Електротехноички факултет, Универзитет у Београду

Катедра за рачунарску технику и унформатику

Други 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 део емулатора који врши статичку рекомпилацију програма намењеног за хипотетички рачунар, тако да се емулација врши на x86 процесорима. Ресурсе мапирати онако како је приказано у првом задатку са вежби (фајл "EmulVezbe.doc" запакован у архиви на линку "V4_Emulatori"). Сматрати да се на улазу добија излаз првог домаћег и да је као излаз потребно изгенерисати текстуални фајл који садржи одговарајући програм на x86 асемблеру. Израчунавање флегова оптимизовати. Ради поједностављења, сматрати да се у улазном фајлу могу појавити само инструкције које су у прилогу маркиране жутом бојом. Програм увек почиње првом инструкцијом после и издељен је на сегменте. Сегменти који се могу појавити су

TXT - Сегмент који садржи код,

DAT - Сегмент који садржи иницијализоване податке,

BSS - Сегмент који садржи неиницијализоване податке.

Опис решења

Домаћи задатак је решен у програмском језику JAVA. Комплетан код решења је приложен у фолдеру sp_dz2. Приликом решавања овог проблема уведене претпоставке из првог домаћег задатка важе и овде. Поред ових уведене су још неке претпоставке:

- са стандардног улаза се могу читати цели бројеви у опсегу од -127 до 127 (огрничење HYPO машине да су подаци 1 B).
- сваки програм почиње инструкцијом mov ECX, 0 , јер је потребно одсећи највиша три бајта регистра ECX због инструкција које раде са индексним регистром као што су ADX, SBX...

Програм се састоји од неколико фаза:

1. Најпре се чита улазни фајл, и свака линија се обрађује засебно. То подразумева пуњење табеле симбола, табеле релокација, као и меморије
2. Затим се уводе декларације свих потребних потпрограма, и дефиниције свих симбола који се увозе, односно извозе
3. Генерише се .data сегмент, при чему увек постоји једна променљива dSTART у односу на коју ће се рачунати померај због меморијског мапирања
4. Стварају се блокови
5. Прелази се на „рефакторисање“ кода, односно машинске инструкције се преводе x86 инструкције
6. Врши се оптимизација кода, помоћу *needs* и *sets* скупова
7. Преведени код се уписује у излазни фајл

Упутство за покретање

Излаз првог домаћег задатка се прекуца или прекопира у фајл *ulaz.txt* који се налази у фолдеру sp_dz2. Програм се покреће двокликом на икону sp_dz2.jar или из командне линије, тако што ће се укуцати java -jar „путања_до_директоријума\sp_dz2.jar“ Резултати рада ће бити исписани у датотеци *izlaz.txt* у истом фолдеру.

Примери

Пример1

Улаз:
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

Излаз:.386
.model flat, stdcall

STD_INPUT_HANDLE equ -10
STD_OUTPUT_HANDLE equ -11
PUBLIC main

INCLUDELIB kernel32.lib

ExitProcess proto:dword
ReadConsoleA proto :dword, :dword, :dword, :dword, :dword
WriteConsoleA proto :dword, :dword, :dword, :dword, :dword

GetStdHandle proto :dword

.data

dSTART	DB	0	
TEMP	DB	1	DUP(0)
BITS	DB	0	
stdin	DD	0	
stdout	DD	0	

```
.code
readc    PROC
        push EBP
        mov EBP, ESP
        push EAX
        push EBX
        push EDX
        cmp stdin, 0
        jne dalje_readc
        invoke GetStdHandle, STD_INPUT_HANDLE
        mov stdin, EAX
dalje_readc:
        sub ESP, 4
        mov EAX, ESP
        sub ESP, 4
        mov EBX, ESP
        invoke ReadConsoleA, stdin, EAX, 1, EBX, 0
        add ESP,4
        pop EAX
        mov [EBP+8], AL
        pop EDX
        pop EBX
        pop EAX
        pop EBP
        ret
readc   ENDP

readdec PROC
        push EBP
        mov EBP, ESP
        push EBX
        push ECX
        push EDX
        mov ECX, 10
        mov EBX, 0
        mov EDX,0
??read: push ECX
        sub ESP, 4
        call readc
        pop EAX
        pop ECX
        cmp AL,'-'
        je jesteminus
        jmp daljeminus
        jesteminus:
        mov EDX,1
        push ECX
        sub ESP, 4
```

```
        call readc
        pop EAX
        pop ECX
daljeminus:
        cmp AL, '0'
        jb kraj_rddec
        cmp AL, '9'
        ja kraj_rddec
        sub AL, '0'
        push EAX
        mov EAX, EBX
        push EDX
        mul ECX
        pop EDX
        mov EBX, EAX
        pop EAX
        and EAX, 0000000fh
        add EBX, EAX
        jmp ??read
kraj_rddec:mov EAX, EBX
        cmp EDX,1
        je  negiraj
        jmp daljenegiraj
        negiraj: neg AL
daljenegiraj:
        pop EDX
        pop ECX
        pop EBX
        pop EBP
        ret
readdec ENDP

printc    PROC par1:dword, par2:dword
        ; push EBP
        ; mov EBP, ESP
        push EAX
        push EBX
        push ECX
        cmp stdout, 0
        jne dalje_printc
        invoke GetStdHandle, STD_OUTPUT_HANDLE
        mov stdout, EAX
dalje_printc:
        mov ECX, [par1]
        mov EBX, EBP
        add EBX, 8
        mov EAX, [par2]
        invoke WriteConsoleA, stdout, EAX, ECX, EBX, 0
```

```
        pop ECX
        pop EBX
        pop EAX
        ; pop EBP
        ret
printc    ENDP

writedec PROC
        push EAX
        push EBX
        push ECX
         push EDX
        mov AH, 0
        mov CL, 10
         mov EBX, 0
        mov EDX, 0
        mov DL, AL
        and DL, 80h
        jnz obrni
        jmp ??write
obrni: neg AL
??write:div CL
        xchg AH, AL
        add AL,'0'
        push EAX
        inc EBX
        xchg AH, AL
        cmp AL, 0
        je minusprovera
        mov AH, 0
        jmp ??write
minusprovera:
        cmp DL, 80h
        je dodajminus
        jmp kraj_wrdec
dodajminus:
        push '-'
        push ESP
        push 1
        call printc
        add ESP, 4
kraj_wrdec:
        cmp EBX, 0
        je kraj_skroz
        push ESP
        push 1
        call printc
        add ESP,4
```

```
        dec EBX
        jmp kraj_wrdec
kraj_skroz:
        pop EDX
        pop ECX
        pop EBX
        pop EAX
        ret
writedec ENDP


main:
        mov ECX, 0
        call readdec
_LOOP:          shr AL, 1
        jnc _EVEN
        mov TEMP, AL
        mov AL, BITS
        inc AL
        mov BITS, AL
        mov AL, TEMP
        inc AL
        dec AL
_EVEN:          jnz _LOOP
        mov AL, BITS
        call writedec
        invoke ExitProcess, 0
end main
```

Пример2

Улаз:
```
LINK
2 1 0
# segments      (name  base   len    desc)
                .text  2      18     RP
                .bss   0      2      RW
# symbols       (name  value  seg    type)
                START2        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
```

Излаз:
```
.386
.model flat, stdcall

PUBLIC START;
STD_INPUT_HANDLE equ -10
STD_OUTPUT_HANDLE equ -11
PUBLIC main

INCLUDELIB kernel32.lib

ExitProcess proto:dword
ReadConsoleA proto :dword, :dword, :dword, :dword, :dword
WriteConsoleA proto :dword, :dword, :dword, :dword, :dword

GetStdHandle proto :dword

.data
dSTART      DB    0
        DB    2     DUP(0)
stdin   DD    0
stdout  DD    0

.code
readc   PROC
        push EBP
        mov EBP, ESP
        push EAX
        push EBX
        push EDX
        cmp stdin, 0
        jne dalje_readc
        invoke GetStdHandle, STD_INPUT_HANDLE
        mov stdin, EAX
```

```
dalje_readc:
        sub ESP, 4
        mov EAX, ESP
        sub ESP, 4
        mov EBX, ESP
        invoke ReadConsoleA, stdin, EAX, 1, EBX, 0
        add ESP,4
        pop EAX
        mov [EBP+8], AL
        pop EDX
        pop EBX
        pop EAX
        pop EBP
        ret
readc   ENDP

readdec PROC
        push EBP
        mov EBP, ESP
        push EBX
        push ECX
        push EDX
        mov ECX, 10
        mov EBX, 0
        mov EDX,0
??read: push ECX
        sub ESP, 4
        call readc
        pop EAX
        pop ECX
        cmp AL,'-'
        je jesteminus
        jmp daljeminus
        jesteminus:
        mov EDX,1
        push ECX
        sub ESP, 4
        call readc
        pop EAX
        pop ECX
daljeminus:
        cmp AL, '0'
        jb kraj_rddec
        cmp AL, '9'
        ja kraj_rddec
        sub AL, '0'
        push EAX
        mov EAX, EBX
```

```
        push EDX
        mul ECX
        pop EDX
        mov EBX, EAX
        pop EAX
        and EAX, 0000000fh
        add EBX, EAX
        jmp ??read
kraj_rddec:mov EAX, EBX
        cmp EDX,1
        je  negiraj
        jmp daljenegiraj
        negiraj: neg AL
daljenegiraj:
        pop EDX
        pop ECX
        pop EBX
        pop EBP
        ret
readdec ENDP

printc    PROC par1:dword, par2:dword
        ; push EBP
        ; mov EBP, ESP
        push EAX
        push EBX
        push ECX
        cmp stdout, 0
        jne dalje_printc
        invoke GetStdHandle, STD_OUTPUT_HANDLE
        mov stdout, EAX
dalje_printc:
        mov ECX, [par1]
        mov EBX, EBP
        add EBX, 8
        mov EAX, [par2]
        invoke WriteConsoleA, stdout, EAX, ECX, EBX, 0
        pop ECX
        pop EBX
        pop EAX
        ; pop EBP
        ret
printc    ENDP

writedec PROC
        push EAX
        push EBX
        push ECX
```

```
        push EDX
        mov AH, 0
        mov CL, 10
        mov EBX, 0
        mov EDX, 0
        mov DL, AL
        and DL, 80h
        jnz obrni
        jmp ??write
obrni: neg AL
??write:div CL
        xchg AH, AL
        add AL,'0'
        push EAX
        inc EBX
        xchg AH, AL
        cmp AL, 0
        je minusprovera
        mov AH, 0
        jmp ??write
minusprovera:
        cmp DL, 80h
        je dodajminus
        jmp kraj_wrdec
dodajminus:
        push '-'
        push ESP
        push 1
        call printc
        add ESP, 4
kraj_wrdec:
        cmp EBX, 0
        je kraj_skroz
        push ESP
        push 1
        call printc
        add ESP,4
        dec EBX
        jmp kraj_wrdec
kraj_skroz:
        pop EDX
        pop ECX
        pop EBX
        pop EAX
        ret
writedec ENDP
```

```
main:
        mov ECX, 0
START   xor CL, CL
        call readdec
        mov [0], AL
        call readdec
        mov [1], AL
        mov AL, 0
        add AL, [1]
        call writedec
        mov AL, 0
        sub AL, [1]
        call writedec
        invoke ExitProcess, 0
end main
```

