Електротехноички факултет, Универзитет у Београду

Катедра за рачунарску технику и унформатику

Први домаћи задатак из предмета Системско програмирање

Професор: др. Драган Бојић

Асистент: Саша Стојановић

Студент: Милан Бранковић 119/07

Септембар 2010

Садржај

Опис проблема

Конструисати и имплементирати једнопролазни асемблер за подскуп инструкција хипотетичког рачунара који је дат у прилогу (потребне инструкције су означене жутом бојом у фајлу "Prilog1.doc"). У адресном пољу инструкција и директива могу да се појаве и једноставни изрази који се формирају само помоћу операција сабирања и одузимања и чији операнди могу бити константе и симболи. Излаз треба да буде по школском формату (по потреби извршити одговарајућа прилагођавања формата).

Опис решења

Домаћи задатак је решен у програмском језику JAVA. Комплетан код решења је приложен у фолдеру sp_dz1. Приликом решавања овог проблема уведене су неке претпоставке и на основу њих је израђено комплетно решење:

- програмер који пише програм за HYPO машину дужан је да правилно разграничи свој програм на сегменте:
  - није дозвољено да се инструкције налазе у *dat* или *bss* сегменту,
  - није дозвољено да се дефиниција променљивих нађе у *txt* сегменту,
- иницијализовани подаци се могу дефинисати искључиво у *dat* сегменту,
- неиницијализовани подаци се могу дефинисати искључиво у *bss* сегменту,
- програмер је дужан за синтаксну исправност кода:
  - није дозвољена непотпуна дефиниција неког симбола,
  - након двобајтне инструкције обавезно се налази операнд,
  - није дозвољено да се након једнобајтне инструкције нађе операнд,
- након директиве BEG, а пре дефинисања било ког сегмента, могу опцино да се нађу директиве USE, DEF или EQU
- потребно је сваки сегмент започети одговарајућом директивом, уколико се жели дефинисати тај сегмент
- директива EQU, уколико се користи, може се појавити само једанпут. У адресном делу може се наћи константа или неки израз који је срачунљив у време асемблирања, иначе се неће генерисати исправан излаз
- није дозвољено да се у програму дефинише податак за извоз, и да се резервише нека локација у оквиру истог сегмента

Приликом покретања програма код писан за HYPO машину треба да се прекуца или прекопира у улазни фајл *ulaz.txt*. Програм се састоји од неколико фаза:

1. Најпре се чита линија по линија из улазног фајла и врши се:
   a. лексичка анализа (издвајање токена на текућој линији)
   b. затим се врши синтаксна анализа (издвајање лабела, мнемоника, адресног дела и коментара)
   c. и на крају се врши семантичка анализа (попуњава се табела симбола, обрађују се директиве и попуњава се табела релокација)

2. Након завршеног читања и обраде улазног фајла, врши се испис резултата у излазни фајл у следећем формату (школски формат са малим изменама које су означене црвеном бојом):

```
LINK
nsegs nsyms nrels
# segments    (name base   len     desc)
# symbols     (name value  seg     type)
# relocations (loc  seg    ref     type)   (name  op  place/len)
# data (one line per segment)
```

name означава име симбола или лабеле која се користи на тој локацији, односно представља други бајт инструкције

op означава опеацију коју треба извршити на тој локацији, односно уколико има више операнада у адресном пољу инструкције да ли се симбол додаје или одузима

place/len означава место и меморији за сибмоле који су дефинисани у DAT секцији, односно број локација које сибол заузима за симболе у BSS секцији

Друга измена која је уведена, а која је видљива тек при завршетку рада, је та да се у табели симбола, осим симбола који се увозе и извозе, може налазити и симбол који је дефинисан помоћу EQU директиве. Једина разлика је та што ће дескриптор бити E

Упутство за покретање

Најпре се код написан за HYPO машину прекуца или прекопира у фајл *ulaz.txt* који се налази у фолдеру sp_dz1. Програм се покреће двокликом на икону sp_dz1.jar или из командне линије, тако што ће се укуцати java -jar „путања_до_директоријума\sp_dz1.jar“ Резултати рада ће бити исписани у датотеци *izlaz.txt* у истом фолдеру.

Примери

Напомена: вредности у излазној датотеци су децимални бројеви

Пример1.

Улаз:

```
        BEG
        TXT
        INI             ; Read(A)
LOOP                    ; REPEAT
        SHR             ;  A := A DIV 2
        BCC   EVEN      ;  IF A MOD 2 # 0 THEN
        STA   TEMP      ;    TEMP := A
        LDA   BITS
        INC
        STA   BITS      ;    BITS := BITS + 1
        LDA   TEMP      ;    A := TEMP
EVEN    BNZ   LOOP      ; UNTIL A = 0
        LDA   BITS      ;
        OTI             ; Write(BITS)
        HLT             ; terminate execution
        BSS
TEMP    DS    1         ; VAR TEMP : BYTE
        DAT
BITS    DC    0         ;    BITS : BYTE
      END
```

Излаз:

```
LINK
3 0 7
# segments      (name   base    len     desc)
                .text   0       19      RP
                .bss    19      1       RW
                .data   20      1       RWP
# symbols       (name   value   seg     type)
# relocations   (loc    seg     ref     type)   (name   op      place/len)
                3       1       1       A1      EVEN    +       13
                5       1       2       A1      TEMP    +       1
                7       1       3       A1      BITS    +       20
                10      1       3       A1      BITS    +       20
                12      1       2       A1      TEMP    +       1
                14      1       1       A1      LOOP    +       1
                16      1       3       A1      BITS    +       20
# data (one line per segment)
```

10 22 58 13 30 19 25 20 5 30 20 25 19 55 1 25 20 14 24
0

Пример2.

Улаз:

```
;=====================================
;  ucitava 2 broja A i B, smesta ih u memoriju i ispisuje redom:
;  A+B
;  A-B
;=====================================
                BEG
                DEF    START
;=====================================
   BSS
   DS   2 ; reserve locations for 2 numbers
;=====================================
                TXT
START
   CLX
   ; ucitavanje
                INI
                STA   0
   INI
   STA   1

   ; zbir
   LDI   0
   ADD   1
   OTI

   ; razlika
   LDI   0
   SUB   1
   OTI

                HLT
   END
```

Излаз:

```
LINK
2 1 0
# segments      (name  base   len     desc)
                .text  2      18      RP
                .bss   0      2       RW
# symbols       (name  value  seg     type)
                START  2      1       D
# relocations   (loc   seg    ref     type)   (name  op      place/len)
# data (one line per segment)
3 10 30 0 10 30 1 27 0 32 1 14 27 0 38 1 14 24
```

Пример3.

Улаз:

```
BEG
DEF WRITE,TEMP      ; exported symbols
TXT
WRITE   OTI              ; Write(BITS)
RET             ; return to main program
BSS
TEMP    DS      1        ; VAR TEMP : BYTE
END
```

Излаз:

```
LINK
2 2 0
# segments      (name  base   len    desc)
                .text  0      2      RP
                .bss   2      1      RW
# symbols       (name  value  seg    type)
                TEMP   2      2      D
                WRITE  0      1      D
# relocations   (loc   seg    ref    type)   (name  op      place/len)
# data (one line per segment)
14 23
```

Пример4.

