Školski mikroračunar Edulent

arhitektura i programiranje u mašinskom jeziku

Novi Sad, 2018.

Sadržaj

REGISTRI MIKROPROCESORA EDULENT

Svi registri mikroprocesora Edulent su 8 bitni. Mikroprocesor ima 11 registara. Samo su dva registra direktno programski dostupna korisniku:

1) *akumulator(A)* – to je registar opšte namene koji se koristi u najvećem broju instrukcija. Njegov izlaz se vodi na ulaz ALU jedinice, a isto tako i direktno na 8 bitnu magistralu.
2) *adresni pokazivač(AP)* – ovaj registar se koristi za smeštanje adrese pri registarskom indirektnom adresiranju, kao i kod direktnog i neposrednog adresiranja za ADD, SUB i MOV instrukcije. Ovaj registar je takođe povezan i sa ALU i sa magistralom.

Dva registra služe kao pomoćni registri pri komunikaciji sa periferijskim uređajima:

3) *izlazni registar (OUT)* – služi za slanje podataka prema periferijskoj jedinici sa kojom je povezan
4) *ulazni registar (IN)* – služi za prihvat podataka od strane periferijske jedinice sa kojom je povezan

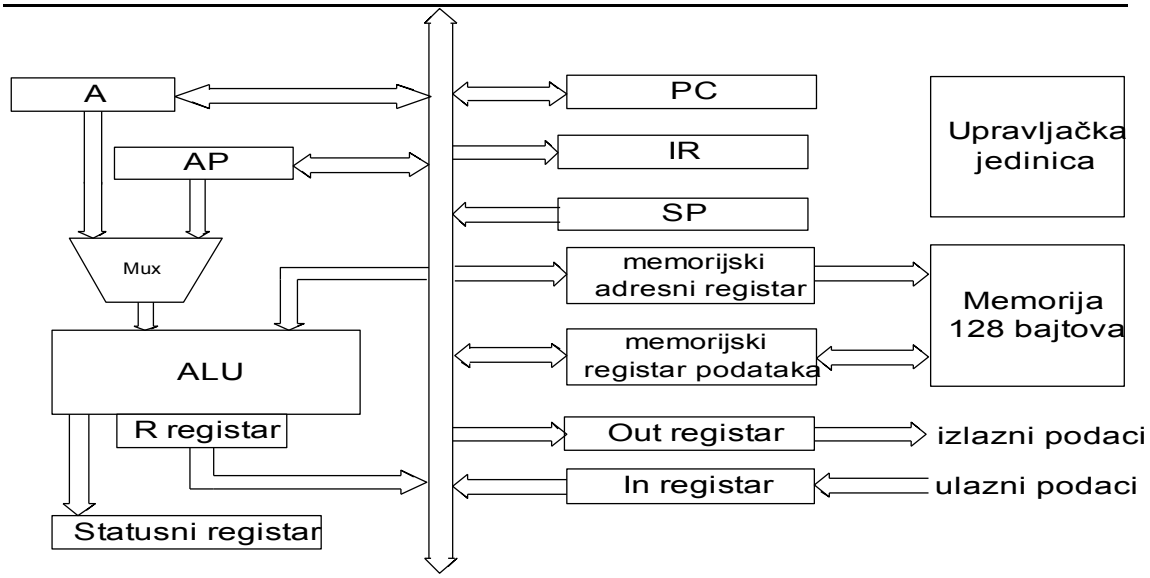
Registri koji imaju uticaja na tok programa, ali im korisnik ne može direktno pristupiti su:

5) *pokazivač steka (SP)* – služi da pokazuje na vrh stek memorije koja je inicijalno prazna i popunjava se od poslednje memorijske lokacije (0x7F) unazad.
6) *programski brojač (PC)* – pokazuje na memorijsku lokaciju sa koje je učitana instrukcija, adresa ili konstanta.
7) *statusni registar* – sadrži 2 indikatora(*flag*): indikator prenosa i indikator nule koji se postavljaju ukoliko ima prenosa odnosno ukoliko je rezultat aritmetičko-logičke operacije jednak nuli.

Ostali registri su korisniku sa programerske tačke gledišta nevidljivi:

8) *instrukcijski registar(IR)* – koristi se za prihvat instrukcija iz memorije i koristi se kod svih instrukcija.
9) *registar rezultata (R)* – služi za privremeno smeštanje rezultata iz ALU jedinice.
10) *memorijski adresni registar (MA)* – je direktno povezan na adresne linije memorije i služi za adresiranje memorijskih lokacija
11) *memorijski registar podataka (MD)* – služi za prihvat podataka iz memorije ili za prihvat podataka koji treba da se upišu u memoriju.

