

Jovan Vunić, jun 2018.

SKRIPTA 44

A.K.A.

THE SPPURVBUSTER

Posebno zahvalan Tanosu našeg vremena,
Opačiću Borisu.

Moguća eliminaciona pitanja za ispit iz SPPuRV1

- 1. Navesti tehnike za definisanje novog lingvističkog nivoa programske podrške?**
 - a. Proširenje – nove procedure koriste primitive osnovnog sistema
 - b. Prevodenje – sa novog jezika na jezik osnovnog sistema
 - c. Interpretacija – faze prevodenja i izvršenja su vremenski zavisne.
- 2. Šta je sistemska programska podrška?**
 - a. SPP je nivo programske podrške koji pripada računarskom sistemu i stoji na raspolaganju svim korisnicima.
- 3. Šta je aplikaciona programska podrška?**
 - a. APP obuhvata programe koje su uneli korisnici radi željene obrade.
- 4. Navesti osobine SPP?**
 - a. Skup programa se instalira posebnim procedurama od strane odgovornih lica.
 - b. U hijerarhiji programske podrške definiše se jedan lingvistički nivo koji se odnosi na sve korisnike skupa programa i omogućuje im definisanje sopstvenog lingvističkog nivoa.
 - c. Unutrašnji lingvistički nivoi hijerarhije programske podrške su skriveni u odnosu na korisnika.
- 5. Navesti osobine APP?**
 - a. Programi definišu novi lingvistički nivo proširivanjem, prevodenjem, interpretacijom ili kombinacijom ovih tehnika.
 - b. Lingvistički nivo koji se definiše razvijenim programom ne obezbeđuje uslove za definisanje gornjih lingvističkih nivoa (tj. direktno je vezan za korisnika).
- 6. Šta je datoteka i iz čega je izgrađena?**
 - a. Datoteka je skup slogova.
 - b. Slog je osnovna jedinica obradivane informacije u obliku liste objekata (polja) informacionog karaktera.
 - c. Polje je skup alfanumeričkih znakova.
- 7. Za šta služe asimptotske notacije i koje sve postoje?**
 - a. Služe za opis vremena izvršenja algoritma $T(n)$, pri čemu je n iz skupa N i predstavlja veličinu ulaznih podataka.
 - b. Postoje Θ , O , Ω , o i ω notacija.
- 8. Kako glasi teorema o asimptotskim notacijama?**
 - a. Za funkcije $f(n)$ i $g(n)$ važi da je $f = \Theta(g)$ akko $f = \Omega(g)$ i $f = O(g)$.
- 9. Čemu je jednak broj anonimnih funkcija u nekom izrazu? Koliko anonimnih funkcija postoji u izrazu $\sum O(i)$?**
 - a. Broj anonimnih funkcija u bilo kom izrazu jednak je broju pojava asimptotskih notacija u tom izrazu.
 - b. U izrazu $\sum O(i)$ postoji jedna anonimna funkcija, zavisna od i .
- 10. Interpretirati značenje izraza $2n^2 + \Theta(n) = \Theta(n^2)$?**
 - a. Ako uzmemo da je $f(n) = \Theta(n)$, tada za $f(n)$ postoji $g(n) \in \Theta(n^2)$, takva da je $2n^2 + f(n) = g(n)$ za svako n .
- 11. Interpretirati značenje izraza $2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$?**

Primenom gornjeg pravila svaka jednačina u lancu se interpretira nezavisno

 - Najpre: $2n^2 + 3n + 1 = 2n^2 + f(n)$
 - A zatim: $2n^2 + f(n) = h(n)$, $f(n) \in \Theta(n)$, $h(n) \in \Theta(n^2)$
 - Zaključak: $2n^2 + 3n + 1 = \Theta(n^2)$
- 12. Čime se opisuje vreme izvršenja algoritma koji sadrži rekurzivne pozive?**
 - a. Rekurentnom jednačinom ili, jednostavnije, rekurencijom.

13. Kako glasi opšta rekurentna jednačina za $T(n)$ algoritma koji je zasnovan na principu PODELI i ZAVLADAJ?

Opšta rekurentna jednačina za $T(n)$ algoritma zasnovanog na pristupu podeli i zavladaj:

$$T(n) = \begin{cases} \Theta(1), & n \leq c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n), & \text{inače} \end{cases}$$

14. Ako je dat kod za rekursivno računanje Fibonačijevog niza, kako treba paralelizovati dati kod?

U kojoj liniji se događa ugnježdeni paralelizam u dobijenom paralelizovanom kodu?

```
Fib(n)
1. if n ≤ 1
2.   return n
3. else x = Fib(n - 1)
4.   y = Fib(n - 2)
5.   return x + y
```

```
P-Fib(n)
1. if n ≤ 1
2.   return n
3. else x = spawn P-Fib(n - 1)
4.   y = P-Fib(n - 2)
5.   sync
6.   return x + y
```

Uvek kada spawn prethodi pozivu procedure se dogodi ugnježdeni paralelizam

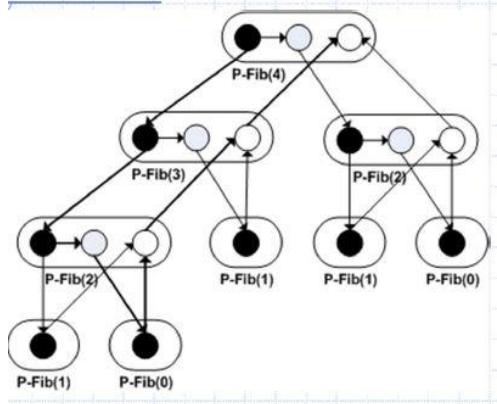
15. Šta je idealni paralelni računar?

a. Skup procesora i sekvencialno konzistentne deljene memorije.

Prepostavka je da su svi procesori iste snage i da je cena raspoređivanja zanemarljiva.

16. Koje su mere efikasnosti paralelnog algoritma?

- a. RAD – ukupno vreme računanja na jednom procesoru. Ako je vreme za svaki čvor 1, onda je RAD=broj čvorova.
- b. RASPON – najveće vreme potrebno da se izvrše linije duž bilo kog puta. Ako je vreme za svaki čvor 1, onda je RASPON=broj čvorova koji se nalaze na najdužoj putanji grafa (KRITIČNOJ PUTANJI).



Primer: graf Fib(4) ima 17 čvorova i 8 je na kritičnoj putanji, ako je vreme za svaki čvor 1, rad=17, raspon=8

17. Zakoni rada i raspona?

- a. ZAKON RADA: $T_p \geq T_1/P$
- b. ZAKON RASPONA: $T_p \geq T_\infty$

18. Ubrzanje na P procesora (linearno i savršeno linearno ubrzanje)?

- a. Ubrzanje na P procesora je odnos T_1 / T_p . (govori koliko puta je brže izvršenje na P procesora u odnosu na 1 procesor)
- b. LINEARNO UBRZANJE ako je $T_1 / T_p = \Theta(P)$.

- c. SAVRŠENO LINEARNO UBRZANJE ako je $T_1 / T_p = P$.

19. Šta je PARALELIZAM paralelnog algoritma?

- To je odnos T_1 / T_∞ .
- Posmatra se kao:
 - Srednja količina rada koja se može obaviti paralelno.
 - Gornja granica, tj. najveće moguće ubrzanje.
 - Ograničenje savršenog linearne ubrzanja (max broj procesora za savršeno linearne ubrzanje).
- PRIMER:
 - Ako je $rad=17$, a paralelizam=8, onda je $17/8=2.125$, što znači da se ne može ostvariti mnogo veće ubrzanje od 2x.

20. Šta je LABAVOST paralelizma?

- To je odnos $T_1 / (P^* T_\infty)$.
- Ako je labavost manja od 1, ne može se ostvariti savršeno linearne ubrzanje.
- Ako je labavost veća od 1, faktor ograničenja je rad po procesoru.

21. Šta radi rasporedivač?

- Direktno preslikava linije na procesore, tj. (u stvarnosti) preslikava linije na statičke niti, a OS raspoređuje niti na procesore.

22. Šta rade POHLEPNI RASPOREDIVAČI?

- Dodeljuju što je moguće više linija u svakom koraku.
- Korak može biti:
 - POTPUN: barem P linija $\rightarrow P$ linija na izvršenje
 - NEPOTPUN: manje od P linija \rightarrow sve linije na izvršenje.

23. Koje je minimalno T_p iz zakona rada, a koje iz zakona raspona?

- ZAKON RADA: $\min T_p = T_1 / P$.
- ZAKON RASPONA: $\min T_p = T_\infty$.

24. Kako glasi teorema o gornjoj granici T_p ?

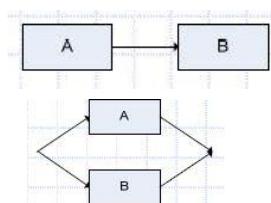
- Na P procesora, pohlepni rasporedivač izvršava paralelni algoritam sa radom T_1 i rasponom T_∞ u vremenu: $T_p \leq T_1 / P + T_\infty$.
- Postoje 2 posledice ove teoreme:
 - $T_p \leq 2 T_p^*$, gde je T_p^* optimalno vreme.
 - Ako je $P \ll T_1 / T_\infty$, onda je $T_p \approx T_1 / P$. (odносно, ubrzanje je približno jednako sa P)

25. Na čemu se zasniva analiza paralelnih algoritama?

- Zasniva se na: analizi rada (analiza SERIJALIZACIJE) i analizi raspona (analiza KRITIČNOG PUTA).

26. Koje slučajeve razlikuje analiza raspona?

- Ako su 2 podgrafovi povezani redno:
- Ako su 2 podgrafovi povezani paralelno:



$$T_\infty(A \cup B) = T_\infty(A) + T_\infty(B)$$

$$T_\infty(A \cup B) = \max(T_\infty(A), T_\infty(B))$$

27. Odrediti rad, raspon i paralelizam sledeće strukture date kodom:

```
Mat-Vec(A, x)
1.n = A.rows
2.neka je y novi vektor dužine n
3.parallel for i = 1 to n
4.    yi = 0
5.parallel for i = 1 to n
6.    for j = 1 to n
7.        yi = yi + aij xj
8.return y
```

Rad: $T_1(n) = \Theta(n^2)$

Raspon: $T_\infty(n) = \Theta(n)$

Paralelizam: $\Theta(n)$

28. Kada je paralelni algoritam DETERMINISTIČAN?

- a. Ako je njegovo ponašanje uvek isto za isti ulaz.

29. Kada se događa TRKA DO PODATAKA?

- a. Događa se između 2 logički paralelne instrukcije koje pristupaju istoj memorijskoj lokaciji i barem 1 od tih instrukcija upisuje u tu lokaciju.
- b. PRIMER: **Race-Example()**
1.x = 0
2.parallel for i = 1 to 2
3. x = x + 1
4.print x

30. Rešenja trke do podataka?

- a. Brave za međusobno isključivanje.
- b. Najbolje rešenje:
 - i. U konstrukciji sa parallel for, sve iteracije treba da budu nezavisne.
 - ii. Između spawn i sync, programski kod potomka treba da bude nezavisan od koda njegovog pretka.

31. Za datu strukturu, koja ima raspon $\Theta(lgn)$, i koja je pogrešno paralelizovana (unutrašnja for petlja je parallel for), napisati ispravnu verziju takvu da raspon i dalje bude $\Theta(lgn)$?

```
Mat-Vec-Wrong(A, x)
1.n = A.rows
2.neka je y novi vector dužine n
3.parallel for i = 1 to n
4.    yi = 0
5.parallel for i = 1 to n
6.    parallel for j = 1 to n
7.        yi = yi + aij xj
8.return y
```

32. Za datu strukturu odrediti rad, raspon i paralelizam?

Nakon toga, paralelizovati unutrašnju for petlju tako da paralelizam bude $\Theta(n^3 / lgn)$?

Obratiti pažnju na to da se prostom zamenom for sa parallel for dolazi do trke do podataka.

P-Square-Matrix-Multiply(A, x)

1. $n = A.\text{rows}$
2. neka je C nova matrica dimenzije $n \times n$
3. **parallel for** $i = 1$ to n
4. **parallel for** $j = 1$ to n
5. $c_{ij} = 0$
6. **for** $k = 1$ to n
7. $c_{ij} = c_{ij} + a_{ik} b_{kj}$
8. **return** C

Rad: $\Theta(n^3)$

Raspon: $\Theta(n)$

Paralelizam: $\Theta(n^2)$

33. Za datu strukturu odrediti rad, raspon i paralelizam?

P-Matrix-Multiply- Recursive(C, A, B)

1. $n = A.\text{rows}$
2. **if** $n == 1$
 3. $c_{11} = a_{11}b_{11}$
4. **else** neka je T nova matrica dimenzije $n \times n$
 5. podeli matrice A, B, C i T u podmatrice dimenzije $n/2 \times n/2$
 6. spawn P-Matrix-Multiply- Recursive(C_{11}, A_{11}, B_{11})
 7. spawn P-Matrix-Multiply- Recursive(C_{12}, A_{11}, B_{12})
 8. spawn P-Matrix-Multiply- Recursive(C_{21}, A_{21}, B_{11})
 9. spawn P-Matrix-Multiply- Recursive(C_{22}, A_{21}, B_{12})
 10. spawn P-Matrix-Multiply- Recursive(T_{11}, A_{12}, B_{21})
 11. spawn P-Matrix-Multiply- Recursive(T_{12}, A_{12}, B_{22})
 12. spawn P-Matrix-Multiply- Recursive(T_{21}, A_{22}, B_{21})
 13. P-Matrix-Multiply- Recursive(T_{22}, A_{22}, B_{22})
 14. **sync**
 15. **parallel for** $i = 1$ to n
 16. **parallel for** $j = 1$ to n
 17. $c_{ij} = c_{ij} + t_{ij}$

Rad: $\Theta(n^3)$ iz rekurencije $T_1(n) = 8T_1(n/2) + \Theta(n^2)$ (8 podela matričnog množenja, 2 paralelne petlje)

Raspon: $\Theta(\lg^2 n)$ iz $T_\infty(n) = T_\infty(n/2) + \Theta(\lg n)$ (ne može se primeniti Master metoda, radi se metodom zamene)

Paralelizam: $\Theta(n^3 / \lg^2 n)$

34. Za datu strukturu odrediti rad, raspon i paralelizam?

Merge-Sort-P(A, p, r)

1. **if** $p < r$
 2. $q = \lfloor (p+r)/2 \rfloor$
 3. spawn Merge-Sort-P(A, p, q)
 4. Merge-Sort-P($A, q+1, r$)
 5. **sync**
 6. Merge(A, p, q, r)

◆ Rad:

- Pošto je procedura Merge serijska, njen rad i raspon su oba $\Theta(n)$
- Rekurencija za rad $T_1(n)$ procedure Merge-Sort-P:
$$T_1(n) = 2 T_1(n/2) + \Theta(n) = \Theta(n \lg n)$$
- Rad je dakle isti kao $T(n)$ serijskog dvojnika

◆ Raspon:

- Kako se dva rekursivna poziva mogu izvršavati u ||:
$$T_\infty(n) = T_\infty(n/2) + \Theta(n) = \Theta(n)$$

◆ Paralelizam: $T_1(n) / T_\infty(n) = \Theta(\lg n)$

35. Za datu strukturu odrediti rad, raspon i paralelizam?

```

Binary-Search( $x, T, p, r$ )
1.  $low = p$ 
2.  $high = \max(p, r + 1)$ 
3. while  $low < high$ 
4.    $mid = \lfloor (low+high)/2 \rfloor$ 
5.   if  $x \leq T[mid]$ 
6.      $high = mid$ 
7.   else  $low = mid + 1$ 
8. return  $high$ 

```

Poziv procedure uzima $\Theta(\lg n)$ serijskog vremena u najgorem slučaju

- gde je $n = r - p + 1$ veličina podniza na kom se procedura izvršava

Pošto je Binary-Search serijska procedura, njen rad i raspon su u najgorem slučaju $\Theta(\lg n)$, ova

36. Za datu strukturu odrediti rad, raspon i paralelizam?

```

P-Merge( $T, p_1, r_1, p_2, r_2, A, p_3$ )
1.  $n_1 = r_1 - p_1 + 1$ 
2.  $n_2 = r_2 - p_2 + 1$ 
3. if  $n_1 < n_2$  // osiguraj da je  $n_1 \geq n_2$ 
4.   zameni  $p_1$  sa  $p_2$ 
5.   zameni  $r_1$  sa  $r_2$ 
6.   zameni  $n_1$  sa  $n_2$ 
7. if  $n_1 == 0$  // oba podniza prazna?
8.   return
9. else  $q_1 = \lfloor (p_1+r_1)/2 \rfloor$ 
10.   $q_2 = \text{Binary-Search}(T[q_1], T, p_2, r_2)$ 
11.   $q_3 = p_3 + (q_1 - p_1) + (q_2 - p_2)$ 
12.   $A[q_3] = T[q_1]$ 
13.  spawn P-Merge( $T, p_1, q_1 - 1, p_2, q_2 - 1, A, p_3$ )
14.  P-Merge( $T, q_1 + 1, r_1, q_2, r_2, A, q_3 + 1$ )
15.  sync

```

Rešenje:

Raspon:

- Dva podniza sadrže ukupno $n = n_1 + n_2$ elemenata
- Rekurzivni pozivi rade |||. Koji je skuplji?
- Ključ: max br. elemenata u bilo koja od ta dva rekurzivna poziva može biti najviše $3n/4$
 - Kako je $n_2 \leq n_1$, sledi: $n_2 = 2n_2/2 \leq (n_1+n_2)/2 = n/2$
 - U najgorem slučaju, jedan od dva rekurzivna poziva spaja $\lfloor n_1/2 \rfloor$ elemenata podniza $T[p_1..r_1]$ sa svih n_2 elemenata podniza $T[p_2..r_2]$, tj. ukupno spaja ovoliko elemenata: $\lfloor n_1/2 \rfloor + n_2 \leq n_1/2 + n_2/2 + n_2/2 = (n_1 + n_2)/2 + n_2/2 \leq n/2 + n/4 = 3n/4$

- Dodavanjem cene $\Theta(\lg n)$ za pozive procedure Binary-Search u liniji 10, dobija se sledeća rekurencija:

$$T_\infty(n) = T_\infty(3n/4) + \Theta(\lg n)$$
- Za osnovni slučaj raspon je $\Theta(1)$
 - Jer se linije 1-8 izvršavaju u konstantnom vremenu
- Rešenje je $T_\infty(n) = \Theta(\lg^2 n)$, što se metodom smene može lako proveriti

Rad:

- Pošto svaki od n elemenata mora biti iskopiran iz niza T u niz A , sledi da je $T_1(n) = \Omega(n)$
- Drugo, pokažimo da je $T_1(n) = O(n)$
- Bin. pretraga u liniji 10 uzima $\Theta(\lg n)$ vremena
 - što dominira nad drugim radom izvan rekurzivnih poziva
- Već pokazano: jedan rekurzivni poziv radi na najviše elemenata $3n/4$, pa je:

$$T_1(n) = T_1(\alpha n) + T_1((1 - \alpha)n) + \Theta(\lg n)$$
 - Parametar α leži u opsegu $1/4 \leq \alpha \leq 3/4$
- Sledi dokaz da je rešenje ove rekurentne jednačine $T_1(n) = O(n)$
 - Metod zamene: prepostavimo da je $T_1(n) \leq c_1 n - c_2 \lg n$ za neke pozitivne konstante c_1 i c_2 . Zamenom:
 - $$T_1(n) \leq (c_1 \alpha n - c_2 \lg(\alpha n)) + (c_1(1 - \alpha)n - c_2 \lg((1 - \alpha)n)) + \Theta(\lg n)$$
 - $= \dots = c_1 n - c_2 \lg n - (c_2(\lg n + \lg(\alpha(1 - \alpha))) - \Theta(\lg n))$
 - $\leq c_1 n - c_2 \lg n$
 - c_2 se izabere dovoljno veliko, tako da član $c_2(\lg n + \lg(\alpha(1 - \alpha)))$ dominira nad $\Theta(\lg n)$
 - c_1 se bira dovoljno veliko da se zadovolji rekurenciju
 - Znači $T_1(n) = \Theta(n)$, pa je paralelizam $T_1(n)/T_\infty(n) = \Theta(n/\lg^2 n)$.

