Προχωρημένα θέματα προγραμματισμού σε συμβολική γλώσσα

Δρ. Παναγιώτης Παπάζογλου
Αν. Καθηγητής

Ανάπτυξη προσομοιωτή μικροεπεξεργαστών

Γενικά (1)

Προσομοιωτές

- Χρησιμοποιούνται σε όλα τα Πανεπιστήμια
- Υποθετικό ή πραγματικό μοντέλο μΕ/μC
- Προγραμματισμός Assembly/Εποπτεία λειτουργιών

Γενικά (2)

Η συγκεκριμένη εκπαιδευτική δραστηριότητα, έχει πολλαπλά οφέλη για τους φοιτητές, όπως:

- προσέγγιση του μικροεπεξεργαστή-μικροελεγκτή από την πλευρά του σχεδιαστή
- κατανόηση σε βάθος των αρχιτεκτονικών συστατικών του συστήματος
- κατανόηση της ουσιαστικής λειτουργίας της γλώσσας Assembly
- εμπέδωση των γνώσεων προγραμματισμού
- συνεργασία στα πλαίσια ομάδων εργασίας
- ανάπτυξη δεξιοτήτων επικοινωνίας και παρουσίασης (εφόσον παρουσιαστεί η αντίστοιχη εργασία στο κοινό)

Ανάπτυξη απλού προσομοιωτή για υποθετικό μικροεπεξεργαστή

Αξιοποίηση του προσομοιωτή (1)

Ο προσομοιωτής που θα αναπτύξουν οι φοιτητές, θα αξιοποιηθεί ως εξής:

- Από τη θέση του σχεδιαστή-προγραμματιστή του λογισμικού του προσομοιωτή
- Από τη θέση του χρήστη-προγραμματιστή που θα αξιοποιήσει τα εργαλεία που προσφέρει ο προσομοιωτής

Αξιοποίηση του προσομοιωτή (2)

Πλεονέκτημα σχεδιαστή-προγραμματιστή του λογισμικού του προσομοιωτή

- Κατανόηση των δομικών χαρακτηριστικών και της λειτουργίας του, αφού απαιτείται η μελέτη προδιαγραφών, περιορισμών, τρόπου λειτουργίας, εντολών συμβολικής γλώσσας, κλπ.

Πλεονεκτήματα χρήστη-προγραμματιστή

- Προγραμματισμός του «νέου» μικροεπεξεργαστή (που θα υπάρχει μόνο στο περιβάλλον του προσομοιωτή), έχοντας δυνατότητα λεπτομερούς παρατήρησης των αποτελεσμάτων και της λειτουργίας του προγράμματος Assembly που θα εκτελεί.

Λειτουργία του προσομοιωτή

Ο χρήστης-προγραμματιστής εισάγει εντολές Assembly σύμφωνα με τις προδιαγραφές του υποθετικού μοντέλου μικροεπεξεργαστή που «αντιπροσωπεύει» ο προσομοιωτής

Στη συνέχεια, γίνεται η εκτέλεση των εντολών που έχουν εισαχθεί και ο χρήστης παρατηρεί τα αντίστοιχα αποτελέσματα, έχοντας την αίσθηση ότι αυτές τρέχουν σε έναν «πραγματικό» μικροεπεξεργαστή

Γενικό μοντέλο εκτέλεσης προγράμματος

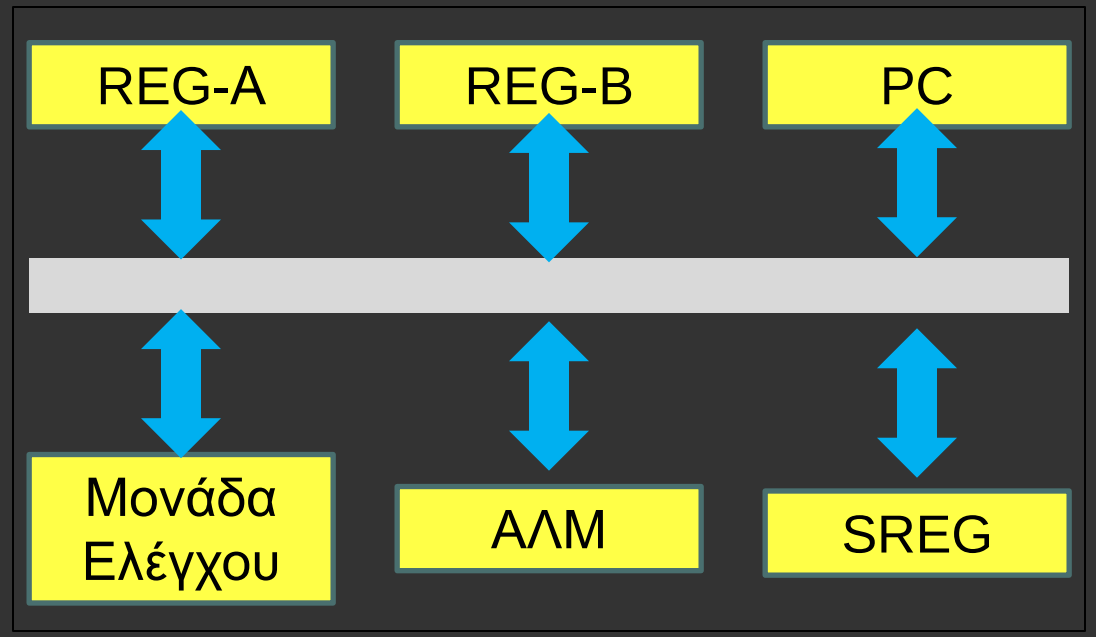
Μνήμη

PC=0000

1

Διεύθυνση (υποθετική)	Περιεχόμενο (υποθετικό)	Μορφή byte (υποθετικά)
0000	Εντολή-1 (π.χ. ADD A,5)	01 00 00 05
0004	Εντολή-2	….
0008	Εντολή-3 (STOP)	12 00 00 00

Μικροεπεξεργαστής



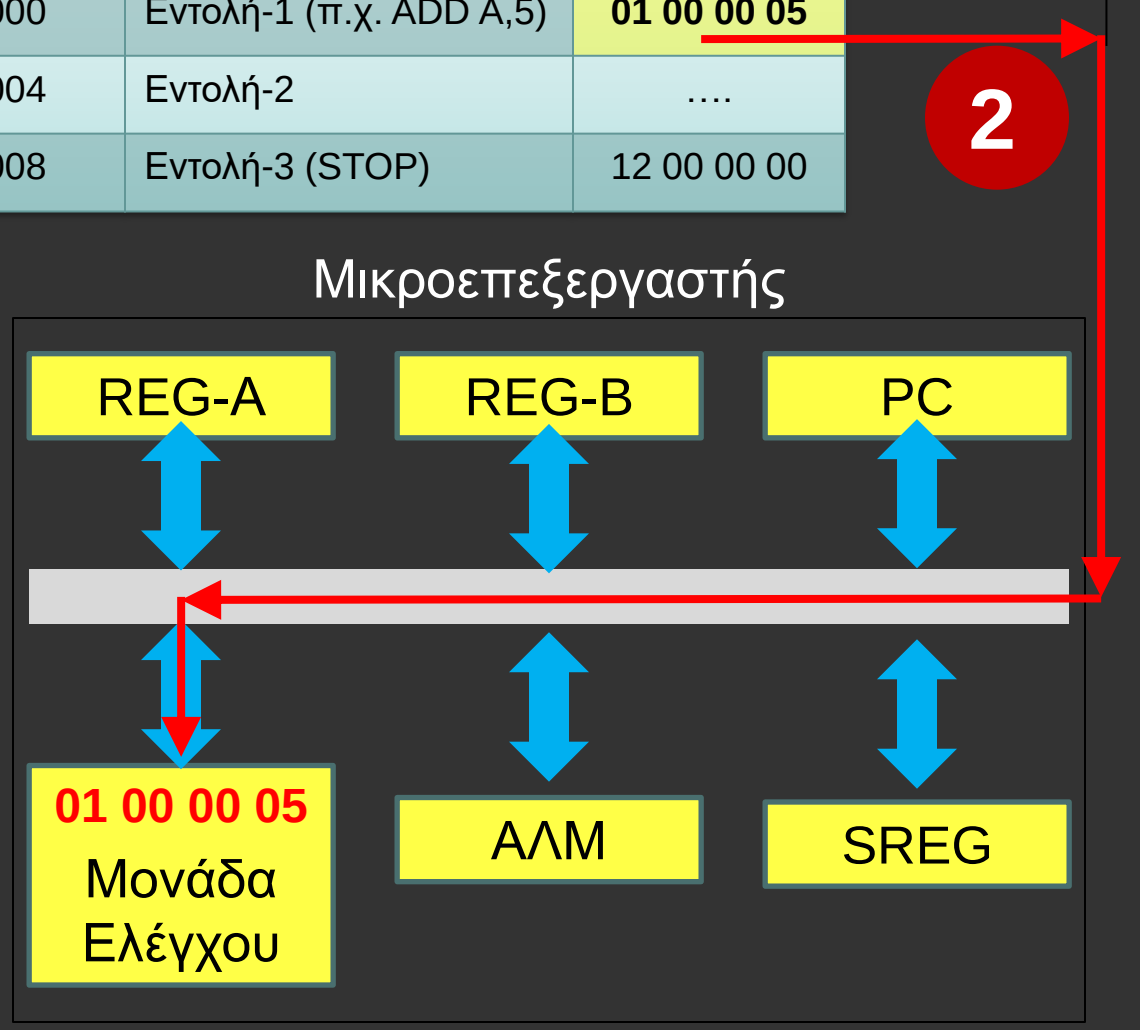
(1) Έναρξη εκτέλεσης

- Θα εκτελεστεί η εντολή που «δείχνει» ο PC
- Ενεργοποίηση διεύθυνσης 0000

Γενικό μοντέλο εκτέλεσης προγράμματος

Μνήμη

Διεύθυνση (υποθετική)	Περιεχόμενο (υποθετικό)	Μορφή byte (υποθετικά)
0000	Εντολή-1 (π.χ. ADD A,5)	**01 00 00 05**
0004	Εντολή-2	….
0008	Εντολή-3 (STOP)	12 00 00 00



