SVE LEKCIJE (ELIMINACIJE, ISPIT I DODATNO)

Uvod

1. Šta je sistemska programska podrška?
Sistemska programska podrška je sistemski softver. Ona podrazumeva pomoćne programe, tj. podsisteme, kao što su: asembler, kompajler, punjač...

2. Navesti 4 lingvistička nivoa programske podrške.
- Upravljanje resursima;
- Direktna komunikacija sa računarskim sistemom;
- Indirektna komunikacija sa računarskim sistemom;
- Skup aplikacionih programa.

3. Navesti 3 tehnike za definisanje novih lingvističkih nivoa programske podrške.
- Proširenje;
- Prevođenje;
- Interpretacija.

4. Navesti i i ukratko objasniti 3 tehnike za definisanje novih lingvističkih nivoa.
Tehnike za definisanje novih lingvističkih nivoa su: proširenje, prevođenje i interpretacija. Kod proširanje, nove procedure koriste primitive osnovnog sistema, tj. uvodi se novi lingvistički nivo, kao npr. biblioteka. Prevođenje, tj. kompajliranje, predstavlja prevođenje sa novog jezika na jezik osnovnog sistema, tj. prevodi se sa jednog lingvističkog nivoa na drugi. Interpretacija podrazumeva da su faze prevođenja i izvršenja vremenski zavisne. Interpretacija funkcioniše kao prevođenje, samo uvodi i izvršavanje.

5. Tehnike definisanja novih lingvističkih nivoa su (zaukružiti tacno):
a) Proširenje
b) Prevođenje
v) Kombinacija
g) Dodavanje

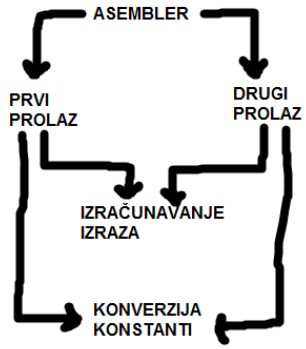
Projektovanje asemblera

6. Prvi prolaz asemblera na svom izlazu daje (zaukružiti tacno)?
a) Tabelu identifikatora
b) Tabelu literala
v) Tabelu kodova operacija
g) Kopiju polaznog programa

7. Moduli asemblera su:
- Modul za prvi prolaz;

- Modul za drugi prolaz;
- Modul za izračunavanje izraza;
- Modul za konverziju konstanti.

8. Kako su četiri osnovna modula asemblera povezana?



9. Šta je izlaz iz prvog prolaza asemblera, a šta iz drugog?
Izlaz iz prvog prolaza asemblera je: tabela simbola, tabela literala, kopija polaznog programa i ostale informacije, a iz drugog: izveštaj o prevođenju i datoteka prevedenog programa.

10. Zbog čega se modul za izračunavanje izraza poziva u prvom, odnosno drugom, prolazu asemblera?
U prvom za obradu pseudo operacije EQU, a u drugom za određivanje vrednosti operanada.

11. Zbog čega se modul za konverziju konstanti poziva u prvom, odnosno drugom, prolazu asemblera?
U prvom za obradu literala, a u drugom za obradu α: DATA β i za obradu literala.

12. Dat je asemblerski kod: **PROVERITI**

```
niz: .word 9 123 234 53 1
Begin:
        subi $t2, $t2, 5 * 7
.space 24
        addi $t1, $0, niz + 3
        b Begin - 3
```

Koliko u njemu ima simbola? 2 (niz, Begin)
Koliko ima literala? 6 (9, 123, 234, 53, 1, 5 * 7)
Koliko puta će se pozvati modul za izračunavanje izraza u drugom prolazu asemblera? 3 (5 * 7, niz + 3, Begin - 3)

