Java tečaj

2. domaća zadaća – drugi dio.

V1.1

Zagreb, 2013.

1. Izrada jednostavnog računalnog sustava

U nastavku slijedi drugi dio zadataka koji čine drugu domaću zadaću. Ako još niste, pročitajte najprije općenite upute koje se nalaze u dokumentu s prvim dijelom domaće zadaće. Sve zadatke potrebno je smjestiti u odgovarajuće podpakete paketa `hr.fer.zemris.java.tecaj_2.jcomp`.

1.1 Priprema

Na *google*-grupi dostupne su još i sljedeće datoteke:

```
parser.jar
models.jar
primjeri.zip
```

Preuzmite ove datoteke. Datoteka `parser.jar` sadrži implementaciju parsera za asemblerski kod pisan prema formatu koji je opisan u nastavku. Datoteka `models.jar` sadrži izvorne i izvršne kodove potrebnih sučelja. Napravite novi Eclipse projekt `homework02b`; unjemu napravite direktorij `lib` te u taj direktorij iskopirajte `parser.jar` i `models.jar`. Potom ih uključite u Eclipsov *Build-Path* kako smo to napravili na predavanju s bibliotekom `prikaz.jar`. U projekt dodajte i `build.xml` datoteku kako ste to napravili i u prvoj zadaći. Skriptu ćete, međutim, morati malo modificirati tako da postane svjesna postojanja direktorija `lib`. Najprije na mjestu gdje definirate varijable `src`, `build` i slično dodajte još i definiciju varijable `lib`:

```
<property name="lib" location="lib"/>
```

Potom modificirajte classpath koji se koristi prilikom prevođenja izvornog koda programa iz:

```
<path id="compile.path">
 <pathelement location="${build.classes}"/>
</path>
```

u

```
<path id="compile.path">
 <pathelement location="${build.classes}"/>
 <fileset dir="${lib}">
  <include name="*.jar"/>
 </fileset>
</path>
```

Ovo je potrebno kako bi prevodioc prilikom prevođenja Vašeg koda imao pristup do potrebnih `.class` datoteka koje koristite u programu.

Datoteka `primjeri.zip` sadrži dva primjera na koja ćemo se pozvati na kraju ovog dokumenta – za sada je pričuvajte negdje sa strane.

1.2 Zadatak

Cilj ovog zadatka je upoznati se još malo bolje sa sučeljima i njihovom uporabom. Stoga ćete tijekom izrade ovog zadatka napraviti jednostavnu implementaciju "mikroprocesora" koji može izvršavati jednostavne programe. Primjerice, neki jednostavan program mogao bi izgledati ovako:

```
#Ovaj program 3 puta ispisuje "Hello world!"

            load r0, @brojac        ; ucitaj 3 u registar r0
            load r1, @nula          ; ucitaj 0 u registar r1
            load r7, @poruka        ; ucitaj poruku u r7
@petlja:    testEquals r0, r1       ; je li r0 pao na nulu?
            jumpIfTrue @gotovo      ; ako je, gotovi smo
            decrement r0            ; umanji r0
            echo r7                 ; ispisi na konzolu poruku
            jump @petlja            ; skoci natrag u petlju
@gotovo:    halt                    ; zaustavi procesor

#podaci koje koristimo u programu

@poruka:    DEFSTR "Hello world!\n" ; poruka na jednoj mem. lokaciji
@brojac:    DEFINT 3                ; broj 3 na drugoj mem. lokaciji
@nula:      DEFINT 0                ; broj 0 na trecoj mem. lokaciji
```

U okviru ovog zadatka računalni sustav i njegove komponente modelirane su kroz niz sučelja. Parser za program napisan asemblerskim jezikom prikazanim u prethodnom primjeru priložen je u JAR datoteci i ne trebate ga raditi. Parser, međutim, ne zna koje sve instrukcije postoje i kako ih treba izvršavati; ono što parser razumije je sintaksa asemblerskog jezika: što su komentari, gdje se nalazi naziv instrukcije, gdje su parametri i slično. Primjerice: pogledajte prvi redak koji sadrži naredbu `load`. Analizom tog retka, parser će zaključiti da je potrebno stvoriti naredbu koja se zove `load` te koja ima dva operanda: prvi je registar r0 a drugi je adresa memorijske lokacije na kojoj se nalazi zapisan brojač (potražite ga u kodu i uočite da je njegova početna vrijednost postavljena na 3). Jednom kada pročita redak, prevodioc će pokušati pronaći razred koji implementira sučelje `Instruction` i koji predstavlja implementaciju ove instrukcije, kako bi dobio primjerak tog razreda i instrukciju pohranio u memoriju mikroračunala. Vaš će zadatak, između ostaloga, biti i pisanje implementacija svih potrebnih instrukcija.