Sledeća slika ilustruje registre mikroprocesora i njihovo povezivanje na magistralu. Na slici se vidi i upravljačka jedinica koja je realizovana *mikroprogramski*.



Slika 1

Pored toga na slici se vidi da je memorija veličine 128 bajta. U memoriju se smeštaju programski kod, varijable i stek memorija.

INSTRUKCIJE MIKROPROCESORA EDULENT

Načini adresiranja

Mikroprocesor Edulent podržava 4 načina adresiranja:

- direktno,
- neposredno,
- registarsko indirektno, i
- predekrement/postinkrement adresiranje.

Direktno adresiranje se koristi kod instrukcija prenosa podataka iz memorije u A ili AP registar (npr.: MOV A, VAR1) i instrukcija prenosa podataka iz A ili AP registra u memoriju (npr.: MOV RES1, AP), kod aritmetičko logičkih operacija gde se jedan operand nalazi u akumulatoru, a drugi operand se nalazi u memoriji.

Neposredno adresiranje se koristi kod instrukcija prenosa podataka gde se A ili AP registar pune konstantom, kod aritmetičko-logičkih operacija gde je jedan operand u akumulatoru, a drugi je konstanta, kod instrukcija grananja i kod poziva podprocedura.

Registarsko indirektno adresiranje podrazumeva da se adresa nalazi u AP registru i koristi se kod operacija prenosa podataka iz memorije u akumulator, kod aritmetičko-logičkih operacija gde je jedan operand u akumulatoru, a drugi se nalazi u memoriji na lokaciji koju pokazuje AP registar.

Predekrement/postinkrement adresiranje se koristi kada se sadržaj A ili AP registra postavlja na vrh steka, odnosno kada se sadržaj vrha steka postavlja u A ili AP registar.

Obratiti pažnju da je za izvršenje instrukcija koje koriste neposredno i direktno adresiranje potrebno **dve** osmobitne memorijske lokacije (2 bajta), jer se u drugom bajtu nalazi operand (adresa ili konstanta). Za instrukcije koje koriste registarsko indirektno ili predekrement/postinkrement adresiranje je dovoljan jedan bajt.

Skup instrukcija

Edulent ima ukupno 39 instrukcija (uključujući različite načine adresiranja) i mogu da se podele na sledeće grupe:

- instrukcije prenosa podataka,
- aritmetičke,
- logičke,
- instrukcije grananja,
- instrukcije za rad sa podprocedurama,
- ulazno/izlazne instrukcije, i
- instrukcije programske kontrole.

Instrukcije prenosa podataka su sledeće:

Sintaksa	**Opis**	**Primeri upotrebe**	**Utiče na indikator**
`MOV A, address`	prebacuje sadržaj memorijske lokacije u akumulator; `address` je konkretna memorijska lokacija definisana u DATA delu programa	`MOV A, VAR1` `MOV A, RES`	-
`MOV AP, address`	prebacuje sadržaj memorijske lokacije u AP registar; `address` je konkretna memorijska lokacija definisana u DATA delu programa	`MOV AP, P1` `MOV AP, VAR1`	-
`MOV A, const`	Akumulator dobija vrednost `const`	`MOV A, 1` `MOV A, 0X7F`	-

`MOV AP, const`	AP registar dobija vrednost `const`; konstanta može biti i adresa konkretne lokacije definisane u DATA delu (koristi se @ znak pre naziva konkretne lokacije; vidi primer)	`MOV AP, 0xA`
		`MOV AP, 2`
		`MOV AP, @VAR1`
			-

