



Arhitektura procesora FRISC

PROJEKT !!!!

- Umjesto objašnjavanja gradiva na "klasičan način", u ovom predmetu ćemo pokušati kroz zamišljeni projektni zadatak postupno "uvesti" osnovne pojmove iz arhitekture procesora, asemblerskog programiranja i povezivanja procesora s ulazno-izlaznim jedinicama

Projekt



Projekt

- Zadano je da upravljačka ploča mora minimalno imati:
 - pokazivač brzine
 - prikaz točnog vremena
 - prikaz trenutne vanjske temperature
 - prikaz uključenih uređaja (svjetla, pokazivači smjera,...)
 - prikaz upozorenja (otvorena vrata, snježni uvjeti,...)
 - prikaz GPS navigacije

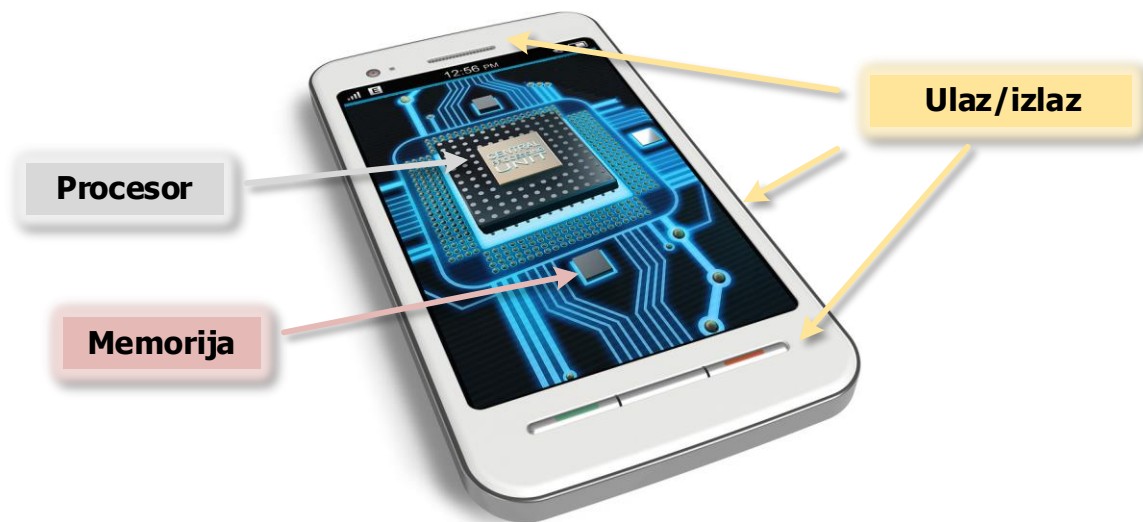
Prvi koraci u projektu

- Na temelju zahtjeva potrebno je donijeti niz odluka:
 - Kakvo računalo odabrati ?
 - Kakav procesor odabrati ?
 - Kako povezati procesor sa sustavom kojim upravljamo ?
 - Kako napraviti program koji će upravljati sustavom ?
 - itd.
- Da bi donijeli ove odluke moramo nešto znati o vrstama računala, procesora, povezivanju sa sustavom, programiranju itd.

Odluka: Vrsta računala

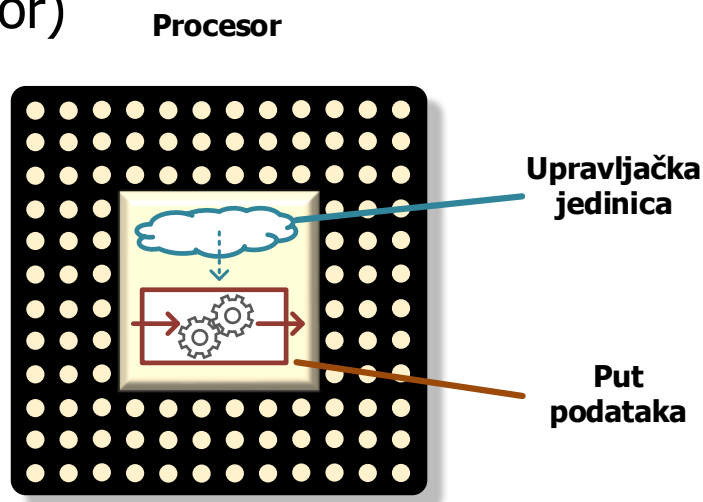
- Izbor vrste računala je očit:
 - za naš projekt odabiremo **UGRADBENO RAČUNALO**
- Sljedeći zadatak:
 - Projektirati ugradbeni računalni sustav
- Kako bi mogli pristupiti projektiranju našeg računalnog sustava moramo znati koji su osnovni dijelovi svakog računala

Sastavni dijelovi svakog računala



Sastavni dijelovi svakog računala

- Najvažniji dio računala je procesor
 - Procesor je aktivni dio računala koji upravlja radom računala i obavlja različite operacije nad podatcima u sustavu
 - Procesor se sastoji od puta podataka (datapath) i upravljačke jedinice
 - Procesor se naziva i CPU (Central Processing Unit) ili GPP (General Purpose Processor)



Osnovni dijelovi računala

- Put podataka
 - Dio procesora koji:
 - obavlja određene operacije (npr. aritmetičke) nad podatcima
 - privremeno pamti podatke i međurezultate
 - prenosi podatke između dijelova za pamćenje i obradu
- Upravljačka jedinica
 - Dio procesora koji upravlja radom puta podataka, memorije i ulazno/izlaznih sklopova na temelju naredaba programa koji se izvodi

