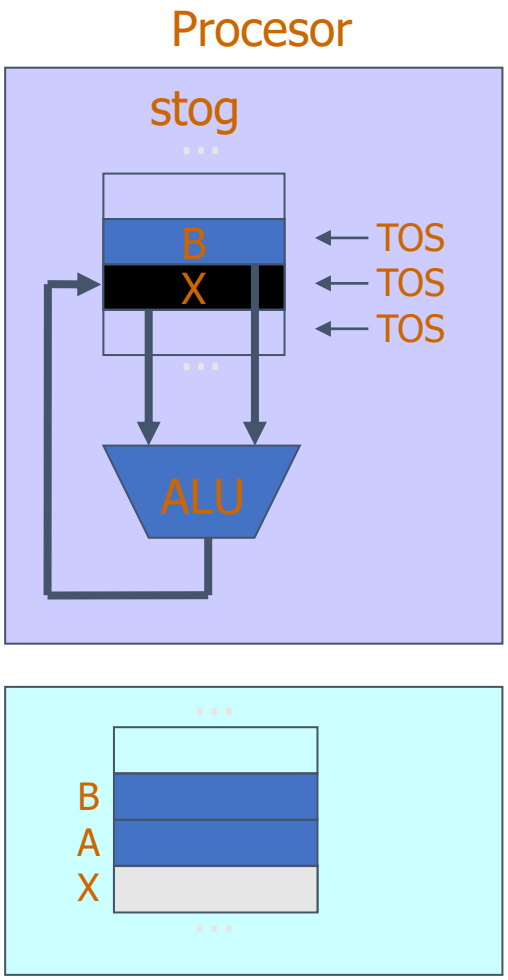
Arhitekture s obzirom na dohvat operanada

- Stogovne arhitekture
- Akumulatorske arhitekture
- Arhitekture registar-memorija
- Arhitekture registar-registar (tzv. load-store)

Stogovna arhitektura

- Upotrebljava se stog za spremanje podataka koji se obrađuju (da bi se riješio ranije spomenuti problem dohvata podataka iz memorije)
- Ovaj stog **nalazi se u procesoru**, kao i pokazivač stoga!!!
- **Operandi su implicitno na vrhu stoga, gdje se smješta i rezultat**
- U naredbama za obradu podataka ne zadaje se eksplicitno gdje se nalaze operandi niti gdje se sprema rezultat
- Naredba za obradu podataka pristupa samo internom stogu

Stogovna arhitektura



- Primjer: računanje funkcije x=a+b

```
push A    ; operand A iz memorije se stavlja na vrh stoga
push B    ; operand B iz memorije se stavlja na vrh stoga
add       ; zbrajaju se operandi sa vrha stoga
          ; i rezultat se stavlja na vrh stoga
pop X     ; rezultat se sa vrha stoga sprema u memoriju
```

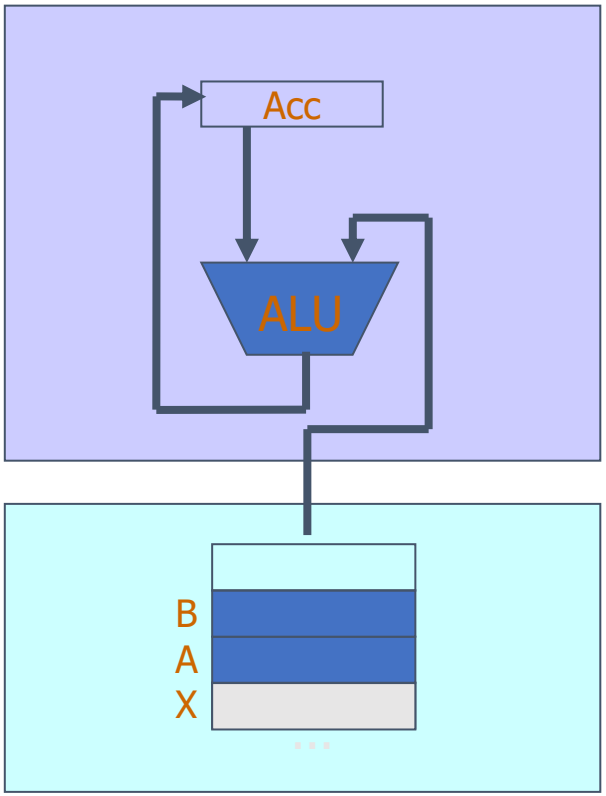
Stogovna arhitektura

- Stogovna arhitektura koristila se kod prvih procesora i danas se u svom osnovnom obliku više ne koristi

- Dobre strane stogovne arhitekture:
  - Naredbe su jednostavne i bez puno opcija što ih čini brzima za izvođenje
  - Izvedba upravljačke jedinice je jednostavna
  - Prevoditelji su jednostavni

- Loše strane stogovne arhitekture:
  - Međurezultati se teško koriste
  - Prevođenje nije efikasno (jednostavna naredba višeg jezika se prevodi u dugačak niz naredaba strojnog jezika)
  - Veliki broj pristupa memoriji !!!