Улаз:

```
BEG          ; count the bits in a number
USE WRITE,TEMP ; external symbols
TXT          ; default, even if not here
INI            ; Read(A)
LOOP                ; REPEAT
SHR            ;  A := A DIV 2
BCC    EVEN       ;  IF A MOD 2 # 0 THEN
STA    TEMP       ;    TEMP := A
LDA    BITS
INC
STA    BITS       ;    BITS := BITS + 1
LDA    TEMP       ;    A := TEMP
EVEN   BNZ    LOOP       ; UNTIL A = 0
LDA    BITS       ;
JSR    WRITE      ; Write(BITS)
HLT            ; terminate execution
DAT
BITS   DC     0          ;     BITS : BYTE
END
```

Излаз:

```
LINK
2 2 8
# segments      (name  base  len    desc)
                .text  0     20     RP
                .data  20    1      RWP
# symbols       (name  value seg    type)
                TEMP   0     0      U
                WRITE  0     0      U
# relocations   (loc   seg   ref    type)  (name  op    place/len)
                3      1     1      A1     EVEN   +     13
                5      1     1      AS1    TEMP   +     0
                7      1     3      A1     BITS   +     20
                10     1     3      A1     BITS   +     20
                12     1     1      AS1    TEMP   +     0
                14     1     1      A1     LOOP   +     1
                16     1     3      A1     BITS   +     20
                18     1     2      AS1    WRITE  +
# data (one line per segment)
10 22 58 13 30 0 25 20 5 30 20 25 0 55 1 25 20 60 0 24
0
```

Пример5.

Улаз:

```
;=====================================
;  Provera steka
;  ucitava 3 broja i ispisuje ih u inverznom redosledu
;=====================================
                BEG
                DEF   START
                TXT
;=====================================
START
                INI    ; read A
                PSH
                INI    ; read B
                PSH
                INI    ; read C
                PSH
                POP
                OTI    ; write C
                POP
                OTI    ; write B
                POP
                OTI    ; write A
                HLT
  END
```

Излаз:

```
LINK
1 1 0
# segments      (name  base   len    desc)
                .text  0      13     RP
# symbols       (name  value  seg    type)
                START  0      1      D
# relocations   (loc   seg    ref    type)   (name  op      place/len)
# data (one line per segment)
10 19 10 19 10 19 20 14 20 14 20 14 24
```

Пример6.

Улаз:

```
        BEG
TXT
        PSH
    LDA BLA
    PSH
    BCS LABELA1
    PSH
LABELA1 PSH
    ADD BLA
    BCS LABELA2
    ADD BLA
LABELA2 BNZ LABELA1
    HLT
    END
```

Излаз:

```
LINK
1 0 6
# segments      (name  base  len   desc)
                .text  0     17    RP
# symbols       (name  value seg   type)
# relocations   (loc   seg   ref   type)  (name  op  place/len)
                2      1     0     A1      BLA    +
                5      1     1     A1      LABELA1      +     7
                9      1     0     A1      BLA    +
                11     1     1     A1      LABELA2      +     14
                13     1     0     A1      BLA    +
                15     1     1     A1      LABELA1      +     7
# data (one line per segment)
19 25 0 19 59 7 19 19 32 0 59 14 32 0 55 7 24
```

Пример7.

Улаз:

```
;=====================================
;  Provera ulaza/izlaza
;  Ucitava broj -> Ispisuje broj
;=====================================
                BEG
                DEF    START
                TXT
;=====================================
START
                INI
                OTI
                HLT
   END
```

Излаз:

```
LINK
1 1 0
# segments      (name   base    len     desc)
                .text   0       3       RP
# symbols       (name   value   seg     type)
                START   0       1       D
# relocations   (loc    seg     ref     type)   (name   op      place/len)
# data (one line per segment)
10 14 24
```

Пример8.

Улаз:

```
BEG
DEF START
TXT
START INI
loop STA nnn
LDI suma
ADD nnn
STA suma
LDI nnn
DEC
BNZ loop
LDI suma
OTI
HLT
DAT
nnn DC 0
suma DC 0
END
```

Излаз:

```
LINK
2 1 7
# segments      (name   base    len     desc)
                .text   0       18      RP
                .data   18      2       RWP
# symbols       (name   value   seg     type)
                START   0       1       D
# relocations   (loc    seg     ref     type)   (name   op      place/len)
                2       1       3       A1      nnn     +       18
                4       1       3       A1      suma    +       19
                6       1       3       A1      nnn     +       18
                8       1       3       A1      suma    +       19
                10      1       3       A1      nnn     +       18
                13      1       1       A1      loop    +       1
                15      1       3       A1      suma    +       19
# data (one line per segment)
10 30 18 27 19 32 18 30 19 27 18 6 55 1 27 19 14 24
0 0
```

Пример9.

Улаз:

```
    BEG
USE LAB
DEF B
USE LAB1,LAB2
M EQU LAB+91
TXT
          LDI B+190-TEMP
LOOP            SHR
      BCC EVEN
      STA BITS-TEMP+10
      LDA BITS-5
      INC
      STA LAB1-LAB
      LDA LAB2
EVEN      BNZ LOOP
      LDA 10-TEMP-M-EVEN
      OTI
      HLT
      DAT
TEMP   DC 150-BITS-M
BITS   DC LAB1-B-BITS-TEMP+M
          DC 20
      BSS
B         DS 1
      END
```

Излаз:

```
LINK
3 5 21
# segments      (name   base    len     desc)
                .text   0       20      RP
                .bss    23      1       RW
                .data   20      3       RWP
# symbols       (name   value   seg     type)
                M       91      0       E
                LAB2    0       0       U
                LAB1    0       0       U
                LAB     0       0       U
                B       23      2       D
# relocations   (loc    seg     ref     type)   (name   op      place/len)
                0       0       4       AS1     LAB     +
                1       1       2       A1      B       +       1
                1       1       3       A1      TEMP    -       20
```

4	1	1	A1	EVEN	+	14
6	1	3	A1	BITS	+	21
6	1	3	A1	TEMP	-	20
8	1	3	A1	BITS	+	21
11	1	3	AS1	LAB1	+	
11	1	4	AS1	LAB	-	
13	1	2	AS1	LAB2	+	
15	1	1	A1	LOOP	+	2
17	1	3	A1	TEMP	-	20
17	1	1	AS1	M	-	91
17	1	1	A1	EVEN	-	14
20	3	3	A1	BITS	-	21
20	3	1	AS1	M	-	
21	3	3	AS1	LAB1	+	0
21	3	2	A1	B	-	1
21	3	3	A1	BITS	-	21
21	3	3	A1	TEMP	-	20
21	3	1	AS1	M	+	

data (one line per segment)
27 193 22 58 14 30 11 25 16 5 30 0 25 0 55 2 25 -115 14 24
38 27 20

Пример10.

Улаз:

```
        BEG
TXT
        INI
        STA A
        INI
        STA B
        JSR SABERI
        LDA ZBIR
        OTI
        HLT

SABERI  LDA A
        ADD B
        STA ZBIR
        RET

BSS
ZBIR DS 1
DAT
A DC 0
B DC 0
        END
```

Излаз:

```
LINK
3 0 7
# segments      (name   base    len     desc)
                .text   0       19      RP
                .bss    19      1       RW
                .data   20      2       RWP
# symbols       (name   value   seg     type)
# relocations   (loc    seg     ref     type)   (name   op      place/len)
                2       1       3       A1      A       +       20
                5       1       3       A1      B       +       21
                7       1       1       A1      SABERI  +       12
                9       1       2       A1      ZBIR    +       1
                13      1       3       A1      A       +       20
                15      1       3       A1      B       +       21
                17      1       2       A1      ZBIR    +       1
# data (one line per segment)
10 30 20 10 30 21 60 12 25 19 14 24 25 20 32 21 30 19 23
0 0
```

Пример11.

Улаз:

```
        BEG
TXT
        PSH
    LDA BLA
    PSH
    JSR POTPR
    PSH
LABELA1 PSH
    ADD BLA
    BCS LABELA2
    ADD BLA
LABELA2 BNZ LABELA1
    HLT

POTPR   PSH
    LDA BLA
    BNZ LABELA3
LABELA3 RET

DAT
BLA     DC 0

    END
```

Излаз:

```
LINK
2 0 8
# segments      (name   base    len     desc)
                .text   0       23      RP
                .data   23      1       RWP
# symbols       (name   value   seg     type)
# relocations   (loc    seg     ref     type)   (name   op      place/len)
                2       1       3       A1      BLA     +       23
                5       1       1       A1      POTPR   +       17
                9       1       3       A1      BLA     +       23
                11      1       1       A1      LABELA2 +       14
                13      1       3       A1      BLA     +       23
                15      1       1       A1      LABELA1 +       7
                19      1       3       A1      BLA     +       23
                21      1       1       A1      LABELA3 +       22
# data (one line per segment)
19 25 23 19 60 17 19 19 32 23 59 14 32 23 55 7 24 19 25 23 55 22 23
0
```

Пример16.

