

```
entity test_shift is
  generic ( width : integer := 17 );
  port ( clk : in std_ulogic;
        reset : in std_ulogic;
        load : in std_ulogic;
        en : in std_ulogic;
        outp : out std_ulogic );
end test_shift;
```

Mikroprocesorska elektronika

Predavanje V

```
shifter : process (en, reset)
begin
  if ( reset = '0' ) then
    shift_reg <= (others => '0');
  elsif rising_edge ( clk ) then
    if (load = '1' ) then
      shift_reg <= unsigned (inp);
    elsif ( en = '1' ) then
```

Sadržaj predavanja

- Organizacija memorije
- Primer centralnog procesora: Edulent

```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```

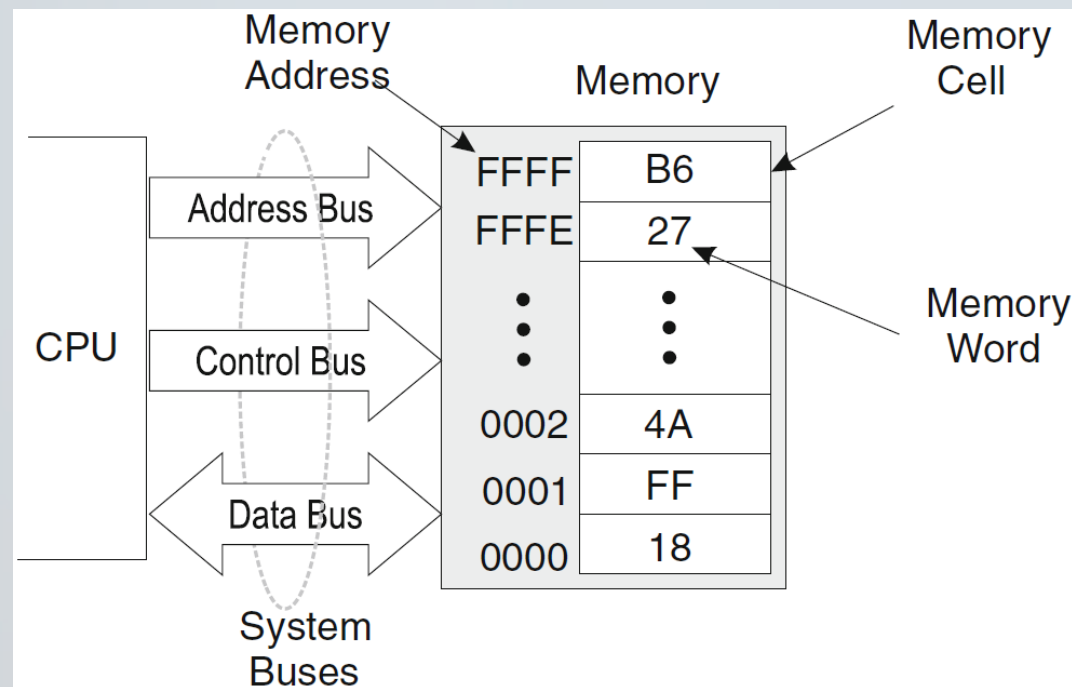
```
entity test_shift is
  generic ( width : integer := 17 );
  port ( clk : in std_ulogic;
        reset : in std_ulogic;
        load : in std_ulogic;
        en : in std_ulogic;
        outp : out std_ulogic );
end test_shift;
```

Organizacija memorije

```
shifter : process (en, reset)
begin
  if ( reset = '0' ) then
    shift_reg <= (others => '0');
  elsif rising_edge ( clk ) then
    if (load = '1' ) then
      shift_reg <= unsigned (inp);
    elsif ( en = '1' ) then
```

Organizacija memorije I

- Memorijski podsistem služi za **smeštanje instrukcija programa** kao i **podataka** koje taj program obrađuje
- Memorija se sastoji od velikog broja hardverskih komponenti koje su u stanju da skladište **1 bit informacije**
- Ovi bitovi su tipično organizovani u ***n*-bitne reči**, koje se tipično nazivaju **registrima** ili **lokacijama**
- Sadržaj** svake memorijske lokacije zove se **memorijska reč**
- Pored toga, svaka memorijska lokacija identifikuje se pomoću jedinstvenog identifikatora, **memorijske adrese**
- Procesor koristi memorijsku adresu da bi pristupio željenoj memorijskoj lokaciji kako bi u nju upisao ili pročitao memorijsku reč



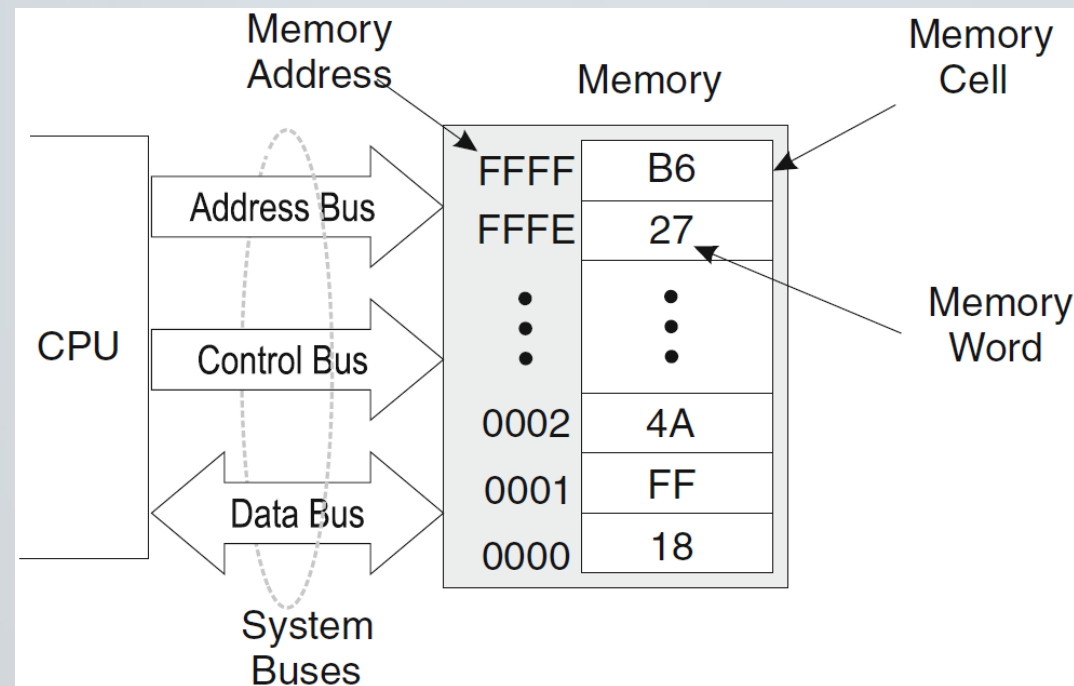
Organizacija memorije II

- U opštem slučaju, **memorijska jedinica** sastoji se iz **m memorijskih lokacija** veličine **n bita**
- Obično se memorija označava kao **$m \times n$ memorija**
- **Kapacitet** memorije navodi se brojem iza kojega sledi slovo **b** ili **B**
- **Na primer** kada kažemo da je memorija veličine 1Mb (jedan megabit) ili 1MB (jedan megabajt) organizacija može biti $1M \times 1$ i $1M \times 8$, respektivno
- **Adrese memorijskih lokacija** su obično numerisane u sekvencijalnom redosledu

```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```

Organizacija memorije III

- Memorija prikazana na slici sastavljena je od **64K memorijskih lokacija**
- Svaka **memorijska lokacija** u stanju je da skladišti **8-bitnu reč** (bajt)
- Imajući ovo u vidu, **kapacitet** memorije iznosi **64KB**
- Procesor koristi **adresnu magistralu** da selektuje memorijsku lokaciju sa kojom želi da komunicira
- Prenos podataka** između procesora i memorijske lokacije obavlja se preko **magistrale podataka**
- Procesor koristi **kontrolnu magistralu** da bi **specificirao tip operacije** koju je potrebno realizovati: upis ili čitanje iz memorije



Vrste memorija I

- Memorije se klasifikuju prema dva kriterijuma:
 - Mogućnost **očuvavanja upisanog sadržaja**
 - Mogućnost **upisa u memoriju**
- Sve memorije omogućavaju čitanje sadržaja iz memorije
- Sa stanovišta očuvanja upisanog sadržaja memorije se dele na:
 - **Memorije bez gubitka sadržaja (*nonvolatile*)** – sadržaj upisan u memoriju ne gubi se sa prestankom napajanja
 - **Memorije sa gubitkom sadržaja (*volatile*)** – sadržaj upisan u memoriju gubi se sa prestankom napajanja

```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```

Vrste memorija II

- Sa stanovišta mogućnosti upisa memorije se dele na:
 - **write/read memorije**
 - **In-system programmable (ISP) memorije**
 - **read only memorije**
- U prvu grupu spadaja memorije u koje **procesor** može **neometano** da **upisuje** podatke tokom rada embeded sistema
- Većina memorija sa gubitkom sadržaja spada u ovu grupu, te su vrlo zgodne za smeštanje **privremenih podataka** ili podataka koji će biti generisani ili modifikovani od strane programa
- Neke od memorija bez gubitka sadržaja (FRAM, *Ferro electric RAM*) takođe spadaja u ovu grupu

```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```


Vrste memorija III

- U grupu **in-system programabilnih memorija** spadaju memorije bez gubitka sadržaja u koje je upis po pravilu dozvoljen samo **tokom učitavanja programa**, ali **ne i tokom izvršavanja programa**
- Jedan od glavnih razloga leži u činjenici da je **brzina upisa** u ovu grupu memorija po pravilu vrlo **mala**
- Drugi aspekt koji upis u memoriju čini otežanim je potreba korišćenja **većih napona napajanja** prilikom upisa od napona koji su neophodni prilikom čitanja podataka
- Ova potreba dovodi do **povećane potrošnje** u sistemu i zahteva složenije sisteme za napajanje

```
shift_reg <= unsigned(inp);  
elsif (en == '1') then
```

Vrste memorija IV

Mogućnost čuvanja podataka	Tip memorije	In-system programabilna	Komentar
Memorija bez gubitka sadržaja	Masked ROM	Ne	Nije programabilna
	OTPROM	Ne	Može se programirati samo jednom, pomoću programatora
	EPROM	Ne	Izbrisiva i programabilna uz korišćenje programatora
	EEPROM	Da	Spora za brisanje/upis. Nije preporučljiv upis tokom izvršavanja programa. Zahteva viši napon prilikom upisa.
	Flash	Da	Slična EEPROM memoriji
	FRAM	Da	Brz upis u memoriju uz korišćenje niskih napona napajanja
Memorija sa gubitkom sadržaja	SRAM (Static RAM)	Da	Najbrža od svih memorije što se tiče upisa/čitanja
	DRAM (Dynamic RAM)	Da	Brzi upis/čitanje, ali ne kao kod SRAM-a

```

shift_reg <= unsigned (inp);
elsif ( en = '1' ) then

```

Programska i memorija podataka I

- **Unutar mikroračunarskog sistema** razlikuju se dve vrste memorije u zavisnosti od tipa informacije koja se u njima smešta:
 - **Programska memorija**
 - **Memorija podataka**
- Programska memorija, predstavlja deo memorije u kojem se smeštaju programi koje procesor izvršava
- Pod programom podrazumevamo **logičku sekvencu instrukcija** koja opisuje željenu funkcionalnost sistema
- U embeded sistemima programi su obično nepromenljivi i moraju uvek biti dostupni procesoru da bi čitav sistem radi ispravno

```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```

Programska i memorija podataka II

- Čak i kada se isključi napajanje sistema pa zatim ponovo uključi programi moraju i dalje biti dostupni kako bi sistem mogao ispravno funkcionisati
- Zbog ovog zahteva **programi** se unutar embeded sistema obično smeštaju u neku vrstu memorije **bez gubljenja sadržaja (ROM)**
- **Kapacitet programske memorije** obično se meri u kilobajtima
- **Memorija podataka** namenjena je skladištenju podataka za koje se očekuje da će se promeniti tokom izvršavanja programa
- Zbog ovog razloga memorija podataka mora da omogući **laku i jednostavnu izmenu svog sadržaja**, pa se obično realizuje kao **RAM memorija**