13. Na osnovu priloženog asemblerskog koda popuniti tabelu simbola. Širina memorije za dati primer je 8 bita, veličina podataka 24 bita (.word = 24 bita), a veličina svake instrukcije 16 bita.

```
.data
tmp:    .word 100       0
a:      .word 70        3
val:    .word 4         6
```

```
.text
main:
        la      $t0, tmp        0
        lw      $t1, 0($t0)     2
        li      $t0, 5          4
        li      $t2, -5         6
lab:
        add     $t1, $t1, $t0   8
        addi    $t2, $t2, 1     10
        blez    $t2, lab        12
        nop                     14
        la      $t0, a          16
        sw      $t1, 0($t0)     18
exit:
        nop                     20
```

Simbol	Vrednost	Sekcija (data / text)
tmp	0	data
a	3	data
val	6	data
main	0	text
lab	8	text
exit	20	text

Projektovanje makroasemblera

14. Dva potrebna proširenja asemblera da bi se napravio makro-asembler su:
- Da prepozna maktoinsturkciju;
- Da proširi makroinstrukciju.

Da bi to uradio, on najpre treba da pronađe i sačuva makrodefiniciju koja odgovara makronstrukciji. Makrodefinicija počinje sa MACRO, a završava sa ENDM pseudo insturkcijom.

15. Kako se vrši prepoznavanje makroinstrukcija?
Najprostiji način je dodati vrstu u tabeli koda operacije. Ova vrsta sadrži naziv makroinstrukcije i ukazivač na odgovarajuću makro definiciju. Definicije makroa se čuvaju u tabeli makro definicija.

Formalne gramatike

16. Šta je rečenica, a šta rečenicna forma?
Rečenična forma je bilo koja reč koja može da se razvije od polaznog simbola, a rečenica je rečenicna forma koja sadrži samo terminalne simbole.

17. Azbuka programskog jezika (T) je:
Konačan skup terminalnih simbola (npr. T={a,b,c}).

18. Programski jezik (L) je:
Podskup rečnika, definisan pomoću jednog ili više znakova smene, gde smena zadatu reč zamenjuje jednom od reči u koju se on transformiše.

19. Šta je rečnik programskog jezika?
Skup svih konačnih reči azbuke je rečnik.

20. Teza formalnog sistema je (zaokružite tačan odgovor):
a) Skup svih konačnih reči azbuke
b) Podskup rečnika definisan pomoću jednog ili više znakova smene
v) Skup posledica izvedenih iz sopstvenih aksioma

21. Kada je formalni sistem odlučiv?
Formalni sistem je odlučiv ukoliko postoji algoritam koji omogućava određivanje da li je neka reč jezika element teze sistema.

22. Formalni sistem je odlučiv ako je moguće odrediti da li je data reč element (zaokružiti tačan odgovor):
a) rečnika
b) teze sistema
v) svih posledica sistema

23. Četiri elementa i jedno ograničenje formalne gramatike su?
1) N je skup neterminalnih simbola;
2) T je skup terminalnih simbola;
3) Σ je početni simbol, i $\Sigma \in N$;
4) P je skup smena $\alpha \rightarrow \beta$, gde su $\alpha,\beta \in (NUT)^*$, i α nije nula;
5) Ograničenje je da su N i T disjunktni, tj. $N \cap T$ je prazan skup.

24. Šta je Bakus-Naurova forma (BNF)?
Bakus-Naurova forma je notacija za izražavanje produkcije. Svaka produkcija definiše sintaksnu klasu. Elementi ovog jezika su:
::= predstavlja strelicu smene;
< > ograđuje ime sintaksne klase;
| simbol “ILI”, omogućava dodelu više smena.

25. Data je gramatika jezika J. Ispod svake rečenice napisati da li je deo J i ako jeste, redosled primene produkcije koji generiše tu reč.