(2) Αντιγραφή

Ο κώδικας της εντολής μαζί με τα ορίσματα, αντιγράφεται στο «σύστημα» της μονάδας ελέγχου

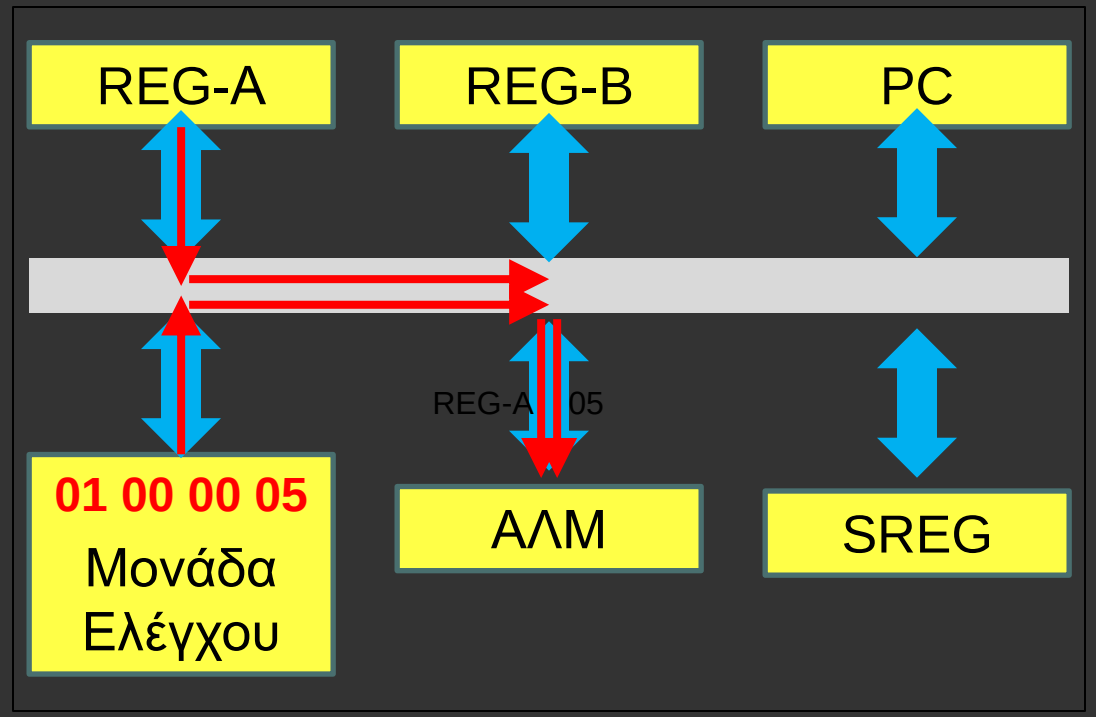
Γενικό μοντέλο εκτέλεσης προγράμματος

Μνήμη

Διεύθυνση (υποθετική)	Περιεχόμενο (υποθετικό)	Μορφή byte (υποθετικά)
0000	Εντολή-1 (π.χ. ADD A,5)	**01 00 00 05**
0004	Εντολή-2	….
0008	Εντολή-3 (STOP)	12 00 00 00

3

Μικροεπεξεργαστής



(3) Αντιγραφή

Το περιεχόμενο του REG-A και το όρισμα (05) αντιγράφονται στην ΑΛΜ

Γενικό μοντέλο εκτέλεσης προγράμματος

Μνήμη

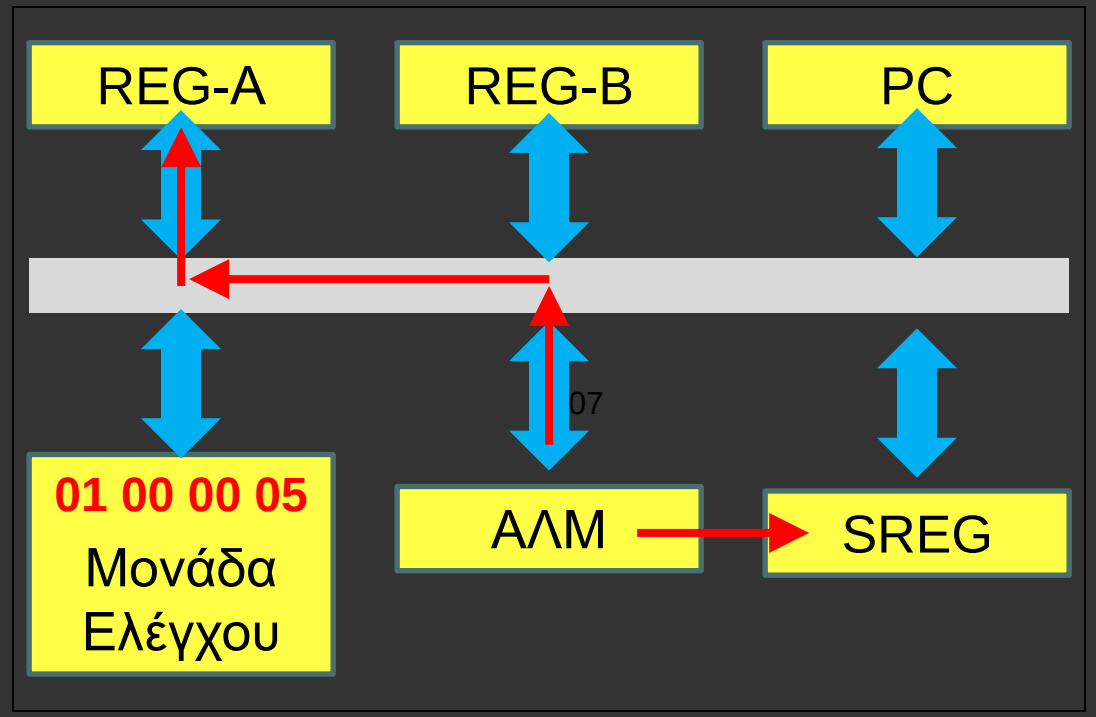
Διεύθυνση (υποθετική)	Περιεχόμενο (υποθετικό)	Μορφή byte (υποθετικά)
0000	Εντολή-1 (π.χ. ADD A,5)	**01 00 00 05**
0004	Εντολή-2	….
0008	Εντολή-3 (STOP)	12 00 00 00

4

(4) Εκτέλεση

- Γίνεται η πρόσθεση (REG-A + 05), έστω REG-A=2
- Ενημερώνεται ο καταχωρητής κατάστασης (SREG)
- Αποθηκεύεται το αποτέλεσμα στον REG-A

Μικροεπεξεργαστής



Γενικό μοντέλο εκτέλεσης προγράμματος

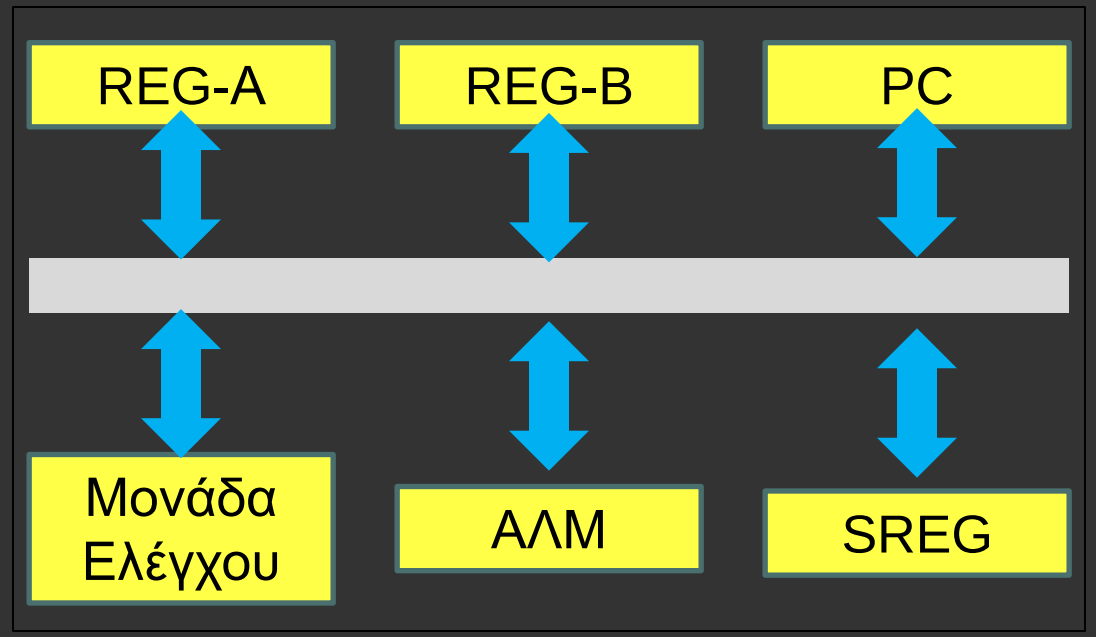
Μνήμη

PC=0004

5

Διεύθυνση (υποθετική)	Περιεχόμενο (υποθετικό)	Μορφή byte (υποθετικά)
0000	Εντολή-1 (π.χ. ADD A,5)	01 00 00 05
0004	Εντολή-2	….
0008	Εντολή-3 (STOP)	12 00 00 00