Osnovni dijelovi računala

- Memorija
 - Dio računalnog sustava u kojem se spremaju programi koji se izvode na procesoru te podatci potrebni za izvođenje tih programa
- Ulaz i Izlaz
 - Dijelovi računalnog sustava namijenjeni povezivanju s vanjskim svijetom
 - **Ulaz:** dio namijenjen primanju podataka iz vanjskog svijeta u računalu
 - **Izlaz:** dio namijenjen slanju podataka iz računala prema vanjskom svijetu

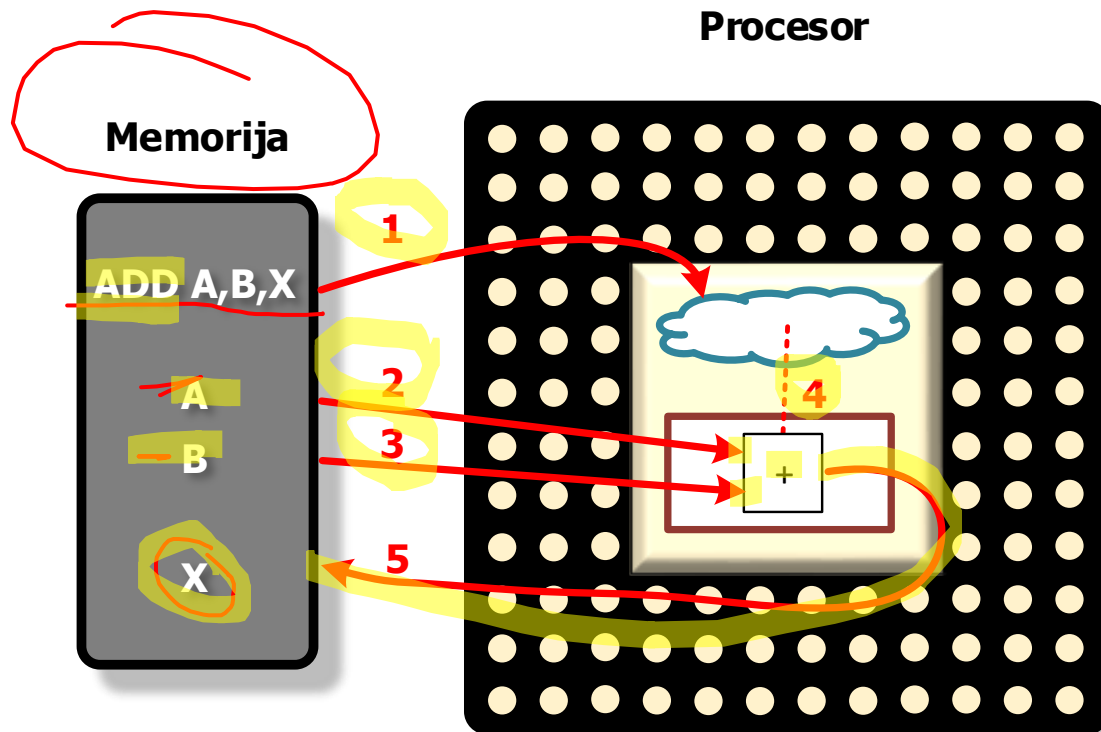
Osnovni način rada procesora

- **Osnovna funkcija procesora je izvođenje programa**
- Program se nalazi u memoriji i sastoji se od niza naredaba
 - svaka vrsta naredbi ima posebni oblik koji je razlikuje od drugih vrsta
- Procesor mora dohvaćati naredbe iz memorije, a dohvaćenu naredbu mora prepoznati da bi je mogao izvesti
- Dakle, rad procesora se odvija u tri osnovna koraka koji se stalno ponavljaju: >>>>
 - dohvat naredbe (fetch)
 - dekodiranje ili prepoznavanje naredbe (decode)
 - izvođenje naredbe (execute)

Osnovni način rada procesora

- **Dohvat** naredbe uvijek se sastoji od operacije čitanja iz memorije
 - može biti više operacija čitanja, ako se naredba sastoji od više memorijskih riječi
- **Dekodiranje** naredbe odvija se interno unutar procesora
- **Izvođenje** se odvija na različite načine, ovisno o vrsti naredbe
 - može se odvijati interno, ako procesor u sebi ima sve podatke za izvođenje
 - može uključivati operacije čitanja iz memorije, ako se podatci za izvođenje naredbe nalaze u memoriji
 - može uključivati pisanja u memoriju, ako rezultate izvođenja naredbe treba spremiti u memoriju

Osnovni način rada procesora



Razine apstrakcije

- Koliko detaljno trebamo razmatrati naše računalo? Do koje dubine ići?
- Pri razmatranju kompleksnih sustava kao što je procesor vrlo često se koriste različite razine apstrakcije
- Razine apstrakcije omogućuju pojedinim sudionicima u procesu projektiranja ili korištenja da se mogu koncentrirati na njihov dio zadatka bez potrebe da brinu o detaljima koji im nisu potrebni

Razine apstrakcije

Primjeri nekih razina apstrakcije kod procesora su:

- **Sistemska razina**
 - Visoka razina apstrakcije koju koriste programeri aplikacija koji ne moraju znati mnogo o načinu kako procesor izvodi program već se koncentriraju na funkcionalnost algoritma i cjelokupne programske podrške.
- **Arhitektura skupa naredaba** (Instruction set architecture level)
 - Najčešće spominjana razina apstrakcije između razine programa i sklopovlja.
 - Uključuje sve podatke o procesoru (registri, pristup memoriji, naredbe, pristup vanjskim sklopovima i ostalo) koji su potrebni da bi se napisali programi u strojnom jeziku koji će se ispravno izvoditi.
 - Sa strane sklopovlja ova razina opisuje funkcionalnost koju sklopovlje treba omogućiti.