Пример3

Улаз:

```
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
```

Излаз:

```
.386
.model flat, stdcall

PUBLIC TEMP,WRITE
STD_OUTPUT_HANDLE equ -11
PUBLIC main

INCLUDELIB kernel32.lib

ExitProcess proto:dword
WriteConsoleA proto :dword, :dword, :dword, :dword, :dword

GetStdHandle proto :dword

.data
dSTART      DB      0
TEMP  DB    1       DUP(0)
stdout  DD  0

.code
printc   PROC par1:dword, par2:dword
        ; push EBP
        ; mov EBP, ESP
        push EAX
        push EBX
        push ECX
        cmp stdout, 0
        jne dalje_printc
        invoke GetStdHandle, STD_OUTPUT_HANDLE
        mov stdout, EAX
dalje_printc:
        mov ECX, [par1]
```

```
        mov EBX, EBP
        add EBX, 8
        mov EAX, [par2]
        invoke WriteConsoleA, stdout, EAX, ECX, EBX, 0
        pop ECX
        pop EBX
        pop EAX
        ; pop EBP
        ret
printc    ENDP

writedec PROC
        push EAX
        push EBX
        push ECX
         push EDX
        mov AH, 0
        mov CL, 10
         mov EBX, 0
        mov EDX, 0
        mov DL, AL
        and DL, 80h
        jnz obrni
        jmp ??write
obrni: neg AL
??write:div CL
        xchg AH, AL
        add AL,'0'
        push EAX
        inc EBX
        xchg AH, AL
        cmp AL, 0
        je minusprovera
        mov AH, 0
        jmp ??write
minusprovera:
        cmp DL, 80h
        je dodajminus
        jmp kraj_wrdec
dodajminus:
        push '-'
        push ESP
        push 1
        call printc
        add ESP, 4
kraj_wrdec:
        cmp EBX, 0
        je kraj_skroz
```

```
        push ESP
        push 1
        call printc
        add ESP,4
        dec EBX
        jmp kraj_wrdec
kraj_skroz:
        pop EDX
        pop ECX
        pop EBX
        pop EAX
        ret
writedec ENDP


main:
        mov ECX, 0
WRITEcall writedec
        pop EIP
end main
```

Пример4

Улаз:

```
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
```

Излаз:

```
.386
.model flat, stdcall

EXTRN TEMP,WRITE;
STD_INPUT_HANDLE equ -10
PUBLIC main

INCLUDELIB kernel32.lib

ExitProcess proto:dword
ReadConsoleA proto :dword, :dword, :dword, :dword, :dword

GetStdHandle proto :dword

.data
dSTART      DB      0
BITS    DB      0
stdin   DD      0

.code
readc   PROC
        push EBP
        mov EBP, ESP
```

```
        push EAX
        push EBX
        push EDX
        cmp stdin, 0
        jne dalje_readc
        invoke GetStdHandle, STD_INPUT_HANDLE
        mov stdin, EAX
dalje_readc:
        sub ESP, 4
        mov EAX, ESP
        sub ESP, 4
        mov EBX, ESP
        invoke ReadConsoleA, stdin, EAX, 1, EBX, 0
        add ESP,4
        pop EAX
        mov [EBP+8], AL
        pop EDX
        pop EBX
        pop EAX
        pop EBP
        ret
readc   ENDP

readdec PROC
        push EBP
        mov EBP, ESP
        push EBX
        push ECX
        push EDX
        mov ECX, 10
        mov EBX, 0
        mov EDX,0
??read: push ECX
        sub ESP, 4
        call readc
        pop EAX
        pop ECX
        cmp AL,'-'
        je jesteminus
        jmp daljeminus
        jesteminus:
        mov EDX,1
        push ECX
        sub ESP, 4
        call readc
        pop EAX
        pop ECX
daljeminus:
```

```
        cmp AL, '0'
        jb kraj_rddec
        cmp AL, '9'
        ja kraj_rddec
        sub AL, '0'
        push EAX
        mov EAX, EBX
        push EDX
        mul ECX
        pop EDX
        mov EBX, EAX
        pop EAX
        and EAX, 0000000fh
        add EBX, EAX
        jmp ??read
kraj_rddec:mov EAX, EBX
        cmp EDX,1
        je  negiraj
        jmp daljenegiraj
        negiraj: neg AL
daljenegiraj:
        pop EDX
        pop ECX
        pop EBX
        pop EBP
        ret
readdec ENDP



main:
        mov ECX, 0
        call readdec
_LOOP:          shr AL, 1
        jnc _EVEN
        mov TEMP, AL
        mov AL, BITS
        inc AL
        mov BITS, AL
        mov AL, TEMP
        inc AL
        dec AL
_EVEN:          jnz _LOOP
        mov AL, BITS
        call WRITE
        invoke ExitProcess, 0
end main
```

Пример5

Улаз:
```
LINK
1 1 0
# segments      (name   base    len     desc)
                .text   0       13      RP
# symbols       (name   value   seg     type)
                START0          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
```

Излаз:
```
.386
.model flat, stdcall

PUBLIC START;
STD_INPUT_HANDLE equ -10
STD_OUTPUT_HANDLE equ -11
PUBLIC main

INCLUDELIB kernel32.lib

ExitProcess proto:dword
ReadConsoleA proto :dword, :dword, :dword, :dword, :dword
WriteConsoleA proto :dword, :dword, :dword, :dword, :dword

GetStdHandle proto :dword

.data
dSTART      DB      0
stdin   DD      0
stdout  DD      0

.code
readc   PROC
        push EBP
        mov EBP, ESP
        push EAX
        push EBX
        push EDX
        cmp stdin, 0
        jne dalje_readc
        invoke GetStdHandle, STD_INPUT_HANDLE
        mov stdin, EAX
dalje_readc:
        sub ESP, 4
```

```
        mov EAX, ESP
        sub ESP, 4
        mov EBX, ESP
        invoke ReadConsoleA, stdin, EAX, 1, EBX, 0
        add ESP,4
        pop EAX
        mov [EBP+8], AL
        pop EDX
        pop EBX
        pop EAX
        pop EBP
        ret
readc   ENDP

readdec PROC
        push EBP
        mov EBP, ESP
        push EBX
        push ECX
        push EDX
        mov ECX, 10
        mov EBX, 0
        mov EDX,0
??read: push ECX
        sub ESP, 4
        call readc
        pop EAX
        pop ECX
        cmp AL,'-'
        je jesteminus
        jmp daljeminus
        jesteminus:
        mov EDX,1
        push ECX
        sub ESP, 4
        call readc
        pop EAX
        pop ECX
daljeminus:
        cmp AL, '0'
        jb kraj_rddec
        cmp AL, '9'
        ja kraj_rddec
        sub AL, '0'
        push EAX
        mov EAX, EBX
        push EDX
        mul ECX
```

```
        pop EDX
        mov EBX, EAX
        pop EAX
        and EAX, 0000000fh
        add EBX, EAX
        jmp ??read
kraj_rddec:mov EAX, EBX
        cmp EDX,1
        je  negiraj
        jmp daljenegiraj
        negiraj: neg AL
daljenegiraj:
        pop EDX
        pop ECX
        pop EBX
        pop EBP
        ret
readdec ENDP

printc    PROC par1:dword, par2:dword
        ; push EBP
        ; mov EBP, ESP
        push EAX
        push EBX
        push ECX
        cmp stdout, 0
        jne dalje_printc
        invoke GetStdHandle, STD_OUTPUT_HANDLE
        mov stdout, EAX
dalje_printc:
        mov ECX, [par1]
        mov EBX, EBP
        add EBX, 8
        mov EAX, [par2]
        invoke WriteConsoleA, stdout, EAX, ECX, EBX, 0
        pop ECX
        pop EBX
        pop EAX
        ; pop EBP
        ret
printc    ENDP

writedec PROC
        push EAX
        push EBX
        push ECX
         push EDX
        mov AH, 0
```

```
        mov CL, 10
         mov EBX, 0
        mov EDX, 0
        mov DL, AL
        and DL, 80h
        jnz obrni
        jmp ??write
obrni: neg AL
??write:div CL
        xchg AH, AL
        add AL,'0'
        push EAX
        inc EBX
        xchg AH, AL
        cmp AL, 0
        je minusprovera
        mov AH, 0
        jmp ??write
minusprovera:
        cmp DL, 80h
        je dodajminus
        jmp kraj_wrdec
dodajminus:
        push '-'
        push ESP
        push 1
        call printc
        add ESP, 4
kraj_wrdec:
        cmp EBX, 0
        je kraj_skroz
        push ESP
        push 1
        call printc
        add ESP,4
        dec EBX
        jmp kraj_wrdec
kraj_skroz:
        pop EDX
        pop ECX
        pop EBX
        pop EAX
        ret
writedec ENDP


main:
        mov ECX, 0
```

```
START call readdec
        push AX
        call readdec
        push AX
        call readdec
        push AX
        pop AX
        call writedec
        pop AX
        call writedec
        pop AX
        call writedec
        invoke ExitProcess, 0
end main
```

Пример6

Улаз:

```
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
```

Излаз:

```
.386
.model flat, stdcall

PUBLIC main

INCLUDELIB kernel32.lib

ExitProcess proto:dword

.data
dSTART      DB      0
BLA     DB      ?

.code

main:
        mov ECX, 0
        push AX
        mov AL, BLA
        push AX
        jc _LABELA1
        push AX
_LABELA1:     push AX
        add AL, BLA
        jc _LABELA2
        add AL, BLA
_LABELA2:     jnz _LABELA1
        invoke ExitProcess, 0
end main
```

Пример7

Улаз:
```
LINK
1 1 0
# segments      (name   base    len     desc)
                .text   0       3       RP
# symbols       (name   value   seg     type)
                START0          1       D
# relocations   (loc    seg     ref     type)   (name   op      place/len)
# data (one line per segment)
10 14 24
```

Излаз:
```
.386
.model flat, stdcall

PUBLIC START;
STD_INPUT_HANDLE equ -10
STD_OUTPUT_HANDLE equ -11
PUBLIC main

INCLUDELIB kernel32.lib

ExitProcess proto:dword
ReadConsoleA proto :dword, :dword, :dword, :dword, :dword
WriteConsoleA proto :dword, :dword, :dword, :dword, :dword

GetStdHandle proto :dword

.data
dSTART      DB      0
stdin   DD      0
stdout  DD      0

.code
readc   PROC
        push EBP
        mov EBP, ESP
        push EAX
        push EBX
        push EDX
        cmp stdin, 0
        jne dalje_readc
        invoke GetStdHandle, STD_INPUT_HANDLE
        mov stdin, EAX
dalje_readc:
        sub ESP, 4
```

```
        mov EAX, ESP
        sub ESP, 4
        mov EBX, ESP
        invoke ReadConsoleA, stdin, EAX, 1, EBX, 0
        add ESP,4
        pop EAX
        mov [EBP+8], AL
        pop EDX
        pop EBX
        pop EAX
        pop EBP
        ret
readc   ENDP

readdec PROC
        push EBP
        mov EBP, ESP
        push EBX
        push ECX
        push EDX
        mov ECX, 10
        mov EBX, 0
        mov EDX,0
??read: push ECX
        sub ESP, 4
        call readc
        pop EAX
        pop ECX
        cmp AL,'-'
        je jesteminus
        jmp daljeminus
        jesteminus:
        mov EDX,1
        push ECX
        sub ESP, 4
        call readc
        pop EAX
        pop ECX
daljeminus:
        cmp AL, '0'
        jb kraj_rddec
        cmp AL, '9'
        ja kraj_rddec
        sub AL, '0'
        push EAX
        mov EAX, EBX
        push EDX
        mul ECX
```

```
        pop EDX
        mov EBX, EAX
        pop EAX
        and EAX, 0000000fh
        add EBX, EAX
        jmp ??read
kraj_rddec:mov EAX, EBX
        cmp EDX,1
        je  negiraj
        jmp daljenegiraj
        negiraj: neg AL
daljenegiraj:
        pop EDX
        pop ECX
        pop EBX
        pop EBP
        ret
readdec ENDP

printc    PROC par1:dword, par2:dword
        ; push EBP
        ; mov EBP, ESP
        push EAX
        push EBX
        push ECX
        cmp stdout, 0
        jne dalje_printc
        invoke GetStdHandle, STD_OUTPUT_HANDLE
        mov stdout, EAX
dalje_printc:
        mov ECX, [par1]
        mov EBX, EBP
        add EBX, 8
        mov EAX, [par2]
        invoke WriteConsoleA, stdout, EAX, ECX, EBX, 0
        pop ECX
        pop EBX
        pop EAX
        ; pop EBP
        ret
printc    ENDP

writedec PROC
        push EAX
        push EBX
        push ECX
         push EDX
        mov AH, 0
```

```
        mov CL, 10
         mov EBX, 0
        mov EDX, 0
        mov DL, AL
        and DL, 80h
        jnz obrni
        jmp ??write
obrni: neg AL
??write:div CL
        xchg AH, AL
        add AL,'0'
        push EAX
        inc EBX
        xchg AH, AL
        cmp AL, 0
        je minusprovera
        mov AH, 0
        jmp ??write
minusprovera:
        cmp DL, 80h
        je dodajminus
        jmp kraj_wrdec
dodajminus:
        push '-'
        push ESP
        push 1
        call printc
        add ESP, 4
kraj_wrdec:
        cmp EBX, 0
        je kraj_skroz
        push ESP
        push 1
        call printc
        add ESP,4
        dec EBX
        jmp kraj_wrdec
kraj_skroz:
        pop EDX
        pop ECX
        pop EBX
        pop EAX
        ret
writedec ENDP


main:
        mov ECX, 0
```

```
START call readdec
      call writedec
      invoke ExitProcess, 0
end main
```

