Mikroprocesorska elektronika

Predavanje IV

Sadržaj predavanja

- Skup instrukcija
- Sistemske magistrale

Skup instrukcija

Skup instrukcija

- **Skup instrukcija CPU**, (ISA, Instruction Set Architecture), predstavlja deo *arhitekture mikroprocesora koji se odnosi na način njegovog programiranja*

- Skup instrukcija uključuje:
  - Definiciju različitih “komandi” (instrukcija) koje CPU prepoznaje i može da izvrši, najčešće organizovanih u odgovarajuće grupe (instrukcije prenosa, aritmetičke i logičke instrukcije, instrukcije kontrole toka izvršavanja programa, itd.)
  - Broj operanada i način pristupa operandima (adresni modovi)
  - Dužinu i format instrukcija

- Skup instrukcija ne čini mikroarhitekturu centralnog procesora; procesori sa različitom mikroarhitekturom mogu izvršavati isti skup instrukcija

- **Instrukcije CPU su smeštene u memoriji** kao i bilo koji podatak

- Ono što ih čini instrukcijama je činjenica da će biti **dekodovane i izvršene tokom CPU ciklusa**

Dužina instrukcija I

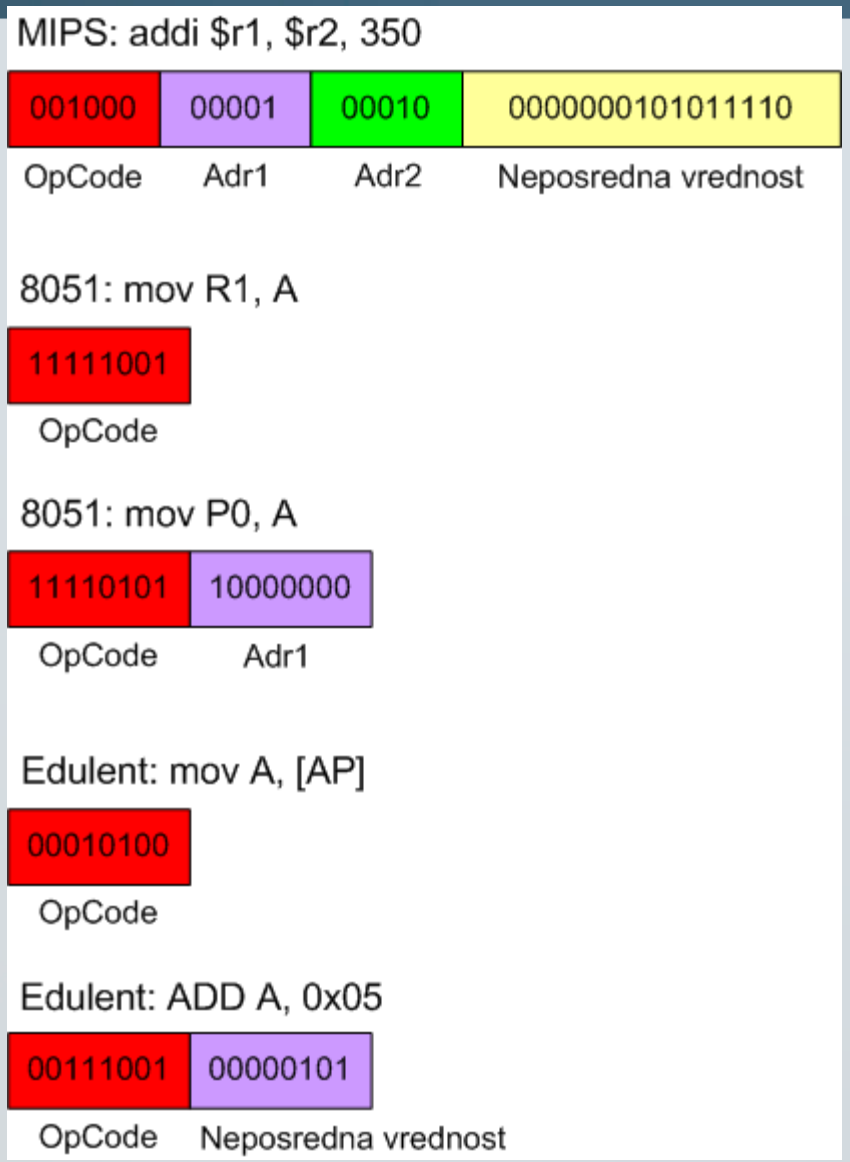
- Dužina instrukcije meri se potrebnim brojem bitova za njeno predstavljanje
- Svaka instrukcija mora sadržati barem jedno **polje**, koje definiše o kojoj instrukciji je reč
- Kompletna instrukcija može sadržati i čitav niz drugih polja, potrebnih da bi se tačno opisala željena operacija koju CPU treba da izvrši
- Dužina instrukcija u praksi varira od instrukcija dužine 4 bita (kod nekih jednostavnih mikrokontrolera), pa sve do instrukcija čija dužina iznosi nekoliko stotina bitova (kod nekih VLIW sistema)
- U praksi postoje dva pristupa problemu izbora dužine instrukcija:
  - Sve instrukcije imaju istu, fiksnu, dužinu
  - Instrukcije imaju varijabilnu dužinu