`MOV address, A`	Sadržaj akumulatora se prebacuje u konkretnu memorijsku lokaciju; `address` je konretna meorijska lokacija definisana u DATA delu programa u koju se vrši prenos	`MOV RES1, A` `MOV P1, A`	-
`MOV address, AP`	Sadržaj AP registra se prebacuje u konkretnu memorijsku lokaciju; `address` je konretna memorijska lokacija definisana u DATA delu programa u koju se vrši prenos	`MOV RES1, AP` `MOV P1, AP`	-
`MOV A, [AP]`	Sadržaj memorijske lokacije čija se adresa nalazi u AP se prebacuje u akumulator		-
`MOV A, [SP+]`	Sadržaj vrha steka se prebacuje u akumulator i potom se vrši inkrement pokazivača steka		-
`MOV AP, [SP+]`	Sadržaj vrha steka se prebacuje u AP registar i potom se vrši inkrement pokazivača steka		-
`MOV [-SP], A`	Prvo se vrši dekrement pokazivača steka i potom se sadržaj akumulatora prebacuje na vrh steka		-
`MOV [-SP], AP`	Prvo se vrši dekrement pokazivača steka i potom se sadržaj AP registra prebacuje na vrh steka		-

Aritmetičke instrukcije su sledeće:

Sintaksa	**Opis**	**Primeri upotrebe**	**Utiče na indikator**
`ADD A, address`	Sabira sadržaj memorijske lokacije i akumulatora; `address` je konkretna memorijska lokacija definisana u DATA delu programa	`ADD A, VAR1` `ADD A, RES`	prenosa, nule
`ADD A, const`	Sabira `const` i sadržaj akumulatora	`ADD A, 1` `ADD A, 0X1F`	prenosa, nule
`ADD AP,const`	Sabira `const` i sadržaj AP registra	`ADD AP, 0` `ADD AP, 0XFF`	prenosa, nule
`ADD A, [AP]`	Sabira sadržaj akumulatora i memorijske lokacije čija se adresa nalazi u AP registru		prenosa, nule
`SUB A, address`	Oduzima sadržaj memorijske lokacije od vrednosti u akumulatoru; `address` je konkretna memorijska lokacija definisana u DATA delu programa	`SUB A, Z1` `SUB A, RES`	prenosa, nule
`SUB A, const`	Oduzima konstantu od vrednosti u akumulatoru	`SUB A, 1` `SUB A, 0X2`	prenosa, nule
`SUB AP, const`	Oduzima konstantu od vrednosti u AP registru	`SUB AP, 1` `SUB AP, 0X2`	prenosa, nule
`SUB A, [AP]`	Oduzima sadržaj memorijske lokacije od vrednosti koja se nalazi u akumulatoru		prenosa, nule

Logičke instrukcije su sledeće:

Sintaksa	**Opis**	**Primeri upotrebe**	**Utiče na indikator**
`NOT`	Prvi komplement vrednosti u akumulatoru		nule
`OR address`	Logičko ILI sadržaja u akumulatoru i memorijske lokacije; `address` je konkretna memorijska lokacija definisana u DATA delu programa	`OR VAR1` `OR RES`	nule
`OR const`	Logičko ILI sadržaja u akumulatoru i konstante `const`	`OR 11` `OR 0X12`	nule
`OR [AP]`	Logičko ILI sadržaja u akumulatoru i memorijske lokacije čija se adresa nalazi u AP registru		nule
`AND address`	Logičko I sadržaja u akumulatoru i memorijske lokacije; `address` je konkretna memorijska lokacija definisana u DATA delu programa	`AND Z1` `AND RES`	nule
`AND const`	Logičko I sadržaja u akumulatoru i konstante `const`	`AND 1` `AND 0X2`	nule
`AND [AP]`	Logičko I sadržaja u akumulatoru i memorijske lokacije čija se adresa nalazi u AP registru		nule
`XOR address`	Logičko ekskluzivno ILI sadržaja u akumulatoru i memorijske lokacije; `address` je konkretna memorijska lokacija definisana u DATA delu programa	`XOR P` `XOR RES`	nule
`XOR const`	Logičko ekskluzivno ILI sadržaja u akumulatoru i konstante `const`	`XOR 1` `XOR 0X2`	nule
`XOR [AP]`	Logičko ekskluzivno ILI sadržaja u akumulatoru i memorijske lokacije čija se adresa nalazi u AP registru		nule
`SHR`	Pomeranje sadržaja akumulatora za 1 bit u desno. (**bit odlazi u indikator prenosa!**)		prenosa, nule

Instrukcije grananja su:

Sintaksa	Opis	Primeri upotrebe	Utiče na indikator

`JMP label`	Bezuslovni skok na labelu	`JMP L1`
		`JMP LAB1`
		`JMP LRES`
			-

`JZ label`	Skok na labelu ukoliko je postavljen indikator nule	`JZ L2`
		`JZ LABELA2`
		`JZ L`
			-