1. S -> y += P	3. P -> x - 2	5. F -> z
2. S -> F += x - P	4. P -> F - 2	6. F -> x - 2

a) y += x - 2 - 2 (1 -> 4 -> 6)

b) z += x - x - 2 - 2 (2 -> 5 -> 4 -> 6)
v) z += x - P (Nije deo jezika - P se ne može preslikati u P)
Da li je gramatika analitička ili generativna? Generativna

26. Date su rečenice tipa: x = 10; y = x; x = 10 + 5 + y;
Dato je da postoje terminalni simboli id, num, +, -, *, /, ;
Napisati gramatiku koja pokriva date rečenice?
S -> x = E; E -> num F -> num + num + y
S -> y = E; E -> x
E -> F

27. Data je sledeća gramatika, navesti terminalne i neterminalne simbole.
S -> get E from S E -> num EQ num L -> quit
S -> begin S L L -> ; S L
S -> print E END
Terminalni: get from begin print END num EQ quit ;
Neterminalni : S E L

Međujezik

Interpretiranje međujezika

Generisanje leksičkih i sintaksnih analizatora

28. Napisati regularan izraz koji definiše realne brojeve u jeziku C (53.7, .28, 4., -.28). Ne pokrivati oblik sa eksponentom.
("-"?[0-9]*"."[0-9]+)|("-"?[0-9]+"."[0-9]*)

Pravljenje sintaksnog analizatora

Školski kompajler

29. Koje su dve vrste prelaznih oblika polaznog programa?
Vrste prelaznih oblika polaznog programa, tj. međureprezentacije, su: stablo sintaksne analize i matrični prelazni oblik. Kod školskog kompajlera, prelazni oblik je stablo sintaksne analize.

30. Napisati matrični prelazni oblik za dati izraz (pre optimizacije i posle optimizacije):
A = 2 * 276 / 92 * B

PRE OPTIMIZACIJE			
M1	*	2	276
M2	/	M1	92
M3	*	M2	B
M4	=	A	M3

POSLE OPTIMIZACIJE			
M1			
M2			
M3	*	6	B
M4	=	A	M3

31. Tri primera mašinski nezavisnih optimizacija kod školskog kompajlera su:
- Eliminacija zajedničkih podizraza;
- Prethodno izračunavanje vrednosti izraza;
- Uočavanje invarjantnog računanja unutar ciklusa.

32. Koje su faze školskog kompajlera?
- Leksička analiza;
- Sintaksna analiza;
- Generisanje (izbor) instrukcija;
- Pridruživanje memorije;
- Asembliranje.

Leksička i sintaksna analiza spadaju u prednji deo kompajlera, a generisanje instrukcija i pridruživanje memorije u zadnji. Asembliranje nije sastavni deo kompajlera.

33. Koja su tri osnovna zadatka leksičke analize?
- Podela polaznog programa, tj. niza znakova, na niz osnovnih simbola jezika, tj. tokena;
- Formiranje tabele literala i tabele idetifikatora;
- Formiranje tabele uniformnih simbola.

34. Šta radi sintaksna analiza?
Sintaksna analiza prepoznaje rečenice, tj. sintaksne konstrukcije, i interpretira značenje tih konstrukcija. Ona ukazuje i na uočene greške i omogućava kompajleru da nastavi pretraživanje drugih grešaka.

35. Koja su četiri zadatka faze dodele memorije?
- Dodela memorije svim promenljivama;
- Dodela memorije svim privremenim lokacijama koje su potrebne za međurezultate;
- Dodela memorije literalima;
- Obezbeđivanje inicijalziacije odgovarajućih lokacija.

Optimizujući kompajler

36. Šta su osnovni zadaci statičke provere semantike?
Statička provera semantike podrazumeva proveru zadovoljenosti pravila kao što su: provera tipova svih operanada u izrazima (svi operandi moraju biti istog tipa), provera broja parametara kod poziva funkcije (broj parametara pri pozivu funkcije mora biti jednak broju parametara u definiciji funkcije), i sl.