Dužina instrukcija II

- **Instrukcije fiksne dužine** se vrlo lako zahvataju iz memorije i dekoduju (najčešće u **jednom ciklusu**), što olakšava uvođenje **protočne obrade** i korišćenje različitih **tehnika paralelizacije**
- Instrukcije fiksne dužine koriste se u MIPS, DLX, ARM procesorima
- **Instrukcije promenljive dužine** po pravilu zahtevaju faze prihvata i dekodovanja koje traju više od jednog ciklusa, što otežava uvođenje protočne i paralelne obrade
- Sa druge strane instrukcije promenljive dužine omogućavaju projektovanje fleksibilnijeg i kompaktnijeg skupa instrukcija
- Instrukcije promenljive dužine koriste se u x86 procesorima
- ATmega328P- većina instrukcija su 16-bitne, neke su 32-bitne

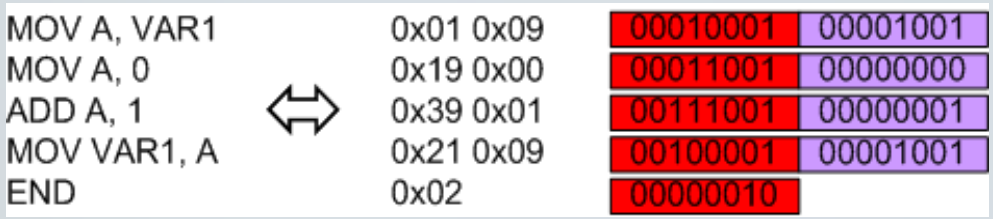
Format instrukcija

- Bitovi unutar instrukcije najčešće su **grupisani u polja**
- Svaka instrukcija mora sadržati barem jedno polje (**Operation Code**, **OpCode**) koje definiše operaciju koju je potrebno izvršiti
- Ostala polja sadrže informacije o operandima i adresnom modu koji je potrebno koristiti u tekućoj instrukciji
- Pod adresnim modom podrazumevamo način na koji se pristupa podacima koje je potrebno koristiti prilikom izvršavanja instrukcije
- Nakon koda operacije, tipično se nalazi informacija o nula, jednom, dva ili tri operanda, u zavisnosti od instrukcije



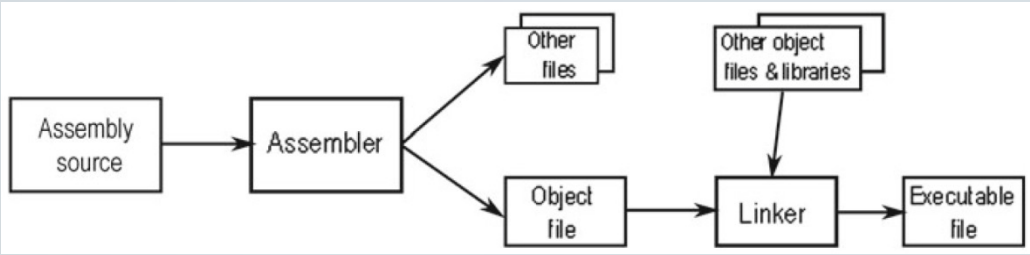
Asembler i asemblerske instrukcije I

- Da bi se izbeglo pisanje programa koristeći *mašinske instrukcije*, osmišljeni su asemblerski i viši programski jezici
- Svaki asemblerski jezik se bazira na korišćenju instrukcija
- **Svaka asemblerska instrukcija asocirana je tačno jednoj mašinskoj instrukciji**
- Ono što čini asemblerski jezik lakšim za korišćenje je njegova notacija koja je mnogo razumljivija čoveku
- Umesto direktnog pisanja 0 i 1, asemblerske instrukcije koriste **simboličke oznake**
- Sintaksa asemblerskih instrukcija prati sledeća pravila:
  - jedinstveno ime (**mnemonik**) za svaki dozvoljeni OpCode praćeno listom **operanada**, zapisanih uz korišćenje specijalnih simbola za definisanje adresnog moda



Asembler i asemblerske instrukcije II

- Programeri pišu asemblerski program koristeći neki od tekst editora, generišući takozvani izvorni **(*source*) fajl**
- Izvorni fajl sadrži asemblerske **instrukcije** kao i čitav niz asemblerskih **direktiva** i **komentara**
- Asemblerske direktive se koriste za kontrolu procesa asembliranja i **ne izvršavaju** se od strane procesora
- Direktive omogućavaju organizaciju memorije, definisanje konstanti, labela, itd...
- Izvorni fajl se zatim prosleđuje asembleru, koji prevodi asemblerske instrukcije u mašinske generišući **objektni (*object*) fajl**, kao i čitav niz pratećih fajlova koji olakšavaju proces otklanjanja grešaka (*debugging*)
- Objektni fajl se zatim šalji linkeru, koji kombinuje više objektnih fajlova kako bi kreirao **izvršni (*executable*) fajl**



Tipovi instrukcija I

- Na najnižem nivou, skup instrukcija svakog procesora sastoji se iz tri grupe instrukcija:

  - Instrukcije prenosa podataka
  - Aritmetičko-logičke instrukcije
  - Instrukcije za kontrolu toka izvršavanja programa

- Broj i format mašinskih instrukcija u svakoj od ovih grupa zavisi od procesora do procesora