Улаз:

```
        BEG
        TXT
        LDI 20+20
        OTI
LAB1    CLC
        ADD LAB1
        HLT
        END
```

Излаз:

```
LINK
1 0 1
# segments      (name   base    len     desc)
                .text   0       7       RP
# symbols       (name   value   seg     type)
# relocations   (loc    seg     ref     type)   (name   op      place/len)
                5       1       1       A1      LAB1    +       3
# data (one line per segment)
27 40 14 2 32 3 24
```

Листинг програма

AS.java

```
package sp_dz1;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.LinkedList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author MB
 */
public class AS {

    private SA syx;
    private TableSymbols table;
    private TableOperation tableOP;
    private TableDirective tableDir;
    private TableRelocations tableRel;
    private static int location;
    private static int nsegs;
    private static int nsyms;
    private static int nrels;
    private static int prevSeg;
    private boolean assembling;
    private static int mem[];
    private int seg;

    private static String[] opForDefinedSymbols;
    private static int place;

    private static int lenSymbInBss[];
    private static int bssPlace;

    public AS(String fileName) {
        table = new TableSymbols();
        tableOP = new TableOperation();
        tableDir = new TableDirective();
        tableRel = new TableRelocations();
        syx = new SA(fileName);
        mem = new int[256];
        seg = 0;
        nsegs = nsyms = nrels = 0;
        prevSeg = 0;

        opForDefinedSymbols = new String[20];
```

```
        place = 0;

        lenSymbInBss = new int[10];
        bssPlace = 0;
    }

    public static void insertNewOp(String op){
        opForDefinedSymbols[place++] = op;
    }

    public void assemble() {
        System.out.println("Assembling...");
        firstPass();
        saveObjCode();
        System.out.println("Assembling complete!");
    }

    public void firstPass() {
        assembling = true;
        location = 0;

        syx.SA_analysis();
        LA.LA_sym symb;
        LinkedList<LA.LA_sym> lista = syx.getLista();
        int indx = 0;
        while (assembling) {
            symb = lista.get(indx);
            assembleLine(symb);
            indx++;
        }
        table.backpatch(mem, tableRel);
        //table.printsymboltable();
    }

    public void assembleLine(LA.LA_sym symb) {
        String op = symb.getMnem();
        if (TableOperation.isDirective(op)) // directives
        {
            if ("BEG".equalsIgnoreCase(op)) {
                location = 0;
            } else if ("END".equalsIgnoreCase(op)) {
                int base = tableDir.getListZapis().get(seg - 1).getBase();
                tableDir.getListZapis().get(seg - 1).setLen(location -
base);
                assembling = false;
            } else if ("ORG".equalsIgnoreCase(op)) {
                location = Integer.parseInt(symb.getAdr());
            } else if ("DS".equalsIgnoreCase(op)) {
                if (symb.isIsLabeled()) {
                    table.enter(symb.getLabel(), location, 2, false,
false);
                }
```

```
                int forAdd = evaluate(symb.getAdr(),
symb.getListSymbInAdrField(), symb.getListOpInAdrField());
                lenSymbInBss[bssPlace++] = location;
                lenSymbInBss[bssPlace++] = forAdd;
                location = location + forAdd;
            } else if ("EQU".equalsIgnoreCase(symb.getMnem())) {
                nsyms++;
                if (isNumber(symb.getAdr())) {
                    table.enter(symb.getLabel(),
Integer.parseInt(symb.getAdr()), 0, true, true);
                }else {
                    if (hasMoreThanOneOperand(symb)) {
                        table.enter(symb.getLabel(), 0, seg, true,
true);//evaluate(symb.getAdr(), symb.getListSymbInAdrField(),
symb.getListOpInAdrField())
                        int size = symb.getSizeListAdrSym();
                        for (int i = 0; i < size; i++) {
                            if
(!isNumber(symb.getListSymbInAdrField().get(i))) {
                                if
(!table.isGlobusedSymb(symb.getListSymbInAdrField().get(i))) {

tableRel.add(symb.getListSymbInAdrField().get(i), location, seg, false);
                                } else {

tableRel.add(symb.getListSymbInAdrField().get(i), location, seg, 0, true);
//table.numGlobusedSymbInTable(symb.getListSymbInAdrField().get(i))
                                }
                                evaluate(symb.getAdr(),
symb.getListSymbInAdrField(), symb.getListOpInAdrField());
                                nrels++;
                            } else {
                                if(symb.getListOpInAdrField().get(i -
1).equalsIgnoreCase("+")){
                                    table.enter(symb.getLabel(),
table.getSymbol(symb.getLabel()).getValue() +
Integer.parseInt(symb.getListSymbInAdrField().get(i)), 0, true, true);
                                }else if(symb.getListOpInAdrField().get(i -
1).equalsIgnoreCase("-")){
                                    table.enter(symb.getLabel(),
table.getSymbol(symb.getLabel()).getValue() -
Integer.parseInt(symb.getListSymbInAdrField().get(i)), 0, true, true);
                                }
                            }
                        }
                    } else {
                        if (!table.isGlobusedSymb(symb.getAdr())) {
                            tableRel.add(symb.getAdr(), location, seg,
false);
                            table.enter(symb.getLabel(),
evaluate(symb.getAdr(), symb.getListSymbInAdrField(),
symb.getListOpInAdrField()), seg, true, true);
                        } else {
```

```
                            table.enter(symb.getLabel(), location, seg,
true, true);
                            tableRel.add(symb.getAdr(), location, seg,
table.numGlobusedSymbInTable(symb.getLabel()),true);
                        }
                        nrels++;
                    }
                }

                } else  if ("TXT".equalsIgnoreCase(symb.getMnem())) {
                seg = 1;
                nsegs++;
                tableDir.getListZapis().get(seg - 1).setBase(location);
                if (prevSeg != 0) {
                    int base = tableDir.getListZapis().get(prevSeg -
1).getBase();
                    tableDir.getListZapis().get(prevSeg -
1).setLen(location - base);
                }
                prevSeg = seg;
            } else if ("BSS".equalsIgnoreCase(symb.getMnem())) {
                seg = 2;
                nsegs++;
                tableDir.getListZapis().get(seg - 1).setBase(location);
                if (prevSeg != 0) {
                    int base = tableDir.getListZapis().get(prevSeg -
1).getBase();
                    tableDir.getListZapis().get(prevSeg -
1).setLen(location - base);
                }
                prevSeg = seg;
            } else if ("DAT".equalsIgnoreCase(symb.getMnem())) {
                seg = 3;
                nsegs++;
                tableDir.getListZapis().get(seg - 1).setBase(location);
                if (prevSeg != 0) {
                    int base = tableDir.getListZapis().get(prevSeg -
1).getBase();
                    tableDir.getListZapis().get(prevSeg -
1).setLen(location - base);
                }
                prevSeg = seg;
            } else if ("USE".equalsIgnoreCase(symb.getMnem())) {
                int size = symb.getSizeListAdrSym();
                if (size > 0) {
                    for (int i = 0; i < size; i++) {
                        table.enter(symb.getListSymbInAdrField().get(i), 0,
0, true, false);
                    }
                    nsyms += size;
                } else {
                    table.enter(symb.getAdr(), 0, 0, true, false);
                    nsyms++;
```

```
                }
            } else if ("DEF".equalsIgnoreCase(symb.getMnem())) {
                int size = symb.getSizeListAdrSym();
                if (size > 0) {
                    for (int i = 0; i < size; i++) {
                        table.enter(symb.getListSymbInAdrField().get(i), 0,
0, false, true);
                    }
                    nsyms += size;
                } else {
                    table.enter(symb.getAdr(), 0, 0, false, true);
                    nsyms++;
                }
            }
        } else // machine ops
        {

            if (symb.isIsLabeled()) {
                table.enter(symb.getLabel(), location, seg, false, false);
            }
            mem[location] = TableOperation.getValueDec(symb.getMnem());
            if (TableOperation.twoBytesInstr(symb.getMnem())) // TwoByteOps
            {
                if(!symb.getMnem().equalsIgnoreCase("DC"))location++;
                mem[location] = evaluate(symb.getAdr(),
symb.getListSymbInAdrField(), symb.getListOpInAdrField());
                if (!symb.getMnem().equalsIgnoreCase("DS") &&
!symb.getMnem().equalsIgnoreCase("EQU")
                        && !isNumber(symb.getAdr())) {
                    if (hasMoreThanOneOperand(symb)) {