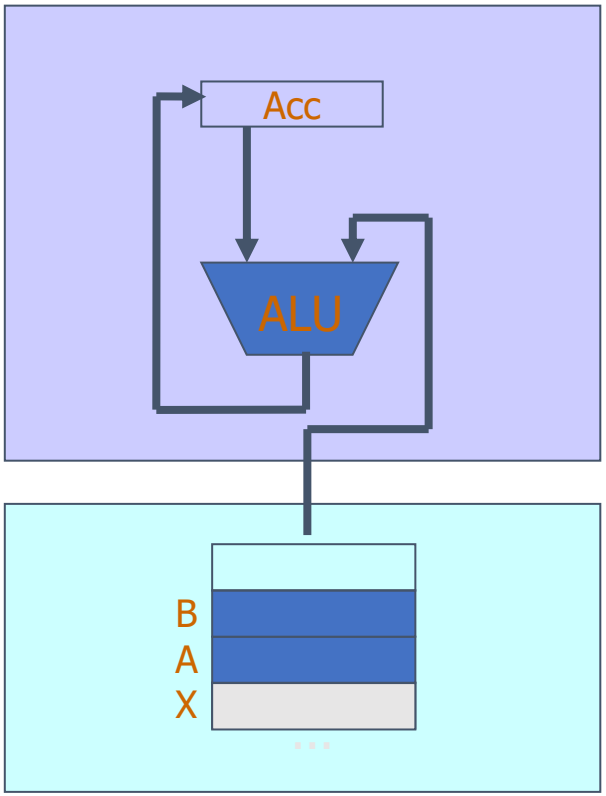
Akumulatorska arhitektura



- **Jedan operand je uvijek u posebnom registru** koji se naziva Akumulator (Acc)
- Ako postoji, **drugi operand se čita iz memorije**
- **Rezultat se sprema u Acc**
- Naredba za obradu podataka pristupa memoriji

Akumulatorska arhitektura

- Primjer: računanje izraza x=a+b




```
load A   ; operand A se iz memorije
         ; stavlja u Acc

add B    ; zbraja se Acc sa operandom
         ; B iz memorije i rezultat
         ; se stavlja u Acc

store X  ; rezultat se iz Acc
         ; sprema u memoriju
```

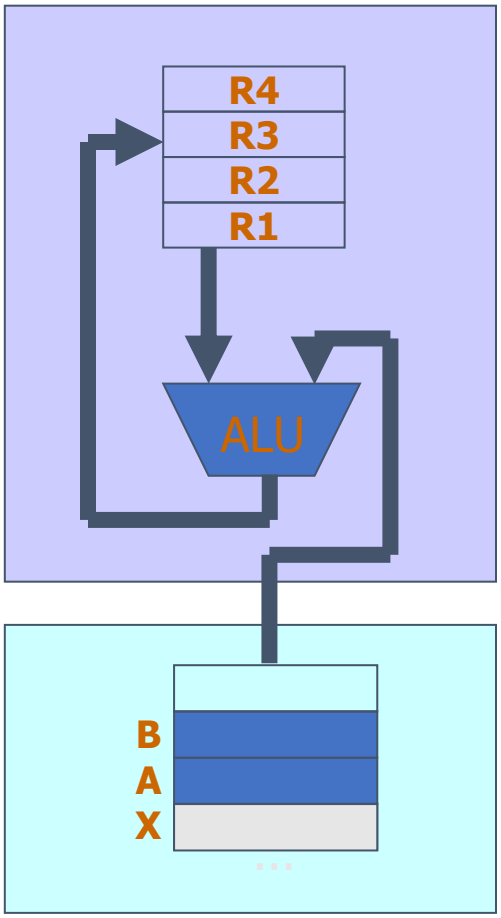
Akumulatorska arhitektura

- Vrlo česta arhitektura prvih procesora, a danas se još može naći kod nekih jednostavnih mikrokontrolera

- Dobre strane ove arhitekture:
  - Jednostavnija za izvedbu od stogovne (Acc umjesto stoga)
  - Naredbe su jednostavne i bez puno opcija
  - Jedan operand je u memoriji (ne mora ga se dohvaćati dodatnom naredbom)
  - Prevoditelji su jednostavni