Пример8

Улаз:

```
LINK
2 1 7
# segments      (name   base    len     desc)
                .text   0       18      RP
                .data   18      2       RWP
# symbols       (name   value   seg     type)
                START0          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
```

Излаз:

```
.386
.model flat, stdcall

PUBLIC START;
STD_INPUT_HANDLE equ -10
STD_OUTPUT_HANDLE equ -11
PUBLIC main

INCLUDELIB kernel32.lib

ExitProcess proto:dword
ReadConsoleA proto :dword, :dword, :dword, :dword, :dword
WriteConsoleA proto :dword, :dword, :dword, :dword, :dword

GetStdHandle proto :dword

.data
dSTART      DB      0
nnn     DB      0
suma    DB      0
stdin   DD      0
stdout  DD      0

.code
readc   PROC
```

```
        push EBP
        mov EBP, ESP
        push EAX
        push EBX
        push EDX
        cmp stdin, 0
        jne dalje_readc
        invoke GetStdHandle, STD_INPUT_HANDLE
        mov stdin, EAX
dalje_readc:
        sub ESP, 4
        mov EAX, ESP
        sub ESP, 4
        mov EBX, ESP
        invoke ReadConsoleA, stdin, EAX, 1, EBX, 0
        add ESP,4
        pop EAX
        mov [EBP+8], AL
        pop EDX
        pop EBX
        pop EAX
        pop EBP
        ret
readc   ENDP

readdec PROC
        push EBP
        mov EBP, ESP
        push EBX
        push ECX
        push EDX
        mov ECX, 10
        mov EBX, 0
        mov EDX,0
??read: push ECX
        sub ESP, 4
        call readc
        pop EAX
        pop ECX
        cmp AL,'-'
        je jesteminus
        jmp daljeminus
        jesteminus:
        mov EDX,1
        push ECX
        sub ESP, 4
        call readc
        pop EAX
```

```
        pop ECX
daljeminus:
        cmp AL, '0'
        jb kraj_rddec
        cmp AL, '9'
        ja kraj_rddec
        sub AL, '0'
        push EAX
        mov EAX, EBX
        push EDX
        mul ECX
        pop EDX
        mov EBX, EAX
        pop EAX
        and EAX, 0000000fh
        add EBX, EAX
        jmp ??read
kraj_rddec:mov EAX, EBX
        cmp EDX,1
        je  negiraj
        jmp daljenegiraj
        negiraj: neg AL
daljenegiraj:
        pop EDX
        pop ECX
        pop EBX
        pop EBP
        ret
readdec ENDP

printc    PROC par1:dword, par2:dword
        ; push EBP
        ; mov EBP, ESP
        push EAX
        push EBX
        push ECX
        cmp stdout, 0
        jne dalje_printc
        invoke GetStdHandle, STD_OUTPUT_HANDLE
        mov stdout, EAX
dalje_printc:
        mov ECX, [par1]
        mov EBX, EBP
        add EBX, 8
        mov EAX, [par2]
        invoke WriteConsoleA, stdout, EAX, ECX, EBX, 0
        pop ECX
        pop EBX
```

```
        pop EAX
        ; pop EBP
        ret
printc    ENDP

writedec PROC
        push EAX
        push EBX
        push ECX
         push EDX
        mov AH, 0
        mov CL, 10
         mov EBX, 0
        mov EDX, 0
        mov DL, AL
        and DL, 80h
        jnz obrni
        jmp ??write
obrni: neg AL
??write:div CL
        xchg AH, AL
        add AL,'0'
        push EAX
        inc EBX
        xchg AH, AL
        cmp AL, 0
        je minusprovera
        mov AH, 0
        jmp ??write
minusprovera:
        cmp DL, 80h
        je dodajminus
        jmp kraj_wrdec
dodajminus:
        push '-'
        push ESP
        push 1
        call printc
        add ESP, 4
kraj_wrdec:
        cmp EBX, 0
        je kraj_skroz
        push ESP
        push 1
        call printc
        add ESP,4
        dec EBX
        jmp kraj_wrdec
```

```
kraj_skroz:
        pop EDX
        pop ECX
        pop EBX
        pop EAX
        ret
writedec ENDP


main:
        mov ECX, 0
STARTcall readdec
_loop:  mov nnn, AL
        mov AL, suma
        add AL, nnn
        mov suma, AL
        mov AL, nnn
        dec AL
        jnz _loop
        mov AL, suma
        call writedec
        invoke ExitProcess, 0
end main
```

Пример9

Улаз:
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 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

Излаз:

```
.386
.model flat, stdcall

PUBLIC B
EXTRN LAB2,LAB1,LAB
M equ LAB+91
```

```
STD_OUTPUT_HANDLE equ -11
PUBLIC main

INCLUDELIB kernel32.lib

ExitProcess proto:dword
WriteConsoleA proto :dword, :dword, :dword, :dword, :dword

GetStdHandle proto :dword

.data
dSTART      DB      0
            DB      20
B           DB      1       DUP(0)
B           DB      1       DUP(0)
TEMP  DB    150 - BITS - M
BITS  DB    LAB1 - B - BITS - TEMP + M
stdout DD   0

.code
printc   PROC par1:dword, par2:dword
        ; push EBP
        ; mov EBP, ESP
        push EAX
        push EBX
        push ECX
        cmp stdout, 0
        jne dalje_printc
        invoke GetStdHandle, STD_OUTPUT_HANDLE
        mov stdout, EAX
dalje_printc:
        mov ECX, [par1]
        mov EBX, EBP
        add EBX, 8
        mov EAX, [par2]
        invoke WriteConsoleA, stdout, EAX, ECX, EBX, 0
        pop ECX
        pop EBX
        pop EAX
        ; pop EBP
        ret
printc   ENDP

writedec PROC
        push EAX
        push EBX
        push ECX
         push EDX
```

```
        mov AH, 0
        mov CL, 10
         mov EBX, 0
        mov EDX, 0
        mov DL, AL
        and DL, 80h
        jnz obrni
        jmp ??write
obrni: neg AL
??write:div CL
        xchg AH, AL
        add AL,'0'
        push EAX
        inc EBX
        xchg AH, AL
        cmp AL, 0
        je minusprovera
        mov AH, 0
        jmp ??write
minusprovera:
        cmp DL, 80h
        je dodajminus
        jmp kraj_wrdec
dodajminus:
        push '-'
        push ESP
        push 1
        call printc
        add ESP, 4
kraj_wrdec:
        cmp EBX, 0
        je kraj_skroz
        push ESP
        push 1
        call printc
        add ESP,4
        dec EBX
        jmp kraj_wrdec
kraj_skroz:
        pop EDX
        pop ECX
        pop EBX
        pop EAX
        ret
writedec ENDP


main:
```

```
        mov ECX, 0
        mov AL, B-TEMP+190
_LOOP:          shr AL, 1
        jnc _EVEN
        mov BITS-TEMP+10, AL
        mov AL, BITS-5
        inc AL
        mov LAB1-LAB, AL
        mov AL, LAB2
        inc AL
        dec AL
_EVEN:          jnz _LOOP
        mov AL, 10-TEMP-M-_EVEN
        call writedec
        invoke ExitProcess, 0
end main
```

Пример10

Улаз:

```
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
```

Излаз:

```
.386
.model flat, stdcall

STD_INPUT_HANDLE equ -10
STD_OUTPUT_HANDLE equ -11
PUBLIC main

INCLUDELIB kernel32.lib

ExitProcess proto:dword
ReadConsoleA proto :dword, :dword, :dword, :dword, :dword
WriteConsoleA proto :dword, :dword, :dword, :dword, :dword

GetStdHandle proto :dword

.data
dSTART      DB      0
A       DB      0
B       DB      0
ZBIR    DB      1       DUP(0)
stdin   DD      0
stdout  DD      0

.code
readc   PROC
```

```
        push EBP
        mov EBP, ESP
        push EAX
        push EBX
        push EDX
        cmp stdin, 0
        jne dalje_readc
        invoke GetStdHandle, STD_INPUT_HANDLE
        mov stdin, EAX
dalje_readc:
        sub ESP, 4
        mov EAX, ESP
        sub ESP, 4
        mov EBX, ESP
        invoke ReadConsoleA, stdin, EAX, 1, EBX, 0
        add ESP,4
        pop EAX
        mov [EBP+8], AL
        pop EDX
        pop EBX
        pop EAX
        pop EBP
        ret
readc   ENDP

readdec PROC
        push EBP
        mov EBP, ESP
        push EBX
        push ECX
        push EDX
        mov ECX, 10
        mov EBX, 0
        mov EDX,0
??read: push ECX
        sub ESP, 4
        call readc
        pop EAX
        pop ECX
        cmp AL,'-'
        je jesteminus
        jmp daljeminus
        jesteminus:
        mov EDX,1
        push ECX
        sub ESP, 4
        call readc
        pop EAX
```

```
        pop ECX
daljeminus:
        cmp AL, '0'
        jb kraj_rddec
        cmp AL, '9'
        ja kraj_rddec
        sub AL, '0'
        push EAX
        mov EAX, EBX
        push EDX
        mul ECX
        pop EDX
        mov EBX, EAX
        pop EAX
        and EAX, 0000000fh
        add EBX, EAX
        jmp ??read
kraj_rddec:mov EAX, EBX
        cmp EDX,1
        je  negiraj
        jmp daljenegiraj
        negiraj: neg AL
daljenegiraj:
        pop EDX
        pop ECX
        pop EBX
        pop EBP
        ret
readdec ENDP

printc    PROC par1:dword, par2:dword
        ; push EBP
        ; mov EBP, ESP
        push EAX
        push EBX
        push ECX
        cmp stdout, 0
        jne dalje_printc
        invoke GetStdHandle, STD_OUTPUT_HANDLE
        mov stdout, EAX
dalje_printc:
        mov ECX, [par1]
        mov EBX, EBP
        add EBX, 8
        mov EAX, [par2]
        invoke WriteConsoleA, stdout, EAX, ECX, EBX, 0
        pop ECX
        pop EBX
```

```
        pop EAX
        ; pop EBP
        ret
printc    ENDP

writedec PROC
        push EAX
        push EBX
        push ECX
         push EDX
        mov AH, 0
        mov CL, 10
         mov EBX, 0
        mov EDX, 0
        mov DL, AL
        and DL, 80h
        jnz obrni
        jmp ??write
obrni: neg AL
??write:div CL
        xchg AH, AL
        add AL,'0'
        push EAX
        inc EBX
        xchg AH, AL
        cmp AL, 0
        je minusprovera
        mov AH, 0
        jmp ??write
minusprovera:
        cmp DL, 80h
        je dodajminus
        jmp kraj_wrdec
dodajminus:
        push '-'
        push ESP
        push 1
        call printc
        add ESP, 4
kraj_wrdec:
        cmp EBX, 0
        je kraj_skroz
        push ESP
        push 1
        call printc
        add ESP,4
        dec EBX
        jmp kraj_wrdec
```

```
kraj_skroz:
        pop EDX
        pop ECX
        pop EBX
        pop EAX
        ret
writedec ENDP


main:
        mov ECX, 0
        call readdec
        mov A, AL
        call readdec
        mov B, AL
        call _SABERI
        mov AL, ZBIR
        call writedec
        invoke ExitProcess, 0
_SABERI:        mov AL, A
        add AL, B
        mov ZBIR, AL
        pop EIP
end main
```