Μικροεπεξεργαστής



(5) Έναρξη εκτέλεσης επόμενης εντολής

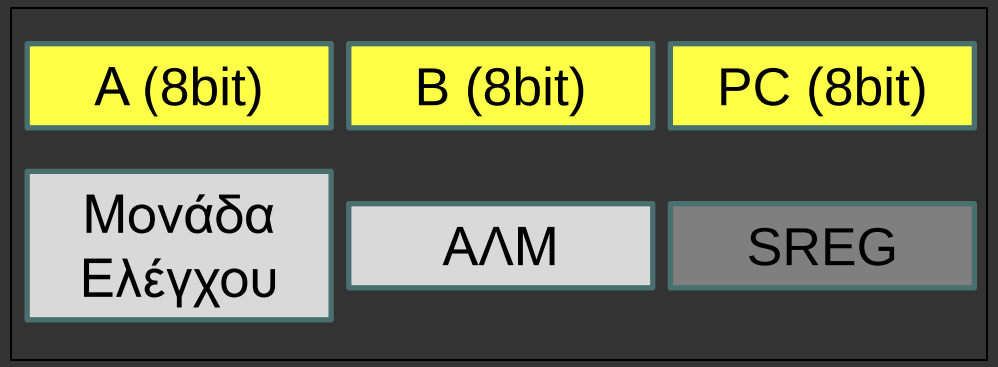
- Θα εκτελεστεί η εντολή που «δείχνει» ο PC
- Ενεργοποίηση διεύθυνσης 0004

Αρχικό υποθετικό μοντέλο μΕ που θα υλοποιηθεί

Μνήμη

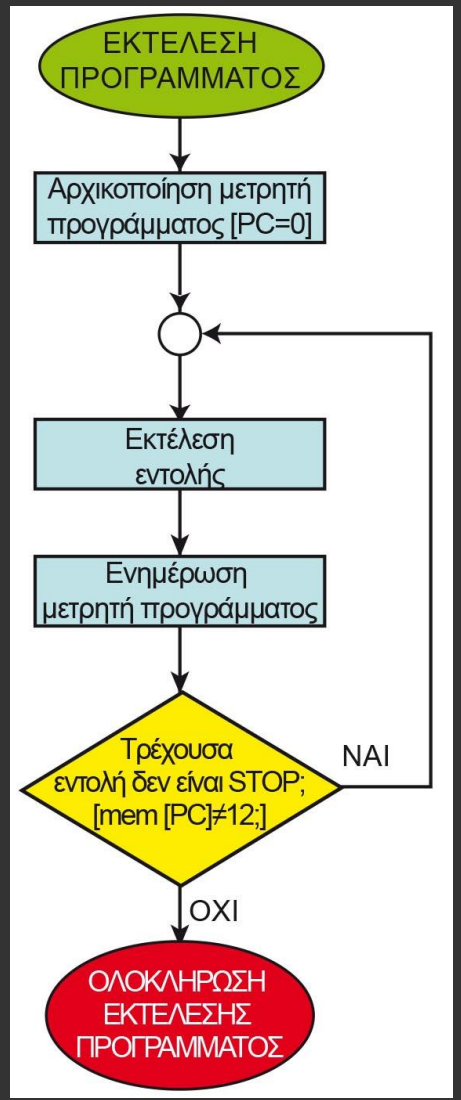
Διεύθυνση (θέση)	Περιεχόμενο
00	
01	
02	
99	

Μικροεπεξεργαστής



Στην αρχική υλοποίηση δεν αναπτύσσεται ξεχωριστή συνάρτηση για την ΑΛΜ και τον SREG.

Γενικό διάγραμμα λειτουργίας προσομοιωτή (εκτέλεση εντολών υποθετικού συστήματος)



- Ο μετρητής προγράμματος (PC) δείχνει πάντα τη διεύθυνση της εντολής που πρόκειται να εκτελεστεί
- Ενημερώνεται πριν την εκτέλεση της επόμενης εντολής
- Η εκτέλεση του προγράμματος ολοκληρώνεται (τερματισμός) όταν η ροή εκτέλεσης φτάσει στην εντολή STOP (κώδικας εντολής=12)

Υλοποίηση του προσομοιωτή σε Java (1)

Δήλωση βασικών μεταβλητών

Μετρητής προγράμματος (PC)

```
static byte PC;
```

Καταχωρητές γενικής χρήσης (A,B)

```
static byte A;
static byte B;
```

Κεντρική μνήμη (πίνακας mem 100 θέσεων)

```
static byte mem[]=new byte[100];
```

Περιγραφή εντολών Assembly (πίνακας ins 13 θέσεων)

```
static String ins[]=new String[13];
```

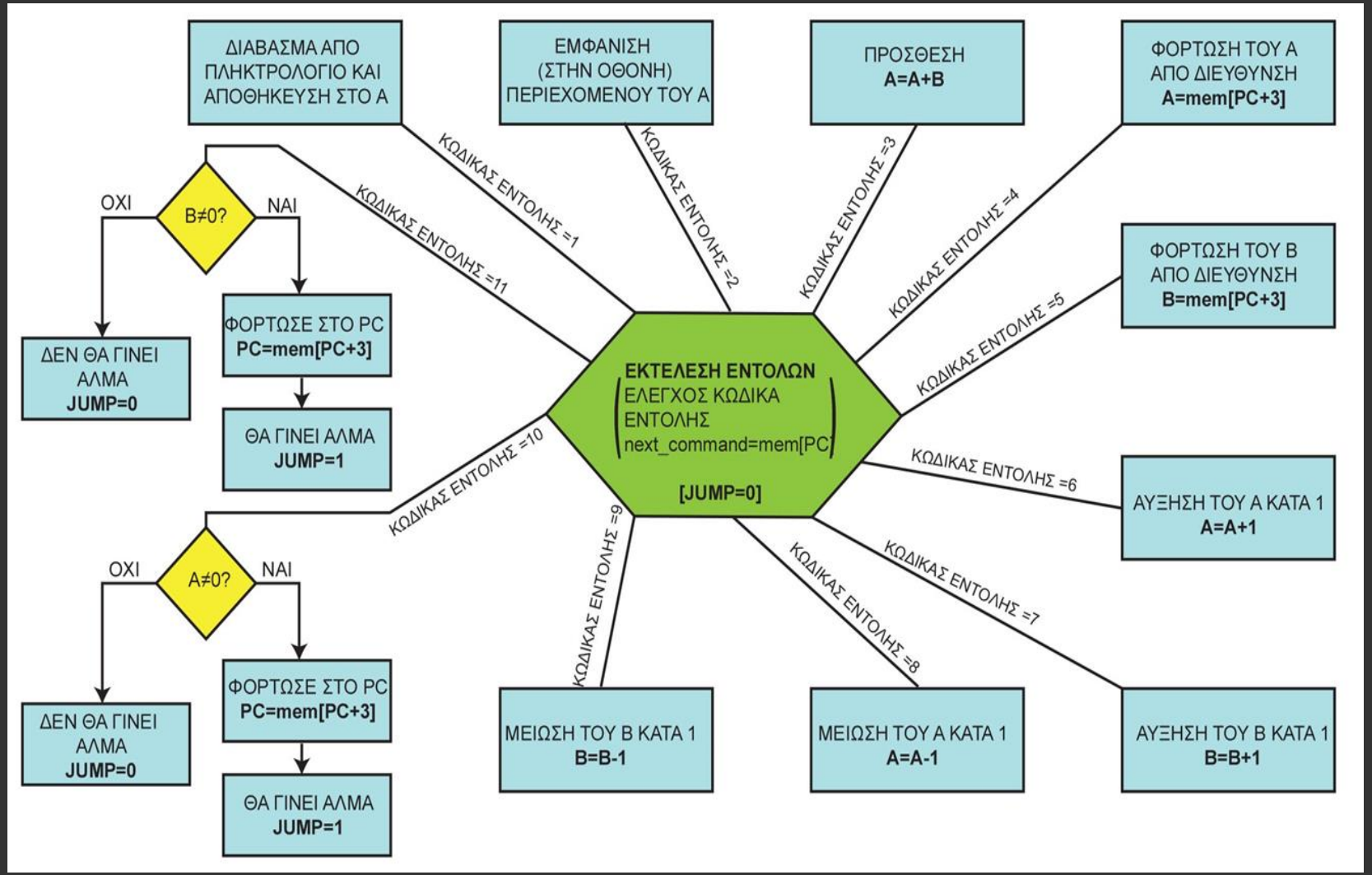
Υλοποίηση του προσομοιωτή σε Java (2)

Πίνακας υποστηριζόμενων εντολών

```
ins[1]= "RD (A <-- KEYB)                01 00 00 00";
ins[2] ="WR (A --> CON)                 02 00 00 00";
ins[3]= "ADD (A <-- A+B)                03 00 00 00";
ins[4]= "MOV (A <-- XX)                 04 00 00 XX";
ins[5]= "MOV (B <-- XX)                 05 00 00 XX";
ins[6]= "INC (A <-- A+1)                06 00 00 00";
ins[7]= "INC (B <-- B+1)                07 00 00 00";
ins[8] ="DEC (A <-- A-1)                08 00 00 00";
ins[9]= "DEC (B <-- B-1)                09 00 00 00";
ins[10]="JNZ (A<>0, goto XX)            10 00 00 XX";
ins[11]="JNZ (B<>0, goto XX)            11 00 00 XX";
ins[12]="STOP (Termination)             12 00 00 00";
```

