Konstrukcija asemblera

Zadatak 1

- Posmatra se sistem sa dvoprolaznim asemblerom, povezivačem i puniocem. Napisati deo koda (x86) asemblerskog programa u kome se:
  - koristi simbol (labela) a, čija je vrednost (na mestu korišćenja) poznata u prvom prolazu asemblera.
  - koristi simbol (labela) b čija je vrednost (na mestu korišćenja) poznata tek u drugom prolazu asemblera.
  - koristi simbol (labela) c1 čija je vrednost (na mestu korišćenja) poznata tek posle povezivanja.
  - koristi simbol (labela) d čija je vrednost (na mestu korišćenja) poznata tek posle punjenja.

Rešenje zadatka 1

```
.equ      a, 10          # (definicija a) lokalni apsolutni simbol
    mov eax, a           # konacna vr. a je poznata u i prolazu
    add eax, b           # (korišcenje b) vrednost b poznata u ii prolazu
.equ b, 20               # (definicija b)
    add eax, c1          # tek ce linker znati vrednost c1
    mov eax, d           # d je relokativan jer je vezan na pc,
d:  .long 10             # pa ce tek punilac znati tacnu vrednost
.end
```

- drugi modul:

```
.global   c1             # c1 je "ulazna tacka"
.equ      c1, 10         # c1 je apsolutan
.end
```

Zadatak 2

- Opisati korišćene strukture podataka asemblera i navesti delove algoritma dvoprolaznog asemblera za obradu direktive:
  - .byte
  - .section .text
  - .global

- Napomene:
  - Rešavati posebno svaku direktivu i razdvojiti prvi i drugi prolaz.
  - Ne opisivati bilo kakve strukture niti delove algoritma koji nisu neposredno vezani za posmatranu direktivu.

.byte

- Korišćene tabele:
  - **Tabela simbola** sadrži korisnički definisane identifikatore (labele) i njihove vrednosti (koje obično predstavljaju neke memorijske adrese).

- Opciono se koristi i:
  - **Tabela direktiva** sadrži parove mnemonika asemblerskih direktiva i odgovarajućeg asemblerskog potprograma za obradu te direktive.

.byte

- Prvi prolaz asemblera:

```
Initialize tables, and set Assembling:=TRUE; Location:=0;
WHILE Assembling DO
        Read line of source and unpack into constituent fields
          Label, Mnemonic, AddressField (* a Name or Number *)
        Use Mnemonic to identify Opcode from OpTable
        Copy line of source to work file for later use by pass 2
        CASE Mnemonic OF
                ...
        all others (* including DB *):
                IF Line.Labelled THEN
                        SymbolTable.Enter(Label, Location)
                Location := Location + number of bytes
        END
END
```

.byte

- Drugi prolaz asemblera:

```
Rewind work file, and set Assembling := TRUE
WHILE Assembling DO
        Read a line from work file and
        unpack Mnemonic, Opcode, AddressField
        CASE Mnemonic OF
        ...
        "DB " : Mem[Location] := ValueOf(AddressField);
                    INC(Location)
        END
END
```

.section .text

- Neposredno se za obradu ove direktive koriste sledeći podaci:
  - tabela zapisa **segment_definition,** sa sledećim podacima: ime segmenta, bazna adresa, dužina, deskriptor.

- Prvi prolaz (obrada direktive):

  segment_definition[current_segment].length =
  location_counter - segment_definition[current_segment].base
  current_segment *:= broj segmenta navedenog u direktivi (1=text)*
  opciono, location_counter se uvećava da bude deljiv veličinom mem. reči, ili veličinom stranice virtuelne memorije
  segment_definition[current_segment].base = location_counter

- Drugi prolaz (obrada direktive):

  u predmetni program emitujemo zapise iz vektora segment_definition u odgovarajuću sekciju predmetnog programa.

.global

- Korišćene tabele:
  - U **tabeli simbola**, za svaki simbol uvodi se dodatno bool. polje **globdef** koje označava da li se simbol pojavio u PUBLIC direktivama.

- Prvi prolaz (jedna moguća implementacija):

  ubacujemo simbole u tabelu simbola, postavljamo odgovarajući **globdef** fleg i označavamo simbole kao nedefinisane.

- Drugi prolaz:

  provera da li svaki od simbola sa postavljenim **globdef** flegom ima definiciju. Ako nema, ispisuje se greška.
  redom se numerišu svi globalni (**globuse**==true ili **globdef**==true) i njihovi podaci iz tabele simbola upisuju se u predmetni program, sekcija **symbols**.

Zadatak 3 (septembar ’05)

- Objasniti izmene u strukturama podataka i algoritmu jednoprolaznog asemblera ukoliko su u adresnom polju dozvoljeni jednostavni izrazi, u odnosu na slučaj kada su u adresnom polju dozvoljene samo konstante i simboli. Sintaksa izraza je:

  Izraz = *Član* { "+" *Član* | "-" *Član* } .

  Član = *Simbol* | *Konstanta* | "*" .

- Navesti izgled mašinskog koda, tabele simbola i tabele obraćanja unapred neposredno pre backpatching-a za zadati asemblerski modul:

```
.section .text
    .long l+3
k:  .long l+1, k+2
.section .bss
l:  skip 5
.end
```

Rešenje 3a)

- Pošto nije rečeno suprotno, ograničićemo se na asembler koji generiše apsolutizovani mašinski kod.
- Menjaju se:
  - leksička analiza, da uključi prepoznavanje tokena koji se odnose na +, - i *,
  - sintaksna analiza da dozvoli prepoznavanje izraza u adresnom polju instrukcije,
  - ukoliko izraz u adresnom polju sadrži labele čija vrednost nije još poznata, u mašinski kod na tom mestu ugrađujemo konstantni deo izraza,
  - tabelu obraćanja unapred modifikujemo tako što, pored polja ***patch*** koje ukazuje na adresu reči mašinskog koda koji treba modifikovati kada bude poznata vrednost labele, dodajemo polje ***action*** koje definiše da li će vrednost labele biti dodavana (+) ili oduzimana (-) na postojeći sadržaj posmatranog bajta.