Instrukcije za prenos podataka I

- Instrukcije prenosa podataka služe za prenos podataka sa jedne lokacije (*source*) u neku drugu lokaciju (*destination*)
- Pod lokacijom podrazumevamo unutrašnje registre procesora, memoriju ili registre koji se nalaze unutar neke od periferijskih jedinica
- Većina instrukcija iz ove grupe zapravo samo kopira podatak koji je smešten u polaznoj lokaciji u odredišnu, bez brisanja podatka u polaznoj lokaciji
- Izuzetak su instrukcije tipa *swap* koje zamenjuju mesta podacima unutar polazne i krajnje lokacije

Instrukcije za prenos podataka II

- Izvršavanje instrukcija za prenos podataka po pravilu uključuje BIL modul (za implementaciju sprežnog interfejsa) i neki od registara procesora

- Bitovi unutar statusnog registra uglavnom se ne modifikuju prilikom izvršavanja ove grupe instrukcija

- Primeri instrukcija koje pripadaju ovoj grupi su:
- ***move*** ili ***load*** – kopira podatak sa polazne lokacije u odredišnu
- ***swap*** – zamenjuje podatke koji se nalaze u polaznoj i odredišnoj lokaciji
- Operacije za rad sa stekom: ***push***, ***pop*** ili ***pull***
- Operacije za rad sa periferijama: ***in*** i ***out*** – manipulacija podacima koji se nalaze na ulazno/izlaznim portovima ili u modulima koji su mapirani u I/O adresnom prostoru

Aritmetičko-logičke instrukcije I

- Aritmetičko-logičke instrukcije su instrukcije namenjene izvođenju aritmetičkih i logičkih operacija nad podacima
- Najčešće u ovu grupu instrukcija spadaju i druge instrukcije koje operišu nad sadržajem registra ili memorijske lokacije: instrukcije poređenja, rotacije i pomeranja, itd...
- Prilikom izvršavanja ove grupe instrukcija koriste se unutrašnji registri, aritmetičko-logička jedinica kao i BIL modul
- Instrukcije iz ove grupe obično modifikuju bitove unutar statusnog registra

Aritmetičko-logičke instrukcije II

- Kod većine procesora instrukcije iz ove grupe uključuju samo dva operanda, odnosno obično imaju sledeći format

$$ciljni\_operand \leftarrow (ciljni\_operand \; \Delta \; polazni\_operand)$$

- gde je sa Δ označena aritmetičko-logička operacija

- Uobičajene operacije bi bile:
- Sabiranje
- Oduzimanje
- Logičke operacije I, ILI, XOR koje se izvode na nivou pojedinačnih bitova

- Operacije množenja i deljenja nisu podržane od strane svih aritmetičko-logičkih jedinica

Aritmetičko-logičke instrukcije III

- Set aritmetičko-logičkih operacija podržan od strane konkretnog procesora zavisi od arhitekture njegove **aritmetičko-logičke jedinice** i ne može se naknadno proširivati

- Operacije koje nisu direktno hardverski podržane moraju se realizovati softverski, ukoliko postoji potreba

Instrukcije poređenja i testiranja

- Ove instrukcije obično uključuju izvođenje operacije oduzimanja ili logičkog I bez stvarne modifikacije bilo kojeg operanda

- Cilj je da se utiče na vrednost statusnih bita unutar statusnog registra, čijom će se kasnijom proverom doneti odgovarajuće odluke

Aritmetičko-logičke instrukcije IV

Instrukcije rotiranja i pomeranja

- Umesto izvođenja binarne operacije, ove instrukcije modifikuju sadržaj ciljnog operanda, rotirajući ili pomerajući njegove **individualne bitove**

Instrukcije bitskih logičkih operacija

- U ovu grupu instrukcija spadaju instrukcije koje izvode neku logičku operaciju ali na nivou pojedinačnih bitova

$$ciljni\_operand(0) \leftarrow (ciljni\_operand(0) \Delta polazni\_operand(0))$$
$$ciljni\_operand(1) \leftarrow (ciljni\_operand(1) \Delta polazni\_operand(1))$$
$$...$$

- Bitske logičke operacije omogućavaju postavljanje, brisanje ili modifikaciju individualnih bitova bez uticaja na ostale

Instrukcije za kontrolu toka izvršavanja programa I

- Instrukcije za kontrolu toka izvršavanja programa omogućavaju da se modifikuje podrazumevani tok izvršavanja programa
- U podrazumevanom toku izvršavanja programa adresa naredne instrukcije uvek neposredno sledi adresu tekuće instrukcije koja se izvršava
- Ova adresa se automatski upisuje u programski brojač nakon što se dekoduje tekuća instrukcija
- U slučaju izvršavanja instrukcije kontrole toka izvršavanja, sadržaj programskog brojača se može promeniti tako da naredna instrukcija koja će se izvršiti ne bude instrukcija koja sledi neposredno nakon tekuće

Instrukcije za kontrolu toka izvršavanja programa II

- Postoji tri tipa instrukcija kontrole toka izvršavanja programa

Bezuslovni skok

- Instrukcija bezuslovnog skoka uvek menja sadržaj programskog brojača tako da pokazuje na adresu koja je deo same instrukcije

$$PC \leftarrow Nova\_adresa$$

Uslovni skok

- Kod ovih instrukcija sadržaj programskog brojača će biti promenjen samo ukoliko je odgovarajući uslov zadovoljen