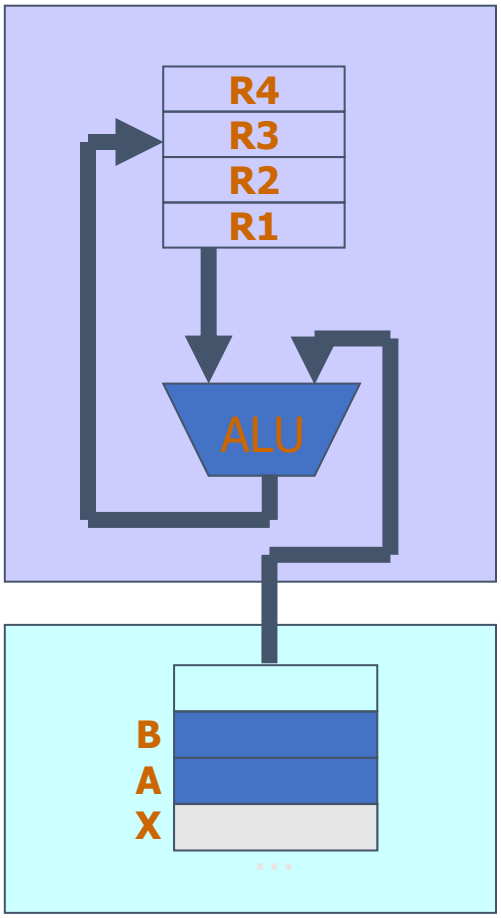
- Loše strane ove arhitekture:
  - Međurezultati (osim zadnjeg) se ne mogu koristiti već sve mora biti pohranjeno u memoriju
  - Jako velik broj pristupa memoriji !!!
  - Prevođenje nije efikasno

Arhitektura registar-memorija



- **Jedan operand se nalazi u skupu registara opće namjene** (registri koji se slobodno koriste i nemaju posebnu funkciju)
  - Kao i kod drugih arhitektura, uvijek može postojati jedan ili više registara specifične namjene, ali se oni ne ubrajaju u općenamjenske registre
- **Drugi operand se čita iz memorije**
- **Rezultat se sprema u neki od registara opće namjene**
- Naredba za obradu podataka pristupa memoriji

Arhitektura registar-memorija



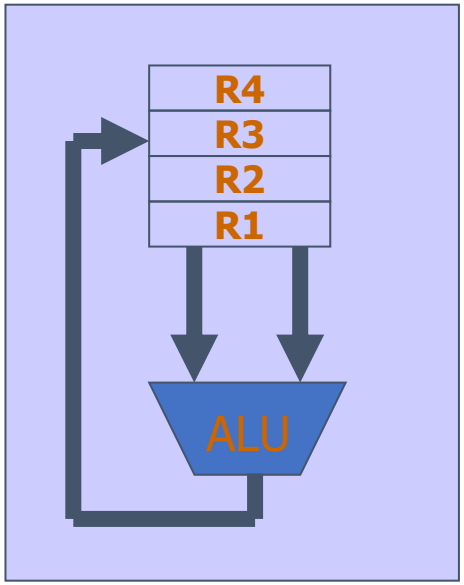
Primjer: računanje izraza x=a+b

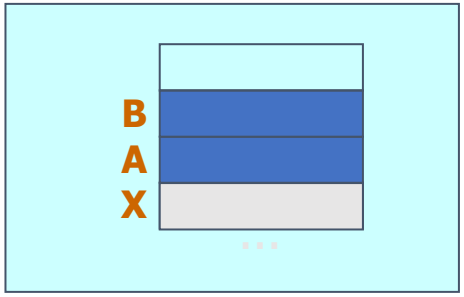
```
load R1,A       ; operand A se iz memorije
                ; stavlja u R1

add R3,R1,B     ; zbraja se R1 sa operandom B iz
                ; memorije i rezultat se stavlja
                ; u R3

store R3,X      ; rezultat se iz R3 sprema u
                ; memoriju
```

Arhitektura registar-memorija

- Karakteristike arhitekture registar-memorija su:
  - Varijable se mogu čuvati u registrima opće namjene (što je više registara to je manje potrebno komunicirati s memorijom, a posljedica je brži pristup podatcima)
  - Prevoditelji efikasnije prevode programe jer su naredbe moćnije i podatci mogu biti u registrima
  - Naredbe veće i sporije
- Prednosti:
  - Jednostavan pristup podatcima u memoriji
- Nedostatci
  - Dodatan pristup memoriji pri izvođenju naredaba, vrijeme izvođenja varira ovisno o naredbi i operandima

Arhitektura registar-registar (load-store)