`JC label`	Skok na labelu ukoliko je postavljen indikator prenosa	`JC L12`
		`JC LINC`
		`JC LDEC`
			-

Instrukcije za rad sa procedurama su:

Sintaksa	Opis	Primeri upotrebe	Utiče na indikator

`CALL procedure`	Poziv procedure	`CALL PROC`
		`CALL P1`
		`CALL INC`
			-

`RET`	Povratak iz procedure u glavni program		-

Instrukcije za rad sa periferijama su:

Sintaksa	Opis	Primeri upotrebe	Utiče na indikator
`IN`	Prebacuje sadržaj IN registra u akumulator		-
`OUT`	Prebacuje sadržaj akumulatora u OUT registar		-

Instrukcije programske kontrole su:

Sintaksa	Opis	Primeri upotrebe	Utiče na indikator
`NOP`	Nema operacije, prelazi na narednu instrukciju.		-
`END`	Označava kraj programa		-

Detaljan opis

U nastavku sledi detaljan opis svih instrukcija koji ima za cilj da ilustruje šta konkretno koja instrukcija radi i koji je njen uticaj na sadržaj registara i memorije. U opisu instrukcija se navode periodi sistemskog takta u kojima se izvršava pojedina *mikrooperacija* i konkretan opis te mikrooperacije. Opis mikrooperacije je urađen u *Register Transfer Notation (RTN)* simboličkom jeziku. Ono što je zajedničko za sve instrukcije je da se u okviru prve tri periode takta vrši prihvat instrukcija iz memorije. Za instrukcije koje zauzimaju dva bajta se u naredne 2 periode vrši prihvat adrese ili konstante. U narednim periodama se vrši izvršavanje instrukcije.

Prihvat instrukcija

Step	RTN
T0	MA ← PC
T1	MD ← M[MA] : PC ← PC+1
T2	IR ← MD

Prihvat operanda (adresa ili konstanta)

Step	RTN
T3	MA ← PC
T4	MD ← M[MA] : PC ← PC+1

Detaljan opis svih instrukcija

Instrukcije prenosa podataka

`MOV A, address` (0x11, address<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat adrese
T5	MA ← MD
T6	MD ← M[MA]
T7	A ← MD

`MOV AP, address` (0x13, address<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat adrese
T5	MA ← MD
T6	MD ← M[MA]
T7	AP ← MD

`MOV A, const` (0x19, const<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat konstante
T5	A ← MD

`MOV AP, const` (0x1B, const<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat konstante
T5	AP ← MD

`MOV A, [AP]` (0x14)

Step	RTN
T0-T2	Prihvat instrukcije
T3	MA ← AP
T4	MD ← M[MA]
T5	A ← MD

`MOV A, [SP+]` (0x1C)

Step	RTN
T0-T2	Prihvat instrukcije
T3	MA ← SP
T4	MD ← M[MA]: SP ← SP+1
T5	A ← MD

`MOV AP, [SP+]` (0x1E)

Step	RTN
T0-T2	Prihvat instrukcije
T3	MA ← SP
T4	MD ← M[MA]: SP ← SP+1
T5	AP ← MD

`MOV address, A` (0x21, address<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat adrese
T5	MA ← MD
T6	MD ← A
T7	M[MA] ← MD

`MOV address, AP` (0x23, address<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat adrese
T5	MA ← MD
T6	MD ← AP
T7	M[MA] ← MD

`MOV [-SP],AP` (0x2E)

Step	RTN
T0-T2	Prihvat instrukcije
T3	SP ← SP – 1
T4	MA ← SP
T5	MD ← AP
T6	M[MA] ← MD

`MOV [-SP],A` (0x2C)

Step	RTN
T0-T2	Prihvat instrukcije
T3	SP ← SP – 1
T4	MA ← SP
T5	MD ← A
T6	M[MA] ← MD

NAPOMENA: Nijedna instrukcija prenosa podataka ne utiče na statusni registar.