                        int size = symb.getSizeListAdrSym();
                        for (int i = 0; i < size; i++) {
                            if
(!isNumber(symb.getListSymbInAdrField().get(i))) {
                                if
(!table.isGlobusedSymb(symb.getListSymbInAdrField().get(i))) {

tableRel.add(symb.getListSymbInAdrField().get(i), location, seg, false);
                                } else {

tableRel.add(symb.getListSymbInAdrField().get(i), location, seg,
table.numGlobusedSymbInTable(symb.getListSymbInAdrField().get(i)), true);
                                }
                                nrels++;
                            }
                        }
                    } else {
                        if (!table.isGlobusedSymb(symb.getAdr())) {
                            tableRel.add(symb.getAdr(), location, seg,
false);
                        } else {
                            tableRel.add(symb.getAdr(), location, seg,
table.numGlobusedSymbInTable(symb.getAdr()), true);
```

```
                        }
                        nrels++;
                    }
                }
            }

            if (!symb.getMnem().equalsIgnoreCase("")
                    && !symb.getMnem().equalsIgnoreCase(" ")) {
                location++;
            }
        }
    }

    public void saveObjCode() {
        BufferedWriter izlaz = null;
        try {
            izlaz = new BufferedWriter(new FileWriter("izlaz.txt"));
        } catch (IOException ex) {
            System.out.println("Neuspesno kreiranje izlaznog fajla!");
            Logger.getLogger(AS.class.getName()).log(Level.SEVERE, null,
ex);
        }

        try {
            izlaz.write("LINK");
            izlaz.newLine();
            izlaz.write(nsegs + " " + nsyms + " " + nrels);
            izlaz.newLine();
            izlaz.write("# segments\t(name\tbase\tlen\tdesc)\n");
            izlaz.newLine();
            for (int i = 0; i < 3; i++) {
                if (tableDir.isDefined(i)) {
                    izlaz.write(tableDir.getZapisString(i));
                    izlaz.newLine();
                }
            }
            izlaz.write("# symbols\t(name\tvalue\tseg\ttype)\n");
            izlaz.newLine();
            for(int i = 0; i < table.getSize(); i++) {
                if (table.getListsym().get(i).isGlobdef() &&
table.getListsym().get(i).isGlobuse()) {
                    izlaz.write(table.getEntryString(i));
                    izlaz.newLine();
                }
            }
            for (int i = 0; i < table.getSize(); i++) {
                if (table.getListsym().get(i).isGlobuse() &&
!table.getListsym().get(i).isGlobdef()) {
                    izlaz.write(table.getEntryString(i));
                    izlaz.newLine();
                }
            }
            for(int i = 0; i < table.getSize(); i++) {
```

```
                if (table.getListsym().get(i).isGlobdef() &&
!table.getListsym().get(i).isGlobuse()) {
                    izlaz.write(table.getEntryString(i));
                    izlaz.newLine();
                }
            }
            izlaz.write("#
relocations\t(loc\tseg\tref\ttype)\t(name\top\tplace/len)");
            izlaz.newLine();
            int defOp = 0;
            for (int i = 0; i < nrels; i++) {
                izlaz.write(tableRel.getZapisString(i));
                if
(table.getOpType(tableRel.getListZapis().get(i).getName(),
tableRel.getListZapis().get(i).getLoc()).equalsIgnoreCase("+")
                        ||
table.getOpType(tableRel.getListZapis().get(i).getName(),
tableRel.getListZapis().get(i).getLoc()).equalsIgnoreCase("-")) {

izlaz.write(table.getOpType(tableRel.getListZapis().get(i).getName(),
tableRel.getListZapis().get(i).getLoc()));
                    if (tableRel.getListZapis().get(i).getRef() == 2 &&
!tableRel.getListZapis().get(i).getType().contains("S")) {
                        for (int j = 0; j < lenSymbInBss.length; j += 2) {
                            if
(table.getSymbol(tableRel.getListZapis().get(i).getName()).getValue() ==
lenSymbInBss[j]) {
                                izlaz.write("\t" + lenSymbInBss[j + 1]);
                            }
                        }
                    }
                    if (tableRel.getListZapis().get(i).getRef() == 3 &&
!tableRel.getListZapis().get(i).getType().contains("S")) {
                        izlaz.write("\t" +
table.getSymbol(tableRel.getListZapis().get(i).getName()).getValue());
                    }

                    if(tableRel.getListZapis().get(i).getRef() ==
tableRel.getListZapis().get(i).getSeg()){
                        for (int j = 0; j < table.getSize(); j++) {

if(tableRel.getListZapis().get(i).getName().equalsIgnoreCase(table.getLists
ym().get(j).getName())){
                                izlaz.write("\t" +
table.getListsym().get(j).getValue());
                                break;
                            }
                        }
                    }
                } else {
                    if (place > 0) {
                        izlaz.write(opForDefinedSymbols[defOp]);
                        defOp++;
```

```
                        if (tableRel.getListZapis().get(i).getRef() ==
tableRel.getListZapis().get(i).getSeg()) {
                            for (int j = 0; j < table.getSize(); j++) {
                                if
(tableRel.getListZapis().get(i).getName().equalsIgnoreCase(table.getListsym
().get(j).getName())) {
                                    izlaz.write("\t" +
table.getListsym().get(j).getValue());
                                    break;
                                }
                            }
                        } else {
                            izlaz.write("+");
                        }

                    }
                }
                izlaz.newLine();
            }
            izlaz.write("# data (one line per segment)");
            izlaz.newLine();

            int base = tableDir.getListZapis().get(0).getBase(),
                    len = base + tableDir.getListZapis().get(0).getLen();
            for (int i = base; i < len; i++) {
                izlaz.write(mem[i] + " ");
            }
            izlaz.newLine();
            base = tableDir.getListZapis().get(2).getBase();
            len = base + tableDir.getListZapis().get(2).getLen();
            for (int i = base; i < len; i++) {
                izlaz.write(mem[i] + " ");
            }
            izlaz.newLine();

        } catch (IOException ex) {
            System.out.println("Neuspesan upis u izlaznu datoteku!");
            Logger.getLogger(AS.class.getName()).log(Level.SEVERE, null,
ex);
        }
        try {
            izlaz.close();
        } catch (IOException ex) {
            System.out.println("Neuspesno zatvaranje izlaznog fajla!");
            Logger.getLogger(AS.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }

    private boolean isNumber(String num) {
        char arr[] = num.trim().toCharArray();
        boolean res = true;
        for (int i = 0; i < arr.length; i++) {
```

```
            res = res && (arr[i] >= '0' && arr[i] <= '9');
        }
        return res;
    }

    private boolean hasMoreThanOneOperand(LA.LA_sym symb) {
        return symb.getSizeListAdrSym() > 0;
    }

    private int evaluate(String adr, List<String> listSymb, List<String>
listOp) {
        int forRet = 0;
        if (listSymb.size() > 0) {
            int ind = 0;
            while (ind < listSymb.size()) {
                if (isNumber(listSymb.get(ind))) {
                    if (ind == 0) {
                        forRet += Integer.parseInt(listSymb.get(ind));
                    } else {
                        if (listOp.get(ind - 1).equals("+")) {
                            forRet += Integer.parseInt(listSymb.get(ind));
                        } else if (listOp.get(ind - 1).equals("-")) {
                            forRet -= Integer.parseInt(listSymb.get(ind));
                        }
                    }

                } else {
                    if (ind == 0) {
                        forRet += table.valueOfSymbol(listSymb.get(ind),
location, "+");
                    } else {
                        if("+".equalsIgnoreCase(listOp.get(ind - 1)))
                            forRet +=
table.valueOfSymbol(listSymb.get(ind), location, listOp.get(ind - 1));
                        if("-".equalsIgnoreCase(listOp.get(ind - 1)))
                            forRet -=
table.valueOfSymbol(listSymb.get(ind), location, listOp.get(ind - 1));
                    }
                }
                ind++;
            }
            return forRet;
        } else{
            if (isNumber(adr)) {
                return Integer.parseInt(adr);
            }
            if (table.isDefinedSymb(adr)) {
                //return table.valueOfSymbol(adr, location, "+");
                insertNewOp("+");
                return
table.getListsym().get(table.findEntry(adr)).getValue();
            }
            table.valueOfSymbol(adr, location, "+");
```

```
            return 0;
        }
    }
}
```