Rešenje 3b)

```
Adr     Maš. Kod                Asembler
00                                      .section .text
00      03 00 00 00                             .long l+3
04      01 00 00 00 06 00 00 00 k:      .long l+1, k+2
0C                                      .section .bss
0C      ?? ?? ?? ?? ??          l:      skip 5
11                                      .end
```

- Symbol Table
- ----------
- l byte ptr 0000000C flink: 00000004+ 00000000+
- K dword ptr 00000004 flink:

Zadatak 4

- Izložiti algoritam za obradu direktiva ekvivalencije čiji izrazi mogu da budu neizračunljivi.
- Primer:

  A EQU B+2

  B EQU C

  C EQU D

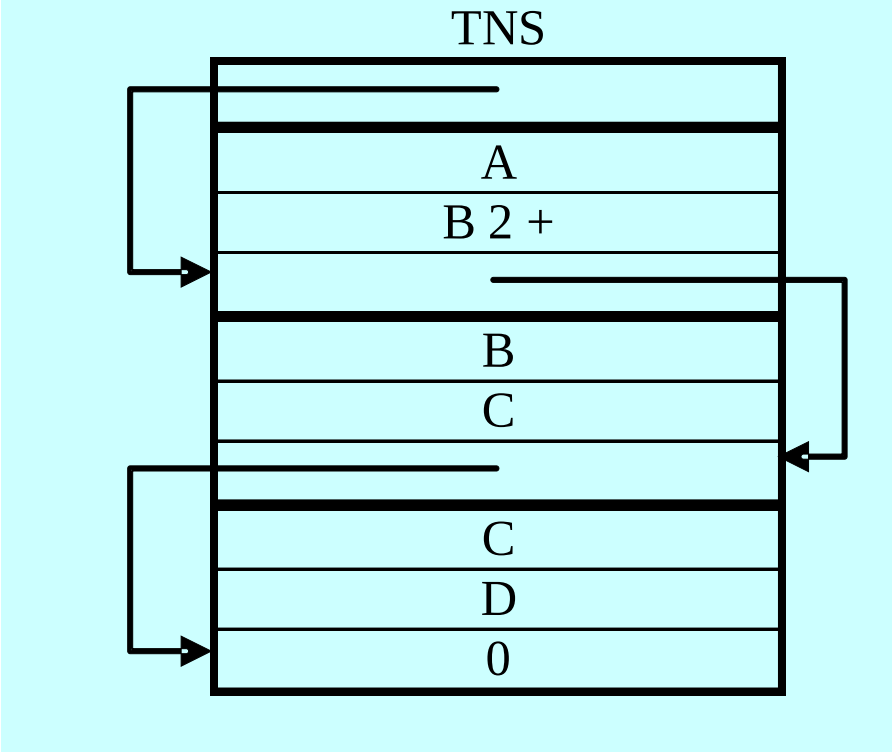
  D EQU 10

Rešenje 4 (1/2)

- Nezgoda je sto se EQU direktiva obrađuje u I prolazu a za navedeni primer se ne može "naivno" rešiti u manje od 4 prolaza.
- Rešenje je u uvođenju tabele neizračunljivih simbola (TNS). Svaki ulaz tabele je oblika kao na sledećoj slici i formiramo ga pri nailasku na EQU direktivu sa neizračunljivim izrazom:
- TNS
  - ukazatelj na sledeći ulaz
  - simbol sa leve strane EQU
  - neizračunljiv izraz u postfiksnoj notaciji
- Izgled tabele za dati primer (D EQU 10 ne ide u TNS, već D ide u tabelu simbola) je dat na sledećem slajdu.

Rešenje 4 (2/2)

- Na kraju prvog prolaza kroz izvorni program iterativno prolazimo kroz TNS, sve dok postoji promena u tabeli. Ako neki simbol možemo izračunati, odgovarajući ulaz se uklanja iz tabele. Ako se tabela isprazni, svi simboli su izračunati. To se neće desiti u slučaju greške kružnog definisanja: A EQU B, B EQU A.



Zadatak 5

- Objasniti obradu literalnih konstanti ako se na mestu literala može pojaviti simbol kojem se vrednost dodeljuje *seta* direktivom, primer u ARM asembleru:

```
    .section .text
    .gbla  l1       @ deklariše se l1 kao glob. simbol
           @ koji sadrži aritmetičku vrednost
l1  seta 4011   @ simbolu l1 se dodeljuje vrednost 4011
    ldr r1, =l1   @ upotreba simboličkog literala
l1  seta 5011   @ dodeljuje se vrednost 5011 simbolu l1
    ldr r2,=l1    @ r2 treba da dobije novu vrednost l1
    .end
```

- Obezbediti da se određena vrednost literala pojavi samo jednom u području literala.

Rešenje 5 (1/3)

- Literali služe za simulaciju neposrednog adresiranja.
- Dati asemblerski program asembler prevodi u sledeći ekvivalentan asemblerski kod:

```
            .section .text
        ldr r1,[pc] ; ucitava se vrednost na koju ukazuje pc
        ldr r2,[pc]; pc ukazuje 8 bajtova iza tekuce instrukcije
lpool:  .long 4011     ; zonu literala asembler smesta odmah iza koda
        .long 5011
        .end
```

- = je oznaka za literal.
- Rezerviše se jedna lokacija u posebnom memorijskom području, u nju upisuje vrednost, a PC relativnim adresiranjem u instrukciji pristupa se toj lokaciji.
- dumpbin /all /disasm primer.obj
- primer.obj je dobijen asembliranjem programa iz postavke (korišćeni Microsoft alati):

```
00011000: E59F1000 ldr      r1, [pc]
00011004: E59F2000 ldr      r2, [pc]
00011008: 00000FAB andeq    r0, r0, r11, lsr #31
0001100C: 00001393
```

Rešenje 5 (2/3)

- Obrada literala u asembleru (prvo varijanta kada su literali konstante npr. =100)
- Strukture podataka
  - tabela literala – jedna od asemblerskih tabela
    - vrednost literala (ključ)
    - dužina literala (bajt, reč,...)
    - adresa u bazenu literala
  - bazen (oblast, područje) literala – jedan segment oper. memorije u vreme izvršavanja programa ("skup svih DCD direktiva")

- Prvi prolaz: kada se naiđe na literal, pristupa se tabeli literala koristeći vrednost literala.
  Ako tražena vrednost već postoji u tabeli, ne radi se ništa, inače se formira novi ulaz u tabeli.
  Definišu se vrednost i dužina, a polje adrese ostaje nepopunjeno.
- Na kraju prvog prolaza saznaje se početna adresa područja literala. Prolazi se kroz sve ulaze u TL i redom definišu adrese, jedna iza druge.
- U drugom prolazu, kada se naiđe na literal, na osnovu vrednosti čita se iz TL adresa i ugrađuje u kod.