Aritmetičke instrukcije

`ADD A, address` (0x31, address<7..0>)
`SUB A, address` (0x41, address<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat adrese
T5	MA ← MD
T6	MD ← M[MA]
T7	R ← A + MD R ← A – MD
T8	A ← R

`ADD A, const` (0x39, const<7..0>)
`SUB A, const` (0x49, const<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat konstante
T5	R ← A + MD R ← A – MD
T6	A ← R

`ADD AP, const` (0x3B, const<7..0>)
`SUB AP, const` (0x4B, const<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat konstante
T5	R ← AP + MD R ← AP – MD
T6	AP ← R

`ADD A, [AP]` (0x34)
`SUB A, [AP]` (0x44)

Step	RTN
T0-T2	Prihvat instrukcije
T3	MA ← AP
T4	MD ← M[MA]
T5	R ← A + MD R ← A – MD
T6	A ← R

NAPOMENA: Sve aritmetičke instrukcije postavljaju indikator nule ukoliko je rezultat aritmetičke operacije jednak nuli. Kod oduzimanja se postavlja i indikator prenosa ukoliko se od manjeg broja oduzima veći broj, a kod sabiranja se postavlja indikator prenosa ukoliko se dobija rezultat koji zauzima više od 8 bita.

Logičke instrukcije

`NOT` (0x50)

Step	RTN
T0-T2	Prihvat instrukcije
T3	R ← ¬A
T4	A ← R

`SHR` (0x90)

Step	RTN
T0-T2	Prihvat instrukcije
T3	CZ<1>←A<0> : R ← 0 # A<7..1>
T4	A ← R

`OR address` (0x61, address<7..0>)
`AND address` (0x71, address <7..0>)
`XOR address` (0x81, address <7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat adrese
T5	MA ← MD
T6	MD ← M[MA]
T7	$R \leftarrow A \begin{matrix} \vee \\ \wedge \\ \oplus \end{matrix} MD$
T8	A ← R

`OR const` (0x69, const<7..0>)
`AND const` (0x79, const<7..0>)
`XOR const` (0x89, const<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat konstante
T5	$R \leftarrow A \begin{matrix} \vee \\ \wedge \\ \oplus \end{matrix} MD$
T6	A ← R

`OR [AP]` (0x64)
`AND [AP]` (0x74)
`XOR [AP]` (0x84)

Step	RTN
T0-T2	Prihvat instrukcije
T3	MA ← AP
T4	MD ← M[MA]
T5	$R \leftarrow A \begin{matrix} \vee \\ \wedge \\ \oplus \end{matrix} MD$
T6	A ← R

NAPOMENA: Sve logičke instrukcije postavljaju indikator nule ukoliko je rezultat operacije jednak nuli. Logička operacija pomeranja u desno za jedan bit (`SHR`) prebacuje bit najmanje težine (LSB) u indikator prenosa, a bit najviše težine (MSB) postaje jednak nuli.

Instrukcije grananja

`JMP label` (0xA1, const<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat adrese
T5	PC ← MD

Bezuslovni skok.

`JZ label` (0xA5, const<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat adrese
T5	CZ<1..0>=1 → PC ← MD

Skok ukoliko je indikator nule postavljen.

`JC label` (0xA9, const<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat adrese
T5	CZ<1..0>=2 → PC ← MD

Skok ukoliko je indikator prenosa postavljen.

I/O instrukcije

`IN` (0xD0)

Step	RTN
T0-T2	Prihvat instrukcije
T4	A ← IN

`OUT` (0xE0)

Step	RTN
T0-T2	Prihvat instrukcije
T4	OUT ← A

I/O instrukcije ne utiču na statusni registar.

Instrukcije za rad sa procedurama

`RET` (0xB0)

Step	RTN
T0-T2	Prihvat instrukcije
T3	MA ← SP
T4	MD ← M[MA]: SP ← SP + 1
T5	PC ← MD

`CALL procedure` (0xC1, const<7..0>)

Step	RTN
T0-T2	Prihvat instrukcije
T3-T4	Prihvat adrese
T5	AP ← MD : SP ← SP - 1
T6	MA ← SP
T7	MD ← PC
T8	M[MA] ← MD
T9	PC ← AP

NAPOMENA: Instrukcija poziva procedure (`CALL`) koristi AP registar i stoga se sadržaj tog registra gubi nakon izvršenja ove instrukcije! Ove instrukcije ne utiču na statusni registar. Instrukcija za povratak u glavni program (`RET`) očekuje da se na vrhu steka nalazi povratna adresa.

Ostale instrukcije

`NOP` (0x00)

Step	RTN
T0-T2	Prihvat instrukcije
T3	nema operacije

Ne utiče na statusni registar.