Mnemonics	Meaning	Mnemonics	Meaning
`jz`	Jump if zero (Z = 1)	`jn`	Jump if negative (N = 1)
`jnz`	Jump if not zero (Z = 0)	`jp`	Jump if positive (N = 0)
`jc`	Jump if carry (C = 1)	`jv`	Jump if overflow (V = 1)
`jnc`	Jump if no carry (C = 0)	`jnv`	Jump if not overflow (V = 0)

Instrukcije za kontrolu toka izvršavanja programa III

Poziv i povratak iz potprograma

- Ove instrukcije omogućavaju programeru da prenese tok izvršavanja programa ka i od posebnih delova programa, poznatih pod nazivom potprogrami

Modovi adresiranja I

- Pod modovima adresiranja podrazumevamo različite načine za specifikaciju lokacije na kojoj se nalazi **operand** potreban za izvršavanje tekuće operacije
- Adresni mod se unutar asemblerske instrukcije označava korišćenjem posebne sintakse, koja može biti različita od procesora do procesora
- Informacije o adresnim modovima koje je neophodno koristiti nalaze se i unutar svake mašinske instrukcije, bilo kao **deo OpCode-a** ili pak u vidu **zasebnog polja** unutar instrukcije
- Dekodovanjem ovih informacija procesor određuje potrebnu sekvencu koraka koje je neophodno izvršiti kako bi se prihvatili polazni operandi potrebni za izvršavanje tekuće instrukcije kao i za smeštanje rezultata

Modovi adresiranja II

- U opštem slučaju, podatak koji je potrebno koristiti prilikom izvršavanja tekuće instrukcije može biti smešten na jednom od sledećih mesta:
  - Može biti neposredno naveden
  - Može biti smešten unutar nekog od unutrašnjih registara procesora
  - Može biti smešten u nekoj memorijskoj lokaciji
  - Može biti smešten u nekom ulazno/izlaznom portu ili registru unutar periferijske jedinice

- Sintaksa korišćena za specifikaciju adresnog moda unutar asemblerskog jezika govori asembleru kako da "dođe" do podatka koji je potreban

- Na žalost, ova sintaksa nije ista za različite procesore tako da je korisnik prinuđen da nauči odgovarajuću sintaksu za tekući procesor koji koristi u embeded sistemu

Modovi adresiranja III

- U slučaju kada se podatak navodi neposredno ili je smešten u nekom od unutrašnjih registara procesora, gotovo svi procesori podržavaju sledeća dva adresna moda:

  - **Neposredno adresiranje – sintaksa: #vrednost** – U ovom modu, vrednost operanda je neposredno navedena i predstavlja **deo asemblerske instrukcije**. Ovaj mod adresiranja može se koristiti samo za specifikaciju lokacije ulaznih operanada ne i za specifikaciju lokacije na kojoj je potrebno smestiti rezultat izvršavanja instrukcije.

  - **Registarsko adresiranje – sintaksa: Rn** – Operand koji je potrebno koristiti nalazi se smešten u registru Rn i tekuća vrednost operanda jednaka je tekućem sadržaju registra Rn. U slučaju ciljnog operanda, Rn određuje registar u koji će biti upisan rezultat izvršavanja tekuće instrukcije.

Modovi adresiranja IV

- U slučaju kada je podatak smešten u nekoj memorijskoj lokaciji, da bi se “došlo” do njega potrebno je na neki način specificirati adresu memorijske lokacije.

- Najčešći modovi adresiranja koji se koriste u ovom slučaju su:
  - **Apsolutno ili direktno adresiranje – sintaksa: adresa** – Vrednost polja *adresa* predstavlja adresu memorijske lokacije u kojoj je smešten ili je potrebno smestiti operand.