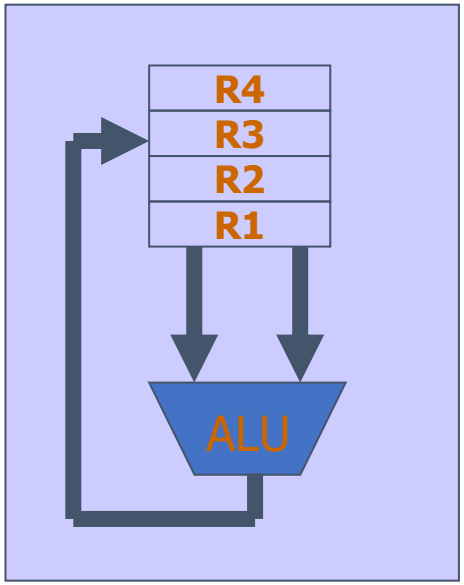


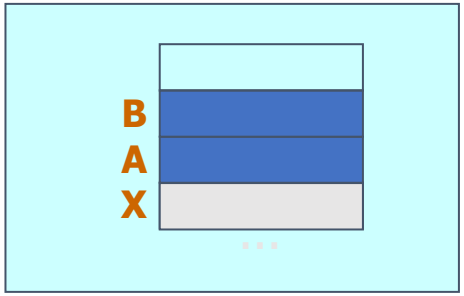


- **Oba operanda su u registrima opće namjene**
- **Rezultat se sprema u neki od registara opće namjene**
- Naredba za obradu podataka pristupa isključivo općim registrima
- Podatci se mogu čitati iz memorije ili pisati u nju isključivo pomoću naredba LOAD i STORE

Arhitektura registar-registar (load-store)

- Primjer: računanje izraza x=a+b






```
load R1,A       ; operand A se iz memorije
                ; stavlja u R1

load R2,B       ; operand B se iz memorije
                ; stavlja u R2

add R3,R1,R2    ; zbraja se R1 sa R2
                ; i rezultat se stavlja u R3

store R3,X      ; rezultat se iz R3 sprema u
                ; memoriju
```

Arhitektura registar-registar (load-store)

- Karakteristike registarsko-registarske arhitekture su (isto kao kod registarsko-memorijske):
  - Varijable se mogu čuvati u registrima opće namjene
  - Prevoditelji efikasnije prevode programe
  - Naredbe veće i sporije

- Prednosti:
  - Brzo izvođenje, jednostavan format naredaba, jednostavno generiranje kôda, jednostavnost protočne strukture, uniformno vrijeme izvođenja
- Nedostatci:
  - Veći broj naredaba u programu zbog zasebnih učitavanja podataka iz memorije

Usporedba prethodnih arhitektura

- Stogovna i akumulatorska arhitektura bile su često korištene u prvim procesorima dok se danas gotovo ne koriste
- Neke ideje revitalizacije stogovne arhitekture postoje kod procesora koji izvode Java bytecode

- većina današnjih procesora ima registarsku arhitekturu (reg-mem ili reg-reg) među kojima su više zastupljene reg-reg (load-store) arhitekture

Primjeri navedenih arhitektura

- Stogovna:
  - WISC CPU/16, MISC M17, picoJava
- Akumulatorska
  - EDSAC
- Registarsko-memorijska
  - x86 (imaju karakeristike reg-mem i reg-reg)
- Registarsko-registarska (load-store)
  - ARM, RISC-V

Arhitekture s obzirom na skup naredaba

- S obzirom na skup naredaba procesora razvijene su dvije arhitekture:
  - CISC (Complex Instruction Set Computer)
  - RISC (Reduced Instruction Set Computer)

- U današnje vrijeme komercijalni procesori nemaju više čistu arhitekturu CISC ili RISC:
  - obično u određenom procesoru prevladava jedna arhitektura, ali...
  - često se uključuju pojedina svojstva druge arhitekture

- Pogledajmo karakteristike CISC i RISC arhitektura...

CISC

- U samom početku procesori su bili vrlo jednostavni, ali su tehnološki vrlo brzo napredovali...
- Uskoro je glavni trend u oblikovanju arhitekture procesora bilo uvođenje procesorskih naredaba bliskih naredbama viših programskih jezika, npr.:

  - umanji registar za 1 i skoči na početak petlje ako je registar veći od nule

  - pomnoži matrice u memoriji

  - pronađi određeni podatak u bloku memorije

- Prednosti takvih procesorskih naredaba bile su:
  - Jednostavnije prevođenje programa iz viših programskih jezika
  - Ušteda memorije zbog manjeg broja naredaba (važno u to doba!!!)
  - Ubrzanje rada zbog manjeg broja dohvata naredaba iz memorije