```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```

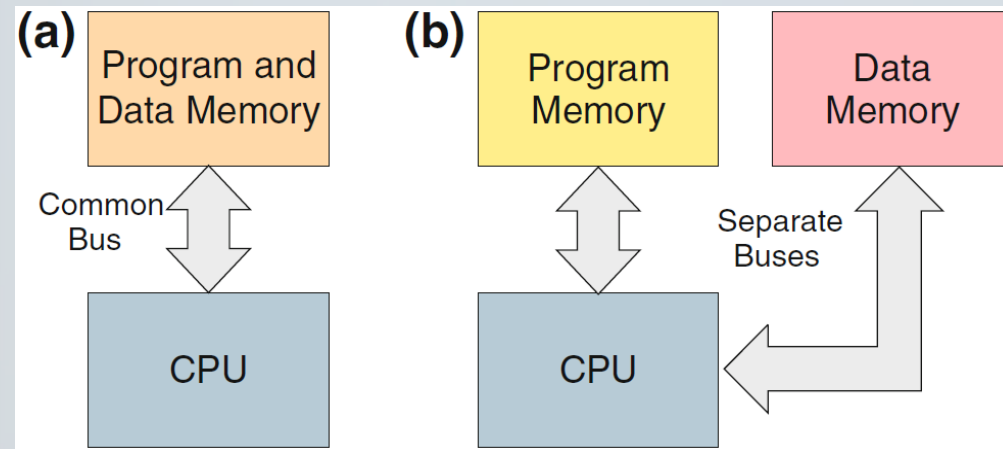
Programska i memorija podataka III

- Obzirom da je memorija podataka namenjena čuvanju privremenih informacija, ona se tipično realizuje kako **memorija sa gubitkom sadržaja**
- Prosečna veličina memorije podataka unutar embeded sistema relativno je mala
- U slučaju da postoje podaci čiji se gubitak nakon isključivanja napona napajanja ne može tolerisati oni se smeštaju unutar ROM sekcije, odnosno unutar programske memorije
- Neki mikrokontroleri (Intel 8051) dozvoljavaju direktan pristup ovim podacima, iako nije moguća njihova modifikacija
- Kod drugih mikrokontrolera prvo je neophodno kopirati podatke u RAM memoriju da bi se sa njima nakon toga moglo manipulirati

```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```

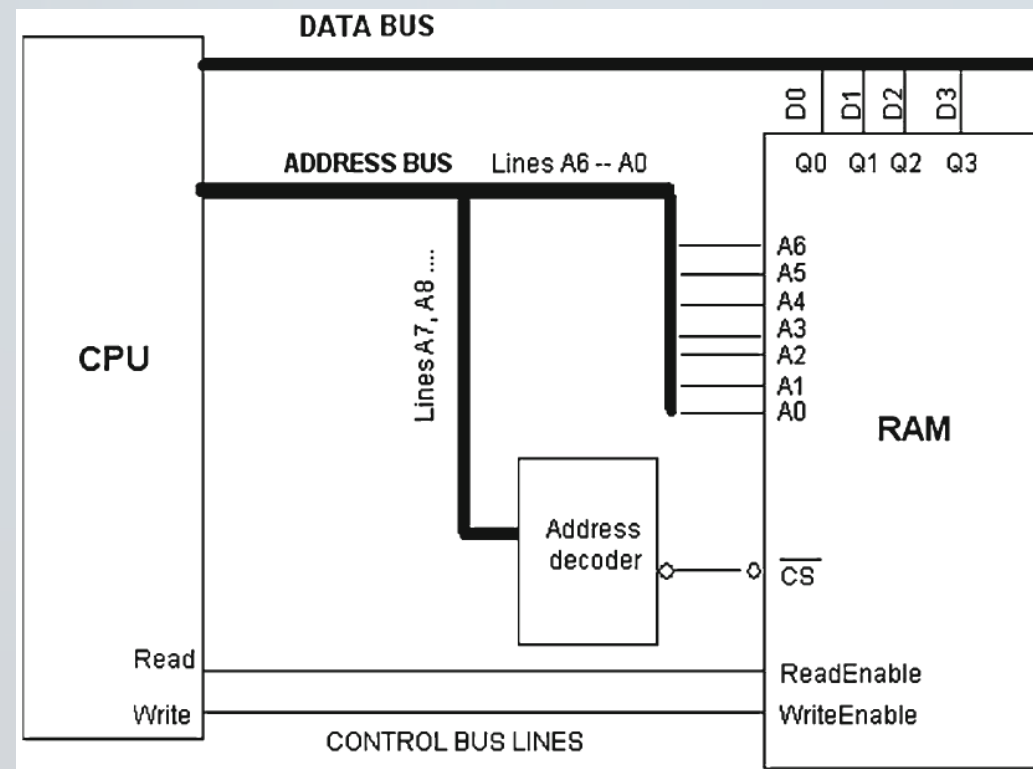
Von Neumann i Harvard arhitekture

- Programska i memorija podataka **moгу da dele sistemske magistrale ili ne**, u zavisnosti od arhitekture procesora
- U zavisnosti od toga da li programska i memorija podataka dele ili ne dele sistemske magistrale **svi mikroračunarski sistemi** mogu se podeliti u dve velike grupe:
 - Sisteme sa **von Neumann** arhitekturom – kod kojih se i programskoj i memoriji za podatke pristupa preko jedne **zajedničke grupe sistemskih magistrala**. U ovom slučaju programska i memorija podataka **dele isti adresni prostor** i imaju **istу širinu magistrale podataka**
 - Sisteme sa **Harvard** arhitekturom – kod koji postoje **dve odvojene grupe sistemskih magistrala** za pristup programskoj i memoriji podataka. U ovom slučaju postoje **fizički odvojeni adresni prostori** za programsku i memoriju podataka. **Širine adresne i magistrale** podataka programske i memorije podataka u ovom slučaju **moгу biti različite**



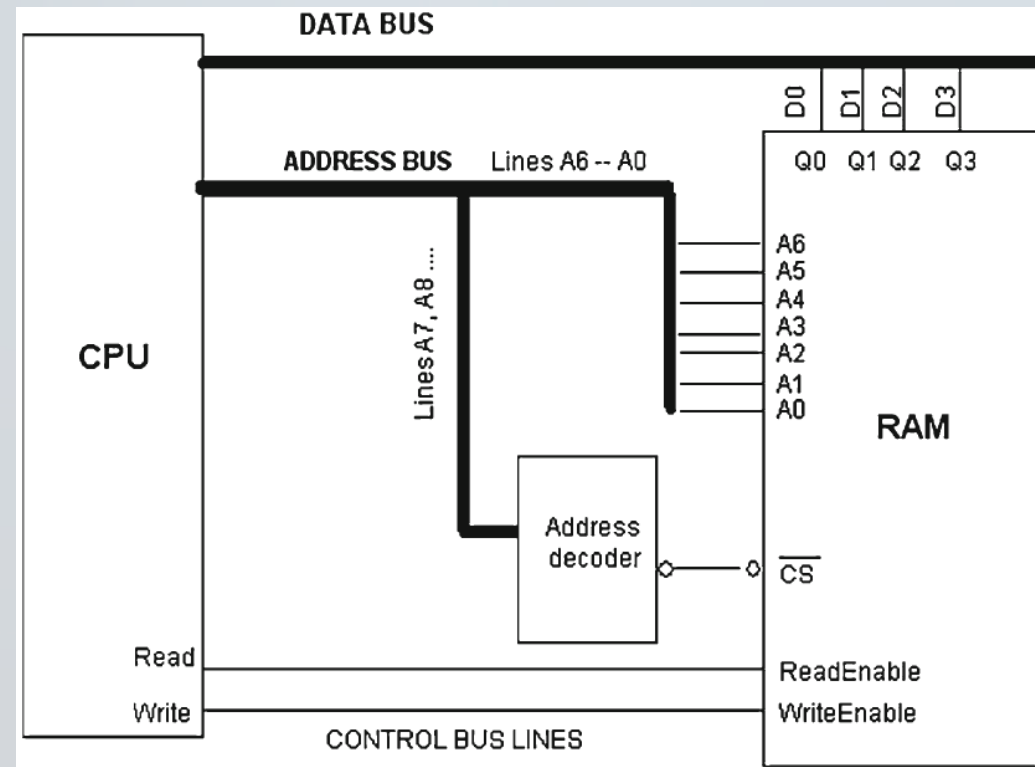
Povezivanje procesora i memorije I

- **Portovi RAM** memorije mogu se podeliti u tri grupe:
 - **Ulazno/izlazni portovi za podatke** (Q0, Q1, ...)
 - **Adresni portovi** (A0, A1, ...)
 - **Kontrolni portovi** (CS, ReadEnable, WriteEnable)
- **Magistrala podataka** procesora povezuje se na ulazno/izlazne portove za podatke memorije
- **Operacije čitanja i upisa** u memoriju pod kontrolom su procesora povezivanjem odgovarajućih signala kontrolne magistrale na **ReadEnable** i **WriteEnable** portove memorije



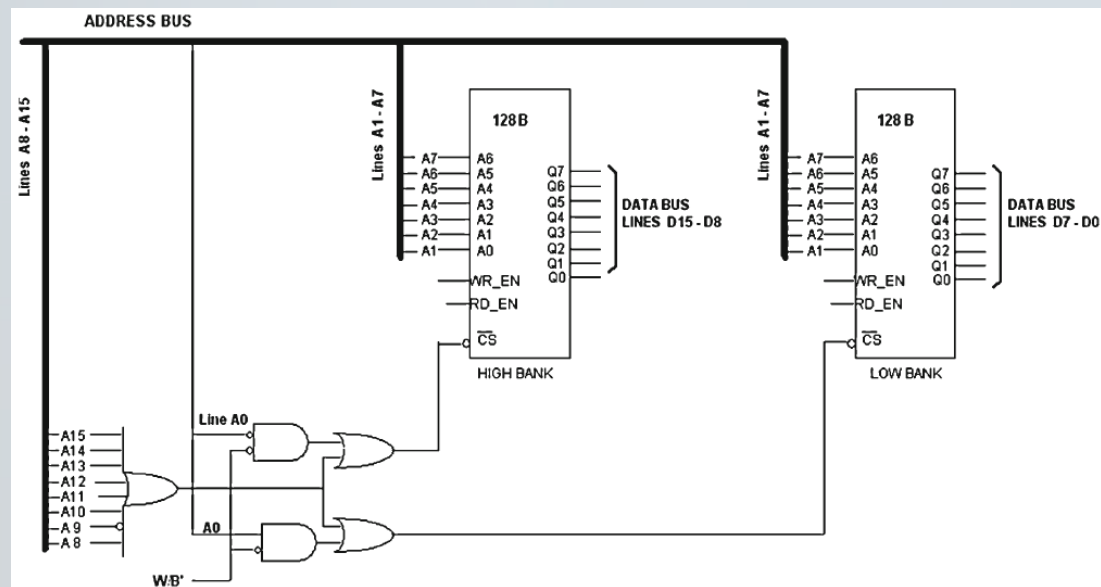
Povezivanje procesora i memorije II

- N signala **adresne magistrale procesora** podeljeni su u dve grupe:
 - Adresne linije koje su **direktno povezane** sa adresnim portovima memorije (A0...A6)
 - Adresne linije koje se dovode na **ulaz adresnog dekodera** (A7, A8, ...) čiji izlaz je povezan na CS ulaz memorije
- **Adresa memorijske lokacije** formira se kao kombinacija dva dela:
 - **Bazne adrese** koja aktivira memoriju, preko CS porta, i
 - **Offset adrese** koja selektuje odgovarajuću memorijsku lokaciju unutar memorije



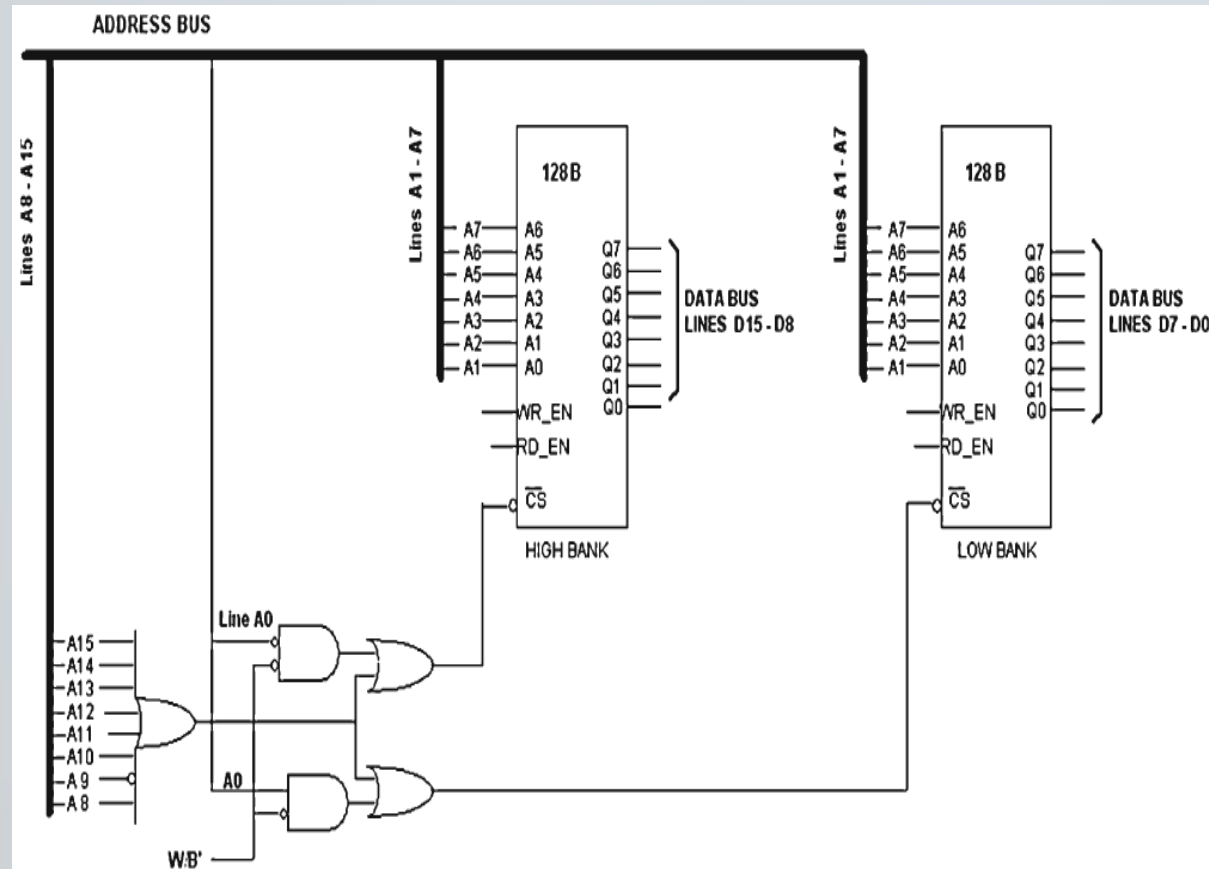
Primer povezivanje procesora i memorije I

- Na slici su prikazana **dva 128 B memorijska modula** povezana na **16-bitnu magistralu podataka**, **16-bitnu adresnu magistralu** i kontrolni signal **W/B**
- Obzirom da svaki od memorijskih modula ima magistralu podataka širine 8 bita, potrebna su dva modula da pokriju svih **16 bita magistrale podataka procesora**
- Kontrolni signal W/B** određuje da li procesor želi da pristupa **dvobajtnoj reči (word)**, ako je **W/B=1**, ili **jednobajtnoj reči (bajt)**, ako je **W/B=0**
- Odrediti opseg adresa memorijskog sistema kao i adresne opsege oba 128B modula*



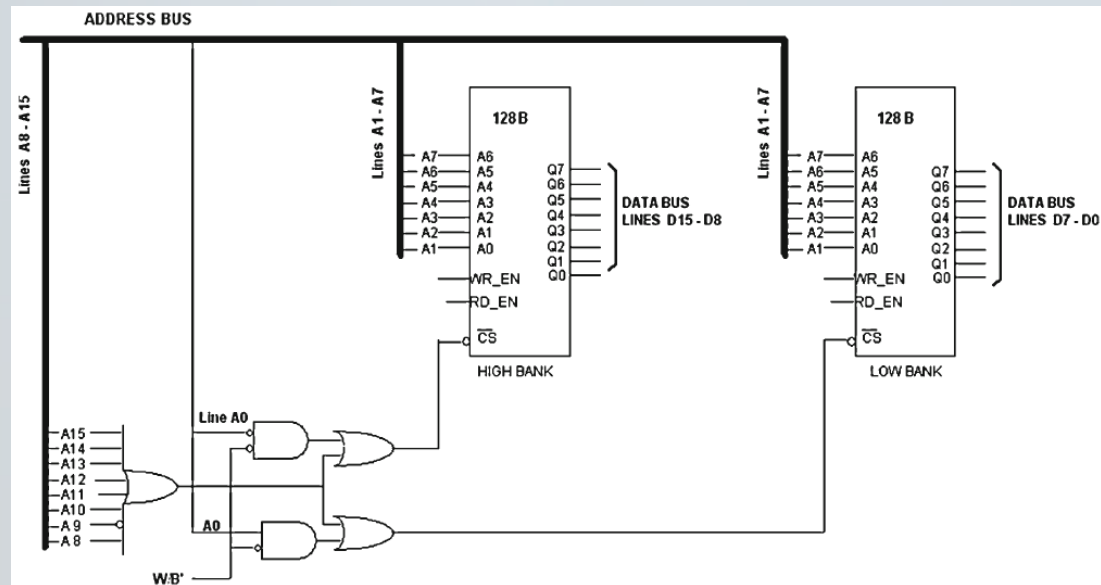
Primer povezivanje procesora i memorije II

- Da bi se memorijski moduli **aktivirali** potrebno je da njihovi **CS signali budu na niskom nivou**
- Analizom blok dijagrama možemo videti da je neophodan uslov da bilo koji od **CS** signala bude **na niskom nivou**
 $A_{15}=A_{14}=A_{13}=A_{12}=A_{11}=A_{10}=0$, $A_9=1$, $A_8=0$
- Adrese A_7-A_1 aktiviraju interne linije odgovarajućeg memorijskog modula
- Ukoliko je signal **$W/B=1$** oba modula su **aktivirana**, bez obzira na vrednost adresnog bita A_0
- U slučaju da je $W/B=0$ tada **$A_0=0$** aktivira donju banku, a **$A_0=1$** aktivira gornju banku



Primer povezivanje procesora i memorije III

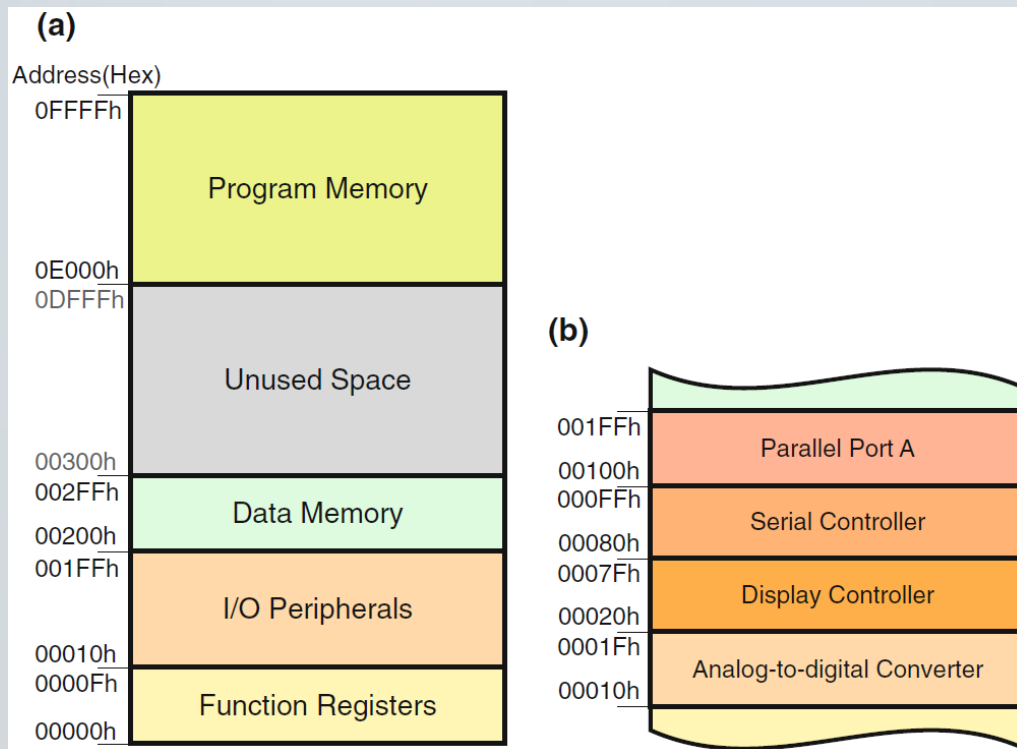
- **Opseg adresa** koje pokriva cela memorija:
 - Početna adresa: 0000 0010 0000 0000B; 0200h
 - Krajnja adresa: 0000 0010 1111 1111B; 02FFh
- **Donja banka:**
 - Aktivirana kada je **A0=0**, što znači da pokriva sve parne adrese u opsegu 0200h-02FFh (0200h, 0202h, 0204h, ..., 02FEh)
- **Gornja banka:**
 - Aktivirana kada je **A0=1**, što znači da pokriva sve neparne adrese u opsegu 0200h-02FFh (0201h, 0203h, 0205h, ..., 02FFh)