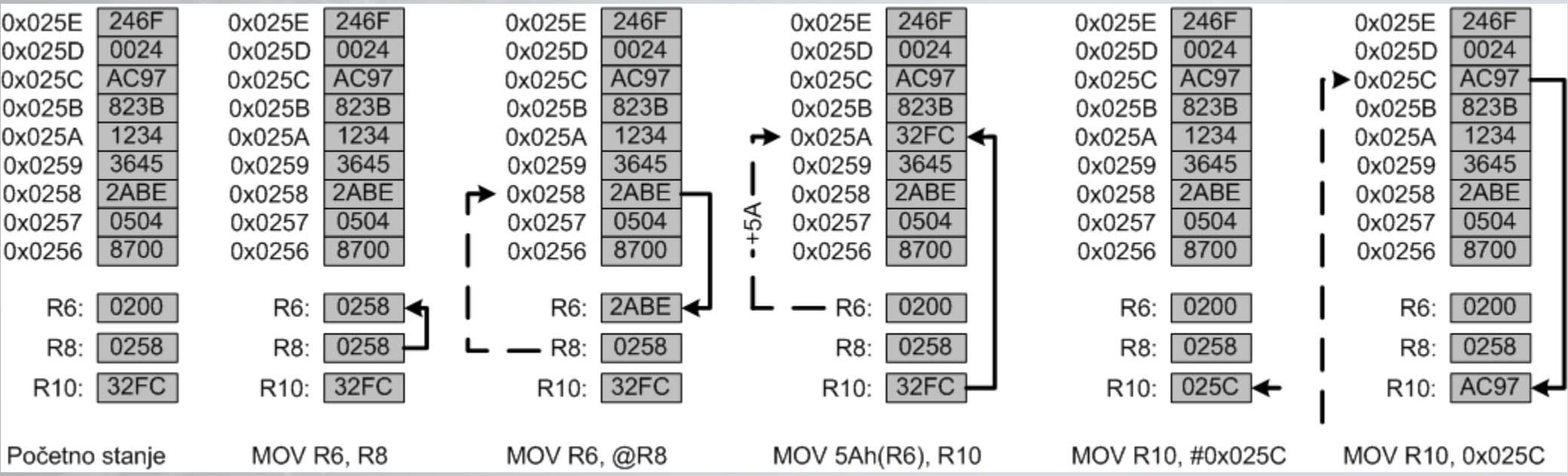
  - **Indirektno adresiranje – sintaksa: @Rn** – Operand se nalazi, ili ga je potrebno smestiti, na memorijskoj lokaciji čija je adresa predstavljena kao sadržaj registra Rn. U ovom slučaju se kaže da registar Rn **“pokazuje”** na memorijsku lokaciju gde je smešten, ili će biti smešten, operand.

  - **Indeksno adresiranje – sintaksa: X(Rn)** – Ovaj mod specificira adresu operanda kao zbir broja X i sadržaja registra Rn. Broj X se često zove bazna adresa (*base address*), a vrednost registra Rn indeks ili ofset (offset).

Primeri adresnih modova

- Primer ilustruje korišćenje različitih adresnih modova unutar instrukcija za prenos podataka

- Unutrašnji registri procesora su 16-bitni, kao i memorijske lokacije



ATmega328P

- **Tipovi instrukcija**:

1. AL instrukcije
2. Instrukcije grananja
3. Bitske instrukcije i bitske test instrukcije
4. Instrukcije prenosa podataka
5. MCU kontrolne instrukcije

- **Veći broj adresnih modova** za direktno i indirektno adresiranje programske (Flash) i memorije podataka (SRAM, registri, I/O memorija)

https://ww1.microchip.com/downloads/en/DeviceDoc/AVR-Instruction-Set-Manual-DS40002198A.pdf

RISC i CISC arhitekture I

- U praksi postoje dve velike grupe ISA:
  - CISC (*Complex Instruction Set Computer*) arhitektura
  - RISC (Reduced Instruction Set Computer) arhitektura

- CISC arhitekturu karakteriše:
  - Veliki broj kompleksnih instrukcija koje se izvršavaja tokom više taktova
  - Instrukcije su promenljive dužine
  - Veliki broj različitih modova adresiranja
  - Sve instrukcije mogu direktno pristupati podacima smeštenim u memoriji
  - Uglavnom ne koristi se protočna obrada
  - Relativno kratki programi

- CISC arhitekture fokusiraju se na obavljanje što je **veće količine posla** sa svakom izvršenom instrukcijom, kako bi se generisali što **jednostavni programi**

- Ovaj pristup olakšava rad programera, ali **usložnjava hardversku strukturu**

RISC i CISC arhitekture II

- RISC arhitekturu karakteriše:
  - Mali broj jednostavnih/specijalizovanih instrukcija
  - Instrukcije su fiksne dužine
  - Instrukcije se izvršavaju tokom konstantnog broja taktova (vrlo često u jednom taktu)
  - Mali broj različitih modova adresiranja
  - Većina instrukcija može da pristupa samo podacima smeštenim u internim registrima
  - Postoje posebne instrukcije za pristup podacima koji se nalaze u memoriji (load/store)
  - Koristi se protočna obrada
  - Dugački programi

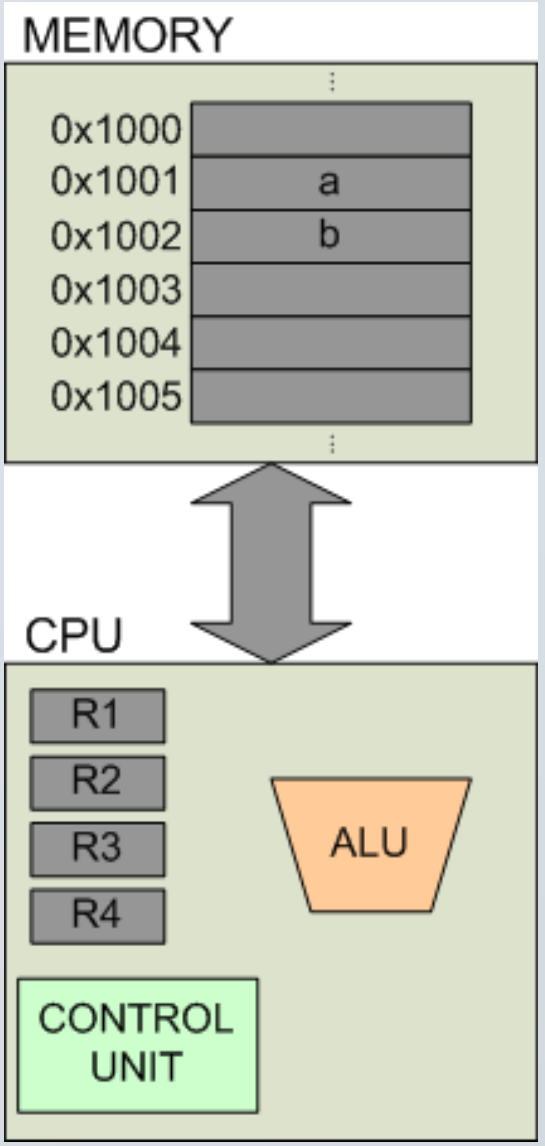
- RISC arhitekture fokusiraju se na pojednostavljivanje hardverske arhitekture sa ciljem da se ubrza učestanost na kojoj ona može da radi