Rešenje 5 (3/3)

- Modifikacija algoritma kada se dozvoljava navođenje simboličke vrednost literala, npr.: =l1
- Jedna moguća implementacija:
  - Za ovakve literale u prvom prolazu ne formira se ulaz u TL (dakle ne radi se ništa) jer potencijalno ne znamo njihovu vrednost.
  - U drugom prolazu, kada naiđemo na takav literal, prvo potražimo odgovarajuću vrenost (koju sada znamo) u TL, ako postoji pročitanu adresu ugrađujemo u kod, a ako ne postoji formiramo novi ulaz u TL, i alociramo novu lokaciju na kraju područja literala i njenu adresu ugrađujemo u kod.

Zadatak 6

- Kod novijih procesora iz x86 familije postoji mogućnost izbora odgovarajuće veličine pomjeraja. Tako se za skok na lokacije koje su dovoljno blizu lokacije instrukcije uslovnog skoka koristi 8-bitni pomjeraj, dok se u suprotnom slučaju koristi 32-bitni označeni pomjeraj.
- Objasniti algoritam kojim je potrebno proširiti školski asembler kako bi se programer oslobodio razmišljanja o optimalnoj veličini pomjeraja kod uslovnih skokova.
- Pretpostaviti da u svakom fajlu postoji samo jedan segment sa kodom, kao i da linker "ne zna" za optimizaciju.

Rešenje 6 (1/8)

- Ovu optimizaciju asembler radi iterativno. U prvom prolazu asembler pretpostavlja da je za sve pomjeraje dovoljano 8 bita. U narednom prolazu provjerava da li je pretpostavka tačna.
- Ako nije, svaka instrukcija uslovnog skoka čiji pomjeraj ne može da stane u jedan bajt, mijenja se odgovarajućom instrukcijom sa 32-bitnim pomjerajem.
- Pri tome se kod povećava, pa može doći i do promene vrednosti pomeraja neke od instrukcija uslovnog skoka koje su već obrađene. Tako se može desiti da neki od pomjeraja više ne može da stanu u 8 bita, već se instrukcija mora zameniti onom sa 32-bitnim pomjerajem.
- Ako je bar jedan skok izmjenjen, neophodno je umetnuti još jedan prolaz identičan prethodnom sa jedinom razlikom što se posmatra stanje nakon prethodnog prolaza a ne stanje nakon prvog prolaza.
- Ovaj postupak se ponavlja sve do trenutka kada se u poslednjem prolazu ne napravi nijedna izmjena. Tada su pomeraji svih instrukcija odgovarajuće veličine, pa se prelazi na poslednji prolaz u kojem se generiše kod (na isti način kako se to radilo bez optimizacije u drugom prolazu).

Rešenje 6 (2/8)

- Uvodi se dodatna tabela uslovnih skokova:
  - adresa instrukcije uslovnog skoka
  - tekuća veličina instrukcije (ili polja za pomjeraj)
  - adresa sledećeg zapisa

- Ova tabela se popunjava u prvom prolazu. Svi naredni prolazi osim poslednjeg prolaze samo kroz ovu listu. Svaki put kada se promeni veličina neke instrukcije skoka:
  - ažuriraju se veličina instrukcije,
  - ažuriraju se adrese drugih skokova u ovoj tabeli koji se nalaze na adresi većoj od adrese upravo prepravljenog skoka
  - ažuriraju se vrednosti svih simboli u tabeli simbola čija vrednost predstavlja adresu koja je veća od adrese upravo prepravljenog uslovnog skoka.
  - ažuriranja adresa skokova i vrednosti simbola se vrše tako što se vrednosti uvećavaju za razliku u veličini nove i stare instrukcije (4 bajta u posmatranom slučaju).
- Opciono, simboli u tabeli simbola, čija vrijednost predstavlja adresu, mogli bi biti ulančani u listu sortiranu po rastućoj vrednosti simbola. Tako bi se u tabeli uslovnih skokova moglo dodati još jedno polje koje bi pokazivalo na prvi sledeći simbol posle instrukcije skoka i time bi bilo ubrzano ažuriranje tabele simbola.

Rešenje 6 (3/8)

- Pod pretpostavkom da je operacioni kod nekog uslovnog skoka sa 8-bitnim pomerajem 0x74, a da je operacioni kod istog tog uslovnog skoka sa 32-bitnim pomerajem 0xF0, 0x84, prikazati proces optimizacije sledećeg programa :

```
                je lab_A
    .byte skip 60
                je lab_B
    .byte skip 64
lab_A:          je lab_C
    .byte skip 59
lab_B: .byte skip 200
lab_C:
```

Rešenje 6 (4/8)

- posle prvog prolaza:
  - tabela uslovnih skokova:

ulaz	adresa	veličina	sledeći
0	00	2	1
1	62	2	2
2	128	2	0 //kraj liste

  - tabela simbola

simbol	vrijednost	adresa ...
lab_A	0x80	da
lab_B	0xBD	da
lab_C	0x185	da

adresa sadržaj memorije

```
00:     0x74 ??   ;lab_A-2 = 0x7E      ok
02:     ?...
62:     0x74 ??   ;lab_B-64 = 0x7D     ok
64:     ?...
128: (lab_A)      0x74 ??   ;lab_C-130 = 0x103  !!!greška
130:    ?...
189: (lab_B)      ?...
389: (lab_C)
```

Rešenje 6 (5/8)