Пример11

Улаз:

```
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
```

Излаз:

```
.386
.model flat, stdcall

PUBLIC main

INCLUDELIB kernel32.lib

ExitProcess proto:dword

.data
dSTART      DB      0
BLA     DB      0

.code


main:
        mov ECX, 0
        push AX
        mov AL, BLA
        push AX
        call _POTPR
        push AX
_LABELA1:    push AX
        add AL, BLA
```

```
        jc _LABELA2
        add AL, BLA
_LABELA2:    jnz _LABELA1
        invoke ExitProcess, 0
_POTPR:         push AX
        mov AL, BLA
        test AL, AL
        jnz _LABELA3
_LABELA3:    pop EIP
end main
```

Пример16

Улаз:

```
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
```

Излаз:

```
.386
.model flat, stdcall

STD_OUTPUT_HANDLE equ -11
PUBLIC main

INCLUDELIB kernel32.lib

ExitProcess proto:dword
WriteConsoleA proto :dword, :dword, :dword, :dword, :dword

GetStdHandle proto :dword

.data
dSTART      DB    0
stdout  DD    0

.code
printc   PROC par1:dword, par2:dword
        ; push EBP
        ; mov EBP, ESP
        push EAX
        push EBX
        push ECX
        cmp stdout, 0
        jne dalje_printc
        invoke GetStdHandle, STD_OUTPUT_HANDLE
        mov stdout, EAX
dalje_printc:
        mov ECX, [par1]
        mov EBX, EBP
        add EBX, 8
        mov EAX, [par2]
        invoke WriteConsoleA, stdout, EAX, ECX, EBX, 0
```

```
        pop ECX
        pop EBX
        pop EAX
        ; pop EBP
        ret
printc    ENDP

writedec PROC
        push EAX
        push EBX
        push ECX
         push EDX
        mov AH, 0
        mov CL, 10
         mov EBX, 0
        mov EDX, 0
        mov DL, AL
        and DL, 80h
        jnz obrni
        jmp ??write
obrni: neg AL
??write:div CL
        xchg AH, AL
        add AL,'0'
        push EAX
        inc EBX
        xchg AH, AL
        cmp AL, 0
        je minusprovera
        mov AH, 0
        jmp ??write
minusprovera:
        cmp DL, 80h
        je dodajminus
        jmp kraj_wrdec
dodajminus:
        push '-'
        push ESP
        push 1
        call printc
        add ESP, 4
kraj_wrdec:
        cmp EBX, 0
        je kraj_skroz
        push ESP
        push 1
        call printc
        add ESP,4
```

```
        dec EBX
        jmp kraj_wrdec
kraj_skroz:
        pop EDX
        pop ECX
        pop EBX
        pop EAX
        ret
writedec ENDP


main:
        mov ECX, 0
        mov AL, 40
        call writedec
_LAB1:          clc
        add AL, _LAB1
        invoke ExitProcess, 0
end main
```

Листинг програма

Block.java

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

package sp_dz2;

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

/**
 *
 * @author MB
 */
public class Block {
    private List<Instruction> listInstr;
    private boolean needZ, needC, needP, setC, setPZ;
    private boolean needOptimisation;
    private int criticLine;
    private Block next1, next2;

    public boolean wasInBlock;

    public Block() {
        listInstr = new LinkedList<Instruction>();
        needOptimisation = false;
        criticLine = -1;
        next1= next2 = null;
    }

    public void addInstr(Instruction i){
        listInstr.add(i);
    }

    public void addOnPlace(Instruction i, int place){
        listInstr.add(place, i);
    }

    public Instruction getLastInsrtInBlock(){
        return listInstr.get(listInstr.size() - 1);
    }

    public Instruction getInstruction(int indx){
        return listInstr.get(indx);
    }

    public int getSize(){
        return listInstr.size();
    }
```

```
public String getJumpMnem(int i){
    String s[] = listInstr.get(i).getReplacement().split("[\\s]+");
    if(listInstr.get(i).isHasLabel()){
        return s[1];
    } else {
        return s[0];
    }
}

public Block getNext1() {
    return next1;
}

public void setNext1(Block next1) {
    this.next1 = next1;
}

public Block getNext2() {
    return next2;
}

public void setNext2(Block next2) {
    this.next2 = next2;
}

public int getCriticLine() {
    return criticLine;
}

public boolean isNeedOptimisation() {
    return needOptimisation;
}

public void setCriticLine(int criticLine) {
    this.criticLine = criticLine;
}

public void setNeedOptimisation(boolean needOptimisation) {
    this.needOptimisation = needOptimisation;
}

public boolean isNeedC() {
    return needC;
}

public void setNeedC(boolean needC) {
    this.needC = needC;
}

public boolean isNeedP() {
    return needP;
}
```

```
    public void setNeedP(boolean needP) {
        this.needP = needP;
    }

    public boolean isNeedZ() {
        return needZ;
    }

    public void setNeedZ(boolean needZ) {
        this.needZ = needZ;
    }

    public boolean isSetC() {
        return setC;
    }

    public void setSetC(boolean setC) {
        this.setC = setC;
    }

    public boolean isSetPZ() {
        return setPZ;
    }

    public void setSetPZ(boolean setPZ) {
        this.setPZ = setPZ;
    }

}
```

Instruction.java

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

package sp_dz2;

import java.util.List;

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

    private int opCode;

    private String label, jumpOnLabel;
    private boolean hasLabel;
    private boolean needZ, needC, needP, setC, setPZ;
    private boolean needOptimisation;
    private String replacement;

    public Instruction() {
        jumpOnLabel = "";
    }

    public int getOpCode() {
        return opCode;
    }

    public void setOpCode(int opCode) {
        this.opCode = opCode;
    }

    public String getReplacement() {
        return replacement;
    }

    public void setReplacement(String replacement) {
        this.replacement = replacement;
    }

    public boolean isHasLabel() {
        return hasLabel;
    }

    public void setHasLabel(boolean hasLabel) {
        this.hasLabel = hasLabel;
    }

    public String getJumpOnLabel() {
```

```
    return jumpOnLabel;
}

public void setJumpOnLabel(String jumpOnLabel) {
    this.jumpOnLabel = jumpOnLabel;
}

public String getLabel() {
    return label;
}

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

public boolean isNeedC() {
    return needC;
}

public void setNeedC(boolean needC) {
    this.needC = needC;
}

public boolean isNeedOptimisation() {
    return needOptimisation;
}

public void setNeedOptimisation(boolean needOptimisation) {
    this.needOptimisation = needOptimisation;
}

public boolean isNeedP() {
    return needP;
}

public void setNeedP(boolean needP) {
    this.needP = needP;
}

public boolean isNeedZ() {
    return needZ;
}

public void setNeedZ(boolean needZ) {
    this.needZ = needZ;
}

public boolean isSetC() {
    return setC;
}

public void setSetC(boolean setC) {
    this.setC = setC;
```

```
    }

    public boolean isSetPZ() {
        return setPZ;
    }

    public void setSetPZ(boolean setPZ) {
        this.setPZ = setPZ;
    }

}
```

Main.java

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

package sp_dz2;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

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

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        try {
            // TODO code application logic here
            ProcessInputFile pif = new ProcessInputFile("ulaz.txt");
            pif.work();
        } catch (IOException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null,
ex);
        }