- Κάθε εντολή αποτελείται από 4 byte
- Το πρώτο είναι ο μοναδικός κώδικας κάθε εντολής
- Το XX αντιπροσωπεύει το όρισμα (αν έχει η εντολή)
- Τα υπόλοιπα byte δεν χρησιμοποιούνται (για μελλοντική επέκταση)

Διάγραμμα υποστηριζόμενων εντολών



Υλοποίηση του προσομοιωτή σε Java (3)

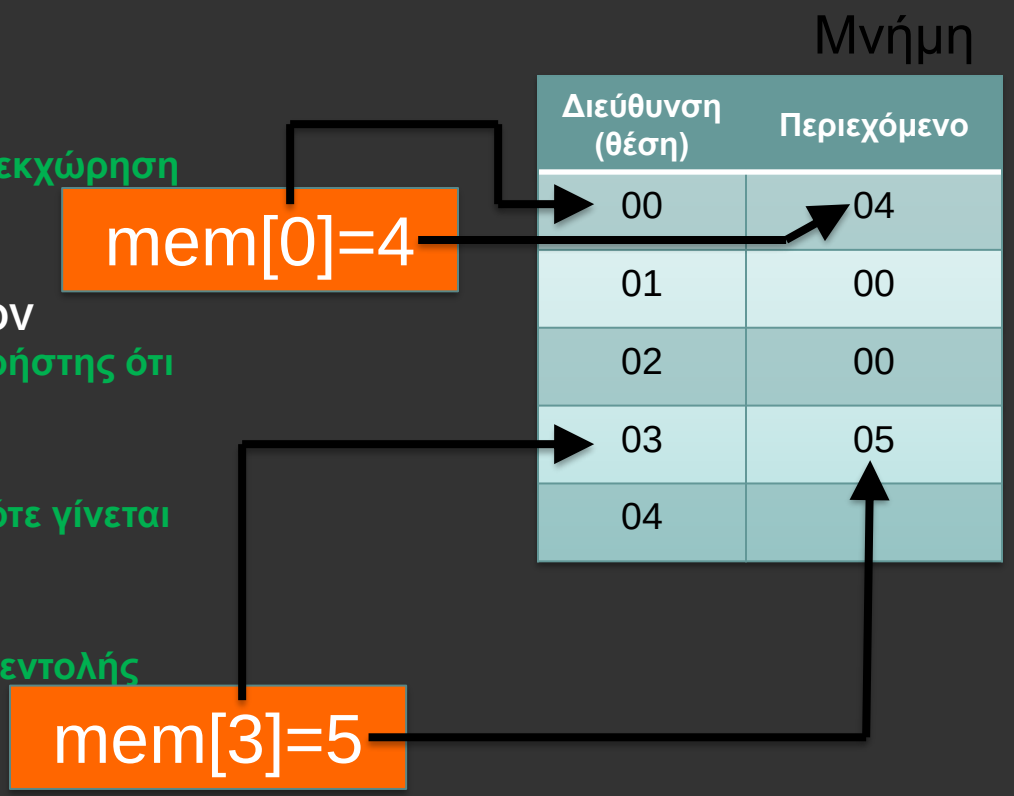
Αποθήκευση πηγαίου κώδικα στη μνήμη

```
System.out.println("Enter Opcode"); //Βοηθητικό μήνυμα για εισαγωγή εντολής
System.out.println("------------");
int command;                        //Αποθήκευση κώδικα εντολής που θα πληκτρολογηθεί
int XX=0;                           //Αποθήκευση ορίσματος που θα πληκτρολογηθεί (αν απαιτηθεί)
PC=0;                               //Τρέχουσα θέση πίνακα (μνήμη) που θα αποθηκευτεί η εντολή
do                                  //το PC εδώ χρησιμοποιείται βοηθητικά
{
print_mem_reg();                    //Εμφάνιση περιεχομένων μνήμης και καταχωρητών
System.out.print("Starting address ["+PC+"]:"); //Εμφάνιση τρέχουσας θέσης στη μνήμη
command=read();                     //Ανάγνωση κώδικα εντολής από το πληκτρολόγιο
mem[PC]=(byte)command;              //Αποθήκευση κώδικα εντολής στη θέση mem[PC]
switch (command)                    //Έλεγχος αν έχει δοθεί εντολή που απαιτεί όρισμα
 {
  case 4:  { System.out.print("MOV A,XX\t[XX]="); break; } //Εμφάνιση μορφής εντολής 4
  case 5:  { System.out.print("MOV B,XX\t[XX]="); break; } //Εμφάνιση μορφής εντολής 5
  case 10: { System.out.print("JNZ A,XX\t[XX]="); break; }  //Εμφάνιση μορφής εντολής 10
  case 11:
 }
if (command==4 || command==5 || command==10 //Αν η εντολή έχει όρισμα, θα διαβαστεί
|| command==11) { XX=read(); mem[PC+3]=(byte)XX; } //Αποθήκευση ως το 4ο byte της εντολής
PC+=4;     //Η επόμενη εντολή θα ξεκινά τέσσερις (4) θέσεις πιο κάτω
}
while (command!=12); //Αν εισαχθεί η εντολή 12 (STOP), η διαδικασία τερματίζεται
```

Υλοποίηση του προσομοιωτή σε Java (4)

Παράδειγμα εισαγωγής της εντολής MOV A,5

- Θεωρούμε αρχικά ότι XX=0, PC=0
- Ανάγνωση από το πληκτρολόγιο, command=read();
- Πληκτρολογούμε 4 και με την εντολή mem[PC]=(byte)command θα εκτελεστεί η εκχώρηση mem[0]=4
- Στη συνέχεια θα εμφανιστεί το μήνυμα "MOV A,XX\t[XX]=" (case 4) για να καταλάβει ο χρήστης ότι πρέπει να εισάγει το όρισμα
- Εφόσον είναι η εντολή 4 (command==4), τότε γίνεται ανάγνωση του ορίσματος (XX=read())
- Το όρισμα αποθηκεύεται ως το 4ο byte της εντολής (mem[PC+3]=(byte)XX), έστω XX=5
- Το PC αυξάνεται (PC+=4) για την εισαγωγή νέας εντολής στην επόμενη διαθέσιμη θέση της μνήμης



Υλοποίηση του προσομοιωτή σε Java (5)

Εκτέλεση εντολών από τη μνήμη-ψευδοκώδικας

(ΨΕΥΔΟΚΩΔΙΚΑΣ)

PC=0 (αρχικοποίηση μετρητή προγράμματος, εκτέλεση από τη διεύθυνση 0)

ΕΠΑΝΕΛΑΒΕ

Jump=0

Θεωρούμε αρχικά ότι δεν έχει ζητηθεί άλμα σε άλλη διεύθυνση (από την τρέχουσα εντολή).

next_command=mem[PC];

Διάβασε τον κώδικα της εντολής (1 byte) από τη διεύθυνση που δείχνει ο μετρητής προγράμματος (PC)

(switch**) Έλεγχος του κώδικα εντολής** (Συνθήκη ΑΝ για τον κώδικα της εντολής)

1: Διάβασε τιμή από το πληκτρολόγιο και αποθήκευσε την στον καταχωρητή Α (μεταβλητή Α)

2: Εμφάνιση στην οθόνη του Α

3: Πρόσθεση των Α+Β και αποθήκευση αποτελέσματος στο Α

4: Αποθήκευση του περιεχομένου της θέσης PC+3 στο Α (η θέση PC+3 περιέχει το όρισμα της εντολής που έχει βάλει ο χρήστης)

5: Αποθήκευση του περιεχομένου της θέσης PC+3 στο Β (η θέση PC+3 περιέχει το όρισμα της εντολής που έχει βάλει ο χρήστης)

6: Το περιεχόμενο του Α αυξάνεται κατά 1

7: Το περιεχόμενο του Β αυξάνεται κατά 1

8: Το περιεχόμενο του Α μειώνεται κατά 1

9: Το περιεχόμενο του Β μειώνεται κατά 1

10: Αν το Α δεν είναι μηδέν (JNZ), τότε διάβασε από τη θέση μνήμης mem[PC+3] σε ποια διεύθυνση θα γίνει το άλμα. Η θέση PC+3 περιέχει το όρισμα της εντολής (διεύθυνση άλματος) που έχει βάλει ο χρήστης. Ενημέρωση της σημαίας άλματος (Jump=1).

11: Αν το Β δεν είναι μηδέν (JNZ), τότε διάβασε από τη θέση μνήμης mem[PC+3] σε ποια διεύθυνση θα γίνει το άλμα. Η θέση PC+3 περιέχει το όρισμα της εντολής (διεύθυνση άλματος) που έχει βάλει ο χρήστης. Ενημέρωση της σημαίας άλματος (Jump=1).