37. Za datu strukturu odrediti rad, raspon i paralelizam?

```
P-Merge-Sort( $A, p, r, B, s$ )
1.  $n = r - p + 1$ 
2. if  $n == 1$ 
3.    $B[s] = A[p]$ 
4. else neka je  $T[1..n]$  novi niz
5.    $q = \lfloor (p+r)/2 \rfloor$ 
6.    $q' = (q - p) + 1$ 
7.   spawn P-Merge-Sort( $A, p, q, T, 1$ )
8.   P-Merge-Sort( $A, q+1, r, T, q'+1$ )
9. sync
10. P-Merge( $T, 1, q', q'+1, n, B, s$ )
```

Rešenje:

Rad:

- Pošto je rad procedure P-Merge $TPM_1(n) = \Theta(n)$, rad procedure P-Merge-Sort je:
$$T_1(n) = 2 T_1(n/2) + TPM_1(n)$$

$$= 2 T_1(n/2) + \Theta(n)$$
- Rešenje je $T_1(n) = \Theta(n \lg n)$, prema drugom slučaju master teoreme

Raspon:

- Pošto dva rekurzivna poziva rade logički u paraleli, jedan od njih se može ignorisati

- Pošto raspon procedure P-Merge iznosi $\Theta(\lg^2 n)$, raspon procedure P-Merge-Sort je:

$$\begin{aligned} T_\infty(n) &= T_\infty(n/2) + TPM_\infty(n) \\ &= T_\infty(n/2) + \Theta(\lg^2 n) \end{aligned}$$

- Master teorema ne može. Rešenje $T_\infty(n) = \Theta(\lg^3 n)$, se može proveriti metodom smene

Paralelizam:

- $T_1(n)/T_\infty(n) = \Theta(n \lg n)/\Theta(\lg^3 n) = \Theta(n / \lg^2 n)$

38. Koji su koraci paralelizacije programa?

- Dekompozicija
- Dodela (zadataka procesima)
- Orkestracija (komunikacije između procesa)
- Preslikavanje (mapiranje).

39. Nabrojati projektantske prostore?

- Pronalaženje paralelizma (izlaganje konkurentnih zadataka)
- Struktura algoritma (preslikavanje zadataka na procese, radi korišćenja paralelnih arhitektura)
- Pomoćne strukture (šabloni koda i struktura podataka)
- Izvedbeni mehanizmi (mehanizmi niskog nivoa, koriste se za pisanje paralelnih programa).

1 i 2 predstavljaju IZRAŽAVANJE ALGORITMA, a 3 i 4 predstavljaju KONSTRUISANJE PROGRAMA.

40. Kako glasi Bernštajnov uslov za paralelizaciju?

- Dva zadatka, T1 i T2, mogu biti paralelni ako:
 - Ulaz T1 nije deo izlaza T2.
 - Ulaz T2 nije deo izlaza T1.
 - Ako se izlazi T1 i T2 ne preklapaju.

- b. Primer: da li zadaci $a=x+y$ i $b=x+z$ mogu biti paralelni?

R - ulaz, W – izlaz.

$R1=\{x, y\}$, $W1=\{a\}$

$R2=\{x, z\}$, $W2=\{b\}$

$R1$ u preseku sa $W2$ daje prazan skup. (ulaz $T1$ nije deo izlaza $T2$)

$R2$ u preseku sa $W1$ daje prazan skup. (ulaz $T2$ nije deo izlaza $T1$)

$W1$ u preseku sa $W2$ daje prazan skup. (izlazi $T1$ i $T2$ se ne preklapaju)

Ovi zadaci mogu biti paralelni, po Bernštajnovom uslovu.

41. Koji tipovi organizovanja programa postoje?

- a. Organizacija po zadacima.
 - i. Rekurzivni program – podeli i zavladaj.
 - ii. Nerekurzivni program – paralelizam zadataka.
- b. Organizacija po podacima.
 - i. Nizovi i linearne strukture podataka – geometrijska dekompozicija.
 - ii. Rekurzivne strukture podataka – rekurzivni podaci.
- c. Organizacija po toku podataka.
 - i. Regularan, jednosmeran, uglavnom stabilan tok – protočna obrada.
 - ii. Iregularan, dinamički, nepredvidivi tok podataka – koordinacija na bazi događaja.

42. Podržavajuće strukture algoritama u paralelnom programiranju?

- a. SMPD
- b. Paralelizam petlje
- c. Vodeći / Radnik
- d. Grananje / Pridruživanje.

43. Šta predstavlja cilk_spawn?

- a. Mrešćenje.

44. Šta predstavlja cilk_sync?

- a. Sinhronizaciju.

45. Šta je nastavak?

- a. Linija koja započinje neposredno nakon tačke mrešćenja.

46. Šta je potomak?

- a. Niz operacija u iskazu mrešćenja.

47. Da li se potomak i nastavak mogu izvršavati u paraleli?

- a. Rasporedivač ih može paralelno izvršavati.

48. Šta je predak?

- a. Predak je Cilk blok koji sadrži početnu liniju, iskaze mrešćenja i njihove nastavke, bez potomaka.

49. Šta označava Sync iskaz u Cilk bloku?

- a. Da se svi potomci moraju završiti pre nastavka izvršenja.
- b. Ako nema potomaka u trenutku sync-a, on nema nikakvog efekta.
- c. Nova linija koja izlazi iz sync-a može biti paralelna sa pretkom i rođacima (drugim potomcima pretka).

50. Ako se spawn pojavi u try bloku, da li postoji implicitni sync?

- a. Da, na kraju tog try bloka.

51. Šta definiše tip reduktora?

- Callback operaciju reduce:
 - Koja spaja dva POGLEDA svojstveno reduktoru.
 - reduce(V1, V2) se označava kao $V1 \circ V2$.
 - Klasičan reduce je asocijativan, tj. važi $(a \circ b) \circ c = a \circ (b \circ c)$.
- Callback operaciju identity:
 - Koja inicijalizuje novi POGLED, pogled I.
 - $I \circ v = v$ i $v \circ I = v$, za svako v tipa `value_type`.
- Trojka $(\circ, I, \text{value_type})$ opisuje matematički monoid.

52. Primeri monoida?

- $(\text{int}, +, 0)$.
- $(\text{list}, \text{concatenate}, \text{empty})$.

53. Šta radi operator sekcije?

- Bira više elemenata niza za paralelnu operaciju. (npr. $A[1:5:2][:]$, prvi indeks 1, broj elemenata 5, korak 2)

54. Primeri operacija preko operatora sekcije?

Dodata

- Paralelna operacija za sve elemente se leve strane
- Npr. $A[4:3] = A[3:3];$
 - $A[3], A[4], A[5]$ se kopiraju u $A[4], A[5], A[6]$
 - Kompajler obezbeđuje dodatni prostor tako da upisi u $A[4]$ i $A[5]$ ne ometaju čitanja iz istih lokacija
- Još primera:
 - // Copy elements 10->19 in A to elements 0->9 in B.
 $B[0:10] = A[10:10];$
 - // Error. Triplets 0:10 and 0:100 are not the same size.
 $B[0:10] = A[0:100];$

```
type fn(type in1, type in2); // declaration of scalar reduction function
type in[N], out; // array input and scalar output result

// accumulate successive elements in in with a user function fn,
// resulting in a single value out
out = __sec_reduce(fm, identity_value, in[x:y:z]);

out = __sec_reduce_add(in[x:y:z]); // out = sum of all values
out = __sec_reduce_mul(in[x:y:z]); // out = product of all values
out = __sec_reduce_all_zero(in[x:y:z]); // 1 if all values are zero
out = __sec_reduce_all_nonzero(in[x:y:z]); // 1 if all values nonzero
out = __sec_reduce_any_nonzero(in[x:y:z]); // 1 if any nonzero
out = __sec_reduce_max(in[x:y:z]); // max value of values in in
out = __sec_reduce_min(in[x:y:z]); // min value of values in in
out = __sec_reduce_max_index(in[x:y:z]); // index of maximum value
out = __sec_reduce_min_index(in[x:y:z]); // index of minimum value
```

// Set all elements of A to 1.0.

$A[:] = 1.0;$

// Element-wise addition of all elements in A and B, result in C.

$C[:] = A[:] + B[:];$

// Matrix addition of the 2x2 matrices in A and B starting at

// $A[3][3]$ and $B[5][5]$.

$C[0:2][0:2] = A[3:2][3:2] + B[5:2][5:2];$

// ??? – za domaći

$C[0:9][0][0:9] = A[0][0:9][0:9] + B[0:9][0:9][4];$

◆ Skupljanje: elementi iz `in[]` specifirani sa `index[x:y:z]` skupljaju se u `out[a:b:c]`

◆ Razbacivanje: elemente iz `in[a:b:c]` razbacuju u `out[]` po rasporedu `index[x:y:z]`

```
unsigned int index[N];
type out[M], in[O];
```

// gather elements from in[], given by index[x:y:z], into out[]
out[a:b:c] = in[index[x:y:z]];

// scatter elements from in[] into various locations in out[],
// given by index[x:y:z]
out[index[x:y:z]] = in[a:b:c];

55. Koje su klauzule SIMD pragme?

◆ Klauzule SIMD pragme

- `vectorlength(num1, num2, ..., numN)`
 - ◆ Izaberi jednu od datih dužina
- `private(var1, var2, ..., varN)`
 - ◆ Privatne promenljive za svaku iteraciju petlje
- `linear(var1:step1, var2:step2, ..., varN:stepN)`
 - ◆ Za svaku iteraciju *var* se menja za zadati *step*
- `reduction(operator:var1, var2,..., varN)`
 - ◆ Primeni redukciju tipa *operator* na zadata promenljive
- `[no]assert`
 - ◆ (ne)reaguj ako se ne može generisati vektorski kod

56. Kako se `for(i=first, i<last, i+=step) {f(i)}` može zapisati preko parallel_for petlje?

- a. Kao parallel_for(first, last, step, f), u slučaju da se f(i) mogu pozivati paralelno.
- Step je opcioni element.

57. Koliko traje jedna grainsize iteracija?

- a. 100.000 ciklusa.

58. Transformisati datu nelinearnu protočnu obradu u linearu protočnu obradu?



59. Šta omogućuju STL kontejneri?

- a. Istovremen pristup za više niti.

60. Šta je HashCompare tip?

To je tip koji određuje:

- i. Kako se računa ključ.
- ii. Kako se 2 ključa porede.

61. Pomoću čega se realizuje međusobno isključivanje niti?

- a. Pomoću mutex.
 - i. Mutex je objekat koji nit može zaključati prethodno dobijenim ključem.
 - ii. Samo jedna nit može imati ključ, ostale niti moraju čekati.
 - iii. Najjednostavniji je spin_mutex.
- b. Pomoću locks.

62. Kada nastaje međusobno blokiranje (deadlock) ?

- a. Ako postoji krug niti.
- b. Ako svaka nit drži bar jedan ključ i čeka na mutex koji je zaključala druga nit.
- c. Ako nijedna nit ne želi da oslobodi svoje ključeve.

63. Šta je paralelni kod?

- a. To je deo koda koji se izvršava paralelno. (u njemu se izvršava više niti)

64. Šta je paralelno mesto?

- a. To je deo koda koji sadrži jedan ili više zadataka, koji se izvršavaju paralelno. (obično se nalazi iznad vruće tačke, tj. u grafu poziva)

65. Šta je sinhronizacija?

- a. To je koordinacija izvršenja više programskih niti.

66. Šta je cilj (target) ?

- a. To je izvršna (.exe) datoteka.

67. Šta je zadatak (task) ?

- a. To je deo koda, sa njegovim podacima, koji se može izvršavati u niti, tj. na jezgru CPU-a.

68. Šta je Total Time?

- a. Procena vremena izvršenja iskaza ili funkcije koja se poziva iz iskaza.

69. Šta je Loop Time?

- a. Sumarno Total Time za sve osnovne blokove u petlji.
- b. Prikazuje se jednom za celu petlju.

70. Šta spada u metrike?

- a. Elapsed time – proteklo vreme.
- b. CPU time – ukupno iskorišćeno vreme za sve niti.
- c. Unused CPU time – ukupno neiskorišćeno vreme za sva jezgra.
- d. Core Count – broj jezgara (logičkih CPU).
- e. Threads Created – broj izvršenih niti.

71. Čemu je jednak ciljni paralelizam (Target Concurrency) ?

- a. Jednak je broju jezgara u CPU. (npr. za Double Core je TC=2)

72. Šta je prosečna upotreba procesora (AU) ?

- a. To je odnos $AU = T_{cpu} / T_e$.
 - i. AU – prosečna upotreba CPU.
 - ii. T_{cpu} – vreme rada svih jezgara.
 - iii. T_e – ukupno proteklo vreme.
- b. AU idealno treba da teži broju jezgara CPU.

73. Šta je Wait Time?

- a. To je ukupno vreme čekanja niti radi sinhronizacije ili završetka U/I radnje.

74. Šta je asembler?

- a. Asembler je program koji prevodi polazni (izvorni) program napisan na asemblerском jeziku u izvršivi mašinski program.

75. Da li definicija simbola mora prethoditi njihovoj upotrebni?

- a. Ne mora.

76. Šta se događa u PRVOM, a šta u DRUGOM prolazu asemblera?

- a. PRVI:
 - i. Definisanje vrednosti simbola koji je pojavljuju u programu.
 - ii. Dodela memorijskih lokacija svim instrukcijama i literalima iz programa.
- b. DRUGI:
 - i. Prevođenje u mašinske instrukcije.
 - ii. Određivanje vrednosti operanada izračunavanjem izraza koji predstavljaju vrednost operanada u instrukciji.

77. Šta je definicija simbola?

- a. To je proces pridruživanja simbola sa brojem koji predstavlja njegovu vrednost.

78. Za šta služi tabela simbola?

- a. Za upisivanje simbola i njima pridruženih vrednosti.

79. Šta su literali?

Šta sadrži tabela literala?

- a. Literali su specijalna vrsta simbola koji odnose se na konstante.
- b. Tabela literala sadrži adresu i binarni kod konstante koja odgovara datom literalu.

80. Šta sadrži tabela koda operacije i koja je njena uloga?

- a. Ona je ugrađena u asembler.
- b. Sadrži po jednu vrstu za svaki simbolički kod operacije.
- c. Njena uloga je u definisanju razlike između pseudo i mašinskih operacija.

81. Nabrojati osnovne module asemblera?

- a. Modul za I prolaz.
- b. Modul za II prolaz.
- c. Modul za izračunavanje izraza.
 - i. Poziva se u I prolazu za obradu ESQ.
 - ii. Poziva se u II prolazu za određivanje vrednosti operanada.
- d. Modul za konverziju konstanti.
 - i. Poziva se u I prolazu za obradu literala.
 - ii. Poziva se u II prolazu za obradu α : DATA β i literala.

82. Opisati tok informacija u I i II prolazu asemblera (ulazi i izlazi I i II prolaza asemblera) ?

- a. I prolaz:
 - i. Ulaz: polazni program napisan na asemblerском jeziku.
 - ii. Izlaz: tabela simbola, tabela literala, kopija polaznog programa, ostale informacije.
- b. II prolaz:
 - i. Ulaz: tabela simbola, tabela literala, kopija polaznog programa, ostale informacije.
 - ii. Izlaz: izveštaj o prevođenju, datoteka prevedenog programa.

83. Asemblerirati dati program?

	TITLE	PRVI	TKOP	
BROJ	DB	5.48	MNEMONIK	KOD
	MOV	AX, CS: BROJ	MOV ...	2E A1
	CMP	AX, BROJ+1	CMP ...	2E 3B 06
	JNE	DVA	JNE ...	75
DVA	HLT		HLT	F4
	END			

Rešenje:

	TITLE	PRVI	MAŠINSKI PROGRAM		TS	
			LOKACIJA	SADRŽAJ	SIMBOL	VREDNOST
BROJ	DB	5.48	0000	05	BROJ	0000
	MOV	AX, CS: BROJ	0001	30	DVA	000D
	CMP	AX, BROJ+1	0002	2E		
	JNE	DVA	0003	A1		
DVA	HLT		0004	00		
	END		0005	00		
			0006	2E		
			0007	3B		
			0008	06		
			0009	01		
			000A	00		
			000B	75		
			000C	00		
			000D	F4		

TKOP	
MNEMONIK	KOD
MOV ...	2E A1
CMP ...	2E 3B 06
JNE ...	75
HLT	F4

84. Na šta ukazuje brojač lokacija?

- a. Na prvi nedodeljen bajt u memoriji.
- b. Uvećava za različite vrednosti, zavisno od iskaza:
 - i. MAŠINSKA INSTRUKCIJA: za dužinu instrukcije.
 - ii. PSEUDO INSTRUKCIJA α : DATA β : za dužinu podataka koje generiše izraz β .
 - iii. PSEUDO INSTRUKCIJA α : BLOK p : za dužinu bloka koji se rezerviše, tj. p.
 - iv. PSEUDO INSTRUKCIJA KOJA NE GENERIŠE IZLAZ (npr. EQU) : za 0.

85. Šta podrazumeva definisanje simbola?

- a. Unos simbola i brojača lokacije u tabelu simbola.

86. Šta podrazumeva obrada literala?

- a. Više literalala može generisati istu konstantu (u tom slučaju, pravi se samo jedna instanca).
- b. Literal se konvertuje u binarnu vrednost.
- c. Prostor za literale se dodeljuje na kraju I prolaza asemblera.

87. Šta sadrži izveštaj o prevodenju (listing) ?

- a. Redni broj linije i ukazivač greške.
- b. Adresu memorijske lokacije.
- c. Generisani mašinski kod i podatke.
- d. Izvorni asemblerski kod.

88. Šta je makroasembler?

- a. Program koji prevodi polazni program napisan na makroasemblerском jeziku u izvršivi mašinski program.

89. Koja su proširenja potrebna asembleru da bi mogao da obradi makroinstrukciju?

- a. Prvo: prepoznavanje makroinstrukcije.
- b. Drugo: proširenje makroinstrukcije.

Da bi ovo mogao da uradi, asembler prvo mora da pronađe i sačuva makrodefiniciju, koja odgovara datoj makroinstrukciji.
Svaka makrodefinicija započinje MACRO, a završava ENDM pseudo instrukcijom.

90. Koji je najjednostavniji način prepoznavanja makroinstrukcija?

- a. Dodavanje vrste u tabelu koda operacije.
- b. Ova vrsta sadrži naziv makroinstrukcije i ukazivač na odgovarajuću makrodefiniciju, koja se nalazi u tabeli makrodefinicija.

91. Šta sadrži jedna vrsta tabele koda operacije?

- a. Simbolički kod operacije.
- b. Tip simboličkog koda operacije.
- c. Odgovarajući numerički kod.
- d. Tip operanada ovog koda.

92. Šta je zadatak liste naziva parametara?

- a. Ona uspostavlja vezu između stvarnih i fiktivnih parametara makroinstrukcije.
- b. PRIMER: instrukcija ADDUP DATA1, DATA2, DATA3.
 - i. Stvarni parametri: DATA1, DATA2, DATA3.
 - ii. Fiktivni parametri: P1, P2, P3. (jer imamo oblik ADDUP(P1, P2, P3)).

93. Šta su kompjaler, procedura i algoritam?

- a. Kompajler – prevodilac sa višeg programskog jezika na mašinski jezik.

- b. Procedura – skup operacija koje se izvršavaju u konačnom vremenu i daju jednoznačan rezultat.
- c. Algoritam – skup operacija i skup pravila o njihovoj primeni nad polaznim podacima u cilju dobijanja rezultata.

94. Šta je formalni sistem?

- a. Neinterpretativan matematički sistem, sačinjen od azbuke, aksioma i pravila skupa zaključivanja.
- b. Formalni sistem je meta-jezik za definisanje programskih jezika.
 - i. Simboli jezika objekta = terminalni simboli.
 - ii. Simboli meta-jezika = netermininalni simboli.
- c. Formalni sistem definiše oblik, tj. sintaksu programskog jezika.

95. Šta je sintaksa, a šta semantika nekog jezika?

- a. Sintaksa je skup pravila kojima se pokoravaju relacije između važećih reči jezika.
- b. Semantika je skup sadržaja (značenja) nekog jezika.

96. Šta su azbuka, rečnik i jezik formalnog sistema?

- a. AZBUKA T je konačan skup terminalnih simbola.
- b. REČNIK T^* formiran u azbuci T je skup svih konačnih reči azbuke T .
- c. JEZIK L je podskup rečnika T^* , definisan je pomoću jednog ili više znakova smene. (pri čemu smena zadatu reč zamjenjuje jednom od reči u koju se ona transformiše)
- d. Rečenica se dobija pripajanjem simbola.

97. Šta je skup posledica formalnog sistema?

Šta je teza formalnog sistema?

Kada je formalni sistem odlučiv?