LA.java

```
/*
 * klasa za leksicku analizu
 */
package sp_dz1;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.LinkedList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author MB
 */
public class LA {

    public class LA_sym {

        private boolean isLabeled;
        private String label;
        private String mnem;
        private String adr;
        private String comment;
        private List<String> listSymbInAdrField;
        private List<String> listOpInAdrField;

        public LA_sym() {
            label = mnem = adr = comment = "";
            isLabeled = false;
            listOpInAdrField = new LinkedList<String>();
            listSymbInAdrField = new LinkedList<String>();
        }

        public String getAdr() {
            return adr;
        }

        public String getComment() {
            return comment;
        }

        public boolean isIsLabeled() {
            return isLabeled;
        }

        public String getLabel() {
            return label;
        }
```

```
        public String getMnem() {
            return mnem;
        }

        public void setAdr(String adr) {
            this.adr = adr;
        }

        public void setComment(String comment) {
            this.comment = comment;
        }

        public void setIsLabeled(boolean isLabeled) {
            this.isLabeled = isLabeled;
        }

        public void setLabel(String label) {
            this.label = label;
        }

        public void setMnem(String mnem) {
            this.mnem = mnem;
        }

        public int getSizeListAdrSym(){
            return listSymbInAdrField.size();
        }

        public int getSizeListAdrOp(){
            return listOpInAdrField.size();
        }

        public List<String> getListOpInAdrField() {
            return listOpInAdrField;
        }

        public List<String> getListSymbInAdrField() {
            return listSymbInAdrField;
        }

    }

    public boolean analiseAdrField(String s, LA_sym la) {
        if (s.contains("+") || s.contains("-")) {
            while (true) {
                int plus = s.indexOf("+"), minus = s.indexOf("-");
                if (plus > minus) {
                    if (minus > 0) {
                        la.getListSymbInAdrField().add(s.substring(0,
s.indexOf("-")));
                        la.getListOpInAdrField().add("-");
                        s = s.substring(s.indexOf("-") + 1, s.length());
```

```
                    } else {
                        la.getListSymbInAdrField().add(s.substring(0,
s.indexOf("+")));
                        la.getListOpInAdrField().add("+");
                        s = s.substring(s.indexOf("+") + 1, s.length());
                    }
                } else if (minus > plus) {
                    if (plus > 0) {
                        la.getListSymbInAdrField().add(s.substring(0,
s.indexOf("+")));
                        la.getListOpInAdrField().add("+");
                        s = s.substring(s.indexOf("+") + 1, s.length());
                    } else {
                        la.getListSymbInAdrField().add(s.substring(0,
s.indexOf("-")));
                        la.getListOpInAdrField().add("-");
                        s = s.substring(s.indexOf("-") + 1, s.length());
                    }
                }
                    if (!s.contains("+") && !s.contains("-")) {
                        s.trim();
                        if (!s.equalsIgnoreCase("") &&
!s.equalsIgnoreCase(" ")) {
                            la.getListSymbInAdrField().add(s);
                        }
                        return true;
                    }

            }
        }
        if (s.contains(",")) {
            while (true) {
                la.getListSymbInAdrField().add(s.substring(0,
s.indexOf(",")));
                s = s.substring(s.indexOf(",") + 1, s.length());
                if (!s.contains(",")) {
                    s.trim();
                    if (!s.equalsIgnoreCase("") && !s.equalsIgnoreCase("
")) {
                        la.getListSymbInAdrField().add(s);
                    }
                    return true;
                }
            }
        }

        return false;

    }

    //analysis the line
    public LA_sym LA_analysis(String linija) {
        String line_to_read = linija;
```

```
        line = line_to_read;
        linelength = line.length();

        LA_sym la_sym_line = new LA_sym();
        /*
        Leksička analiza odnosi se grupisanje tokena na tekućoj liniji, u
identifikatore/mnemonike, konstante, komentare itd.
        Sintaksna analiza odnosi se na izdvajanje polja labele, polja
direktive/opkoda i adresnog polja sa linije (upisuju se odvojeno u radni
fajl da drugi prolaz ne bi opet morao da sprovodi leksičku analizu).
        Semantička analiza odnosi se na popunjavanje tabele simbola, obradu
direktiva i proveru korektnosti izvornog programa i prijavu grešaka.

         */
        String komentar = "";
        int comment = line_to_read.indexOf(";");
            if (comment != -1) {
                komentar = line_to_read.substring(comment,
line_to_read.length());
                la_sym_line.setComment(komentar);

                line_to_read = line_to_read.substring(0, comment);
            }

        String[] list = line_to_read.split("[\\s]+");

        for(int i = 0; i<list.length; i++) {

            if(TableOperation.oneByteInstr(list[i]) ||
TableOperation.twoBytesInstr(list[i])){
                la_sym_line.setMnem(list[i]);
            }
            else {
                if(i == 0 && !list[i].equalsIgnoreCase("") &&
!list[i].equalsIgnoreCase(" ")){
                    la_sym_line.setLabel(list[i]);
                    la_sym_line.setIsLabeled(true);
                } else{
                    if(!list[i].equalsIgnoreCase("") &&
!list[i].equalsIgnoreCase(" ")){
                        if(!analiseAdrField(list[i], la_sym_line))
                            la_sym_line.setAdr(list[i]);
                        else la_sym_line.setAdr(list[i]);
                    }
                }
            }
        }

        return la_sym_line;
    }

    public LA(String file) {
```

```
        try {
            src = new BufferedReader(new FileReader(file));
        } catch (FileNotFoundException ex) {
            System.out.print("Nije pronadjen fajl: " + file + " \n");
            Logger.getLogger(LA.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }

    BufferedReader src;     // source file
    int linelength;         // line length
    String line;            // last line read
}
```

Main.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sp_dz1;

/**
 *
 * @author MB
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        AS as = new AS("ulaz.txt");
        as.assemble();
    }