    }

}
```

ProcessInputFile.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package sp_dz2;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
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 ProcessInputFile {

    private BufferedReader src;
    private BufferedWriter izlaz;
    private int nsegs, nsyms, nrels;
    private TableSymbols table;
    private TableOperation tableOP;
    private TableRelocations tableRel;
    private int txtLen, bssLen, datLen;
    private int txtBase, bssBase, datBase;
    private static int[] mem;
    private static boolean isINI, isOTI;
    private static boolean hasUSE, hasDEF, hasEQU;
    private List<Block> listBlocks;

    private static String[] names;
    private static int[] locs;


    public ProcessInputFile(String fileName) throws IOException {
        nsegs = nsyms = nrels = 0;
        bssLen = txtLen = datLen = 0;
        bssBase = txtBase = datBase = 0;
        isINI = isOTI = false;
        hasDEF = hasEQU = hasUSE = false;
        mem = new int[256];
        table = new TableSymbols();
        tableOP = new TableOperation();
        tableRel = new TableRelocations();
        listBlocks = new LinkedList<Block>();
```

```
        try {
            src = new BufferedReader(new FileReader(fileName));
            izlaz = new BufferedWriter(new FileWriter("izlaz.txt"));
        } catch (FileNotFoundException ex) {
            System.out.println("Neuspesno otvaranje fajla ili kreiranje
izlaznog fajla!");

Logger.getLogger(ProcessInputFile.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }

    public int getNrels() {
        return nrels;
    }

    public void setNrels(int nrels) {
        this.nrels = nrels;
    }

    public int getNsegs() {
        return nsegs;
    }

    public void setNsegs(int nsegs) {
        this.nsegs = nsegs;
    }

    public int getNsyms() {
        return nsyms;
    }

    public void setNsyms(int nsyms) {
        this.nsyms = nsyms;
    }

    public static boolean isIsINI() {
        return isINI;
    }

    public static boolean isIsOTI() {
        return isOTI;
    }

    public static boolean isHasDEF() {
        return hasDEF;
    }

    public static void setHasDEF(boolean hasDEF) {
        ProcessInputFile.hasDEF = hasDEF;
    }

    public static boolean isHasEQU() {
```

```
        return hasEQU;
    }

    public static void setHasEQU(boolean hasEQU) {
        ProcessInputFile.hasEQU = hasEQU;
    }

    public static boolean isHasUSE() {
        return hasUSE;
    }

    public static void setHasUSE(boolean hasUSE) {
        ProcessInputFile.hasUSE = hasUSE;
    }

    private void processLinesInFile() {
        String line_for_work = null;
        int numTaraba = 0;
        boolean isLink = false, firstPass = true;
        try {
            line_for_work = src.readLine();
            while (line_for_work != null) {
                if(line_for_work.equalsIgnoreCase("")
                    || line_for_work.equalsIgnoreCase(" ")){
                    line_for_work = src.readLine();
                    continue;
                }
                if (line_for_work.contains("LINK")) {
                    isLink = true;
                    line_for_work = src.readLine();
                    continue;
                }
                if (line_for_work.contains("#")) {
                    numTaraba++;
                    line_for_work = src.readLine();
                    continue;
                }
                if (!line_for_work.contains("LINK") ||
!line_for_work.contains("#")) {
                    if (isLink) {
                        String[] splitNum = line_for_work.split("[\\s]+");
                        nsegs = Integer.parseInt(splitNum[0]);
                        nsyms = Integer.parseInt(splitNum[1]);
                        nrels = Integer.parseInt(splitNum[2]);
                        isLink = false;
                        names = new String[nrels];
                        locs = new int[nrels];
                    }
                    if (numTaraba == 1) {
                        processSegmentLine(line_for_work);
                    } else if (numTaraba == 2) {
                        processSymbolLine(line_for_work);
                    } else if (numTaraba == 3) {
```

```
                    processRelocLine(line_for_work);
                } else if (numTaraba == 4) {
                    processDataLine(line_for_work, firstPass);
                    firstPass = false;
                }
            }
            line_for_work = src.readLine();
        }
    } catch (IOException ex) {
        System.out.println("Neuspesno citanje linije iz ulaznog
fajla!");

Logger.getLogger(ProcessInputFile.class.getName()).log(Level.SEVERE, null,
ex);
    }
}

private void processSegmentLine(String line) {
    String[] segs = line.split("[\\s]+");
    if (segs[1].equalsIgnoreCase(".bss")) {
        bssBase = Integer.parseInt(segs[2]);
        bssLen = Integer.parseInt(segs[3]);
    }
    if (segs[1].equalsIgnoreCase(".text")) {
        txtBase = Integer.parseInt(segs[2]);
        txtLen = Integer.parseInt(segs[3]);
    }
    if (segs[1].equalsIgnoreCase(".data")) {
        datBase = Integer.parseInt(segs[2]);
        datLen = Integer.parseInt(segs[3]);
    }
}

private void processSymbolLine(String line) {
    String[] syms = line.split("[\\s]+");
    if (syms[4].equalsIgnoreCase("U")) {
        table.enter(syms[1], Integer.parseInt(syms[2]),
Integer.parseInt(syms[3]), true, false);
        if (!hasUSE) {
            hasUSE = true;
        }
    }
    if (syms[4].equalsIgnoreCase("D")) {
        table.enter(syms[1], Integer.parseInt(syms[2]),
Integer.parseInt(syms[3]), false, true);
        if (!hasDEF) {
            hasDEF = true;
        }
    }
    if (syms[4].equalsIgnoreCase("E")) {
        table.enter(syms[1], Integer.parseInt(syms[2]),
Integer.parseInt(syms[3]), true, true);
        if (!hasEQU) {
```

```
                hasEQU = true;
            }
        }
    }

    private void processRelocLine(String line) {
        String[] rels = line.split("[\\s]+");
        tableRel.add(rels[5], Integer.parseInt(rels[1]),
Integer.parseInt(rels[2]), Integer.parseInt(rels[3]), rels[4], rels[6],
rels.length == 8 ? Integer.parseInt(rels[7]) : -1);
    }

    private void processDataLine(String line, boolean firstPass) {
        String[] data = line.split("[\\s]+");
        int place = 0;
        if (firstPass) {
            place = txtBase;
        } else {
            place = datBase;
        }
        for (int i = 0; i < data.length; i++) {
            mem[place++] = Integer.parseInt(data[i]);

        }
        if (firstPass) {
            place = bssBase;
            for (int i = 0; i < bssLen; i++) {
                mem[place++] = 0;
            }
        }
        for (int i = txtBase; i < txtLen; i++) {
                if (tableOP.oneByteInstr(mem[i])) {
                    if (mem[i] == 10) isINI = true;
                    if (mem[i] == 14)isOTI = true;
                    continue;
                }
                i++;
            }
    }

    private void refactorCode() {
        Block b = new Block();
        getLabels(locs, names);
        for (int i = txtBase; i < txtBase + txtLen; ) {
            Instruction instr = new Instruction();
            instr.setOpCode(mem[i]);
            boolean optim = tableOP.needOptimisation(mem[i]);
            boolean isADD = false;

            if (optim) {
                b.setNeedOptimisation(true);
                b.setCriticLine(b.getSize());
            }
```

```
            if (tableOP.oneByteInstr(mem[i])) {
                instr.setReplacement(repleceOneByteInstr(mem[i]));
                if (tableOP.isInstructionThatSetsC(mem[i])) {
                    instr.setSetC(true);
                    b.setSetC(true);
                }
                if (tableOP.isInstructionThatSetsPZ(mem[i])) {
                    instr.setSetPZ(true);
                    b.setSetPZ(true);
                }
                //adds label to the line if there is any
                for (int j = 0; j < locs.length && names[j]!= null; j++) {
                    if (locs[j] == i) {
                        instr.setHasLabel(true);
                        instr.setLabel("_" + names[j] + ":");

                        //add previous block to graph if not first
instruction
                        if (i != txtBase && b.getSize()!= 0) {
                            listBlocks.add(b);

                            //make new block and add instruction
                            b = new Block();
                            b.addInstr(instr);
                            listBlocks.get(listBlocks.size() -
1).setNext1(b);
                        } else {
                            b.addInstr(instr);
                        }
                        isADD = true;
                        break;
                    }
                }
                if(!isADD){
                    for (int j = 0; j < table.getSize(); j++) {
                        if(table.getListsym().get(j).isGlobdef() &&
!table.getListsym().get(j).isGlobuse()
                                && table.getListsym().get(j).getValue() ==
i){
                            instr.setHasLabel(true);

instr.setLabel(table.getListsym().get(j).getName());
                            break;
                        }
                    }
                    if (i != txtBase && instr.isHasLabel()) {
                        listBlocks.add(b);

                        //make new block and add instruction
                        b = new Block();
                        b.addInstr(instr);
                        listBlocks.get(listBlocks.size() - 1).setNext1(b);
```

```
                    } else {
                        b.addInstr(instr);
                    }
                }
                i++;
                continue;
            }

            if (tableOP.twoBytesInstr(mem[i])) {
                String splitedL[];// = tableRel.getAdrField(i +
1).split("[\\s]+");
                String label = "";

                if(tableRel.getAdrField(i + 1) == null) {
                    splitedL = new String[0];
                    label = Integer.toString(mem[i + 1]);
                } else {
                    splitedL = makeAdrPart(i +
1).split("[\\s]+");//tableRel.getAdrField(i + 1).split("[\\s]+");
                }

                for (int j = 0; j < splitedL.length && splitedL[j] != null;
j++) {
                    if(splitedL[j].equalsIgnoreCase("+") ||
splitedL[j].equalsIgnoreCase("-")){
                        label += splitedL[j];
                        continue;
                    }
                    boolean added = false;
                    for (int k = 0; k < names.length && names[k] != null;
k++) {
                        if(splitedL[j].equalsIgnoreCase(names[k])){
                            String nameLab = "_" + names[k];
                            if(!label.equalsIgnoreCase("")) label +=
nameLab;
                            else label +=nameLab;
                            added = true;
                        }
                    }
                    if (!added) {
                            if(!label.equalsIgnoreCase("")) label +=
splitedL[j];
                            else label += splitedL[j];
                        }
                    }

                instr.setReplacement(replaceTwoBytesInstr(mem[i], label));
                instr.setJumpOnLabel(label);

                if (tableOP.jumpInstrC(mem[i])) {
                    instr.setNeedC(true);
                    b.setNeedC(true);
                }
```

```
                if (tableOP.jumpInstrPZ(mem[i])) {
                    instr.setNeedP(true);
                    instr.setNeedZ(true);
                    b.setNeedP(true);
                    b.setNeedZ(true);
                }

                for (int j = 0; j < locs.length && names[j]!= null; j++) {
                    if (locs[j] == i) {
                        instr.setHasLabel(true);
                        instr.setLabel("_" + names[j] + ":");
                        break;
                    }
                }

                if(instr.isHasLabel() && tableOP.isJumpInstr(mem[i])){
                    listBlocks.add(b);
                    b = new Block();
                    b.addInstr(instr);
                    listBlocks.get(listBlocks.size() - 1).setNext1(b);
                    listBlocks.add(b);
                    b = new Block();
                    listBlocks.get(listBlocks.size() - 1).setNext1(b);
                }else {
                    b.addInstr(instr);
                    if (tableOP.isJumpInstr(mem[i])) {
                        listBlocks.add(b);
                        b = new Block();
                        listBlocks.get(listBlocks.size() - 1).setNext1(b);
                    }
                }
                i+=2;
            }
        }
        listBlocks.add(b);
        setBlock2();
        for (int i = 0; i < listBlocks.size(); i++) {
            optimise(listBlocks.get(i));
            writeCodeFromBlock(listBlocks.get(i));
        }
    }

    private void getLabels( int[] locs, String[] names){
        int pos = 0;
        for (int i = 0; i < tableRel.getListZapis().size(); i++) {
            if (tableRel.getZapis(i).getRef() ==
tableRel.getZapis(i).getSeg()
                    && tableRel.getZapis(i).getRef() == 1
                    && !tableRel.getZapis(i).getType().contains("S")) {
                boolean inList = false;
                for (int j = 0; j < names.length && names[j] != null; j++)
{
```

```
                    if
(tableRel.getZapis(i).getName().equalsIgnoreCase(names[j])) {
                        inList = true;
                    }
                }
                if (!inList) {
                    locs[pos] = tableRel.getZapis(i).getLocOrLen() == -1 ?
mem[tableRel.getZapis(i).getLoc()] : tableRel.getZapis(i).getLocOrLen();
                    names[pos++] = tableRel.getZapis(i).getName();
                }
            }
        }
    }

    //sets all second blocks for jump instruction for all blocks
    private void setBlock2() {
        for (int i = 0; i < listBlocks.size(); i++) {
            Block block = listBlocks.get(i);
            if(tableOP.isJumpInstr(block.getLastInsrtInBlock().getOpCode())
&&

!block.getLastInsrtInBlock().getJumpOnLabel().equalsIgnoreCase("") &&

!block.getLastInsrtInBlock().getJumpOnLabel().equalsIgnoreCase(" ") ){
                for (int j = 0; j < listBlocks.size(); j++) {
                    Block block1 = listBlocks.get(j);
                    if(block1.getInstruction(0).isHasLabel() &&

block1.getInstruction(0).getLabel().equalsIgnoreCase(block.getLastInsrtInBl
ock().getJumpOnLabel() + ":")){
                        block.setNext2(block1);
                    }
                }
            }
        }
    }

    private static boolean optimise(Block block) {
        if (!block.isNeedOptimisation()) {
            return false;
        }
        block.wasInBlock = true;
        if (TableOperation.jumpInstrPZ(block.getInstruction(block.getSize()
- 1).getOpCode())) {
            for (int i = block.getCriticLine(); i < block.getSize() - 1 -
1; i++) {
                if
(TableOperation.isInstructionThatSetsPZ(block.getInstruction(i).getOpCode()
)) {
                    return false;
                }
            }
            Instruction i = new Instruction();
```

```
            i.setReplacement("test AL, AL");
            block.addOnPlace(i, block.getCriticLine() + 1);
            return true;
        } else if (block.getNext1() != null && tryDeeper(block.getNext1()))
{
            Instruction i = new Instruction();
            i.setReplacement("dec AL");
            block.addOnPlace(i, block.getCriticLine() + 1);
            i = new Instruction();
            i.setReplacement("inc AL");
            block.addOnPlace(i, block.getCriticLine() + 1);
        } else if (block.getNext2() != null && tryDeeper(block.getNext2()))
{
            Instruction i = new Instruction();
            i.setReplacement("dec AL");
            block.addOnPlace(i, block.getCriticLine() + 1);
            i = new Instruction();
            i.setReplacement("inc AL");
            block.addOnPlace(i, block.getCriticLine() + 1);