```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```

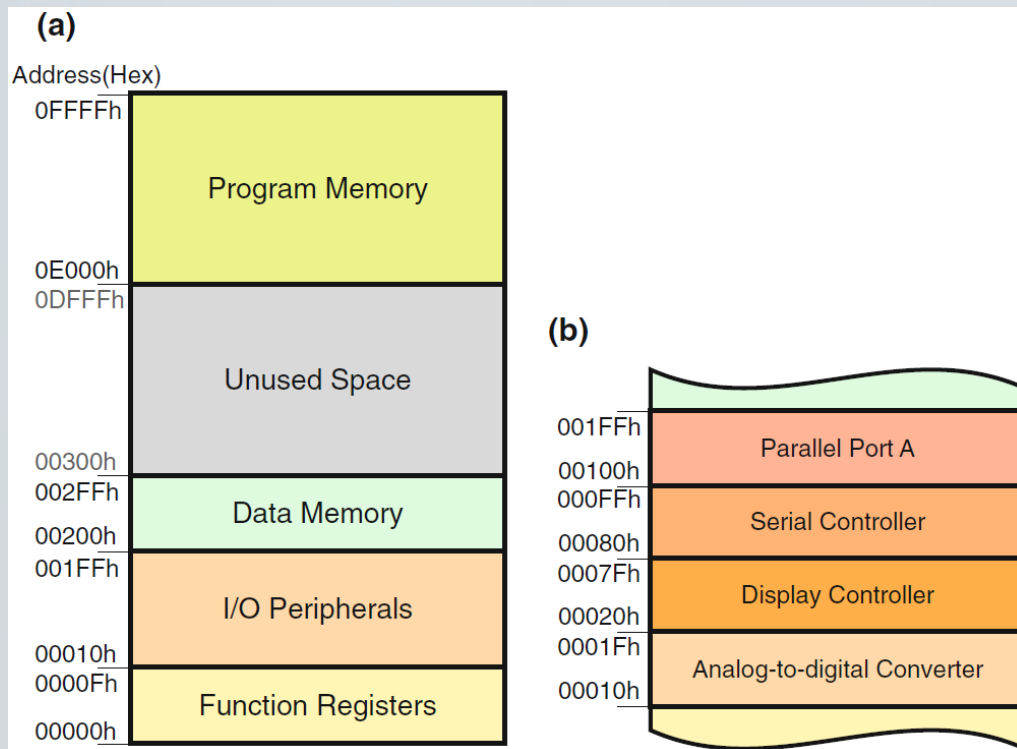
Memorijska mapa I

- **Memorijska mapa** predstavlja model načina korišćenja raspoloživog adresnog prostora unutar mikroprocesorskog sistema
- Predstavlja vrlo **važan podatak** prilikom **planiranja programa** i odabira odgovarajućeg procesora
- U memorijskoj mapi navedeni su **blokovi adresa** koji su adresirani različitim perifernim jedinicama prisutnim **unutar embeded sistema**
- Na slici je prikazan grafički prikaz moguće memorijske mape embeded sistema sa 16-bitnom **adresnom magistralom** i **adresnim prostorom** od 64 KB



Memorijska mapa II

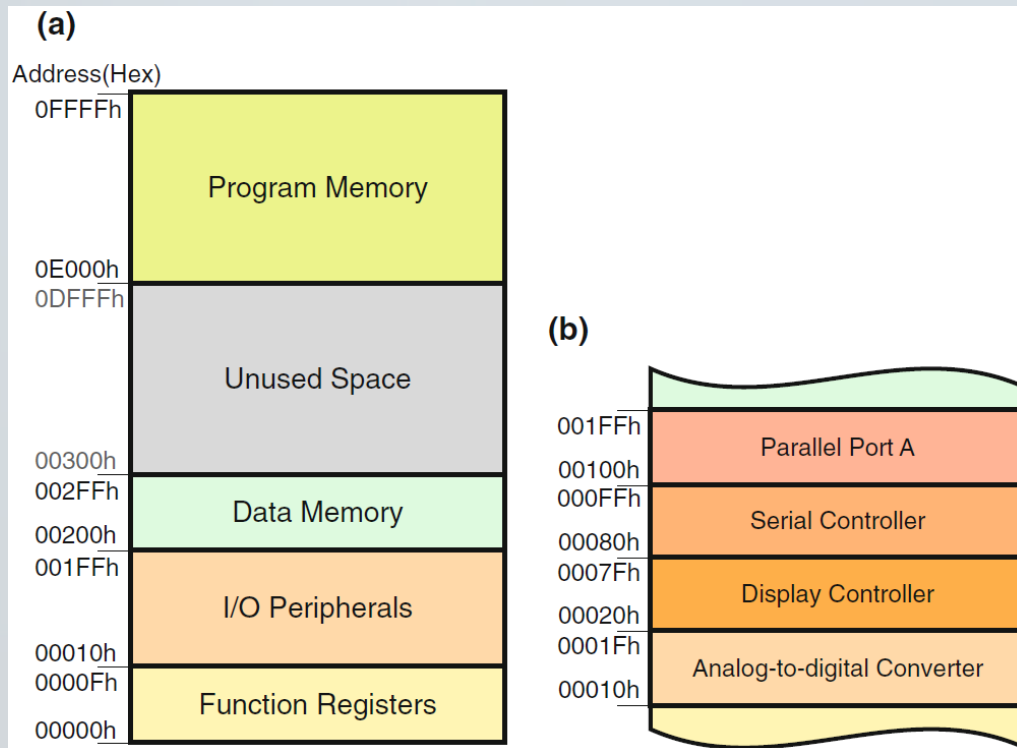
- Ovo je samo jedan od mogućih načina organizovanja memorijskog prostora
- U ovom primeru, prvih **16** memorijskih reči (adrese 0h do 0Fh) alocirano je za smeštanje **funkcijskih registara**
- Sledećih **496** (10h - 1FFh) lokacija alocirano je **ulazno/izlaznim periferijskim jedinicama**
- Narednih **256** lokacija (200h - 2FFh) rezervisano je **za memoriju podataka**
- **Programska memorija**, veličine **8K** lokacija, locirana je na kraj adresnog prostora (0E000h – 0FFFFh)
- Adrese iz opsega od 00300h do 0DFFFh predstavljaju **neiskorišćeni memorijski prostor** i mogu se koristiti u budućim verzijama sistema



```
shift_reg <= unsigned(inp);  
elseif (en == '1') then
```