- Ovaj pristup **pojednostavljuje hardversku strukturu**, ali otežava razvoj programa

RISC i CISC arhitekture III

- Na primer, pogledajmo kako bi se realizovalo množenje dva broja, *a* i *b*, smeštena u memoriji, na lokacijama 0x1001 i 0x1002, koristeći CISC i RISC arhitekturu
- U slučaju **CISC arhitekture** postojala bi **jedna instrukcija** (zvaćemo je MULT) tako da bi se operacija množenja mogla realizovati izvršavanjem samo jedne instrukcije

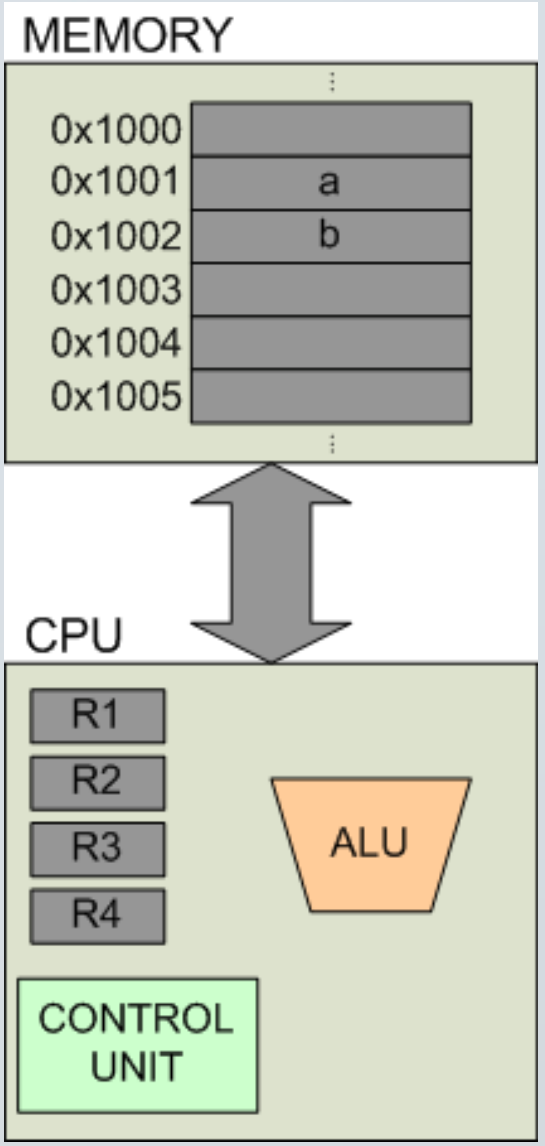
  MULT a, b
- Tokom izvršavanja ove instrukcije, procesor bi:
  - Učitao vrednosti promenljivih *a* i *b* u interne registare (R1-R4)
  - Izvršio operaciju množenja
  - Smestio rezultat u memorijsku lokaciju gde se čuva promenljiva *a*
- MULT instukcija je primer "kompleksne instrukcije", koja direktno radi nad operandima smeštenim u memoriji i ne zahteva od programera da eksplicitno učitava i upisuje podatke u memoriju
- MULT instrukcija podseća na naredbe viših programskih jezika



RISC i CISC arhitekture IV

- U slučaju RISC arhitekture operacija množenja bi se morala realizovati korišćenjem više "jednostavnih instrukcija":
  - LOAD instrukcije, koja prebacuje podatak iz memorije u neki od internih registara procesora
  - PROD instrukcije, koja množi sadržaj dva interna registra i rezultat smešta u interni registar
  - STORE instrukcije, koja prebacuje podatak iz nekog od internih registara u memoriju
- Da bi izvršio operaciju množenja, programer sada eksplicitno mora da napiše sledeće četiri linije asemblerskog koda:

LOAD R1, a
LOAD R2, b
PROD R1, R2
STORE a, R1



RISC i CISC arhitekture V

- Na prvi pogled, čini se da **RISC arhitekture** predstavljaju manje efikasan način za realizaciju računskih operacija
- Obzirom da se RISC programi sastoje iz većeg broja instrukcija, potrebna je veća memorija za njihovo smeštanje
- Kompajler sada ima komplikovaniji zadatak da konvertuje naredbe nekog visokog programskog jezika (C, C++, ...) u odgovarajući asemblerski program
- Međutim, RISC arhitektura takođe ima i nekoliko bitnih prednosti:
  - Obzirom da svaka instrukcija zahteva jedan takt za izvršenje, ukupno trajanje izvršavanja operacije množenja je **približno jednako** kao i u slučaju MULT instrukcije koja se izvršava tokom više taktova
  - RISC instrukcije su jednostavnije za hardversku implementaciju, zahtevajući manji broj hardverskih resursa od CISC instrukcija što rezultuje u manjim, bržim procesorima koji troše manje električne energije prilikom izvršavanja instrukcija
  - Obzirom da su sve RISC instrukcije iste širine, uvođenje protočne obrade je znatno jednostavnije