- a. Skup posledica je skup reči koji se dobija primenom pravila zaključivanja nad skupom reči jezika.
- b. Teza formalnog sistema je skup posledica izvedenih iz sopstvenih aksioma.
- c. Formalni sistem je odlučiv ako se može odrediti da li je zadata reč element teze sistema. (ukoliko postoji algoritam za ovakvo određivanje)

98. Kako se deli specifikacija jezika?

- a. GENERATIVNA: ako je neku rečenicu moguće generisati kada god je to potrebno.
- b. ANALITIČKA: ako algoritam određuje da li je zadata reč važeća ili nevažeća u jeziku.

99. Šta je formalna gramatika?

- a. Skup od 4 elemenata $G(N, T, \Sigma, P)$:
 - i. N – skup neterminalnih simbola.
 - ii. T – skup terminalnih simbola.
 - iii. Σ - početni / polazni simbol.
 - iv. P – skup smene.
- b. Ograničenje formalne gramatike je to što su N i T disjunktni skupovi.

100. Šta je rečenična forma?

Šta je rečenica?

Kako je jezik L definisan gramatikom G ?

- a. Rečenična forma je bilo koja reč koja može da se razvije od polaznog simbola Σ .
- b. Rečenica je rečenična forma koja sadrži samo terminalne simbole.
- c. Jezik L je definisan gramatikom G obrascem: $L(G)=\{W \text{ je element rečnika } T^*, \text{ takav da se indirektno razvija iz } \Sigma\}$.

101. Koji skupovi su potrebni za definisanje sintakse jezika?

Koji skupovi su potrebni za pisanje kompjajlera?

Šta je ANALIZA, tj. PARSING?

- a. Sintaksa jezika se definiše pisanjem skupa generacionih pravila, tj. PRODUKCIJA.
- b. Za pisanje kompjajlera, potrebni su skup generacionih pravila (produkcije) i skup pravila prepoznavanja.
- c. Analiza ili parsing jeste primena pravila prepoznavanja.

102. Šta je Bakus-Naurova forma?

- BNF je notacija za izražavanje produkcije, pri čemu svaka produkcija definiše sintaksnu klasu.
- Osnovne oznake BNF:
 - $::=$ predstavlja strelicu smene.
 - $<>$ ogradjuju ime sintaksne klase.
 - $|$ tj. simbol ILI omogućava dodelu više smena.

103. Šta je zadatak semantičkog opisa?

- Davanje smisla rečenicama koje se formiraju programskim jezikom.

104. Nabrojati i opisati komponente PREVODIOCA?

- PREPROCESOR: prelaz sa spoljnog oblika jezika na osnovni jezik.
- KOMPAJLER: prelaz sa osnovnog programskog jezika na medujezik.
- INTERPRETER: simulira pseudomašinu na realnoj mašini.

105. Objasnjenje sintakse pravolinijskog jezika:

$Stm \rightarrow Stm ; Stm$	(CompoundStm)	$ExpList \rightarrow Exp , ExpList$	(PairExpList)
$Stm \rightarrow id := Exp$	(AssignStm)	$ExpList \rightarrow Exp$	(LastExpList)
$Stm \rightarrow \text{print} (ExpList)$	(Printstm)	$Binop \rightarrow +$	(Plus)
$Exp \rightarrow id$	(IdExp)	$Binop \rightarrow -$	(Minus)
$Exp \rightarrow \text{num}$	(NumExp)	$Binop \rightarrow \times$	(Times)
$Exp \rightarrow Exp Binop Exp$	(OpExp)	$Binop \rightarrow /$	(Div)
$Exp \rightarrow (Stm , Exp)$	(EseqExp)		

Svaki **Stm** je iskaz. Svaki **Exp** je izraz.
 Iskaz *s1* ; *s2* izvršava napre *s1*, zatim *s2*.
 Iskaz *i* := *e* izračunava izraz *e*, i njegovu vrednost smešta u promenljivu *i*.
 print(*e1*, *e2*, ..., *en*) prikazuje vrednosti izraza, izračunatih s leva u desno, razdvojeno praznim mestima i prelaskom u novu liniju na kraju.

106. Šta sadrži svaki struct (iz lekcije Medujezik) ?

- Polje kind – enum različitih varijanti (jedna po svakom pravilu).
- Polje u – unija.

107. Pravila imenovanja?

- Typedef imena iza prefiksa počinju malim slovima.
- Imena konstruktora iza prefiksa počinju velikim slovima.
- Enumeracioni atomi iza prefiksa počinju malim slovima.
- Varijante unija (nemaju prefiks) počinju malim slovima.

108. Navesti principe modularnosti?

- Svaka faza kompjlera predstavlja poseban modul.
- Svaki modul ima jedinstven prefiks.
- Sve funkcije treba da imaju prototipove.
- Svi moduli uključuju util.h .

- e. Tip string je string u malloc-dodeljenoj memoriji.
- f. C funkcija malloc vraća NULL ako nema memorije.
- g. Školska realizacija ne koristi FREE.

109. Nabrojati 5 osnovnih faza kompjlera?

- a. Leksička analiza
- b. Sintaksna analiza (prepoznavanje strukture programa)
- c. Optimizacija koda
- d. Generisanje koda (prevodenje strukture programa u mašinski jezik)
- e. Asembliranje.

110. Koji su zadaci leksičke analize?

- a. Raščlanjivanje izvornog programa (niz ASCII znakova) na niz osnovnih simbola (tokena) višeg programskog jezika.
- b. Formiranje tabele literalna i tabele identifikatora.
- c. Formiranje tabele uniformnih simbola.

111. Šta sadrže vrste: tabele terminalnih simbola?

tabele literalna?

tabele identifikatora?

tabele uniformnih simbola?

- a. TABELA TERMINALNIH SIMBOLA: njena vrsta sadrži simbol, ukazivač i element za definiciju prethođenja.
- b. TABELA LITERALA: njena vrsta sadrži literal, skalu, osnovu, preciznost, adresu i drugu informaciju.
- c. TABELA IDENTIFIKATORA: njena vrsta sadrži naziv identifikatora, atribute podataka i adresu.
- d. TABELA UNIFORMNIH SIMBOLA: njena vrsta sadrži početnu adresu tabele simbola i indeks unutar tabele.

112. Za sledeći kod popuniti tabelu literalna i tabelu identifikatora?

```
main () {
    int x, y;
    scanf ("%d", &x);
    y = 10 * x;
    printf ("%d", y);
}
```

Rešenje:

TABELA LITERALA	
IND	LITERAL
1	" %d "
2	10

TABELA IDENTIFIKATORA	
IND	IDENTIFIKATOR
1	main
2	x
3	y
4	scanf
5	printf

113. Šta su zadaci sintaksne analize?

- a. Omogućuje kompjleru prepoznavanje sintaksnih konstrukcija (rečenica) i interpretaciju njihovih značenja.
- b. Ukazuje na uočene greške i omogućuje kompjleru nastavak traženja drugih grešaka.

114. Šta su redukcije?

- a. Redukcije su pravila koja definišu osnovne sintaksne konstrukcije i odgovarajuće rutine prevodenja (rutine akcija).

115. Koji prelazni oblici polaznog programa postoje?

- a. Stablo sintaksne analize.
- b. Matrični prelazni oblik.

116. Šta je zadatak optimizacije?

- a. Radi nad prelaznim oblikom programa.
- b. Postoje mašinski zavisne i mašinski nezavisne optimizacije.
- c. Mašinski zavisne: uklanjanje viška LOAD i STORE, JUMP na JUMP, itd.
- d. Mašinski nezavisne: eliminacija zajedničkih podizraza, prethodno izračunavanje vrednosti, uočavanje invarijantnog računanja unutar ciklusa.

117. Šta podrazumeva pridruživanje memorije?

- a. Dodelu memorije svim promenljivama, literalima i svim privremenim lokacijama potrebnim za međurezultate.
- b. Obezbeđuje inicijalizaciju odgovarajućih lokacija.
- c. Algoritam dodele memorije koristi tabelu identifikatora.

118. Primeri smena za operacije sabiranja, oduzimanja i množenja:

SUBM	MACRO	MI, OPRND1, OPRND2
	MOV	AX, OPRND1
	SUB	AX, OPRND2
	MOV	MI, AX
	ENDM	
ADDM	MACRO	MI, OPRND1, OPRND2
	MOV	AX, OPRND1
	ADD	AX, OPRND2
	MOV	MI, AX
	ENDM	

MULM	MACRO	MI, OPRND1, OPRND2
	MOV	AX, OPRND1
	MUL	AX, OPRND2
	MOV	MI, AX
	ENDM	
ASNM	MACRO	OPRND1, OPRND2
	MOV	AX, OPRND2
	MOV	OPRND1, AX
	ENDM	

119. Kojih 7 faza školskog kompjlera postoji?

- a. Leksička analiza
- b. Sintaksna analiza
- c. Interpretacija
- d. Optimizacija (mašinski nezavisna)
- e. Dodata memorije
- f. Selekcija koda
- g. Asemblerски izlaz.

120. Šta je IDE?

- a. To je integriran skup sistemskih programa koji se koristi za komforan i pouzdan razvoj programske podrške.
- b. IDE je skraćenica za Integrисано Razvoјно Okruženje.

121. Nabrojati faze protočne obrade DSP MASx?

- a. X, Y, Z, A, B, C, D, E, F.

Primer instrukcije:

$$fc0 = dc0 + ac1 * z10$$

Iz instrukcije se vidi da se podatak iz memorije uzima u fazi Z (oznaka z10), podatak iz akumulatora 1 u narednoj fazi A (oznaka ac1), podatak iz akumulatora 0 u fazi D (oznaka dc0), a rezultat se upisuje u akumulator 0 u zadnjoj fazi F (oznaka fc0).

Instrukcija	Faza protočne obrade
Skok	D
Prva iza skoka	C
Druga iza skoka	B
Treća iza skoka	A
Četvrta iza skoka	Z
Peta iza skoka	Y
Šesta iza skoka	X

122. Šta je zadatak GUI?

Šta je zadatak rukovaoca projektom?

Šta je zadatak editora?

- a. GUI obezbeđuje grafičku spregu za korisnika IDE.
- b. Rukovalac projektom omogućava definisanje projekta programske podrške koja se razvija.
- c. Editor je uslužni program za uređivanje teksta, i on pruža klasične usluge ekranskog uređivanja.

123. Šta omogućuju C kompjajler i asembler?

Šta sadrže matematička i UI biblioteka?

- a. C kompjajler i asembler omogućuju prevođenje programskih modula.
Prevedeni mašinski kod se upisuje u objektnu datoteku u MOF obliku.
- b. Matematička i UI biblioteka sadrže standardne potprograme za realizaciju matematičkih funkcija i U/I radnji.

124. Koji je zadatak povezivača?

- a. Povezivanje objektnih datoteka pojedinih modula u jedan izvršivi program u tzv. MOB obliku.

125. Koji je zadatak kompaktora?

- a. Kompaktor ima zadatak da primenom različitih tehnika kompaktovanja preuređuje kod, i na taj način neke NOP i korisne instrukcije učini suvišnim, a zatim i da ih eliminiše.

126. Šta je dibager?

- a. Uslužni program za kontrolisano izvršenje programa koji se razvija, omogućava njegovo kontrolisano izvršenje na simulatoru procesora.

127. Šta je MASx simulator?

- a. Programska komponenta koja modelira fizičke komponente arhitekture, na kojoj se izvršava optimizovan i povezan program koji je predmet ispitivanja.

128. Šta čini sadržaj projekta?

- a. Datoteke izvornog koda.
- b. Biblioteke potrebne za povezivanje.
- c. Argumenti komandi koji se koriste prilikom poziva kompjajlера, povezivača i kompaktora.
- d. Nazivi promenljivih čija vrednost se posmatra u bloku za prikaz vrednosti izraza. (watchpoint)
- e. Brojevi linija prekida. (breakpoint)

129. Koji su koraci metodologije razvoja programa?

- a. MATLAB simulacija.
- b. C referentni kod.
- c. MAS-like C kod.
- d. MAS kod.
- e. Provera usaglašenosti na nivou bita na zadatim test vektorima.

130. Koji su moduli (faze) prednjeg, a koji zadnjeg dela kompjajlera?

- a. PREDNJI DEO KOMPAJLERA:
 - i. Preprocesor.
 - ii. Leksička analiza.
 - iii. Sintaksna analiza.
 - iv. Prevođenje u međukod.
 - v. Prevođenje u kanonički oblik.
 - vi. Izbor instrukcija.
 - vii. Mašinski zavisna optimizacija.
- b. ZADNJI DEO KOMPAJLERA:
 - i. Analiza toka upravljanja.
 - ii. Analiza toka podataka.
 - iii. Dodela resursa.
 - iv. Prilagođavanje koda protočnoj strukturi.
 - v. Generisanje koda.
 - vi. Povezivanje.

131. Kako se drugačije naziva leksički analizator?

- a. Leksički analizator ili SKENER.

132. Kako se drugačije naziva sintaksni analizator?

- a. Sintaksni analizator ili PARSER.

133. Šta je zadatak sintaksnog analizatora?

- a. Zadužen je za proveru redosleda terminalnih simbola (leksema) u izvornom kodu programa.
- b. Formira stablo sintaksne analize.

134. Koji tipovi objekta postoje u tabeli simbola?

- ◆ *CSymTabEntry*, to je osnovna klasa iz koje se izvode sve klase objekata tabele simbola,
- ◆ *CVarEntry*, predstavlja promenljivu,
- ◆ *CArrayEntry*, predstavlja vektor,
- ◆ *CFuncEntry*, predstavlja funkciju,
- ◆ *CTypeEntry*, predstavlja tip podataka,
- ◆ *CLabelEntry*, predstavlja labelu,
- ◆ *CConstEntry*, predstavlja konstantu,
- ◆ *CTypeDefEntry*, predstavlja definiciju korisničkog tipa, i
- ◆ *CTypeEnumEntry*, predstavlja jednu moguću vrednost prebrojivog tipa (enum).

135. Šta je stablo sintaksne analize?

- a. Predstavlja sve iskaze i izraze polaznog programa.
- b. Čvorovi su instance sintaksnih klasa (tj. klasa apstraktne sintakse).
- c. Apstraktna sintaksa se deli na 2 grupe: iskaze i izraze.

136. Šta je primarni, a šta sekundarni zadatak modula kompjajlera za određivanje tipova?

- a. PRIMARNI ZADATAK: određivanje tipova svih izraza iz polaznog programa.
- b. SEKUNDARNI ZADATAK: u slučaju izraza sa konstantama, nakon što se odredi tip izraza, određuje se i njegova vrednost.

137. Šta podrazumeva statistička provera semantike?

- a. Proveru tipova svih operanada u izrazima.
- b. Proveru broja parametara kod poziva funkcije. (mora biti jednak broj parametara u definiciji funkcije)
- c. Dinamička provera semantike se obavlja u toku izvršenja programa.

138. Koja su 2 oprečna zahteva kod analize pokazivača?

- a. Radi izbegavanja potrebe za dinamičkim dereferenciranjem pokazivača, treba sve promenljive, na koje pokazivač može pokazati, smestiti u istu memoriju zonu (D0 ili D1).
- b. U cilju dobijanja optimalnog koda za binarne operacije, treba podatke (operande) smestiti u različite memorije zone.

139. Koja su ograničenja smeštanja u analizi pokazivača?

- a. Analiza pokazivača se obavlja nad stablom sintaksnog analizatora.
- b. Izlaz je niz ograničenja za smeštanje promenljivih. (smeštanje u istu zonu – znak =, smeštanje u različite zone – znak !=)

140. Objasnenje analize iskaza dodele vrednosti u analizi pokazivača?

- 0 nema indirekcije, tj. u pokazivač se upisuje adresa
- 1 jedan nivo indirekcije, tj. vrednost s desne strane operatora dodele vrednosti se upisuje na adresu na koju pokazuje pokazivač, itd.

$*p = *(q + 4 * i);$
rezultuje u relaciji $(p,1)-(q,1)$

Za iskaz $p=q$ se dobija relacija $(p,0)-(q,0)$

- označava da pokazivač p može pokazivati na sve promenljive na koje može pokazivati q (p i q su tzv. aliasi)

Za iskaz $p = &q$ se dobija relacija $(p,0)-(q,-1)$

- gde nivo indirekcije -1 odgovara operaciji uzimanja adrese.

141. Objasnjenje analize skupova promenljivih u analizi pokazivača?

- Nakon što se skupe sve relacije, utvrđuju se skupovi promenljivih na koje pokazuju pojedini pokazivači.
Za svaki ovakav skup se generišu ograničenja smeštanja u istu zonu podataka.
- PRIMER: analizom relacija $(p,0)-(a,-1)$ i $(p,0)-(b,-1)$ zaključuje se da pokazivač p može da pokazuje na promenljive a i b , pa se generiše ograničenje $a=b$.

142. Koje ograničenje se generiše kao rezultat analize izraza $a+b$?

- Generiše se ograničenje $a!=b$. (u pitanju je binarna operacija, promenljive ne mogu biti smeštene u istu zonu)

143. Koliko produkcija ima sledeća gramatika?

$S \rightarrow \text{let } E \text{ in } S$
 $S \rightarrow \text{begin } S \text{ L}$
 $S \rightarrow \text{print } E$

$L \rightarrow \text{end}$
 $L \rightarrow ; \text{ S L}$
 $E \rightarrow \text{num} = \text{num}$

Broj produkcija:

To su sledeće produkcije:

144. Šta se događa u fazi prevodenja u međukod kod kompjajlera?

- Stablo sintaksne analize se prevodi u stablo međukoda (IR), koje nije vezano ni za jedan programski jezik, niti za određenu ciljnu platformu.
- Centralna osobina međukoda je njegova NEZAVISNOST od:
 - Izvornog koda (programskog jezika).
 - Ciljnog procesora.

145. Koje osobine poseduje dobar međukod?

- Pogodan je za davanje značenja čvorovima stabla sintaksne analize.
- Pogodan je za prevođenje u mašinski jezik odredišnog procesora.
- Čvorovi stabla međukoda treba da imaju jasna i jednostavna značenja.
(radi lako specifikacije i realizacije optimizacionih transformacija)

146. Šta su čvorovi stabla međukoda?

- Čvorovi stabla međukoda su elementarni izrazi i iskazi.

Izrazi

- CONST(λ): celobrojna konstanta λ
- NAME(n): labela n
- TEMP(t): priv. prom. t (pseudo registar)
- BINOP(o, e_1, e_2): izraz $e_1 o e_2$
 - Aritmetičke o : PLUS, MINUS, MUL, DIV
 - Logičke o : AND, OR, XOR, LSHIFT, RSHIFT,...
- MEM(e): sadržaj mem. objekta na adresi e
- CALL(f, l): poziv funkcije f sa arg. listom l
- ESEQ(s, e): obradi iskaz s radi ivičnih efekata, a zatim izračunaj e kao rezultat

Iskazi (1/2)

- MOVE(TEMP t, e): izračunaj izraz e i upiši u privrmenu prom. t
- MOVE(MEM(e_1), e_2): izračunaj e_1 i e_2 , i upiši e_2 u memoriju na adresu e_1
- EXP(e): izračunaj e i odbaci rezultat
- JUMP($e, labs$): skoči na adresu e
 - e može biti literal labela, npr. NAME($/lab$) ili
 - adresni izraz, npr. switch(i)

Iskazi (2/2)

■ CJUMP($o, e1, e2, t, f$):

- Izračunaj $e1$ i $e2$ i uporedi njihove vrednosti primenom relacionog operatora o
- Ako je rezultat true skoči na labelu t , inače na labelu f
 - o : EQ, NE (označeni ili neoznačeni operandi)
 - o : LT, GT, LE, GE (označeni operandi)
 - o : ULT, UGT, ULE, UGE (neoznačeni operandi)

■ SEQ($s1, s2$): iskaz $s1$ praćen iskazom $s2$

■ LABEL(n): labela n (kao labela u asembleru)

147. Šta se pridružuje čvoru ako se međujezik realizuje u C, C++ i Java jeziku?

- a. C: čvoru se pridružuje struktura.
- b. C++ / Java: čvoru se pridružuje klasa.

148. Do čega dovode ivični efekti?

- a. Ivični efekti dovode do toga da se podizrazi nekog izraza ne mogu izračunavati u bilo kom redosledu.

149. Primeri neslaganja medukoda i mašinskog koda?

◆ CJUMP skače na labelu t ili f

- Mašinski uslovni skok skače ili produžava
 - Npr.: JE JEDNAK

◆ ESEQ i CALL izazivaju da različiti redosledi izračunavanja podizraza daju različite rezultate celog izraza

◆ CALL u okviru argumenata drugih CALL

- Problem: smeštanje argumenata u fiksan broj registara za formalne parametre

150. Koja su 3 koraka transformacije medukoda?

- a. Stablo medukoda se pretvara u listu kanoničkih stabala, koja nemaju ni SEQ i ESEQ čvorove.
 - i. SEQ čvor je niz od dva iskaza.
 - ii. ESEQ čvor je niz od jednog iskaza i jednog izraza. (prvo se izvrši iskaz, a zatim se izračunava izraz)
- b. Lista kanoničkih stabala se pretvara u skup osnovnih programskih blokova.
- c. Osnovni programski blokovi se uređuju u programske tragove (trace), u kojima svaki CJUMP prati labela tačnog uslova. Svaki osnovni blok je pokriven isključivo jednim tragom.