Memorijska mapa III

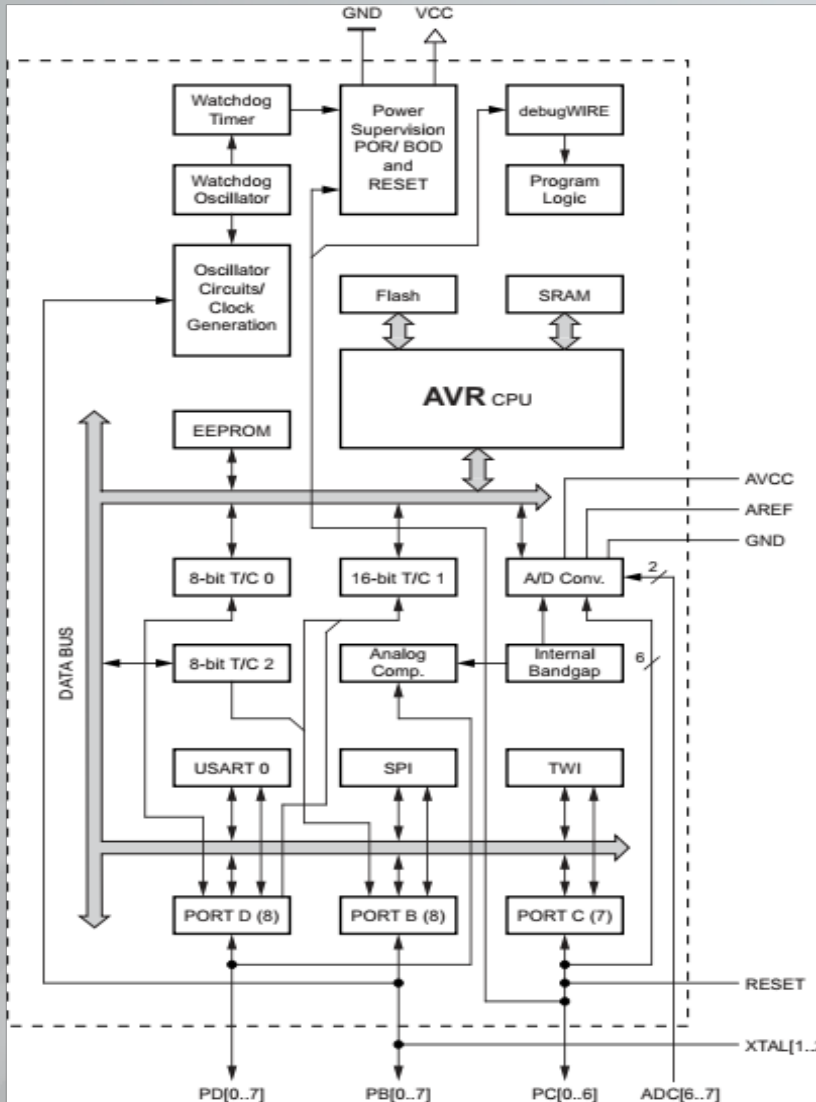
- Memorijske mape mogu biti **globalne** i **parcijalne**
- Globalna memorijska mapa prikazuje strukturu **čitavog adresnog prostora**, kao što je ilustrovano na slici a)
- Parcijalna memorijska mapa pruža **detalje o samo jednom delu adresnog prostora**, omogućavajući mnogo bolji uvid u taj deo globalne mape
- Na primer, na slici b) prikazana je parcijalna mapa moguće alokacije adresa između perifernih jedinica prisutnih u sistemu



```
shift_reg <= unsigned (inp);  
elseif (en == '1') then
```

ATmega328P- memorija

- Harvard arhitektura
- 32 KB programske flash memorije (16K x 16)
- 2 KB SRAM memorije za podatke
- 1 KB EEPROM memorije

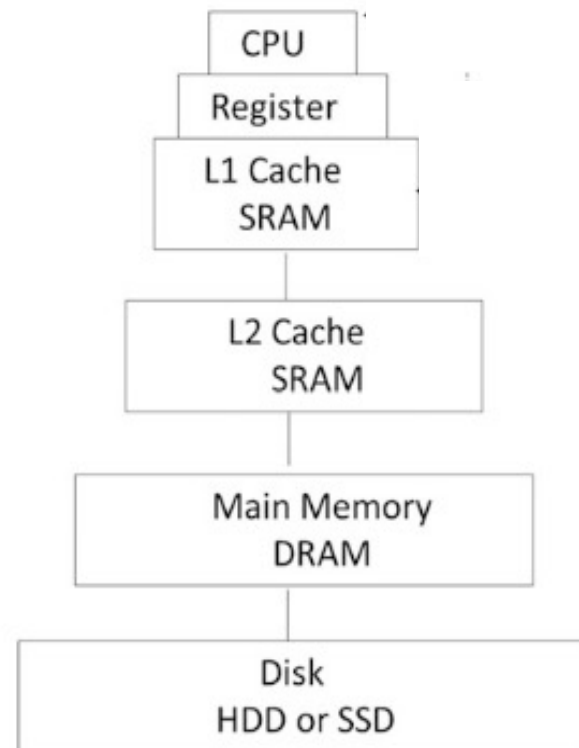


Data Memory	
32 Registers	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F
160 Ext I/O Registers	0x0060 - 0x00FF
Internal SRAM (1024 x 8)	0x0100
	0x08FF

```
shift_reg <= unsigned (inp);  
elseif ( en == '1' ) then
```


Memorijska hijerarhija

- Memorijski podsistem mikroračunari sastoji se iz tri dela: skrivena memorija (*engl. cache*), operativna i masovna memorija koja su u hijerarhiji povezane kao na slici
- Hijerarhija se formira na osnovu razlike u veličini, brzini pristupa, veličini i tip transfera i ceni
- **Keš memorija** (cache memory) – najbrži tip memorije, najčešće unutra CPU, brža je od glavne memorije, samim tim i skuplja. Može biti realizovana u tri nivoa: L1, L2 i L3.
- **Operativna (radna) memorija** – primer: DRAM, program koji se izvršava se prebacuje iz sekundarne u glavnu memoriju.
- **Diskovi (masovna memorija)** – odnosi se na sekundarne memoriju kao što je hard disk (HDD), SSD,...



```
entity test_shift is
  generic ( width : integer := 17 );
  port ( clk : in std_ulogic;
        reset : in std_ulogic;
        load : in std_ulogic;
        en : in std_ulogic;
        outp : out std_ulogic );
end test_shift;
```

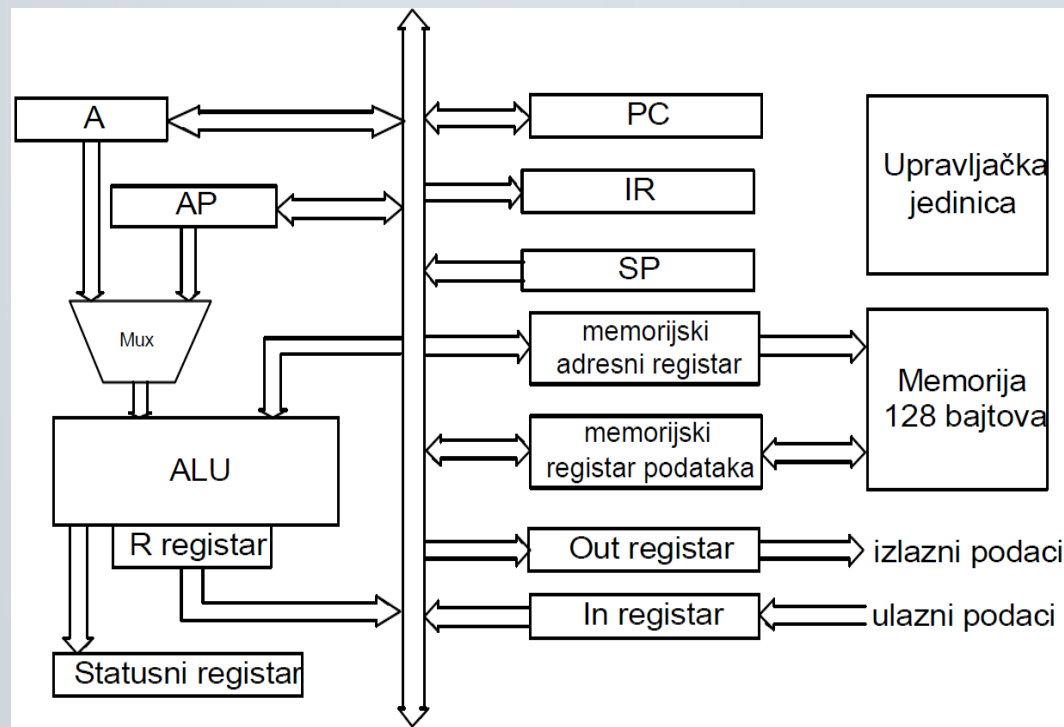
Primer centralnog procesora: Edulent

```
shifter : process (en, reset)
begin
  if (reset = '0') then
    shift_reg <= others => '0';
  elsif rising_edge ( clk ) then
    if (load = '1') then
      shift_reg <= unsigned (inp);
    elsif ( en = '1' ) then
```

Edulent – hardverske karakteristike I

Hardverske karakteristike:

- **Procesor sa von Neumann arhitekturom** – programskoj memoriji i memoriji za podatke pristupa se preko zajedničkih magistrala
- **Skalarni 8-bitni procesor bez protočne obrade** – jedna ALU koja je u stanju da izvodi aritmetičko-logičke operacije nad podacima veličine 1 bajta
- Ima **2 registra opšte namene** (A i AP), **statusni registar** sastoji se iz dva bita: indikator prenosa i indikator nule ; ima hardversku podršku za rad sa stekom
- **BIL modul je vrlo jednostavan** – sastoji se iz dva registra i jednog kontrolnog signala:
 - **MAR** – memorijskog adresnog registra, u kome se smešta **adresa naredne memorijske lokacije** kojoj se želi pristupiti
 - **MRP** – memorijskog registra podataka, u koji se smešta **podatak** koji se želi upisati u memoriju ili je pročitana iz memorije
 - **W/R signala** – pomoću kojega se određuje da li te tekuća memorijska operacija operacija **upisa** (W/R=1) ili **čitanja** (W/R=0)

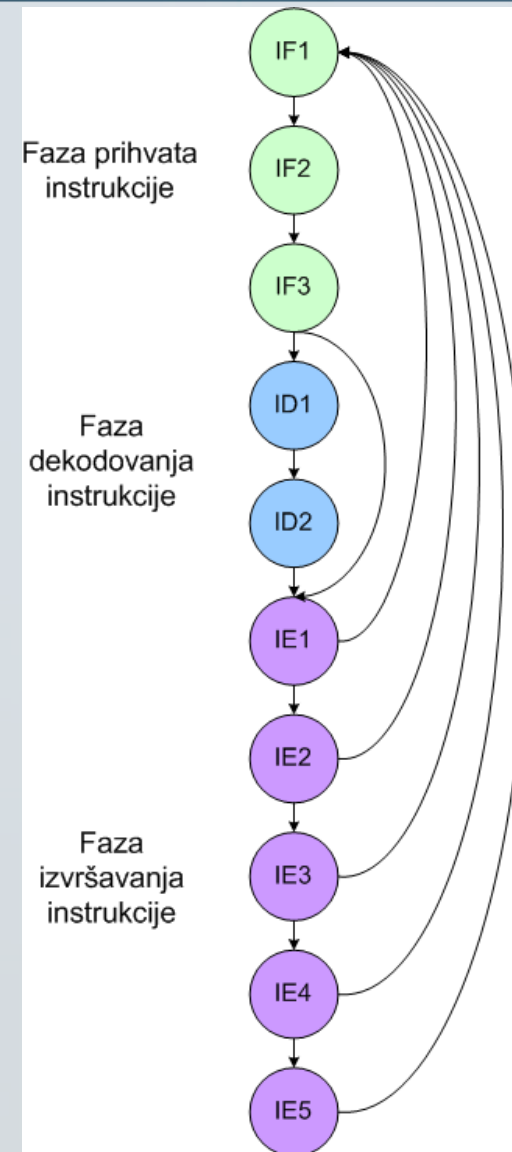


```
shift_reg <= unsigned(inp);  
elsif (en = '1') then
```

Edulent – hardverske karakteristike II

▪ Upravljačka jedinica:

- **Sve instrukcije** izvršavaju se tokom određenog broja **taktova**
- Broj taktova potrebnih za izvršavanje instrukcije varira od minimalno **4 takta** do maksimalno **10 taktova**
- Izvršavanje instrukcija nije realizovano korišćenjem protočne obrade, pre **početka izvršavanja naredne instrukcije tekuća instrukcija mora da se kompletira**
- Moguće je izvršavati **samo jednu instrukciju** u svakom trenutku



```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```

Edulent – softverske karakteristike I

- Softverske karakteristike:
 - Skup od **39 instrukcija** (uključujući različite modove adresiranja)
 - **Instrukcije** su promenljive **dužine**, postoje instrukcije dužine **1 bajt** i instrukcije dužine **2 bajta**
 - **Prvi bajt uvek sadrži kod instrukcije**
 - **Drugi bajt** je opcioni, ukoliko je prisutan sadrži **adresu memorijske lokacije** kojoj je potrebno pristupiti prilikom izvršavanja instrukcije ili vrednost numeričke konstante
 - Postoji ukupno 4 moda adresiranja:
 - **Direktno** – adresa operanda smeštena je u drugom bajtu instrukcije
 - **Neposredno** – vrednost operanda koju je potrebno koristiti smeštena je u drugom bajtu instrukcije
 - **Registarsko indirektno** – adresa operanda smeštena je u AP registar
 - **Predekrement/postinkrement** – koristi se za rad sa stekom; adresa vrha steka smeštena je u SP registru

Format jednobajtnih instrukcija:

Kod instrukcije

Format dvobajtnih instrukcija:

Kod instrukcije

Adresa ili konstanta

```
shift_reg <= unsigned (inp);  
elsif (en = '1') then
```

Edulent – instrukcije prenosa podataka

Sintaksa	Opis	Utiče na indikatore	Kod instrukcije
MOV A, address	Prebacuje sadržaj memorijske lokacije sa adresom address u akumulator	-	0x11
MOV AP, address	Prebacuje sadržaj memorijske lokacije sa adresom address u AP registar	-	0x13
MOV A, const	Akumulator dobija vrednost const	-	0x19
MOV AP, const	AP registar dobija vrednost const	-	0x1B
MOV address, A	Prebacuje sadržaj akumulatora u memorijsku lokaciju sa adresom address	-	0x21
MOV address, AP	Prebacuje sadržaj AP registra u memorijsku lokaciju sa adresom address	-	0x23
MOV A, [AP]	Prebacuje sadržaj memorijske lokacije čija je adresa u AP registru u akumulator	-	0x14
MOV A, [SP+]	Sadržaj vrha steka prebacuje u akumulator i nakon toga inkrementuje sadržaj SP registra	-	0x1C
MOV AP, [SP+]	Sadržaj vrha steka prebacuje u AP registar i nakon toga inkrementuje sadržaj SP registra	-	0x1E
MOV [-SP], A	Dekrementuje sadržaj SP registra i nakon toga sa sadržaj akumulatora upisuje na vrh steka	-	0x2C
MOV [-SP], AP	Dekrementuje sadržaj SP registra i nakon toga sa sadržaj AP registra upisuje na vrh steka	-	0x2E

Edulent – aritmetičke instrukcije

Sintaksa	Opis	Utiče na indikatore	Kod instrukcije
ADD A, address	Sabira sadržaj akumulatora sa sadržajem memorijske lokacije, čija adresa je address , i rezultat smešta u akumulator	prenosa, nule	0x31
ADD A, const	Sabira sadržaj akumulatora sa konstantom const , i rezultat smešta u akumulator	prenosa, nule	0x39
ADD AP, const	Sabira sadržaj AP registra sa konstantom const , i rezultat smešta u AP registar	prenosa, nule	0x3B
ADD A, [AP]	Sabira sadržaj akumulatora sa sadržajem memorijske lokacije, čija adresa je u AP registru, i rezultat smešta u akumulator	prenosa, nule	0x34
SUB A, address	Oduzima sadržaj memorijske lokacije, čija adresa je address , od sadržaja akumulatora i rezultat smešta u akumulator	prenosa, nule	0x41
SUB A, const	Oduzima konstantu const , od sadržaja akumulatora i rezultat smešta u akumulator	prenosa, nule	0x49
SUB AP, const	Oduzima konstantu const , od sadržaja AP registra i rezultat smešta u AP registar	prenosa, nule	0x4B
SUB A, [AP]	Oduzima sadržaj memorijske lokacije, čija adresa je u AP registru, od sadržaja akumulatora i rezultat smešta u akumulator	prenosa, nule	0x44

```
shift_reg <= unsigned (inp);  
elseif (en == '1') then
```


Edulent – logičke instrukcije

Sintaksa	Opis	Utiče na indikatore	Kod instrukcije
NOT	Negacija vrednosti akumulatora	nule	0x50
OR address	Logičko ILI sadržaja akumulatora i memorijske lokacije čija adresa je address	nule	0x61
OR const	Logičko ILI sadržaja akumulatora i konstante const	nule	0x69
OR [AP]	Logičko ILI sadržaja akumulatora i memorijske lokacije čija adresa je smeštena u AP registru	nule	0x64
AND address	Logičko I sadržaja akumulatora i memorijske lokacije čija adresa je address	nule	0x71
AND const	Logičko I sadržaja akumulatora i konstante const	nule	0x79
AND [AP]	Logičko I sadržaja akumulatora i memorijske lokacije čija adresa je smeštena u AP registru	nule	0x74
XOR address	Logičko ekskluzivno ILI sadržaja akumulatora i memorijske lokacije čija adresa je address	nule	0x81
XOR const	Logičko ekskluzivno ILI sadržaja akumulatora i konstante const	nule	0x89
XOR [AP]	Logičko ekskluzivno ILI sadržaja akumulatora i memorijske lokacije čija adresa je smeštena u AP registru	nule	0x84
SHR	Pomeranje sadržaja akumulatora za jedan bit u desno, pri čemu bit odlazi u indikator prenosa	prenosa, nule	0x90

```
shift_reg <= unsigned (inp);
elsif (en = '1') then
```

Edulent – instrukcije grananja

Sintaksa	Opis	Utiče na indikatore	Kod instrukcije
JMP label	Bezuslovni skok na adresu instrukcije na koju pokazuje labela label	-	0xA1
JC label	Skok na adresu instrukcije na koju pokazuje labela label ukoliko je vrednost indikatora prenosa jednaka jedinici	-	0xA9
JZ label	Skok na adresu instrukcije na koju pokazuje labela label ukoliko je vrednost indikatora nule jednaka jedinici	-	0xA5

```
shift_reg <= unsigned (inp);  
elsif ( en == '1' ) then
```

Edulent – instrukcije za rad sa procedurama

Sintaksa	Opis	Utiče na indikatore	Kod instrukcije
CALL procedure	Prenos toka izvršavanja programa na prvu instrukciju potprograma procedure	-	0xC1
RET	Povratak iz potprograma na mesto poziva potprograma	-	0xB0

```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```

Edulent – instrukcije za rad sa periferijama

Sintaksa	Opis	Utiče na indikatore	Kod instrukcije
IN	Prebacuje sadržaj IN registra u akumulator	-	0xD0
OUT	Prebacuje sadržaj akumulatora u OUT registar	-	0xE0

```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```

Edulent – instrukcije programske kontrole

Sintaksa	Opis	Utiče na indikatore	Kod instrukcije
NOP	Nema operacije, prelazi se na sledeću instrukciju	-	0x00
END	Označava kraj programa	-	0x02

```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```

Edulent – primer asemblerskog i mašinskog programa

▪ Asemblerski program

PROGRAM "Instrukcije prenosa podataka"

DATA

VAR1 DB 0X9

VAR2 DB 0XF

RES1 DB 0X0

RES2 DB 0X0

RES3 DB 0X0

ENDDATA

CODE

MOV A, 0x1 ➔ 0x19 0x01

MOV A, VAR1 ➔ 0x11 0x12

MOV AP, @VAR1 ➔ 0x1B 0x12

MOV A, [AP] ➔ 0x14

MOV AP, 0XAA ➔ 0x1B 0xAA

MOV RES1, AP ➔ 0x23 0x14

MOV AP, VAR2 ➔ 0x13 0x13

MOV RES2, AP ➔ 0x23 0x15

MOV RES3, A ➔ 0x21 0x16

END ➔ 0x02

ENDPROGRAM

▪ Mašinski program

▪ Sadržaj memorije

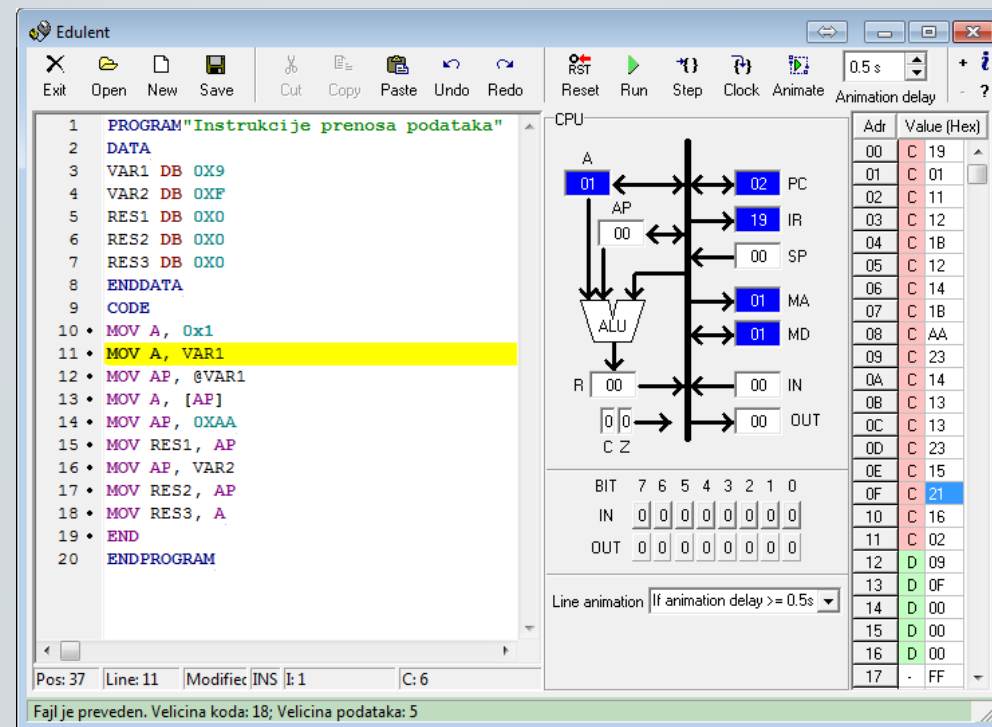
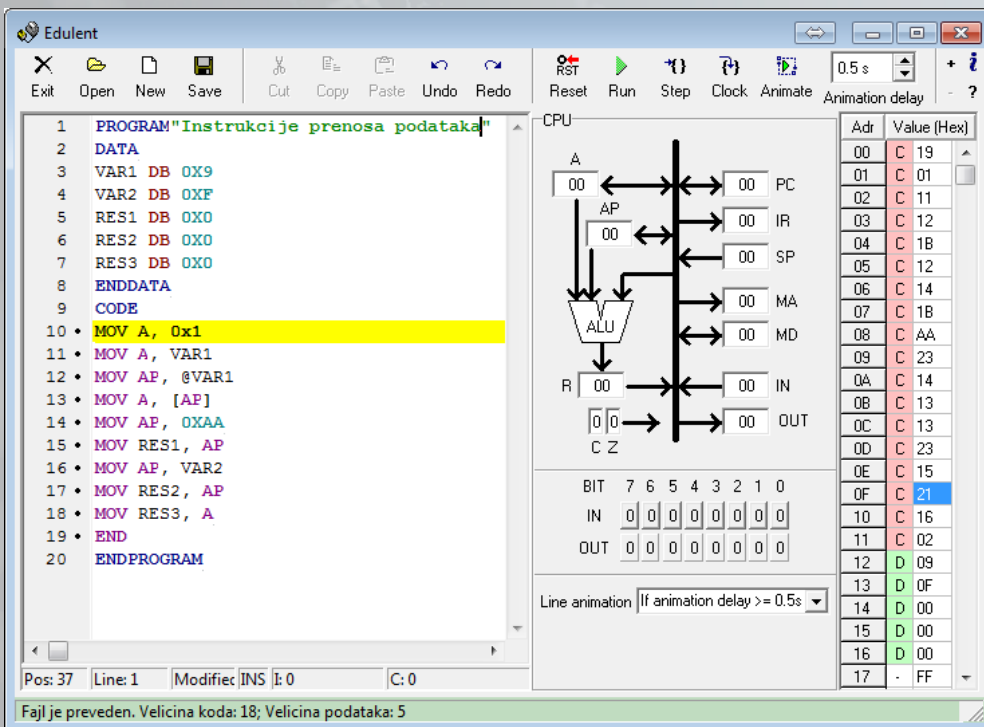
Adr	Value (Hex)
00	C 19
01	C 01
02	C 11
03	C 12
04	C 1B
05	C 12
06	C 14
07	C 1B
08	C AA
09	C 23
0A	C 14
0B	C 13
0C	C 13
0D	C 23
0E	C 15
0F	C 21
10	C 16
11	C 02
12	D 09
13	D 0F
14	D 00
15	D 00
16	D 00
17	- FF

```
shift_reg <= unsigned (inp);  
elseif (en == '1') then
```