STRUKTURA PROGRAMA

Kostur programa

U nastavku sledi kostur programa pisanog u asembleru za Edulent:

```
PROGRAM 'Naziv programa'
DATA
      <rezervisanje memorijskih lokacija za varijable>
      ...
ENDDATA
CODE
      <instrukcije glavnog programa>
      ...
END   ; kraj glavnog programa
PROCEDURE <naziv_prve_procedure>
      <instrukcije prve procedure>
      ...
RET   ; povratak u glavni program
ENDPROCEDURE
...
PROCEDURE < naziv_poslednje_procedure >
      <instrukcije poslednje procedure>
      ...
RET
ENDPROCEDURE
ENDPROGRAM
```

Dodatna pravila

Rezervisane reči PROGRAM, DATA, ENDDATA, CODE i ENDPROGRAM se moraju naći u svakom programu, što se vidi i u programskom kosturu. Rezervisane reči PROCEDURE i ENDPROCEDURE mogu biti izostavljene ukoliko ni jedna procedura nije definisana. Ukoliko jeste, tada naziv procedure mora početi slovom.

Broj memorijskih lokacija koje korisnik može da rezerviše za varijable je 16. Naziv varijable mora početi slovom. Rezervisana reč `DB` se koristi u DATA delu za dodelu vrednosti varijablama.(npr. VAR1 DB 0x1A ili RES DB 12)

Programske labele moraju početi slovom L i završiti se dvotačkom (npr. L123:). Programske labele se moraju pisati ispred instrukcije koju označavaju
(npr. L123: MOV A, 0x1).

Konstante koje se koriste u programu se navode u *decimalnoj* ili *heksadecimalnoj* notaciji sa 0x ili 0X ispred broja. (npr. MOV A, 0x1F). Ostale notacije za sada nisu podržane i izazvaće grešku.
Komentar može biti postavljen nakon instrukcije, u istom redu, iza znaka tačka-zarez. (npr. MOV AP, 0x7F ; napuni AP sa 127). Prazni redovi ili redovi koji sadrže samo komentar mogu da postoje samo **nakon** rezervisane reči ENDDATA.

PRIMERI PROGRAMA

U nastavku su dati primeri programa koji ilustruju upotrebu seta instrukcija, kao i izgled programa.

```
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
      MOV A, VAR1
      MOV AP, @VAR1
      MOV A, [AP]
      MOV AP, 0XAA
      MOV RES1, AP
      MOV AP, VAR2
      MOV RES2, AP
      MOV RES3, A
      END
ENDPROGRAM
```

```
PROGRAM "Jumps and calls"
DATA
      RES1 DB 0X0
ENDDATA
CODE
      MOV A, 0X3
L1:   CALL DEC    ;dekrement
      JZ L2
      JMP L1
L2:   CALL INC    ;inkrement
      MOV RES1, A
      END
PROCEDURE INC
      ADD A,0X1
      RET
ENDPROCEDURE
PROCEDURE DEC
      SUB A,0X1
      RET
ENDPROCEDURE
ENDPROGRAM
```

```
PROGRAM "Instrukcije sabiranja"
DATA
      VAR1 DB 0X1
      VAR2 DB 0XA
      RES1 DB 0X1
      RES2 DB 0X0
      RES3 DB 0X0
      RES4 DB 0X0
ENDDATA
CODE
      MOV A, 0X1
      ADD A, 0X1
      MOV RES1, A
      MOV AP, 0X5
      ADD AP, 0X3
      MOV RES2, AP
      ADD A, VAR1
      MOV RES3, A
      MOV AP,@VAR1
      ADD A, [AP]
      MOV RES4, A
      END
ENDPROGRAM
```

```
PROGRAM "Carry i zero"
DATA
ENDDATA
CODE
      MOV A, 0XFF
      ADD A, 0X1
      ADD A, 0XFF
      ADD A,0X10
      END
ENDPROGRAM
```

```
PROGRAM "Rad sa stekom"
DATA
      VAR1 DB 0X1
      VAR2 DB 0XA
      RES1 DB 0X1
      RES2 DB 0X0
      RES3 DB 0X0
      RES4 DB 0X0
ENDDATA
CODE
      MOV A, 0X1
      ADD A, 0X1
      MOV [-SP], A
      MOV RES1, A
      ADD A, VAR1
      MOV [-SP], A
      MOV RES2, A
      MOV AP,@VAR1
      ADD A, [AP]
      MOV [-SP], A
      MOV RES3, A
      MOV A, [SP+]
      MOV AP, @VAR2
      MOV [-SP], AP
      ADD A, VAR1
      MOV RES4, AP
      MOV [-SP], AP
      END
ENDPROGRAM
```

```
PROGRAM "Logicke instrukcije"
DATA
      VAR1 DB 0X1
      VAR2 DB 0XA
      RES1 DB 0X0
      RES2 DB 0X0
      RES3 DB 0X0
      RES4 DB 0X0
ENDDATA
CODE
      MOV A, 0X1
      NOT

      MOV RES1, A
      AND 0XD
      MOV RES2, A
      OR 0X2
      MOV RES3, A
      XOR 0X3
      SHR
      MOV RES4, A
      END
ENDPROGRAM
```