37. Zaokružiti relaciju koja se dobija kao rezultat analize pokazivača za iskaz p=q, gde su p i q pokazivači.
a) (p, 0)-(q, 0)
b) (p, 1)-(q, 1)

38. Prilikom analize pokazivača u optimizujućem koompajleru, koje se relacije dobijaju za sledece iskaze:
a) p = q: (p, 0)-(q, 0)
b) p = &q: (p, 0)-(q, -1)
v) *p = *(q + 4 * i); (p, 1)-(q, 1)

39. Napisati uslove za veoma dobro definisanu petlju sa stanovišta otkrivanja petlji koje mogu biti fizički podržane (hardverske petlje).
- Poznat je naziv i početna vrednost indeksne promenljive;
- Poznat je korak iteracije, tj. vrednost promene indeksa pre naredne iteracije;
- Poznat je broj iteracija.

40. Navesti dve glavne grupe klasa apstraktne sintakse.
Prva grupa klasa apstraktne sintakse predstavlja iskaze, a druga predstavlja izraze.

41. Šta je ulaz u fazu izbora instrukcija kod optimizujućeg kompajlera?
Ulaz u fazu izbora instrukcija je stablo međukoda.

42. Šta je izlaz iz faze analize toka upravljanja, a šta iz faze analize toka podataka kod optimizujućeg kompajlera?
Izlaz iz faze analize toka upravljanja je graf toka, a iz faze analize toka podataka graf smetnji.

43. Šta je izlaz iz faze sintaksne analize kod optimizujućeg kompajlera?
Izlaz iz faze sintaksne analize je apstraktna sintaksa.

44. Koje su faze u zadnjem delu visoko optimizujućeg kompajlera?
- Izbor instrukcija;
- Mašinski zavisna optimizacija;
- Analiza toka upravljanja;
- Analiza toka podataka;
- Dodela resursa;
- Prilagođavanje koda protočnoj strukturi;

- Generisanje koda;
- Povezivanje.

45. Zarad povećanja brzine pristupa, tabela simbola u optimizujućem kompajleru se realizuje kao:
Hash (heš) tabela.

Međukod

46. Tri osobine dobrog međukoda su:
- Pogodan je za davanje značenja čvorovima stabla sintaksne analize;
- Pogodan je za prevođenje u mašinski jezik bilo kog odredišnog procesora;
- Čvorovi stabla međukoda treba da imaju jasna i jednostavna značenja, da bi se optimizacione transformacije međukoda mogle lako specificirati i realizovati, tj. da bi se njime lako baratalo.

47. Čvorovi međukoda treba da imaju jasna i jednostavna značenja da bi se (zaokružiti tačan odgovor):
a) Lako specificirao i realizovao sintaksni analizator
b) Lako specificirala i realizovala kompozicija međukoda
v) Lako specificirale i realizovale optimizacione transformacije

48. Šta je centralna (glavna) osobina međukoda?
Centralna osobina međukoda je njegova nezavisnost o izvornog koda, tj. programskog jezika i od ciljnog procesora, tj. platforme.

49. Navesti vrste čvorova izraza i iskaza u međukodu.
Čvorovi izraza u međukodu su: CONST, NAME, TEMP, BINOP, MEM, CALL i ESEQ.
Čvorovi iskaza u međukodu su: MOVE, EXP, JUMP, CJUMP, SEQ, LABEL.

Kanonizacija međukoda i osnovni blokovi

50. Koje su (dve) osobine stabla kanonizacije?
- Ne sadrži SEQ (niz od dva iskaza) i ESEQ (niz od jednog iskaza i jednog izraza) čvorove;
- Predak CALL (poziv funkcije) čvora je EXP (izraz) ili MOVE (prebaci rezultat izraza u zadatu lokaciju) čvor.