Sistemske magistrale

Sistemske magistrale

- Procesor pristupa memoriji i ulazno/izlaznim jedinicama pomoću sistemskih magistrala
- Magistrala predstavlja skup linija koje ostvaruju zajedničku funkciju
- Svaka od linija nosi jedan bit informacije, a grupa ovakvih bitova može da se interpretira kao celina
- Sistemske magistrale grupisane su u tri klase:
  - Magistralu podataka
  - Adresnu magistralu
  - Upravljačku magistralu

Magistrala podataka

- Skup linija koje prenose podatke ili instrukcije od i ka procesoru naziva se **magistrala podataka**

- Dve operacije mogu se difinisati za magistralu podataka:

  - Operacija **čitanja** – izvršava se svaki put kada se informacija prenosi **ka procesoru**
  - Operacija **upisa** – izvršava se svaki put kada podaci prenose **od procesora** ka memoriji ili periferijskoj jedinici

- Jedan transfer informacija naziva se **transakcijom na magistrali podataka**

- Broj individulanih linija koje čine magistralu podataka određuje maksimalnu širinu podatka koju procesor može da pošalje ili primi tokom jedne transakcije

Adresna magistrala I

- Procesor u svakom trenutku interaguje samo sa jednom memorijskim blokom ili periferijskom jedinicom
- Svaki memorijski ili registarski element, ima jedinstvenu identifikacionu oznaku koja se naziva **adresa**
- Skup linija koje prenose informaciju o adresi registra sa kojim se želi ostvariti komunikacija čine **adresnu magistralu**
- Linije koje čine adresnu magistralu su obično unidirekcione i polaze od procesora
- Broj linija koje čine adresnu magistralu određuje maksimalnu veličinu adresnog prostora koji dati procesor može da adresira
- Adresna magistrala od m bita može da adresira najviše 2^m različitih lokacija

Adresna magistrala II

- **Primer 1:**

  Koliko različitih memorijskih lokacija je moguće adresirati i koji je njihov adresni opseg (početna i krajnja vrednost adrese) u slučaju da se koristi adresna magistrala od:

  a) 12 bita

  b) 22 bita

Adresna magistrala II

- **Primer 1:**

  Koliko različitih memorijskih lokacija je moguće adresirati i koji je njihov adresni opseg (početna i krajnja vrednost adrese) u slučaju da se koristi adresna magistrala od:

  a) 12 bita

  b) 22 bita

- **Rešenje:**

a) U slučaju korišćenja 12-bitne adresne magistrale, postoji 2^{12}=4096 različitih memorijskih lokacija koje je moguće adresirati. Početna i krajnja vrednost adresa su 0000 0000 0000B i 1111 1111 1111B ako koristimo binarnu reprezentaciju, odnosno 0x000 i 0xFFF ako koristimo heksadecimalnu reprezentaciju.

b) U slučaju 22-bitne adresne magistrale, postoji 2^{22}= 4194304 različitih memorijskih lokacija koje je moguće adresirati. Početna i krajnja vrednost adresa su 0x000000 i 0x3FFFFF.

Adresna magistrala III

- **Primer 2:**

  Odgovarajući sistem raspolaže memorijom čija je veličina 32768 memorijskih lokacija. Koji je najmanji broj linija adresne magistrale koja se može koristiti za adresiranje ove memorije?

Adresna magistrala III

- **Primer 2:**

  Odgovarajući sistem raspolaže memorijom čija je veličina 32768 memorijskih lokacija. Koji je najmanji broj linija adresne magistrale koja se može koristiti za adresiranje ove memorije?

- **Rešenje:**

  Minimalni broj potrebnih linija adresne magistrale, *m*, koji je neophodan da bi se adresirala memorija od *k* memorijskih lokacija, može se izračunati korišćenjem sledećeg izraza:

$$m = \lceil \log_2(k) \rceil$$

U našem primeru, $m = \lceil \log_2(32768) \rceil = 15.$

Upravljačka magistrala

- Upravljačka magistrala grupiše sve linije koje prenose signale koji upravljaju nekom od aktivnosti unutar sistema
- Za razliku od adresne i magistrale podataka, kod kojih se vrednosi individualnih linija posmatraju kao celina (adresa, odnosno podatak), signali upravljačke magistrale obično rade i interpretiraju se odvojeno
- Upravljački signali uključuju:
  - Signale koji pružaju informaciju o tome da li procesor izvršava operaciju čitanja ili upisa
  - Signale koji sinhronizuju prenos podataka, označavajući kada transakcija počinje i završava
  - Signale koji pružaju informaciju da li neki od uređaja pristunih u sistemu zahteva servis od strane procesora
- Većina linija upravljačke magistrale su unidirekcione i one ili ulaze ili napuštaju procesor, u zavisnosti od njihove namene
- Broj i funkcija linija upravljačke magistrale zavisi od arhitekture procesora