Razine apstrakcije

- **Mikroarhitektura**
 - Detaljan sklopovski opis arhitekture procesora. Opisuje načine povezivanja pojedinih dijelova procesora te signale potrebne za njihovo upravljanje.
- **Razina logičkih vrata (gate level)**
 - Potpun sklopovski opis procesora koji, između ostalog, često uključuje i podatke o svim vremenskim kašnjenjima signala unutar sklopa.
- **Razina rasporeda (layout level)**
 - Fizički opis svih sklopova procesora koji koristi definiranu tehnologiju izvedbe. Prikazuje sve detalje potrebne za preslikavanje ove razine u fizičko sklopovlje u postupku proizvodnje.

Razine apstrakcije

- Razine apstrakcije koje ćemo koristiti na ovom predmetu su:
 - **Arhitektura skupa naredaba** (Instruction set architecture - ISA)
 - **Mikroarhitektura**

Arhitektura računala s obzirom na memorijski pristup

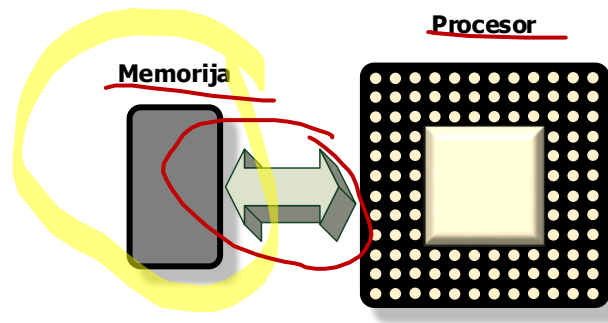
- Izvođenje zadatka na procesoru zahtijeva čitanje naredaba koje je zadao programer te čitanje i pisanje podataka koji se obrađuju (kao što smo vidjeli u jednostavnom primjeru ranije)
- Naredbe i podatci nalaze se u **memoriji**
- Da bi se obavila jednostavna operacija zbrajanja, kao u našem prethodnom primjeru, procesor mora više puta pristupiti memoriji:
 - dohvat naredbe ADD iz memorije
 - dohvat prvog operanda A
 - dohvat drugog operanda B
 - spremanje rezultata X

Arhitektura računala s obzirom na memorijski pristup

- Pri definiranju našeg procesora stoje nam na raspolaganju dvije arhitekture s obzirom na način dovodenja naredaba i podataka iz memorije u procesor:
 - von Neumannova arhitektura
 - Harvardska arhitektura

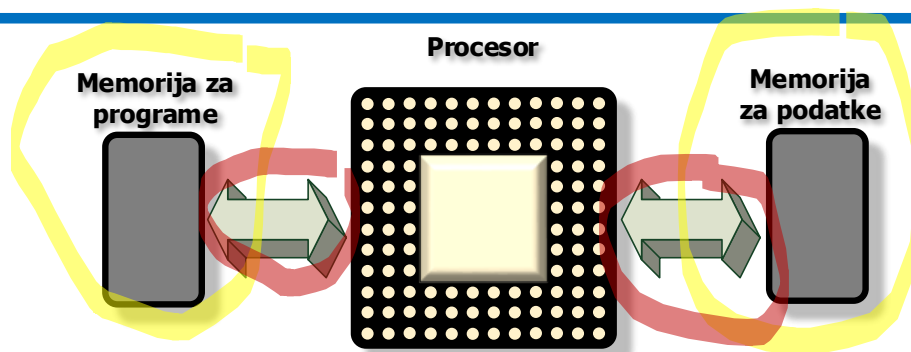
>>>>

Von Neumannova arhitektura



- Značajka von Neumannove arhitekture je u tome da su **program i podatci** smješteni u **jedinstvenu memoriju** koja ima samo jednu vezu prema procesoru
- Jednostavna (i jeftina) arhitektura
- Nedostatak: Procesor ne može dohvatiti naredbu i podatke u istom trenutku => **"Von Neumannovo usko grlo"**
- Usko grlo predstavlja značajno ograničenje za brzinu obrade podataka kod računala koja jednostavnim operacijama obrađuju puno podataka, a trebaju biti efikasna

Harvardska arhitektura



- Jednostavno rješenje von Neumannovog uskog grla je **razdvajanje memorije za pohranu programa od memorije za podatke**
- Procesor ima neovisne veze prema tim memorijama
- Ova arhitektura dozvoljava istovremeno dohvaćanje naredbe i jednog operanda čime se efikasnost znatno poboljšava
- Skuplja od von Neumannove arhitekture te se koristi samo kad je potrebno povećanje performansi

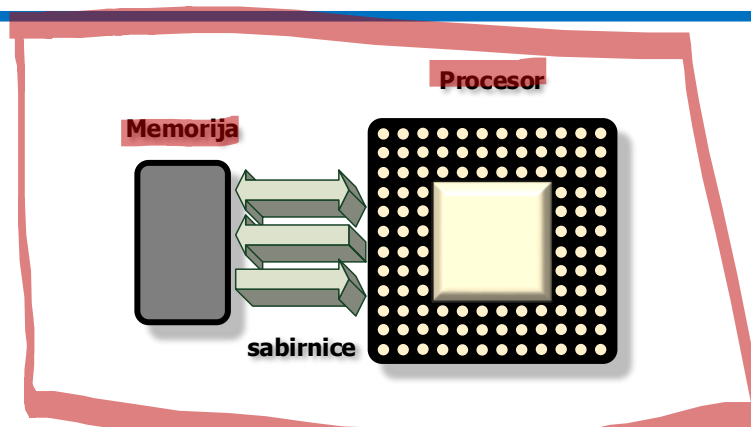
Odluka: Arhitektura mem. pristupa

- S obzirom da želimo projektirati jednostavan sustav i da nam performanse sustava nisu u popisu primarnih zahtjeva, odlučujemo se za jednostavniju i jeftiniju arhitekturu memorijskog pristupa:

Von Neumannova arhitektura

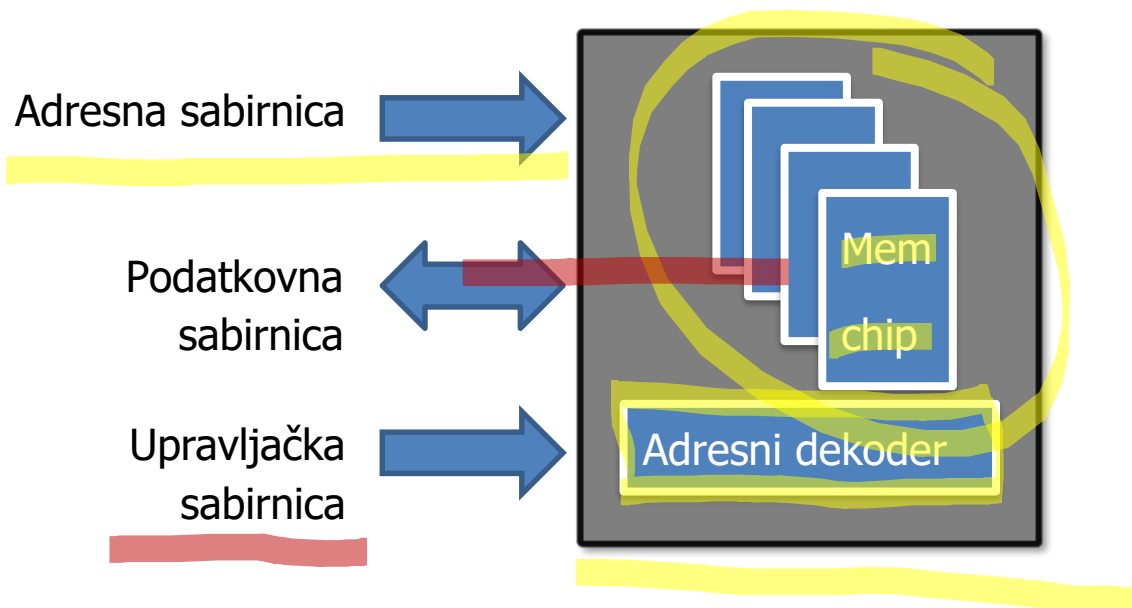


Memorijski podsustav (osnovno)



- Memorijski podsustav načelno se spaja na procesor pomoću **sabirnica**
- Sabirnice su spojni putovi (vodovi) koji povezuju dijelove računala
- **Adresna sabirnica** – određuje mem. lokaciju kojoj se pristupa
- **Podatkovna sabirnica** – služi za prijenos podataka između procesora i memorije
- **Upravljačka sabirnica** – upravlja prijenosom podataka

Memorijski podsustav



- Organizacija memorijske riječi razlikuje se od procesora do procesora: jedna adresa može odgovarati memorijskoj lokaciji širine 8, 16, 32 ili više bita.

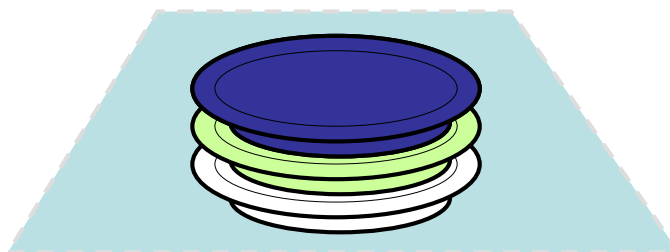
Povezivanje memorije i procesora

- Naš primjer povezivanja MEM-CPU:
 - Von Neumannova arhitektura: jedna (zajednička) memorija za naredbe i podatke
 - Veza prema CPU preko tri sabirnice: adresna, podatkovna i upravljačka
 - Adresna sabirnica: služi za izbor memorijske lokacije
 - Podatkovna sabirnica: prijenos podataka prema/iz memorije
 - Upravljačka sabirnica:
 - određivanje smjera toka podataka, tj. operacije čitanja ili pisanja - RD i WR (read, write)
 - Određivanje širine podatka – SIZE
- Koristeći gore navedeno možemo reći da smo definirali najjednostavniji sustav za pristup memoriji

Mala digresija: Stog

- Struktura podataka koja radi po načelu LIFO (engl. last in first out): zadnji spremljeni (stavljeni) podatak je prvi koji se čita (uzima)
- **Oprez: stog na razini arhitekture računala različit je od stoga na razini višeg programskog jezika (povezana lista i alokacija memorije) !!!**
- Ova struktura se može slikovito zamisliti kao niz tanjura složenih jedan na drugi: 😊