51. Šta predstavlja čvor međukoda ESEQ (zaokružiti tačan odgovor)?
a) Niz iskaza
b) Niz iskaza i izraza
v) Niz izraza

52. Šta je trag?
Trag je niz iskaza koji se mogu uzastopno izvršiti tokom izvršavanja programa. On može sadržati i uslovne skokove.

53. Po definiciji, koje tri osobine zadovoljava osnovni programski blok?
- Započinje labelom LABEL;
- Završava se iskazom skoka JUMP ili CJUMP;
- Unutar bloka ne postoji LABEL, JUMP ili CJUMP.

54. Sve instrukcije osnovnog bloka, osim prve i poslednje, predstavljaju njegovu unutrašnjost Zaokružiti slova ispred instrukcija međureprezentacije koje mogu biti u unutrašnjosti osnovnog bloka:
a) Pristup memoriji
b) Labela
c) Poziv potprograma
d) Uslovni skok
e) Bezuslovni skok
f) Move instrukcija

Izbor instrukcija 1

Izbor instrukcija 2

55. Trošak posmatranog čvora stabla međukoda (prilikom izbora instrukcija primenom dinamičkog programiranja) definisan je kao:
Zbir troškova pojedinačnih instrukcija iz najboljeg niza instrukcija, koje popločavaju podstablo čiji koren je posmatrani čvor.

56. Koja su dva tipa troška koji se definiše prilikom izbora instrukcija?
Idealizovano posmatrano, tip troška može biti: broj instrukcija, ako se kod optimizuje po veličini, ili potrebno vreme procesora, ako se kod optimizuje po brzini.

57. Pisanje modula za izbor instrukcija je složen zadatak koji uključuje dva podzadatka. Koja?
- Modeliranje odredišnog procesora;
- Realizacija algoritma dinamičkog programiranja.

58. Emisija instrukcija u fazi izbora instrukcija kod optimizujućeg kompajlera se obavlja tako što se za svaki čvor međukoda (zaokružiti tačan odgovor):
a) Emituju instrukcije za taj čvor, a onda instrukcije za listove čvora
b) Emitiju instrukcije za listove čvora, a onda istrukcije za taj čvor

---

Analiza životnog veka

59. Kada je promenljiva živa na strelici grafa toka upravljanja?
Promenljiva je živa na strelici grafa ako postoji usmerena putanja od te strelice do 'use' čvora, a da putanja ne prelazi preko ijednog 'def' čvora.

60. Napisati jednačine životnog veka promenljivih za čvor n.
in[n] = use[n] U (out[n] – def[n])
out[n] = Usucc[n]in[s]

61. Kako glasi pravilo za dodavanje smetnje u graf smetnji za čvorove koji nisu MOVE?
Za svaku def promenljivu a, u čvoru koji nije MOVE, dodaj smetnje između a i svake promenljive žive na izlazu čvora.

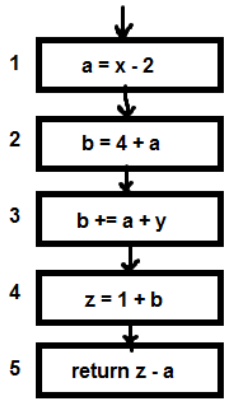
62. Kako glase dva pravila za dodavanje nove smetnje u graf smetnji?
- Za svaku def promenljivu a, u čvoru koji nije MOVE, dodaj smetnje između a i svake promenljive žive na izlazu čvora;
- Za MOVE instrukciju a = c dodaj smetnje izmedju promenljive a i svake promenljive zive na izlazu čvora koja nije c.