- Procesori s takvim skupom naredaba nazivaju se CISC procesori

CISC

- Karakteristike CISC procesora
  - Velik broj naredaba i njihovih inačica
  - Velik broj načina adresiranja (više o tome kasnije)
  - Većinom su se naredbe unutar procesora izvodile korištenjem načela mikroprograma, tj. kompleksne naredbe izvodile su se u nizu ciklusa tijekom kojih je procesor izvodio niz jednostavnijih operacija (više o tome ćemo govoriti kasnije)
  - Registri imaju posebne namjene (brojači za petlje, za adresiranje, za podatke itd.)
  - Problem kompleksnih naredaba rješava se "unutar procesora" (možemo reći “sklopovski”)
  - Skupo projektiranje i visoka cijena

- Primjeri CISC procesora: Intel 80x86, Motorola 68000

RISC

- 70tih i početkom 80tih dominaciju na tržištu imali su 8-bitni CISC procesori

- Međutim, kompleksne naredbe zahtjevaju kompleksnu logiku za dekodiranje (sporo dekodiranje i izvođenje) i izuzetno skup i dugotrajan postupak projektiranja takvih procesora

- Početkom 80-tih, u okviru tri gotovo usporedna istraživačka projekta (IBM 801, Berkeley RISC i Stanford MIPS) razvijena je potpuno nova arhitektura procesora zasnovana na jednostavnim instrukcijama koje se mogu izvoditi velikom brzinom

- Procesori s takvim skupom naredaba nazivaju se RISC (Reduced Instruction Set Computer)

- RISC procesor razvijen na sveučilištu Berkeley imao je izuzetne performanse u usporedbi s komercijalnim CISC-procesorima uz znatno jednostavniju i jeftiniju sklopovsku izvedbu.

RISC

- Primjeri jednostavnih "RISC-naredaba"
  - učitaj operand iz memorije u registar
  - zbroji dva podatka iz registra
  - spremi sadržaj registra u memoriju
  - umanji sadržaj registra za 1
  - skoči na neku naredbu (npr.) početak petlje ako je zastavica ZERO=1

- Umjesto jedne kompleksne "CISC-naredbe" može se napisati niz jednostavnijih "RISC-naredaba"
  - jednostavnije naredbe se puno brže izvode pa je rezultat ubrzanje izvođenja programa bez obzira na veći broj naredaba

RISC

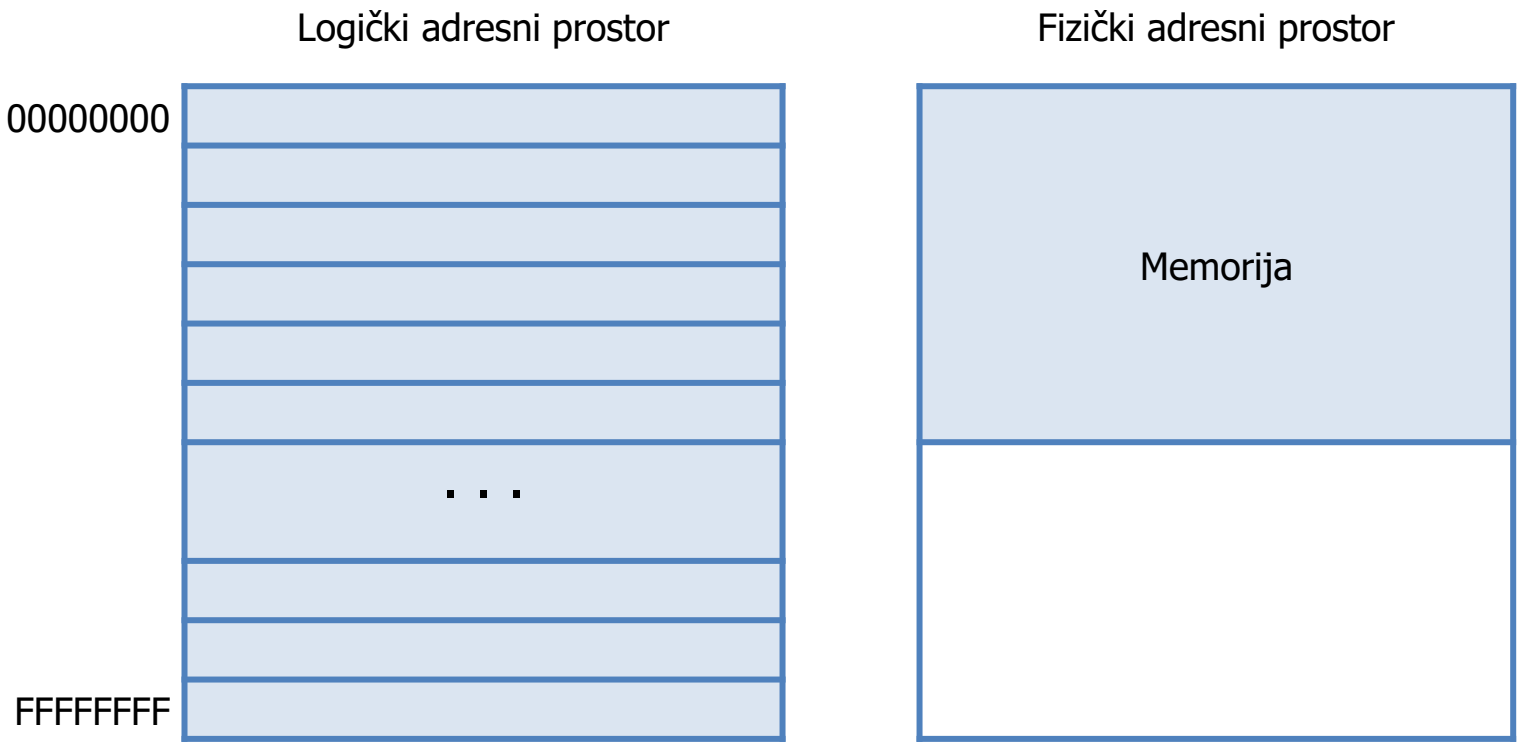
- Karakteristike RISC procesora
  - Relativno malen skup jednostavnih naredaba, manji broj inačica svake naredbe
  - Mali broj načina adresiranja
  - Pojedina naredba brzo se izvodi
  - Velik broj ravnopravnih registara opće namjene unutar procesora
  - Problem kompleksnih naredaba rješava se izvan procesora (možemo reći “programski”)
  - Korištenje protočne strukture za ubrzanje rada (više o tome kasnije)
  - Relativno jeftino projektiranje i niska cijena
- Primjeri RISC procesora: MIPS, ARM, RISC-V