}
```

SA.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package sp_dz1;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.LinkedList;
import java.util.logging.Level;
import java.util.logging.Logger;
import sp_dz1.LA.LA_sym;

/**
 *
 * @author MB
 */
public class SA { //sintaksna analiza

    private LinkedList<LA.LA_sym> lista;
    private LA lex;
    private BufferedReader src;
    private String fileName;

    public SA(String file) {
        lista = new LinkedList<LA_sym>();
        lex = new LA(file);
        fileName = file;
        try {
            src = new BufferedReader(new FileReader(file));
        } catch (FileNotFoundException ex) {
            System.out.print("Nije pronadjen fajl:" + file + "\n");
            Logger.getLogger(SA.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }

    public LinkedList<LA_sym> getLista() {
        return lista;
    }

    public String getFileName() {
        return fileName;
    }

    public void SA_analysis() {
        String line = null;
        while (true) {
            try {
```

```
                line = src.readLine();

            } catch (IOException ex) {
                System.out.println("Neuspesno citanje linije!");
                Logger.getLogger(SA.class.getName()).log(Level.SEVERE,
null, ex);
            }
            if (line != null) {
                lista.addLast(lex.LA_analysis(line));
            } else break;
        }
        try {
            src.close();
        } catch (IOException ex) {
            System.out.println("Neuspesno zatvaranje ulaznog fajla!");
            Logger.getLogger(SA.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }

    public void ispis(){
        for(int i = 0; i<lista.size(); i++){
            System.out.println(lista.get(i).getLabel() + " " +
lista.get(i).getMnem() + " " +
                    lista.get(i).getAdr() + " " + lista.get(i).getComment()
+ "\n");
        }
    }
}
```

ST_entries

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package sp_dz1;

import java.util.LinkedList;
import sp_dz1.TableSymbols.ST_forwardrefs;

/**
 *
 * @author MB
 */
public class ST_entries {

    String name;          // name
    int value;         // value once defined
    boolean defined;           // true after defining occurrence
encountered
    LinkedList<ST_forwardrefs> flink;  // to forward references
    int seg;            // segment in which simbol is defined
    boolean globuse, globdef;

    public ST_entries(String name, int value, boolean defined,
LinkedList<ST_forwardrefs> flink, int seg) {
        this.name = name;
        this.value = value;
        this.defined = defined;
        this.flink = flink;
        this.seg = seg;
        globdef = globuse = false;
    }

    public boolean isDefined() {
        return defined;
    }

    public LinkedList<ST_forwardrefs> getFlink() {
        return flink;
    }

    public String getName() {
        return name;
    }

    public int getValue() {
        return value;
    }

    public void setDefined(boolean defined) {
        this.defined = defined;
```

```
    }

    public void setFlink(LinkedList<ST_forwardrefs> flink) {
        this.flink = flink;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setValue(int value) {
        this.value = value;
    }

    public int getSeg() {
        return seg;
    }

    public void setSeg(int seg) {
        this.seg = seg;
    }

    public boolean isGlobdef() {
        return globdef;
    }

    public void setGlobdef(boolean globdef) {
        this.globdef = globdef;
    }

    public boolean isGlobuse() {
        return globuse;
    }

    public void setGlobuse(boolean globuse) {
        this.globuse = globuse;
    }

}
```

TableDirective.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package sp_dz1;

import java.util.LinkedList;
import java.util.List;

/**
 *
 * @author MB
 */
public class TableDirective {//tabela direktiva

    class Zapis {

        private String nameSeg;
        private int base, len;
        private String desc;

        public Zapis(String nameSeg, int base, int len, String desc) {
            this.nameSeg = nameSeg;
            this.base = base;
            this.len = len;
            this.desc = desc;
        }

        public void setBase(int base) {
            this.base = base;
        }

        public void setLen(int len) {
            this.len = len;
        }

        public int getBase() {
            return base;
        }

        public String getDesc() {
            return desc;
        }

        public int getLen() {
            return len;
        }

        public String getNameSeg() {
            return nameSeg;
        }
```

```
    }
    private List<Zapis> listZapis;

    public TableDirective() {
        listZapis = new LinkedList<Zapis>();
        addItems();
    }

    private void addItems() {
        listZapis.add(new Zapis(".text", 0, 0, "RP"));
        listZapis.add(new Zapis(".bss", 0, 0, "RW"));
        listZapis.add(new Zapis(".data", 0, 0, "RWP"));
    }

    public List<Zapis> getListZapis() {
        return listZapis;
    }

    @Override
    public String toString() {
        String res = "TableDirective:\n";
        for (int i = 0; i < listZapis.size(); i++) {
            if (listZapis.get(i).getBase() != listZapis.get(i).getLen()) {
                res += (listZapis.get(i).getNameSeg() + "\t" +
listZapis.get(i).getBase() + "\t"
                        + listZapis.get(i).getLen() + "\t" +
listZapis.get(i).getDesc() + "\n");
            }
        }
        return res;
    }

    public String getZapisString(int index){
        return ("\t\t" + listZapis.get(index).getNameSeg() + "\t" +
listZapis.get(index).getBase() + "\t"
                        + listZapis.get(index).getLen() + "\t" +
listZapis.get(index).getDesc() + "\n");
    }

    public boolean isDefined(int index){
        return listZapis.get(index).getBase() !=
listZapis.get(index).getLen();
    }


}
```

TableOperation.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sp_dz1;

import java.util.LinkedList;
import java.util.List;

/**
 *
 * @author MB
 */
public class TableOperation {//tabela operacionih kodova
    class Polje{
        String mnemonic;
        String valueHex;
        int valueDec;

        public Polje(String mnemonic, String valueHex, int value) {
            this.mnemonic = mnemonic;
            this.valueHex = valueHex;
            this.valueDec = value;
        }

    };

    private static List<Polje> listMnem;

    public TableOperation() {
        listMnem = new LinkedList<Polje>();
        addItems();
    }

    private void addItems(){
        //jednobajtne
        listMnem.add(new Polje("CLC", "02h", 2));
        listMnem.add(new Polje("CLX", "03h", 3));
        listMnem.add(new Polje("CMC", "04h", 4));
        listMnem.add(new Polje("INC", "05h", 5));
        listMnem.add(new Polje("DEC", "06h", 6));
        listMnem.add(new Polje("INX", "07h", 7));
        listMnem.add(new Polje("DEX", "08h", 8));
        listMnem.add(new Polje("TAX", "09h", 9));
        listMnem.add(new Polje("INI", "0Ah", 10));
        listMnem.add(new Polje("OTI", "0Eh", 14));
        listMnem.add(new Polje("PSH", "13h", 19));
        listMnem.add(new Polje("POP", "14h", 20));
        listMnem.add(new Polje("SHL", "15h", 21));
        listMnem.add(new Polje("SHR", "16h", 22));
```

```
        listMnem.add(new Polje("RET", "17h", 23));
        listMnem.add(new Polje("HLT", "18h", 24));

        //dvobajtne
        listMnem.add(new Polje("LDA", "19h", 25));
        listMnem.add(new Polje("LDX", "1Ah", 26));
        listMnem.add(new Polje("LDI", "1Bh", 27));
        listMnem.add(new Polje("STA", "1Eh", 30));
        listMnem.add(new Polje("STX", "1Fh", 31));
        listMnem.add(new Polje("ADD", "20h", 32));
        listMnem.add(new Polje("ADX", "21h", 33));
        listMnem.add(new Polje("SUB", "26h", 38));
        listMnem.add(new Polje("SBX", "27h", 39));
        listMnem.add(new Polje("CMP", "2Ch", 44));
        listMnem.add(new Polje("ANA", "2Fh", 47));
        listMnem.add(new Polje("ORA", "32h", 50));
        listMnem.add(new Polje("BRN", "35h", 53));
        listMnem.add(new Polje("BZE", "36h", 54));
        listMnem.add(new Polje("BNZ", "37h", 55));
        listMnem.add(new Polje("BPZ", "38h", 56));
        listMnem.add(new Polje("BNG", "39h", 57));
        listMnem.add(new Polje("BCC", "3Ah", 58));
        listMnem.add(new Polje("BCS", "3Bh", 59));
        listMnem.add(new Polje("JSR", "3Ch", 60));