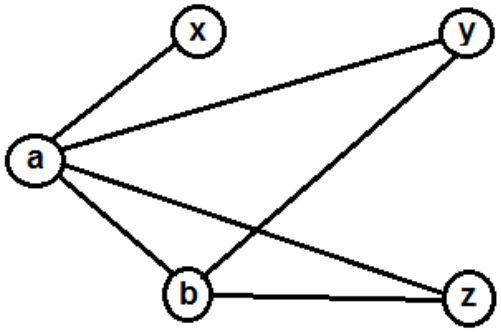
63. Pravilo za definisanje smetnje u grafu smetnji za cvor koji nije MOVE glasi (zaokružiti tačan odgovor)?
a) Za svaku definiciju promenljive A, dodaj smetnju između promenljive A i svake promenljive na ulazu čvora
b) Za svaku definiciju promenljive A, dodaj smetnju između promenljive A i svake promenljive na izlazu čvora

64. Na priloženom grafu svaki čvor predstavlja jednu promenljivu iz datog C koda. Povezati čvorove između kojih u C kodu postoji smetnja usled preklapanja opsega životnog veka. Prvo nacrtati graf toka upravljanja.

```
int foo(int x, int y) {
        int b, z, a = x - 2;
        b = 4 + a;
        b += a + y;
        z = 1 + b;
        return z - a;
```

}





Čvor	pred	succ	use	def	out	in
5	4		a, z			a, z
4	3	5	b	z	a, z	a, b
3	2	4	a, b, y	b	a, b	a, b, y
2	1	3	a	b	a, b, y	a, y
1		2	x	a	a, y	x, y

65. Dato je sledeđe parče koda. Gde počinje, a gde se završava životni vek promenljive a?

```
1. int foo(int x, int y) {
2.     int b, z, a = x - 2;
3.     b = 4 + a;
4.     b += a * y;
5.     z = 1 + b;
6.     return z - a;
7. }
```

Počinje u 2. a završava se u 6. liniji.

Implemetacioni detalji analize životnog veka

Dodela resursa. Bojanje grafova

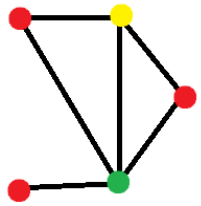
66. Navesti osnovne primitive algoritma za dodelu resursa.

- Formiraj;
- Uprosti;
- Prelij;
- Izaberi;
- Ponovi.

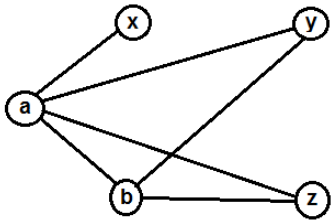
67. Po sigurnoj strategiji Brigsa, čvorovi grafa smetnj a i b se mogu spojiti ukoliko:
Čvorovi a i b se mogu spojiti ukoliko rezultantni čvor ab ima manje od k zajedničkih čvorova suseda, tj. čvorova čiji rang je veći od ili jednak K).

68. Po sigurnoj strategiji Džodža, čvorovi grafa smetnji a i b se mogu spojiti ukoliko:
Čvorovi a i b se mogu spojiti ukoliko svaki sused čvora a ima smetnju sa b ili nema značajan rang.

69. Obojiti dati graf sa 3 boje.



70. Dat je graf smetnji. Treba odrediti minimalan broj resursa, i redosled uklanjanja čvorova na stek prilikom uprošćavanja grafa.



Stek prilikom uprošćavanja grafa: z x a b y (z je poslednji ubačen čvor).
Minimalan broj resursa je 3, jer se graf može obojiti sa najmanje 3 boje (npr: a b x y z).

Raspoređivanje instrukcija

71. Kako su definisane tačka izvršavanja i tačka pristupa u fazi pripreme koda za protočnu obradu?
Tačka izvršenja (ep) je jednaka indeksu instrukcije u nizu instrukcija, a tačka pristupa operandu (ap) se dobija tako što se na ep doda redni broj faze protočne obrade u kojoj se pristupa operandu.