151. Koje bitne osobine poseduje svako kanoničko stablo?

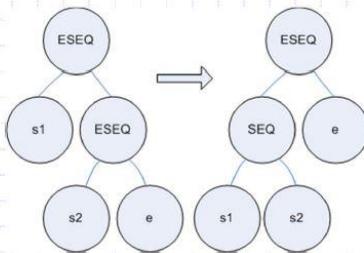
- a. Ne sadrži SEQ i ESEQ čvorove.
- b. Predak CALL čvora (poziv funkcije) je ili MOVE čvor (prebací rezultat izraza u zadatu lokaciju) ili EXP čvor (izraz).

152. Objašnjenja I, II, III i IV grupe pravila pretvaranja?

I grupa pravila pretvaranja

$$\text{ESEQ}(s_1, \text{ESEQ}(s_2, e)) = \text{ESEQ}(\text{SEQ}(s_1, s_2), e)$$

- Direktno iz definicije $\text{ESEQ}(s, e)$ koja glasi:
 - ♦ "izvrši prvo iskaz s , zbog ivičnih efekata, a zatim izračunaj vrednost izraza e "



IV grupa pravila pretvaranja

- U slučaju da s nema ivičnih efekata na e_1 gornja pravila se mogu uprostiti:

$$\begin{aligned}\text{BINOP}(op, e_1, \text{ESEQ}(s, e_2)) &= \\ &\quad \text{ESEQ}(s, \text{BINOP}(op, e_1, e_2)) \\ \text{CJUMP}(op, e_1, \text{ESEQ}(s, e_2), l_t, l_f) &= \\ &\quad \text{SEQ}(s, \text{CJUMP}(op, e_1, e_2, l_t, l_f))\end{aligned}$$

II grupa pravila pretvaranja

$$\begin{aligned}\text{BINOP}(op, \text{ESEQ}(s, e_1), e_2) &= \\ &\quad \text{ESEQ}(s, \text{BINOP}(op, e_1, e_2)) \\ \text{MEM}(\text{ESEQ}(s, e)) &= \text{ESEQ}(s, \text{MEM}(e)) \\ \text{JUMP}(\text{ESEQ}(s, e)) &= \text{SEQ}(s, \text{JUMP}(e)) \\ \text{CJUMP}(op, \text{ESEQ}(s, e_1), e_2, l_t, l_f) &= \\ &\quad \text{SEQ}(s, \text{CJUMP}(op, e_1, e_2, l_t, l_f))\end{aligned}$$

- Direktno slede iz definicija BINOP, MEM, JUMP i CJUMP

III grupa pravila pretvaranja

$$\begin{aligned}\text{BINOP}(op, e_1, \text{ESEQ}(s, e_2)) &= \\ &\quad \text{ESEQ}(\text{MOVE}(\text{TEMP } t, e_1), \text{ESEQ}(s, \text{BINOP}(op, \text{TEMP } t, e_2))) \\ \text{CJUMP}(op, e_1, \text{ESEQ}(s, e_2), l_t, l_f) &= \\ &\quad \text{SEQ}(\text{MOVE}(\text{TEMP } t, e_1), \text{SEQ}(s, \text{CJUMP}(op, \text{TEMP } t, e_2, l_t, l_f)))\end{aligned}$$

- Da bi se sprecili ivični efekat s na e_1 ,
 - ♦ vrednost e_1 se smešta u privremenu promenljivu t

Specijalan slučaj pravila iz IV grupe pravila

- U opštem slučaju, ne zna se da li su dva izraza komutativna
 - ♦ Npr. da li su izraz $\text{MOVE}(\text{MEM}(x), y)$ i $\text{MEM}(z)$ komutativni zavisi od toga da li je $x=z$
- Bilo koji izraz jeste komutativan sa
 - ♦ Praznim iskazom (NOP) i sa
 - ♦ izrazom $\text{CONST}(n)$ – odatle sledi spec. slučaj I pravila iz IV grupe:

$$\begin{aligned}\text{BINOP}(op, \text{CONST}(n), \text{ESEQ}(s, e)) &= \\ &\quad \text{ESEQ}(s, \text{BINOP}(op, \text{CONST}(n), e))\end{aligned}$$

153. Objašnjenje linearizacije međukoda?

- ◆ Na kraju se dolazi do stabla u kom su svi SEQ na vrhu stabla

- Linearizacija se obavlja primenom pravila:

$$\text{SEQ}(\text{SEQ}(a, b), c) = \text{SEQ}(a, \text{SEQ}(b, c))$$

- Tako se dolazi do linearizovanog izraza:

$$\text{SEQ}(s_1, \text{SEQ}(s_2, \dots, \text{SEQ}(s_{n-1}, s_n) \dots))$$

- SEQ više ne daje nikakvu struktturnu informaciju i može se ukloniti – dobija se prosta lista:

$$s_1, s_2, \dots, s_{n-1}, s_n$$

154. Šta je osnovni programski blok?

- a. To je niz susednih iskaza koji:
 - i. Započinje labelom LABEL.
 - ii. Završava iskazom skoka JUMP ili CJUMP.
 - iii. Unutar bloka ne postoji LABEL, JUMP ili CJUMP.

155. Šta je trag?

- a. To je niz iskaza koji se mogu uzastopno izvršavati u toku izvršavanja programa.
- b. Trag može sadržati i uslovne skokove.

156. Šta opisuje tip troška i kakav on može biti?

- a. Tip troška opisuje željeni kriterijum optimizacije.
- b. Trošak može biti:
 - i. Broj instrukcija (ako se kod optimizuje po veličini).
 - ii. Vreme potrebno procesoru (ako se kod optimizuje po brzini izvršenja).

157. Opisati fazu emisije instrukcija za čvor n?

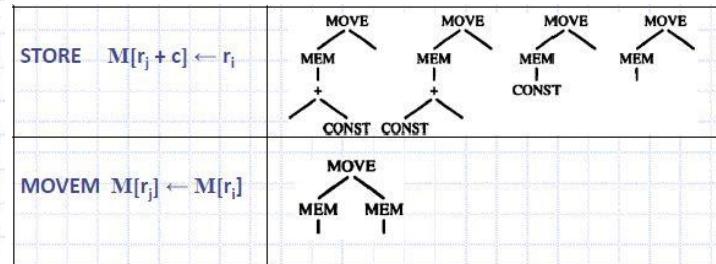
- a. Prvo se emituju instrukcije za listove čvora n, a zatim instrukcije za čvor n.

158. Koje tehnike poboljšanja generisanog koda postoje?

- a. Traženje mogućnosti primene složenijih oblika MAC mašinske instrukcije. (CMAC, MAC, MACN, NMAC, NMACN)
- b. Traženje mogućnosti za korišćenje adresnih registara.
- c. Izbacivanje suvišnih mašinskih instrukcija.

159. Objasnjenje Jouette arhitekture?

Ime	Efekt	Stabla
	r_i	TEMP
ADD	$r_i \leftarrow r_i + r_k$	
MUL	$r_i \leftarrow r_i \times r_k$	
SUB	$r_i \leftarrow r_i - r_k$	
DIV	$r_i \leftarrow r_i / r_k$	
ADDI	$r_i \leftarrow r_i + c$	
SUBI	$r_i \leftarrow r_i - c$	
LOAD	$r_i \leftarrow M[r_i + c]$	



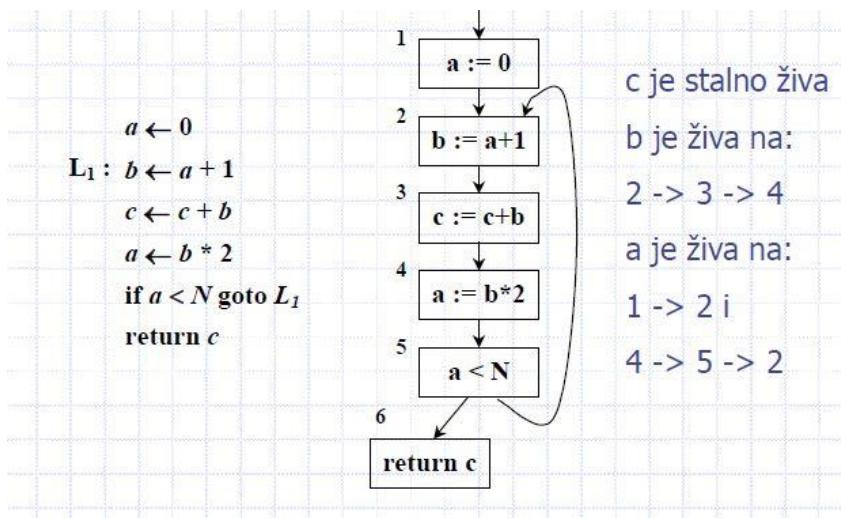
160. Šta je spill promenljivih i kako se rešava?

- a. Spill ili prelivanje promenljivih je problem koji nastaje kada broj privremenih promenljivih prevaziđe broj raspoloživih registara.
- b. Rešava se smeštanjem prekobrojnih promenljivih u sledeću klasu resursa.

161. Kada započinje i kada se završava životni vek promenljive?

- a. Životni vek promenljive započinje prvom definicijom promenljive (upisom početne vrednosti u promenljivu).
- b. Životni vek promenljive se završava zadnjom upotrebom promenljive (poslednjim očitavanjem njenog sadržaja).
- c. Prvi korak u analizi životnog veka promenljive je konstruisanje grafa toka upravljanja.

162. Primer grafa toka upravljanja, uz kod:



163. Objasnenje grafa toka podataka?

- ◆ Čvor grafa toka podataka ima izlazne strelice koje vode do čvorova naslednika i ulazne strelice koje vode od čvorova prethodnika.
- ◆ Skup $\text{pred}[n]$ je skup svih čvorova prethodnika za čvor n , a skup $\text{succ}[n]$ je skup čvorova naslednika čvora n .
- ◆ Nadalje, se definiše $\text{def}[\text{čvor}]$ kao skup promenljivih koje čvor definiše (zadaje im vrednost), a $\text{use}[\text{čvor}]$ kao skup promenljivih koje čvor koristi (očitava).

164. Kada je promenljiva živa na: strelici grafa?

ulazu?
izlazu?

- Promenljiva je živa na strelici grafa ako postoji usmerena putanja od te strelice do USE čvora, bez da ta putanja prelazi preko ijednog DEF čvora.
- Promenljiva je živa na ulazu ako je živa na bilo kojoj ulaznoj strelici čvora.
- Promenljiva je živa na izlazu ako je živa na bilo kojoj izlaznoj strelici čvora.

165. Napisati jednačine životnoj veka za čvor n ?

- Neka su $\text{in}[n]$ i $\text{out}[n]$ redom skupovi promenljivih živih na ulazu, odnosno na izlazu čvora n .
- $\text{in}[n] = \text{use}[n] \cup (\text{out}[n] - \text{def}[n])$.
- $\text{out}[n] = \bigcup_{s \in \text{succ}[n]} \text{in}[s]$.

166. Šta je smetnja?

- Smetnja je problem koji onemogućava da se isti registar dodeli dvema promenljivama, npr. promenljivama a i b .
- Najčešći uzrok smetnje je preklapanje opsega života promenljivih a i b .

167. Opisati matricu smetnji?**Kako izgleda matrica smetnji za promenljive a, b i c iz pitanja 162?**

- a. Matrica smetnji ima onoliko vrsta i onoliko kolona koliko ima promenljivih.

U preseku vrsta i kolona za promenljive između kojih postoji smetnja unosi se znak X.

- b. Matrica za promenljive iz pitanja 162:

	A	b	c
a			X
b			X
c	X	X	

168. Opisati graf smetnji?

- a. Graf smetnji je neusmeren graf čiji čvorovi odgovaraju promenljivama.
- b. Dva čvora grafa se spajaju neusmerenim lukom ako postoji smetnja između odgovarajućih promenljivih na koje se ta dva čvora odnose.

169. Kako glasi pravilo za dodavanje nove smetnje u graf smetnji?

- a. Za svaku definiciju promenljive a u čvoru koji nije MOVE, dodaj smetnju između promenljive a i svake promenljive žive na izlazu čvora.
- b. Za MOVE instrukciju (u čvoru koji jeste MOVE) a=c, dodaj smetnju između promenljive a i svake promenljive žive na izlazu čvora, sem promenljive c.

170. Kako se obavlja dodela registara promenljivama iz grafa smetnji?

- a. Bojenjem grafa smetnji:
- Svakom registru odredišnog procesora odgovara jedna boja, pa se kod procesora sa K registara graf smetnji boji sa K boja, što se naziva K-bojenje.

171. Kada su dva čvora grafa smetnji susedni čvorovi?**Definisati rang čvora i značajan rang čvora?**

- a. Dva čvora grafa smetnji su susedni čvorovi ako između njih postoji smetnja, tj. ako su povezani.
- b. Rang čvora grafa smetnji je jednak broju njemu susednih čvorova.
- c. Značajan rang čvora grafa smetnji je rang koji je veći ili jednak K. (čvorovi sa ovakvim rangom su značajni čvorovi)

172. Koje su osnovne primitive algoritma za dodelu resursa?

- a. FORMIRAJ:
- Formiranje grafa smetnji.
- b. UPROSTI:
- Čvor koji nije značajan se izvlači iz grafa i gura se na stek (jer on nije kandidat za spill).
- c. PRELIJ:
- Primenjuje se u situaciji kada su svi preostali čvorovi grafa smetnji značajni.
Odabere se jedan značajan čvor, označi se kao kandidat za spill, izvuče se iz grafa i gurne se na stek.
- d. IZABERI:
- Dodeljuje boju čvoru sa steka, koji se potom skida sa steka i vraća u graf.
 - Ukoliko nema boje, čvor se preliva.
- e. PONOVI:
- Ukoliko je došlo do prelivanja, čvor je smešten u resurs druge klase (npr. memorijsku lokaciju).
Tada se polazni program mora modifikovati dodavanjem sledećih instrukcija:
 - Instrukcije za dohvatanje promenljive iz odgovarajuće memorijске lokacije pre svake upotrebe promenljive.
 - Instrukcije za ažuriranje odgovarajuće memorijске lokacije nakon svake definicije promenljive.

173. Kada se može ukloniti MOVE instrukcija?

- a. Ako između izvořista i odredišta MOVE instrukcije ne postoji luk grafa smetnji, ona se može ukloniti, a odgovarajući čvorovi se mogu spojiti u novi čvor.

174. Koje su sigurne strategije spajanja čvorova grafa smetnji?

- a. Briggs: čvorovi a i b se mogu spojiti ukoliko rezultujući čvor ab ima manje od K značajnih suseda. K – broj boja.
- b. George: čvorovi a i b se mogu spojiti ukoliko svaki sused čvora a ima smetnju sa b ili nema značajan rang (rang < K).

175. Koje primitive se dodaju u kombinovanom algoritmu za dodelu registara?

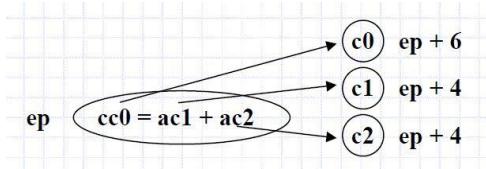
- a. Modifikovana UPROSTI:
 - i. 1 po 1 uklanjaj čvorove koji nisu u vezi sa MOVE instrukcijom i nemaju značajan rang.
- b. SPOJ:
 - i. Obavlja konzervativno spajanje čvorova.
- c. ZAMRZNI:
 - i. Ukoliko se ni UPROSTI ni SPOJ ne mogu primeniti, traži se čvor u vezi sa MOVE koji nema značajan rang.
 - ii. Ako on postoji, svi MOVE u vezi sa njim se zamrzavaju.

176. Objašnjenje prisilnih MOVE čvorova?

- ◆ Postoje MOVE čvorovi koji nije moguće ni spojiti ni zamrznuti.
- ◆ Npr. $x <- y$ i $y <- z$. Prva dva se mogu spojiti, ali resultantni xy ima smetnju sa z , pa se ne može spojiti u xyz .
- ◆ Rezultat je da se xy i z moraju posmatrati kao obični čvorovi.

177. Definisati tačku izvršenja i tačku pristupa operandu u fazi protočne obrade?

- a. Tačka izvršenja je jednaka indeksu instrukcije u nizu instrukcija. (prva instrukcija u nizu ima ep=0)
- b. Tacka pristupa operandu, ap, se dobija tako što se na ep doda redni broj faze protočne obrade u kojoj se pristupa operandu.

**178. Na kojim nivoima radi faza dodele resursa?**

- a. Nivo funkcije:
 - i. Dodela akumulatora i registara opšte namene za parametre funkcija, steka i registra povratne adrese.
- b. Nivo modula:
 - i. Dodela preostalih registara opšte namene i memorije.

179. Šta je ulaz, a šta izlaz iz faze preimenovanja resursa?

- a. ULAZ:
 - i. Graf poziva funkcija (koji se formira u fazi dodele resursa).
 - ii. Informacija o korišćenju akumulatora u pojedinim funkcijama (skupljena se u fazi dodele resursa).
 - iii. Informacija o akumulatorima koji žive preko poziva funkcija.
- b. IZLAZ:
 - i. Skup tabela za preimenovanje akumulatora.
 - ii. Tabela za preimenovanje akumulatora sadrži:
 - 1. Vrste za akumulatore čije ime treba promeniti.
 - 2. Dve kolone – prva sadrži staro ime akumulatora, a druga novo ime akumulatora.

180. Šta je izdavanje (emisija) koda?

- a. To je završna faza u prevođenju programa.
- b. Rezultat rada kompjlera je objektna datoteka u MOF obliku.

181. Na koje čvorove se odnosi: simplifyWorklist?

worklistMoves?

freezeWorklist?

spillWorklist?

- a. simplifyWorklist se odnosi na čvorove niskog ranga bez MOVE vezu.
- b. worklistMoves se odnosi na možda spojive MOVE čvorove.
- c. freezeWorklist se odnosi na čvorove niskog ranga sa MOVE vezom.
- d. spillWorklist se odnosi na čvorove visokog ranga.

182. Nabrojati MOVE (disjunktne) skupove?

- a. coalescedMoves: spojeni MOVE.
- b. constrainedMoves: MOVE čiji src i dst imaju smetnju.
- c. frozenMoves: MOVE koji nisu kandidati za spajanje.
- d. worklistMoves: MOVE koji jesu kandidati za spajanje.
- e. activeMoves: MOVE koji još nisu spremni za spajanje.

183. Šta su ulaz i izlaz modula Color?

- a. ULAZ:
 - i. Graf smetnji.
 - ii. Početna dodela temps.
 - iii. Lista raspoloživih boja.
- b. IZLAZ:
 - i. Konačna dodela boja svih temps u grafu toka upravljanja.

184. Šta je punjač?

- a. Punjač je posebna komponenta programske podrške čiji je zadatak da puni programe na različite adrese u operativnoj memoriji.
- b. Odnosno, punjač je program koji smešta druge programe u memoriju.

185. Koje su osnovne funkcije punjača?

- a. Alokacija – dodela memorije programima.
- b. Relokacija – razmeštaj programa, počev od prve slobodne adrese tj. podešavanje svih adresno osetljivih lokacija na odgovarajući dodeljeni prostor. Punjač relocira program tako što dodaju konstantu na svaku relokabilnu adresu. Program na izlazu iz prevodioca je relokabilan, a izvršivi mašinski je apsolutan.
- c. Povezivanje – određivanje vrednosti simboličkih referenci između relokabilnih programa.
- d. Punjenje – fizički prenos podataka i mašinskih instrukcija u memoriju.

186. Tipovi punjača?

- a. Prevedi i kreni:
 - i. Punjač u okviru programa prevodioca, kompjlera ili asemblera.
 - ii. Prednost: veoma je prost.
 - iii. Mane: 1. Nekorisno trošenje memorijskog prostora i procesorskog vremena.
2. Teško je održati veći broj različitih segmenata.
- b. Apsolutni:
 - i. Apsolutni punjač uvodi sa spoljne memorije program u definisane memorijske lokacije.
 - ii. Mane: 1. Negira suštinu asemblera.
2. Nemoguće je rešiti problem biblioteka ili programa koji se sastoji iz više nezavisnih modula.
- c. BSS:
 - i. Omogućava punjenje više segmenata programa nad samo jednim zajedničkim segmentom podataka.
 - ii. Nedostaci: 1. Vektor prelaza zauzima prostor u memoriji, a služi samo za povezivanje.
2. Oni obrađuju segmente procedura, ali ne olakšavaju pristup segmentima deljivih podataka.
- d. Relokabilni sa direktnim povezivanjem:
 - i. Univerzalni relokabilni punjač.
 - ii. Omogućuje punjenje višestrukih segmenata procedure i višestrukih segmenata podataka.