- posle drugog prolaza:
  - tabela uslovnih skokova:

ulaz	adresa	veličina	sledeći
0	00	2	1
1	62	2	2
2	128	6	0 //kraj liste

  - tabela simbola

simbol	vrijednost	adresa ...
lab_A	0x80	da
lab_B	0xC1	da
lab_C	0x189	da

adresa sadržaj memorije

```
00:     0x74 ??   ;lab_A-2 = 0x7E      ok
02:     ?...
62:     0x74 ??   ;lab_B-64 = 0x81     !!!greška
64:     ?...
128: (lab_A)      0xF0 0x84 ?? ?? ?? ??  ;lab_C-134 = 0x103  ok
134:    ?...
193: (lab_B)      ?...
393: (lab_C)
```

Rešenje 6 (6/8)

- posle treceg prolaza:
  - tabela uslovnih skokova:

ulaz	adresa	veličina	sledeći
0	00	2	1
1	62	6	2
2	128	6	0 //kraj liste

  - tabela simbola

simbol	vrijednost	adresa ...
lab_A	0x84	da
lab_B	0xC5	da
lab_C	0x18D	da

adresa	sadržaj memorije
00:	0x74 ?? ;lab_A-2 = 0x82 !!!greška
02:	?...
62:	0xF0 0x84 ?? ?? ?? ?? ;lab_B-68 = 0x81 ok
68:	?...
132: (lab_A)	0xF0 0x84 ?? ?? ?? ?? ;lab_C-138 = 0x103 ok
138:	?...
197: (lab_B)	?...
397: (lab_C)	

Rešenje 6 (7/8)

- posle cetvrtog prolaza prolaza:
  - tabela uslovnih skokova:

ulaz	adresa	veličina	sledeći
0	00	6	1
1	62	6	2
2	128	6	0 //kraj liste

  - tabela simbola

simbol	vrijednost	adresa ...
lab_A	0x88	da
lab_B	0xC9	da
lab_C	0x191	da

adresa	sadržaj memorije	
00:	0xF0 0x84 lab_A	;lab_A-6 = 0x82 ok
06:	?...	
66:	0xF0 0x84 lab_B	;lab_B-72 = 0x81 ok
72:	?...	
136: (lab_A)	0xF0 0x84 lab_C	;lab_C-142 = 0x103 ok
142:	?...	
201: (lab_B)	?...	
401: (lab_C)		

Rešenje 6 (8/8)

- U petom prolazu će se utvrditi da je proces optimizacije završen i u sledećem, poslednjem prolazu će se izgenerisati izlaz koji se u standardnom dvoprolaznom asembleru generiše u drugom prolazu.
- **Napomene**:
- 1. Ovdje je u toku algoritma prikazivan sadržaj memorije radi lakšeg razumjevanja. Taj sadržaj u tim trenutcima ne postoji, već se generiše u poslednjem prolazu. Umjesto ovakvog sadržaja memorije postoji radni fajl koji sadrži program koji se prevodi. Radni fajl se koristi za dalju obradu i generisanje koda. Primjetiti da bi se algoritam mogao implementirati i bez dodatnih struktura podataka, ali bi tada međuprolazi morali da prolaze čitav kod, a ne samo kroz kritične instrukcije (instrukcije uslovnog skoka).
- 2. U ovom primjeru su svi uslovni skokovi na kraju prepravljeni na 32-bitni pomjeraj, što u opštem slučaju ne mora da se desi.
- 3. U ovom primjeru se u svakoj iteraciji razrješavao samo po jedan skok. U opštem slučaju, može se desiti da se u nekoj iteraciji primjeti više pomjeraja koji ne mogu stati u 8-bitno polje i tada se sve te instrukcije mijenjaju odgovarajućom instrukcijom sa 32-bitnim pomjerajem.

Zadatak 7 (1/2)

- U x86 asembler želi se uvesti direktiva .common kojom se uvodi poseban memorijski segment zajednički za sve module. Na primer:

  .common
  x: skip 10
  y: skip 1
  z: skip 5

- rezerviše tri uzastopna bloka memorijskih lokacija i to prvi od 10, drugi od jedan i treći od 5 bajtova. Za svaki prvi bajt u blokovima vezan je jedan simbol (u gornjem primeru to su x, y i z), pri čemu je reč o lokalnim simbolima modula. Kod povezivanja alocira se isti memorijski prostor (početna adresa je zajednička) za sve common segmente svih modula. Veličina prostora odgovara veličini najvećeg segmenta.
- a) Opisati potrebne mogućnosti formata predmetnog programa da bi se obradila direktiva .common.
- b) Navesti izgled predmetnih programa po formatu opisanom u prethodnoj tački za date module (moduli su dati na sledećem slajdu).
- c) Za sve segmente datih modula navesti početne adrese posle povezivanja i punjenja u memoriju.

Zadatak 7 (2/2)

```
.common
m:   .skip 5
n:   .skip 1
.text
     .long  m, n
.bss
l:   .skip 10
.end
```

```
.common
i:   .skip 1
j:   .skip 2
k:   .skip 1
.text
     .long i
.data
     .long j, k
.end
```

Rešenje 7 (1/2)

- a) Pojava atributa *common* u asemblerskom programu povlači uvođenje segmenta pod imenom .common u objektni program. Taj segment će biti označen tako da se pri povezivanju i punjenju common segmenti pojedinih modula preklapaju od iste početne adrese operativne memorije.

- b)

#tabela simbola

#naziv	sekcija	vr.	Vez.	R.b.	atr
	UND	0	local	0	
.comm	.comm	0	local	1	AW
.text	.text	0	local	2	AXP
.bss	.bss	0	local	3	AW

#.rel.text

# ofset	tip	redni_broj
0	R_386_32	1
4	R_386_32	1

#.text

00 00 00 00 05 00 00 00

#tabela simbola

#naziv	sekcija	vr.	Vez.	R.b.	atr
	UND	0	local	0	
.comm	.comm	0	local	1	AW
.text	.text	0	local	2	AXP
.data	.data	0	local	3	AWP

#.rel.text.

