktan opis operacije: R2 ← R3 + R6

Konkretan opis operacije:

A ← R6 ; sadržaj spremnika R6 se upisuje u privremeni spremnik A

C ← R3 + A ; postavljanje drugog operanda (R3) na sabirnicu te pohrana rezultata zbrajanja u izlazni spremnik C

R2 ← C sadržaj spremnika C se prebacuje u odredišni spremnik R2

Potrebno je generirati slijedeće upravljače signale:

$R6_{out}$, A_{in}, $R3_{out}$, C_{in}, C_{out}, $R2_{in}$