        //ostalo
/*        listMnem.add(new Polje("DC", "0h", 0));
        listMnem.add(new Polje("DS", "0h", 0));
        listMnem.add(new Polje("BEG", "FFh", -1));
        listMnem.add(new Polje("END", "FFh", -1));
        listMnem.add(new Polje("TXT", "FFh", -1));
        listMnem.add(new Polje("BSS", "FFh", -1));
        listMnem.add(new Polje("DAT", "FFh", -1));
*/    }

    public static boolean exists(String mnem){
        for(int i = 0; i<listMnem.size(); i++){
            if((listMnem.get(i).mnemonic).equalsIgnoreCase(mnem) ) return
true;
        }
        return false;
    }

    public static boolean oneByteInstr(String s){
        if("CLC".equalsIgnoreCase(s)) return true;
        if("CLX".equalsIgnoreCase(s)) return true;
        if("CMC".equalsIgnoreCase(s)) return true;
        if("INC".equalsIgnoreCase(s)) return true;
        if("DEC".equalsIgnoreCase(s)) return true;
        if("INX".equalsIgnoreCase(s)) return true;
        if("DEX".equalsIgnoreCase(s)) return true;
        if("TAX".equalsIgnoreCase(s)) return true;
        if("INI".equalsIgnoreCase(s)) return true;
```

```
    if("OTI".equalsIgnoreCase(s)) return true;
    if("PSH".equalsIgnoreCase(s)) return true;
    if("POP".equalsIgnoreCase(s)) return true;
    if("SHL".equalsIgnoreCase(s)) return true;
    if("SHR".equalsIgnoreCase(s)) return true;
    if("RET".equalsIgnoreCase(s)) return true;
    if("HLT".equalsIgnoreCase(s)) return true;

    if("BEG".equalsIgnoreCase(s)) return true;
    if("END".equalsIgnoreCase(s)) return true;
    if("TXT".equalsIgnoreCase(s)) return true;
    if("BSS".equalsIgnoreCase(s)) return true;
    if("DAT".equalsIgnoreCase(s)) return true;

    return false;
}

public static boolean twoBytesInstr(String s){
    if("LDA".equalsIgnoreCase(s)) return true;
    if("LDX".equalsIgnoreCase(s)) return true;
    if("LDI".equalsIgnoreCase(s)) return true;
    if("STA".equalsIgnoreCase(s)) return true;
    if("STX".equalsIgnoreCase(s)) return true;
    if("ADD".equalsIgnoreCase(s)) return true;
    if("ADX".equalsIgnoreCase(s)) return true;
    if("SUB".equalsIgnoreCase(s)) return true;
    if("SBX".equalsIgnoreCase(s)) return true;
    if("CMP".equalsIgnoreCase(s)) return true;
    if("ANA".equalsIgnoreCase(s)) return true;
    if("ORA".equalsIgnoreCase(s)) return true;
    if("BRN".equalsIgnoreCase(s)) return true;
    if("BZE".equalsIgnoreCase(s)) return true;
    if("BNZ".equalsIgnoreCase(s)) return true;
    if("BPZ".equalsIgnoreCase(s)) return true;
    if("BNG".equalsIgnoreCase(s)) return true;
    if("BCC".equalsIgnoreCase(s)) return true;
    if("BCS".equalsIgnoreCase(s)) return true;
    if("JSR".equalsIgnoreCase(s)) return true;

    if("DC".equalsIgnoreCase(s)) return true;
    if("DS".equalsIgnoreCase(s)) return true;
    if("ORG".equalsIgnoreCase(s)) return true;
    if("EQU".equalsIgnoreCase(s)) return true;
    if("DEF".equalsIgnoreCase(s)) return true;
    if("USE".equalsIgnoreCase(s)) return true;
    return false;
}

public static boolean isDirective(String dir){
    dir = dir.trim();
    if("BEG".equalsIgnoreCase(dir)) return true;
    if("END".equalsIgnoreCase(dir)) return true;
    if("ORG".equalsIgnoreCase(dir)) return true;
```

```
        if("DS".equalsIgnoreCase(dir)) return true;
//        if("DC".equalsIgnoreCase(dir)) return true;
        if("EQU".equalsIgnoreCase(dir)) return true;
        if("TXT".equalsIgnoreCase(dir)) return true;
        if("BSS".equalsIgnoreCase(dir)) return true;
        if("DAT".equalsIgnoreCase(dir)) return true;
        if("DEF".equalsIgnoreCase(dir)) return true;
        if("USE".equalsIgnoreCase(dir)) return true;
        return false;
    }

    public static boolean isJumpInstr(String s){
        if("BRN".equalsIgnoreCase(s)) return true;
        if("BZE".equalsIgnoreCase(s)) return true;
        if("BNZ".equalsIgnoreCase(s)) return true;
        if("BPZ".equalsIgnoreCase(s)) return true;
        if("BNG".equalsIgnoreCase(s)) return true;
        if("BCC".equalsIgnoreCase(s)) return true;
        if("BCS".equalsIgnoreCase(s)) return true;
        if("JSR".equalsIgnoreCase(s)) return true;
        return false;
    }

    public static int getValueDec(String mnem){
        for(int i = 0; i<listMnem.size(); i++){
            if((listMnem.get(i).mnemonic).equals(mnem) ) return
listMnem.get(i).valueDec;
        }
        return 0;
    }

    public static String getValueHex(String mnem){
        int indx = 0;
        for(int i = 0; i<listMnem.size(); i++){
            if((listMnem.get(indx).mnemonic).equals(mnem) ) return
listMnem.get(indx).valueHex;
        }
        return "0";
    }

    public static List<Polje> getListMnem() {
        return listMnem;
    }


}
```

TableRelocation.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package sp_dz1;

import java.util.LinkedList;
import java.util.List;

/**
 *
 * @author MB
 */
public class TableRelocations {

    class Zapis {
        private String name;
        private int loc, seg, ref;
        private String type;
        private String action;

        public Zapis(String name, int loc, int seg, int ref, String type) {
            this.name = name;
            this.loc = loc;
            this.seg = seg;
            this.ref = ref;
            this.type = type;
        }

        public int getLoc() {
            return loc;
        }

        public int getRef() {
            return ref;
        }

        public int getSeg() {
            return seg;
        }

        public String getType() {
            return type;
        }

        public void setLoc(int loc) {
            this.loc = loc;
        }

        public void setRef(int ref) {
            this.ref = ref;
```

```
        }

        public void setSeg(int seg) {
            this.seg = seg;
        }

        public void setType(String type) {
            this.type = type;
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

    }
    private static List<Zapis> listZapis;

    public TableRelocations() {
        listZapis = new LinkedList<TableRelocations.Zapis>();
    }

    public static List<Zapis> getListZapis() {
        return listZapis;
    }

    //add if in segment
    public void add(String name, int loc, int seg, boolean jump){
        String type = "A1";
        if(jump) type = "R1";
        listZapis.add(new Zapis(name, loc, seg, 0, type));
    }

    public void add(String name, int loc, int seg,int ref, boolean
globused){
        String type = "A1";
        if(globused) type = "AS1";
        listZapis.add(new Zapis(name, loc, seg, ref, type));
    }

    public void fix(String name, int ref, String type){
        for(int i = 0; i<listZapis.size(); i++){
            if(listZapis.get(i).getName().equalsIgnoreCase(name)){
                listZapis.get(i).setRef(ref);
                if(!type.equalsIgnoreCase(""))
listZapis.get(i).setType(type);
            }
        }
    }
```

```
    public boolean calculateIK(String name, String type){
        for (int i = 0; i < listZapis.size(); i++) {
            if(listZapis.get(i).getName().equalsIgnoreCase(name)){
                if(listZapis.get(i).getSeg() == 0) return false;
                int loc = listZapis.get(i).getLoc() - 1;
                String mnem = "";
                for (int j = 0; j < TableOperation.getListMnem().size();
j++) {
                    if(TableOperation.getListMnem().get(j).valueDec == loc)
                        mnem =
TableOperation.getListMnem().get(j).mnemonic;
                }
                if(TableOperation.isJumpInstr(mnem) &&
!mnem.equalsIgnoreCase("JSR")
                        && !( listZapis.get(i).getRef() ==
listZapis.get(i).getSeg())){
                    if(listZapis.get(i).getType().contains("S")) type =
"RS1";
                    else type = "R1";
                    return true;
                }
            }
        }
        return false;
    }

    @Override
    public String toString() {
        String res = "TableRelocations:\n";
        for(int i = 0; i<listZapis.size(); i++)
            res+= (listZapis.get(i).getLoc() + "\t" +
listZapis.get(i).getSeg() + "\t"
                    + listZapis.get(i).getRef() + "\t" +
listZapis.get(i).getType() + "\n");
        return res;
    }

        public String getZapisString(int index) {
        return ("\t\t" + listZapis.get(index).getLoc() + "\t" +
listZapis.get(index).getSeg() + "\t"
                    + listZapis.get(index).getRef() + "\t" +
listZapis.get(index).getType() + "\t"
                    + listZapis.get(index).getName() + "\t"); //dodatna
polja
    }

}
```