# offset	tip	redni_broj
0	R_386_32	1

#.rel.data

# ofset	tip	redni_broj
0	R_386_32	1
4	R_386_32	1

#.text

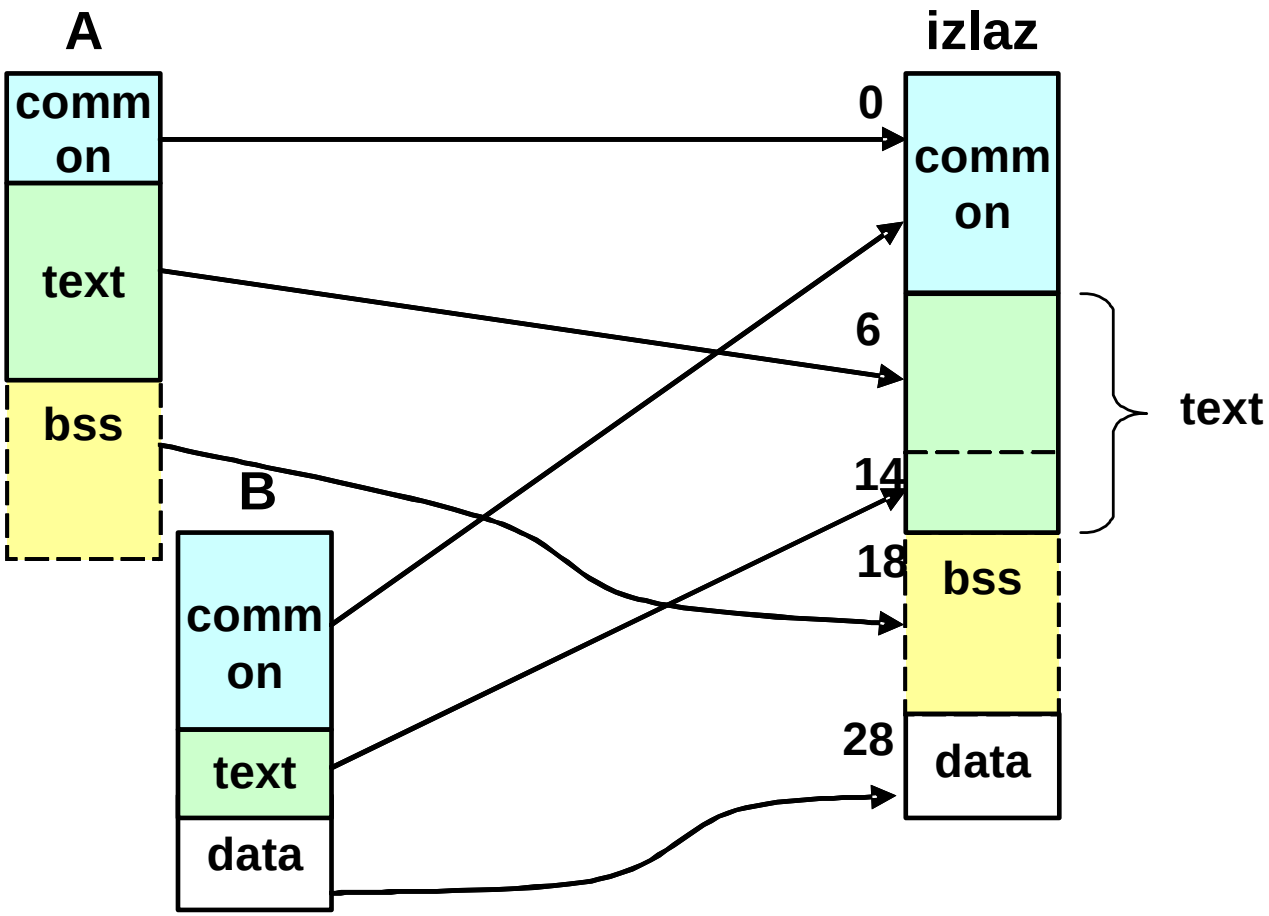
00 00 00 00

#.data

01 00 00 00 03 00 00 00

Rešenje 7 (2/2)

- c)



Zadatak 8 (1/2)

- U nekim asemblerima za x86 procesor za segment je moguće navesti COMMON atribut kombinovanja sa istoimenim segmentima iz drugih modula. Taj atribut označava da svi segmenti sa istim nazivom iz svih objektnih fajlova koji se povezuju treba da se poklope na istu adresu. Veličina takvog segmenta se određuje kao veličina najvećeg segmenta koji je kombinovan.Na primer:
- ;fajl1.asm: rezervišu se 4 bajta

```
.common
X: .skip 1
Y: .skip 1
Z: .skip 2
```

- ;fajl2.asm rezerviše se 5 bajtova

```
.common
A: .skip 2
B: .skip 2
C: .skip 1
```

Zadatak 8 (2/2)

- U navedenom primjeru u fajlu "fajl1.asm" pomoću X i Y pristupa se istim bajtovima kojima se u fajlu "fajl2.asm" pristupa sa A. Isto važi i za Y i B, dok za C ne postoji odgovarajući simbol u fajlu "fajl1.asm". Odrediti koji su od sledećih izraza programski relokativni, koji COMMON relokativni a koji nekorektni:

1. P1-A	2. P1-P2+P3	3. *+A	4. *+P1-*
5. C1+P1	6. C1-P1	7. C2+C1-P2	8. C1-A
9. C1-C2+C3	10. *-P1+P2	11. C1-C2+C3-A	
12. -C1+C2+C3+P1-P2			

- P1,P2,P3 su programski relokativne promenljive,
- C1,C2,C3 su COMMON relokativne promenljive
- A je apsolutna promenljiva, * je programski brojač.

Rešenje 8

- Klasifikacija asemblerskih izraza vrši se na osnovu indeksa klasifikacije:
  - Najpre se svakom apsolutnom simbolu iz asemblerskog izraza dodeli 0, a relokativnom 1, pa se izraz sračuna sa tim vrednostima Vrednost izraza je indeks klasifikacije IK. Ako je IK=0, izraz je apsolutni. Ako je IK=1, izraz je relokativan. Ako je IK $\neq$0 ili IK $\neq$1 izraz je nekorektan.
  - u slučaju postojanja COMMON zona, relokativne simbole iz tih zona označavaćemo sa 1c, a relokativne simbole iz programa sa 1p. Vrednost izraza 1c - 1p može zavisiti od redosleda punjenja modula, pa zato uzimamo da je takav izraz nekorektan.