STAVI



UZMI

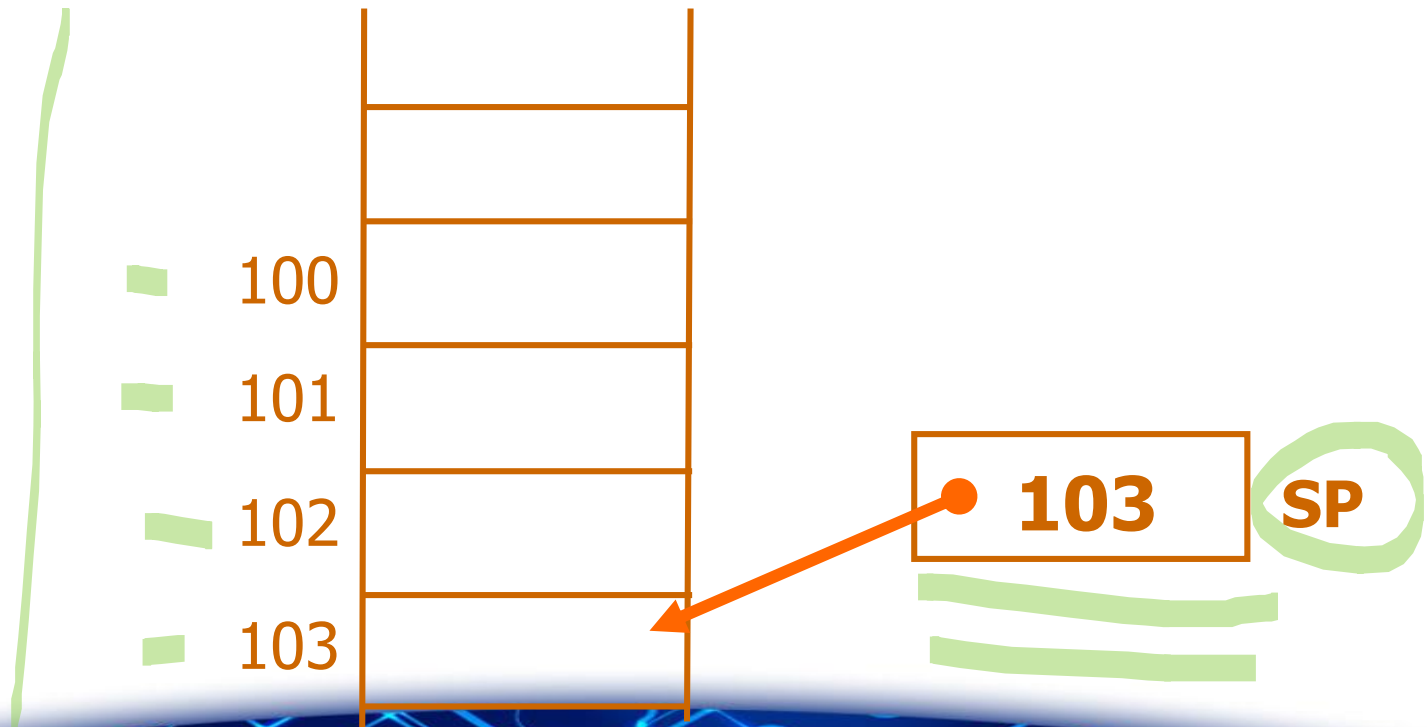
- S tanjurima je lako: uvijek znamo gdje stavljamo tanjur ili od kuda uzimamo tanjur
- Kako znati u koju memorijsku lokaciju treba stavljati ili iz koje uzimati podatak?
- Svaka lokacija se može adresirati, a adresa (položaj) vrha stoga (ToS=Top of Stack) pamti se u pokazivaču stoga (SP=Stack Pointer)
- SP nije ništa drugo, nego obični registar u kojem se pamti adresa vrha stoga

- Podatci se stavljaju na stog i uzimaju sa stoga pomoću posebnih naredaba:
 - PUSH (stavi podatak na vrh stoga) i
 - POP (uzmi podatak s vrha stoga)
- Ove naredbe koriste pokazivač stoga (SP) da bi znale odakle uzimaju (čitaju) ili gdje stavljaju (pišu) podatak
- Kod rada sa stogom uvijek moramo paziti da:
 - ne pokušamo staviti više podataka nego što ima mjesta na stogu (da se stog ne "prepuni")
 - ne pokušamo uzimati podatke s praznog stoga (jer bi čitali podatke iz dijela memorije koja nije dio stoga)
 - ukratko: koliko se stavi na stog, toliko se treba uzeti

Stog

- Kako izgleda stog u memoriji računala?
- Određene memorijske lokacije koriste se kao područje za stog
- Registar SP se obično nalazi u procesoru

>>>>



Stog

- Kako izgleda stog u memoriji računala?
- Određene memorijske lokacije koriste se kao područje za stog
- Registar SP se obično nalazi u procesoru

STAVI

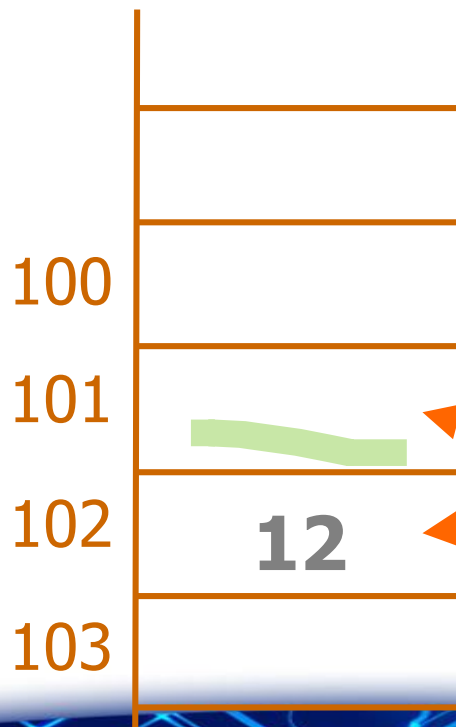


Stog

- Kako izgleda stog u memoriji računala?
- Određene memorijske lokacije koriste se kao područje za stog
- Registar SP se obično nalazi u procesoru