- e. Povezivač i punjač:
 - i. Povezivač obavlja funkciju dodele memorije, relocacije i povezivanja, dok punjač modula obavlja funkciju punjenja.
- f. Punjač pokrivača:
 - i. Punjač strukture koja se naziva PREKRIVAČ (overlay) i koja se koristi za identifikaciju međusobno isključivih potprograma. Ova struktura se može realizovati samo kada je eksplicitno poznato koji programi i potprogrami pozivaju druge.
- g. Dinamički povezivač:
 - i. Najopštiji tip punjača.
 - ii. To je mehanizam kod kog se punjenje i povezivanje eksternih referenci vrši u vreme izvršavanja.
 - iii. Punjač se poziva tek u slučaju poziva na spoljnu adresu ili globalnu promenljivu.

187. Šta je segment?

- a. Segment je skup kontinualnih reči poređanih jedna za drugom.
- b. Segment je najmanja jedinica za smeštanje programa ili podataka koju prepoznaju prevodilac i punjač.

188. Koji tipovi povezivanja postoje?

Kako se klasificuju simboli u proceduri?

- a. Tipovi povezivanja:
 - i. Lokalno – odnosi se na lokacije u istom segmentu.
 - ii. Međusegmentno – odnosi se na lokacije u nekom drugom, spoljnem segmentu.
- b. Klasifikacija simbola u proceduri:
 - i. Lokalni – relocacija definiše lokalne simbole.
 - ii. Globalni – povezivanje definiše globalne simbole.

189. Šta je relocaciona informacija?

- a. Dobija se kao izlaz asemblera, i to je informacija o tome koje je reference potrebno menjati.

190. Koje tipove tablica izdaje program prevodilac?

- a. Rečnik eksternih simbola (ESD) – sadrži informaciju o svim ulaznim i svim izlaznim simbolima.
- b. Tabelu teksta (TXT) – sadrži prevedenu verziju relokabilnog mašinskog programa.
- c. Relokacioni katalog povezivanja (RLD) – sadrži informaciju o svim adresno osetljivim lokacijama unutar programa.

191. Koja je uloga punjača u I fazi punjenja, a koja u II fazi punjenja?

- a. I faza:
 - i. Dodela i pridruživanje memorije svakom segmentu programa i biblioteke.
 - ii. Formiranje tabele simbola, u koju se unose globalni simboli i absolutne adrese.
- b. II faza:
 - i. Punjenje stvarnog programa.
 - ii. Vršenje relocacione modifikacije nad adresnom komponentom koja tu modifikaciju zahteva.

192. Šta je sekcija?

- a. Sekcija je najmanji deo objektne datoteke koji se relocira. (najčešće korišćene sekcijske su .code, .bss, .data)

193. Kako izgleda zaglavje objektne datoteke?

Okteti	Tip	Naziv	Opis
0-3	long int	f_magic	Magični broj opisuje tip određenog MASx procesora.
4-7	long int	f_stack	Memorijska zona u kojoj se nalazi magacinska memorija.
8-11	long int	f_bpreg	Broj registra u kom se nalazi bazni pokazivački registar.
12-15	long int	f_bpzone	Zona u kojoj se nalazi bazni pokazivački registar.
16-19	long int	f_nscns	Broj sekcijskih objektnih datoteci.
20-23	long int	f_nsyms	Broj podataka u tabeli simbola.

Zaglavljeno objektne datoteke sadrži 16 okteta sa opštim podacima.

194. Kako izgleda MOF format?

ZAGLAVLJE DATOTEKE
TABELA SIMBOLA
Zaglavlj sekcije 1
Podaci sekcije 1
Argumenti sekcije 1
Resursi sekcije 1
Pozvane funkcije iz sekcije 1
Relokacioni podaci za sekciju 1
Brojevi linija za sekciju 1
...
Zaglavlj sekcije n
Podaci sekcije n
Argumenti sekcije n
Resursi sekcije n
Pozvane funkcije iz sekcije n
Relokacioni podaci za sekciju n
Brojevi linija za sekciju n

195. Kako izgleda tabela simbola?

naziv datoteke 1
funkcija 1
promenljive funkcije 1
funkcija 2
promenljive funkcije 2
statičke promenljive
definisani globalni simboli
nedefinisani globalni simboli

- ◆ Raspored elemenata u tabeli simbola nije proizvoljan.
- ◆ Svaki elemenat table simbola je struktura koja sadrži podatke o vrednosti i tipu, i dodatne informacije.
- ◆ Indeks elementa u tabeli simbola je broj elementa pre tog elementa ne računajući dodatne elemente.

Simboli vezani za funkcije

naziv funkcije
argumenti funkcije
.bf
lokalni simboli
.ef

- ◆ U tabeli simbola se posle imena funkcije, a pre specijalnog simbola ".bf", navode argumenti funkcije.
- ◆ Lokalni simboli funkcije se navode između specijalanog simbol ".bf" i specijalanog simbol ".ef"

Spec. simboli u Tabela simbola

Simbol	Značenje
.file	Naziv datoteke.
.bf	Adresa početka funkcije.
.ef	Adresa kraja funkcije.
.xfake	Označava početak strukture, nabranja ili unije.
.eos	Označava kraj strukture, nabranja ili unije.

- ◆ Specijalni simboli, osim simbola ".file", se pojavljuju u parovima.
- ◆ Simboli ".bf" i ".ef" ograničavaju funkciju, dok simboli ".xfake" i ".eos" ograničavaju strukturu, nabranje ili uniju.

Format elementa table simbola

Oktet i	Deklaracija	Naziv	Opis
0-32	char[33]	n_name	Naziv elementa (simbola).
33-36	long int	n_value	Vrednost simbola, koja zavisi od klase za čuvanje.
37-38	short	n_snum	Broj sekcije simbola.
39-42	unsigned long	n_type	Specifikacije osnovnog i izvedenih tipova.
43	char	n_sclass	Klasa za čuvanje simbola.
44	char	n_numaux	Broj dodatnih elemenata.

- ◆ Klasa za čuvanje simbola ima neku od vrednosti iz dozvoljenog skupa vrednosti.

Vrednosti klase za čuvanje simbola

Mnemonik	Vrednost	Klasa za čuvanje
C_MEM0	0	memorijska zona 0
C_MEM1	1	memorijska zona 1
C_ACC	2	akumulator
C_REG0	3	registarsko polje 0
C_REG1	4	registarsko polje 1
C_STACK	5	magacinska memorija
C_DROM0	6	DROM 0
C_DROM1	7	DROM 1
C_FILE	8	naziv datoteke
C_FCN	9	početak i kraj funkcije
C_EXTDEF	10	eksterna definicija
C_EOS	11	kraj strukture
C_STRTAG	12	struktura
C_UNTAG	13	unija
C_TPDEF	14	definicija tipa
C_ENTAG	15	nabranje

Klase za čuvanje spec. simbola

Specijalan simbol	Klasa za čuvanje
.file	C_FILE
.bf	C_FCN
.ef	C_FCN
.xfake	C_STRTAG, C_UNTAG, C_ENTAG
.eos	C_EOS

- ◆ Specijalni simboli imaju samo određene klase za čuvanje.

8

196. Koje mogu biti vrednosti polja BROJ SEKCIJE?

Mnemonik	Broj sekcije	Značenje
N_DEBUG	-2	Specijalan simbol za sistem za otkrivanje grešaka.
N_ABS	-1	Apsolutni simbol.
N_UNDEF	0	Nedefinisani eksterni simbol.
N_SCNUM	1-077777	Broj sekcije u kojoj je simbol definisan.

- ◆ Polje broja sekcije može da sadrži jednu od vrednosti iz ove tabele. Broj sekcije -2 označava simbol koji se koristi u sistemu za otkrivanje grešaka, uključujući i simbole za strukture, unije, nabranja i ime datoteke. Broj sekcije -1 označava da simbol ima vrednost, ali nije relokabilan, dok broj sekcije 0 označava relokabilan simbol koji nije definisan u posmatranom modulu.

10

197. Koji su dodatni elementi za nizove?

Okteti	Deklaracija	Naziv	Opis
0-3	long int	dimnum	broj dimenzija
4-7	long int	dim[0]	prva dimenzija
8-11	long int	dim[1]	druga dimenzija
12-15	long int	dim[2]	treća dimenzija
16-19	long int	dim[3]	četvrta dimenzija

- ◆ Dodatni elemenat za nizove je prikazan u tabeli iznad.
- ◆ Definisanje nizova koji imaju više od 4 dimenzije prouzrokuje generisanje upozorenja.

198. Koji su dodatni elementi za promenljive?

Okteti	Deklaracija	Naziv	Opis
0-3	long int	flags	zastavice
4-7	long int	size	veličina promenljive
8-11	long int	begin	adresa na kojoj je promenljiva definisana
12-15	long int	end	poslednja adresa na kojoj se promenljiva koristi
16-19	long int	offset	pomeraj promenljive od početaka sekcije

- ◆ Zastavice dodatnog elementa predstavljaju podatke o promenljivima i funkcijama koje se ne mogu preneti na drugi način. One su opisane u narednoj tabeli.

199. Značenje i vrednosti određenih zastavica (flags) u dodatnim elementima za promenljive?

Mnemonik	Zastavica	Značenje
AUX_EXT	0x01	funkcija ili promenljiva je eksterna
AUX_EXTDEF	0x02	funkcija ili promenljiva je definisana u ovoj datoteci (eksterna definicija)
AUX_STATIC	0x04	funkcija ili promenljiva je statička
AUX_CONST	0x08	povratna vrednost funkcije ili promenljiva je konstantna
AUX_VALUE	0x400	vrednost je dodeljena promenljivoj

200. Kako izgledaju zaglavlje i okteti sekcije?

Oktet	Deklaracija	Naziv	Opis
0-7	char	s_name	naziv sekcije (8 karaktera)
8-9	unsigned short	s_ninst	broj instrukcija ili broj promenljivih
10-11	unsigned short	s_narg	broj argumenata
12-13	unsigned short	s_nrsc	broj resursa
14-15	unsigned short	s_ncall	broj pozivajućih funkcija
16-17	unsigned short	s_nreloc	broj podataka o relokaciji
18-19	unsigned short	s_nlnno	broj elemenata sa brojevima linija
20-23	long int	s_flags	zastavice (pogledati tabelu na slici 5.40.)

- ◆ Svaka sekcija ima zaglavje sekcije da bi se opisao položaj podataka u datoteci.

Okteti	Deklaracija	Naziv	Opis
0-1	unsigned short	resource	resurs gde se nalazi promenljiva (npr. akumulator)
2-5	long	addr	broj registra ili memorijске lokacije u kojoj se nalazi promenljiva

201. Koje su faze povezivanja programa?

- a. Učitavanje ulaznih i komandnih datoteka.
- b. Prvi prolaz povezivanja.
- c. Optimizacija (NEOBAVEZNA faza). (ova faza je neophodna samo za PROJEKTNE i KOMANDNE datoteke)
- d. Drugi prolaz povezivanja.
- e. Pisanje izlaznih datoteka i datoteka sa informacijama za otkrivanje grešaka.

202. Koje faze obuhvate povezivanje tastature i monitora?

- a. Učitavanje argumenata iz komandne linije (sa tastature).
- b. Prvi prolaz povezivanja.
- c. Optimizacija (opet OPCIONA faza).
- d. Drugi prolaz povezivanja.
- e. Generisanje izveštaja (na monitor).

203. Koje faze obuhvata čitanje objektnih datoteka i biblioteka?

- i. Učitavanje objektnih datoteka (.mof) i biblioteka (.lib) sa diska.
- ii. Konvertovanje tabele simbola u memorijске strukture.
- iii. Punjenje memorije strukturama podataka iz mof sekcija.
- iv. Formiranje memorijске strukture objektne datoteke.

204. Koji su koraci I prolaza povezivanja?

- a. Dodavanje labela iz memorijске strukture objektne datoteke u memorijsku strukturu povezivača:
 - i. U jednoj listi se nalaze: nazivi labela, efektivne adrese i zastavice.
 - ii. U drugoj listi se nalaze: operandi, kod operacije i zastavice, koji se koriste kada se u kodu poziva određena labela.
- b. Dodavanje promenljivih iz memorijске strukture objektne datoteke u memorijsku strukturu povezivača:
 - i. Delovi koda koje treba smestiti u memoriju podataka, zajedno sa promenljivama.
 - ii. Podaci o globalnim, statičkim i lokalnim promenljivama.
 - iii. Izračunavanje adrese steka, formiranje tabele slobodne memorije.
- c. Dodavanje mašinskog koda (dodavanje instrukcija) u memorijsku strukturu povezivača:
 - i. Može biti bez optimizacije ili sa optimizacijom (tada se poziva kompaktor).
 - ii. Spajanje delova koda u jednu celinu.

205. Koji su koraci II prolaza povezivanja?

- a. Zamena simboličkih imena labela njihovim pravim vrednostima.
- b. Nakon toga, instrukcije se iz privremene strukture prebacuju u izlaznu strukturu.
- c. Postoji 9 različitih vrsta izlaznih datoteka:
 - i. MOB datoteka – sadrži izvršni kod.
 - ii. MAS datoteka – sadrži asemblerски kod.
 - iii. MVL datoteka – sadrži kod za prekrivače.
 - iv. MAS datoteka – kod koje postoje labele i komentari.
 - v. TXT datoteka – sadrži podatke o instrukcijama i labelama, nazine promenljivih i funkcija, itd.
 - vi. MAP datoteka – sadrži sve informacije o promenljivim i funkcijama.
 - vii. TXT datoteka – sadrži sve informacije o sačuvanim resursima pri pozivima funkcije.
 - viii. TXT datoteka – sadrži objektnu datoteku napisanu u obliku koji je moguće čitati.
 - ix. DBG datoteka – sadrži sve podatke potrebne za rad Debugger-a.

206. Šta je kompaktor?

Šta je ulaz, a šta izlaz kompaktora?

- a. Kompaktor je uslužni program za mašinski zavisnu optimizaciju.
- b. Ulaz u kompaktor je memorijска struktura koju formira povezivač nakon svog I prolaza.
- c. Izlaz iz kompaktora je kompaktovana memorijска predstava povezanog programa, nad kojom nastavlja rad II prolaz povezivača.

207. Šta je osnovni zadatak kompaktora?

- a. Kompaktor treba da minimizira broj mašinskih instrukcija, tako što iterativno menja kod i uklanja nepotrebne mašinske instrukcije (pre svega NOP instrukcije).
- b. Ovim se postiže ubrzanje izvršenja programa. (sekundarni cilj)

208. Koje su osnovne komponente kompaktora?

- a. Model programa.
- b. Model procesora.
- c. Tehnike kompaktovanja koda nižeg i višeg nivoa.

209. Šta je program?

Šta je kontekst?

Šta je trajektorija?

Šta je značenje?

- a. Program je niz instrukcija čiji objekti su resursi procesora. (registri, U-I prolazi i memorijске lokacije)
- b. Kontekst je skup sadržaja objekata.
- c. Trajektorija je niz konteksta C0, C1, ..., Ci koji je posledica izvršenja niza instrukcija.
- d. Značenje je niz spoljnih rezultata (podataka i akcija) koji su proizvedeni izvršenjem programa.

210. Koji je osnovni zahtev koji kompaktor mora da zadovolji?

- a. Očuvanje semantike programa.

211. Šta je linearni programski segment?

- a. Linearni programski segment ili OSNOVNI BLOK PROGRAMA je niz instrukcija koji započinje labelom, a završava se instrukcijom grananja.
- b. Svaki asemblerски program se može razložiti na skup programskih segmenata.

212. Koje su osnovne faze kompaktovanja?

- a. Faza analize.
- b. Faza sinteze.

213. Šta karakteriše zadatke sa podelom vremena, a šta zadatke sa podelom resursa?

- a. Kod zadataka sa podelom vremena potrebno je smenjivati njihov kontekst.
- b. Kod zadataka sa podelom resursa nema potrebe za smenjivanjem konteksta, tj. za deljenjem vremena.
- c. Primer učešljavanja dva zadatka:

zadatak A:A1, NOP, A2, NOP ... , NOP, An
zadatak B:B1, NOP, B2, NOP... , NOP, Bn

zadatak A+B: A1, NOP, A2, NOP ... , NOP,
An, B1, NOP, B2, NOP... , NOP, Bn

zadatak A||B: A1, B1, A2, B2, ... , An, Bn

214. Koje su tehnike kompaktovanja koda nižeg nivoa?

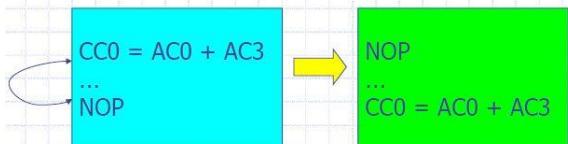
- a. ZAMENI: zameni redosled zadate instrukcije sa zadatim NOP na postojećim adresama unutar segmenta.
- b. IZBACI: izbaci zadati NOP, a da se pritom ne promeni značenje programa.
- c. ZAMENI-PRETHODNI: zameni redosled zadate instrukcije segmenta predak sa zadatim NOP iz prethodnog segmenta.
- d. ZAMENI-NAREDNI: zameni redosled zadate instrukcije segmenta predak sa zadatim NOP iz narednog segmenta.
- e. PREIMENUJ: preimenuj resurs (akumulator) sa korišćenog imena na slobodno. Zameni resurs u instrukciji slobodnim resursom istog tipa.
- f. MUTIRAJ: mutiraj instrukciju.

215. Koje su tehnike kompaktovanja koda višeg nivoa?

- TRAŽI-UNAPRED: polazi od početka programskega segmenta i u smeru njegovog izvršenja traži NOP instrukciju.
- TRAŽI-UNAZAD: ova tehnika polazi od kraja programskega segmenta i u smeru prema početku traži NOP instrukciju.
- KOMPAKTUJ-SA-PRETHODNIM: ova tehnika koristi primitive ZAMENI-PRETHODNI i IZBACI.
- KOMPAKTUJ-SA-NAREDNIM: ova tehnika koristi primitive ZAMENI-NAREDNI i IZBACI.

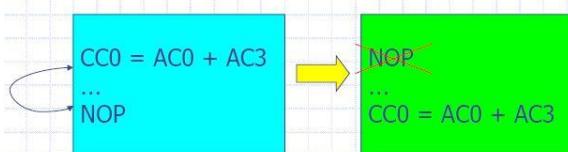
216. Primeri primene tehnika kompaktovanja:

ZAMENI instrukciju i NOP



◆ Ova tehnika pokušava da zameni redosled zadate instrukcije i zadatog NOP u istom segmentu, a da ne promeni značenje programa (pri tom se konsultuje model programa u prostoru objekti-vreme).

TRAŽI-UNAPRED

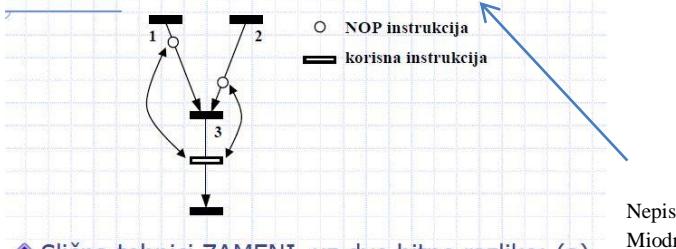


ZAMENI

IZBACI

◆ Ova tehnika višeg nivoa koristi primitive ZAMENI i IZBACI. Polazi od početka programskega segmenta i u smeru njegovog izvršenja (unapred) traži NOP instrukciju.

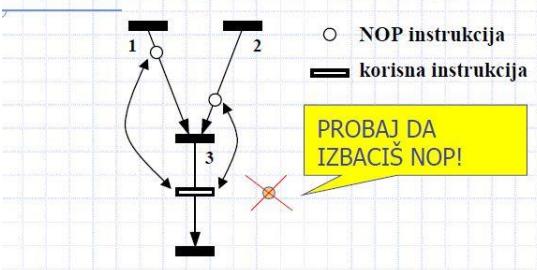
ZAMENI-PREDHODNI



◆ Slična tehnici ZAMENI, uz dve bitne razlike: (a) kontekst je: segment-preteci i (b) u svakom pretku se zamenjuje po jedan NOP.