O računalu

Naše se računalo sastoji od uobičajenih dijelova: registara i memorije. Od registara na raspolaganju su nam registri opće namjene, programsko brojilo te jedna zastavica. Za razliku od konvencionalnih procesora, naši registri opće namjene (kao i sama memorija) umjesto jednostavnih brojeva mogu pamtiti proizvoljne Java objekte.

Zadatak 1.2.1.

U paketu `hr.fer.zemris.java.tecaj_2.jcomp` nalaze se sučelja `Computer`, `Memory` te `Registers`. Proučite ova sučelja (svako sučelje dodatno je dokumentirano) i za svako napišite jedan razred koji ih implementira. Ovi razredi dostupni su u biblioteci `models.jar` i ako ste podesili *Build-Path*, morali biste ih vidjeti u *package exploreru*. Razrede nazovite kao i samo sučelje i nadodajte `Impl` na kraju (tako će razred `ComputerImpl` implementirati sučelje `Computer`). Ove implementacije

smjestite u paket `hr.fer.zemris.java.tecaj_2.jcomp.impl`. Za pohranu registara opće namjene te za memoriju koristite obično polje. Implementacija memorije trebala bi imati konstruktor koji prima broj memorijskih lokacija (tj. veličinu memorije):

```
public MemoryImpl(int size) { ... }
```

a implementacija registara trebala bi imati konstruktor koji prima broj registara koji će procesoru biti na raspolaganju:

```
public RegistersImpl(int regsLen) { ... }
```

Jednom kada su definirani razredi potrebni za rad našeg računala, možemo pogledati kako se to računalo može koristiti:

```
// Stvori računalo s 256 memorijskih lokacija i 16 registara
Computer comp = new ComputerImpl(256, 16);

// Stvori objekt koji zna stvarati primjerke instrukcija
InstructionCreator creator = new InstructionCreatorImpl(
      "hr.fer.zemris.java.tecaj_2.jcomp.impl.instructions"
);

// Napuni memoriju računala programom iz datoteke; instrukcije stvaraj
// uporabom predanog objekta za stvaranje instrukcija
ProgramParser.parse(
   "examples/asmProgram1.txt",
   comp,
   creator
);

// Stvori izvršnu jedinicu
ExecutionUnit exec = new ExecutionUnitImpl();

// Izvedi program
exec.go(comp);
```

Prethodni primjer sastoji se od nekoliko cjelina:

1. stvaranje novog računala,
2. stvaranje objekta pomoću kojeg će parser stvarati primjerke instrukcija,
3. prevođenje programa zapisanog u tekstualnoj datoteci i njegovo punjenje u memoriju računala,
4. stvaranje izvršne jedinice (tj. sklopa koji će izvoditi program) te
5. pokretanja izvršne jedinice.

Ako ste riješili zadatak 1.2.1, prva cjelina će raditi. U drugoj liniji poziva se parser iz JAR datoteke čiji je zadatak obraditi datoteku s programom. Kako sam parser ne zna na koji način stvoriti odgovarajuću instrukciju, potrebna mu je pomoć. Tu u igru ulazi razred `InstructionCreatorImpl` čija Vam je implementacija također unaprijed dana i dostupna je u istom paketu u kojem se nalazi i sam parser. Zadaća ovog razreda je stvoriti primjerak razred koji implementira instrukciju koju je parser pronašao u kodu. Parser s ovim razredom može razgovarati jer sam razred implementira sučelje `InstructionCreator` (pogledajte izvorni kod tog sučelja). Svaki puta kada parser otkrije neku instrukciju, poziva metodu:

```
public Instruction getInstruction(
  String name, List<InstructionArgument> arguments
```

```
);
```

Prvi argument je naziv pronađene instrukcije a kao drugi argument se predaje lista argumenata instrukcije koje je parser pronašao u datoteci. Npr. kod obrade retka u kojem piše:

```
mul r0, r1, r2
```

ime će biti "`mul`", a lista će sadržavati tri objekta (za svaki registar po jedan; za detalje pogledati izvorni kod sučelja `InstructionArgument`).