- 1. 1p - 0 = 1p .....programski relokativan
- 2. 1p - 1p + 1p = 1p .....programski relokativan
- 3. 1p + 0 = 1p .....programski relokativan
- 4. 1p + 1p - 1p = 1p .....programski relokativan
- 5. 1c+1p .....nekorektan
- 6. 1c - 1p .....nekorektan
- 7. 1c + 1c - 1p .....nekorektan
- 8. 1c - 0 = 1c .....COMMON relokativan
- 9. 1c - 1c + 1c = 1c .....COMMON relokativan
- 10. 1p - 1p + 1p = 1p .....:programski relokativan
- 11. 1c - 1c + 1c - 0 = 1c .....COMMON relokativan
- 12. -1c + 1c + 1c + 1p - 1p = 1c .....COMMON relokativan

Zadatak 9

- Prikazati predmetni program po školskom formatu sledećeg programa napisanog na asembleru x86 procesora.

```
.global a, c
.extern b
.text
        jz      a
        jz      e
        jz      b
        jz      d
d:      .long    d
        mov     eax, b
        mov     c, eax
        mov     eax, e
.data
.skip           8
e:      .long   a - e + d
        .long   c
        .long   .bss
a:      .long   b
.bss
c:      .skip   8
.end
```

```
#.ret.text
#ofset          tip             vr  [.text]:
00000014  R_386_32      1
00000019  R_386_32      8
0000001e  R_386_32      7
00000023  R_386_32      2
00000002  R_386_PC32    6
00000008  R_386_PC32    2
0000000e  R_386_PC32    8
#.ret.data
 # ofset         tip             vr [.data]:
00000008 R_386_32       1
0000000c R_386_32       7
00000010 R_386_32       3
00000014 R_386_32       8
#.text
0f  84 fc  ff   ff  ff  0f  84   04 00 00 00   0f  84 fc  ff
ff  ff   74 00  14 00 00 00   a1 00 00 00   00 a3 00 00
00 00 a1 08  00 00 00
#.data
00 00 00 00   00 00 00 00   20 00 00 00   00000000
00 00 00 00   00 00 00 00
```

```
#tabela simbola
#ime  sek   vr.  vid.  r.b.
      UND     0   l     0
.text   text    0   l     1
.data   data    0   l     2
.bss    bss     0   l     3
e       data    8   l     4
d       text    14  l     5
a       data    14  g     6
c       bss     0   g     7
b     UND     0   g     8
```

Zadatak 9

- Prikazati predmetni program po školskom formatu sledećeg programa napisanog na asembleru x86 procesora.

```
.global a, c
.extern b
.text
        jz      a
        jz      e
        jz      b
        jz      d
d:      .long     d
        mov     eax, b
        mov     c, eax
        mov     eax, e
.data
.skip           8
e:      .long   a - e + d
        .long   c
        .long   .bss
a:      .long   b
.bss
c:      .skip   8
.end
```

```
#.ret.text
#ofset          tip             vr  [.text]:
00000014  R_386_32      1
00000019  R_386_32      8
0000001e  R_386_32      7
00000023  R_386_32      2
00000002  R_386_PC32    6
00000008  R_386_PC32    2
0000000e  R_386_PC32    8
#.ret.data
 # ofset         tip             vr [.data]:
00000008 R_386_32       1
0000000c R_386_32       7
00000010 R_386_32       3
00000014 R_386_32       8
#.text
0f  84 fc  ff   ff  ff   0f  84   04 00 00 00   0f  84 fc  ff
ff  ff   74 00  14 00 00 00   a1 00 00 00   00 a3 00 00
00 00 a1 08  00 00 00
#.data
00 00 00 00   00 00 00 00   20 00 00 00   00000000
00 00 00 00   00 00 00 00
```

```
#tabela simbola
#ime  sek   vr.  vid.  r.b.
      UND     0   l     0
.text   text    0   l     1
.data   data    0   l     2
.bss    bss     0   l     3
e       data    8   l     4
d       text    14  l     5
a       data    14  g     6
c       bss     0   g     7
b     UND     0   g     8
```

Zadatak 9

- Prikazati predmetni program po školskom formatu sledećeg programa napisanog na asembleru x86 procesora.

```
.global a, c
.extern b
.text
      jz      a
      jz      e
      jz      b
      jz      d
d:    .long    d
      mov     eax, b
      mov     c, eax
      mov     eax, e
.data
.skip         8
e:    .long   a - e + d
      .long   c
      .long   .bss
a:    .long   b
.bss
c:    .skip   8
.end
```

```
#.ret.text
#ofset        tip              vr  [.text]:
00000014  R_386_32       1
00000019  R_386_32       8
0000001e  R_386_32       7
00000023  R_386_32       2
00000002  R_386_PC32     6
00000008  R_386_PC32     2
0000000e  R_386_PC32     8
#.ret.data
 # ofset        tip              vr [.data]:
00000008 R_386_32        1
0000000c R_386_32        7
00000010 R_386_32        3
00000014 R_386_32        8
#.text
0f  84 fc  ff   ff  ff   0f  84    04 00 00 00   0f  84 fc  ff
 ff  ff   74 00  14 00 00 00   a1 00 00 00   00 a3 00 00
00 00 a1 08  00 00 00
#.data
00 00 00 00   00 00 00 00   20 00 00 00   00000000
00 00 00 00   00 00 00 00
```

```
#tabela simbola
#ime  sek   vr.  vid.  r.b.
      UND     0    l      0
.text   text    0    l      1
.data   data    0    l      2
.bss    bss     0    l      3
e       data    8    l      4
d       text    14   l      5
a       data    14   g      6
c       bss     0    g      7
b     UND     0    g      8
```