ZADATAK – Napisati najmanji mogući program koji bi prošao kompajliranje.

ZADATAK – Dat je niz od 4 broja. Izračunati sumu ova 4 broja i srednju vrednost.

Šta radi prevodioc (kompajler)?

Svrha kompajlera je da programski kod unešen u simboličkom jeziku (asembleru) prevede u binarni kod, koji se potom puni u memoriju. U tabeli u prilogu je dat binarni ekvivalent svake instrukcije mikroprocesora Edulent. Bez detaljnijeg ulaženja u rad kompajlera, ovde će biti objašnjena struktura memorije koja se puni sadržajem koji kompajler pravi u .hex fajlu. Memorija ima 128 bajtova, od 0 – 0x7F. Počevši od memorijske lokacije nula, počinje programski kod, koji počinje prvom instrukcijom nakon rezervisane reči CODE i programski kod završava sa 0x02 (binarni ekvivalent instrukcije END) ili sa 0xB0 (binarni ekvivalent instrukcije RET, ukoliko je bilo procedura u programu). Obratiti pažnju da neke instrukcije zauzimaju jedan bajt, a neke dva bajta. Ukoliko instrukcije zauzimaju dva bajta, tada se u drugom bajtu nalazi operand (adresa ili konstanta).
Radi ilustracije sledi prikaz dela programa i binarni ekvivalent (u heksadecimalnoj notaciji) u memoriji(pogledati tabelu iz priloga ili detaljan opis instrukcija):

```
...
        mov a, 1            0x19
                            0x1
        add  a, 0xF         0x39
                            0xF
        shr                 0x90
        and 0xa             0x79
                            0xA

        and [ap]            0x74
        call proc           0xC1
                            0x4A
        ...
```

U navedenom primeru imamo 4 instrukcije koje zauzimaju 2 bajta i 2 koje zauzimaju 1 bajt.

Nakon programskog koda (koji se sastoji od glavnog programa i procedura) počinje deo u kome su rezervisane memorijske lokacije za varijable definisane u DATA delu programa.

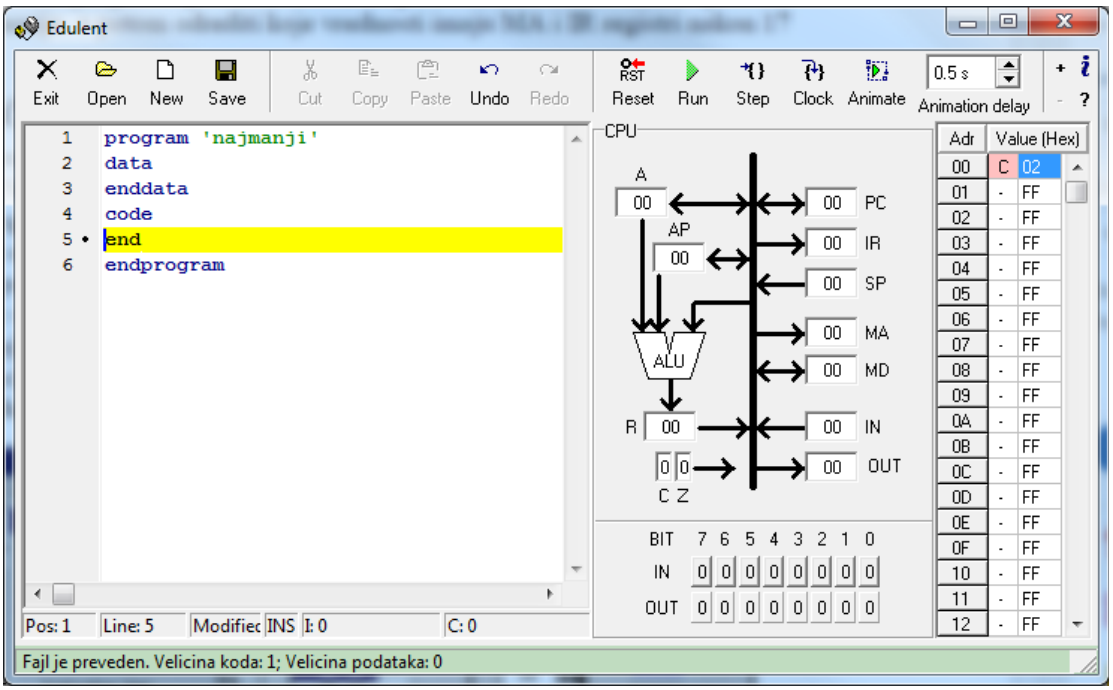
Nakon poslednje definisane varijable sve ostale lokacije se pune vrednošću 0xFF sve do kraja memorije. Stek memorija se popunjava od lokacije 0x7F prema manjim i u početku je prazna.