Edulent – izvršavanje programa I

- Početno stanje procesora

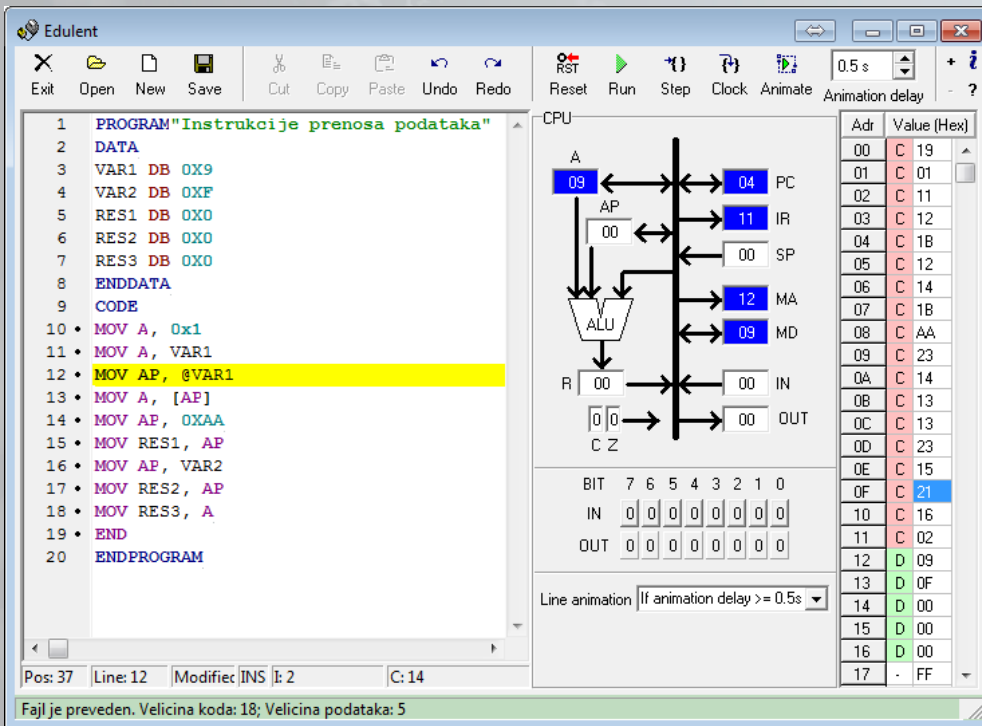
- Stanje procesora nakon izvršavanja instrukcije `MOV A, 0x1`. **0x19 0x01**



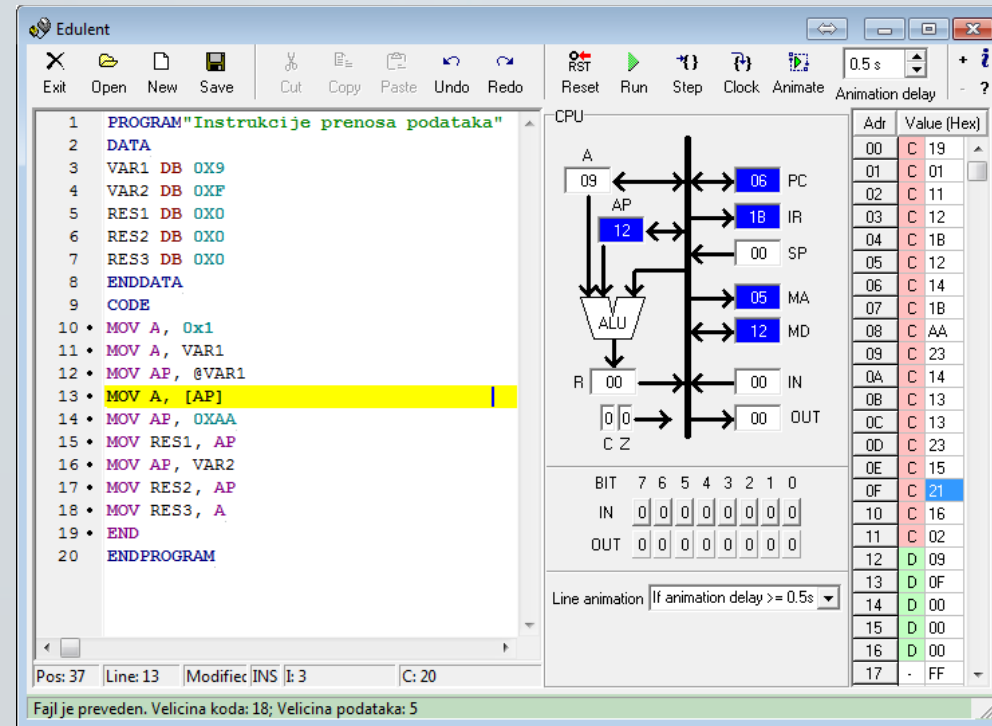
```
shift_reg <= unsigned(inp);
elsif (en == '1') then
```