Zadatak 9

- Prikazati predmetni program po školskom formatu sledećeg programa napisanog na asembleru x86 procesora.

```
.global a, c
.extern b
.text
      jz      a
      jz      e
      jz      b
      jz      d
d:    .long   d
      mov     eax, b
      mov     c, eax
      mov     eax, e
.data
.skip         8
e:    .long   a - e + d
      .long   c
      .long   .bss
a:    .long   b
.bss
c:    .skip   8
.end
```

```
#.ret.text
#ofset        tip              vr  [.text]:
00000014  R_386_32       1
00000019  R_386_32       8
0000001e  R_386_32       7
00000023  R_386_32       2
00000002  R_386_PC32     6
00000008  R_386_PC32     2
0000000e  R_386_PC32     8
#.ret.data
 # ofset        tip              vr [.data]:
00000008 R_386_32        1
0000000c R_386_32        7
00000010 R_386_32        3
00000014 R_386_32        8
#.text
0f  84 fc  ff   ff  ff   0f  84   04 00 00 00   0f  84 fc  ff
ff  ff   74 00  14 00 00 00   a1 00 00 00   00 a3 00 00
00 00 a1 08  00 00 00
#.data
00 00 00 00   00 00 00 00   20 00 00 00   00000000
00 00 00 00   00 00 00 00
```

```
#tabela simbola
#ime  sek   vr.  vid.  r.b.
      UND     0    l      0
.text   text    0    l      1
.data   data    0    l      2
.bss    bss     0    l      3
e       data    8    l      4
d       text   14    l      5
a       data   14    g      6
c       bss     0    g      7
b     UND     0    g      8
```

Zadatak 9

- Prikazati predmetni program po školskom formatu sledećeg programa napisanog na asembleru x86 procesora.

```
.global a, c
.extern b
.text
      jz      a
      jz      e
      jz      b
      jz      d
d:    .long   d
      mov     eax, b
      mov     c, eax
      mov     eax, e
.data
.skip         8
e:    .long   a - e + d
      .long   c
      .long   .bss
a:    .long   b
.bss
c:    .skip   8
.end
```

```
#.ret.text
#ofset        tip              vr  [.text]:
00000014  R_386_32       1
00000019  R_386_32       8
0000001e  R_386_32       7
00000023  R_386_32       2
00000002  R_386_PC32     6
00000008  R_386_PC32     2
0000000e  R_386_PC32     8
#.ret.data
 # ofset        tip              vr [.data]:
00000008 R_386_32        1
0000000c R_386_32        7
00000010 R_386_32        3
00000014 R_386_32        8
#.text
0f  84 fc  ff   ff  ff   0f  84   04 00 00 00   0f  84 fc  ff
ff  ff   74 00  14 00 00 00  a1 00 00 00  00 a3 00 00
00 00 a1 08  00 00 00
#.data
00 00 00 00   00 00 00 00   20 00 00 00   00000000
00 00 00 00   00 00 00 00
```

```
#tabela simbola
#ime  sek   vr.  vid.  r.b.
      UND     0   l     0
.text   text    0   l     1
.data   data    0   l     2
.bss    bss     0   l     3
e       data    8   l     4
d       text    14  l     5
a       data    14  g     6
c       bss     0   g     7
b     UND     0   g     8
```

Zadatak 9

- Prikazati predmetni program po školskom formatu sledećeg programa napisanog na asembleru x86 procesora.

```
.global a, c
.extern b
.text
      jz      a
      jz      e
      jz      b
      jz      d
d:    .long     d
      mov     eax, b
      mov     c, eax
      mov     eax, e
.data
.skip         8
e:    .long   a - e + d
      .long   c
      .long   .bss
a:    .long   b
.bss
c:    .skip   8
.end
```

```
#.ret.text
#ofset          tip              vr  [.text]:
00000014  R_386_32       1
00000019  R_386_32       8
0000001e  R_386_32       7
00000023  R_386_32       2
00000002  R_386_PC32     6
00000008  R_386_PC32     2
0000000e  R_386_PC32     8
#.ret.data
 # ofset          tip              vr [.data]:
00000008 R_386_32        1
0000000c R_386_32        7
00000010 R_386_32        3
00000014 R_386_32        8
#.text
0f  84 fc  ff   ff  ff   0f  84    04 00 00 00   0f  84 fc  ff
ff  ff   74 00  14 00 00 00   a1 00 00 00   00 a3 00 00
00 00 a1 08  00 00 00
#.data
00 00 00 00   00 00 00 00   20 00 00 00   00000000
00 00 00 00   00 00 00 00
```

```
#tabela simbola
#ime  sek   vr.  vid.  r.b.
      UND     0   l     0
.text   text    0   l     1
.data   data    0   l     2
.bss    bss     0   l     3
e       data    8   l     4
d       text    14  l     5
a       data    14  g     6
c       bss     0   g     7
b     UND     0   g     8
```

Zadatak 9

- Prikazati predmetni program po školskom formatu sledećeg programa napisanog na asembleru x86 procesora.

```
.global a, c
.extern b
.text
      jz      a
      jz      e
      jz      b
      jz      d
d:    .long     d
      mov     eax, b
      mov     c, eax
      mov     eax, e
.data
.skip         8
e:    .long   a - e + d
      .long   c
      .long   .bss
a:    .long   b
.bss
c:    .skip   8
.end
```

```
#.ret.text
#ofset        tip              vr  [.text]:
00000014  R_386_32       1
00000019  R_386_32       8
0000001e  R_386_32       7
00000023  R_386_32       2
00000002  R_386_PC32     6
00000008  R_386_PC32     2
0000000e  R_386_PC32     8
#.ret.data
 # ofset        tip              vr [.data]:
00000008 R_386_32        1
0000000c R_386_32        7
00000010 R_386_32        3
00000014 R_386_32        8
#.text
0f  84 fc  ff   ff  ff   0f  84    04 00 00 00   0f  84 fc  ff
ff  ff   74 00  14 00 00 00   a1 00 00 00   00 a3 00 00
00 00 a1 08  00 00 00
#.data
00 00 00 00   00 00 00 00   20 00 00 00   00000000
00 00 00 00   00 00 00 00
```

```
#tabela simbola
#ime  sek   vr.  vid.  r.b.
      UND     0   l     0
.text   text    0   l     1
.data   data    0   l     2
.bss    bss     0   l     3
e       data    8   l     4
d       text    14  l     5
a       data    14  g     6
c       bss     0   g     7
b     UND     0   g     8
```