Broj i širina registara

- Širinu registara (u bitovima) određuje statistika najčešće korištenih podataka u programima koje želimo izvoditi na procesoru. Na primjer:
  - Byte 8b
  - Short 16b
  - Int 32b
  - Long 64b
  - Float 32b
  - Double 64b

- Daleko najčešće korišteni tipovi podataka u općim primjenama su 32-bitni int i float
- Zato je vrlo često za jednostavnije procesore izabrana širina od **32b kao optimalna širina registara.**
- **Procesori novih generacija koji su nisu predviđeni za jednostavnije primjene za širinu danas koriste 64b**
- Ako se registri opće namjene izabere širina od 32b ili 64b, tada se implicitno definira i preferirana širina interne sabirnice podataka kao i širina ulaza i izlaza aritmetičko-logiče jedinice

Adresni prostor

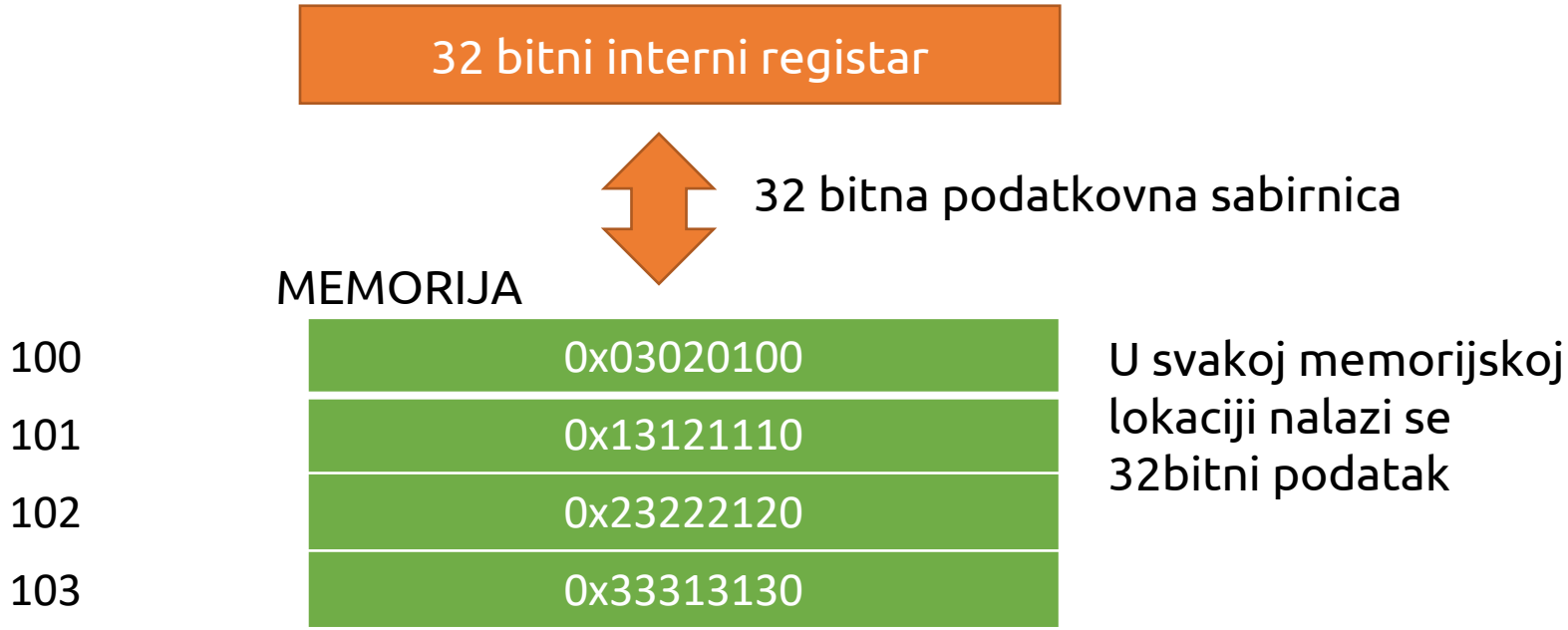


Redoslijed zapisa podataka u mem.

Zapis podatka $1234_{(16)}$ u memoriji

Adresiranje podataka u memoriji

- Način organizacije adresiranja podataka može se razlikovati (uzmimo primjer 32b procesora)



Adresiranje podataka u memoriji

- Češće je memorija organizirana tako da se može adresirati pojedinačan bajt, a onda procesor može čitati više bajtova istovremeno

32 bitna podatkovna sabirnica

MEMORIJA

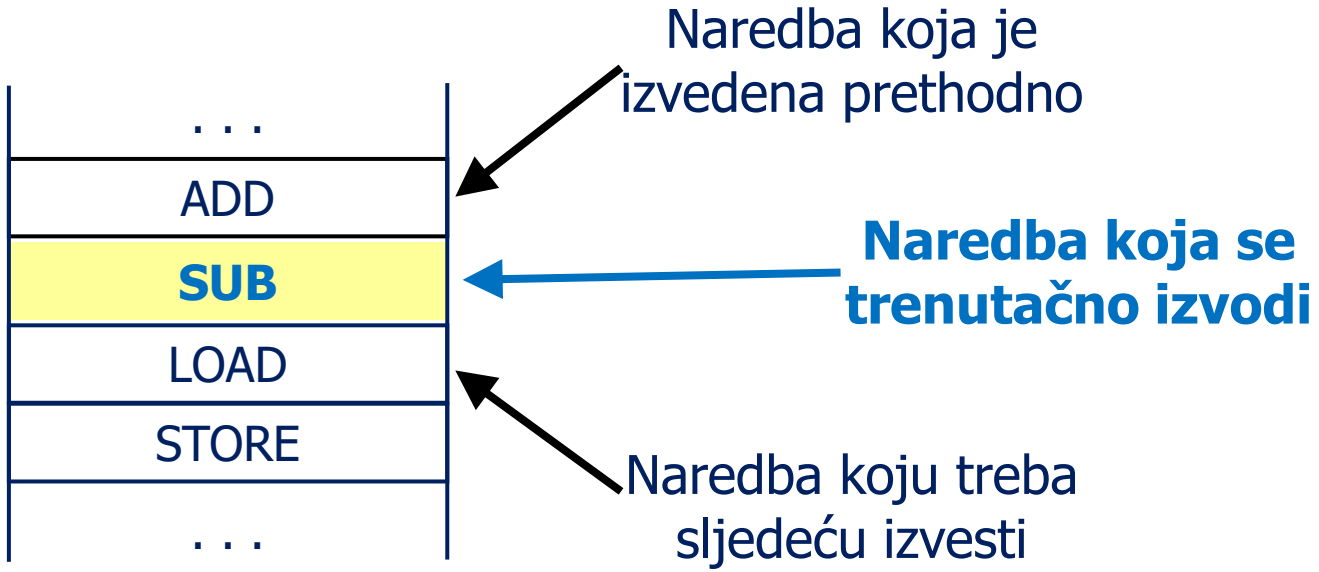
100	0x03	0x02	0x01	0x00
104	0x13	0x12	0x11	0x10
108	0x23	0x22	0x21	0x20
10B	0x33	0x32	0x31	0x30