Τέλος switch (Τέλος) ελέγχου κώδικα εντολής

Αν δεν έχει ζητηθεί άλμα (Jump=0) => **PC=PC+4** (4 byte = μήκος κάθε εντολής)

ΟΣΟ (Επανέλαβε) ο κώδικας εντολής δεν είναι 12 (εντολή τερματισμού, **STOP**)

Υλοποίηση του προσομοιωτή σε Java (6)

Εκτέλεση εντολών από τη μνήμη-πραγματικές εντολές

```
PC=0;       //Η εκτέλεση θα αρχίσει από την εντολή που ξεκινά στη διεύθυνση 00
do
{
int jump=0; //Θεωρούμε αρχικά ότι δεν πρόκειται για εντολή άλματος
int next_command=mem[PC]; //Διαβάζεται ο κώδικας της τρέχουσας εντολής από τη θέση mem[PC]
switch (next_command) //Ανάλογα την εντολή (next_command) θα γίνουν οι κατάλληλες διαδικασίες
{                              //Εκτέλεση εντολής κατά περίπτωση
case 1: System.out.print("A=");A=(byte)read();break;
case 2: System.out.println(A); break;
case 3: A=(byte)(A+B); break;
case 4: A=mem[PC+3]; break; //π.χ. στην εντολή 4, αποθηκεύουμε στον καταχωρητά Α, το όρισμα
case 5: B=mem[PC+3]; break;
case 6: A+=1; break;
case 7: B+=1; break;
case 8: A-=1; break;
case 9: B-=1; break;
case 10: if (A!=0) { PC=mem[PC+3]; jump=1; } else jump=0; break; //Άλμα στη διεύθυνση όρισμα (A<>0)
case 11: if (B!=0) { PC=mem[PC+3]; jump=1; } else jump=0; break; //Άλμα στη διεύθυνση όρισμα (B<>0)
}
if (jump==0) PC+=4; //Αν δεν πρόκειται να γίνει άλμα, συνεχίζουμε από την επόμενη εντολή
}
while (mem[PC]!=12); //Εκτέλεση επόμενης εντολής, εφόσον δεν φτάσαμε στην εντολή STOP
```

Υλοποίηση του προσομοιωτή σε Java (7)

Εκτέλεση της εντολής MOV A,5

Αρχικά θεωρούμε ότι PC=0, jump=0

Διαβάζουμε από τη μνήμη, την τρέχουσα εντολή

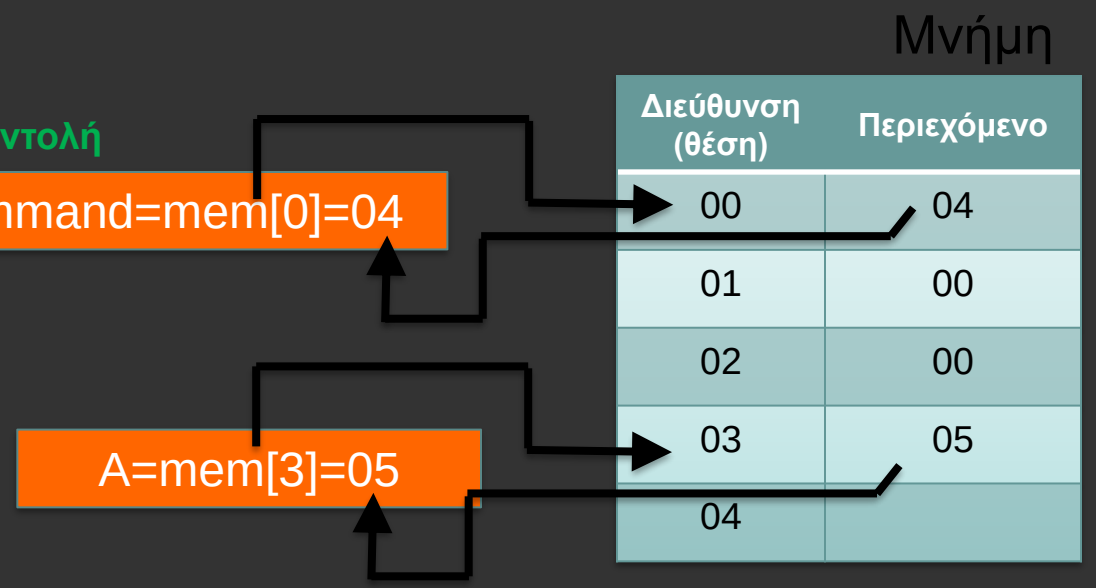
next_command=mem[PC]

Ελέγχουμε το είδος της εντολής

switch (next_command)

Εφόσον πρόκειται για την εντολή 4

case 4: A=mem[PC+3]; break;



Δεν πρόκειται να γίνει άλμα και το PC θα πρέπει να δείχνει τη διεύθυνση της επόμενης εντολής

if (jump==0) PC+=4;

Υλοποίηση του προσομοιωτή σε Java (8)

Εμφάνιση περιεχομένων μνήμης

Για κάθε θέση μνήμης, εμφανίζουμε την εντολή που περιέχει

```
System.out.print("["+i+"]\t\t"+ins[mem[i]]);
```

π.χ. ins[mem[0]]

mem[0]=04 ➔ ins[4]

```
ins[4]="MOV (A <-- XX)          04 00 00 XX";
ins[mem[0]]="MOV (A <-- XX)  04 00 00 XX";
```

```
ins[1]= "RD (A <-- KEYB)       01 00 00 00";
ins[2] ="WR (A --> KEYB)       02 00 00 00";
ins[3]= "ADD (A <-- A+B)       03 00 00 00";
ins[4]= "MOV (A <-- XX)        04 00 00 XX";
ins[5]= "MOV (B <-- XX)        05 00 00 XX";
ins[6]= "INC (A <-- A+1)       06 00 00 00";
ins[7]= "INC (B <-- B+1)       07 00 00 00";
ins[8] ="DEC (A <-- A-1)       08 00 00 00";
ins[9]= "DEC (B <-- B-1)       09 00 00 00";
ins[10]="JNZ (A<>0, goto XX)10 00 00 XX";
ins[11]="JNZ (B<>0, goto XX)11 00 00 XX";
ins[12]="STOP (Termination)12 00 00 00";
```

Πίνακας ins

Υπόδειγμα Μνήμης

Διεύθυνση (θέση)	Περιεχόμενο
00	04
01	00
02	00
03	05
04	

Πίνακας mem

Αν το όρισμα δεν είναι μηδέν, το εμφανίζουμε στην οθόνη

```
if (mem[i+3]!=0)
System.out.println(",
VALUE="+mem[i+3]);
else
System.out.println("");
```

Υλοποίηση του προσομοιωτή σε Java (9)

Κεντρικό μενού επιλογών

```
do
{
  do
   {
     menu();
     c=read(); //Ανάγνωση από το πληκτρολόγιο
   }
 while ((c<0) || (c>6));
 switch (c)
 {
   case 1: help(); break;
   case 2: execute(); break;
   case 3: enter_source(); break;
   case 4: clear_mem(); break;
   case 5: command_list(); break;
   case 6: print_mem_reg(); break;
 }
}
  while (c!=0);
}
```

1.Help
2.Execute
3.Enter source
4.Clear MEM
5.Command list
6.Print MEM/REG
0.Exit

Υλοποίηση του προσομοιωτή σε Java (10)

Πρόσθεση νέας εντολής – Παρέμβαση στη λίστα εντολών

- Έστω ότι, η νέα εντολή εμφανίζει το περιεχόμενο του καταχωρητή B στην οθόνη
- Συμβολισμός εντολής: **WRB (B -->CON)**
- Κώδικας εντολής: **13**

```
ins[1]= "RD (A <-- KEYB)    01 00 00 00";
ins[2] ="WR (A --> CON)     02 00 00 00";
ins[3]= "ADD (A <-- A+B)    03 00 00 00";
ins[4]= "MOV (A <-- XX)     04 00 00 XX";
ins[5]= "MOV (B <-- XX)     05 00 00 XX";
ins[6]= "INC (A <-- A+1)    06 00 00 00";
ins[7]= "INC (B <-- B+1)    07 00 00 00";
ins[8] ="DEC (A <-- A-1)    08 00 00 00";
ins[9]= "DEC (B <-- B-1)    09 00 00 00";
ins[10]="JNZ (A<>0, goto XX) 10 00 00 XX";
ins[11]="JNZ (B<>0, goto XX) 11 00 00 XX";
ins[12]="STOP (Termination) 12 00 00 00";
```

Νέα λίστα

```
ins[1]= "RD (A <-- KEYB)    01 00 00 00";
ins[2] ="WR (A --> CON)     02 00 00 00";
ins[3]= "ADD (A <-- A+B)    03 00 00 00";
ins[4]= "MOV (A <-- XX)     04 00 00 XX";
ins[5]= "MOV (B <-- XX)     05 00 00 XX";
ins[6]= "INC (A <-- A+1)    06 00 00 00";
ins[7]= "INC (B <-- B+1)    07 00 00 00";
ins[8] ="DEC (A <-- A-1)    08 00 00 00";
ins[9]= "DEC (B <-- B-1)    09 00 00 00";
ins[10]="JNZ (A<>0, goto XX) 10 00 00 XX";
ins[11]="JNZ (B<>0, goto XX) 11 00 00 XX";
ins[12]="STOP (Termination) 12 00 00 00";
ins[13]="WRB (B --> CON) 13 00 00 00"
```

Περιγραφή εντολών Assembly (**πίνακας ins 14 θέσεων**)

static String ins[]=new String[14];

Υλοποίηση του προσομοιωτή σε Java (11)

Πρόσθεση νέας εντολής – Παρέμβαση στην εκτέλεση εντολών

```
PC=0;
do
{
int jump=0;
int next_command=mem[PC];
switch (next_command)
{
case 1: System.out.print("A=");A=(byte)read();break;
case 2: System.out.println(A); break;
case 3: A=(byte)(A+B); break;
case 4: A=mem[PC+3]; break;
case 5: B=mem[PC+3]; break;
case 6: A+=1; break;
case 7: B+=1; break;
case 8: A-=1; break;
case 9: B-=1; break;
case 10: if (A!=0) { PC=mem[PC+3]; jump=1; } else jump=0; break;
case 11: if (B!=0) { PC=mem[PC+3]; jump=1; } else jump=0; break;
case 13: System.out.println(B); break;
}
if (jump==0) PC+=4;
}
while (mem[PC]!=12);
```