72. U fazi pripreme koda za protočnu obradu, tacka pristupa je definisana kao (zaokružiti tačan odgovor):
a) Indeks instrukcije
b) Indeks instrukcije + broj faze protočne obrade
c) Indeks lokacije podatka kom instrukcija pristupa

73. Koje su dve grupe pristupa kod aspekta optimizacije?
-Slagajući raspoređivač;
- Perkolacioni raspoređivač.

Kompaktor

74. Nabrojati tehnike kompaktovanja koda nižeg reda.
- Zameni;
- Izbaci;
- Zameni-prethodni;
- Zameni-naredni;
- Preimenuj;
- Mutiraj.

75. Nabrojati tehnike kompaktovanja koda višeg reda.
- Traži-unapred;
- Traži-unazad;
- Komapktuj-sa-prethodnim;
- Kompaktuj-sa-narednim.

76. Ukratko opisati tehniku kompaktovanja TRAŽI-UNAPRED.
TRAŽI-UNAPRED je tehnika višeg nivoa kompaktovanja koda koja koristi primitive ZAMENI i IZBACI. Ova tehnika polazi od početka programskog segmenta i u smeru njegovog izvršenja, tj. unapred, traži NOP instrukciju. Kad je pronađe, pokušava da zameni redosled instrukcije koja sledi NOP i pronađenog NOP. Ako zamena uspe, ova tehnika pokušava da izbaci NOP na toj novoj poziciji. Bez obzira na rezultat tog pokušaja, isti postupak se nastavlja od prve instrukcije nakon NOP, pre njegovog pomeranja, do kraja segmenta. Rezultat ovog postupka je da se NOP pomeraju prema kraju segmenta, gde se grupišu i uklanjaju iz programa, kad se prikupi dovoljna količina

---

Punjač

77. Koje su četiri osnovne funkcije punjača?
- Dodela memorijskog prostora programima (alokacija);
- Određivanje vrednosti simboličkih referenci između relokatibilnih programa (povezivanje);
- Podešavanje svih adresno osetljivih lokacija na odgovarajući dodeljeni prostor (relokacija);
- Fizički prenos mašinskih instrukcija i podataka u memoriji (punjenje).

78. Kako punjač relocira program?
Punjač relocira program dodavanjem konstante na svaku relokatibilnu adresu u programu.

79. Koji su nedostaci apsolutnog punjača?
- Negira suštinu asemblera;
- Nemoguće je rešiti problem biblioteka ili programa koji se sastoje od više nezavisnih modula.

80. Koje su prednosti apsolutnog punjača?

- Asembler formira u celini prevedeni program, što znači da su adrese prvih instrukcija u programu i proceduri definisane u vreme asembliranja;
- Jednostavno unosi sa spoljne memorije program u definisane memorijske lokacije.

81. Koje su dobre, a koje loše osobine BSS punjača?

Dobre osobine BSS punjača su:

- Omogućava punjenje više segmenata programa nad samo jednim zajedničkim segmentom podataka;
- Izlaz iz asemblera je prevedeni program i informacija o svim drugim programima na koje se on poziva;
- Navodi se i informacija o tome koje reference treba menjati, tj. relokaciona informacija.

Loše osobine BSS punjača su:

- Vektor prelaza zauzima prostor u memoriji, a koristi se samo za povezivanje;
- Obrađuju segmente procedure, ali ne olakšavaju pristup segmentima deljivih podataka.

82. Koje su faze i funkcije relokatibilnog punjača sa direktnim povezivanjem?

- Glavna uloga punjača u prvoj fazi je dodela i pridruživanje memorije svakom segmentu programa i biblioteke, i formiranje tabele simbola u koju se unose globalni simboli i apsolutne adrese;
- Glavna funkcija druge faze je punjenje stvarnog programa i obavljanje relokacione modifikacije neke adresne komponente koja traži tu modifikaciju.

83. Zaokružiti instrukcije koje predstavljaju adresno osetljive lokacije.

```
main:
        la      $t0, 7
lab1:
        lw      $t1, 0($t0)
        li      $t0, 5
        b       lab2 + 1
        li      $t2, -5
        blez    lab1
lab2:
        nop
```