ZADATAK – Za primer programa pod nazivom „Jumps and calls“ sa 13. strane, nacrtati izgled i sadržaj memorije, a potom odrediti koje vrednosti imaju MA i IR registri nakon 17 perioda takta.

NAČIN KORIŠĆENJA EDULENT SIMULATORA

Cilj Edulent simulatora je da se omogući vizuelni prikaz izvršavanja programa napisanog u Edulent asembleru. Stoga je redosled operacija koji to omogućava sledeći:

1. Napisati program u nekom programu za unos teksta (Editpad, Notepad i sl.) i snimiti ga sa ekstenzijom .asm pri čemu je ime proizvoljno ili uneti direktno kod u Edulent simulatoru (pogledaj sliku 1). U donjoj statusnoj liniji se pojavljuju informacije o tome šta se očekuje kao unos kao i eventualne greške koje se pojave.
2. Ukoliko je program unesen bez grešaka, on je automatski preveden (kompajliran) a sadržaj memorije je učitan.



Slika 1 Izgled glavnog prozora Edulent simulatora

Slika 2 Izvršavanje programa

3. Dalje je potrebno pokrenuti izvršavanje programa pomoću dugmadi sa slike 2. Moguća su tri načina izvršavanja programa. Prvi je izvršavanje celog programa u *Run* modu (*Run* dugme). Drugi je izvršavanje programa instrukciju po instrukciju u *Single stepping* modu (*Step* dugme). Treći načina je izvršavanje programa "takt po takt" (*Clock* dugme). Nakon bilo koje od ove tri operacije može se pratiti uticaj programa, pojedinih instrukcija ili pojedinih mikroinstrukcija na registre i sadržaj memorije. Reset dugme služi za reset programa.

Dodatak

Tabela instrukcija

Dvobajtne instrukcije		Jednobajtne instrukcije	
		NOP	0x00
		END	0x02
MOV A, address	0x11	MOV A, [AP]	0x14
MOV AP, address	0x13	MOV A, [SP+]	0x1C
MOV A, const	0x19	MOV AP, [SP+]	0x1E
MOV AP, const	0x1B		
MOV address, A	0x21	MOV [-SP], A	0x2C
MOV address, AP	0x23	MOV [-SP], AP	0x2E
ADD A, address	0x31	ADD A, [AP]	0x34
ADD A, const	0x39		
ADD AP,const	0x3B		
SUB A, address	0x41	SUB A, [AP]	0x44
SUB A, const	0x49		
SUB AP, const	0x4B		
		NOT	0x50
OR address	0x61	OR [AP]	0x64
OR const	0x69		
AND address	0x71	AND [AP]	0x74
AND const	0x79		
XOR address	0x81	XOR [AP]	0x84
XOR const	0x89		
		SHR	0x90
JMP label	0xA1		
JZ label	0xA5		
JC label	0xA9		
CALL procedure	0xC1	RET	0xB0
		IN	0xD0
		OUT	0xE0

Napomena: u tabeli je naveden binarni ekvivalent samo prvog bajta instrukcija. Ukoliko instrukcija zahteva dva bajta, onda će drugi bajt biti operanda (odgovarajuća adresa ili konstanta). Dakle `address`, `const`, `label` i `procedure` će biti prevedeni u odgovarajući binarni broj i predstavljaće drugi bajt instrukcije.