Ολοκληρωμένη Υλοποίηση του προσομοιωτή

Συναρτήσεις προσομοιωτή

Υποπρόγραμμα	Περιγραφή
menu	Εμφανίζει το κεντρικό μενού
command_list	Εμφανίζει στην οθόνη το όνομα (μνημονικό) κάθε διαθέσιμης εντολής μαζί με τον αντίστοιχο κώδικα
init_instructions	Αρχικοποιεί τον πίνακα με τις ονομασίες και τα ορίσματα των διαθέσιμων εντολών
execute	Πρόκειται για τον πυρήνα του συστήματος, αφού αναλαμβάνει την εκτέλεση των υποθετικών εντολών Assembly που έχει εισάγει ο χρήστης
print_mem_reg	Εμφανίζει στην οθόνη το περιεχόμενο των καταχωρητών A,B, του μετρητή προγράμματος (PC) καθώς και της υποθετικής μνήμης
enter_source	Διαδικασία εισαγωγής εντολών προγράμματος Assembly (με τον κώδικα εντολής) στην υποθετική μνήμη
help	Εμφανίζει βοηθητικό μήνυμα για το μενού επιλογών
clear_mem	Αρχικοποιεί με μηδενικά (καθαρισμός) τον πίνακα της υποθετικής μνήμης
empty_lines	Εμφανίζει κενές γραμμές για τον καθαρισμό περιοχής της οθόνης
read	Διαβάζει αλφαριθμητικό από το πληκτρολόγιο (αριθμό για το χρήστη) και επιστρέφει την αριθμητική του αξία

Ολοκληρωμένη Υλοποίηση του προσομοιωτή

Υποστηριζόμενες εντολές

Εντολή	Κωδικοποίηση	Λειτουργία
RD	01 00 00 00	**(A ← KEYB)** Ανάγνωση αριθμού από το πληκτρολόγιο και αποθήκευση στον καταχωρητή A
WR	02 00 00 00	**(A → CON)** Εμφάνιση περιεχομένων καταχωρητή A στην οθόνη
ADD B	03 00 00 00	**(A ← A+B)** Πρόσθεση περιεχομένων καταχωρητών A και B και αποθήκευση στον καταχωρητή A
MOV A,XX	04 00 00 XX	**(A ← XX)** Φόρτωση ακέραιου αριθμού στον καταχωρητή A
MOV B,XX	05 00 00 XX	**(B ← XX)** Φόρτωση ακέραιου αριθμού στον καταχωρητή B
INC A	06 00 00 00	**(A ← A+1)** Αύξηση του περιεχομένου του καταχωρητή A κατά 1
INC B	07 00 00 00	**(B ← B+1)** Αύξηση του περιεχομένου του καταχωρητή B κατά 1
DEC A	08 00 00 00	**(A ← A-1)** Μείωση του περιεχομένου του καταχωρητή A κατά 1
DEC B	09 00 00 00	**(B ← B-1)** Μείωση του περιεχομένου του καταχωρητή B κατά 1
JNZ A,XX	10 00 00 XX	**(Αν A<>0, τότε μετάβαση στη διεύθυνση XX)** Μετάβαση στη διεύθυνση XX αν το περιεχόμενο του καταχωρητή A δεν είναι μηδέν
JNZ B,XX	11 00 00 XX	**(Αν B<>0, τότε μετάβαση στη διεύθυνση XX)** Μετάβαση στη διεύθυνση XX αν το περιεχόμενο του καταχωρητή B δεν είναι μηδέν
STOP	12 00 00 00	**(Exit)** Τερματισμός της εκτέλεσης προγράμματος

Ολοκληρωμένος κώδικας (1)

```
import java.io.*;
import java.lang.*;

class simulator
{
static byte PC;
static byte A;
static byte B;
static byte mem[]=new byte[100];
static byte IR[]=new byte[3];
static String ins[]=new String[13];

public static void main(String[] args)
{
int c;
empty_lines();
init_instructions();
System.out.println("Dr Panayotis M. Papazoglou\n");
System.out.println("Microprocessor Simulator\n");
do
{
do
{
menu();
c=read();
}
while ((c<0) || (c>6));
switch (c)
{
case 1: help(); break;
case 2: execute(); break;
case 3: enter_source(); break;
case 4: clear_mem(); break;
case 5: command_list(); break;
case 6: print_mem_reg(); break;
}
}
while (c!=0);
}
```

Ολοκληρωμένος κώδικας (2)

```
 public static void menu()
 {
System.out.println("\n 1.Help 2.Execute
3.Enter source 4.Clear MEM 5.Command list
6.Print MEM/REG 0.Exit\n");
 }
public static void command_list()
{
for(int i=1;i<=12;i++)
System.out.println(ins[i]);
}
public static void init_instructions()
 {
ins[1]= "RD (A <-- KEYB) 01 00 00 00";
ins[2] ="WR (A --> CON) 02 00 00 00";
ins[3]= "ADD (A <-- A+B) 03 00 00 00";
ins[4]= "MOV (A <-- XX) 04 00 00 XX";
ins[5]= "MOV (B <-- XX) 05 00 00 XX";
ins[6]= "INC (A <-- A+1) 06 00 00 00";
ins[7]= "INC (B <-- B+1) 07 00 00 00";
ins[8] ="DEC (A <-- A-1) 08 00 00 00";
ins[9]= "DEC (B <-- B-1) 09 00 00 00";
ins[10]="JNZ (A<>0, goto XX) 10 00 00 XX";
ins[11]="JNZ (B<>0, goto XX) 11 00 00 XX";
ins[12]="STOP (Termination) 12 00 00 00";
}
```

```
 public static void execute()
 {
System.out.println("Executing program...\n\n\n");
PC=0;
do
{
int jump=0;
int next_command=mem[PC];
switch (next_command)
{
case 1: System.out.print("A=");A=(byte)read();break;
case 2: System.out.println(A); break;
case 3: A=(byte)(A+B); break;
case 4: A=mem[PC+3]; break;
case 5: B=mem[PC+3]; break;
case 6: A+=1; break;
case 7: B+=1; break;
case 8: A-=1; break;
case 9: B-=1; break;
case 10: if (A!=0) { PC=mem[PC+3]; jump=1; } else jump=0; break;
case 11: if (B!=0) { PC=mem[PC+3]; jump=1; } else jump=0; break;
}
if (jump==0) PC+=4;
}
while (mem[PC]!=12);
System.out.println("\n Done!!!");
}
```

Ολοκληρωμένος κώδικας (3)

```
public static void print_mem_reg()
{
int i=0;
while (i<=99)
{
System.out.print("["+i+"]\t\t"+ins[mem[i]]);
if (mem[i+3]!=0) System.out.println(",
                VALUE="+mem[i+3]); else
System.out.println("");
i+=4;
}
System.out.println("\n PC="+PC+", A="+A+", B="+B)
}
```

```
public static void enter_source()
 {
System.out.println("Enter Opcode");
System.out.println("------------");
int command;
int XX=0;
PC=0;
do
{
print_mem_reg();
System.out.print("Starting address ["+PC+"]:");
command=read();
mem[PC]=(byte)command;
switch (command)
{
case 4:      {System.out.print("MOV A,XX\t[XX]="); break;}
case 5:      {System.out.print("MOV B,XX\t[XX]="); break;}
case 10:     {System.out.print("JNZ A,XX\t[XX]="); break;}
 }
if (command==4 || command==5 || command==10 ||
command==11) {XX=read(); mem[PC+3]=(byte)XX;}
 PC+=4;
}
while (command!=12);
}
```

Ολοκληρωμένος κώδικας (4)

```
 public static void help()
 {
System.out.println("Usage:\n");
System.out.println("1.Help = This help text");
System.out.println("2.Execute = Execute Assembly starting from address 0 (PC=0)");
System.out.println("3.Enter source = Enter assembly code in to memory using a instruction
menu");
System.out.println("4.Clear MEM = initialize memory (set all contents to zero");
System.out.println("5.Command list = display instructions with opcodes");
System.out.println("6.Print MEM/REG = display memory and register contents");
System.out.println("0.Exit = exit simulator\n\n");
}
 public static void clear_mem()
 {
for (int i=0;i<=99;i++)
mem[i]=0;
System.out.println("Memory Cleared!!");
}
```

Ολοκληρωμένος κώδικας (5)

```
public static void empty_lines()
      {
         for (byte i=1; i<=100; i++)
            System.out.print("\n");
      }
public static int read()
{
String sample="";
System.out.print(":");
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in) );
Try
{
sample = br.readLine();
}
catch (IOException e){;}
int choice = Integer.parseInt(sample);
return(choice);
      }
   }
```