Teorijska pitanja P4

Predavanja 4- Organizacija mikroračunarskog sistema 2

Spisak teorijskih pitanja uz Predavanja 4

1. Skup instrukcija - dužina i format instrukcija.

2. Asembler i asemblerske instrukcije.

3. Navesti osnovne tipove instrukcija i ukratko objasniti svaku grupu.

4. Instrukcije za prenos podataka.

5. Aritmetičko-logičke instrukcije.

6. Instrukcije za kontrolu toka izvršavanja programa.

7. Navesti tipove instrukcija mikrokontrolera ATmega328P.

8. Modovi adresiranja.

Predavanja 4- Organizacija mikroračunarskog sistema 2

Spisak teorijskih pitanja uz Predavanja 4

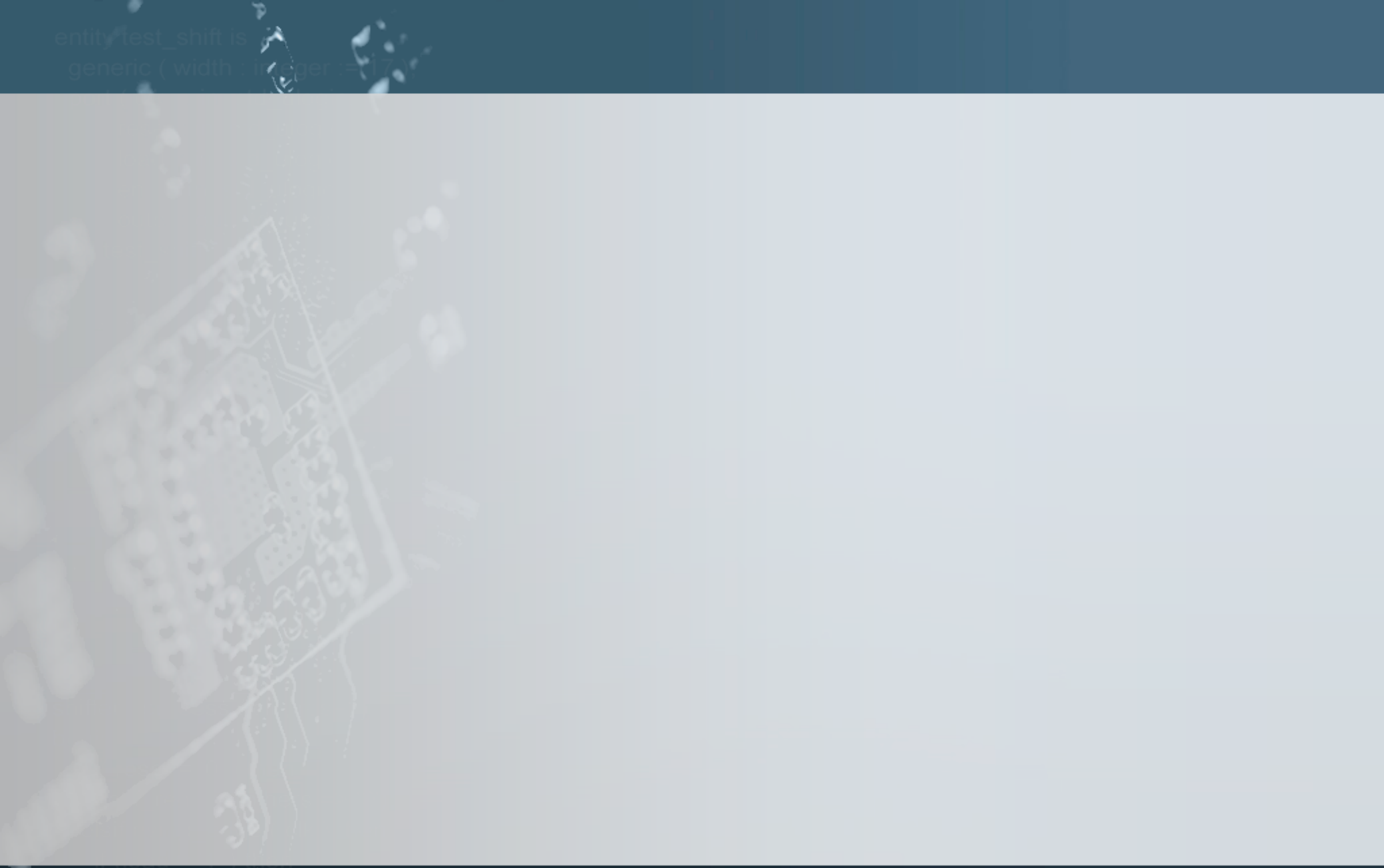
9. RISC i CISC arhitekture procesora. Navesti osnovne razlike, prednosti i mane. Ilustrovati princip rada jedne i druge grupe na primeru.

10. Navesti osnovne vrste i funkciju sistemskih magistrala.

11. Magistrala podataka.

12. Adresna magistrala.

13. Upravljačka magistrala.


entity test_shift is
generic (width : integer :=
shift_reg <= unsigned (inp);
elsif (en = '1') then