        }

        return false;
    }

        // try on that branch
    private static boolean tryDeeper(Block next) {
      if(next.wasInBlock)return false;
      next.wasInBlock=true;
      for(int i=0; i<next.getSize(); i++)

      if(TableOperation.isInstructionThatSetsPZ(next.getInstruction(i).getO
pCode()))return false;
      if(TableOperation.jumpInstrPZ(next.getInstruction(next.getSize() -
1).getOpCode()))return true;
      if(next.getNext1()!=null && tryDeeper(next.getNext1())) return true;
      if(next.getNext2()!=null && tryDeeper(next.getNext2())) return true;
      return false;
    }

    private void writeCodeFromBlock(Block block){
        for (int i = 0; i < listBlocks.size(); i++) {
            listBlocks.get(i).wasInBlock = false;
        }
        for (int i = 0; i < block.getSize(); i++) {
            try {
                if(block.getInstruction(i).getReplacement() != null){
                    if(block.getInstruction(i).isHasLabel()){
                        izlaz.write(block.getInstruction(i).getLabel() +
"\t");
                    }else {
                        izlaz.write("         ");
                    }
                    izlaz.write(block.getInstruction(i).getReplacement());
```

```
                    izlaz.newLine();
                }
            } catch (IOException ex) {
                System.out.println("Neuspesan upis u fajl!");

Logger.getLogger(ProcessInputFile.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 String repleceOneByteInstr(int num) {
        String rep = "";

        if (num == 2) {
            rep = "clc";
        }
        if (num == 3) {
            rep = "xor CL, CL";
        }
        if (num == 4) {
            rep = "cmc";
        }
        if (num == 5) {
            rep = "inc AL";
        }
        if (num == 6) {
            rep = "dec AL";
        }
        if (num == 7) {
            rep = "inc CL";
        }
        if (num == 8) {
            rep = "dec CL";
        }
        if (num == 9) {
            rep = "mov CL, AL";
        }
        if (num == 10) {
            rep = "call readdec";
        }
        if (num == 14) {
            rep = "call writedec";
        }
```

```
    if (num == 19) {
        rep = "push AX";
    }
    if (num == 20) {
        rep = "pop AX";
    }
    if (num == 21) {
        rep = "shl AL, 1";
    }
    if (num == 22) {
        rep = "shr AL, 1";
    }
    if (num == 23) {
        rep = "pop EIP";
    }
    if (num == 24) {
        rep = "invoke ExitProcess, 0";
    }

    return rep;
}

private String replaceTwoBytesInstr(int num, String adr) {
    String rep = "";

    if (num == 25) {
        if(isNumber(adr)){
            rep = "mov AL, [" + adr + "]";
        } else {
            rep = "mov AL, " + adr;
        }
    }
    if (num == 26) {
        rep = "mov AL, [" + adr + " + ECX]";
    }
    if (num == 27) {
        if(isNumber(adr)){
            rep = "mov AL, " + adr;
        } else {
            rep = "mov AL, " + adr;
        }
    }
    if (num == 30) {
        if(isNumber(adr)){
            rep = "mov [" + adr + "], AL";
        } else {
            rep = "mov " + adr + ", AL";
        }
    }
    if (num == 31) {
        rep = "mov [" + adr + " + ECX]" + ", AL";
    }
    if (num == 32) {
```

```
    if(isNumber(adr)){
        rep = "add AL, [" + adr + "]";
    } else {
        rep = "add AL, " + adr;
    }
}
if (num == 33) {
    rep = "add AL, [" + adr + " + ECX]";
}
if (num == 38) {
    if(isNumber(adr)){
        rep = "sub AL, [" + adr + "]";
    } else {
        rep = "sub AL, " + adr;
    }
}
if (num == 39) {
    rep = "sub AL, [" + adr + " + ECX]";
}
if (num == 44) {
    if(isNumber(adr)){
        rep = "cmp AL, [" + adr + "]";
    } else {
        rep = "cmp AL, " + adr;
    }
}
if (num == 47) {
    if(isNumber(adr)){
        rep = "and AL, [" + adr + "]";
    } else {
        rep = "and AL, " + adr;
    }
}
if (num == 50) {
    if (isNumber(adr)) {
        rep = "or AL, [" + adr + "]";
    } else {
        rep = "or AL, " + adr;
    }
}
if (num == 53) {
    rep = "jmp " + adr;
}
if (num == 54) {
    rep = "jz " + adr;
}
if (num == 55) {
    rep = "jnz " + adr;
}
if (num == 56) {
    rep = "jg " + adr;
}
if (num == 57) {
```

```
            rep = "jng " + adr;
        }
        if (num == 58) {
            rep = "jnc " + adr;
        }
        if (num == 59) {
            rep = "jc " + adr;
        }
        if (num == 60) {
            rep = "call " + adr;
        }

        return rep;
    }

    //insert symbols in .data sections
    private void insertData() {
        String inserted[] = new String[nrels];
        int loc = 0;
        try {
            boolean insertedZ = false;
            //insert reserved location if any

            //insert DS
            if (bssLen > 0) {
                int bss_len = 0;
                for (int i = 0; i < tableRel.getListZapis().size(); i++) {
                    if (tableRel.getZapis(i).getRef() == 2 &&
!tableRel.getZapis(i).getType().contains("S")) {
                        for (int j = 0; j < inserted.length && inserted[j]
!= null; j++) {
                            if
(inserted[j].equalsIgnoreCase(tableRel.getZapis(i).getName())) {
                                insertedZ = true;
                                break;
                            }
                        }
                        if(!insertedZ){
                            bss_len += tableRel.getZapis(i).getLocOrLen();
                            inserted[loc++] =
tableRel.getZapis(i).getName();
                        }
                        insertedZ = false;
                    }
                }
                bss_len = bssLen - bss_len;

                if (bss_len > 0) {
                    izlaz.write("\tDB\t" + bssLen + "\tDUP(0)");
                    izlaz.newLine();
                }
            }
```

```
            //insert DC
            inserted = new String[nrels];
            insertedZ = false;
            loc = 0;
            if (datLen > 0) {
                int dat_len = 0;
                int [] locs = new int[nrels];
                for (int i = 0; i < tableRel.getListZapis().size(); i++) {
                    if (tableRel.getZapis(i).getRef() == 3 &&
!tableRel.getZapis(i).getType().contains("S")) {
                        for (int j = 0; j < inserted.length && inserted[j]
!= null; j++) {
                            if
(inserted[j].equalsIgnoreCase(tableRel.getZapis(i).getName())) {
                                insertedZ = true;
                                break;
                            }
                        }
                        if(!insertedZ){
                            locs[loc] = tableRel.getZapis(i).getLocOrLen();
                            inserted[loc++] =
tableRel.getZapis(i).getName();
                            dat_len ++;
                        }
                        insertedZ = false;
                    }
                }
                dat_len = datLen - dat_len;

                if (dat_len > 0) {
                    for (int i = datBase; i < datBase + datLen; i++) {
                        for (int j = 0; j < locs.length; j++) {
                            if( locs[j] == i){
                                insertedZ = true;
                                break;
                            }
                        }
                        if(!insertedZ){
                            izlaz.write("\tDB\t" + mem[i]);
                            izlaz.newLine();
                        }
                        insertedZ = false;
                    }
                }
            }

            //insert globdef symbols
            for (int i = 0; i < table.getSize(); i++) {
                if(table.getListsym().get(i).isGlobdef() &&
!table.getListsym().get(i).isGlobuse() &&
                        table.getListsym().get(i).getSeg() != 1){
                    // to do
                    // if more than one DS Def
```

```
                    izlaz.write(table.getListsym().get(i).getName());
                    izlaz.write("\tDB\t");
                    if(table.getListsym().get(i).getSeg() == 2){
                        izlaz.write(Integer.toString(bssLen));
                        izlaz.write("\tDUP(0)");
                    } else if(table.getListsym().get(i).getSeg() == 3){

izlaz.write(Integer.toString(mem[table.getListsym().get(i).getValue()]));
                    }
                    izlaz.newLine();
                }
            }

            //insert other symbols
            insertedZ = false;
            inserted = new String[nrels];
            loc = 0;
            for (int i = 0; i < nrels; i++) {
                if (tableRel.getZapis(i).getRef() != 1 &&
!tableRel.getZapis(i).getType().contains("S")) {
                    //tableRel.getZapis(i).getLocOrLen() != -1 &&
                    for (int j = 0; j < inserted.length && inserted[j] !=
null; j++) {
                        if
(inserted[j].equalsIgnoreCase(tableRel.getZapis(i).getName())) {
                            insertedZ = true;
                            break;
                        }
                    }
                    if (!insertedZ) {

                        izlaz.write(tableRel.getZapis(i).getName());
                        izlaz.write("\tDB\t");
                        if (tableRel.getZapis(i).getRef() == 2) {

izlaz.write(Integer.toString(tableRel.getZapis(i).getLocOrLen()));
                            izlaz.write("\tDUP(0)");
                        } else if (tableRel.getZapis(i).getRef() == 3) {

izlaz.write(makeAdrPart(tableRel.getZapis(i).getLocOrLen()));
                        }
                        izlaz.newLine();
                        inserted[loc++] = tableRel.getZapis(i).getName();
                        insertedZ = false;
                    }
                }
            }

            //insert for INI and OTI
            if (isINI) {
                izlaz.write("stdin\tDD\t0");
                izlaz.newLine();
            }
```

```
            if (isOTI) {
                izlaz.write("stdout\tDD\t0");
                izlaz.newLine();
            }
        } catch (IOException ex) {
            System.out.println("Neuspesan upis u izlazni fajl!");

Logger.getLogger(ProcessInputFile.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }

    //make adres part for DC
    private String makeAdrPart(int loc){
        int value = mem[loc], valueRes = 0, numOp = 0;
        String val = "";
        boolean first = false;
        for (int i = 0; i < tableRel.getListZapis().size(); i++) {
            if(tableRel.getListZapis().get(i).getLoc() == loc){
                numOp++;
                if(!first){

if(tableRel.getListZapis().get(i).getOp().equalsIgnoreCase("+")){
                        valueRes +=
symbolValue(tableRel.getListZapis().get(i).getName());

                    } else
if(tableRel.getListZapis().get(i).getOp().equalsIgnoreCase("-")){
                        valueRes -=
symbolValue(tableRel.getListZapis().get(i).getName());
                        val += "- ";
                    }
                    first = true;
                    val+= tableRel.getZapis(i).getName();
                } else {
                    val += " " + tableRel.getListZapis().get(i).getOp() + "
" + tableRel.getListZapis().get(i).getName();

if(tableRel.getListZapis().get(i).getOp().equalsIgnoreCase("+")){
                        valueRes +=
symbolValue(tableRel.getListZapis().get(i).getName());

                    } else
if(tableRel.getListZapis().get(i).getOp().equalsIgnoreCase("-")){
                        valueRes -=
symbolValue(tableRel.getListZapis().get(i).getName());
                    }
                }
            }
        }
        if(numOp > 0 && !isNumber(val)){
            if (Math.abs(valueRes) - Math.abs(value) == 0) return val;
            else {
```

```
                int num = 0;
                if (value >= 0) {
                    num = value - valueRes;
                } else {
                    if(Math.abs(value) > Math.abs(valueRes)){

                        num = value - valueRes;
                    } else{
                        num = Math.abs(valueRes) - Math.abs(value);
                    }
                }

                if(val.charAt(0) == '-'){
                    val = Integer.toString(num) + " " + val;
                } else {
                    if(num > 0){
                        val = val + " + " + Integer.toString(num);
                    } else {
                        val += " " + Integer.toString(num);
                    }
                }
            }
        } else {
            return Integer.toString(value);
        }

        return val;
    }

    private int symbolValue(String name){
        int val = 0;
        for (int i = 0; i < tableRel.getListZapis().size(); i++) {
            if(tableRel.getZapis(i).getName().equalsIgnoreCase(name) &&
                    !tableRel.getZapis(i).getType().contains("S")){
                if(tableRel.getZapis(i).getRef() == 3){
                    val = tableRel.getZapis(i).getLocOrLen();
//tableRel.getZapis(i).getLoc()
                    break;
                } else if(tableRel.getZapis(i).getRef() == 2){
                    //to do
                    //if more DS
                    val = bssBase + bssLen -
tableRel.getZapis(i).getLocOrLen();
/*                    String inserted[]  =new String[nrels];
                    int loc = 0;

                    for (int j = 0; j < tableRel.getListZapis().size();
j++) {
                        boolean added = false;

if(tableRel.getListZapis().get(j).getName().equalsIgnoreCase(name) ||
                                tableRel.getListZapis().get(j).getRef() !=
2 ||
```

```
tableRel.getListZapis().get(j).getType().contains("S")) continue;
                        for (int k = 0; k < inserted.length && inserted[k]
!= null; k++) {

if(tableRel.getListZapis().get(j).getName().equalsIgnoreCase(name) ||
                                    tableRel.getListZapis().get(j).getRef()
!= 2 ||

tableRel.getListZapis().get(j).getType().contains("S")) continue;

if(tableRel.getListZapis().get(j).getName().equalsIgnoreCase(inserted[k])){
                                added = true;
                                break;
                            }
                        }
                        if(!added){
                            inserted[loc++] =
tableRel.getListZapis().get(j).getName();
                            val -= tableRel.getZapis(i).getLocOrLen();
                        }
                    }
                    break;
 */
                } else if(tableRel.getZapis(i).getRef() == 1){
                    for (int j = 0; j < names.length; j++) {
                        if(name.equalsIgnoreCase(names[j])){
                            val = locs[j];
                            break;
                        }
                    }
                }
                return val;
            }
        }
        for (int i = 0; i < table.getSize(); i++) {
            if(table.getListsym().get(i).getName().equalsIgnoreCase(name)){
                return table.getListsym().get(i).getValue();
            }
        }
        return val;
    }

    public void work() {
        System.out.println("Emulating...");
        processLinesInFile();
        try {
            ReadWrite.header(izlaz, table);

            insertData();

            izlaz.newLine();
            izlaz.write(".code");
```

```
            izlaz.newLine();

            if (isINI) {
                ReadWrite.read(izlaz);
                izlaz.newLine();
            }
            if (isOTI) {
                ReadWrite.write(izlaz);
            }

            izlaz.newLine(); izlaz.newLine();
            izlaz.write("main:");
            izlaz.newLine();
            izlaz.write("         mov ECX, 0");
            izlaz.newLine();

            refactorCode();

            izlaz.write("end main");
            izlaz.newLine();

            src.close();
            izlaz.close();

        } catch (IOException ex) {

Logger.getLogger(ProcessInputFile.class.getName()).log(Level.SEVERE, null,
ex);
        }
        System.out.println("Emulation completed!");
    }
}
```