Για αποφυγή σφαλμάτων
χρησιμοποιήστε τον κώδικα
από το αρχείο simulator.java

Δοκιμή-Χρήση του προσομοιωτή

Δοκιμαστικό πρόγραμμα

Κώδικας

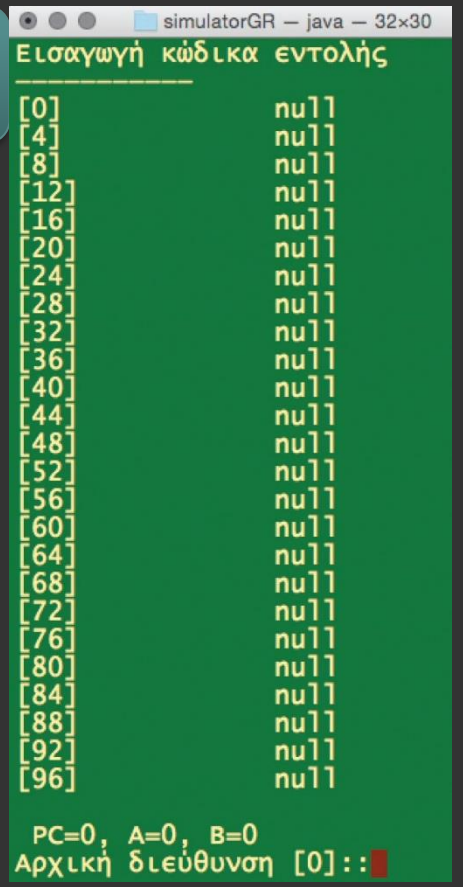
```
MOV A,10          ;Αρχικοποίηση μετρητή βρόχου
again:            ;Σημείο επαναφοράς
        WR                ;Εμφάνιση περιεχομένου μετρητή (καταχωρητής A) στην οθόνη
        DEC A     ;Μείωση τιμής μετρητή (ενημέρωση τιμής καταχωρητή)
JNZ A, again      ;Όσο η τιμή του μετρητή δεν είναι μηδέν, επαναφορά στο σημείο again
STOP              ;Τερματισμός προγράμματος
```

Κωδικοποίηση και λειτουργία

Κώδικας Assembly	Κωδικοποίηση 4 byte	Περιγραφή
MOV A,10	04 00 00 **10**	10 = όρισμα εντολής (A=10)
again:		
WR	02 00 00 00	Εμφάνιση περιεχομένου A στην οθόνη
DEC A	08 00 00 00	Μείωση του A κατά 1
JNZ A, again	10 00 00 **again**	Όσο A<>0 επαναφορά στη θέση again
STOP	12 00 00 00	Τερματισμός προγράμματος

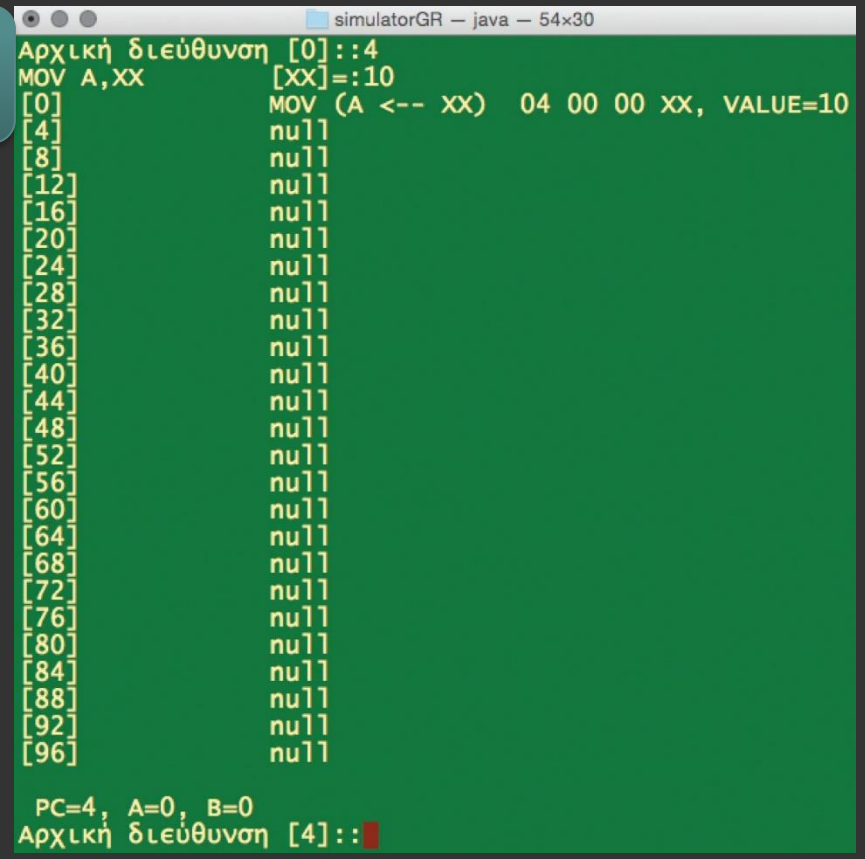
Εισαγωγή κώδικα στον προσομοιωτή (1)

1



- Επιλογή <3>: ΕΙΣΑΓΩΓΗ ΚΩΔΙΚΑ

2



- Εισάγουμε <4> (κώδικας εντολής MOV) και στη συνέχεια το όρισμα
- (εισάγουμε <10>)

Εισαγωγή κώδικα στον προσομοιωτή (2)

3

```
simulatorGR — java — 54×30
 PC=4, A=0, B=0
Αρχική διεύθυνση [4]::2
[0]             MOV (A <-- XX)  04 00 00 XX, VALUE=10
[4]             WR (A --> CON) 02 00 00 00
[8]             null
[12]            null
[16]            null
[20]            null
[24]            null
[28]            null
[32]            null
[36]            null
[40]            null
[44]            null
[48]            null
[52]            null
[56]            null
[60]            null
[64]            null
[68]            null
[72]            null
[76]            null
[80]            null
[84]            null
[88]            null
[92]            null
[96]            null

 PC=8, A=0, B=0
Αρχική διεύθυνση [8]::
```

- Εισάγουμε <2> (κώδικας εντολής WR)

4

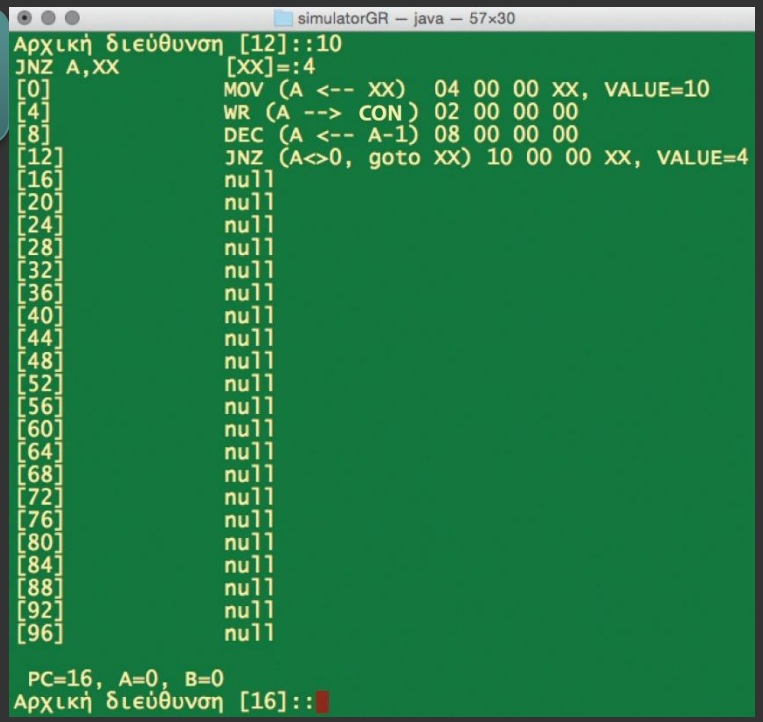
```
simulatorGR — java — 54×30
 PC=8, A=0, B=0
Αρχική διεύθυνση [8]::8
[0]             MOV (A <-- XX)  04 00 00 XX, VALUE=10
[4]             WR (A --> CON) 02 00 00 00
[8]             DEC (A <-- A-1) 08 00 00 00
[12]            null
[16]            null
[20]            null
[24]            null
[28]            null
[32]            null
[36]            null
[40]            null
[44]            null
[48]            null
[52]            null
[56]            null
[60]            null
[64]            null
[68]            null
[72]            null
[76]            null
[80]            null
[84]            null
[88]            null
[92]            null
[96]            null

 PC=12, A=0, B=0
Αρχική διεύθυνση [12]::
```

- Εισάγουμε <8>
- (κώδικας εντολής DEC A)

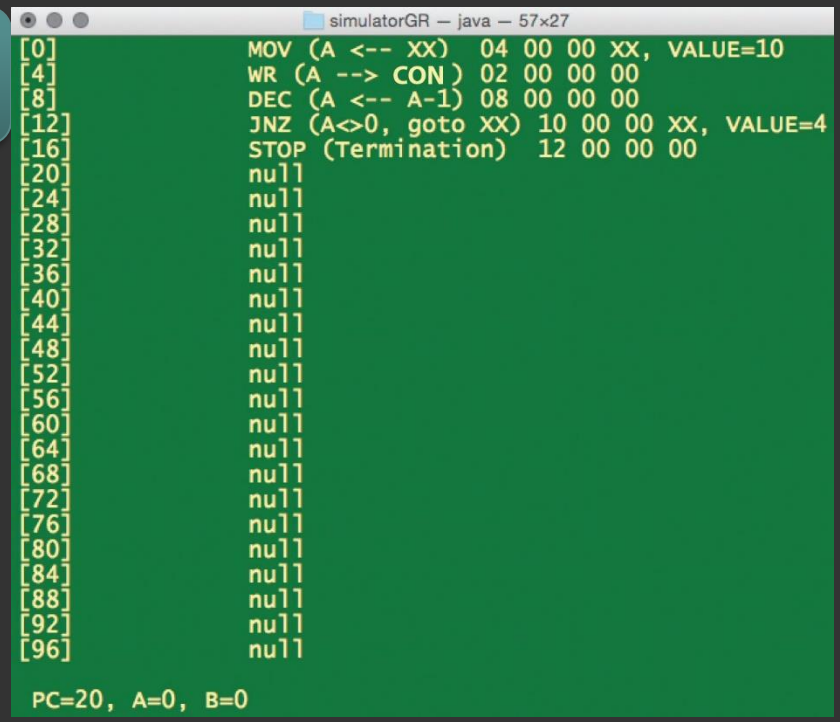
Εισαγωγή κώδικα στον προσομοιωτή (3)