STAVI

Prvo se SP
umanjuje za 1



uočite kako se mijenja
SP za vrijeme rada sa
stogom

102 SP

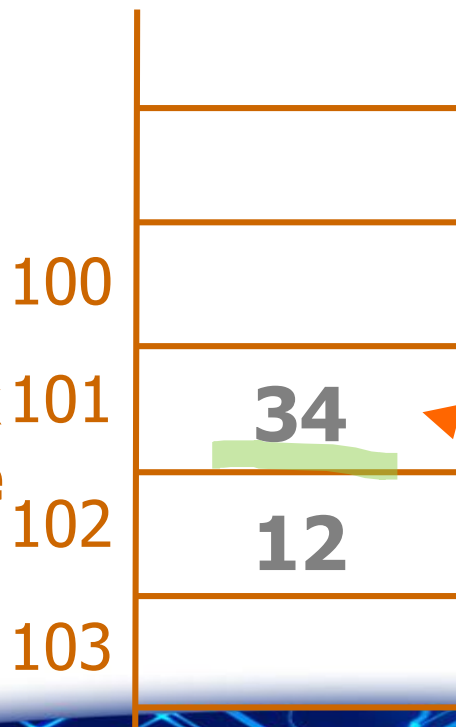
Stog

- Kako izgleda stog u memoriji računala?
- Određene memorijske lokacije koriste se kao područje za stog
- Registar SP se obično nalazi u procesoru

STAVI

**uočite kako se mijenja
SP za vrijeme rada sa
stogom**

**Zatim se podatak
stavlja tamo gdje
pokazuje SP**

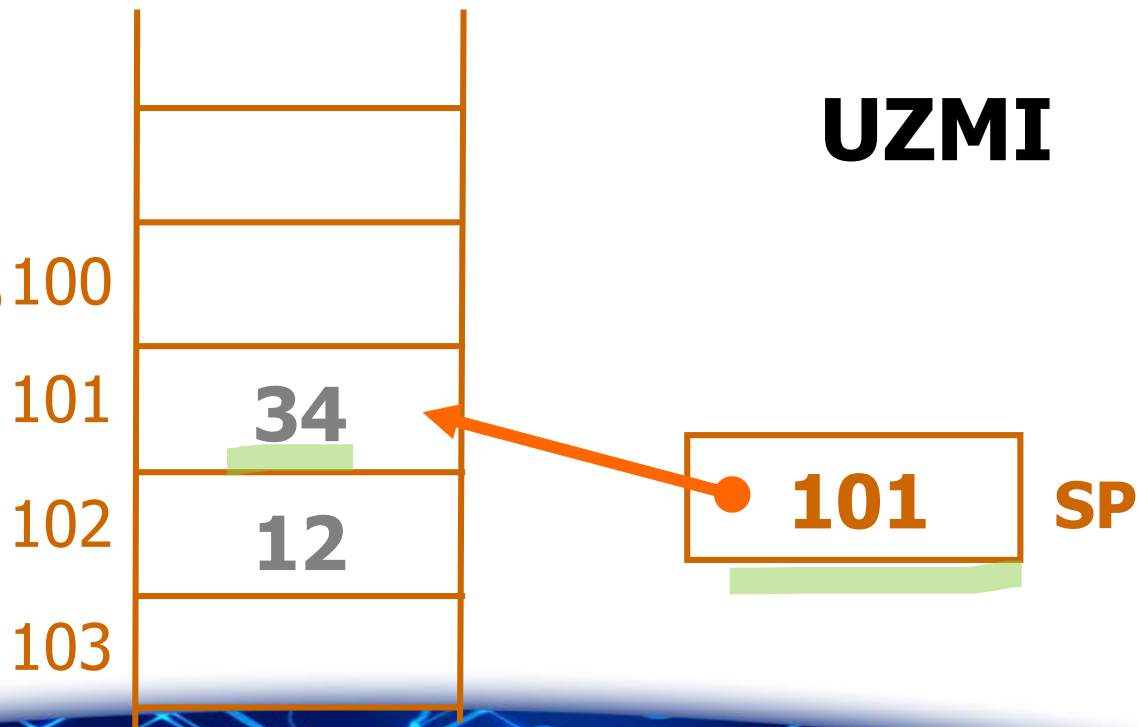


101 SP

Stog

- Kako izgleda stog u memoriji računala?
- Određene memorijske lokacije koriste se kao područje za stog
- Registar SP se obično nalazi u procesoru

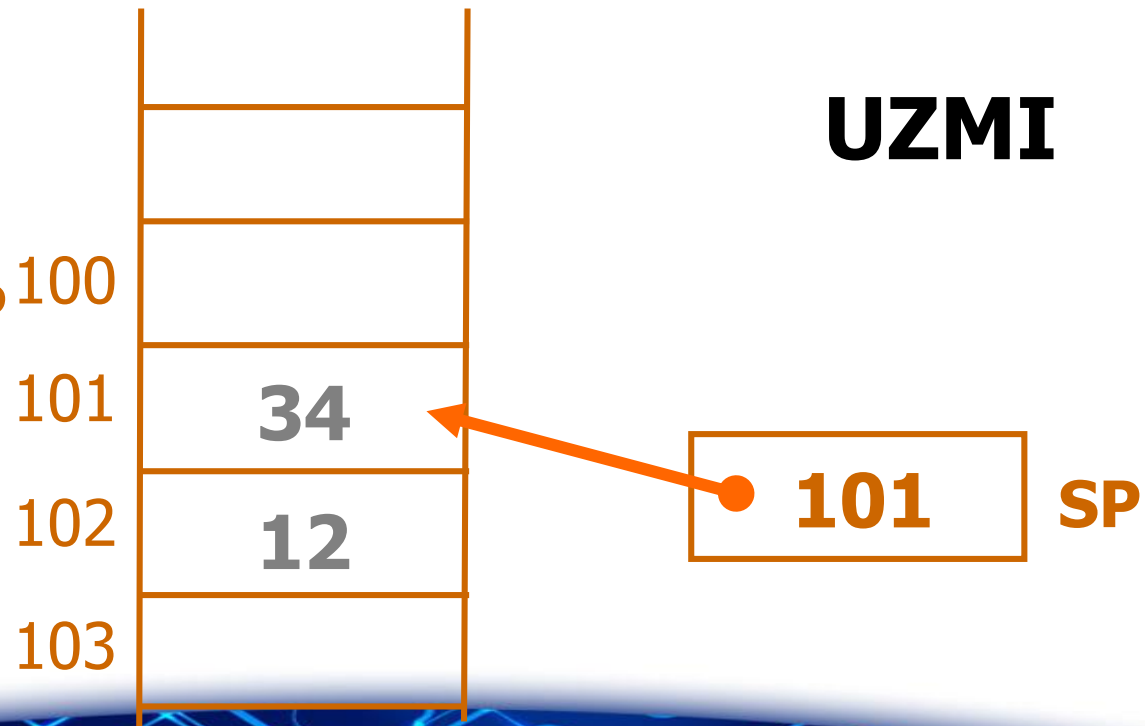
Kod uzimanja se
prvo podatak
uzima s mjesta
koje pokazuje SP