ReadWrite.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package sp_dz2;

import java.io.BufferedWriter;
import java.io.IOException;

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

    static public void header(BufferedWriter izlaz, TableSymbols table)
throws IOException {
        izlaz.write(".386");izlaz.newLine();
        izlaz.write(".model flat, stdcall");
        izlaz.newLine(); izlaz.newLine();

        if (ProcessInputFile.isHasDEF()) {
            izlaz.write("PUBLIC ");
            int num = table.getNumGlobDef();
            for (int i = 0; i < table.getSize(); i++) {
                if (table.getListsym().get(i).isGlobdef() &&
!table.getListsym().get(i).isGlobuse()) {
                    izlaz.write(table.getListsym().get(i).getName());
                    num--;
                    if(num > 0) izlaz.write(",");
                }
            }
            izlaz.write(";"); izlaz.newLine();
        }
        if (ProcessInputFile.isHasUSE()) {
            izlaz.write("EXTRN ");
            int num = table.getNumGlobUse();
            for (int i = 0; i < table.getSize(); i++) {
                if (!table.getListsym().get(i).isGlobdef() &&
table.getListsym().get(i).isGlobuse()) {
                    izlaz.write(table.getListsym().get(i).getName());
                    num--;
                    if(num > 0) izlaz.write(",");
                }
            }
            izlaz.write(";"); izlaz.newLine();
        }
        if (ProcessInputFile.isHasEQU()) {
            for (int i = 0; i < table.getSize(); i++) {
                if (table.getListsym().get(i).isGlobdef() &&
table.getListsym().get(i).isGlobuse()) {
                    izlaz.write(table.getListsym().get(i).getName());
```

```
                    izlaz.write(" equ ");
                    boolean addedName = false;
                    for (int j = 0; j <
TableRelocations.getListZapis().size(); j++) {
                        if(TableRelocations.getListZapis().get(i).getLoc()
== 0 &&

TableRelocations.getListZapis().get(i).getSeg() == 0){

izlaz.write(TableRelocations.getListZapis().get(i).getName());
                            addedName = true;
                            break;
                        }
                    }
                    if(!addedName){

izlaz.write(Integer.toString(table.getListsym().get(i).getValue()));
                    } else {
                        if(table.getListsym().get(i).getValue() >= 0){
                            izlaz.write("+");

izlaz.write(Integer.toString(table.getListsym().get(i).getValue()));
                        } else {
                            //izlaz.write("-");

izlaz.write(Integer.toString(table.getListsym().get(i).getValue()));
                        }
                    }
                }
            }
            izlaz.write(";"); izlaz.newLine();
        }
        if (ProcessInputFile.isIsINI()) {//INI
            izlaz.write("STD_INPUT_HANDLE equ -10");
            izlaz.newLine();
        }
        if (ProcessInputFile.isIsOTI()) {//OTI
            izlaz.write("STD_OUTPUT_HANDLE equ -11");
            izlaz.newLine();
        }

        izlaz.write("PUBLIC main");izlaz.newLine();izlaz.newLine();
        izlaz.write("INCLUDELIB
kernel32.lib");izlaz.newLine();izlaz.newLine();

        izlaz.write("ExitProcess proto:dword");izlaz.newLine();

        if (ProcessInputFile.isIsINI()) {//INI
            izlaz.write("ReadConsoleA proto :dword, :dword, :dword, :dword,
:dword  ");
            izlaz.newLine();
        }
        if (ProcessInputFile.isIsOTI()) {//OTI
```

```
            izlaz.write("WriteConsoleA proto :dword, :dword, :dword,
:dword, :dword");
            izlaz.newLine();
        }
        izlaz.newLine();

        if (ProcessInputFile.isIsINI() || ProcessInputFile.isIsOTI())
{//INI || OTI
            izlaz.write("GetStdHandle proto :dword");
            izlaz.newLine();izlaz.newLine();
        }
        //DATA sekcija
        izlaz.write(".data");izlaz.newLine();
        izlaz.write("dSTART\tDB\t0");izlaz.newLine();

    }

    static public void read(BufferedWriter izlaz) throws IOException {
        String[] s = {"readc  PROC",
            "         push EBP",
            "         mov EBP, ESP",
            "         push EAX",
            "         push EBX",
            "         push EDX",
            "         cmp stdin, 0",
            "         jne dalje_readc",
            "         invoke GetStdHandle, STD_INPUT_HANDLE",
            "         mov stdin, EAX",
            "dalje_readc:",
            "         sub ESP, 4",
            "         mov EAX, ESP",
            "         sub ESP, 4",
            "         mov EBX, ESP",
            "         invoke ReadConsoleA, stdin, EAX, 1, EBX, 0",
            "         add ESP,4",
            "         pop EAX",
            "         mov [EBP+8], AL",
            "         pop EDX",
            "         pop EBX",
            "         pop EAX",
            "         pop EBP",
            "         ret",
            "readc   ENDP",
            "readdec    PROC",
            "         push EBP",
            "         mov EBP, ESP",
            "         push EBX",
            "         push ECX",
            "         push EDX",
            "         mov ECX, 10",
            "         mov EBX, 0",
            "         mov EDX,0",
            "??read:    push ECX",
```

```
            "         sub ESP, 4",
            "         call readc",
            "         pop EAX",
            "         pop ECX",
            "         cmp AL,'-'",
            "         je jesteminus",
            "         jmp daljeminus",
            "         jesteminus: ",
            "         mov EDX,1",
            "         push ECX",
            "         sub ESP, 4",
            "         call readc",
            "         pop EAX",
            "         pop ECX",
            "daljeminus:",
            "         cmp AL, '0'",
            "         jb kraj_rddec",
            "         cmp AL, '9'",
            "         ja kraj_rddec",
            "         sub AL, '0'",
            "         push EAX",
            "         mov EAX, EBX",
            "         push EDX",
            "         mul ECX",
            "         pop EDX",
            "         mov EBX, EAX",
            "         pop EAX",
            "         and EAX, 0000000fh",
            "         add EBX, EAX",
            "         jmp ??read",
            "kraj_rddec:mov EAX, EBX",
            "         cmp EDX,1",
            "         je  negiraj",
            "         jmp daljenegiraj",
            "         negiraj: neg AL",
            "daljenegiraj:",
            "         pop EDX",
            "         pop ECX",
            "         pop EBX",
            "         pop EBP",
            "         ret",
            "readdec   ENDP"};

        for (int i = 0; i < s.length; i++) {
            izlaz.write(s[i]);
            izlaz.newLine();
            if (s[i].equals("readc   ENDP")) {
                izlaz.newLine();
            }
        }

    }
```

```
static public void write(BufferedWriter izlaz) throws IOException {
    String[] s = {"printc PROC par1:dword, par2:dword",
        "         ; push EBP",
        "         ; mov EBP, ESP",
        "         push EAX",
        "         push EBX",
        "         push ECX",
        "         cmp stdout, 0",
        "         jne dalje_printc",
        "         invoke GetStdHandle, STD_OUTPUT_HANDLE",
        "         mov stdout, EAX",
        "dalje_printc:",
        "         mov ECX, [par1]",
        "         mov EBX, EBP ",
        "         add EBX, 8",
        "         mov EAX, [par2]",
        "         invoke WriteConsoleA, stdout, EAX, ECX, EBX, 0",
        "         pop ECX ",
        "         pop EBX",
        "         pop EAX",
        "         ; pop EBP",
        "         ret",
        "printc    ENDP",
        "writedec PROC",
        "         push EAX",
        "         push EBX",
        "         push ECX",
        "          push EDX",
        "         mov AH, 0",
        "         mov CL, 10",
        "          mov EBX, 0",
        "         mov EDX, 0",
        "         mov DL, AL",
        "         and DL, 80h",
        "         jnz obrni",
        "         jmp ??write",
        "obrni: neg AL",
        "??write:div CL",
        "         xchg AH, AL",
        "         add AL,'0'",
        "         push EAX",
        "         inc EBX",
        "         xchg AH, AL   ",
        "         cmp AL, 0",
        "         je minusprovera",
        "         mov AH, 0",
        "         jmp ??write",
        "minusprovera:",
        "         cmp DL, 80h",
        "         je dodajminus",
        "         jmp kraj_wrdec",
        "dodajminus:",
        "         push '-'",
```

```
            "         push ESP",
            "         push 1",
            "         call printc",
            "         add ESP, 4",
            "kraj_wrdec:",
            "         cmp EBX, 0",
            "         je kraj_skroz",
            "         push ESP",
            "         push 1",
            "         call printc",
            "         add ESP,4",
            "         dec EBX",
            "         jmp kraj_wrdec",
            "kraj_skroz:",
            "         pop EDX",
            "         pop ECX",
            "         pop EBX",
            "         pop EAX",
            "         ret",
            "writedec ENDP",};

        for (int i = 0; i < s.length; i++) {
            izlaz.write(s[i]);
            izlaz.newLine();
            if (s[i].equals("printc ENDP")) {
                izlaz.newLine();
            }
        }