U svakoj memorijskoj lokaciji nalazi se 8 bitni podatak

Programsko brojilo

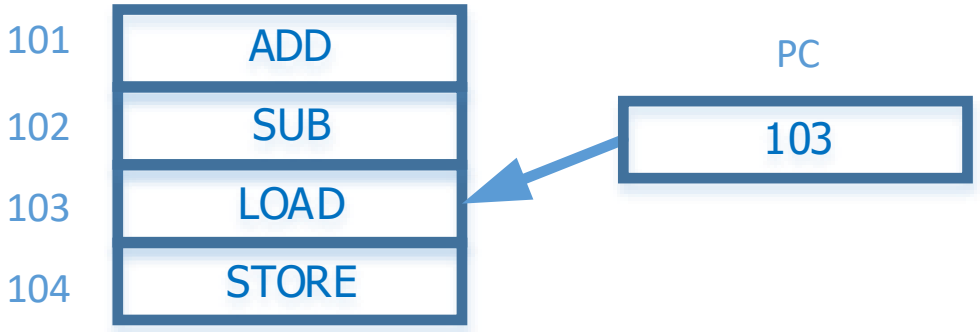
- Spomenuli smo da procesor dohvaća naredbe iz memorije i izvodi ih

- Naredbe su u memoriji smještene slijedno jedna iza druge i tim redoslijedom se dohvaćaju i izvode (izuzetak su naredbe skoka)



Programsko brojilo

- To znači da procesor u svakom trenutku mora "znati" adresu naredbe koju treba dohvatiti i izvesti
  - Kako procesor to "zna"?
  - Jednostavno: procesor ima jedan registar koji služi samo toj svrsi: PC (program counter) ili programsko brojilo (iako se tu zapravo ništa ne broji)



* Za sada zanemarimo širinu naredaba i točne adrese

Programsko brojilo

- Registar PC bit će širok onoliko koliko je širina adresne sabirnice jer je logično da se registrom PC može adresirati cijeli memorijski prostor

- Registar PC se automatski uvećava (nakon dohvata svake naredbe) tako da pokazuje na sljedeću naredbu u memoriji

Strojni kod naredbe

- Pomoću tog kôda procesor razlikuje naredbe
- RISC – strojni kôd uobičajeno je širine procesorske riječi

Polje operacijskog koda	Polje prvog operanda	Polje drugog operanda	Polje trećeg operanda

Polje operacijskog koda	Polje adrese (1.dio)	Polje prvog operanda	Polje adrese (2.dio)

Širina strojnog koda naredbe - CISC

ADD R0, R1, R2

operacijski kôd {ADD}	1. operand {R0}	2. operand {R1}	3. operand {R2}

JP_uvjet 200

operacijski kôd {naredba skoka}	Polje uvjeta {uvjet}	...	Proširenje op.koda {JP}
Adresa {200}			

ADD R0, (200), (R2+6)

operacijski kôd {ALU naredba}	1. operand {R0}	Način adresiranja {apsolutno i reg.ind. s pomakom}
Adresa {200}		
Proširenje op.koda {ADD}	3. operand {R2}	Adresni pomak {6}

Strojni kôd

Treba strogo razlikovati:

- **naredbu zapisanu tekstom** kao npr. "ADD R1,R2,R3" što je samo način kako programer piše naredbe u nekom programu za upis teksta prilikom programiranja
- **strojni kôd naredbe** što je zapis naredbe u obliku niza nula i jedinica u memoriji računala

Odabir skupa naredaba

- Jedna od najvažnijih odluka u projektiranju procesora je odabir skupa naredaba (instruction set)

- Postoji više vrsta naredaba, ovisno o procesoru, na primjer:
  - Aritmetičko-logičke naredbe obavljaju AL operacije
  - Registarske naredbe premještaju podatke između registara
  - Memorijske naredbe čitaju i spremaju podatke u/iz memorije
  - Naredbe za premještanje podataka mogu premještati podatak između registara, memorijskih lokacija i/ili puniti brojeve u registre i memorijske lokacije
  - Upravljačke naredbe omogućuju programske skokove

>>>>

Odabir skupa naredaba

<<<< (nastavak)

- Ulazno-izlazne naredbe služe za rad s ulazno-izlaznim jedinicama
- Naredbe za rad s bitovima omogućuju ispitivanje i mijenjanje pojedinih bitova u podatku
- Naredbe za rad s blokovima podataka omogućuju pretraživanje, čitanje i pisanje većeg broja podataka (tj. bloka) koji se nalazi u memoriji
- Specijalne naredbe s posebnom namjenom (npr. odabir načina prekidnog rada, dozvoljavanje ili zabranjivanje prekida, programsko izazivanje iznimaka, naredbe za atomarno ispitivanje i postavljanje memorijskih lokacija, rad s koprocesorima itd.)
- itd. ...

Ukratko: postoji puno vrsta naredaba