Edulent – izvršavanje programa II

- Stanje procesora nakon izvršavanja instrukcije MOV A, VAR1
- 0x11 0x12

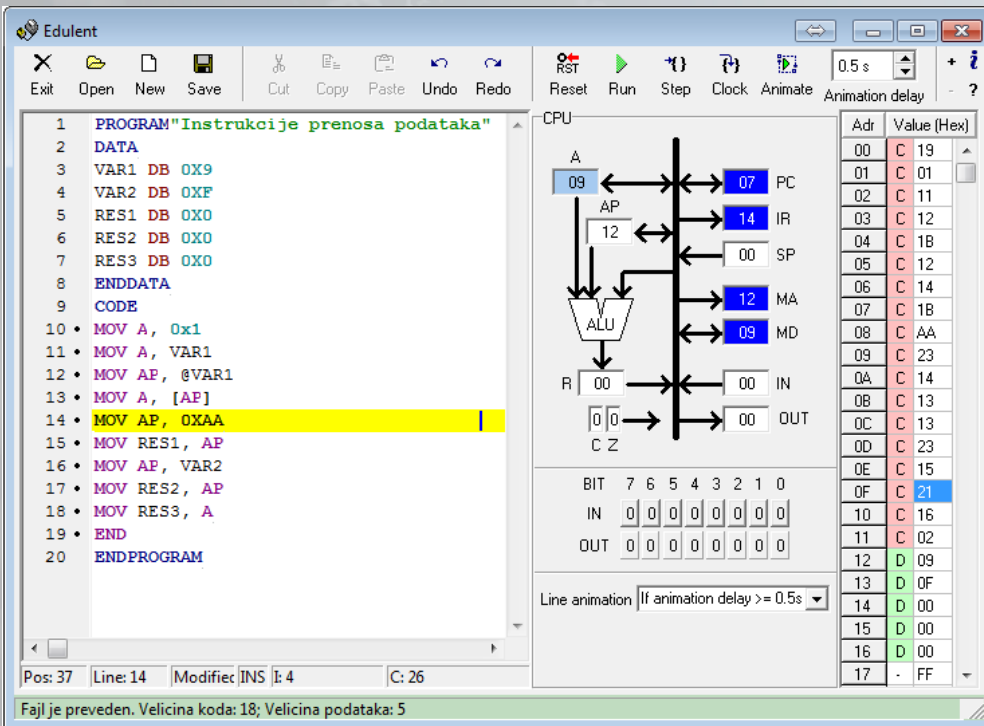


- Stanje procesora nakon izvršavanja instrukcije MOV AP, @VAR1
- 0x1B 0x12

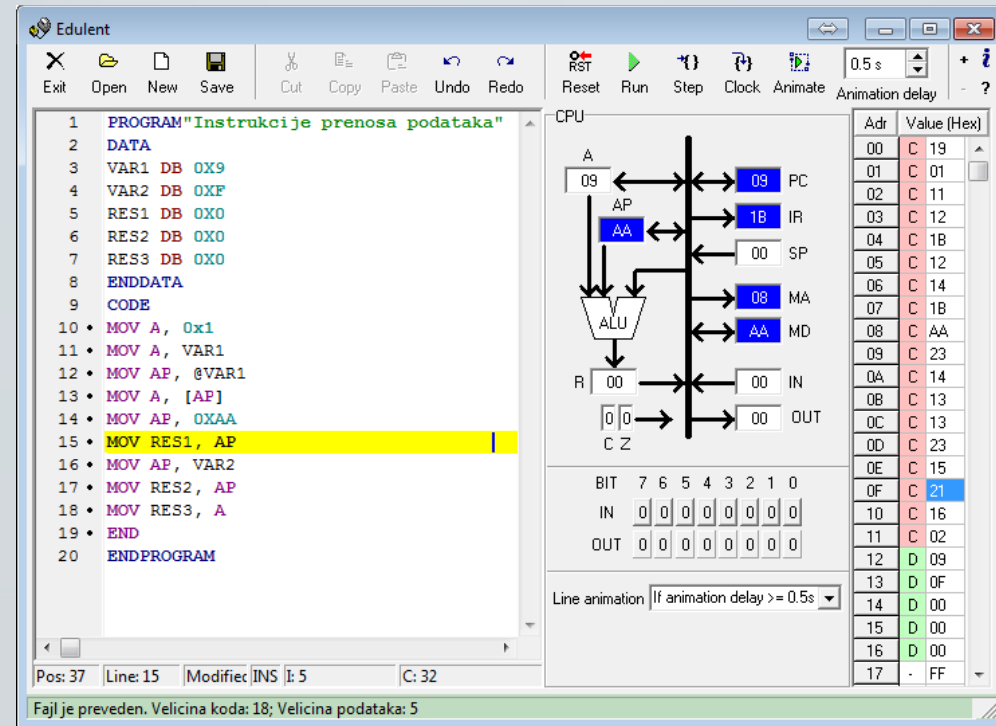


Edulent – izvršavanje programa III

- Stanje procesora nakon izvršavanja instrukcije MOV A, [AP]
- 0x14

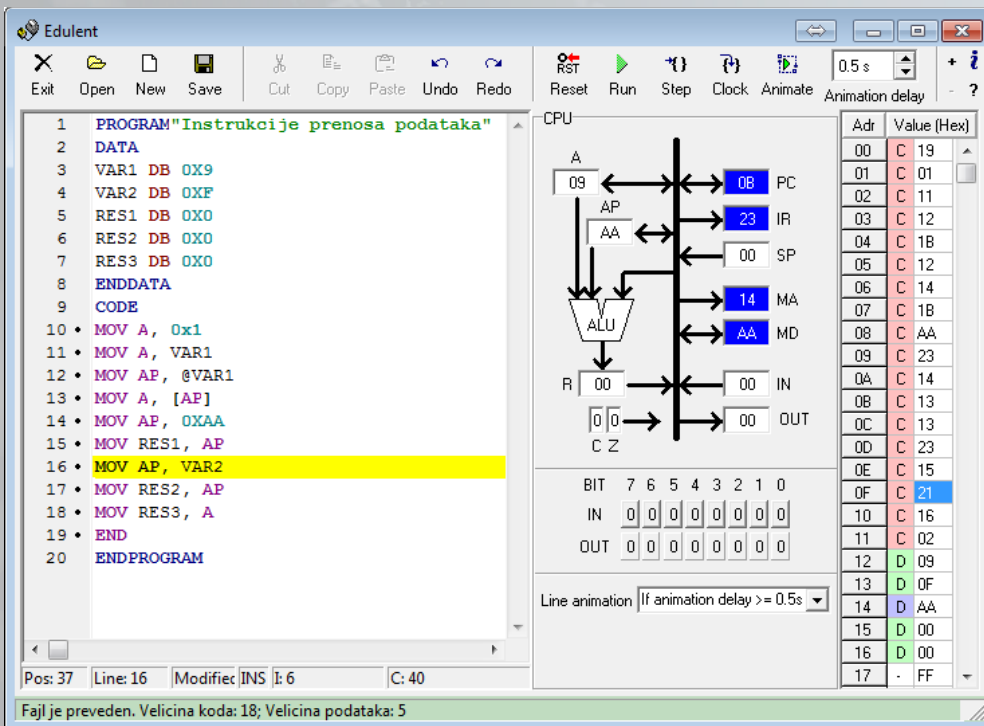


- Stanje procesora nakon izvršavanja instrukcije MOV AP, 0xAA
- 0x1B 0xAA

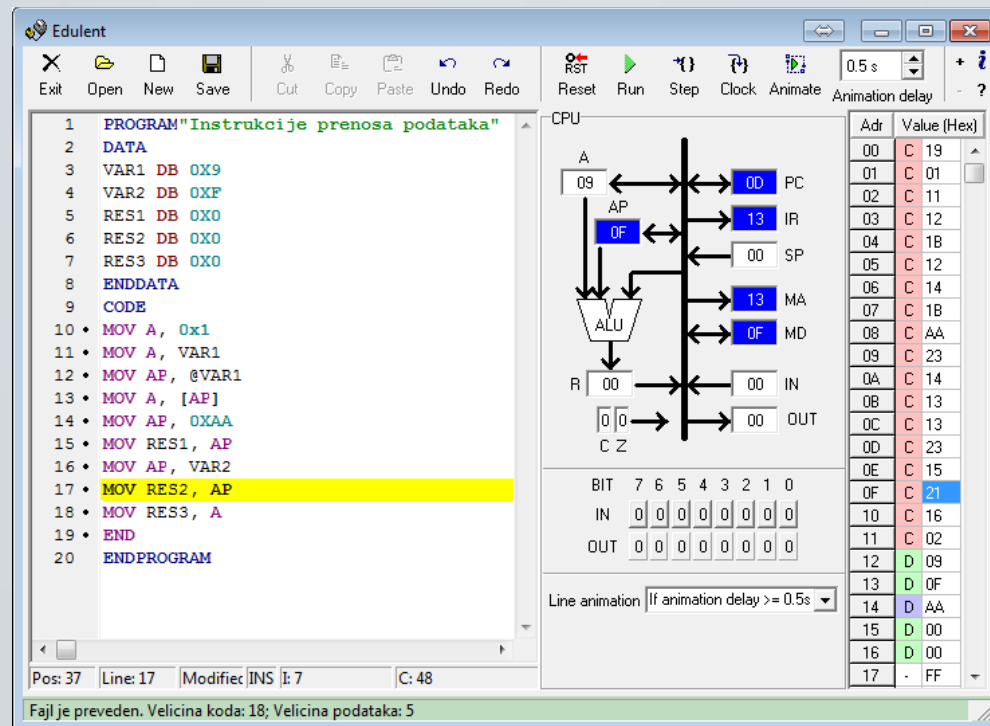


Edulent – izvršavanje programa IV

- Stanje procesora nakon izvršavanja instrukcije MOV RES1, AP
- 0x23 0x14

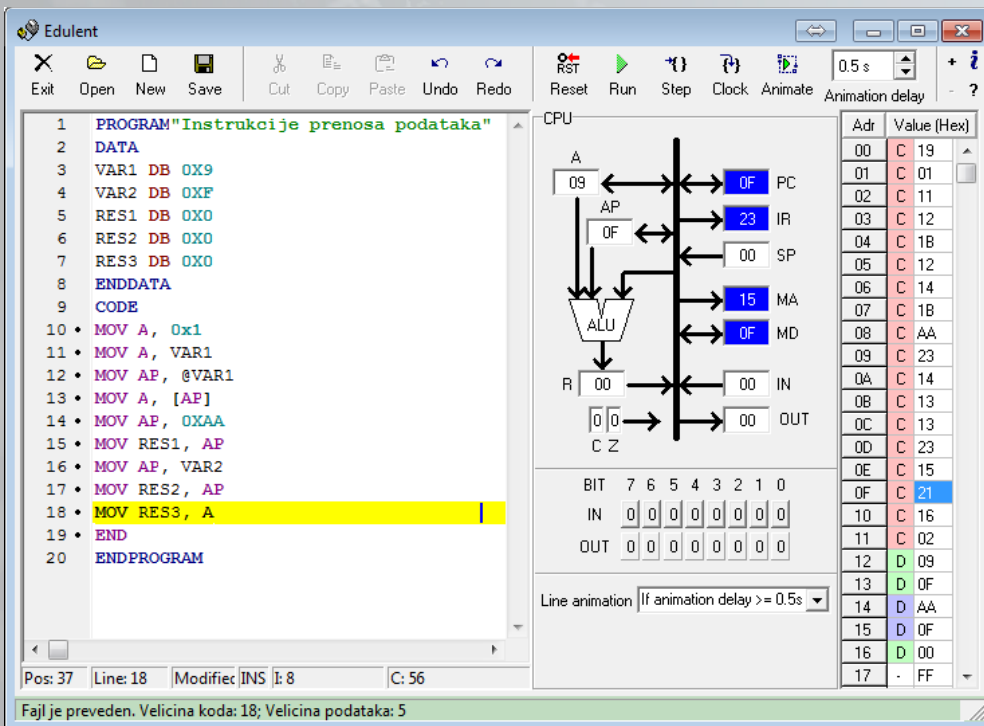


- Stanje procesora nakon izvršavanja instrukcije MOV AP, VAR2
- 0x13 0x13

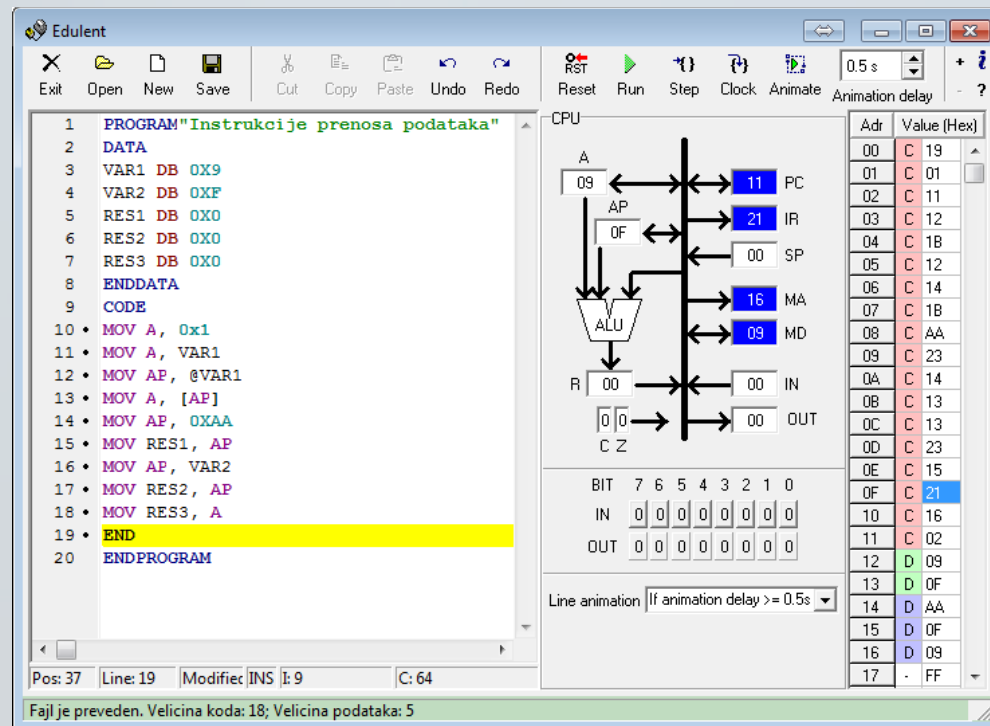


Edulent – izvršavanje programa V

- Stanje procesora nakon izvršavanja instrukcije MOV RES2, AP
- 0x23 0x15



- Stanje procesora nakon izvršavanja instrukcije MOV RES3, A
- 0x21 0x16



Edulent – izvršavanje programa VI

- Stanje procesora nakon izvršavanja instrukcije END
- 0x02

The screenshot shows the Edulent software interface. The main window displays the assembly code for a program named "Instrukcije prenosa podataka". The code is as follows:

```
1 PROGRAM "Instrukcije prenosa podataka"
2 DATA
3 VAR1 DB 0x9
4 VAR2 DB 0xF
5 RES1 DB 0x0
6 RES2 DB 0x0
7 RES3 DB 0x0
8 ENDDATA
9 CODE
10 • MOV A, 0x1
11 • MOV A, VAR1
12 • MOV AP, @VAR1
13 • MOV A, [AP]
14 • MOV AP, 0x0A
15 • MOV RES1, AP
16 • MOV AP, VAR2
17 • MOV RES2, AP
18 • MOV RES3, A
19 • END
20 ENDPROGRAM
```

The CPU state is shown on the right side of the interface. The CPU registers and their values are:

Register	Value (Hex)
PC	19
IR	02
SP	00
MA	11
MD	02
IN	00
OUT	00

The status bar at the bottom indicates "Kraj programa - CPU zaustavljen".

```
shift_reg <= unsigned (inp);
elseif ( en == '1' ) then
```

Edulent – izvršavanje instrukcije MOV A, 0x01 I

- Potrebne mikrooperacije i redosled njihovog izvršavanja kako bi se kompletirala instrukcija

MOV A, 0x01

Faza	Korak	Mikrooperacije
Prihvat instrukcije	IF1	$MA \leftarrow PC$
	IF2	$MD \leftarrow M[MA]$ $PC \leftarrow PC+1$
	IF3	$IR \leftarrow MD$
Dekodovanje instrukcije	ID1	$MA \leftarrow PC$
	ID2	$MD \leftarrow M[MA]$ $PC \leftarrow PC+1$
Izvršavanje instrukcije	IE1	$A \leftarrow MD$