    }
}
```

ST_entries.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package sp_dz2;

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

    String name;          // name
    int value;         // value once defined
    boolean defined;           // true after defining occurrence
encountered
    int seg;            // segment in which simbol is defined
    boolean globuse, globdef;

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

    public boolean isDefined() {
        return defined;
    }

    public String getName() {
        return name;
    }

    public int getValue() {
        return value;
    }

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

    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;
    }

}
```

TableOperation.java

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

package sp_dz2;

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

/**
 *
 * @author MB
 */
public class TableOperation {//tabela operacionih kodova
    class Polje{
        String mnemonic;
        String valueHex;
        int valueDec;
        boolean needZ, needC, needP, setC, setPZ;

        public Polje(String mnemonic, String valueHex, int valueDec,
boolean needZ, boolean needC, boolean needP, boolean setC, boolean setPZ) {
            this.mnemonic = mnemonic;
            this.valueHex = valueHex;
            this.valueDec = valueDec;
            this.needZ = needZ;
            this.needC = needC;
            this.needP = needP;
            this.setC = setC;
            this.setPZ = setPZ;
        }
    };

    private static List<Polje> listMnem;

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

    private void addItems(){
        //jednobajtne
        listMnem.add(new Polje("CLC", "02h", 2, false, false, false, true,
false));
        listMnem.add(new Polje("CLX", "03h", 3, false, false, false, false,
false));
        listMnem.add(new Polje("CMC", "04h", 4, false, false, false, true,
false));
        listMnem.add(new Polje("INC", "05h", 5, false, false, false, false,
true));
```

```
        listMnem.add(new Polje("DEC", "06h", 6, false, false, false, false,
true));
        listMnem.add(new Polje("INX", "07h", 7, false, false, false, false,
true));
        listMnem.add(new Polje("DEX", "08h", 8, false, false, false, false,
true));
        listMnem.add(new Polje("TAX", "09h", 9, false, false, false, false,
false));
        listMnem.add(new Polje("INI", "0Ah", 10, false, false, false,
false, true));
        listMnem.add(new Polje("OTI", "0Eh", 14, false, false, false,
false, false));
        listMnem.add(new Polje("PSH", "13h", 19, false, false, false,
false, false));
        listMnem.add(new Polje("POP", "14h", 20, false, false, false,
false, true));
        listMnem.add(new Polje("SHL", "15h", 21, false, false, false, true,
true));
        listMnem.add(new Polje("SHR", "16h", 22, false, false, false, true,
true));
        listMnem.add(new Polje("RET", "17h", 23, false, false, false,
false, false));
        listMnem.add(new Polje("HLT", "18h", 24, false, false, false,
false, false));

        //dvobajtne
        listMnem.add(new Polje("LDA", "19h", 25, false, false, false,
false, true));
        listMnem.add(new Polje("LDX", "1Ah", 26, false, false, false,
false, true));
        listMnem.add(new Polje("LDI", "1Bh", 27, false, false, false,
false, true));
        listMnem.add(new Polje("STA", "1Eh", 30, false, false, false,
false, false));
        listMnem.add(new Polje("STX", "1Fh", 31, false, false, false,
false, false));
        listMnem.add(new Polje("ADD", "20h", 32, false, false, false, true,
true));
        listMnem.add(new Polje("ADX", "21h", 33, false, false, false, true,
true));
        listMnem.add(new Polje("SUB", "26h", 38, false, false, false, true,
true));
        listMnem.add(new Polje("SBX", "27h", 39, false, false, false, true,
true));
        listMnem.add(new Polje("CMP", "2Ch", 44, false, false, false, true,
true));
        listMnem.add(new Polje("ANA", "2Fh", 47, false, false, false, true,
true));
        listMnem.add(new Polje("ORA", "32h", 50, false, false, false, true,
true));
        listMnem.add(new Polje("BRN", "35h", 53, false, false, false,
false, false));
```

```
        listMnem.add(new Polje("BZE", "36h", 54, true, false, false, false,
false));
        listMnem.add(new Polje("BNZ", "37h", 55, true, false, false, false,
false));
        listMnem.add(new Polje("BPZ", "38h", 56, false, false, true, false,
false));
        listMnem.add(new Polje("BNG", "39h", 57, false, false, true, false,
false));
        listMnem.add(new Polje("BCC", "3Ah", 58, false, true, false, false,
false));
        listMnem.add(new Polje("BCS", "3Bh", 59, false, true, false, false,
false));
        listMnem.add(new Polje("JSR", "3Ch", 60, false, false, false,
false, false));
    }

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

    public static boolean oneByteInstr(int num){
        return num == 2 || num == 3 || num == 4 || num == 5 || num == 6
                || num == 7 || num == 8 || num == 9 || num == 10 || num ==
14
                || num == 19 || num == 20 || num == 21 || num == 22
                || num == 23 || num == 24;
}

    public static boolean twoBytesInstr(int num){
        return num == 25 || num == 26 || num == 27 || num ==30
                || num == 31 || num == 32 || num == 33 || num == 38
                || num == 39 || num == 44 || num == 47 || num == 50
                || num == 53 || num == 54 || num == 55 || num == 56
                || num == 57 || num == 58 || num == 59 || num == 60;
    }

    public static boolean isJumpInstr(int  num){ //JSR & BRN not included
        return num == 54 || num == 55 || num == 56
                || num == 57 || num == 58 || num == 59;

    }

    public static boolean isInstructionThatSetsPZ(int num){
        return num == 5 || num == 6 || num == 7 || num == 8|| num == 10
                || num == 20 || num == 21 || num == 22 || num == 25
                || num == 26 || num == 27 || num == 32 || num == 33
                || num == 38 || num == 39 || num == 44 || num == 47
                || num == 50;
    }
```

```
    public static boolean isInstructionThatSetsC(int num){
        return num == 2 || num == 4 || num == 21 || num == 22
                || num == 32 || num == 33 || num == 38 || num == 39
                || num == 44 || num == 47 || num == 50;
    }

    public static boolean jumpInstrPZ(int num){
        return num == 54 || num == 55 || num == 56 || num == 57;

    }

    public static boolean jumpInstrC(int num){
        return num == 58 || num == 59;
    }

    //delete this and next
    public static boolean jumpInstrPZ(String s){
        return s.equalsIgnoreCase("jz") || s.equalsIgnoreCase("jnz") ||
                s.equalsIgnoreCase("jg") || s.equalsIgnoreCase("jng");

    }

    public static boolean jumpInstrC(String s){
        return s.equalsIgnoreCase("jnc") || s.equalsIgnoreCase("jc");
    }

    public static boolean needOptimisation(int num){
        return num == 10 || num == 20 || num == 25 || num == 26;
    }

    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_dz2;

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

/**
 *
 * @author MB
 */

public class TableRelocations {
    final static int NUM_OF_NAMES = 10;

    public class Zapis {
        private int loc, seg, ref;
        private String type;
        private String name;
        private String op;
        private int locOrLen;

        public Zapis(String name, int loc, int seg, int ref, String type,
String op) {
            this.name = name;
            this.loc = loc;
            this.seg = seg;
            this.ref = ref;
            this.op = op;
            this.type = type;
            locOrLen = -1;
        }

        public String getType() {
            return type;
        }

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

        public int getLoc() {
            return loc;
        }

        public int getRef() {
            return ref;
        }
```

```
    public int getSeg() {
        return seg;
    }

    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 String getName() {
        return name;
    }

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

    public String getOp() {
        return op;
    }

    public void setOp(String op) {
        this.op = op;
    }

    public int getLocOrLen() {
        return locOrLen;
    }

    public void setLocOrLen(int locOrLen) {
        this.locOrLen = locOrLen;
    }

}
private static List<Zapis> listZapis;

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

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

public Zapis getZapis(int index){
    return listZapis.get(index);
```

```
    }

    public void add(String name, int loc, int seg, int ref, String type,
String op, int locOrLen){
        listZapis.add(new Zapis(name, loc, seg, ref, type, op));
        if(locOrLen != -1) listZapis.get(listZapis.size() -
1).setLocOrLen(locOrLen);
    }

    public boolean isLabeled (int loc){
        for(int i = 0; i<listZapis.size(); i++){
            if(listZapis.get(i).getLoc() == loc &&
!listZapis.get(i).getType().contains("S")){
                if(listZapis.get(i).getSeg() == listZapis.get(i).getRef())
return true;
            }
        }
        return false;
    }

    public String getLabelName(int loc){
        String name = "";
        for(int i = 0; i<listZapis.size(); i++){
            if(listZapis.get(i).getLoc() == loc &&
!listZapis.get(i).getType().contains("S")){
                if(listZapis.get(i).getSeg() == listZapis.get(i).getRef())
name = listZapis.get(i).getName();
            }
        }
        return name;
    }

    public String[] getNamesForLoc(int loc){
        String [] names = new String[NUM_OF_NAMES];
        int pos = 0;
        for(int i = 0; i<listZapis.size(); i++){
            if(listZapis.get(i).getLoc() == loc){
                names[pos++]
                        = listZapis.get(i).getName();
            }
        }
        return names;
    }

    public String[] getOperationsForLoc(int loc){
        String [] op = new String[NUM_OF_NAMES];
        int pos = 0;
        for(int i = 0; i<listZapis.size(); i++){
            if(listZapis.get(i).getLoc() == loc){
                op[pos++] = listZapis.get(i).getOp();
            }
        }
        return op;
```

```
    }

    public String getAdrField(int loc){
        String adr = "";
        String names[] = getNamesForLoc(loc),
                op[] = getOperationsForLoc(loc);
        if (names == null || op == null) {
            return adr;
        }
        if (names.length > 1 && names[1] != null) {
            for (int i = 0; i < names.length && names[i] != null; i++) {
                if(i == 0 && op[i].equalsIgnoreCase("-")) adr+="-";
                adr += names[i] + " ";
                if (i != names.length - 1) {
                    adr += op[i + 1] + " ";
                }
            }
        } else {
            adr = names[0];
        }

        return adr;
    }

    public boolean isJumpOnLabel(String s){
        for(int i = 0; i< listZapis.size(); i++){
            if(listZapis.get(i).getRef() == listZapis.get(i).getSeg()
                    && listZapis.get(i).getName().equalsIgnoreCase(s))
                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() + "\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).getName() + "\n");
    }

}
```

TableSymbols.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package sp_dz2;

import java.util.LinkedList;

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

    private int numGlobUse, numGlobDef, numEqu;

    public TableSymbols() {
        listsym = new LinkedList<ST_entries>();
        numEqu = numGlobDef = numGlobUse = 0;
    }

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

    public int getNumEqu() {
        return numEqu;
    }

    public int getNumGlobDef() {
        return numGlobDef;
    }

    public int getNumGlobUse() {
        return numGlobUse;
    }

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

    // 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);
            if(globuse && !globdef) numGlobUse++;
```

```
            else if(!globuse && globdef) numGlobDef++;
            else if(globdef && globuse) numEqu++;
        }
    }

    // Returns value of required name
    // location is the current value of the instruction location counter
 /*   public int valueOfSymbol(String name, int location) {
        int index = findEntry(name);
        if (listsym.get(index).isDefined()) {
            return listsym.get(index).getValue();
        } else if (listsym.get(index).isGlobuse()){
            return 0;
        } else{
            return -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, true, 0);
        listsym.add(symentryE);
        return index;
    }

    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;
    }

    private LinkedList<ST_entries> listsym;
}
```