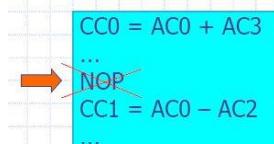
Nepismeni
Miodrag
napada

KOMPACTUJ-SA-PREDHODNIM



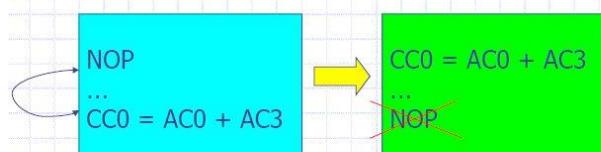
◆ Ova tehnika višeg nivoa koristi primitive ZAMENI-PREDHODNI i IZBACI.

IZBACI zadati NOP



◆ Ona pokušava da IZBACI zadati NOP iz programa, a da ne promeni značenje prorama. Uklanjanje NOP instrukcije iz niza instrukcija, dovodi do približavanja instrukcija koje slede NOP, instrukcijama koje ga prethode.

TRAŽI-UNAZAD

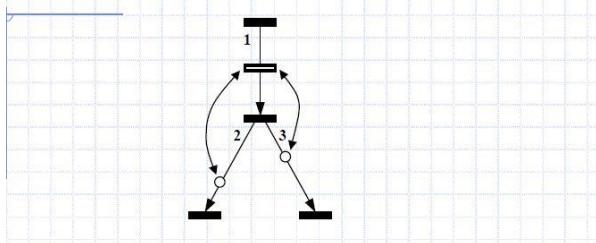


ZAMENI

IZBACI

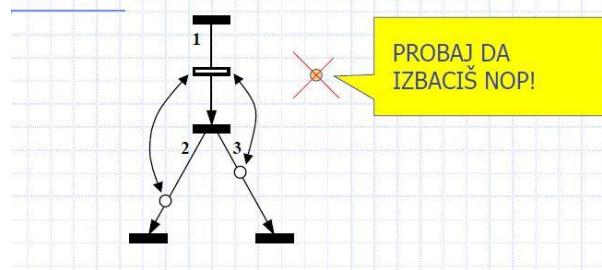
◆ Polazi od kraja programskega segmenta i u smeru prema početku (unazad) traži NOP instrukciju.

ZAMENI-NAREDNI



◆ Ilustracija operacije ZAMENI-NAREDNI: korisna instrukcija iz segmenta 1 zamenjuje NOP instrukcije iz segmenta 2 i 3.

KOMPACTUJ-SA-NAREDNIM



◆ Ova tehnika višeg nivoa koristi primitive ZAMENI-NAREDNI i IZBACI.

Moguća pitanja za ispit iz SPPuRV1

1. Ukratko opisati glavne stavke Insertion sort-a?

- Efikasan algoritam za sortiranje malog broja elemenata.
- Pripada klasi inkrementalnih algoritama.
- Procedura Insertion sort-a:
 - Ulagni niz brojeva predstavljen kao $A[1, 2, \dots, n]$.
 - Broj elemenata (n) zadat je atributom niza $A.length$.
 - Sortiranje brojeva u istom tom nizu (in place).
 - Početni niz A sada sadrži sortiran niz brojeva.

2. Opisati postupak provere korektnosti algoritma?

- Invarijanta petlje se koristi za dokaz da je algoritam korektan. Invarijanta petlje je osobina da je niz $A[1, \dots, j-1]$ uvek sortiran, gde elementi ovog niza odgovaraju već sortiranim brojevima, a j je indeks brojeva koji još nisu sortirani.
- Potrebito je pokazati 3 osobine invarijante petlje:
 - Inicijalizacija: istinita je pre prve iteracije petlje.
 - Održavanje: ako je istinita pre iteracije petlje, ostaje istinita i posle iteracije petlje.
 - Završetak: kada se petlja završi, invarijanta daje korisnu osobinu za dokazivanje korektnosti.

3. Proveriti korektnost procedure Insertion sort?

- Prezentacija 1.3, slajd broj 7.

4. Koje je najgore moguće, a koje najbolje moguće vreme izvršenja za algoritam Insertion sort?

Insertion-Sort(A)	Cena	Broj izvršenja
1. for $j = 2$ to $A.length$	c_1	n
2. $key = A[j]$	c_2	$n - 1$
3. // Umetni $A[j]$ u $A[1..j-1]$	$c_3 = 0$	$n - 1$
4. $i = j - 1$	c_4	$n - 1$
5. while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2..n} t_j$
6. $A[i + 1] = A[i]$	c_6	$\sum_{j=2..n} (t_j - 1)$
7. $i = i - 1$	c_7	$\sum_{j=2..n} (t_j - 1)$
8. $A[i + 1] = key$	c_8	$n - 1$

- ◆ t_j je broj ispitivanja uslova **while** petlje u liniji 5
- ◆ Ispitivanje uslova petlje se izvršava jednom više nego telo petlje
- ◆ Vreme izvršenja $T(n)$ se dobija sabiranjem proizvoda:

$$T(n) = c_1n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1)$$

- ◆ Vreme izvršenja zavisi od toga kakav ulaz te veličine je zadat
- ◆ Najbolji slučaj se dobija kada je ulaz već sortiran
- ◆ Tada je $A[] \leq key$, za svako j , u liniji 5, pa je $t_j = 1$, i najbolje (najkraće) vreme izvršenja je:

$$T(n) = c_1n + c_2(n - 1) + c_4(n - 1) + c_5(n - 1) + c_8(n - 1) = \\ = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) = an + b$$

- ◆ Ako je ulazni niz sortiran u obratnom redosledu, dobija se najgore (najduže) vreme izvršenja
- ◆ Svaki $A[j]$ se mora poređiti sa celim podnizom $A[1..j-1]$, pa je $t_j = j$
- ◆ Uzimajući u obzir:

$$\sum_{j=2}^n j = \frac{n(n+1)}{2} - 1 \quad \sum_{j=2}^n (j - 1) = \frac{n(n-1)}{2}$$

- ◆ sledi da je najgore vreme izvršenja:

$$\begin{aligned}
 T(n) &= c_1n + c_2(n - 1) + c_4(n - 1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + \\
 &\quad + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n - 1) = \\
 &= an^2 + bn - c
 \end{aligned}$$

- ◆ Najgore vreme izvršenja je dakle kvadratna funkcija veličine ulaza $an^2 + bn - c$, gde konstante a , b i c zavise od cena iskaza

5. Ukratko opisati glavne stavke Merge sorta?

- Koristi pristup PODELI I ZAVLADAJ (to su rekurzivni algoritmi).
- Na svakom nivou rekurzije postoje sledeća 3 koraka:
 - PODELI: deljenje problema na nekoliko podproblema (manje instance istog problema).
 - ZAVLADAJ: rekurzivno rešavanje podproblema (ako su podproblemi dovoljno mali, direktno rešavanje).
 - KOMBINUJ: kombinovanje rešenja podproblema u ukupno rešenje originalnog problema.
- Konkretno, algoritam za niz od n elemenata radi na sledeći način:
 - PODELI niz od n elemenata na dva podniza od $n/2$ elemenata.
 - ZAVLADAJ tako što će sortirati oba podniza rekurzivno korišćenjem istog algoritma.
 - KOMBINUJ tako što će spojiti ta dva podniza u jedan sortiran niz.

6. Opisati postupak spajanja 2 sortirana podniza?

- ◆ Spajanje obavlja procedura Merge(A, p, q, r)

- A je niz, a p, q i r su indeksi niza: $p \leq q < r$
- Pretpostavka: $A[p..q]$ i $A[q+1..r]$ već sortirani
- Merge ih spaja u jedan sortiran niz $A[p..r]$

- ◆ Potrebno $\Theta(n)$ vremena, $n = r-p+1$, to je broj elemenata koje treba spojiti

7. Ako je dat pseudo kod procedure Merge, šta svaki od njenih koraka radi?

```

Merge(A, p, q, r)
1.  $n_1 = q - p + 1$ 
2.  $n_2 = r - q$ 
3. dodeli  $L[1..n_1+1]$  i  $R[1..n_2+1]$ 
4. for  $i = 1$  to  $n_1$ 
   5.  $L[i] = A[p + i - 1]$ 
6. for  $j = 1$  to  $n_2$ 
   7.  $R[j] = A[q + j]$ 
8.  $L[n_1+1] = \infty$ 
9.  $R[n_2+1] = \infty$ 
10.  $i = 1$ 
11.  $j = 1$ 
12. for  $k = p$  to  $r$ 
13.   if  $L[i] \leq R[j]$ 
14.      $A[k] = L[i]$ 
15.      $i = i + 1$ 
16.   else  $A[k] = R[j]$ 
17.      $j = j + 1$ 

```

- ◆ 1: računa dužinu n_1 podniza $A[p..q]$
- ◆ 2: računa dužinu n_2 podniza $A[q+1..r]$
- ◆ 3: pravi nizove L i R (levi i desni), dužine n_1+1 i n_2+1
- ◆ 4-5: kopira niz $A[p..q]$ u niz $L[1..n_1]$
- ◆ 6-7: kopira niz $A[q+1..r]$ u niz $R[1..n_2]$
- ◆ 8-9: smeštaju specijalnu vrednost ∞ na kraj nizova L i R
- ◆ 10-17: ponavljaju osnovni korak $r-p+1$ puta

8. Analiza procedure Merge sort daje koje $T(n)$?

◆ Vremena po koracima

- **Podeli:** računa sredinu podniza, pa je $D(n) = \Theta(1)$
- **Zavladaj:** dva podproblema, svaki veličine $n/2$, što daje doprinos ukupnom vremenu izvršenja od $2T(n/2)$
- **Kombinuj:** Merge nad nizom od n elemenata uzima $\Theta(n)$ vremena, pa je $C(n) = \Theta(n)$

◆ Pošto je $C(n)+D(n)=\Theta(n)$, rekurentna jednačina za $T(n)$ za proceduru Merge-Sort glasi:

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n), & n > 1 \end{cases}$$

◆ Primenom master metode, slučaj 2, dobija se rešenje $T(n) = \Theta(n \lg n)$

9. Čemu služi platforma dinamičke paralelizacije?

- Omogućava specificiranje paralelizma u aplikaciji.

10. Šta sadrži konkurentna platforma?

- Sadrži raspoređivač koji uravnovežuje opterećenje procesorskih jezgara.

11. Koje apstrakcije podržava platforma?

- UGNJEŽDENI PARALELIZAM – potprogram se može pokrenuti kao paralelna aktivnost (spawn).
- PARALELNE PETLJE – iteracije se mogu izvršavati paralelno.

12. Koje su prednosti modela dinamičke paralelizacije?

- Pseudo kod sa 3 nove ključne reči: parallel, spawn i sync.
Njihovim brisanjem iz pseudo koda se vrši serijalizacija paralelnog algoritma tj. to je serijski algoritam.
- Ovakav model obezbeđuje kvantifikaciju paralelizma, zasnovanu na pojmovima RAD i RASPON.
- Paralelni algoritmi zasnovani na sistemu podeli-i-zavladaj podležu analizi pomoću rešavanja rekurencija.
- Moderne konkurentne platforme podržavaju neku od varijanti dinamičke paralelizacije.

13. Opisati model paralelnog izvršavanja?

◆ Graf računanja $G = (V, E)$

- Čvorovi u V su instrukcije
- Grane u E prestavljaju zavisnosti između instrukcija
- $(u, v) \in E$ znači da se u mora izvršiti pre v

◆ Serijski iskazi se mogu grupisati u jednu liniju (eng. strand) izvršenja

◆ Instr. za kontrolu || izvršenja se ne uključuju u linije već se prestavljaju u strukturi grafa

- Dakle, skup V formira skup linija, dok skup usmerenih grana iz E prestavlja zavisnosti između njih

◆ Ako u G postoji usmerena putanja od linije u do linije v , dve linije su logički u rednoj vezi

- U suprotnom, linije u i v su logički u paralelnoj vezi

◆ Graf linija izvršenja je ugrađen u stablo instanci procedure

- On je uvećan detalj tog stabla (zoom-in)
- Npr. u prethodnom stablu instanci procedure za poziv P-Fib(5), možemo uvećati deo za računanje P-Fib(4)

4 vrste grana grafa računanja:

- GRANA NASTAVKA (u, u') se crta udesno, povezuje liniju u sa njenim sledbenikom u' unutar iste instance procedure

- GRANA MREŠĆENJA (u, v) se crta nadole, i pokazuje da je linija izvršenja v izmestila liniju v

- GRANA POZIVA se crta takođe nadole, i prestavlja običan poziv procedure (ali ne indukuje nastavak)

- GRANA POVROTAKA (u, x) se crta na gore, i pokazuje da se linija izvršenja u vraća njenoj pozivajućoj proceduri x

14. Kratak opis i analiza Štrasenovog metoda množenja matrica?

◆ Metod se sastoji od sledeća četiri koraka:

- Podeliti matrice A , B i C na podmatrice dimenzije $n/2 \times n/2$. Ovaj korak uzima $\Theta(1)$ vreme.
- Napraviti 10 matrica S_1, S_2, \dots, S_{10} . Ovaj korak uzima $\Theta(n^2)$ vremena.
- Rekursivno izračunati sedam matričnih proizvoda P_1, P_2, \dots, P_7 .
- Izračunati željenje podmatrice $C_{11}, C_{12}, C_{21}, C_{22}$ matrice C . Ovaj korak uzima $\Theta(n^2)$ vremena.

◆ Cilj: odrediti vreme izvršenja $T(n)$

- Za $n=1$ svodi se na prosto skalarno množenje: $\Theta(1)$
- Za $n > 1$, koraci 1, 2 i 4 nose $\Theta(n^2)$ vremena, a korak 3 zahteva sedam množenja matrica dim. $n/2 \times n/2$
- Rekurencija za vreme izvršenja $T(n)$:

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ 7T\left(\frac{n}{2}\right) + \Theta(n^2), & n > 1 \end{cases}$$

- Primenom master metode dobija se rešenje ove jednačine: $T(n) = \Theta(n^{\lg 7})$
- Asimptotski brže od direktnog moženja matrica

15. Paralelizacija Štrasenovog metoda?

◆ Sastoji se od sledeća četiri koraka:

- Podeliti matrice A , B i C na podmatrice dimenzije $n/2 \times n/2$. Ovaj korak uzima rad $\Theta(1)$ i isti raspon.
- Korišćenjem dve ugnježdene **parallel for** petlje napraviti 10 matrica S_1, S_2, \dots, S_{10} . Rad je $\Theta(n^2)$ i raspon je $\Theta(\lg n)$.
- Rekursivno izračunati sedam matričnih proizvoda P_1, P_2, \dots, P_7 .
- Korišćenjem dve ugnježdene **parallel for** petlje izračunati željenje podmatrice $C_{11}, C_{12}, C_{21}, C_{22}$. Rad je $\Theta(n^2)$ i raspon je $\Theta(\lg n)$.

◆ Rad:

- Serijalizacija = originalni algoritam $\Rightarrow T_1(n) = \Theta(n^{\lg 7})$

◆ Raspon:

- Sedam rekursivnih poziva izvršava u paraleli
- Dobija se identična rekurencija kao za P-Matrix-Multiply-Recursive $\Rightarrow T_\infty(n) = \Theta(\lg^2 n)$

◆ Paralelizam: $T_1(n)/T_\infty(n) = \Theta(n^{\lg 7} / \lg^2 n)$

- Malo niži od paralelizma P-Matrix-Multiply-Recursive

16. Gde je usko grlo paralelizma?

- a. To je serijska procedura Merge.

17. Šta podrazumeva dekompozicija u paralelizaciji?

- a. Pronalaženje paralelizma i određivanje nivoa njegove upotrebe.
- b. Razbijanje obrade na zadatke, koji će biti dodeljeni procesima.
 - i. Zadaci mogu nastati dinamički.
 - ii. Broj zadataka može varirati u vremenu.
 - iii. Broj raspoloživih zadataka u određenom trenutku predstavlja gornju granicu ostvarivog ubrzanja.

18. Šta podrazumeva dodela (granularnost) u paralelizaciji?

- a. Određivanje mehanizma podele posla na dostupan broj jezgara.

19. Šta predstavlja dekompozicija zadataka?

- a. Nezavisne grube obrade, svojstvene algoritmu.
- b. Posmatra se niz iskaza koji operišu kao grupa.
- c. Predstavlja paralelizam u aplikaciji.

20. Šta predstavlja dekompozicija podataka?

- a. Ista obrada se primenjuje na malim blokovima podataka, koji su izvedeni iz velikog skupa podataka.
- b. Predstavlja istu obradu puno podataka (velikog broja podataka).

21. Šta predstavlja dekompozicija protočne obrade?

- a. Linije sklapanja podataka.
- b. Lunci proizvođača-potrošača.

22. Koje su 2 česte dekompozicije zadataka?

- a. Pozivi funkcija.
- b. Različite iteracije u petlji.

23. Kada je dekompozicija podataka dobra početna tačka (tj. kada je dobro početi njom, a ne dekompozicijom zadataka) ?

- a. Kada je glavna obrada organizovana oko manipulacije velike strukture podataka.
- b. Kada se primenjuju slične operacije nad različitim delovima strukture podataka.

24. Koje su česte dekompozicije podataka?

- a. Dekompozicija nizova po vrstama / kolonama / blokovima. (strukture podataka u obliku nizova)
- b. Dekompozicija stabla u podstabla. (rekurzivne strukture podataka)

25. Opisati ponovni inženjering radi paralelizacije programa?

- a. Definisanje predmeta rada i dobijanje saglasnosti korisnika.
- b. Definisanje protokola testiranja / prijema radova.
- c. Određivanje vrućih tačaka (gde se troši najviše vremena?)
 - i. Analiza uvidom u kod.
 - ii. Analiza pomoću alata za profilisanje koda.
- d. Paralelizacija.

26. Ukratko opisati paralelizam zadataka?

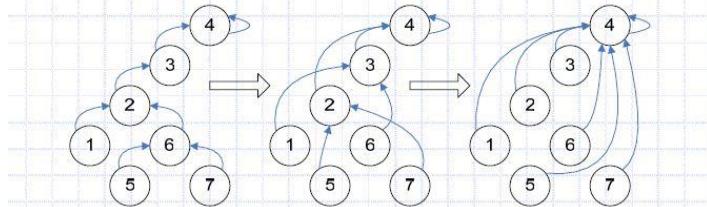
- a. Odlikuju ga:
 - i. **Praćenje zraka** (ray tracing) – obrada za svaki zrak je odvojena i nezavisna.
 - ii. **Dinamika molekula** – obrada neograničenih sila.
 - iii. **Zajednički faktori** – zadaci su pridruženi iteracijama petlje i uglavnom su poznati na početku obrade, ne moraju se završiti svi zadaci da bi se došlo do konačnog rešenja.

27. Ukratko opisati rekurzivne podatke?

- a. Obrane nad listama, stablima, grafovima.
- b. Primer rekurzivnih podataka: pronalaženje korena

◆ Ako je data šuma usmerenih stabala sa korenom, pronađi koren stabla koje sadrži čvor

- Paralelni pristup: za svaki čvor pronađi prethodnikovog prethodnika, ponavljam dok ima izmena
 - ♦ $O(\log n)$ naspram $O(n)$



28. Ukratko opisati paralelizam protočne obrade?

- a. Ograničen je brojem stepeni protočne obrade.
- b. Radi dobro ako je vreme punjenja i pražnjenja protočne obrade malo u odnosu na vreme obrade.
- c. Mere performanse:
 - i. Propusnost – stopa kojom se podaci pojavljuju na kraju protočne obrade.
 - ii. Kašnjenje – vremenski interval od ulaza podataka u protočnu obradu do izlaza. (bitno za aplikacije u realnom vremenu)

29. Ukratko opisati koordinaciju zasnovanu na dogadjajima?

- Interakcija zadataka radi obrade podataka se događa u nepredvidivim vremenskim intervalima.
- U aplikacijama koje koriste ovaj šablon moguća su međusobna blokiranja zadataka (deadlock).

30. Ukratko opisati SPMD (Single Program Multiple Data) šablon?

- Jedan program, više podataka – iz jednog izvornog koda stvara programe koji se izvršavaju na više procesora.
- Faze:
 - Inicijalizacija
 - Dobavljanje jedinstvenog identifikatora
 - Izvršenje istog programa na svakom procesoru (identifikator i ulazni podaci dovode do različitog ponašanja)
 - Distribuiranje podataka
 - Dovršavanje (finalizacija).

31. Ukratko opisati šablon Vodeći / Radnik ?

- Relevantan za probleme koji koriste paralelizam zadataka, gde zadaci nemaju zavisnost. (zastižujuće paralelni programi – embarrassingly parallel)
- Glavni izazov je određivanje kada je ceo problem obrađen.