Zadatak 9

- Prikazati predmetni program po školskom formatu sledećeg programa napisanog na asembleru x86 procesora.

```
.global a, c
.extern b
.text
        jz      a
        jz      e
        jz      b
        jz      d
d:      .long    d
        mov     eax, b
        mov     c, eax
        mov     eax, e
.data
.skip           8
e:      .long   a - e + d
        .long   c
        .long   .bss
a:      .long   b
.bss
c:      .skip   8
.end
```

```
#.ret.text
#ofset          tip             vr  [.text]:
00000014  R_386_32      1
00000019  R_386_32      8
0000001e  R_386_32      7
00000023  R_386_32      2
00000002  R_386_PC32    6
00000008  R_386_PC32    2
0000000e  R_386_PC32    8
#.ret.data
 # ofset         tip             vr [.data]:
00000008 R_386_32       1
0000000c R_386_32       7
00000010 R_386_32       3
00000014 R_386_32       8
#.text
0f  84 fc  ff   ff  ff   0f  84   04 00 00 00   0f  84 fc  ff
ff  ff   74 00  14 00 00 00   a1 00 00 00   00 a3 00 00
00 00 a1 08  00 00 00
#.data
00 00 00 00   00 00 00 00   20 00 00 00   00000000
00 00 00 00   00 00 00 00
```

```
#tabela simbola
#ime  sek   vr.  vid.  r.b.
      UND     0    l      0
.text   text    0    l      1
.data   data    0    l      2
.bss    bss     0    l      3
e       data    8    l      4
d       text    14   l      5
a       data    14   g      6
c       bss     0    g      7
b     UND     0    g      8
```

Zadatak 9

- Prikazati predmetni program po školskom formatu sledećeg programa napisanog na asembleru x86 procesora.

```
.global a, c
.extern b
.text
      jz      a
      jz      e
      jz      b
      jz      d
d:    .long     d
      mov     eax, b
      mov     c, eax
      mov     eax, e
.data
.skip         8
e:    .long   a - e + d
      .long   c
      .long   .bss
a:    .long   b
.bss
c:    .skip   8
.end
```

```
#.ret.text
#ofset          tip              vr  [.text]:
00000014  R_386_32       1
00000019  R_386_32       8
0000001e  R_386_32       7
00000023  R_386_32       2
00000002  R_386_PC32     6
00000008  R_386_PC32     2
0000000e  R_386_PC32     8
#.ret.data
 # ofset          tip              vr [.data]:
00000008 R_386_32        1
0000000c R_386_32        7
00000010 R_386_32        3
00000014 R_386_32        8
#.text
0f  84 fc  ff   ff  ff   0f  84    04 00 00 00   0f  84 fc  ff
ff  ff   74 00  14 00 00 00   a1 00 00 00   00 a3 00 00
00 00 a1 08  00 00 00
#.data
00 00 00 00   00 00 00 00   20 00 00 00   00000000
00 00 00 00   00 00 00 00
```

```
#tabela simbola
#ime  sek   vr.  vid.  r.b.
      UND     0   l     0
.text   text    0   l     1
.data   data    0   l     2
.bss    bss     0   l     3
e       data    8   l     4
d       text    14  l     5
a       data    14  g     6
c       bss     0   g     7
b     UND     0   g     8
```

Zadatak 9

- Prikazati predmetni program po školskom formatu sledećeg programa napisanog na asembleru x86 procesora.

```
.global a, c
.extern b
.text
        jz      a
        jz      e
        jz      b
        jz      d
d:      .long    d
        mov     eax, b
        mov     c, eax
        mov     eax, e
.data
.skip           8
e:      .long   a - e + d
        .long   c
        .long   .bss
a:      .long   b
.bss
c:      .skip   8
.end
```

```
#.ret.text
#ofset          tip             vr  [.text]:
00000014  R_386_32      1
00000019  R_386_32      8
0000001e  R_386_32      7
00000023  R_386_32      2
00000002  R_386_PC32    6
00000008  R_386_PC32    2
0000000e  R_386_PC32    8
#.ret.data
 # ofset         tip             vr [.data]:
00000008 R_386_32       1
0000000c R_386_32       7
00000010 R_386_32       3
00000014 R_386_32       8
#.text
0f  84 fc  ff   ff  ff   0f  84    04 00 00 00   0f  84 fc  ff
ff  ff   74 00  14 00 00 00   a1 00 00 00   00 a3 00 00
00 00 a1 08  00 00 00
#.data
00 00 00 00   00 00 00 00   20 00 00 00   00000000
00 00 00 00   00 00 00 00
```

```
#tabela simbola
#ime  sek   vr.  vid.  r.b.
      UND     0   l     0
.text   text    0   l     1
.data   data    0   l     2
.bss    bss     0   l     3
e       data    8   l     4
d       text    14  l     5
a       data    14  g     6
c       bss     0   g     7
b     UND     0   g     8
```

Zadatak 9

- Prikazati predmetni program po školskom formatu sledećeg programa napisanog na asembleru x86 procesora.

```
.global a, c
.extern b
.text
      jz      a
      jz      e
      jz      b
      jz      d
d:    .long     d
      mov     eax, b
      mov     c, eax
      mov     eax, e
.data
.skip         8
e:    .long   a - e + d
      .long   c
      .long   .bss
a:    .long   b
.bss
c:    .skip   8
.end
```

```
#.ret.text
#ofset          tip              vr  [.text]:
00000014  R_386_32       1
00000019  R_386_32       8
0000001e  R_386_32       7
00000023  R_386_32       2
00000002  R_386_PC32     6
00000008  R_386_PC32     2
0000000e  R_386_PC32     8
#.ret.data
 # ofset          tip               vr [.data]:
00000008 R_386_32        1
0000000c R_386_32        7
00000010 R_386_32        3
00000014 R_386_32        8
#.text
0f  84 fc  ff   ff  ff   0f  84    04 00 00 00   0f  84 fc  ff
ff  ff   74 00  14 00 00 00   a1 00 00 00   00 a3 00 00
00 00 a1 08  00 00 00
#.data
00 00 00 00   00 00 00 00   20 00 00 00   00000000
00 00 00 00  00 00 00 00
```

```
#tabela simbola
#ime  sek   vr.  vid.  r.b.
      UND     0    l      0
.text   text    0    l      1
.data   data    0    l      2
.bss    bss     0    l      3
e       data    8    l      4
d       text    14   l      5
a       data    14   g      6
c       bss     0    g      7
b     UND     0    g      8
```