Stog

- Kako izgleda stog u memoriji računala?
- Određene memorijske lokacije koriste se kao područje za stog
- Registar SP se obično nalazi u procesoru

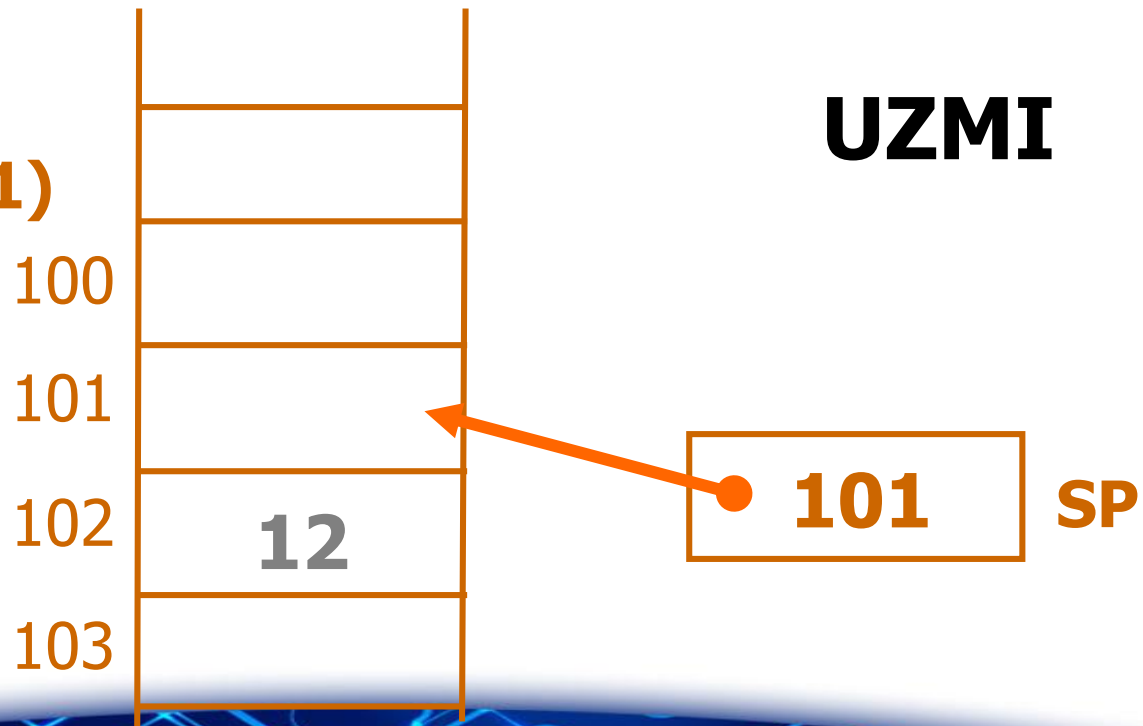
**Kod uzimanja se
prvo podatak
uzima s mjesta
koje pokazuje SP**



Stog

- Kako izgleda stog u memoriji računala?
- Određene memorijske lokacije koriste se kao područje za stog
- Registar SP se obično nalazi u procesoru

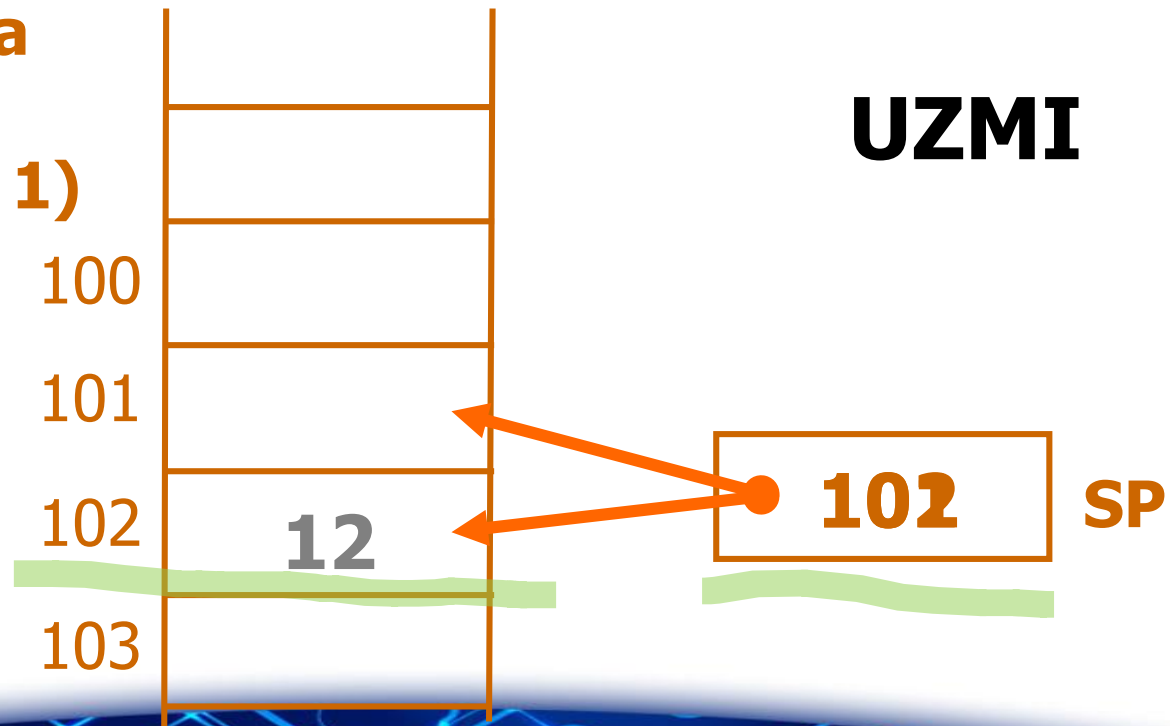
**Tek nakon toga
pomiče se SP
(povećava se za 1)**