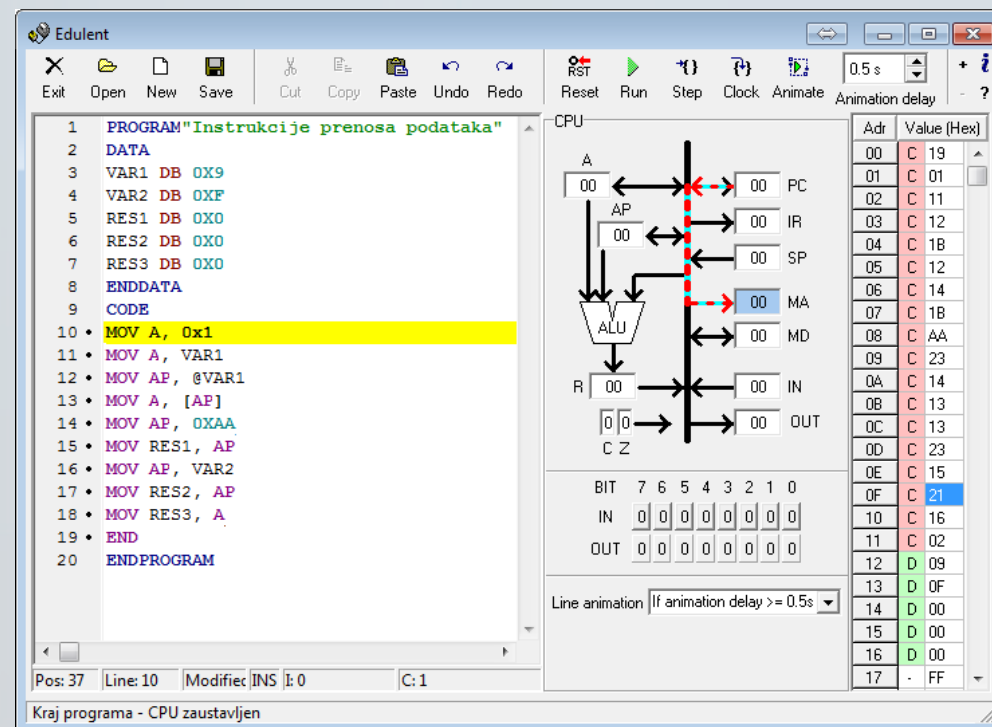
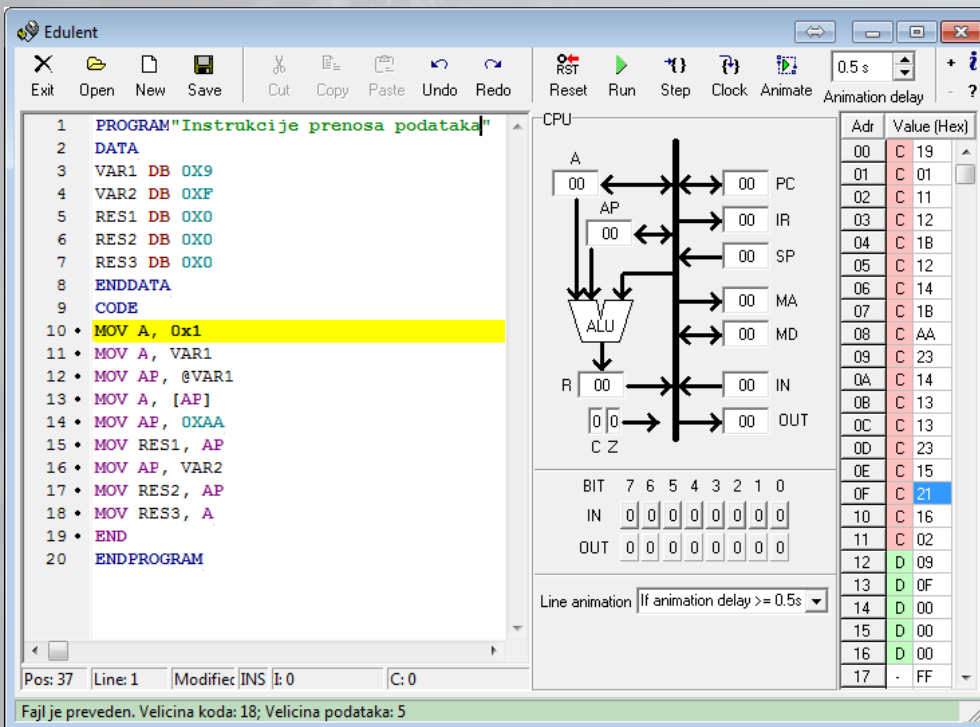
```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```

Edulent – izvršavanje instrukcije MOV A, 0x01 II

- Početno stanje procesora

- Stanje procesora nakon izvršavanja mikrooperacija iz faze IF1:

$MA \leftarrow PC$

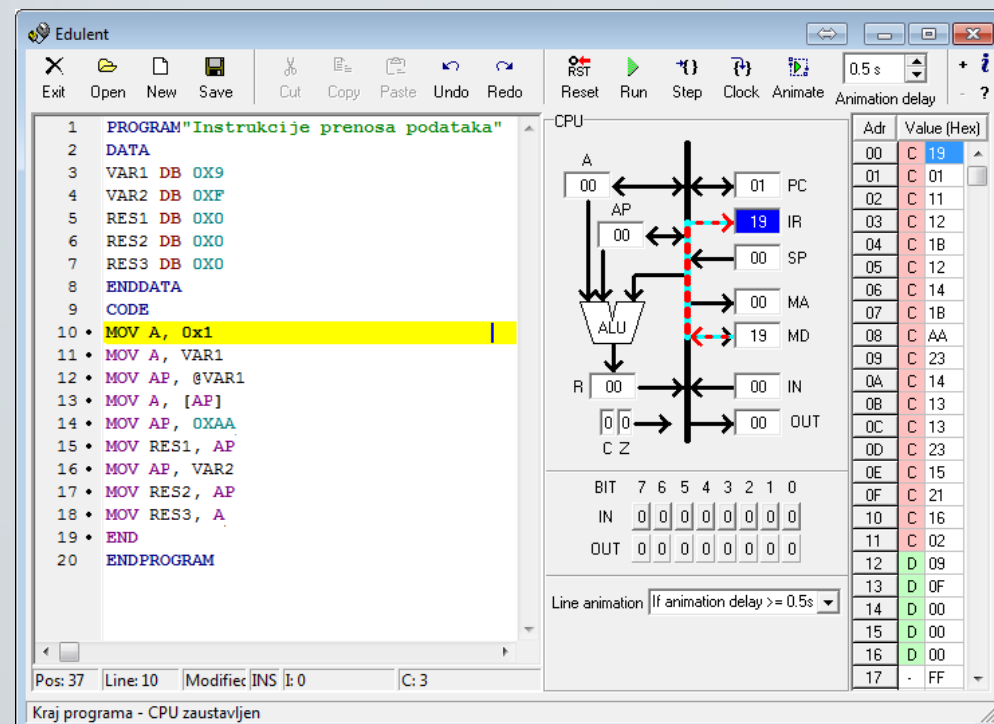
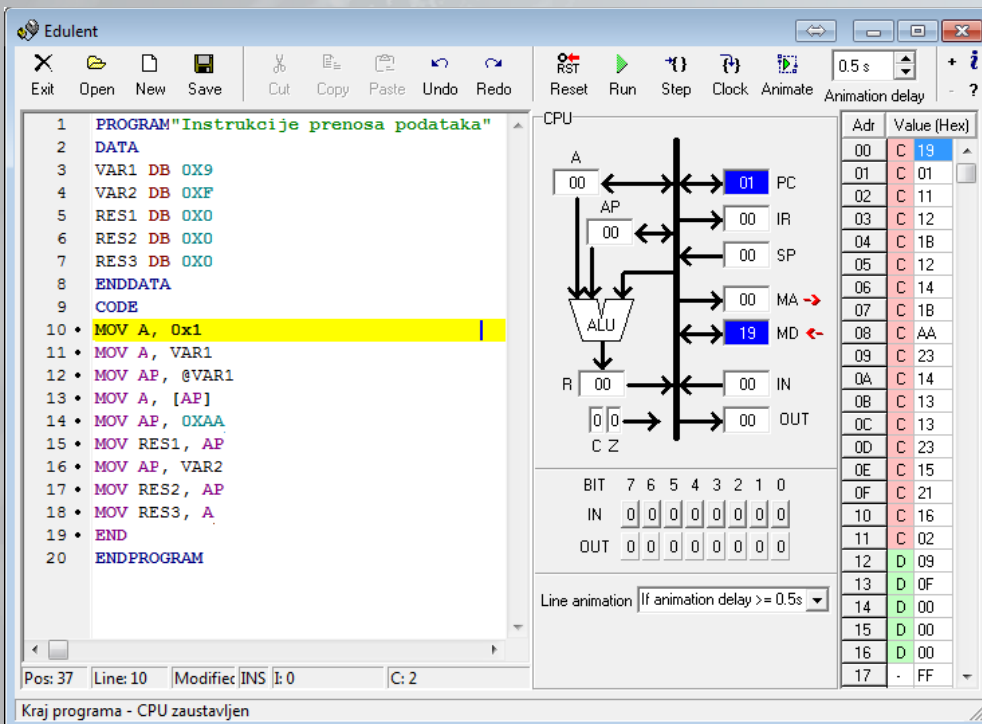


Edulent – izvršavanje instrukcije MOV A, 0x01 III

- Stanje procesora nakon izvršavanja mikrooperacija iz faze IF2:

$$MD \leftarrow M[MA]$$
$$PC \leftarrow PC+1$$

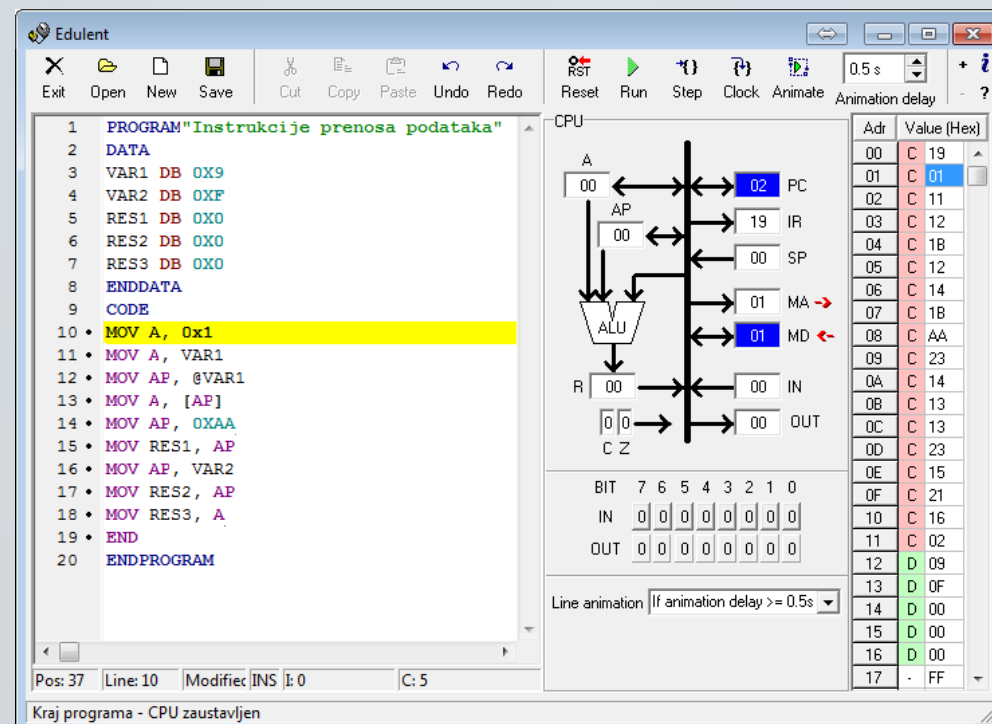
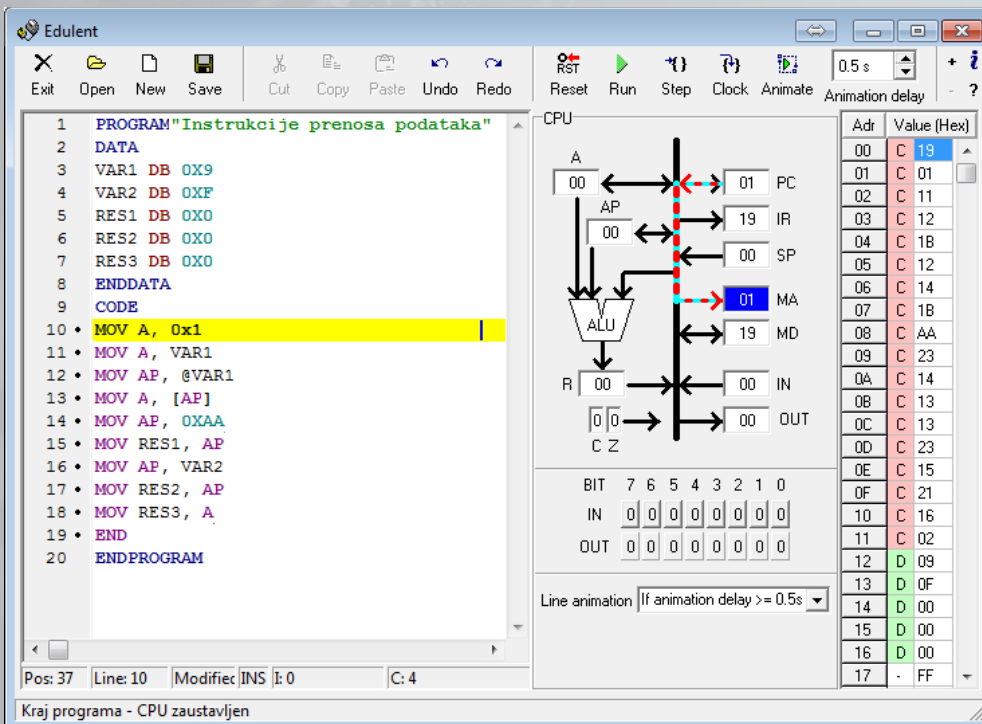
- Stanje procesora nakon izvršavanja mikrooperacija iz faze IF3:

$$IR \leftarrow MD$$


Edulent – izvršavanje instrukcije MOV A, 0x01 IV

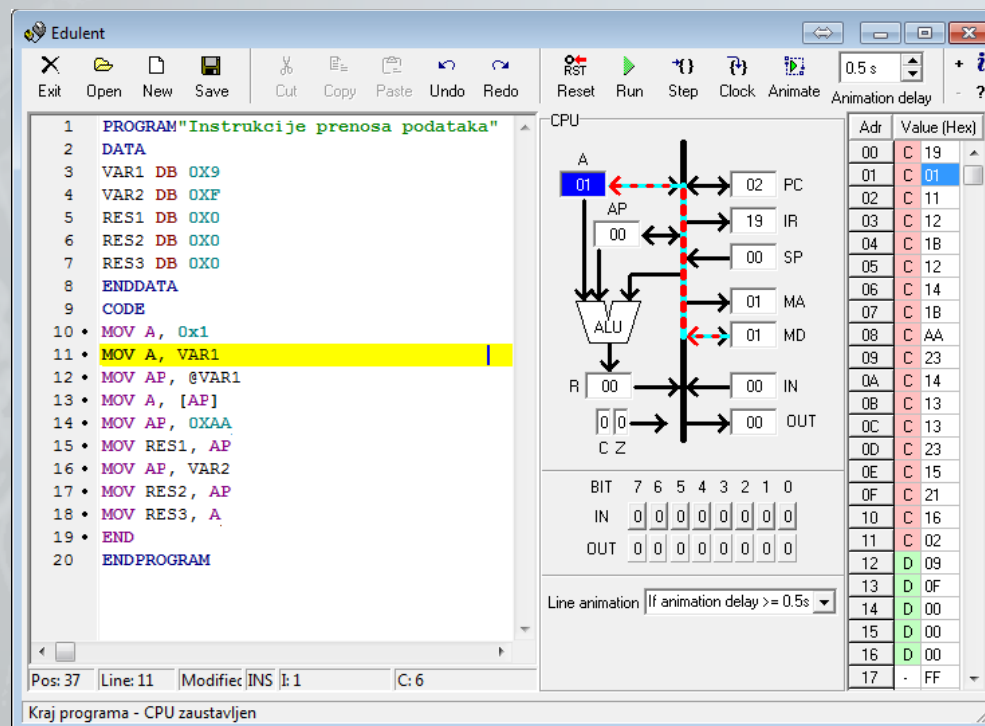
- Stanje procesora nakon izvršavanja mikrooperacija iz faze ID1:
 $MA \leftarrow PC$

- Stanje procesora nakon izvršavanja mikrooperacija iz faze ID2:
 $MD \leftarrow M[MA]$
 $PC \leftarrow PC+1$



Edulent – izvršavanje instrukcije MOV A, 0x01 V

- Stanje procesora nakon izvršavanja mikrooperacija iz faze IE1:
 $A \leftarrow MD$



```
entity test_shift is
  generic ( width : integer := 17 );
  port ( clk : in std_ulogic;
        reset : in std_ulogic;
        load : in std_ulogic;
        en : in std_ulogic;
        outp : out std_ulogic );
end test_shift;
```

Teorijska pitanja P5

```
shifter : process (en, reset)
begin
  if ( reset = '0' ) then
    shift_reg <= (others => '0');
  elsif rising_edge ( clk ) then
    if (load = '1' ) then
      shift_reg <= unsigned (inp);
    elsif ( en = '1' ) then
```

Predavanja 5- Organizacija mikroračunarskog sistema 3

Spisak teorijskih pitanja uz Predavanja 5

1. Organizacija memorije
2. Vrste memorija koje se koriste u embeded sistemima
3. Programska i memorija podataka
4. Von Neumann i Harvard arhitekture
5. Povezivanje procesora i memorije. Ilustrovati način povezivanja na jednom primeru.
6. Memorijska mapa embeded Sistema
7. Memorijske mikrokontrolera ATmega328P
8. Memorijska hijerarhija

```
shift_reg <= unsigned (inp);  
elsif ( en = '1' ) then
```

```
entity test_shift is  
  generic ( width : integer := 17 )  
end entity;
```



```
  shift_reg <= unsigned (inp);  
  elsif ( en = '1' ) then
```