32. Ukratko opisati šablon Grananje / Pridruživanje ?

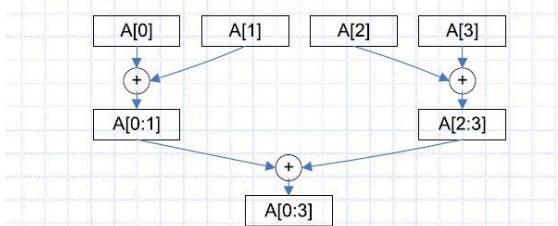
- Zadaci se stvaraju dinamički (zadaci mogu stvoriti još zadataka).
- Zadacima se rukuje u skladu sa njihovim odnosima.
- Zadatak predak stvara nove zadatke (GRANANJE), a zatim čeka da se završe (PRIDRUŽIVANJE) pre nego što nastavi sa obradom.

33. Ukratko opisati serijsku redukciju?

- ◆ Kada operator redukcije nije asocijativan
- ◆ Obično ga prati slanje rezultata svima
- ◆ Npr. serijsko skupljanje A[0], A[1], A[2], A[3] u čvoru koji ima A[0]:
 - Skupi A[1], dobija se A[0:1]
 - Skupi A[2], dobija se A[0:2]
 - Skupi A[3], dobija se A[0:3]
 - Kraj

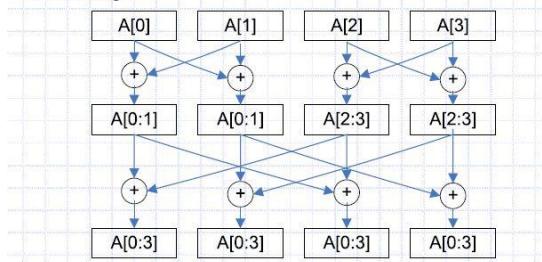
34. Ukratko opisati redukciju zasnovanu na stablu?

- ◆ n koraka za 2^n jedinica izvršenja
- ◆ Kada je operator redukcije asocijativan
- ◆ Posebno atraktivan ako je rezultat potreban samo jednom zadatku



35. Ukratko opisati rekurzivno-udvajajuću redukciju?

- ◆ n koraka za 2^n jedinica izvršenja
- ◆ Ako svim jedinicama izvršenja treba rezultat redukcije



36. Opisati model izvršenja zadataka u Cilk-u ?

- a. Linija izvršenja (strand): niz instrukcija bez spawn i sync.
- b. Tačka mrešćenja (spawn): jedna linija se pretvara u dve.
- c. Tačka sinhronizacije:
 - i. Jedna ili dve linije se pretvaraju u jednu.
 - ii. Inicijalne linije se mogu izvršavati paralelno.
 - iii. Inicijalne linije se izvršavaju sekvencijalno sa novom linijom.
 - iv. Linija se može podeliti na kraće linije.
 - v. Max linija je ona koja se ne može uključiti u dužu.

37. Ukratko opisati hiperobjekte?

- a. Omogućavaju siguran pristup deljenim objektima, tako što svakoj paralelnoj liniji daju posebnu instancu.
- b. Obraćanje hiperobjektu rezultuje u referenci koja se naziva POGLED.
- c. Hiperobjekti najčešće spadaju u reduktore.

38. Kada se može izvršiti parallel_reduce?

- a. Kada je nit radnik (worker) raspoloživa.

39. Kada se koristi parallel_do?

- a. Kada iteracioni prostor nije unapred poznat.

40. Gde se koriste kružni baferi? Šta je usko grlo protočne obrade?

- a. Kružni baferi se koriste između filtera protočne obrade.
- b. Usko grlo protočne obrade je najsporiji filter.

41. Ukratko opisati pravljenje konvoja (convoying) ?

- a. OS prekine (neku) nit koja drži ključ.
- b. Druge niti moraju da sačekaju da prekinuta nit oslobodi svoj ključ.
- c. Kod fair_mutex-a moraju da sačekaju nastavak prekinute niti.

42. Kako se izbegava konvoj?

- a. Minimalnim vremenom držanja ključa.
- b. Korišćenjem atomskih operacija umesto ključeva.

43. Šta su prednosti, a šta nedostaci atomskih operacija u odnosu na operacije sa ključevima?

- a. Prednosti:
 - i. Brže su od operacija sa ključevima.
 - ii. Nema deadlock-a niti konvoja.
- b. Nedostaci:
 - i. Rade ograničen skup operacija.

44. Koje su osnovne operacije nad x tipa atomic<T> ?

- Pet osnovnih operacija nad x tipa atomic< T >
 - ♦ = x, pročitaj x
 - ♦ x =, upiši vrednost u x i vrati tu vrednost
 - ♦ x.fetch_and_store(y), uradi x=y i vrati staru vrednost od x
 - ♦ x.fetch_and_add(y), uradi x+=y i vrati staru vrednost od x
 - ♦ x.compare_and_swap(y,z), ako je x==z, uradi x=y. Uvek vrati staru vrednost od x

45. Šta radi rasporedivač zadataka?

- a. Realizuje paralelne apstrakcije visokog nivoa.
- b. Programske niti se preslikavaju na fizička jezgra.

46. Šta radi alat Survey?

- a. Otkriva mesta na kojima se troši najviše vremena (tzv. hotspot-ovi).
- b. Procenjuje performanse nekih linija koda.
- c. Ne zahteva niti gleda oznake.

47. Šta radi alat VS editor?

- a. Omogućava unos i izmene oznaka.
- b. Nakon unosa oznaka, vrši se zamena oznaka stvarnim kodom paralelnog okruženja.
- c. Zahteva Release konfiguraciju.

48. Šta radi alat Suitability?

- a. Vrši procenu performanse za označene delove.
- b. Ima ulogu u matematičkom modeliranju performanse teorijski idealne paralelne mašine.
- c. Zahteva Release konfiguraciju.
- d. Zahteva oznake mesta i zadataka. Prepoznaće neke oznake.

49. Šta radi alat Correctness?

- a. Prikazuje moguće probleme deljenja podataka. (na osnovu oznaka paralelnih mesta, zadataka, ključeva, itd.)
- b. Zahteva Debug konfiguraciju i ograničenje veličine podataka. (100 puta sporije izvršenje spram normalnog)
- c. Zahteva oznake mesta i zadataka. Prepoznaće sve oznake.

50. Šta radi selekcija (označavanje) koda?

- a. Vrši odabir i označavanje najbolje mogućnosti paralelizacije.

51. Šta radi alat Summary?

- a. Donosi odluku o prihvatanju predloženih zadataka.

52. Ukratko opisati proširivanje makroinstrukcija?

- a. Obavlja se u fazi prevođenja.
- b. Makroinstrukcija se zamenjuje telom odgovarajuće makrodefinicije.
- c. Formalni parametar u telu makrodefinicije se zamenjuje vrednošću stvarnog parametra iz polja operanda makroinstrukcije koja se proširuje.
- d. Dalje sledi I prolaz asemblera.

53. Ukratko opisati proces generisanja reči?

- a. Svi nizovi jezika se razvijaju iz početnog simbola, koji je označen neterminalni simbol.
- b. Proces generisanja se sastoji od primene jedne redukcije ili smene u svakom koraku.
- c. Ovakav proces pretvara jednu reč u drugu.

54. Šta karakteriše interpretore bez ivičnih efekata?

- a. Bez ivičnih efekata znači bez iskaza dodele.
- b. Vrednost se dodeljuje samo pri inicijalizaciji. Npr. int i=j+3.

55. Navesti operacije u matrici prelaza:

- a. Accept – dolazak do kraja – program je ispravan.
- b. Error – sledeća vrednost na traci je invalidna u tekućem stanju.
- c. Shift n – sledeća vrednost na traci je prihvatljiva u tekućem stanju; stanje i vrednost se guraju na stek.
- d. Goto n – stanje i neterminal se guraju na stek.
- e. Reduce m – sa steka se skida onoliko elemenata koliko ima simbola sa desne strane m-te redukcije.

56. Ukratko opisati fazu generisanja koda kompjajlera?

- a. Ulaz algoritma ove faze je matrica i kod smena (makrodefinicija), koji definišu operator koji se pojavljuju u matrici.
- b. Konsultuju se tabele identifikatora i literala, određivanjem tipa i lokacije promenljivih (operanda matrice).
- c. Operandi matrice se koriste kao argumenti makrodefinicija koje definišu operatore.

57. Ukratko opisati fazu asembleriranja kompjajlera?

- a. Ako se u fazi generisanja proizvede asemblerski kod, jednopravni ili dvopravni asembleri da bi se dobio potreban kod na mašinskom jeziku.

58. Nabrojati operacije API U/I biblioteka?

- a. open() – otvara datoteku kojom je predstavljen uređaj ili kontroler UI sprege.
- b. read() – operacija čitanja.
- c. write() – operacija upisa.
- d. close() – operacija zatvaranja datoteke kojom je program predstavljen.
- e. ioctl() – služi za nadzor stanja i upravljanje režimom uređaja.

59. Ukratko opisati kompjajler za dijalekt C-a?

- a. Prevodi izvorni program napisan na MAS-like C jeziku na mašinski jezik.
- b. MAS-like C jezik je podskup standardnog ANSI C jezika, koji sadrži 2 nova ugradena tipa promenljivih koje su označene sa dsp20 i dsp32.

60. Ukratko opisati postupak otkrivanja petlji koje mogu biti fizički podržane (hardware loop) ?

- a. Treba pronaći petlje iz polaznog programa koje se mogu podržati tzv. fizičkom petljom procesora (hardware loop).
- b. Radi otkrivanja ovakvih petlji, analiziraju se čvorovi iskaza petlji (for, while, do_while) u stablu sintaksne analize.
- c. Uslovi su:
 - i. Poznati su naziv i početna vrednost indeksne promenljive.
 - ii. Poznat je korak iteracije.
 - iii. Poznat je broj iteracija.
 - iv. U kodu nema SKIP i JUMP instrukcija. (CONTINUE i BREAK su dozvoljene)
 - v. U kodu nema instrukcija fizičke petlje.
 - vi. U telu petlje ima manje od 128 instrukcija.

61. Ukratko opisati sintaksni analizator sa rekurzivnim spuštanjem?

- a. Ima po jednu međurekursivnu funkciju za svaki neterminal (sintaksnu klasu).
- b. Ograničenje mu je što radi dobro samo ako prvi terminalni simbol u svakom podizrazu jednoznačno određuje produkciiju, u suprotnom dolazi do konflikta.

62. Za sledeću gramatiku odrediti sve neterminalne i terminalne simbole?

**S → let E in S
S → begin S L
S → print E**

**L → end
L → ; S L
E → num = num**

Rešenje: neterminalni simboli su S, E, L

terminalni simboli su let, in, begin, print, end, ; , num, =

Neterminalni simboli definišu dozvoljene iskaze i izraze u jeziku.
 Terminalni simboli su ključne reči, punktuatori i operatori u jeziku.

63. Slediće gramatiku svesti na gramatiku bez konflikta?

$S \rightarrow E \$$		
$E \rightarrow E + T$	$T \rightarrow T * F$	$F \rightarrow id$
$E \rightarrow E - T$	$T \rightarrow T / F$	$F \rightarrow num$
$E \rightarrow T$	$T \rightarrow F$	$F \rightarrow (E)$

Rešenje:

Gramatika sa konfliktima

- Tada je potrebno odrediti skupove FIRST i FOLLOW
- Za niz terminala i neterminala γ , skup FIRST čine svi terminali kojima započinju simboli razvijeni iz γ
 - Npr. za $\gamma = \{T * F\}$, $FIRST(\gamma) = \{id, num, ()\}$
- Simbol X je poništiv (nullable) ako se iz njega može razviti prazan niz
- Skup FOLLOW(X) je skup terminala koji u rečenicama jezika mogu pratiti X

6

Pored transformacije jezika, potrebno je odrediti sadržaje skupova

- Za svaki neterminal: nullable, FIRST i FOLLOW

Simbol	nullable	FIRST	FOLLOW
S	false	(id num)
E	false	(id num) \$
E'	true	+ -) \$
T	false	(id num) + - \$
T'	true	* /) + - \$
F	false	(id num) * / + - \$

◆ Opšte pravilo za prepisivanje regularnih izraza sa levom rekurzijom u izraze sa desnom rekurzijom

- Primjenjeno na posmatrani primer

Početni skup produkcija	Rezultat transformacije
$X \rightarrow X \gamma_1$	$X \rightarrow \alpha_1 X^*$
$X \rightarrow X \gamma_2$	$X \rightarrow \alpha_2 X^*$
$X \rightarrow \alpha_1$	$X^* \rightarrow \gamma_1 X^*$
$X \rightarrow \alpha_2$	$X^* \rightarrow \gamma_2 X^*$
$X^* \rightarrow$	$X^* \rightarrow$

Jezik nakon transformacije:

$S \rightarrow E \$$		
$E \rightarrow T E'$	$T \rightarrow F T'$	
$E' \rightarrow + T E'$	$T' \rightarrow * F T'$	$F \rightarrow id$
$E' \rightarrow - T E'$	$T' \rightarrow / F T'$	$F \rightarrow num$
$E' \rightarrow$	$T' \rightarrow$	$F \rightarrow (E)$

8

64. Opisati fazu izbora instrukcija kod kompjlera?

- Izbor instrukcija predstavlja problem popločavanja stabla međukoda minimalnim skupom šablonu stabla.
- Najbolje popločavanje stabla međukoda dovodi do niza instrukcija sa najmanjim troškom.

65. Ukratko opisati algoritam dinamičkog programiranja?

- Ovaj algoritam dodeljuje trošak svakom čvoru stabla međukoda.
- Trošak posmatranog čvora je definisan kao zbir troškova pojedinačnih instrukcija iz najboljeg niza instrukcija koje popločavaju podstablo, čiji je koren posmatrani čvor.

66. Šta uključuje pisanje modula za izbor instrukcija?

- Modeliranje određenog procesora.
- Realizaciju algoritma dinamičkog programiranja.

67. Ukratko opisati sve 3 tehnike poboljšanja generisanog koda?

- Primena složenih MAC:
 - Traži se kombinacija jednostavne MAC i ALU instrukcije (ne moraju biti susedne, ali MAC mora prethoditi ALU instrukciji), u kojoj se rezultat množenja dva operanda iz prve instrukcije akumulira drugom instrukcijom.
 - Kad se takva kombinacija pronađe, prva instrukcija se zamjenjuje složenijom MAC instrukcijom.

- b. Korišćenje adresnih registara:
 - i. Nakon dodele adresnog registra pokazivaču iz polaznog programa, najpre se analiziraju sve mašinske instrukcije koje koriste pokazivač radi prepoznavanja mogućnosti primene adresnog režima samouvećanja (auto increment) ili samoumanjenja (auto decrement) adresnog registra.
- c. Izbacivanje suvišnih mašinskih instrukcija:
 - i. Metoda DeadCodeElimination uklanja instrukcije iz ALU i MAC grupe ukoliko rezultat instrukcije ne koristi nijedna druga instrukcija do kraja posmatranog bloka.
 - ii. Metoda RemoveUnnecessaryVarLoads uklanja definicije promenljive koja je izgubila važnost novom definicijom (overridden).

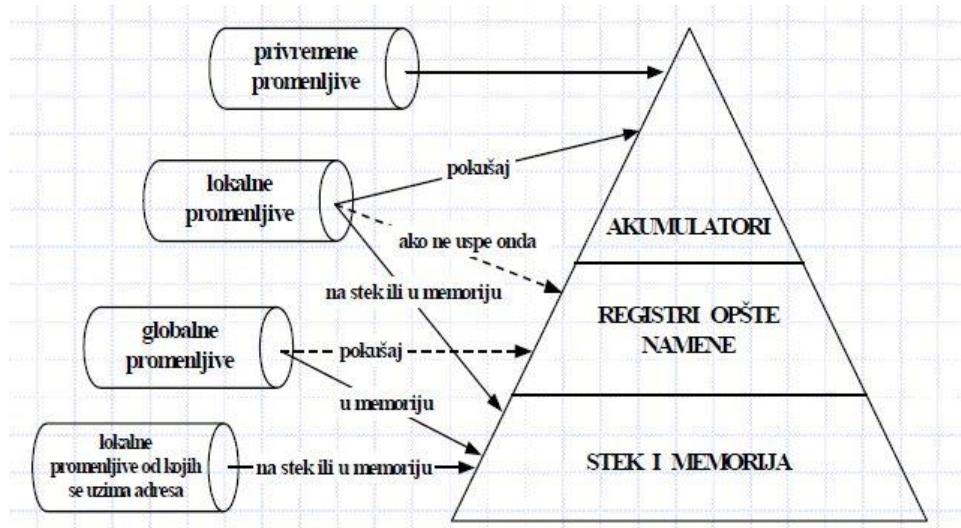
68. Koje su anomalije (greške, nedostaci) primitive PONOVI?

- a. Prelivena promenljiva se pretvara u više privremenih promenljivih sa vrlo kratkim opsegom života.
- b. Ove privremene promenljive izazivaju nove smetnje sa postojećim promenljivama u grafu smetnji.
- c. Zbog toga se ceo postupak dodele resursa mora ponoviti na novoj verziji programa.

69. Na čemu se zasniva agresivno bojenje grafa sa spajanjem prosutih čvorova?

1. Konstruisati graf smetnji za prosute čvorove.
2. Spoj parove prosutih povezanih sa MOVE, koji nemaju međusobne smetnje.
3. Primeni UPROSTI i IZABERI – dodeljene boje sada odgovaraju memorijskim lokacijama prosutih čvorova.

70. Kako izgleda piramida resursa?



71. Ukratko opisati editor povezivanja?

- a. On prati relocacionu informaciju tako da rezultujući modul punjenja kao celina može nadalje da se relocira i napuni u proizvoljnu zonu u memoriji.

72. Šta je zadatak povezivača za MASx IDE?

- a. Povezivanje objektnih datoteka, nastalih prevodenjem pojedinačnih modula izvornog koda, u jedan izvršivi program koji se puni u PROM ili u jedan prekrivač koji se puni u DRAM, a zatim smešta u programski adresni prostor.

73. Šta radi kompaktor u fazi analize?

- a. Na osnovu polaznog asemblerorskog koda i ugrađenog modela procesora, kompaktor formira model polaznog programa.

74. Od čega se sastoji model polaznog programa?

- a. Od skupa instanci klasa instrukcija, koji je sortiran po tački izvršenja.
- b. Od nekoliko kolekcija labela (programske, simboličke, labele podataka, labele skokova).
- c. Od jedne liste objekata programskih segmenata.

75. Koji atributi osnovne klase su ključni za rad kompaktora?

- a. Adresa tačke izvršenja instrukcije.
- b. Niz objekata klase CObject, koje instrukcija na neki način koristi.

76. Koji su ključni atributi objekta?

- a. Faza protočne obrade u kojoj se objekat koristi.
- b. Način korišćenja objekta: čitanje ili upis.
- c. Tip objekta (akumulator, registar, itd.).

77. Pomoću kojih klasa se modelira tok upravljanja?

- a. Osnovni segment.
- b. Segment.
- c. Virtuelni segment.

78. Ukratko opisati osnovni programski segment?

- a. Modelira jedan izolovan programski segment.
- b. Njegovi atributi su početna adresa i završna adresa segmenta.

79. Ukratko opisati programski segment?

- a. Modelira programski segment kao sastavni deo programa.
- b. Njegovi atributi su:
 - i. Lista prethodnih programskih segmenata.
 - ii. Lista narednih segmenata.
 - iii. Lista virtuelnih programskih segmenata.

80. Ukratko opisati virtuelni programski segment?

- a. Nelinearni segment (sadrži instrukcije skoka) koji obuhvata posmatrani linearni programski segment.
- b. Virtuelni segment je deo traga (trace).

81. Šta radi kompaktor u fazi sinteze?

- a. Polazni program se bolje prilagođava protočnoj obradi, tako što se međusobno nezavisni zadaci učešljavaju (interleave).
- b. Faza sinteze zapravo predstavlja kompaktovanje.