Kako bi Vam se olakšao rad, uporabom ponuđenog razreda `InstructionCreatorImpl` dovoljno je razred koji implementira svaku instrukciju smjestiti u paket koji se predaje u konstruktoru od korištenog razreda `InstructionCreatorImpl`. Pri tome sama implementacija instrukcije mora imati jedan javni konstruktor koji prima listu argumenata (drugi argument metode `getInstruction`). Primjerice, instrukcija `mul r0, r1, r2` koja uzima sadržaje registara `r1` i `r2`, množi ih i rezultat pohranjuje u `r0` može se implementirati ovako:

```
package hr.fer.zemris.java.tecaj_2.jcomp.impl.instructions;

import java.util.List;
import hr.fer.zemris.java.tecaj_3.dz2.Computer;
import hr.fer.zemris.java.tecaj_3.dz2.Instruction;
import hr.fer.zemris.java.tecaj_3.dz2.InstructionArgument;

public class InstrMul implements Instruction {
	private int indexRegistra1;
	private int indexRegistra2;
	private int indexRegistra3;

	public InstrMul(List<InstructionArgument> arguments) {
		if(arguments.size()!=3) {
			throw new IllegalArgumentException("Expected 3 arguments!");
		}
		if(!arguments.get(0).isRegister()) {
			throw new IllegalArgumentException("Type mismatch for argument 0!");
		}
		if(!arguments.get(1).isRegister()) {
			throw new IllegalArgumentException("Type mismatch for argument 1!");
		}
		if(!arguments.get(2).isRegister()) {
			throw new IllegalArgumentException("Type mismatch for argument 2!");
		}
		this.indexRegistra1 = ((Integer)arguments.get(0).getValue()).intValue();
		this.indexRegistra2 = ((Integer)arguments.get(1).getValue()).intValue();
		this.indexRegistra3 = ((Integer)arguments.get(2).getValue()).intValue();
	}

	public boolean execute(Computer computer) {
		Object value1 = computer.getRegisters().getRegisterValue(indexRegistra2);
		Object value2 = computer.getRegisters().getRegisterValue(indexRegistra3);
		computer.getRegisters().setRegisterValue(
				indexRegistra1,
				Integer.valueOf(
					((Integer)value1).intValue() * ((Integer)value2).intValue()
				)
		);
		return false;
	}
}
```

Zadatak 1.2.2.

Napišite implementacije instrukcija:

`load rX, memorijskaAdresa`

koja uzima sadržaj memorijske lokacije (dobit će to kao broj u drugom argumentu) i pohranjuje taj sadržaj u registar `rX` (index će dobiti kao broj u prvom argumentu),

`echo rX`

koja uzima sadržaj registra `rX` i ispisuje ga na ekran (pozivom metode `System.out.print()`), te

`halt`

koja zaustavlja rad procesora.

Zadatak 1.2.3

Napišite razred koji implementira sučelje `ExecutionUnit`. Za pseudo-kod pogledajte u izvorni kod tog sučelja.

Nakon što ovo napravite, stvorite razred `Test`, u njega iskopirajte prethodno dani primjer stvaranja računala, prevođenja i pokretanja programa te stvorite datoteku `prim1.txt` sljedećeg sadržaja:

```
          load r7, @poruka    ; ucitaj poruku u r7
          echo r7             ; ispisi na konzolu poruku
          halt                ; zaustavi procesor

@poruka:  DEFSTR "Hello world!\n"
```

Prilikom prevođenja ovog programa, parser će u memoriju na lokaciju 0 pohraniti instrukciju `load` (odnosno primjerak vašeg razreda `InstrLoad`), na lokaciju 1 instrukciju `echo`, na lokaciju 2 instrukciju `halt` te na lokaciju 3 string `"Hello world!\n"` (za ovo posljednje je zaslužna direktiva `DEFSTR` – *define string*; integeri se u memoriju pohranjuju s `DEFINT`). Direktive `DEFSTR` i `DEFINT` ne pišete Vi – to parser sam zna obaviti. Također, kod instrukcija koje primaju memorijsku lokaciju, u argumentima Vašeg konstruktora parser Vam neće dati ime te lokacije (tipa `@poruka`) već ćete dobiti stvarnu lokaciju (broj).

Sada biste trebali moći pokrenuti program `Test`, i na ekranu dobiti ispis `"Hello world!"`. Ako nešto ne radi, sada je pravo vrijeme za otkriti u čemu je problem.

Konačni cilj

Cilj je napraviti procesor koji će moći "izvrtiti" kodove priložene u primjerima u ZIP datoteci; primjere (odnosno čitav direktorij examples iskopirajte u Eclipseov projekt, tako da u direktoriju projekta uz src i bin imate i examples). Prvi primjer nekoliko puta ispisuje istu poruku a drugi generira dobro poznati slijed brojeva. Da bi to ostvarili, još Vam trebaju neke instrukcije koje morate ostvariti.

Zadatak 1.2.4

Instrukcija	*Opis*
`add rx, ry, rz`	rx ← ry + rz
`decrement rx`	rx ← rx - 1
`increment rx`	rx ← rx + 1
`jump lokacija`	pc ← lokacija
`jumpIfTrue lokacija`	ako je flag=1 tada pc ← lokacija
`move rx, ry`	rx ← ry
`testEquals rx, ry`	postavlja zastavicu flag na true ako su sadržaji registara rx i ry isti, odnosno na false ako nisu.

Ostvarite ove instrukcije i provjerite rad programa oba priložena programa.