TableSymbols.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package sp_dz1;

import java.util.LinkedList;

/**
 *
 * @author MB
 */
public class TableSymbols { //tabela simbola

    class ST_forwardrefs {   // forward references for undefined labels

        int patch;          // to be patched
        String action;      //action to perform

        public ST_forwardrefs(int patch, String action) {
            this.patch = patch;
            this.action = action;
        }

        public void setPatch(int patch) {
            this.patch = patch;
        }

        public int getPatch() {
            return patch;
        }

        public String getAction() {
            return action;
        }

        public void setAction(String action) {
            this.action = action;
        }

    };

    public TableSymbols() {
        listsym = new LinkedList<ST_entries>();
    }

    public int getSize(){
        return listsym.size();
    }

    public LinkedList<ST_entries> getListsym() {
        return listsym;
```

```
    }

    //prints symbol table on std.out
    public void printsymboltable() {

        System.out.println("\nSymbol Table\n------------\n");
        int index = 0;
        while (index != listsym.size()) {
            System.out.print(listsym.get(index).getName() + "   ");
            if (!listsym.get(index).isDefined()) {
                System.out.print("--- undefined");
            } else {
                System.out.print(listsym.get(index).getValue() + "   ");
            }
            System.out.print("\tFLINK:");
            int indx = 0;
            if (listsym.get(index).getFlink() != null) {
                while (indx != listsym.get(index).getFlink().size()) {

System.out.print(listsym.get(index).getFlink().get(indx).getPatch() +

listsym.get(index).getFlink().get(indx).getAction() + "  ");
                    indx++;
                }
            }
            index++;
            System.out.println();
        }
    }

    // Adds name to table with known value
    public void enter(String name, int value, int seg, boolean globuse,
boolean globdef) {
        int index = findEntry(name);
        listsym.get(index).setName(name);
        listsym.get(index).setValue(value);
        listsym.get(index).setSeg(seg);
        if(globdef || globuse){
            listsym.get(index).setGlobuse(globuse);
            listsym.get(index).setGlobdef(globdef);
            listsym.get(index).setDefined(false);
        }else{
            listsym.get(index).setDefined(true);
        }
    }

    // Returns value of required name
    // location is the current value of the instruction location counter
    public int valueOfSymbol(String name, int location, String action) {
        int index = findEntry(name);
        if (listsym.get(index).isDefined()) {
            AS.insertNewOp(action);
            return listsym.get(index).getValue();
```

```
        } else{
            ST_forwardrefs forwardentry = new ST_forwardrefs(location,
action);
            listsym.get(index).getFlink().addFirst(forwardentry);
            return
listsym.get(index).isGlobuse()?0:listsym.get(index).getValue(); //-1
        }
    }

    //returns the index of the element in list if the
    //element is found, else adds new entry
    public int findEntry(String name) {
        boolean found = false;
        int index = 0;
        while (!found && index != listsym.size()) {
            if (name.equals(listsym.get(index).getName())) {
                found = true;
            } else {
                index++;
            }
        }
        if (found) {
            return index;
        }

        ST_entries symentryE = new ST_entries(name, 0, false, new
LinkedList<ST_forwardrefs>(), 0);
        listsym.addFirst(symentryE);
        return 0;
    }

    public void backpatch(int[] mem, TableRelocations tr) {
        for (int i = 0; i < listsym.size(); i++) {
            ST_entries entry = listsym.get(i);
            if (entry.getFlink() != null) {
                int indx = 0;
                if (entry.getFlink().size() != 0) {
                    while (entry.getFlink().size() != indx) {

if(entry.getFlink().get(indx).getAction().equalsIgnoreCase("+")){
                            mem[entry.getFlink().get(indx).getPatch()] +=
entry.getValue();
                        } else
if(entry.getFlink().get(indx).getAction().equalsIgnoreCase("-")){
                            mem[entry.getFlink().get(indx).getPatch()] -=
entry.getValue();
                        }
                        indx++;
                    }
                }
            }
            if(!this.isGlobusedSymb(entry.getName())) {
                    tr.fix(entry.getName(), entry.getSeg(), "");
```

```
            }
            if(this.isGlobusedSymb(entry.getName()) &&
this.isGlobudefSymb(entry.getName())){
                tr.fix(entry.getName(),
this.numGlobusedSymbInTable(entry.getName()), "");
            }

        }
    }

    public String getEntryString(int i){
        return ("\t\t" + listsym.get(i).getName() + "\t" +
listsym.get(i).getValue() +
                "\t" + listsym.get(i).getSeg() + "\t" +
                (listsym.get(i).isGlobdef() &&
!listsym.get(i).isGlobuse()?"D":"") +
                (listsym.get(i).isGlobuse() &&
!listsym.get(i).isGlobdef()?"U":"") +
                (listsym.get(i).isGlobuse() &&
listsym.get(i).isGlobdef()?"E":""));
    }

    public boolean isDefinedSymb(String s){
        for(int i=0; i<listsym.size(); i++)
            if(s.equalsIgnoreCase(listsym.get(i).getName()) &&
listsym.get(i).isDefined()) return true;
        return false;
    }

    public boolean isGlobusedSymb(String s){
        for(int i=0; i<listsym.size(); i++)
            if(s.equalsIgnoreCase(listsym.get(i).getName()) &&
listsym.get(i).isGlobuse()) return true;
        return false;
    }

    public boolean isGlobudefSymb(String s){
        for(int i=0; i<listsym.size(); i++)
            if(s.equalsIgnoreCase(listsym.get(i).getName()) &&
listsym.get(i).isGlobdef()) return true;
        return false;
    }

    public int numGlobusedSymbInTable(String s){
        int num = 1;
        for(int i =0; i<listsym.size();i++)
            if(listsym.get(i).isGlobuse()){
                if(s.equalsIgnoreCase(listsym.get(i).getName())) return
num;
                else num++;
            }
        return -1;
    }
```

```
    public ST_entries getSymbol(String s){
        for(int i = 0; i<listsym.size(); i++)
            if(listsym.get(i).getName().equalsIgnoreCase(s)) return
listsym.get(i);
        return null;
    }

    public String getOpType(String s, int loc){
        for (int i = 0; i < listsym.size(); i++) {
            if (listsym.get(i).getName().equalsIgnoreCase(s)) {
                for (int j = 0; j < listsym.get(i).getFlink().size(); j++)
{
                    if (listsym.get(i).getFlink().get(j).getPatch() == loc)
{
                        return
listsym.get(i).getFlink().get(j).getAction();
                    }
                }
            }
        }
        return "";
    }
    private LinkedList<ST_entries> listsym;
}
```