PRIMERI ZADATAKA ZA ELIMINACIJE I ISPIT IZ SPPURV1

MASTER TEOREMA:

$$1. T(n) = 3T(n/2) + n^2$$

$$7. T(n) = 2T(n/2) + n/\log n$$

$$15. T(n) = 3T(n/4) + n \log n$$

$$2. T(n) = 4T(n/2) + n^2$$

$$8. T(n) = 2T(n/4) + n^{0.51}$$

$$16. T(n) = 3T(n/3) + n/2$$

$$3. T(n) = T(n/2) + 2^n$$

$$9. T(n) = 0.5T(n/2) + 1/n$$

$$17. T(n) = 6T(n/3) + n^2 \log n$$

$$4. T(n) = 2^n T(n/2) + n^n$$

$$10. T(n) = 16T(n/4) + n!$$

$$18. T(n) = 4T(n/2) + n/\log n$$

$$5. T(n) = 16T(n/4) + n$$

$$12. T(n) = 3T(n/2) + n$$

$$20. T(n) = 7T(n/3) + n^2$$

$$6. T(n) = 2T(n/2) + n \log n$$

$$13. T(n) = 3T(n/3) + \sqrt{n}$$

$$21. T(n) = 4T(n/2) + \log n$$

$$14. T(n) = 4T(n/2) + cn$$

$$22. T(n) = T(n/2) + n(2 - \cos n)$$

Solutions

$$1. T(n) = 3T(n/2) + n^2 \implies T(n) = \Theta(n^2) \text{ (Case 3)}$$

$$2. T(n) = 4T(n/2) + n^2 \implies T(n) = \Theta(n^2 \log n) \text{ (Case 2)}$$

$$3. T(n) = T(n/2) + 2^n \implies \Theta(2^n) \text{ (Case 3)}$$

$$4. T(n) = 2^n T(n/2) + n^n \implies \text{Does not apply (}a\text{ is not constant)}$$

$$5. T(n) = 16T(n/4) + n \implies T(n) = \Theta(n^2) \text{ (Case 1)}$$

$$6. T(n) = 2T(n/2) + n \log n \implies T(n) = n \log^2 n \text{ (Case 2)}$$

$$7. T(n) = 2T(n/2) + n/\log n \implies \text{Does not apply (non-polynomial difference between }f(n)\text{ and }n^{\log_b a}\text{)}$$

$$8. T(n) = 2T(n/4) + n^{0.51} \implies T(n) = \Theta(n^{0.51}) \text{ (Case 3)}$$

$$9. T(n) = 0.5T(n/2) + 1/n \implies \text{Does not apply (}a < 1\text{)}$$

$$10. T(n) = 16T(n/4) + n! \implies T(n) = \Theta(n!) \text{ (Case 3)}$$

$$11. T(n) = \sqrt{2}T(n/2) + \log n \implies T(n) = \Theta(\sqrt{n}) \text{ (Case 1)}$$

$$12. T(n) = 3T(n/2) + n \implies T(n) = \Theta(n^{\lg 3}) \text{ (Case 1)}$$

13. $T(n) = 3T(n/3) + \sqrt{n} \implies T(n) = \Theta(n)$ (Case 1)

14. $T(n) = 4T(n/2) + cn \implies T(n) = \Theta(n^2)$ (Case 1)

15. $T(n) = 3T(n/4) + n \log n \implies T(n) = \Theta(n \log n)$ (Case 3)

16. $T(n) = 3T(n/3) + n/2 \implies T(n) = \Theta(n \log n)$ (Case 2)

17. $T(n) = 6T(n/3) + n^2 \log n \implies T(n) = \Theta(n^2 \log n)$ (Case 3)

18. $T(n) = 4T(n/2) + n/\log n \implies T(n) = \Theta(n^2)$ (Case 1)

19. $T(n) = 64T(n/8) - n^2 \log n \implies$ Does not apply ($f(n)$ is not positive)

20. $T(n) = 7T(n/3) + n^2 \implies T(n) = \Theta(n^2)$ (Case 3)

21. $T(n) = 4T(n/2) + \log n \implies T(n) = \Theta(n^2)$ (Case 1)

22. $T(n) = T(n/2) + n(2 - \cos n) \implies$ Does not apply. We are in Case 3, but the regularity condition is violated. (Consider $n = 2\pi k$, where k is odd and arbitrarily large. For any such choice of n , you can show that $c \geq 3/2$, thereby violating the regularity condition.)

$$T(n) = 3T\left(\frac{n}{3}\right) + n \log n.$$

In this case, $n^{\log_b a} = n$. While f is asymptotically larger than n , it is larger only by a logarithmic factor; it is not the case that $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$. Therefore, the master theorem makes no claim about the solution to this recurrence.

$$T(n) = 2T\left(\frac{n}{2}\right) + n.$$

$n^{\log_b a} = n$ and $f(n) = n$, so case 2 of the master theorem gives $T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n \log n)$.

Similarly, as mentioned before, traversing a binary tree takes time

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1).$$

$n^{\log_b a} = n$, which is asymptotically larger than a constant factor, so case 1 of the master theorem gives $T(n) = \Theta(n^{\log_b a}) = \Theta(n)$.

You *cannot* use the Master Theorem if

- $T(n)$ is not monotone, ex: $T(n) = \sin n$
- $f(n)$ is not a polynomial, ex: $T(n) = 2T\left(\frac{n}{2}\right) + 2^n$
- b cannot be expressed as a constant, ex: $T(n) = T(\sqrt{n})$

EXAMPLE

Consider the recurrence

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2.$$

In this case, $n^{\log_b a} = n$ and $f(n) = n^2$. Since $f(n)$ is asymptotically larger than $n^{\log_b a}$, case 3 of the master theorem asks us to check whether $af\left(\frac{n}{b}\right) \leq cf(n)$ for some $c < 1$ and all n sufficiently large. This is indeed the case, so $T(n) = \Theta(f(n)) = \Theta(n^2)$.

EXAMPLE

Consider the recurrence

$$T(n) = 8T\left(\frac{n}{2}\right) + \frac{n^3}{\log n}.$$

In this case, $n^{\log_b a} = n^3$ and $f(n) = \frac{n^3}{\log n}$. $f(n)$ is smaller than $n^{\log_b a}$ but by less than a polynomial factor. Therefore, the master theorem makes no claim about the solution to the recurrence.

EXAMPLE

Consider the recurrence

$$T(n) = 6T\left(\frac{n}{3}\right) + n.$$

In this case, $n^{\log_b a} = n^2$ and $f(n) = n$. Since $f(n)$ is polynomially smaller than $n^{\log_b a}$, case 1 of the master theorem implies that $T(n) = \Theta(n^2)$.

EXAMPLE

Consider the recurrence

$$T(n) = 27T\left(\frac{n}{3}\right) + n^3.$$

In this case, $n^{\log_b a} = n^3$ and $f(n) = n^3$. Since $f(n)$ is asymptotically the same as $n^{\log_b a}$, case 2 of the master theorem implies that $T(n) = \Theta(n^3 \log n)$.

Example

$$T(n) = 2T\left(\frac{n}{2}\right) + 10n$$

As we can see in the formula above the variables get the following values:

$$a = 2, b = 2, c = 1, f(n) = 10n$$

$$f(n) = \Theta(n^c \log^k n) \text{ where } c = 1, k = 0$$

Next, we see if we satisfy the case 2 condition:

$$\log_b a = \log_2 2 = 1, \text{ and therefore, yes, } c = \log_b a$$

So it follows from the second case of the master theorem:

$$T(n) = \Theta(n^{\log_b a} \log^{k+1} n) = \Theta(n^1 \log^1 n) = \Theta(n \log n)$$

Let $T(n) = T\left(\frac{n}{2}\right) + \frac{1}{2}n^2 + n$. What are the parameters?

$$a = 1$$

$$b = 2$$

$$d = 2$$

Therefore which condition?

Since $1 < 2^2$, case 1 applies.

Thus we conclude that

$$T(n) \in \Theta(n^d) = \Theta(n^2)$$

Let $T(n) = 3T\left(\frac{n}{2}\right) + \frac{3}{4}n + 1$. What are the parameters?

$$\begin{array}{rcl} a & = & 3 \\ b & = & 2 \\ d & = & 1 \end{array}$$

Therefore which condition?

Since $3 > 2^1$, case 3 applies. Thus we conclude that

$$T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3})$$

Say that we have the following recurrence relation:

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

Clearly, $a = 2, b = 2$ but $f(n)$ is not a polynomial. However,

$$f(n) \in \Theta(n \log n)$$

for $k = 1$, therefore, by the 4-th case of the Master Theorem we can say that

$$T(n) \in \Theta(n \log^2 n)$$

Let $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n} + 42$. What are the parameters?

$$\begin{array}{rcl} a & = & 2 \\ b & = & 4 \\ d & = & \frac{1}{2} \end{array}$$

Therefore which condition?

Since $2 = 4^{\frac{1}{2}}$, case 2 applies.

Thus we conclude that

$$T(n) \in \Theta(n^d \log n) = \Theta(\sqrt{n} \log n)$$

Example

$$T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$$

As one can see from the formula above:

$$a = 8, b = 2, f(n) = 1000n^2, \text{ so } f(n) = \Theta(n^c), \text{ where } c = 2$$

Next, we see if we satisfy the case 1 condition:

$$\log_b a = \log_2 8 = 3 > c.$$

It follows from the first case of the master theorem that

$$T(n) = \Theta\left(n^{\log_b a}\right) = \Theta(n^3)$$

Example

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

As we can see in the formula above the variables get the following values:

$$a = 2, b = 2, f(n) = n^2$$

$$f(n) = \Theta(n^c), \text{ where } c = 2$$

Next, we see if we satisfy the case 3 condition:

$$\log_b a = \log_2 2 = 1, \text{ and therefore, yes, } c > \log_b a$$

So it follows from the third case of the master theorem:

$$T(n) = \Theta(f(n)) = \Theta(n^2).$$

Inadmissible equations

The following equations cannot be solved using the master theorem:^[2]

- $T(n) = 2^n T\left(\frac{n}{2}\right) + n^n$
 a is not a constant; the number of subproblems should be fixed
- $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$
non-polynomial difference between $f(n)$ and $n^{\log_b a}$ (see below)
- $T(n) = 0.5T\left(\frac{n}{2}\right) + n$
 $a < 1$ cannot have less than one sub problem
- $T(n) = 64T\left(\frac{n}{8}\right) - n^2 \log n$
 $f(n)$ which is the combination time is not positive
- $T(n) = T\left(\frac{n}{2}\right) + n(2 - \cos n)$
case 3 but regularity violation.

GRAMATIKA:

ДОВРШИТИ ДАТУ ГРАМАТИКУ, односно недостајуће продукције 2 и 3, тако да обе дате реченице буду део језика J, ако је почетни симбол S. Испод сваке реченице ДОВРШИТИ ЗАПОЧЕТ РЕДОСЛЕД ПРИМЕНЕ ПРОДУКЦИЈА који генерише ту реченицу (продукције су обележене бројевима). За тако одређену граматику навести НЕТЕРМИНАЛНЕ и ТЕРМИНАЛНЕ симболе.

1. $S \rightarrow P + E$	3. $P \rightarrow$	5. $E \rightarrow x^* E$
2. $S \rightarrow$	4. $P \rightarrow x + y = E$	6. $E \rightarrow y$

НЕТЕРМИНАЛНИ:

ТЕРМИНАЛНИ:

$$z += x^* y + x^* x^* y \quad \begin{matrix} 1 \rightarrow \\ 2 \rightarrow \end{matrix} \quad o^* x + y = x^* x^* y / x^* y$$

Дата је спецификација граматике језика J. Навести НЕТЕРМИНАЛНЕ и ТЕРМИНАЛНЕ симболе. Испод сваке реченице написати да ли је та реченица део језика J, и ако јесте, редослед примене продукција који генерише ту реченицу (продукције су обележене бројевима). Почетни симбол је S.

1. $S \rightarrow P^* z$	3. $P \rightarrow x = E$	5. $E \rightarrow z^* E$
2. $S \rightarrow o + P$	4. $P \rightarrow o + x = E$	6. $E \rightarrow y$

НЕТЕРМИНАЛНИ:

ТЕРМИНАЛНИ:

$$o + x = z^* z^* z^* y \quad o + x = z^* z^* y^* z$$

Довршити дату граматику, односно две недостајуће продукције за почетни симбол S, тако да обе дате реченице буду део језика J. Испод сваке реченице написати редослед примене продукција који генерише ту реченицу (продукције су обележене бројевима). За тако одређену граматику навести НЕТЕРМИНАЛНЕ и ТЕРМИНАЛНЕ симболе.

1. $S \rightarrow$	3. $P \rightarrow x = E^* z$	5. $E \rightarrow z^* E$
2. $S \rightarrow$	4. $P \rightarrow o + x = E$	6. $E \rightarrow y$

НЕТЕРМИНАЛНИ:

ТЕРМИНАЛНИ:

$$z^* x = z^* y^* z + y \quad x = z^* z^* y^* z^* y$$

MASTER TEOREMA (još par primera sa ranijih ispita):

Применом мастер методе за задату рекуренцију, одредити $T(n)$. Користити одговарајући случај мастер теореме.

$$T(n) = 3T(n/4) + nlgn$$

Применом мастер методе на дати пример, одредити $T(n)$. Користити одговарајући случај мастер теореме.

$$T(n) = T(2n/3) + 1$$

*dodatni: dat je izraz $T(n)=4T(n/5) + nlgn$, primenom master teoreme odrediti asimptotske granice $T(n)$?

POVEZIVANJE:

10. У којем пролазу повезивача се обавља замена симбола правим вредностима? Уколико је симбол релокатибилан и налази се у модулу M1, а његова дефиниција у модулу M2, који су подаци неопходни да би се израчунала његова вредност након повезивања? Претпоставимо да су модули повезани у редоследу M1, M2.

10. Уколико је симбол релокатибилан и налази се у модулу M3, а његова дефиниција у модулу M2, који је минималан скуп података неопходних да би се израчунала његова вредност након повезивања модула M1, M2 и M3 у истом редоследу?

ASEMBLERSKI KODOVI:

1. U sledećim kodovima zaokružiti adresno osetljive instrukcije:

```
a:    mov bx, 1000
      get
      mov [bx], ax
      add bx, 2
      cmp ax, 0
      jne a
      mov cx, bx
      mov bx, 1000
      mov ax, 0
      add ax, [bx]
      add bx, 2
      cmp bx, cx
      jb b
```

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

```
.data
tmp: .word 100
a:   .word 70
val: .word 4
.text
main:
         la      $t0, tmp
         lw      $t1, 0($t0)
         li      $t0, 5
         li      $t2, -5
lab:
         add   $t1, $t1, $t0
         addi  $t2, $t2, 1
         blez $t2, lab
         nop
         la      $t0, a
         sw      $t1, 0($t0)
exit:   nop
```

2. Za treći kod po redu sa gornje slike, popuniti tabelu simbola.

Širina memorije za dati primer je 8 bita, veličina podataka 24 bita (.word = 24 bita), a veličina svake instrukcije 16 bita.

Rešenje:

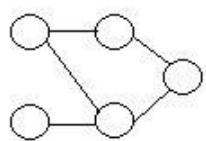
simbol	vrednost	sekcija

simbol	vrednost	sekcija
tmp	0	data
a	3	data
val	6	data
main	9	text
lab	17	text
exit	29	text

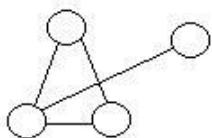
3. Podeliti drugi i treći kod iz 1. zadatka na osnovne programske blokove horizontalnim linijama.

GRAFOVI SMETNJI:

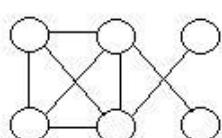
1. Kod kojih od sledećih grafova se mora uraditi spill, ako je broj boja (registara) jednak broju 3?
2. Kod onih grafova gde se ne mora uraditi spill, obojiti graf sa 3 boje.



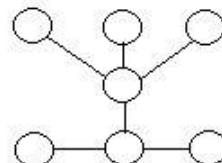
A



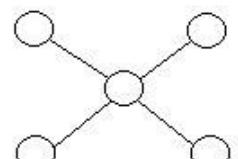
B



C



D



E

ŽIVOTNI VEK PROMENLJIVIH:

1. Data su dva izraza $x=a+b$ i $y=a-b$, da li oni mogu biti paralelni? Odgovor obrazložiti.

2. Dato je parče koda. Odrediti gde počinju i gde se završavaju životni vekovi promenljivih x , y , z , a i b .

Nacrtati graf smetnji za promenljive x , y , z , a i b .

Nacrtati graf toka za dato parče koda.

Odrediti matricu smetnji za promenljive x , y , z , a i b .

```
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. }
```

3. Nacrtati grafove toka i odrediti životni vek promenljivih za sledeće kodove:

1)

```

int evensum(int i)
{
    int sum = 0;

    while (i <= 10) {
        if (i/2 == 0)
            sum = sum + i;
        i++;
    }
    return sum;
}

```

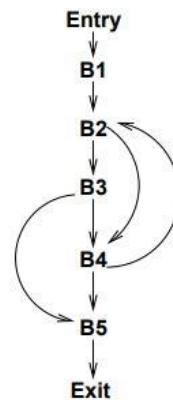
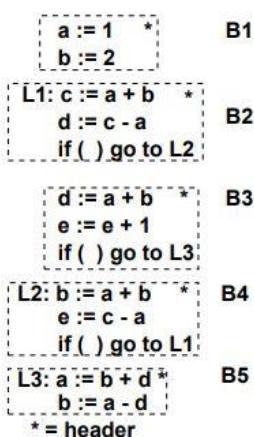
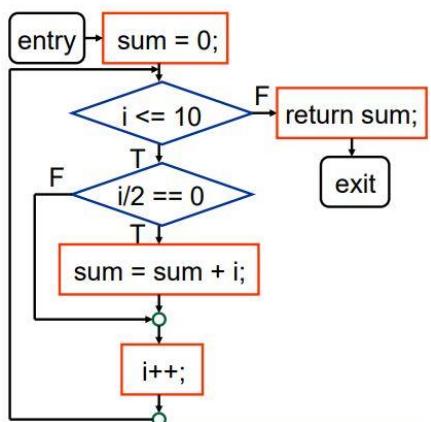
2)

```

a := 1
b := 2
L1: c := a + b
d := c - a
if ( ) go to L2
d := a + b
e := e + 1
if ( ) go to L3
L2: b := a + b
e := c - a
if ( ) go to L1
L3: a := b + d
b := a - d

```

Rešenja:



RAD, RASPONI I PARALELIZAM:

3. Odrediti РАД, РАСПОН и ПАРАЛЕЛИЗАМ за процедуру Foo(A, B):

```

Foo(A, B)
n = A.rows
neka je C nova matrica
dimenzija n x n
parallel_for i = 1 to n
    parallel_for j = 1 to n
        Cij = 0
        for k = 1 to n
            Cij = Cij + AikBkj
return C

```

3. Одредити РАД, РАСПОН и ПАРАЛЕЛИЗАМ за процедуру Foo(A, x):

```
FOO(A, x)
n = A.rows
neka je y novi vector dužine n
parallel_for i = 1 to n
    yi = 0
parallel_for i = 1 to n
    for j = 1 to n
        yi = yi + aijxj
return y
```

3. Одредити РАД, РАСПОН и ПАРАЛЕЛИЗАМ за процедуру Foo(A, B):

```
FOO(A, B)
n = A.rows
neka je C nova matrica nxn
parallel_for i = 1 to n
    for j = 1 to n
        Cij = 0
parallel_for i = 1 to n
    parallel_for j = 1 to n
        for k = 1 to n
            Cij = Cij + AikBkj
return C
```

DODATNI ZADACI:

6. За дату МАКРО ДЕФИНИЦИЈУ, одредити вредност на локацији Z након извршења сваке од макроинструкција из дате табеле.

MS	MAKRO	N, R
	IF	N=0, 8
I	=	1
	MOV	AX, 2
	IF	I=N, 9
	MUL	AX, 2
I	=	I+1
	IF	=, 4
	MOV	AX, 1
	MOV	R, AX
	MKRAJ	

макроинструкција	Z
MF 1, Z	
MF 5, Z	
MF 0, Z	

6. Дефинисати излазе ПРВОГ пролаза асемблера за дати MIPS асемблерски код, под претпоставком да секција података почиње на адреси 0x00000500, а програмска секција на адреси 0x00001000, да су сви подаци и инструкције 32-битне, а да резервисање меморијског блока подразумева параметар о броју заузетих бајтова:

```
.globl main
.data
arr1: .word 5 53
val1: .word 16
arr2: .space 8
val2:
.text
main:
    lw $t3, val2
    la $t1, arr1
    la $t2, arr2
    lw $t3, val1
end:
    jr $ra
```

6. За дату МАКРО ДЕФИНИЦИЈУ, одредити вредност на локацији Z након извршења сваке од макроинструкција из дате табеле.

MF	MAKRO	N, R
	IF	N=0, 8
I	=	1
	MOV	AX, I
	IF	I=N, 9
	MUL	AX, I
I	=	I+1
	IF	=, 4
	MOV	AX, =1
	MOV	R, AX
	MKRAJ	

макроинструкција	Z
MF 4, Z	
MF 5, Z	
MF 0, Z	