5



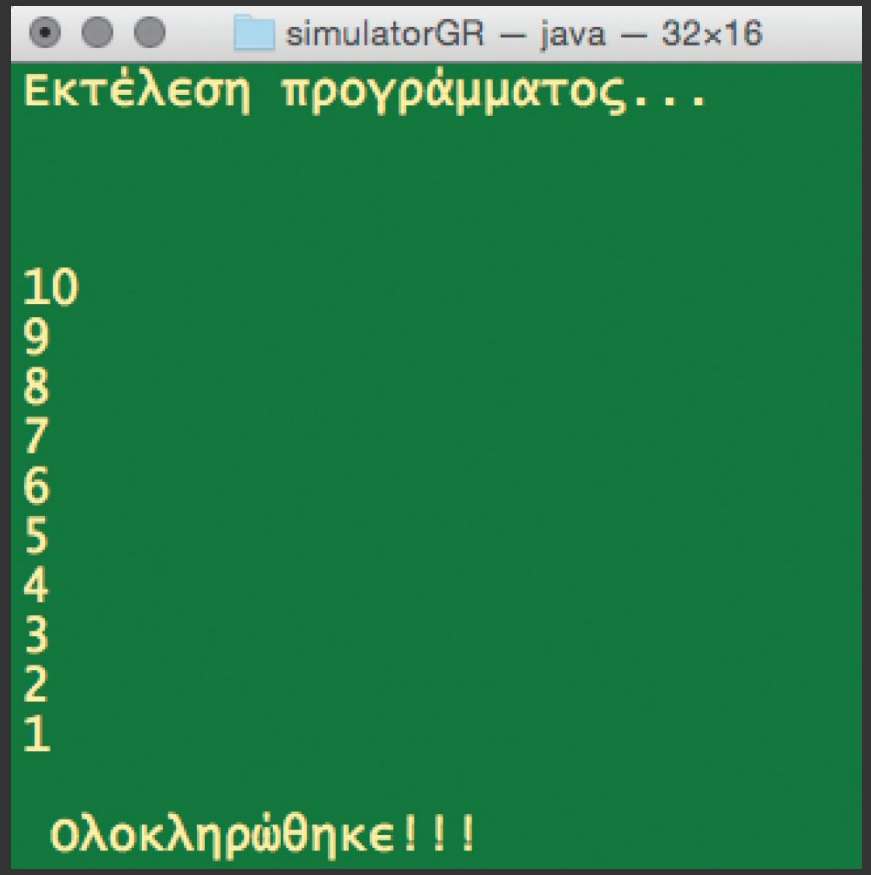
- Εισάγουμε <10> (κώδικας εντολής JNZ) και στη συνέχεια το όρισμα (04)

6



- Εισάγουμε <12>
- (κώδικας εντολής STOP)

Εκτέλεση προγράμματος



Επιλέγουμε <2> (από το κεντρικό μενού)

Εκτέλεση προγράμματος με σφάλμα

```
MOV A,10
again:
        WR
JNZ A, again
STOP
```

Αποτελέσματα στην οθόνη

- Ο χρήστης-προγραμματιστής εισάγει κώδικα με σφάλμα
- Λείπει η εντολή ενημέρωσης του μετρητή (A)
- Ο βρόχος δεν τερματίζεται

- Ο προσομοιωτής εκτελεί «τυφλά» τις εντολές του χρήστη-προγραμματιστή
- Ο βρόχος δεν τερματίζεται (εμφανίζεται πάντα το περιεχόμενο του μετρητή)

Εργασία για το σπίτι

Να γίνουν οι απαραίτητες τροποποιήσεις στον κώδικα simulator.java προκειμένου να υλοποιηθούν τα ακόλουθα ερωτήματα:

(1) Υλοποίηση των εντολών μεταφοράς δεδομένων:

RD (B←KEYB), WR (B→CON)

MOV (A←B) ST ([M]←A)

LD (A←[M])

Υποδείξεις (π.χ. WR (B→CON))

(1) Ενημέρωση της λίστας εντολών **ins[13]="WRB (B --> CON) 13 00 00 00"**

(2) Ενημέρωση κώδικα εκτέλεσης εντολών **case 13: System.out.println(B); break;**

(3) Και ό,τι άλλο απαιτείται συμπληρωματικά

Εργασία για το σπίτι

(2) Υλοποίηση του SREG (Status Register) που θα περιέχει τα παρακάτω flag

C: Carry Flag Z: Zero Flag

N: Negative Flag

V: Two's complement overflow indicator

S: N ⊕ V, For signed tests

Υποδείξεις

(1) Δημιουργία πίνακα SREG[5] με SREG[0]=C και SREG[4]=S

(2) Όταν πρέπει να ενεργοποιηθεί το C, θα πρέπει να εκτελείται η εντολή SREG[0]=1

(3) Εφόσον υλοποιηθεί ξεχωριστή ΑΛΜ, τότε τα Flag θα ενημερώνονται αποκλειστικά από αυτήν

Εργασία για το σπίτι

(3) Υλοποίηση των εντολών αριθμητικών πράξεων με χρήση του SREG

ADD (A←A+B) (πρόσθεση ακέραιων αριθμών 8-bit) SUB (A←A-B) (αφαίρεση ακέραιων αριθμών 8-bit)

MUL (A:B←A*B) (πολλαπλασιασμός ακέραιων αριθμών 8-bit)

DIV (A (ΠΗΛΙΚΟ), B (ΥΠΟΛΟΙΠΟ)←A/B) (διαίρεση ακέραιων αριθμών 8-bit)

Υποδείξεις

(1) Για την ενεργοποίηση του κρατούμενου μετά την πρόσθεση, θα μπορούσε να εκτελείται η εντολή if (A>255) SREG[0]=1

(2) Η δημιουργία νέων εντολών ακολουθεί τις υποδείξεις του βήματος 1

(3) Προσοχή στην ενεργοποίηση των κατάλληλων Flag

(4) Η ενεργοποίηση γίνεται με απλούς ελέγχους, εκτός του S που απαιτεί μια πράξη XOR

Εργασία για το σπίτι

(4) Υλοποίηση των εντολών σύγκρισης με χρήση του SREG

CMP (A, B) (σύγκριση ακέραιων αριθμών 8-bit)

CMP (A, XX) [XX=ακέραια τιμή] (σύγκριση ακέραιων αριθμών 8-bit)

(5) Υλοποίηση των εντολών διακλάδωσης με χρήση του SREG

JNZ k [k=διεύθυνση μνήμης] (Αν το Zero flag είναι Clear τότε ο PC←k) JZ k [k=διεύθυνση μνήμης] (Αν το Zero flag είναι Set τότε ο PC←k)

Υποδείξεις

(1) Στην εντολή σύγκρισης της μορφής CMP (A,B), γίνεται η αφαίρεση A-B. Αν το αποτέλεσμα είναι μηδέν, τότε τα A και B είναι ίσα και ενεργοποιείται το Flag Z (δηλαδή SREG[1]=1)

(2) Γνωρίζετε ήδη πώς δημιουργούμε μια νέα εντολή

(3) Μελετήστε την υλοποιημένη εντολή άλματος και συνδυάστε τον έλεγχο των κατάλληλων Flag

Εργασία για το σπίτι

(6) Υλοποίηση νέας δυνατότητας της εφαρμογής για εκτέλεση εντολών βήμα προς βήμα (step by step) και εμφάνιση στην οθόνη των αντίστοιχων περιεχομένων των καταχωρητών (A, B, SREG: C Z N V S flags, PC)

Υποδείξεις

(1) Για κάθε εντολή που θα εισάγεται, θα γίνεται ερώτηση αν επιθυμεί ο χρήστης Breakpoint

(2) Το Breakpoint θα καταχωρείται σε ένα νέο πίνακα (BPoint) όπως αυτός της μνήμης (δηλαδή Bpoint[PC]=1)

(3) Κάθε φορά που θα εκτελείται μια εντολή, το πρόγραμμα θα ελέγχει αν στην αντίστοιχη θέση PC του πίνακα Bpoint (δηλαδή Bpoint[PC]) υπάρχει η καταχώρηση 1. Αν ναι, τότε θα περιμένει το πρόγραμμα για να πατηθεί ένα πλήκτρο

(4) Τα παραπάνω αντιστοιχούν στην απλή υλοποίηση, ενώ θα πρέπει να βελτιστοποιηθεί τόσο ο πίνακας Bpoint, όσο και ο τρόπος εισαγωγής Breakpoint (να μπορούν να προστεθούν ή να διαγραφούν αργότερα)

Πίνακας mem

Διεύθυνση (θέση)	Περιεχόμενο
00	…
01	…
02	…
03	…
04	…

Πίνακας BPoint

Διεύθυνση (θέση)	Περιεχόμενο
00	01
01	00
02	00
03	00
04	00

Έστω PC=0. Αν Bpoint[PC]=1, τότε αναμονή για πλήκτρο