Stog

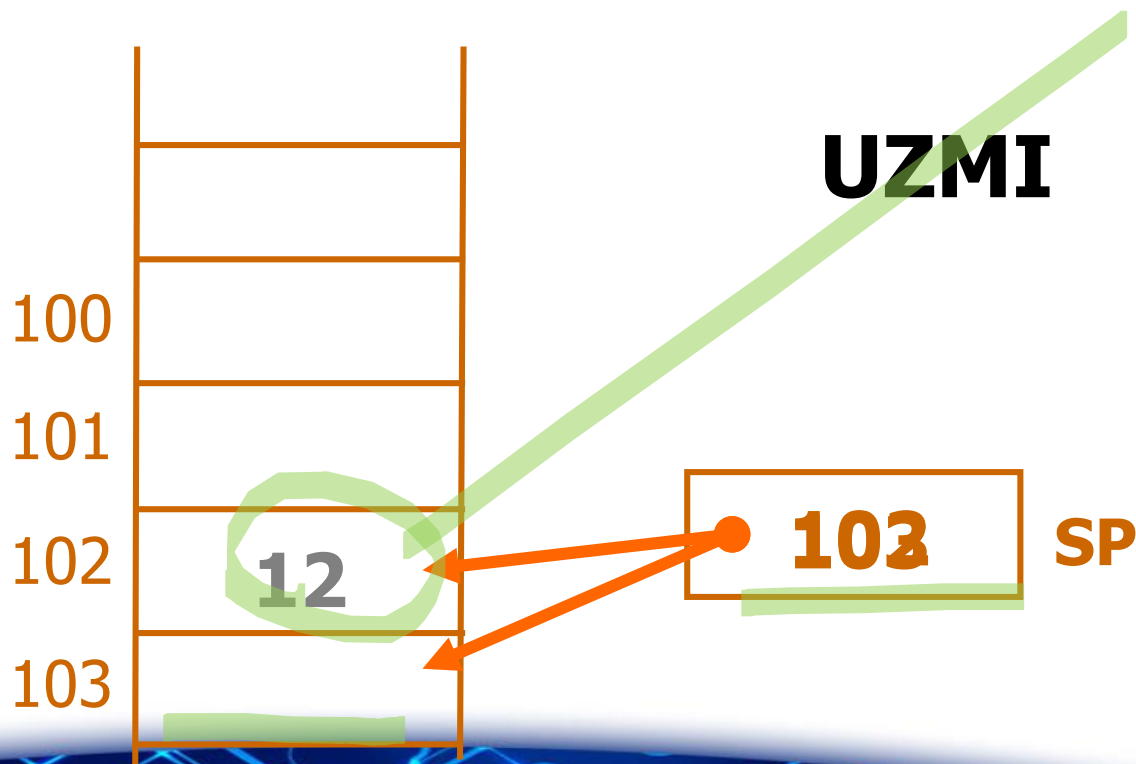
- Kako izgleda stog u memoriji računala?
- Određene memorijske lokacije koriste se kao područje za stog
- Registar SP se obično nalazi u procesoru

**Tek nakon toga
pomiče se SP
(povećava se za 1)**



Stog

- Kako izgleda stog u memoriji računala?
- Određene memorijske lokacije koriste se kao područje za stog
- Registar SP se obično nalazi u procesoru



Stog

- Upravo smo vidjeli da se pokazivač stoga SP mijenja prilikom operacija sa stogom:
 - kod stavljanja podatka na stog \rightarrow SP se smanjuje
 - kod uzimanja podatka sa stoga \rightarrow SP se povećava *
- Vidjeli smo i redoslijed koraka prilikom operacija sa stogom:
 - stavljanje: umanji SP, zatim stavi podatak
 - uzimanje: uzmi podatak, zatim uvećaj SP **

* moguć je i obrnut smjer "rasta/pada" stoga, ali se on u praksi rijeđe koristi

** moguća je i drugačija realizacija stoga - više o tome kasnije

Stog

- Kod stavljanja na stog, treba u naredbi **PUSH** reći što se stavlja na stog (npr. neki broj ili sadržaj nekog registra)
- Kod uzimanja sa stoga, treba u naredbi **POP** reći gdje se upisuje pročitani podatak (npr. u neki registar)
- Važno je još napomenuti:
 - Pri uzimanju, tj. čitanju podatka, taj se podatak ne "uzima" doslovno sa stoga, tj. nakon čitanja se podatak ne briše sa stoga.
 - U stvarnosti, podatak koji je "uzet" ostaje zapisan na stogu, a samo je registar SP promijenio svoju vrijednost tako da pokazuje na prethodni podatak!!!

Samostalno ponoviti gradivo

- DZ1 Podaci u računalu

Dokument se nalazi u repozitoriju predmeta AR1